McIDAS Developer/Operator Training Manual

Issued October 1995

Space Science and Engineering Center
University of Wisconsin-Madison
1225 West Dayton Street
Madison, WI 53706
Telephone (608) 262-2455
TWX (608) 263-6738

Table of Contents

Welcome to McIDAS Developer/Operating Training iii

Session 1. The McIDAS Programming Environment. 1-1

Session 2. Applications Development in the ADDE 2-1

Session 3. Development Environment for McIDAS-X and -OS2 . . . 3-1

Session 4. Writing GUIs for McIDAS using Tcl/Tk 4-1

Session 5. McIDAS Navigation and Calibration Subsystems 5-1

Session 6. Designing and Implementing Calibration Modules. 6-1

Session 7. Designing and Implementing Navigation Modules. 7-1

Session 8. Developing Local Decoders in McIDAS-XCD. 8-1

Session 9. Moving to a Distributed System 9-1

Session 10. McIDAS-XSD Operations. 10-1

Session 11. McIDAS-XCD Operations 11-1

Session 12. McIDAS Operations on a Distributed System. 12-1

Welcome to McIDAS Operator/Developer Training

Welcome to the 1995 McIDAS Developer/Operator Training sessions. SSEC is providing this training to give you the information you need to develop and support locally-created software in your McIDAS environment.

In the next 2½ days, the McIDAS training staff will do the following:

- teach you how to write applications using the new Applications Program Interfaces (APIs) for argument and data fetching
- explain the new McIDAS paradigm of the client/server relationship
- demonstrate how to write client applications and data servers
- give you instructions for setting up a McIDAS development arena
- tell you how to interface McIDAS applications with a Graphical User Interface (GUI)
- show you on how to write calibration and navigation modules for local data sources and geographic projections
- introduce you to the processes and subsystems included in the McIDAS-XCD package
- explain some of the processes and procedures associated with the administration of a distributed operational environment
- give you hands-on experience

About this training manual

This training manual provides detailed information about the content of each talk. You shouldn't need to take notes.

To follow along in the manual as the trainers present their talks, look for the small number printed in the lower-left corner of their slides. This number references a page or group of pages in the manual.

Throughout this manual, references are occasionally made to lines of source code included with a training session. These line number references are shown as bold characters surrounded by brackets; for example: **[A101-A110]**.

Hands-on exercises

The hands-on exercises are designed to give you practical experience with the information discussed during the talks. You will work in pairs on these exercises. Each xterminal in the training room has an account with the password **mug95trn!**. Each account is set up with a typical McIDAS configuration, as shown below.

Environment variables:

PATH=~mcidas/bin:~/mcidas/bin, plus necessary compiler directories
MCPATH=$HOME/mcidas/data:~trainer/mcidas/data:
~/mcidas/data:~mcidas/data:~mcidas/help

Directories:

$HOME/mcidas
$HOME/mcidas/bin
$HOME/mcidas/data
$HOME/mcidas/help
$HOME/mcidas/lib
$HOME/mcidas/src

You will write sections of code necessary for:

- an ADDE data server
- an ADDE client application
- a graphical user interface for the ADDE client application

The ADDE data server

Your first task will be to write a section of code for an ADDE data server to deliver MRF gridded fields to a client application as image data. The data set provided contains gridded temperature data at 1000 mb from the Medium Range Forecast Model (MRF) in ASCII format. The data is in the directory ~trainer/mcidas/data in the files MRF1000T*nn* where *nn* is the forecast hour of the grids. The first line contains geographic information about the data. The second line contains the valid forecast day and time of the data. The third line contains the filing format used for the data and the units. The remainder of the file contains the actual data oriented in a row-major format, so the first data point occurs at the North Pole and the dateline, and moves eastward around the globe. Subsequent lines move southward, ending at the South Pole. Your task will be to write a portion of the data server that delivers a line of data to the client.

The source code for the server exercise is in $(HOME)/mcidas/src in the files below.

File name	Description
subserv.c	main to all servers
mugadir.c	directory server
mugaget.c	data server
mcmugutil.c	specific functions for training
mcservutil.c	generic functions for training

Due to time constraints, you will only write the function **ReadMugLine**. This function reads a line of data from the file. The interface for this function is shown below.

```
int
ReadMugLine (char *src_file, READPARM *read, int band, short *buf, char *err)

input:
          src_file        ascii file containing source data
          read            READPARM struct containing read specifications
          band            band number to read
output:
          buf             buffer containing image data
          err             error string returned when a failure occurs
```

The READPARM structure contains specifications that may be needed in the function.

```
typedef struct READPARM_
{
          char        src_type[4];/* source data type: GVAR, MSAT, etc..
*/
          char        des_unit[4];/* destination units: RAW, BRIT, etc...
*/
          char        src_unit[4];/* source units: RAW, BRIT, etc...  */

          int         begele;/* beginning element                     */
          int         beglin;/* beginning line                        */
          int         bufsiz;/* size of the buffer to read            */
          int         des_len;/* destination byte length of one data point */
          int         elem_res/* resolution in element direction      */
          int         line_res/* resolution in line direction         */
          int         maxele;/* last element in image                 */
          int         maxlin;/* last line in image                    */
          int         minele;/* first element in image                */
          int         minlin;/* first line in image                   */
          int         numband;/* number of bands in the image         */
          int         numele;/* number of elements to read            */
          int         numlin;/* number of lines to read               */
          int         src_len;/* source byte length of one data point */
          int         ul_elem;/* element in upper left corner of image*/
          int         ul_line;/* line in upper left corner of image   */
} READPARM;
```

Your second task will be to write portions of a client application, MUGAREA, that takes data from two forecast time periods, performs simple math operations on the data, and files the resulting values in a McIDAS Area file. You can then display the created destination file with the McIDAS ADDE command IMGDISP.

You will add the necessary code to perform the following tasks in mugarea.pgm:

- read from the command line:
  - the destination dataset name and position location and separate the two values
  - the number of lines and elements to request from the server
- append the values for the number of lines and elements to the sort strings
- insert **mcaget** calls to get the data from the two source files
- get the nav and cal blocks for the source data
- start the server transaction to write the data
- read in the lines of data and write the results to the destination data
- write out comment cards

The user interface for MUGAREA will look like this when complete:

```
MUGAREA source pos1 operation pos2 destination <Keywords>
Parameters:
   source       source dataset name containing MRF grids
   pos1         position of the first grid to use (default=most recent)
   operation    mathematical operations: ADD, SUB, AVG
   pos2         position of 2nd grid to use (default=second most recent)
   destination  destination dataset and position (no default)
Keywords:
   SIZE=nlines neles     size of area to get from server
```

This completes the exercises scheduled for the first day.

On the second day, you will add the following sections to the GUI code:

- the message text to the beginning of the application
- option widgets for preparing the call to the client application MUGAREA
- the call to the client application MUGAREA
- the online help to the GUI

We hope you find these sessions rewarding and educational. If there is anything we can do for you during your participation in these training sessions, please let us know.

The McIDAS Programming Environment

Presented by

Tom Whittaker

McIDAS Development Team Manager

Session 1

McIDAS Developer/Operator Training

October 23-25, 1995

Table of Contents

Overview 1-1
History 1-1
Our goals 1-2

The McIDAS library 1-4
Conventions 1-4
Generic data fetchers 1-8

Command parameter/argument fetching 1-10

Sample code 1-12

mccmd functions 1-14
General argument fetching utilities 1-14
Character strings 1-16
Numeric 1-17
Time 1-18
Date 1-19
Latitude/longitude (or other angles) 1-20

mcarg functions 1-21
Internal use functions 1-22
Useful utilities that replace older functions 1-24

Argument fetching status 1-26

Overview

This section discusses the software components of McIDAS-X and McIDAS-OS2. First, it introduces you to the historical perspective by describing the evolution of the software development. Then it describes our goals for developing applications in McIDAS. Finally, it provides the information you need for developing applications, including McIDAS library conventions and command parameter/argument fetching.

History

McIDAS was originally developed on a Raytheon-440 computer using punch cards, paper tape and magnetic tape. The second phase moved the code to Harris minicomputers in a distributed network, with two data servers and several applications boxes. The system was then centralized onto an IBM mainframe, which became the McIDAS-MVS system.

Shortly thereafter, the first *smart* workstation was developed on DOS-based personal computers. These machines front-ended display hardware, like the Tower. There were very few local applications, since the multitasking required had to be simulated in the McIDAS software.

The first large-scale port of applications software came when the OS/2 operating system was embraced for PCs. Our efforts to create an environment similar to the mainframe resulted in a relatively easy port of many applications. Soon, however, we learned that modifications were necessary because of special hardware; for example, VGA displays had only 16 color/gray levels. In addition, we needed to write drivers for each display head (VGA, Tower, WIDE WORD, SDA) and make them appear to the applications as the same kind of raster-oriented devices, with some varying characteristics (frame size, number of colors, etc.). At the same time, we needed communications drivers for the common modes (asynchronous, ProNET, TCP/IP) that satisfied all our applications.

The success of McIDAS-OS2 led us to consider migrating to the Unix environment. Sites were requesting support for the applications on these faster, larger hardware platforms. Our first attempt to accomplish this was to take the OS/2 code and write specialized routines for keyboard/mouse, text display, and image/graphics display, plus the system-level interfaces required for communications and disk I/O. McIDAS-X was born. The implementation was done completely using the X Windows system, except for the ASK command and the Graphical User Interface. Early plans to support the WIDE WORD display head from the micro-channel in RS/6000 machines has been dropped.

Moving to support McIDAS-X on more platforms made us increasingly sensitive to industry standards. We have significantly changed the base code to make it more portable and less platform- and vendor-dependent.

Our goals

Last year, we embarked on a project to *reconnect* McIDAS to Unix. Work on this project began early in 1995 and will be completed by the end of the year. Here are some its goals:

Move the low level code to C

This move will help eliminate *infinities*, such as command line length, in the code for such things as array dimensions. C is also generally easier to interface with essential operating system functions.

Create more code that is common between OS/2 and Unix

Changes in I/O, for example, let us use the same **lbi/lbo** routines in Unix and OS/2 versions of McIDAS, producing more efficient maintenance.

Modify the implementation of User Common

User Common will remain available only as long as needed; in some cases, only for the duration of a single command execution.

Rework the image/graphic window

At present, all McIDAS-X routines link with the X Windows library; those that write images and graphics do so by direct calls to the X Windows system. The rework includes the following changes.

- The applications can write display output to a shared memory frame object.
- The display routine reads from the frame object and displays as needed.
- The colorizing algorithms are more flexible.
- Frame objects can have multiple windows.
- Windows can be zoomed, roamed and resized.
- The number of color levels can go beyond the current limit of 128.

Rework the status and command windows

Our survey showed that having these separate windows caused more problems for users than any other single user interface issue. To correct this and also reduce clutter on the desktop, the command and text windows will be combined, much like McIDAS-OS2.

Allow applications to run from the Unix shell

This change will enable automated processing. For example, programs may be started from the Unix crontab; McIDAS commands will no longer require a McIDAS X Windows environment in order to run. These two modes will be available:

- run a single command, making a User Common and McIDAS environment as needed
- create a McIDAS environment, with or without the image/graphics and text/command windows, and allow several commands to run

The McIDAS library

To develop applications in McIDAS, you must understand its library conventions. This section explains those conventions, and provides information about the generic data and argument fetchers.

Conventions

Functions

Although the interface to old functions did not change, some functions were rewritten and some functionality was replaced. For example, reading and writing data is now done with the ADDE functions; command line parameter fetching is done with new argument fetchers.

New functions have a unique prefix:

Prefix	Description
Mc	C-callable, API level
mc	Fortran-callable, API level
M0	C-callable, non-API
m0	Fortran-callable, non-API

Additional functions for reading and writing data using the ADDE model are being written. Image and grids are currently supported; point source and text are in development. All new functions employ the new argument fetchers.

Argument fetchers

The new argument fetchers, released in the June 1995 upgrade, are functional equivalents to all current functions, except **mclex**, which has no analogy. These argument fetchers do the following:

- remove the 12-character limit on command line arguments
- provide convenient range checking on parameter values
- consolidate functions
- provide more systematic error messages

Data fetching APIs

New functions were created for image and grid point data using new APIs to implement the ADDE (Abstract Data Distribution Environment) client/server approach to reading and writing datasets. We are currently creating new applications to functionally replace existing ones in these areas, and are developing APIs for point source and textual data.

The new APIs treat the data in a more abstract manner. For example, requests to read data use a descriptive list to define the appropriate subset of the dataset that the server will return to the client. The data returned to the client is in a defined, internal format, irrespective of the actual file format of the data. For example, a server written to read grid point data may return the data in the form of an image.

Coding in C

Most new library functions are coded in C, with Fortran interfaces provided as needed. There are two considerations here:

- The ordering of elements in multi-dimensional arrays in Fortran and C are different.
- The strings of characters between Fortran and C are different.

Using C provides the opportunity to eliminate the hard-dimensioned arrays that Fortran requires. Below is an example of cross-language interfacing.

From Fortran, calling a Fortran interface:

```
CALL SDEST('THIS IS A TEST',99)
```

From C, calling a C-coded interface looks much the same:

```
(void) sdest("this is a test",99);
```

From C, calling a Fortran-coded interface looks different. First, the name of the function must end with an underscore character, for example: **sdest_**. Second, the Fortran convention is to pass arguments by address; C passes by value. Finally, Fortran appends an extra hidden argument for each character string passed to a function containing the length of the string.

```
char message[] = "THIS IS A TEST";
int value = 99;

(void) sdest_(message, &value, strlen(message));
```

New utilities

New library utilities are continually being added to the McIDAS library. With the June 1995 upgrade, functions were added for converting strings to time, date, and angle. A new units conversion function was also released. We plan to expand upon this set with each upgrade.

Interface documentation block

All new software contains a standard, formatted Interface Documentation Block that provides complete interface information with the code. The components of this block include the following:

Field	Description
Name	name and short description
Interface	details of the interface/call sequence
Input	input variables
Input/Output	variables used for both input and output
Output	output variables
Return values	values and a description of what they mean
Remarks	useful information, algorithms, etc.
Categories	a list of words from which to choose, for example: grid, image, pt_src, text, system, event, file, etc.

This information is currently in Chapter 5 of the preliminary version of the *McIDAS Programmer's Manual*. In the future, cross-reference searching tools, using the Categories values, will be provided to help identify functions of particular classes.

Below is a sample template that SSEC uses for Fortran API functions and subroutines.

```
C   THIS IS SSEC PROPRIETARY SOFTWARE - ITS USE IS RESTRICTED.

C *** McIDAS Revision History ***
C *** McIDAS Revision History ***

*$ Name:
*$      mcname - short description of purpose/use/etc
*$
*$ Interface:
*$      subroutine
*$      integer function
*$      double precision function
*$      mcname(integer param1, character*(*) param2, integer param3(64))
*$
*$ Input:
*$      none
*$      param1  - description of it
*$
*$ Input and Output:
*$      none
*$      param2  - description of it
*$
```

```
*$ Output:
*$       none
*$       param3  - description of it
*$
*$ Return values:
*$        0      - success
*$
*$ Remarks:
*$       Important use info, algorithm, etc.
*$
*$ Categories:
*$       grid
*$       image
*$       pt_src
*$       text
*$       system
*$       event
*$       file
*$       sys_config
*$       display
*$       graphic
*$       utility
*$       converter
*$       day/time
*$       calibration
*$       navigation
*$       ingest/decode
*$       met/science
*$       user_interface

         SUBROUTINE MCNAME (.....)
         IMPLICIT NONE
C --- symbolic constants & shared data
C --- external functions
C --- local variables
C --- initialized variables
```

Compatibility library

During the transition from old functions to new, SSEC will support the old interfaces. Once a function is no longer called by any McIDAS application, it will move to the compatibility library. No further testing of that function will occur. Each site is responsible for maintaining any code that uses these functions. After a period of one year, SSEC will cease issuing the function with McIDAS software upgrades.

McIDAS is moving toward a client/server paradigm, where the applications do not have direct knowledge of the data's location or format. To accomplish this, both communications and an abstract model of the data formats must be created; the job of the server is then to map the actual file format into this abstraction and deliver it to the client. Client software can then be written to the definition of the abstraction to work with particular data formats, such as: image, grid, point source and text.

The image and grid data formats are defined and included in the suite of applications in core McIDAS. Work continues on point source and text.

Note that some of the terminology has changed.

image	refers to data that may be viewed as a picture, but may also contain information about geo-referencing and calibration; this is to distinguish the McIDAS Area file format from the form in which data arrive at the client, namely *image format*.
grid	hasn't changed meaning; however, the abstraction of the grid point data is somewhat different than the McIDAS grid file format.
point source	refers to single point data; for example, the McIDAS MD file format.
text	refers to plain text information that is undecoded, line-oriented, and intended to be read

In all cases, the client makes a request of the server using an abstract description that includes a dataset name and other selection criteria. In many cases, choices of returned units may also be specified. After the request is made, the client then asks for *records* to be returned. The format of each record follows:

Data type	**Record format**
image	one-dimensional array of values of a single quantity
grid	two-dimensional array of values of a single quantity
point source	array of values at a single point of several quantities
text	lines of text

Image data functions

Functions are available to initiate transactions for reading and writing image data, as well as the prefix, comment card, calibration, navigation, directory and image data.

These are described in Chapter 5 of the preliminary version of the *McIDAS Programmer's Manual (10/95)*.

Grid point data functions

Functions are available for requesting one or more grids from a server, reading and writing grids, etc. These also are described in Chapter 5 of the preliminary version of the *McIDAS Programmer's Manual (10/95)*.

Command parameter/argument fetching

The McIDAS environment provides interfaces for applications programs to pick up values of command-line arguments. Interfacing functions also provide a mechanism for parsing arbitrary strings from a source other than the command line.

All applications-level interfaces are prefixed with **mccmd** (Fortran) or **Mccmd** (C). The *cmd* means that diagnostic messages are displayed by the argument fetching subsystem for any syntax or format errors that occur. If an error does occur, the functions return a status of less than zero.

Note these significant changes:

- The string limit is no longer limited to 12-characters; it can be whatever the calling program declares.
- The syntax for keywords is now *aaa.bbb*, where *aaa* is required, and any of *bbb* may be used.
- The keyword " " (blank) signifies the command line for getting positional parameters; thus the previous functions **ikwp** and **ipp** are combined into one routine, **mccmdint**.

The functions for command line parameter fetching are listed below.

Name	Description
Mccmd	fetches the current McIDAS command line
Mccmdkey	validates the defined and command line keywords
Mccmdnam	fetches all keyword names occurring in the command line
Mccmdnum	returns the number of values associated with a command line keyword
Mccmdquo	fetches the quote field string command line argument
Mccmdstr	fetches a program command line argument in character format
Mccmdint	fetches a program command line argument in integer format
Mccmddbl	fetches a program command line argument in double format
Mccmdiyd	fetches a program argument in integer date format, yyyyddd
Mccmdihr	fetches a program argument in integer time format, hhmmss
Mccmddhr	fetches a program argument in fractional hours, hh.fffff

Name	Description
Mccmdill	fetches a program argument in integer latitude/longitude format, ddmmss
Mccmddl	fetches a program argument in latitude/longitude format, dd.fffff

The functions below are general-purpose utilities.

Name	Description
mcstrtoint	converts the numeric token to integer type format
mcstrtodbl	converts the numeric token to double type format
mcstrtohex	converts the hexadecimal token to integer type format
mcstrtoiyd	converts the date token to integer date format, yyyyddd
mcstrtodhr	converts the time token to double fractional hours, hh.ffff
mcstrtoill	converts the latitude/longitude token to integer type format, dddmmss
mcstrtodll	converts the token to double fractional latitude/ longitude, ddd.fffff
mcucvtr	converts physical units; real values
mcucvtd	converts physical units; double precision values

Example

```
      INTEGER MCCMDINT
      INTEGER FRAME
      INTERGER VALUE
C
C   Replacement for IPP and IKWP
C
C   MCCMDINT(keyword, position, message, def, min, max, value)
C

C   Get first parameter on keyword "FRAME"; use LUC(51) as the
C   default, and limit the range from 1 to LUC(13)
C
      if (MCCMDINT('FRA.ME', 1, 'Frame number', LUC(51),
     &  1, LUC(13), FRAME)) .lt. 0) return

C
C   Get first positional parameter on command; use -1 as the default
C   and do NOT check the range (allow any integer value)
C
      if (MCCMDINT('', 1, 'The first value', -1,
     &  1, 0, VALUE)) .lt. 0) return
```

Sample code

```
C   THIS IS SSEC PROPRIETARY SOFTWARE - ITS USE IS RESTRICTED.

C *** McIDAS Revision History ***
C *** McIDAS Revision History ***

C ? EXAM -- Describe the purpose of this command
C ?    EXAM color <keywords> "quote
C ? Parameters:
C ?    color    | the color level to write the text in (def=1)
C ? Keywords:
C ?    GRAPHIC=    | graphics frame number (def=current)
C ?    WIDTH=      | line width (def=1)

C ? ------

      subroutine main0
      implicit none


C --- symbolic constants & shared data

C --- external functions

      integer mccmdkey
      integer mccmdint
      integer mccmdstr
      integer mccmdquo

C --- local variables

      character*80 quote
      character*10 option
      character *255 quotestring
      integer colorlevel
      integer graphicframe
      integer width

C --- initialized variables

      character*10 keynames(2)
      data keynames/'GRA.PHIC', 'W.IDTH'/

C  Verify the keywords...

      if (mccmdkey(2,keynames).lt.0) goto 999

C  Get the color level
      if (mccmdint(' ',1,'Text color',1,
     & itrmch('MIN_COLOR_LEV',-1), itrmch('MAX_COLOR_LEV',-1),
     & colorlevel)  .lt. 0) goto 999

C  Get the graphic frame number

      if (mccmdint('GRA.PHIC',1,'Graphics frame',luc(56),1,
     & luc(14), graphicframe) .lt. 0) goto 999

C  Get the line width

      if (mccmdint('W.IDTH'),1,'Line width',1,1,10,width)
     & .lt. 0) goto 999

C  Finally, get the quote string

      if (mccmdquo(quotestring).lt.0) goto 999
```

```
C  Start the graphics with the desired frame and width

       call initpl(graphicframe,width)

C  Write the line of text

       call wrtext(luc(63),luc(64),10,quotestring,
      & len_trim(quotestring),colorlevel)

       call endplt

999    return
       end
```

```
EXAM 1 GRA=10 "TEST

EXAM:
EXAM: Invalid Graphics frame.
EXAM: first GRA= argument is too big --> 10
EXAM: Must be valid 'Graphics frame' integer value within range 1 thru 4.
EXAM:

EXAM 10 GRA=1 "TEST
EXAM:
EXAM: Invalid Text color.
EXAM: first positional argument is too big --> 10
EXAM: Must be valid 'Text color' integer value within range 1 thru 8.
EXAM:

EXAM 1 GRX=1 "TEST
EXAM:
EXAM: Invalid command keywords: GRX,
EXAM:
```

mccmd functions

The sections below give detailed information and examples of the most commonly used functions.

General argument fetching utilities

The utilities **mccmdkey**, **mccmdnam**, **mccmdnum** and **mccmd** are further defined below.

mccmdkey

```
mccmdkey - Validate command line keywords, printing errors to edest.
     Replaces function:   keychk

  integer function
     mccmdkey(integer numkey, character*(*) valid_keywords(numkey))

     Example:
          INTEGER NUMKWS
            INTEGER MCCMDKEY
            PARAMETER (NUMKWS=3)
            CHARACTER*10 KEYWORDS(NUMKWS)
            ...
            DATA KEYWORDS/'FRA.ME','COL.OR','LIN.ES'/
  C--- this means that for the keyword FRAME, at
  C--- least 'FRA' must be specified; however the
  C--- values 'FRA', 'FRAM' or 'FRAME' may be used.
            ...
            IF (MCCMDKEY(NUMKWS, KEYWORDS).LT.0) RETURN
            ...
```

mccmdnam

```
mccmdnam - Get all keyword names occurring in the command line.
     Replaces function:   kwnams

  integer function
     mccmdnam(integer maxkey, character*(*) keywords(maxkey))

     Example:
          INTEGER MAXKEY
            INTEGER MCCMDNAM
            INTEGER NUMKEYS
            PARAMETER (MAXKEY=25)
            CHARACTER*255 KEYWORDS(MAXKEY)
            ...
            NUMKEYS = MCCMDNAM(MAXKEY, KEYWORDS)
            IF (NUMKEYS.LT.0) RETURN
            ...
```

mccmdnum

```
mccmdnum - Return # values associated with given command line keyword.
     Replaces function:   nkwp

  integer function
     mccmdnum(character*(*) keyword)

     Example:
          INTEGER MCCMDNUM
           INTEGER NUMKEYS
           ...
           NUMKEYS = MCCDNUM('FRA.ME')
```

mccmd

```
mccmd - Build and return the current McIDAS command line.

  character*(*) function
     mccmd()

     Example:
           CHARACTER*255,MCCMD,COMMAND
           ...
           COMMAND = MCCMD()
```

The character string functions **mccmdquo** and **mccmdstr** are defined below.

mccmdquo

```
mccmdquo - Get the quote field string command line argument.
     Replaces function:   cqfld
        Note: there is NO leading " character on the string
              returned by this function

     integer function
     mccmdquo(character*(*) value)

     Example:
          INTEGER MCCMDQUO
            INTEGER STATUS
            CHARACTER*255 QUOTESTRING
            ...
            STATUS = MCCMDQUO(QUOTESTRING)

  C--- if the status is < 0 then an error happened
            IF (STATUS.LT.0) RETURN

            IF (LEN_TRIM(QUOTESTRING).EQ.0) THEN
  C--- there is no quote string
```

mccmdstr

```
mccmdstr - Get a program command line argument in character form.
     Replaces functions:   ckwp, cpp

     integer function
     mccmdstr(character*(*) keyword, integer position,
            character*(*) default, character*(*) value)

     Example:
          INTEGER MCCMDSTR
            CHARACTER*255 VALUE
            ...
  C--- get the 2nd positional parameter as a string
            IF (MCCMDSTR(' ', 2, 'NONE', VALUE).LT.0) RETURN

  C--- now get the first keyword parameter for TYPE=
            IF (MCCMDSTR('TYP.E', 1, 'NONE', VALUE).LT.0) RETURN
            IF (VALUE.EQ.'NONE') THEN
              -- default returned or user typed "NONE" --
```

The numeric functions **mccmdint** and **mccmddbl** are described below.

mccmdint

```
mccmdint - Get a program command line argument in integer type format.
     Replaces functions:   ikwp, ipp

     integer function
     mccmdint(character*(*) keyword, integer position,
            character*(*) printmsg, integer def, integer min,
            integer max, integer value)

     Example:
            INTEGER MCCMDINT
            INTEGER FRAME
            ...
  C--- if the first parameter of the keyword FRAME has
  C--- a syntax error, let the system print out an
  C--- appropriate message and just return

            IF (MCCMDINT('FRA.ME', 1, 'Frame number', LUC(51),
           &  1, LUC(13), FRAME)) .LT.0) RETURN
```

mccmddbl

```
mccmddbl - Get a program command line argument in double type format.
     Replaces functions:   dkwp, dpp

     integer function
     mccmddbl(character*(*) keyword, integer position,
            character*(*) printmsg, double precision def,
            double precision min, double precision max,
            double precision value)

     Example:
            INTEGER MCCMDDBL
            DOUBLE PRECISION TEMP
            ...
  C--- Note that the range and default values must be
  C--- DOUBLE PRECISION values

            IF (MCCMDDBL(' ', 1, 'Temperature', 27316.D-2,
           &  -300.D0, 400.D0, TEMP).LT.0) RETURN
```

The time functions **mccmdihr** and **mccmddhr** are described below.

mccmdihr

```
mccmdihr - Get a program argument in integer type time format hhmmss.
     Replaces functions:   ikwphr, ipphr, mkwp, mpp

     integer function
     mccmdihr(character*(*) keyword, integer position,
       character*(*) printmsg, integer def, integer min,
            integer max, integer value)

       Example:
            INTEGER MCCMDIHR
            INTEGER BEGTIME
            INTEGER ENDTIME
            ...
  C--- Note that for the INTEGER functions, the form
  C--- of the returned value is HHMMSS; also, the
  C--- range and default values must be specified the same way

            IF (MCCMDIHR('TIM.E', 1, 'Starting time', 0, 0,
           &  235959, BEGTIME).LT.0) RETURN

            IF (MCCMDIHR('TIM.E', 2,'Ending time', BEGTIME, 0,
           &  235959, ENDTIME).LT.0) RETURN
```

mccmddhr

```
mccmddhr - Get a program argument in fractional hours format hh.fffff.
     Replaces functions:   dkwphr, dpphr

     integer function
     mccmddhr(character*(*) keyword, integer position,
       character*(*) printmsg, double precision def,
       double precision min, double precision max,
       double precision value)

       Example:
            INTEGER MCCMDDHR
            DOUBLE PRECISION BEGTIME
            DOUBLE PRECISION ENDTIME
            ...
  C--- Note that the returned value is DOUBLE PRECISION
  C--- in the form: hh.nnnnn; the range and default values
  C--- must be specified in the same way

            IF (MCCMDDHR('TIM.E', 1, 'Starting time', 0.D0,
           &  0.D0, 24.D0, BEGTIME).LT.0) RETURN

            IF (MCCMDDHR('TIM.E', 2, 'Ending time', BEGTIME,
           &  0.D0, 24D0, ENDTIME).LT.0) RETURN
```

The date routine **mccmdiyd** is described below.

mccmdiyd

```
mccmdiyd - Get a program argument in integer type date format yyyyddd.
     Replaces functions:   ikwpyd, ippyd

       integer function
       mccmdiyd(character*(*) keyword, integer position,
         character*(*) printmsg, integer def, integer min,
            integer max, integer value)

         Example:
            INTEGER MCCMDIYD
            ...
      C--- Note that the value returned is in the form: YYYYDDD
      C--- and that the range and default values must be
      C--- specified in the same manner

            IF (MCCMDIYD('DAT.E', 1, 'Date', 0, 1970001,
         &  2035001).LT.0)  RETURN
```

The lat/lon functions **mccmdill** and **mccmddll** are described below.

mccmdill

```
mccmdill - Get a program argument in integer type lat/lon form ddmmss.
     Replaces functions:   ikwpll, ippll

       integer function
       mccmdill(character*(*) keyword, integer position,
            character*(*) printmsg, integer def, integer min,
            integer max, integer value

       Example:
            INTEGER MCCMDILL
            INTEGER LATNW
            INTEGER LATSE
            ...
     C--- Note that the value returned is in the form: DDDMMSS
     C--- and that the range and default values must be
     C--- specified in the same way

            IF (MCCMDILL('LAT.ITUDE', 1, 'Upper left latitude',
        &   0, -900000, 900000, LATNW).LT.0) RETURN

            IF (MCCMDILL('LAT.ITUDE', 2, 'Lower right latitude',
        &   LATNW, -900000,900000, LATSE).LT.0) RETURN
```

mccmddll

```
mccmddll - Get a program argument in fractional lat/lon form dd.fffff.
     Replaces functions:   dkwpll, dppll

       integer function
       mccmddll(character*(*) keyword, integer position,
            character*(*) printmsg, double precision def,
            double precision min, double precision max,
            double precision value)

       Example:
            INTEGER MCCMDDLL
            DOUBLE PRECISION LATNW
            DOUBLE PRECITION LATSE
            ...
     C--- Note that the value returned is a DOUBLE PRECISION value
     C--- and is in the form: ddd.nnnn; the range and default
     C--- values must be specified in the same way

            IF (MCCMDDLL('LAT.ITUDE', 1, 'Upper left latitude',
        &   0.D0, -90.D0, 90.D0, LATNW).LT.0) RETURN

            IF (MCCMDDLL('LAT.ITUDE', 2, 'Lower right latitude',
        &   LATNW, -90.D0,90.D0, LATSE).LT.0) RETURN
```

mcarg functions

The **mccmd** functions call another series of functions whose names begin with **mcarg**. While the mccmd functions provide for error message output and operate only on the command line, while the **mcarg** functions use a handle to identify the string being parsed and translated, and you can configure the parsing of tokens.

To use these functions, first call **mcargparse** to place the string of characters in the internal structure. The return from **mcargparse** is a unique handle used in subsequent calls to fetch values. At the end, call **mcargfree** to free the handle and the internal space.

For example:

```
ihand = mcargparse('PGM first second 1 2 3',0,length)
istat = mcargstr(ihand, ' ',1,' ',string)
istat = mcargint(ihand,' ',3,0,1,0,ival,global)
istat = mcargfree(ihand)
```

Note that support layer functions (those below the API) are used by the argument fetchers; you should avoid using them in applications programs since their interfaces or function may change. One possible exception during this transition phase is if your applications call **mclex**, which has no direct, API-level replacement. If you need an arbitrary string to be treated as the command line, you must make these calls:

```
istat = mcargfree(0)
istat = m0cmdput( m0cmdparse(cline, lenlin) )
```

The **mcarg** functions are listed below.

General

```
mcargparse - Parse the given text into arg-fetching structure.
      (return the handle)

      integer function
      mcargparse(character*(*) txtstr, character*(*) given_syntax(10),
           integer parsed_len)


mcargfree - Free parsed arg-fetching structure for the given handle.

      integer function
      mcargfree(integer arg_handle)


mcargdump - Display parsed arg-fetching to McIDAS debug destination.

      subroutine
      mcargdump(integer arg_handle)


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

      character*(*) function
      mcargcmd(integer arg_handle)


mcargkey - Validate arg-fetching keywords, optionally printing errors.

      integer function
      mcargkey(integer arg_handle, integer numkey,
           character*(*) valid_keywords(numkey), integer printflag)


mcargnam - Fetch all keyword names within parsed arg-fetching text.

      integer function
      mcargnam(integer arg_handle, integer maxkey,
           character*(*) keywords(maxkey))


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

      integer function
      mcargnum(integer arg_handle, character*(*) keyword)
```

Character strings

```
mcargquo - Fetch the quote field string argument.

      integer function
      mcargquo(integer arg_handle, character*(*) value)


mcargstr - Fetch an argument in character form.

      integer function
      mcargstr(integer arg_handle, character*(*) keyword,
           integer position, character*(*) def, character*(*) value)
```

Numeric

mcargint - Fetch an argument in integer type format.

```
integer function
mcargint(integer arg_handle, character*(*) keyword,
     integer position, integer def, integer min,
         integer max, integer value, character*(*) arg)
```

mcargdbl - Fetch an argument in double type format.

```
integer function
mcargdbl(integer arg_handle, characer*(*) keyword,
     integer position,
     double precision def, double precision min,
     double precision max, double precision value,
     character*(*) arg)
```

Time

mcargihr - Fetch an argument in integer type time format hhmmss.

```
integer function
mcargihr(integer arg_handle, character*(*) keyword,
     integer position, integer def, integer min,
         integer max, integer value, character*(*) arg)
```

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

```
integer function
mcargdhr(integer arg_handle, character*(*) keyword,
     integer position,
     double precision def, double precision min,
     double precision max, double precision value,
     character*(*) arg)
```

Date

mcargiyd - Fetch an argument in integer type date format yyyyddd.

```
integer function
mcargiyd(integer arg_handle, character*(*) keyword,
     integer position, integer def, integer min,
         integer max, integer value, character*(*) arg)
```

Latitude/longitude (or other angles)

```
mcargill - Fetch an argument in integer type lat/lon format dddmmss.

       integer function
       mcargill(integer arg_handle, character*(*) keyword,
            integer position, integer def, integer min,
                integer max, integer value, character*(*) arg)


mcargdll - Fetch argument in double fractional lat/lon format ddd.fffff.

       integer function
       mcargdll(integer arg_handle, character*(*) keyword,
            integer position,
            double precision def, double precision min,
            double precision max, double precision value,
            character*(*) arg)
```

Useful utilities that replace older functions

Numeric

```
mcstrtoint - Convert given numeric token to integer type format.
             Replaces function: iftok

       integer function
       mcstrtoint(character*(*) token, integer value)


mcstrtodbl - Convert given numeric token to double type format.
             Replaces function: dftok

       integer function
       mcstrtodbl(character*(*) token, double precision value)


mcstrtohex - Convert given hexadecimal token to integer type format.
             Replaces function: iftok

       integer function
       mcstrtohex(character*(*) token, integer value)
```

Time

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

       integer function
       mcstrtohms(character*(*) token, integer hour,
         integer min, integer sec)

mcstrtoihr - Convert given time token to integer time format hhmmss.
             Replaces function: itokhr, iftok

       integer function
       mcstrtoihr(character*(*) token, integer ihr)


mcstrtodhr - Convert given time token to double fractional hours hh.ffff
             Replaces function: dtokhr, dftok

       integer function
       mcstrtodhr(character*(*) token, double precision dhr)
```

Date

```
mcstrtoiyd - Convert given date token to integer date format yyyyddd.
             Replaces function: itokyd, iftok

       integer function
       mcstrtoiyd(character*(*) token, integer iyd)
```

Latitude/longitude (or other angles)

```
mcstrtoill - Convert given lat/lon token to integer type format dddmmss.
             Replaces function: itokll, ifto

       integer function
       mcstrtoill(character*(*) token, integer ill)


mcstrtodll - Convert given token to double fractional lat/lon ddd.fffff.
             Replaces function: dtokll, dftok

       integer function
       mcstrtodll(character*(*) token, double precision dll)


mcucvtr - Convert an array of real values from one unit to another.

       integer function
       mcucvtr(integer num, character*(*) unitin, real bufin(*),
            character*(*) unitout, real bufout(*), integer idif)

mcucvtd - Convert an array of double precision values from one unit
             to another.

       integer function
       mcucvtd(integer num, character*(*) unitin, double precision bufin(*),
            character*(*) unitout, double precision bufout(*), integer idif
```

Argument fetching status

The argument fetching status codes and their descriptions are listed below.

Status code	Definition
[-] 0nnn	argument comes from the default
[-] 1nnn	argument comes from the command line
[-] 2nnn	argument comes from the system string table
[-] n00n	character string argument
[-] n01n	quote field string argument
[-] n10n	integer argument
[-] n11n	integer hexadecimal argument
[-] n20n	decimal argument (double)
[-] n21n	double hexadecimal argument
[-] n30n	date argument
[-] n31n	current date argument
- n32n	year within date argument is invalid
- n33n	*mon* month within date argument is invalid
- n34n	*mm* month within date argument is invalid
- n35n	day of month (dd) within date argument is invalid
- n36n	day of year (ddd) within date argument is invalid
[-] n40n	integer time argument
[-] n41n	current integer time argument
- n42n	hours within integer time argument are invalid
- n43n	minutes within integer time argument are invalid
- n44n	seconds within integer time argument are invalid
[-] n45n	double time argument
[-] n46n	current double time argument
- n47n	hours within double time argument are invalid
- n48n	minutes within double time argument are invalid
- n49n	seconds within double time argument are invalid
[-] n50n	integer lat/lon argument
- n52n	degrees within integer lat/lon argument are invalid
- n53n	minutes within integer lat/lon argument are invalid
- n54n	seconds within integer lat/lon argument are invalid
[-] n55n	double lat/lon argument
- n57n	degrees within double lat/lon argument are invalid
- n58n	minutes within double lat/lon argument are invalid
- n59n	seconds within double lat/lon argument are invalid
[-] 90n	keyword status
nnn0	argument is ok
- nnn1	argument is invalidly formatted (invalid char)
- nnn2	integer argument can't contain a fraction
- nnn3	argument exceeds system limits for desired format
- nnn4	out-of-range argument < given min
- nnn5	out-of-range argument > given max

Applications Development in the ADDE

Presented by

Dave Santek

McIDAS Applications Project Leader

Session 2

McIDAS Developer/Operator Training

October 23-25, 1995

Table of Contents

Overview 2-1
- General terminology 2-2
- Client/server concepts 2-3
- ADDE name constructs 2-4

Image data 2-5
- Terminology 2-5
- Image data API 2-7
- Reading the image directory block 2-8
- Reading the data block 2-11
- Reading the navigation block 2-15
- Reading the calibration block 2-16
- Reading the comment block 2-17
- Writing image data 2-18

Gridded data 2-22
- Gridded data API 2-23
- mgsort 2-24
- Reading grids 2-25

ADDE servers 2-27
- Sending data 2-27
- Performance 2-27
- Error reporting 2-28
- Debugging 2-30
- 'Under the hood' 2-30

Exercise 2-32

Sample code 2-35
- mugaget.c 2-35
- mugadir.c 2-46
- mcmugutil.c 2-50
- mug.h 2-72
- subserv.c 2-75
- servutil.h 2-77
- mugarea.pgm 2-79

Overview

The ADDE (Abstract Data Distribution Environment) provides the following enhancements to McIDAS.

Easier access to data on remote machines

As satellite and weather data are acquired on a distribution of workstations rather than just the mainframe, our current mechanism for identifying data sets (using numbers for Area, Grids, and MDs) becomes very limiting. A use of NFS (Network File System) with the McIDAS File Redirection would be very difficult to manage. ADDE uses names to denote datasets; file numbers are irrelevant from the users' perspective, but are still used on the workstation serving the McIDAS data. For example, RT may point to a machine that has Real-time data on it.

Improved performance

The servers on MVS are always running and the data is not spooled up before sending it. The user's application will make use of the data as soon as it starts being sent. For Unix data servers, the performance is much better than NFS.

Better data management capabilities

The use of names, instead of numbers, is a more intuitive way to manage data. Rather than remembering that the GOES-7 full resolution visible data is stored in Area files 101 to 104, in ADDE they may be stored in WEST/VIS-CONUS. And if the operational west satellite is switched to GOES-9, the ADDE name remains the same, though the file numbers may be different. The user doesn't need to know, only the administrator of the server.

More flexibility in data handling

We've identified three basic data types for use in ADDE: Image, Grid and Point. These data types have one thing in common: the ability to earth locate an individual data element. Also, Image and Grid data are both 2-D arrays of data elements. These similarities make it possible to serve data to applications in a format other than its native format. In this training session, the data source is 2-D grids but the server will serve it to the application as Image data.

Transparent incorporation of non-McIDAS data formats

You can write servers that allow McIDAS ADDE applications to read and write in non-McIDAS data formats. We have prototyped this capability with SSM/I Pathfinder HDF files and GI (Global Imaging) satellite image data format. In today's exercise, we will work with grids stored in text files.

This training session will provide the infomation you need to write your own image data and gridded data applications and servers in the ADDE. If you have not used ADDE, you should read the section titled Introduction to the ADDE, in either the *McIDAS-X* or *McIDAS-OS2 Users Guide*.

General terminology

The terms below are used throughout this section.

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.
inetd	Unix system daemon that listens to various network ports.
position	absolute position number; corresponds to an Area or Grid file in a range of files.
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.

The following is an analogy to ADDE: The server is the cook in a restaurant. The patron (ADDE application) examines the menu (named datasets), places an order (request) with the waiter (client) who walks through the door (pipe) into the kitchen (server machine) and gives the order to the cook (server application). The cook prepares the food (data) and makes it available for pickup (in the pipe). The waiter brings it to your table (workstation). You, as the application, get only one chance to order and cannot make any changes. Also, you promise to eat everything on your plate (read all the data); no more, no less.

ADDE distributes data using networked servers and clients. Servers store the data and send it to a client. Clients request and receive data, and run applications on the data such as displaying imagery or contouring. When you run McIDAS commands that manipulate data, such as IMGDISP, these are client applications. When you run the DSSERVE command to manage local datasets, that is an example of running a server application.

Each McIDAS session acts as both a client and a local server. The client can request data from either its local server or from a remote server. The remote server can either be a McIDAS-MVS mainframe or McIDAS-X workstation configured as a server of data.

Clients and servers communicate via a TCP/IP communications protocol.

Views of the world

We've identified two distinct domains: the client and server. Because of the intentional separation, their view of the other's domain is abstract.

The applications that run on the client are instructed by the user to operate on a dataset along with some selection criteria (sort clauses). The application makes a request to have the data returned in a particular format; Image, for example. By virtue of this abstraction, the application is assured of having the data available as requested or is notified that this is not possible.

On the other hand, the server promises to send back the data as requested or notify the application if there is a problem. Each server operates on a specific data format; there are individual servers for each of the McIDAS data formats. Additionally, the way the server sends the data back will necessitate more servers. For instance, we have a text file containing grid point data. We may want to serve it up as an image, to display it as a grayscale picture (IMGDISP), or as a grid to contour it (GRDDISP).

In non-ADDE commands, all image, grid and point source files are referenced by file numbers. If you don't know the file numbers, finding data can be difficult. The ADDE commands use dataset names composed of three parts: *type*, *group* and *descriptor*. *Type* indicates the type of data: image, grid or point. McIDAS Area files are of type image; McIDAS Grid files are of type grid; McIDAS MD files are of type point.

To incorporate a non-McIDAS file format, the *type* of data must be identified; it is related to the organization of the data and how it will be used. For example, 2-D arrays of data can be thought of as image or grid data in our model. To classify the data as image or grid, you should consider the size, source of data, and display format (grayscale picture vs. contour). With ADDE, it can be both.

The *group* and *descriptor* point to a particular dataset. On the client, a routing table determines which server to route the request to based on the *group*. On the server, the *group* and *descriptor* resolve to a dataset, such as a range of McIDAS Area files.

Image data

Although accessing image data in ADDE is different than the traditional methods, there is enough similarity to make the porting of applications a surmountable task. This phase in the implementation of ADDE provides a transition to the new data access procedures without extensive changes to existing code.

You must adhere to some new rules in ADDE. The previous method allowed for random access and more freedom in taking alternative actions based on the data being read. In ADDE, the data access is sequential, and once requested, must all be read in. There are now some global keywords related to image file data. Most of these specifications are related to sectorizing the data (time and center point, for example) so that each application does not have to retrieve and validate the keywords independently. Also, the limitation of three open McIDAS Area files was lifted; the limiting factor now is resources on the client (memory) and the server (number of server processes running concurrently).

A general structural difference in the logic is also apparent. Previously, any kind of searching or sectorizing was done in the application and it was scattered throughout. Now, search conditions and sector specifications are defined at request time and are processed by the server. By the time the application is reading the data, it should be what is precisely needed.

Terminology

The terms below are used throughout this section on image data.

calibration block	the block that holds the information to transform the image element's sensor units to 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.
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
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.

The ADDE image data API routines are listed below by function, along with equivalent routines for the old API.

A cursory look at this list gives the appearance of a one-to-one correspondence. There are similarities which make the transition of existing code to ADDE plausible; but logic restructuring is also part of this process, as the new routines will not just drop in. The next few sections will contrast the previous method for accessing McIDAS Area files with the ADDE method for accessing image data.

The complete description of each routine can be found in Chapter 5 of the preliminary version of the *McIDAS Programmer's Manual* (10/95).

Routine	Function	Old API
mcaget	opens a connection to read the data block from an image file	opnara
mcaput	opens a connection to write image data to an image file	makara
mcadir	opens a connection to read the directory block from an image file	
mcadrd	reads the directory block from an image file	readd
mcalin	reads the data portion of the current image line	redara
mcapfx	reads the prefix portion of the current image line	redpfx
mcanav	reads the navigation block from an image file	araget
mcacal	reads the calibration block from an image file	araget
mcacrd	reads the comment block from an image file	icget
mcaout	writes the prefix and data portions of an image line	wrtara
mcacou	writes the comment block to an image file	icput
mcadel	deletes an image file	

Traditionally, McIDAS applications that access image objects (Areas) read the directory block before accessing the image elements. **readd** takes an Area number as input and returns the directory block associated with that number. If the application is searching for files that meet certain criteria, the application will do that selection.

The following code segment is an old Area directory read example.

Code segment showing the area directory read loop.

```
c --- I want an Area with the following
      SS = 32          ! GOES-7 Visible
      DAY  = 95001     ! Jan 1 1995
      TIME = 100000    ! 10 Z

c --- search the following range of Areas
      beg_area = 100
      end_area = 199

c --- read Area directory for Area range beg_area to end_area
      do 100 area=beg_area,end_area
      call readd(area,directory)

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

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

c --- do something using the values in the area directory
            ...

      endif
100   continue
```

mcadir and mcadrd

The ADDE interface to the image directory is through **mcadir** and **mcadrd**. **mcadir** opens a connection based on a set of sort clauses for a given dataset name; **mcadrd** returns the directories and comments blocks.

Sort clauses restrict the search based on the image day, in the format YYDDD; 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 **mcadir**.

Sort clause format	**Description**
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 *ss1 ss2*	SSEC sensor source number range
SUBSET *bpos epos*	ADDE position range, or SUBSET ALL to retrieve all directories
TIME *btime etime*	image time range

The sort clause AUX (Auxiliary) provides an enhanced comment block.

When the sort clause AUX YES is 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 = latitude
Center longitude = longitude
Latitude resolution (km) = resolution
Longitude resolution (km) = resolution
Valid calibration for band band = unit
```

Once the connection is opened with **mcadir**, the application makes repeated calls to the **mcadrd** function. As long as the function status returns zero, **mcadrd** has returned an image object directory block and the associated comment block. A function status of one means all directory blocks in the specified dataset matching the sort conditions were returned. Note that a **mcadir** call must precede a call to **mcadrd** as shown in the code segment below.

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

c --- dataset name
      dataset = 'RT/GOES-7'

c --- turn error reporting on
      error_flag = 1

c --- open a connection for the specified dataset
      if( mcadir(dataset,nsort,sorts,error_flag).lt.0 ) return

 100  continue
c --- read an image directory block meeting the search conditions
      status = mcadrd(directory,comment_cards)

c --- read failed
      if( status.lt.0 ) then
         call edest('Failed during directory read of '/dataset,0)
         return

      else if( status.eq.0 ) then

c --- found one, do something using the values in the area directory

         ...
         goto 100

      endif
```

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

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

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

c --- open the Area
      call opnara(area)

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

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

c --- read area lines
      do 100 line = beg_line,end_line
      call redara(area, line-1, beg_elem, nelems, band, data_buffer)

c --- scan the data elements
      do 200 element = beg_elem, end_elem
        ....
200   continue

100   continue

c --- close the area
      call clsara(area)
```

mcaget, mcalin and mcafree

The ADDE version of the image object data block read is based on defining a request for an image sector, which may be 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.

mcafree frees the memory allocated by **mcaget**. It should be called after **mcalin** exhausts the transaction.

mcaget passes the request from the client to the server. The return status shows if the connection is opened. If it is, **mcalin** is called repeatedly to retrieve 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 sector was transferred, the transaction is complete and the connection is closed.

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 spacial, 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.

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

Below is a code segment illustrating the necessary steps for reading the data block of an ADDE dataset image.

Code segment showing the ADDE data block read loop.

```
c --- get the "standard" sort conditions
      if( mcasort(nsorts,sorts,parm_pos).lt.0 ) then
         call edest('Failed to return standard sort parms',0)
         return
      endif

c --- get remaining (non-standard) sort conditions
      sorts(nsort+1) = "SIZE 100 100"
      nsort = nsort+1

c --- set the format of the returned data buffer
      format = 'I4'

c --- set the units of the returned data
      unit = 'TEMP'

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

100   continue
c --- read the data block
      status = mcalin(handle, data_buffer)
      if( status.lt.0 ) then
         call edest('Read failed',0)
         return

c --- got a line of data
      else if( status.eq.0 ) then
         ...
         goto 100

      endif


c --- free the handle
      status = mcafree( handle )

      ...
```

mcaget 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 **mcaget**. Units may be any physical quantity valid for the image object type; for example, TEMP or RAD. A list of valid unit identifiers is available to an application through **mcadir** 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 I1 (1 byte/element), I2 (2 bytes/element), and I4 (4 bytes/element).

mcasort

The call to **mcasort** in the above code segment provides general translation of command line keyword parameters into equivalent **mcaget** sort clauses. Any image application level program may call **mcasort** to retrieve the command line keywords and return them as **mcaget** sort clauses. Because the command line keywords that **mcasort** translates were identified as applicable to the majority of image applications, they can be thought of as global keywords. Users can be assured that using the same keyword in different applications will give the same result.

Below is a list of the ADDE image object access keywords and their equivalent sort clauses.

Command line keyword	**MCASORT translated sort clause**	**Remarks**
	AUX YES	always set by MCASORT
BAND=*band*	BAND *band*	only one spectral band in the clause
DAY=*day*	DAY *day*	
LATLON= *lat lon*	LOCATE E*loc lat lon*	if keyword PLACE is not, specified, *loc* defaults to center (EC)
LINELE= *line ele*	LOCATE I*loc line ele*	if keyword PLACE is not specified, *loc* defaults to center (IC)
MAG=*lmag emag*	MAG *lmag emag*	
PLACE=*loc*	none	sets *loc* for the LATLON, LINELE and STATION keywords
RTIME= *bmin emin*	TIME *btime etime*	RTIME overrides TIME
STATION= *stn*	LOCATE E*loc lat lon*	if keyword PLACE is not specified, loc defaults to center (EC)
TIME=*btime etime*	TIME *btime etime*	

Applications rarely access the navigation block directly, except when copying it to another file. 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 **araget** 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 = aradir(35)
      if( aradir(63).eq.0 ) then
         length = pos - aradir(34)
      else
         length = pos - aradir(63)
      endif

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

mcanav

The ADDE interface to the navigation block is through **mcanav**. 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 **mcanav**.

Code segment showing the ADDE image navigation block read.

```
c --- open a connection
      status = mcaget(dataset, nsort, sorts, unit, format,
     &                max_byte, msg_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)
```

Applications use **araget** 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 read.

Code segment showing the area calibration block read.

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

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

      endif
      ...
```

mcacal

mcacal reads the calibration block of an ADDE image dataset. **mcacal** can be called any time after the connection is opened by **mcaget**. 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, msg_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
      ...
```

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 movwc(comment_card, line_out)
         call spout( line_out )

         goto 100
      endif
      ...
```

mcadrd and mcacrd

The ADDE image API has two interfaces for reading the comment block. The first is available through the **mcadir/mcadrd** interface. When a connection is opened with **mcadir**, all **mcadrd** transactions return the comment block for an image object The second interface is available through connections opened by **mcaget**. After all the data is read, **mcacrd** is used to read the comment block. Unlike **icget**, **mcacrd** returns all 80-character per record entries with a single call.

Code segment showing the ADDE image comment block read.

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

100   continue
c --- read the data block
      status = mcalin(handle, data_buffer)
      if( status.lt.0 ) then
         call edest('Read failed',0)
         return

c --- get a line of data
      else 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
      ...
```

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 **makara**. 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 filed with **icput**.

Code segment showing the write.

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

c --- fill the essential directory block entries
      directory_block(2) = 4                ! area version
      directory_block(3) = sss              ! satellite number
      directory_block(4) = jday             ! Julian day of image
      directory_block(5) = time             ! nominal start time of image
      directory_block(6) = start_line       ! starting image line number
      directory_block(7) = start_elem       ! starting image element number
      directory_block(9) = num_lines        ! number of lines of image data
      directory_block(10)= num_elems        ! number of data elements per line
      directory_block(11)= num_bytes        ! number of bytes per data element
      directory_block(12)= line_res         ! line resolution
      directory_block(13)= elem_res         ! element resolution
      directory_block(14)= num_bands        ! number of bands
      directory_block(19)= 2**(band-1)      ! band map
      call movcw(memo, directory_block(25))! memo 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)= lev_length       ! length of prefix lev section
      directory_block(52)= lit( stype )     ! sensor source type
      directory_block(53)= lit( ctype )     ! calibration type

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

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

c --- fill the navigation block entries
c     NOTE: "navigation_params" is an array of navigation parameters
c     which describes the geo-location of the elements of the area.
      do 10 i = 1,nav_size
      nav_block(i) = navigation_params(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 zeros(cal_block, cal_size)

c --- fill the calibration block entries
c     NOTE: "calibration_parms" is an array of calibration
c      parameters which transform the data elements to physical units.
      do 20 i = 1,cal_size
      cal_block(i) = calibration_params(i)
```

```
20    continue

c --- write the calibration block to the area
      call araput(area, cal_offset, cal_size*4, cal_array)

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

c --- loop to write image lines to the area
      do 100 line = 1,num_lines

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

c --- write a line of data to the area
c     "data_array" is a (num_lines) by (num_elems) array of data
c      elements each of which is (num_bytes) long. The elements
c      represent data for (band) from the sensor numbered (sss) on
c      (jday) at (time).
      call wrtara(area, line-1, data_array(line))

100   continue

c --- close the area
      call clsara( area )

c --- write the comment block
      call getday( day )
      call gettim( time )
      cday = cfu( day )
      ctime = cfu( time )
      comment = cday(1:5)//' '//ctime(1:6)//' This is a comment '
      call icput( area, comment)
```

mcaput and mcaout

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 **mcaput**, which 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**.

mcacou

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 image object write.

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

c --- create a comment card
      call getday( day )
      call gettim( time )
      cday = cfu( day )
      ctime = cfu( time )
      comment = cday(1:5)//' '//ctime(1:6)//' This is a comment '
      ncard = ( len_trim(comment) / 80 ) + 1

c --- fill the essential directory block entries
      directory_block(2) = 4                ! version
      directory_block(3) = sss              ! satellite number
      directory_block(4) = jday             ! Julian day of image
      directory_block(5) = time             ! nominal start time of image
      directory_block(6) = start_line       ! starting image line number
      directory_block(7) = start_elem       ! starting image element number
      directory_block(9) = num_lines        ! number of lines of image data
      directory_block(10)= num_elems        ! number of data elements per line
      directory_block(11)= num_bytes        ! number of bytes per data element
      directory_block(12)= line_res         ! line resolution
      directory_block(13)= elem_res         ! element resolution
      directory_block(14)= num_bands        ! number of bands
      directory_block(19)= 2**(band-1)      ! band map
      call movcw(memo, directory_block(25))! memo 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)= lev_length       ! length of prefix lev section
      directory_block(52)= lit( stype )     ! sensor source type
      directory_block(53)= lit( ctype )     ! calibration type
      directory_block(63)= cal_offset       ! byte offset to the cal block
      directory_block(64)= ncard            ! number of comment cards

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

c --- fill the navigation block entries
c     NOTE: "navigation_params" is an array of navigation parameters
c     which describes the geo-location of the elements of the
c     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
c     NOTE: "calibration_parms" is an array of calibration
c     parameters which transform the data elements to physical
c     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 '//cfu(position)

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

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

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

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

100   continue

c --- write the comment block
      if( mcacou( comment ).ne.0 ) then
         call edest('Failed to write comment block',0)
         return
      endif
```

Gridded data

The access to gridded data in ADDE is different from the traditional method. There is an option that minimizes the differences to allow using the new procedures without extensive changes to existing code, but this is not a recommended long-term solution.

Software written in the past that used information within grids was required to know the following:

- the grid file number, which contained the grid
- the grid number, which contained the grid of data

The ADDE grid software allows a user to access grids by the data contained within them. This data is described, for example, by the parameter (T, Z, etc.), level (SFC, 850, 500, etc.), and day and time. Knowledge of the actual number of the grid and/or grid file where the data resides is not necessary. However, accessing data using the number of the grid or grid file is, of course, still possible.

McIDAS grid data is still stored in grid files; however, the applications are not cognizant of these data structures. It is sufficient to know there is a grid in a dataset that can be retrieved. Datasets can have many positions, or just one position. The former is the case if a dataset spans many grid files; the latter if the dataset consists of just one grid file. The position number of the dataset defines which grid file in that dataset is being accessed.

The grid data API routines are listed below by function, along with the equivalent routines for the old API. The complete description of each routine can be found in Chapter 4 of the preliminary version of the *McIDAS Programmer's Manual* (10/95).

Routine	Function	Old API
mcgdir	opens a connection to read grid headers and grid file headers	
mcgget	opens a connection to read grids	
mcgput	opens a connection to write grids	
mcgfdrd	reads the grid file headers	
mcgdrd	reads grid headers	igget
mcgridc	receives grids (C 2-D array ordered)	
mcgridf	receives grids (Fortran 2-D array ordered)	igget
mcgoutc	writes grids (C 2-D array ordered)	
mcgoutf	writes grids (Fortran 2-D array ordered)	igput

Both **mcgget** and **mcgdir** search local or remote machines for grids that match given search conditions. These conditions are called *sorts* and they are passed to the server in a sort clause. As parameters are added to the sort clause, the search is refined and, presumably, fewer grids are matched.

If the sort clause is passed to **mcgget**, then the grids matching the sort conditions and the corresponding grid headers are sent to the application. If the sort clause is passed to **mcgdir**, the grid file header and the grid headers of the grids, but not the grids themselves, are sent back. For both **mcgget** and **mcgdir**, entries are provided to read the data sent (**mcgridf** and **mcgridc** for **mcgget**; **mcgfdrd** and **mcgdrd** for **mcgdir**).

The table below enumerates the valid sort clauses. The **mcgsort** function translates most of the user-entered keywords into sort clauses.

Sort clause format	Description
LEV *lev1 .. levn*	limits the search to particular levels
PARM *parm1 .. parmn*	limits the search to particular parameters
DAY *day1 .. dayn*	limits the search to specified days, YYYYDDD or YYDDD
TIME *time1 .. timen*	limits the search to specific times, HHMMSS
DRANGE *bday eday inc*	specifies a range of days (with increment) to search
TRANGE *btim etim inc*	specifies a range of times (with increment) to search
SRC *src1 .. srcn*	specifies the source of the data: MRF, NGM, etc.
FHOUR *hour1 .. hourn*	specifies the valid forecast hours for a model run, HHMMSS
FDAY *day1 .. dayn*	specifies the days on which a forecast is valid
FTIME *time1 .. timen*	specifies the times at which a forecast is valid
FRANGE *bhour ehour inc*	specifies a range of forecast hours, with increment
GRID *bgrid egrid*	range of grid numbers; supersedes all the sort conditions above
POS *pos*	position number in the dataset
NUM *num*	retrieves *num* grids, or ALL (default=first match)

mcgsort

The utility subroutine, **mcgsort**, retrieves sort conditions from the command line related to information in the grid header. All but the POS and NUM sort clauses are processed by **mcgsort**. When **mcgsort** picks up the keyword GRID, the grid number itself, all other sort conditions are bypassed.

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. If rep_flag is greater than zero, more than one LEV, PARM, DAY, TIME, SRC, FDAY, FTIME, GRID may be specified. In the case of GRID, though, a range is specified instead of a list. The rep_flag must be greater than zero to use TRANGE and DRANGE. Note that FHOUR and FRANGE can always retrieve more than one grid.

Any application level program may call **mcgsort** to retrieve the command line keywords and return them as **mcgget** sort clauses. Note that the syntax is similar to what the user enters on the command line. For example, the user might enter the keyword TIME=12, **mcgsort** changes that to TIME 120000.

Suppose an application was written to compute a parameter based on the temperature at 850 mb and 500 mb from a 12-hour forecast for the MRF model 00 Z run. The application would have to read the grid headers and determine which grids matched the specification, analogous to the image directory example presented earlier.

The old API for reading grids is a call to **igget** with the grid file and grid number as input. This is used by the majority of McIDAS grid applications and is shown in the code segment below.

Code segment for searching for particular grids.

```
c
c---- Determine grid file to search:  gridfile
c---- Determine maximum number of grids in gridfile:  maxgrid
c
      do 100 gridnum = 1, maxgrid

      status =  igget(gridfile, gridnum, maxsize, grid, nrows, ncols, header)

      if( header(parm_index) .ne. lit('T   ')) go to 100
      if( header(src_index)  .ne. lit('MRF ')) go to 100
      if( header(vt_index)   .ne. 120000)      go to 100
      if( header(time_index) .ne. 0)           go to 100
      if( header(lev_index)  .ne. 850 .or.
     &    header(lev_index)  .ne. 500)         go to 100
c
c---- Found a match, do something with the grid
c
     ..............

100   continue
```

On McIDAS-MVS there is a function **fndgrd**, also used by the grid software in the DDE Demo package, which uses selection criteria similar to the sort clause for retrieving grids. This was used as the basis for retrieving grids in ADDE. A sample code segment is shown below.

Code segment for searching for particular grids in ADDE

```
c
c--- Determine the name of the dataset where MRF grids are stored
c
      name = 'RT/MRF'
c
c--- Set up sorts array for grids wanted.
c
      sorts(1) = 'PARM T'
      sorts(2) = 'LEV 850 500'
      sorts(3) = 'FHOUR 120000'
      sorts(4) = 'SRC MRF'
c
c--- The default is to send only the first grid that matches unless
c--- NUM ALL is specified
c
      sorts(5) = 'NUM ALL'
      nsorts = 5

      status=mcgget(name,nsorts,sorts,' ','I4',maxsiz,msgflg,numgrids,numbytes)
c
c--- A status is set to indicate success or failure
c--- The number of grids that match (numgrids) is returned
c--- If numgrids is more or less than expected, change the sort clause or
c--- dispose of unneeded grids when they are returned
c
c--- Now read the grids
c
      do 100 I = 1, numgrids

      status = mcgridf(grid, header)
c
c--- Do something with the grid
c
        ..................
100   continue
```

ADDE servers

This section contains information specific to the ADDE servers.

Sending data

It is the server's responsibility to send only valid data back through the pipe to the client. Calls to **m0sxsend** send the data. The server must never write to stdout since the data is passed between processes via stdin and stdout. For image data, the code provided for the exercise includes a template that can be used when writing any new image data server. The data server is mugaget.c **[A1-A731]**and the directory server is mugadir.c **[B1-B238]**. The additional code and include files support these modules. As you will see in the exercise, these templates provide a framework for the applications programmer to add the data specific code. A template for grids is not available at this time.

Performance

The client can request that data be sent back as 1, 2, or 4 bytes/data value through the *form* parameter in the **mcaget** call. The server picks up this value from the SPAC keyword **[A138]**. Because the server has knowledge of the data, it should recognize that a smaller size can be used and adjust the directory entry, word 12, accordingly. By optimally packing the data, the amount of bytes transferred across the network is reduced. For example, the unit BRIT (screen brightness) is requested when an application wants the grayscale value for image data. This is always a 1 byte quantity. If the value from SPAC is not 1, the server should set directory word 12 to a value of 1 before sending the directory, and pack the data to 1 byte. The client interface (**mcaget, mcalin**) will recognize that what was requested was not returned, and will unpack the data to the application's specification.

To report an error back to the application, the server sets an error flag in the request block, fills an error string, and calls **m0sxdone**. In Fortran, set word 43 in the request block to the error code, and words 44 through 63 to an error message. In C, fill in the *returncode* and *errormsg* fields in the **servacct** structure (in servacct.h).

The following are the currently defined error codes:

Code	Source	Condition
0	various	Successful return
+1	various	End-of-data condition; no data was returned
-1	serviceman	Accounting data not acceptable to this host
-2	server_main	Transaction type not supported by this host
-5	lwpr	Can't find requested data
-10	gget	Client MAXWDS is too large for server
-11	gget	fndgrd could not find the grid file
-12	gget	fndgrd could not find the grid
-13	gget	igget call failed
-17	gget	A unit conversion requested couldn't be done
-18	gget	Unknown sort condition was specified
-19	gget	GRID and PARM can't both be specified on sort
-20	mdks	Can't open MD file
-21	mdks	One or more requested return keys is not available
-22	mdks	There was an error while calling mdsin
-23	mdks	An invalid unit was on a sort specification
-24	mdks	An invalid key was specified on a sort condition
-25	mdks	A unit conversion was requested that could not be done
-26	mdks	Error specifying a LIST as a sort condition
-27	mdks	An invalid format specifier was given: not I, F, or C
-28	mdks	Too many sort clauses were specified; the current max is 20
-29	mdks	The SIZE parameter is smaller than the return vector
-30	aget	No area found to fit search criteria
-31	aget	Navigation error
-32	aget	Requested data not in area
-33	aget	Bad size requested
-34	aget	Band not present
-35	aget	Bad descriptor in a SORT clause
-36	aget	MAXBYT too small for returned line
-37	aget	Illegal format requested, not I1, I2, I4

Code	Source	Condition
-38	aget	An entry point of mcaget was called with a bad handle
-39	aget	Too many Areas open at a time
-40	aget	Cal not present
-41	aget	SU= name not found
-42	aget	More than 200 Areas and time-ordered search were requested
-50	adir	Can't resolve Area names
-51	adir	No Areas found fitting selection criteria
-52	adir	Can't search this many Area directory entries
-53	adir	Bad TIME clause
-80	atok	Area WQP protection violation
-81	atok	Area number out of valid range
-82	atok	Name of area object was not recognized
-95	writ	Write to server failed
-96	read	Communications with the server timed out
-97	read	Communications with the server are terminated
-98	cxout	cxcomm could not find program module - install err
-99	clserv	soc_init() failed, TCPIP not installed or not active
-100	mcserv	Cannot connect to foreign host
-101	mcserv	socket() call failed; this should not happen
-102	mcserv	Initial transmit to server failed; host or network died
-103	cxout	cxaddr found a bad SERVER.RTE file
-104	mcserv	Bad command arguments to clserv
-105	mcserv	Cannot open null device (should not happen)
-106	mcserv	dup() returned an error (should not happen)
-107	mcserv	server running as root: etc/inetd/conf in error
-108	mcserv	Cannot find password entry
-109	mcserv	malloc() failed (should not happen)
-110	mcserv	putenv() failed (should not happen)
-111	mcserv	error in prefixing MCPATH
-112	mcserv	error prefixing PATH
-113	mcserv	error trying to chdir
-114	mcserv	server requested was not found
-115	mcserv	server requested cannot execute
-116	mcserv	wrong protocol version found

Debugging

Since the servers are started indirectly through **inetd** and **mcserv**, debugging can be problematic. A call to **m0sxtrce** from within the server will write messages to the trce file if the trace flag is set. In Fortran, the trace flag is a single integer variable in COMMON/TRACE/. In C, declare **extern int trace_**. In both cases, setting the variable to one turns on the trace; setting it to zero suppresses the messaging.

When developing server and client applications, invariably the *wanted..did* error message will be emitted by the client application. This means the server did not send as much data as the client expected. The usual cause of this is on the server side, since the server must calculate and notify the client how many bytes will be sent, and then send the data.

'Under the Hood'

The following describes what occurs under the hood on Unix workstations when a request is made to another Unix workstation. You don't have to understand this to be able to write ADDE applications or servers.

When a client requests a connection to a server, the request causes the creation of a pipe, a *fork*, and the *exec* of the ADDE communications module, **mcserv**. The client transmits over the pipe, and then receives on it. The first utterance on the pipe from the client is a 16-byte preamble, organized as four quantities of four bytes each. They are:

Field	Contents
version number of the protocol	0x0001
IP address of the server machine	
port number	500
service name; for example, **aget**	four ASCII characters

mcserv examines the server address. If it indicates the request will be handled locally, it *execs* a server, based on the service name. This server inherits the pipe, and does all further communication with the client.

If the IP address signifies a remote server, **mcserv** continues, and acts as a pipe extender, using TCP/IP to the remote system. It next reads the 160-byte request block, whose fields are as follows:

Field	Length, in bytes
server IP address	4
server port	4
client IP address	4
user initials	4 ascii
project number	4
password	12 ascii (now ignored)
service name	4 ascii
input data length	4
request text	120 ascii

mcserv attempts to connect to the port and IP address of the server. If it fails, it reads the number of bytes equal to the input data length from the pipe to empty it, and sends a 92-byte trailer record back to the client indicating that the connect failed.

If **mcserv** succeeds in connecting to the port, it first sends a resynthesized 16-byte preamble and then the 160-byte request block to the server. **mcserv** then reads and sends the number of bytes stored in the input data length field. At this point, all the information has been sent to the server. **mcserv** continues as an intermediary between the client application and the server by copying the bytes sent by the server to the pipe being read by the application.

On the server machine, a **mcserv** is started by **inetd**. It goes through the same steps, except this time the service is found to be local. **mcserv** execs the server based on the service name, which reads the request and sends the response. When the server is finished sending its response, it sends the 92-byte trailer block, and exits.

The size of data sent to or from the server may be many megabytes. The design is explicitly stream oriented, so both the client and the server can be working simultaneously. The server locates the data and transmits it to the client via a pipe and/or TCP/IP. The client reads out of the pipe and operates on the data. Intermediate storage of the data is not needed on either end for the whole amount of data being sent. Since the pipe is a finite size, the server will wait to write if the pipe is full or the client will wait to read if the pipe is empty. If two minutes elapse with no activity on the pipe, the process stops. The process on the other end of the pipe also stops at this time.

Exercise

The image server consists of five main parts:

- main to all servers: subserv.c **[E1-E102]**
- directory server: mugadir.c **[B1-B238]**
- data server: mugaget.c **[A1-A731]**
- MUG-specific functions: mcmugutil.c **[C1-C1515]**
- generic functions: mcservutil.c

These modules have three include files: mug.h **[D1-D210]**, servacct.h and servutil.h **[F1-F136]**. The only one unique to your server development is mug.h, which contains error messages/codes, default values and struct declarations.

The only piece from the above list that you will be concerned with is the fourth one: MUG-specific functions. This source file is based on seven basic functions that you must write whenever a new format of image data will be served:

Function	**Description**	
IsMugFormat	validates the file format	**[C412-C448]**
ReadMugDir	reads a directory	**[C449-C600]**
MugNavImgToEarth	converts line/elem to lat/lon	**[C601-C691]**
MugNavEarthToImg	converts lat/lon to line/elem	**[C692-C782]**
ReadMugLine	reads a line of data	**[C1047-C1209]**
ReadMugCalCod	reads a calibration codicil	**[C1210-C1246]**
ReadMugNavCod	reads a navigation codicil	**[C1247-C1369]**

The other functions in the file are based on one or more of the above functions. They will also need to be written in future server development, but the existing ones in mcmugutil.c can be used as templates.

In the interest of time, you will only write the function to read a line of data (ReadMugLine). In addition, you will receive the basic skeleton including the interface and data calibration. Keep these three things in mind when completing the function:

- All lines are the same size.
- The function must handle skipped lines.
- Each file has three lines of header at the top.

Below is an example of the ReadMugLine interface.

```
int
ReadMugLine(char *name, READPARM *read, int band, short *buf, char *err)

/*
 * read a line of a MUG Training Course image
 *
 * name    - filename to read
 * read    - READPARM struct containing read specs
 * band    - band number of elements to read
 * buf     - buffer containing image data
 * err     - error string to return on failure
 *
 */
```

The READPARM struct contains specifications that may be needed in the function. It is also a way to add parameters that may be needed in the future without changing the interface of the function. A sample READPARM struct is shown below. You will not use everything contained in the struct.

```
typedef struct READPARM_
{
        char    src_type[4];    /* source type (GVAR, MSAT, ...)          */
        char    des_unit[4];    /* destination units (RAW, BRIT, ...)     */
        char    src_unit[4];    /* source type (RAW, BRIT, ...)           */

        int     begele;         /* beginning element                      */
        int     beglin;         /* beginning line                         */
        int     bufsiz;         /* size of buffer to read                 */
        int     des_len;        /* destination byte length of one pixel   */
        int     elem_res;       /* resolution in element direction        */
        int     line_res;       /* resolution in line direction           */
        int     maxele;         /* last element in image                  */
        int     maxlin;         /* last line in image                     */
        int     minele;         /* first element in image                 */
        int     minlin;         /* first line in image                    */
        int     numband;        /* number of bands in image               */
        int     numele;         /* number of elements to read             */
        int     numlin;         /* number of lines to read                */
        int     src_len;        /* source byte length of one pixel        */
        int     ul_elem;        /* elem in upper left corner of image     */
        int     ul_line;        /* line in upper left corner of image     */
} READPARM;
```

After you've written ReadMugLine, you may compile it with the Makefile provided. Below are the options available for this Makefile:

Makefile options	**Description**
make mugadir	make directory server
make mugaget	make data server
make	make both servers (default)
make clean	clean the environment of relevant .o files

The Makefile has an option to set a DEBUG flag, which you'll want to set for development. It causes a file named trce to be written in the first writable directory in your MCPATH. You may add lines to this trace file with a call to the function, m0sxtrce_(char *, FsLen *) in your source code.

To turn off the trace (debugging) from the servers, you only need to unset the DEBUG flag in the Makefile, make clean and make. You do not need to remove or comment any calls to m0sxtrce_ in your source code.

You may also notice that main to your servers (in this case mugadir.c and mugaget.c) is always the code contained in subserv.c. The Makefile automatically compiles subserv.c as the main to your server. You must be careful to adhere to the Makefile compilation sequence in any future server development.

See **[C1047-C1209]** for a possible solution to ReadMugLine.

Sample code

mugaget.c

```
A    1: #include      <stdio.h>
A    2: #include      <stdlib.h>
A    3: #include      <strings.h>
A    4: #include      "mug.h"
A    5:
A    6: int
A    7: mugaget(servacct *control)
A    8: {
A    9:
A   10: const char      *dir;           /* name of directory to search  */
A   11: const char      *dum;           /* dummy for arg fetchers       */
A   12: const char      *img_place;     /* image placement key (EC..EU) */
A   13:
A   14: char       dbg[MAX_ERR_LEN];    /* textual debug message        */
A   15: char       err[MAX_ERR_LEN];    /* err string from SelectImages */
A   16:
A   17: double      p_lat;              /* center lat of returned image */
A   18: double      p_lon;              /* center lon of returned image */
A   19:
A   20: float      p_elem;              /* center elem of returned image*/
A   21: float      p_line;              /* center line of returned image*/
A   22:
A   23: int       cards[MAX_NUM_CARDS*4];/* mcidas comment cards        */
A   24:
A   25: int       a_elem;              /* requested area element       */
A   26: int       a_line;              /* requested area line          */
A   27: int       aradir[IMG_DIR_LEN]; /* mcidas area directory        */
A   28: int       band;                /* band number to read          */
A   29: int       base_res;            /* base resolution              */
A   30: int       begele;              /* beg element of ret image     */
A   31: int       bday, eday;          /* beginning/ending day         */
A   32: int       bpos, epos;          /* beginning/ending position    */
A   33: int       bss,  ess;           /* beginning/ending ss          */
A   34: int       btim, etim;          /* beginning/ending time        */
A   35: int       bytes_per_line=0;    /* number of bytes/image line   */
A   36: int       bytes_per_pix=0;     /* number of bytes/pixel        */
A   37: int       bytes_to_send=0;     /* number of data bytes to send */
A   38: int       bytes_to_zero=0;     /* number of bytes to zero      */
A   39: int       calcod[MAX_CAL_LEN]; /* mcidas calibration codicil   */
A   40: int       curline;             /* loop variable for lines      */
A   41: int       data_bytes;          /* number of bytes in read      */
A   42: int       elem_res;            /* element resolution           */
A   43: int       elem_to_send;        /* number of elem after resred  */
A   44: int       four=4;              /* size of total_bytes to send  */
A   45: int       i;                   /* loop variable                */
A   46: int       istat;               /* function return status       */
A   47: int       len_calcod=0;        /* length of mcidas cal codicil */
A   48: int       len_navcod=0;        /* length of mcidas nav codicil */
A   49: int       line_res;            /* line resolution              */
A   50: int       navcod[MAX_NAV_LEN]; /* mcidas navigation codicil    */
A   51: int       nelems;              /* number of elements to send   */
A   52: int       nlines;              /* number of lines to send      */
A   53: int       one=1;               /* the number one               */
A   54: int       orig_nelems;         /* unalterd # of lines to send  */
A   55: int       orig_nlines;         /* unalterd # of lines to send  */
A   56: int       p_elem_off;          /* center elem offset           */
A   57: int       p_line_off;          /* center line offset           */
A   58: int       pt_elem;             /* elem number to navigate      */
```

```
A   59: int      pt_line;                /* line number to navigate      */
A   60: int      spac;                   /* byte size of one pixel       */
A   61: int      total_bytes=0;          /* number of bytes to send      */
A   62: int      ul_elem;                /* upper left elem of ret image */
A   63: int      ul_line;                /* upper left line of ret image */
A   64: int      zeros_on_left;          /* number of zero bytes on left */
A   65:
A   66: short      *data=NULL;           /* line of image data           */
A   67:
A   68: CRITERIA *request=NULL;
A   69: FILELIST *satisfy=NULL;
A   70: READPARM *read=NULL;
A   71:
A   72: strcpy(dbg, "starting mugaget");
A   73: m0sxtrce_(dbg, strlen(dbg));
A   74:
A   75: /*
A   76:  * directory to be searched
A   77:  */
A   78:
A   79: istat = Mcargquo(0, &dir);
A   80:
A   81: /*
A   82:  * beginning position
A   83:  */
A   84:
A   85: istat = Mcargint(0, " ", 2, 0, 999, -999, &bpos, &dum);
A   86:
A   87: /*
A   88:  * set the ending position equal to the beginning
A   89:  * position. the underlying function, SelectImages, allows
A   90:  * a range to be specified, but the current ADDE protocol
A   91:  * does not support this.
A   92:  */
A   93:
A   94: epos = bpos;
A   95:
A   96: /*
A   97:  * image placement key (EC, EU, ...)
A   98:  */
A   99:
A  100: istat = Mcargstr(0, " ", 3, " ", &img_place);
A  101:
A  102: /*
A  103:  * base resolution
A  104:  */
A  105:
A  106: istat = Mcargint(0, " ", 6, 1, 999, -999, &base_res, &dum);
A  107:
A  108: /*
A  109:  * number of lines and elements to read
A  110:  */
A  111:
A  112: istat = Mcargint(0, " ", 7, DEF_NUM_LINES, 999, -999, &nlines, &dum);
A  113: istat = Mcargint(0, " ", 8, DEF_NUM_ELEMS, 999, -999, &nelems, &dum);
A  114:
A  115: orig_nlines = nlines;
A  116: orig_nelems = nelems;
A  117:
A  118: /*
A  119:  * band number to read
A  120:  */
A  121:
A  122: istat = Mcargint(0, "BAN.D", 1, 1, 999, -999, &band, &dum);
A  123:
A  124: /*
A  125:  * line and element resolution
A  126:  */
A  127:
A  128: istat = Mcargint(0, "LMA.G", 1, base_res, 999, -999, &line_res, &dum);
A  129: istat = Mcargint(0, "EMA.G", 1, base_res, 999, -999, &elem_res, &dum);
A  130:
```

```
A  131: line_res = abs(line_res);
A  132: elem_res = abs(elem_res);
A  133:
A  134: /*
A  135:  * size of one pixel (spac)
A  136:  */
A  137:
A  138: istat = Mcargint(0, "SPA.C", 1, 0, -1, 0, &spac, &dum);
A  139:
A  140: /*
A  141:  * this server will only serve one byte data
A  142:  */
A  143:
A  144: if (spac > 1)
A  145:       spac = 1;
A  146:
A  147: /*
A  148:  * beginning and ending time
A  149:  */
A  150:
A  151: istat = Mcargihr(0, "TIM.E", 1,   -1, -1, 240000, &btim, &dum);
A  152: istat = Mcargihr(0, "TIM.E", 2, btim, -1, 240000, &etim, &dum);
A  153:
A  154: /*
A  155:  * beginning and ending day
A  156:  */
A  157:
A  158: istat = Mcargint(0, "DAY", 1,   -1, 999, -999, &bday, &dum);
A  159: istat = Mcargint(0, "DAY", 2, bday, 999, -999, &eday, &dum);
A  160:
A  161: /*
A  162:  * beginning and ending mcidas ss number
A  163:  */
A  164:
A  165: istat = Mcargint(0, "SS", 1,  -1, 999, -999, &bss, &dum);
A  166: istat = Mcargint(0, "SS", 2, bss, 999, -999, &ess, &dum);
A  167:
A  168: sprintf(dbg, "dir = [%s]", dir);
A  169: m0sxtrce_(dbg, strlen(dbg));
A  170:
A  171: sprintf(dbg, "bpos = %d epos = %d", bpos, epos);
A  172: m0sxtrce_(dbg, strlen(dbg));
A  173:
A  174: sprintf(dbg, "btim = %d etim = %d", btim, etim);
A  175: m0sxtrce_(dbg, strlen(dbg));
A  176:
A  177: sprintf(dbg, "bday = %d eday = %d", bday, eday);
A  178: m0sxtrce_(dbg, strlen(dbg));
A  179:
A  180: sprintf(dbg, "bss = %d ess = %d", bss, ess);
A  181: m0sxtrce_(dbg, strlen(dbg));
A  182:
A  183: /*
A  184:  * malloc space to fill a struct
A  185:  * containing search criteria
A  186:  */
A  187:
A  188: request = (CRITERIA *)malloc(sizeof(CRITERIA));
A  189:
A  190: if (request == NULL)
A  191: {
A  192:       (void)strcpy(err, ERR_MSG_MALLOC);
A  193:       return ERR_STAT_MALLOC;
A  194: }
A  195:
A  196: request->begday = bday;
A  197: request->endday = eday;
A  198: request->begtim = btim;
A  199: request->endtim = etim;
A  200: request->begss  = bss;
A  201: request->endss  = ess;
A  202:
```

```
A  203: /*
A  204:  * malloc space to return a linked
A  205:  * list of files meeting CRITERIA
A  206:  */
A  207:
A  208: satisfy = (FILELIST *)malloc(sizeof(FILELIST));
A  209:
A  210: if (satisfy == NULL)
A  211: {
A  212:       (void)strcpy(err, ERR_MSG_MALLOC);
A  213:       return ERR_STAT_MALLOC;
A  214: }
A  215:
A  216: /*
A  217:  * malloc space for the read parameters
A  218:  */
A  219:
A  220: read = (READPARM *)malloc(sizeof(READPARM));
A  221:
A  222: if (read == NULL)
A  223: {
A  224:       (void)strcpy(err, ERR_MSG_MALLOC);
A  225:       return ERR_STAT_MALLOC;
A  226: }
A  227:
A  228: /*
A  229:  * get a list of qualified files
A  230:  */
A  231:
A  232: istat = SelectMugImages((char *)dir, bpos, epos, request, &satisfy, err);
A  233:
A  234: if (istat == FAILURE)
A  235: {
A  236:       (void)strcpy(control->errormsg, err);
A  237:       control->returncode = ERR_STAT_SELECT;
A  238:       return (control->returncode);
A  239: }
A  240:
A  241: /*
A  242:  * no images were found
A  243:  */
A  244:
A  245: if (satisfy == NULL)
A  246: {
A  247:       (void)strcpy(control->errormsg, ERR_MSG_NOIMG);
A  248:       control->returncode = ERR_STAT_NOIMG;
A  249:       return (control->returncode);
A  250: }
A  251:
A  252: /*
A  253:  * loop through the MUG files
A  254:  */
A  255:
A  256: while (satisfy != NULL)
A  257: {
A  258:       /*
A  259:        * read the directories
A  260:        */
A  261:
A  262:       sprintf(dbg, "reading file %s", satisfy->name);
A  263:       m0sxtrce_(dbg, strlen(dbg));
A  264:
A  265:       istat = ReadMugDir(satisfy->name, aradir, err);
A  266:
A  267:       if (istat == FAILURE)
A  268:       {
A  269:            sprintf(dbg, "failure reading file %s", satisfy->name);
A  270:            m0sxtrce_(dbg, strlen(dbg));
A  271:
A  272:            sprintf(dbg, "ERR: %s", err);
A  273:            m0sxtrce_(dbg, strlen(dbg));
A  274:
```

```
A  275:              continue;
A  276:        }
A  277:
A  278:        /*
A  279:         * fill read specifications with
A  280:         * the size of the read buffer,
A  281:         * the min and max number of lines/elements,
A  282:         * the size of a source pixel and
A  283:         * the number of bands in the image
A  284:         */
A  285:
A  286:        read->bufsiz   = READ_BUFFER_SIZE;
A  287:        read->elem_res = elem_res;
A  288:        read->line_res = line_res;
A  289:        read->minlin   = 0;
A  290:        read->minele   = 0;
A  291:        read->maxlin   = aradir[8];
A  292:        read->maxele   = aradir[9];
A  293:        read->src_len  = sizeof(short);
A  294:        read->numband  = aradir[13];
A  295:
A  296:        /*
A  297:         * readjust the number of lines
A  298:         * and elements if needed. a number
A  299:         * of 99999 from the client means that
A  300:         * the entire image was requested
A  301:         */
A  302:
A  303:        if (nlines == 99999)
A  304:        {
A  305:              nlines = aradir[8];
A  306:              orig_nlines = nlines;
A  307:        }
A  308:
A  309:        if (nelems == 99999)
A  310:        {
A  311:              nelems = aradir[9];
A  312:              orig_nelems = nelems;
A  313:        }
A  314:
A  315:        /*
A  316:         * calculate center line and element
A  317:         * of requested image object
A  318:         */
A  319:
A  320:        /*
A  321:         * pick up coordinate parameters
A  322:         */
A  323:
A  324:        switch (img_place[0])
A  325:        {
A  326:              case 'E':      /* earth coordinates      */
A  327:
A  328:                    (void)strcpy(dbg, "earth coord \n");
A  329:                    m0sxtrce_(dbg, strlen(dbg));
A  330:
A  331:                    istat = Mcargdll(0, " ", 4, (double)0,
A  332:                          (double)999, (double)-999, &p_lat, &dum);
A  333:
A  334:                    istat = Mcargdll(0, " ", 5, (double)0,
A  335:                          (double)999, (double)-999, &p_lon, &dum);
A  336:
A  337:                    /*
A  338:                     * convert lat/lon to line/elem
A  339:                     */
A  340:
A  341:                    istat = MugNavEarthToImg(satisfy->name,
A  342:                          (float)p_lat, (float)p_lon,
A  343:                          &p_line, &p_elem,
A  344:                          err);
A  345:
A  346:                    if (istat == FAILURE)
```

```
A  347:                     {
A  348:                           (void)strcpy(control->errormsg, err);
A  349:                           control->returncode = ERR_STAT_NONAV;
A  350:                           return (control->returncode);
A  351:                     }
A  352:
A  353:                     break;
A  354:
A  355:               case 'A':      /* file coordinates      */
A  356:               case 'I':
A  357:               default:
A  358:                     istat = Mcargint(0, " ", 4, 0,
A  359:                                 999, -999, &a_line, &dum);
A  360:
A  361:                     istat = Mcargint(0, " ", 5, 0,
A  362:                                 999, -999, &a_elem, &dum);
A  363:
A  364:                     p_line = (float)a_line;
A  365:                     p_elem = (float)a_elem;
A  366:
A  367:                     break;
A  368:         }
A  369:
A  370:         (void)sprintf(dbg, "place coord = %f %f \n", p_line, p_elem);
A  371:         m0sxtrce_(dbg, strlen(dbg));
A  372:
A  373:         /*
A  374:          * calculate line/elem of upper left corner line/elem
A  375:          */
A  376:
A  377:         switch (img_place[1])
A  378:         {
A  379:               case 'D':      /* lower right      */
A  380:
A  381:                     p_line_off = orig_nlines;
A  382:                     p_elem_off = orig_nelems;
A  383:                     break;
A  384:
A  385:               case 'C':      /* centered      */
A  386:
A  387:                     p_line_off = (orig_nlines / 2);
A  388:                     p_elem_off = (orig_nelems / 2);
A  389:                     break;
A  390:
A  391:               default:      /* upper left      */
A  392:
A  393:                     p_line_off = 1/line_res;
A  394:                     p_elem_off = 1/elem_res;
A  395:                     break;
A  396:         }
A  397:
A  398:         ul_line = (int)p_line - (p_line_off * line_res) + 1;
A  399:         ul_elem = (int)p_elem - (p_elem_off * elem_res) + 1;
A  400:
A  401:         (void)sprintf(dbg, "start coord = %d %d \n", ul_line, ul_elem);
A  402:         m0sxtrce_(dbg, strlen(dbg));
A  403:
A  404:         /*
A  405:          * initialize read struct with
A  406:          * starting line and element
A  407:          */
A  408:
A  409:         read->ul_line = ul_line;
A  410:         read->ul_elem = ul_elem;
A  411:
A  412:         /*
A  413:          * check the bounds of the returned image
A  414:          * to make sure it will contain data
A  415:          * if not, return an error
A  416:          */
A  417:
A  418:         istat = CheckImgBounds(read, orig_nlines, orig_nelems);
```

```
A  419:
A  420:        if (istat == FAILURE)
A  421:        {
A  422:              (void)strcpy(control->errormsg, ERR_MSG_BLANK_IMG);
A  423:              control->returncode = ERR_STAT_BLANK_IMG;
A  424:              return (control->returncode);
A  425:        }
A  426:
A  427:        /*
A  428:         * adjust output area directory
A  429:         */
A  430:
A  431:        aradir[8] = nlines;
A  432:        aradir[9] = 4 * ((nelems + 3) / 4);
A  433:
A  434:        nelems = aradir[9];
A  435:
A  436:        if (spac == 0)
A  437:              spac = aradir[10];
A  438:
A  439:        aradir[10] = spac;
A  440:
A  441:        aradir[11] = aradir[11] * elem_res;
A  442:        aradir[12] = aradir[12] * line_res;
A  443:
A  444:        /*
A  445:         * fill read specifications with
A  446:         * the size of a destination pixel
A  447:         */
A  448:
A  449:        read->des_len = aradir[10];
A  450:
A  451:        /*
A  452:         * read mcidas navigation codicil
A  453:         */
A  454:
A  455:        pt_line =  2 - ul_line;
A  456:        pt_elem =  2 - ul_elem;
A  457:
A  458:        istat = ReadMugNavCod(satisfy->name, navcod,
A  459:                              pt_line, pt_elem, &len_navcod);
A  460:
A  461:        if (istat == FAILURE)
A  462:        {
A  463:              (void)strcpy(control->errormsg, ERR_MSG_NONAV);
A  464:              control->returncode = ERR_STAT_NONAV;
A  465:              return (control->returncode);
A  466:        }
A  467:
A  468:        /*
A  469:         * read mcidas calibration codicil
A  470:         */
A  471:
A  472:        istat = ReadMugCalCod(satisfy->name, calcod, &len_calcod);
A  473:
A  474:        if (istat == FAILURE)
A  475:        {
A  476:              (void)strcpy(control->errormsg, ERR_MSG_NOCAL);
A  477:              control->returncode = ERR_STAT_NOCAL;
A  478:              return (control->returncode);
A  479:        }
A  480:
A  481:        /*
A  482:         * compute total byte transfer to client
A  483:         */
A  484:
A  485:        bytes_per_line =       aradir[9]      *
A  486:                               spac           *
A  487:                               aradir[13]     +
A  488:                               aradir[14];
A  489:
A  490:        total_bytes    =       MAX_CARD_LEN      *
```

```
A  491:                          aradir[63]      +
A  492:                          aradir[33]      +
A  493:                          aradir[8]      *
A  494:                          bytes_per_line;
A  495:
A  496:        /*
A  497:         * send total bytes
A  498:         */
A  499:
A  500:        control->reply_length += total_bytes;
A  501:
A  502:        swbyt4_(&total_bytes, &one);
A  503:
A  504:        sprintf(dbg, "sending %d bytes", total_bytes);
A  505:        m0sxtrce_(dbg, strlen(dbg));
A  506:
A  507:        m0sxsend_(&four, &total_bytes);
A  508:
A  509:        /*
A  510:         * send the directory
A  511:         */
A  512:
A  513:        aradir[0] = satisfy->pos;
A  514:        (void)SendDir(aradir);
A  515:
A  516:        /*
A  517:         * send the nav
A  518:         */
A  519:
A  520:        (void)SendCod(navcod, len_navcod);
A  521:
A  522:        /*
A  523:         * send the cal
A  524:         */
A  525:
A  526:        (void)SendCod(calcod, len_calcod);
A  527:
A  528:        /*
A  529:         * read and send the image one line at a time
A  530:         */
A  531:
A  532:        data_bytes =       orig_nelems      *
A  533:                    read->src_len      *
A  534:                    read->numband      +
A  535:                    aradir[14];
A  536:
A  537:        data_bytes *= read->bufsiz;
A  538:
A  539:        data = (short *)malloc(data_bytes);
A  540:
A  541:        if (data == NULL)
A  542:        {
A  543:              (void)strcpy(control->errormsg, ERR_MSG_MALLOC);
A  544:              control->returncode = ERR_STAT_MALLOC;
A  545:              return (control->returncode);
A  546:        }
A  547:
A  548:        /*
A  549:         * total elements needed
A  550:         */
A  551:
A  552:        curline = ul_line;
A  553:        begele  = ul_elem;
A  554:        read->numlin  = read->bufsiz;
A  555:        bytes_to_send = bytes_per_line;
A  556:        bytes_per_pix = read->des_len * read->numband;
A  557:
A  558:        for (i=1; i<=orig_nlines; i++)
A  559:        {
A  560:
A  561:              /*
A  562:               * if the requested line is less than one
```

```
A  563:                 * or greater than the max number of lines,
A  564:                 * fill the buffer with zeros and return
A  565:                 */
A  566:
A  567:                if (curline < 0 || curline > read->maxlin)
A  568:                {
A  569:                      bytes_to_zero = aradir[9]       *
A  570:                                      read->des_len   *
A  571:                                      read->numband;
A  572:
A  573:                      sprintf(dbg, "zero line %d", curline);
A  574:                      m0sxtrce_(dbg, strlen(dbg));
A  575:
A  576:                      (void)SendZeros(data, bytes_to_zero);
A  577:
A  578:                      /*
A  579:                       * increment the line pointer
A  580:                       */
A  581:
A  582:                      curline += line_res;
A  583:
A  584:                      bytes_to_zero = 0;
A  585:
A  586:                      continue;
A  587:                }
A  588:
A  589:                /*
A  590:                 * if the image requested does not
A  591:                 * contain data on the left side,
A  592:                 * send the appropriate number of
A  593:                 * zeros and adjust the starting
A  594:                 * element
A  595:                 */
A  596:
A  597:                zeros_on_left = 0;
A  598:
A  599:                if (ul_elem < 0)
A  600:                {
A  601:                      bytes_to_zero = abs(ul_elem/elem_res)  *
A  602:                                      read->des_len *
A  603:                                      read->numband;
A  604:
A  605:                      zeros_on_left = bytes_to_zero;
A  606:
A  607:                      sprintf(dbg, "z left %d %d bytes",curline,bytes_to_zero);
A  608:                      m0sxtrce_(dbg, strlen(dbg));
A  609:
A  610:                      (void)SendZeros(data, bytes_to_zero);
A  611:
A  612:                      begele = 0;
A  613:                      nelems = read->maxele;
A  614:
A  615:                      bytes_to_send = bytes_per_line - bytes_to_zero;
A  616:                      bytes_to_zero = 0;
A  617:                }
A  618:
A  619:                /*
A  620:                 * if the image requested does not
A  621:                 * contain data on the right side,
A  622:                 * adjust the number of elements.
A  623:                 * the appropriate number of zero filler
A  624:                 * will be send later (after the read)
A  625:                 */
A  626:
A  627:                /* if (ul_elem/elem_res + aradir[9] > read->maxele/elem_res) */
A  628:
A  629:                if (ul_elem + (aradir[9] * elem_res) > read->maxele)
A  630:                {
A  631:
A  632:                   bytes_to_zero = (ul_elem/elem_res + aradir[9]          -
A  633:                                   (read->maxele/elem_res))              *
A  634:                                   read->des_len                         *
```

```
A  635:                                 read->numband;
A  636:
A  637:                 nelems = read->maxele - ul_elem/elem_res;
A  638:
A  639:                 if (ul_elem < 0)
A  640:                       nelems = read->maxele;
A  641:
A  642:               sprintf(dbg, "zero left %d bytes", zeros_on_left);
A  643:                 m0sxtrce_(dbg, strlen(dbg));
A  644:
A  645:                 bytes_to_send = bytes_per_line      -
A  646:                                 bytes_to_zero      -
A  647:                                 zeros_on_left;
A  648:
A  649:                 sprintf(dbg, "send %d bytes", bytes_to_send);
A  650:                 m0sxtrce_(dbg, strlen(dbg));
A  651:
A  652:                 sprintf(dbg, "zero %d bytes", bytes_to_zero);
A  653:                 m0sxtrce_(dbg, strlen(dbg));
A  654:           }
A  655:
A  656:           /*
A  657:            * fill read specifications with
A  658:            * beginning line/element and
A  659:            * the number of lines and elements
A  660:            * to read
A  661:            */
A  662:
A  663:           read->beglin  = curline;
A  664:           read->begele  = begele;
A  665:           read->numele  = nelems;
A  666:
A  667:           /*
A  668:            * read a line of data
A  669:            */
A  670:
A  671:           istat = ReadMugLine(satisfy->name, read, band, data, err);
A  672:
A  673:           if (istat == FAILURE)
A  674:           {
A  675:                 (void)strcpy(control->errormsg, err);
A  676:                 control->returncode = ERR_STAT_BADLINE;
A  677:                 return (control->returncode);
A  678:           }
A  679:
A  680:           /*
A  681:            * reduce the resolution
A  682:            */
A  683:
A  684:           elem_to_send = nelems / elem_res;
A  685:
A  686:           (void)m0resred_((char *)data, &elem_res,
A  687:                           &elem_to_send, &bytes_per_pix);
A  688:
A  689:           /*
A  690:            * send line of image data
A  691:            */
A  692:
A  693:           (void)SendLine(data, bytes_to_send);
A  694:
A  695:           /*
A  696:            * if we still have bytes_to_zero (set
A  697:            * because the image is padded on the right,
A  698:            * then we send those
A  699:            */
A  700:
A  701:           if (bytes_to_zero != 0)
A  702:           {
A  703:                 (void)SendZeros(data, bytes_to_zero);
A  704:
A  705:                 bytes_to_zero = 0;
A  706:           }
```

```
A  707:
A  708:                 /*
A  709:                  * increment the line pointer
A  710:                  */
A  711:
A  712:                 curline += line_res;
A  713:         }
A  714:
A  715:         /*
A  716:          * send the comment cards
A  717:          */
A  718:
A  719:         if (aradir[63] != 0)
A  720:                 (void)SendCards(aradir, cards);
A  721:
A  722:         /*
A  723:          * increment the pointer in the list of files
A  724:          */
A  725:
A  726:         satisfy = satisfy->next;
A  727: }
A  728:
A  729: return SUCCESS;
A  730:
A  731: }
```

```
B     1: #include      <stdio.h>
B     2: #include      <stdlib.h>
B     3: #include      <strings.h>
B     4: #include      "mug.h"
B     5:
B     6: int
B     7: mugadir(servacct *control)
B     8: {
B     9:
B    10: const char      *dir;           /* name of directory to search */
B    11: const char      *dum;           /* dummy for arg fetchers      */
B    12:
B    13: char      dbg[MAX_ERR_LEN];     /* textual debug message       */
B    14: char      err[MAX_ERR_LEN];     /* err string from SelectImages*/
B    15:
B    16: int      cards[MAX_NUM_CARDS*4];/* mcidas comment cards        */
B    17:
B    18: int      aradir[IMG_DIR_LEN];  /* mcidas area directory       */
B    19: int      bday, eday;           /* beginning/ending day        */
B    20: int      bpos, epos;           /* beginning/ending position   */
B    21: int      bss,  ess;            /* beginning/ending ss         */
B    22: int      btim, etim;           /* beginning/ending time       */
B    23: int      dummy=9999;           /* dummy pos number to send    */
B    24: int      four=4;               /* size of nbytes to send      */
B    25: int      istat;                /* function return status      */
B    26: int      nbytes=0;             /* number of bytes to send     */
B    27: int      one=1;                /* the number one (1)          */
B    28:
B    29: CRITERIA *request=NULL;
B    30: FILELIST *satisfy=NULL;
B    31:
B    32: strcpy(dbg, "starting mugadir");
B    33: m0sxtrce_(dbg, strlen(dbg));
B    34:
B    35: /*
B    36:  * directory to be searched
B    37:  */
B    38:
B    39: istat = Mcargquo(0, &dir);
B    40:
B    41: /*
B    42:  * beginning and ending position
B    43:  */
B    44:
B    45: istat = Mcargint(0, " ", 2,    0, 999, -999, &bpos, &dum);
B    46: istat = Mcargint(0, " ", 3, bpos, 999, -999, &epos, &dum);
B    47:
B    48: /*
B    49:  * if the bpos is negative, reoranize the positions
B    50:  * since the ADDE convention for time ordered requests
B    51:  * is to always have zero as the epos
B    52:  */
B    53:
B    54: if (bpos < 0)
B    55: {
B    56:       epos = bpos;
B    57:       bpos = 0;
B    58: }
B    59:
B    60: /*
B    61:  * beginning and ending time
B    62:  */
B    63:
B    64: istat = Mcargihr(0, "TIM.E", 1,   -1, -1, 240000, &btim, &dum);
B    65: istat = Mcargihr(0, "TIM.E", 2, btim, -1, 240000, &etim, &dum);
B    66:
```

```
B   67: /*
B   68:  * beginning and ending day
B   69:  */
B   70:
B   71: istat = Mcargint(0, "DAY", 1,   -1, 999, -999, &bday, &dum);
B   72: istat = Mcargint(0, "DAY", 2, bday, 999, -999, &eday, &dum);
B   73:
B   74: /*
B   75:  * beginning and ending mcidas ss number
B   76:  */
B   77:
B   78: istat = Mcargint(0, "SS", 1,  -1, 999, -999, &bss, &dum);
B   79: istat = Mcargint(0, "SS", 2, bss, 999, -999, &ess, &dum);
B   80:
B   81: sprintf(dbg, "dir = [%s]", dir);
B   82: m0sxtrce_(dbg, strlen(dbg));
B   83:
B   84: sprintf(dbg, "bpos = %d epos = %d", bpos, epos);
B   85: m0sxtrce_(dbg, strlen(dbg));
B   86:
B   87: sprintf(dbg, "btim = %d etim = %d", btim, etim);
B   88: m0sxtrce_(dbg, strlen(dbg));
B   89:
B   90: sprintf(dbg, "bday = %d eday = %d", bday, eday);
B   91: m0sxtrce_(dbg, strlen(dbg));
B   92:
B   93: sprintf(dbg, "bss = %d ess = %d", bss, ess);
B   94: m0sxtrce_(dbg, strlen(dbg));
B   95:
B   96: /*
B   97:  * malloc space to fill a struct
B   98:  * containing search criteria
B   99:  */
B  100:
B  101: request = (CRITERIA *)malloc(sizeof(CRITERIA));
B  102:
B  103: if (request == NULL)
B  104: {
B  105:       (void)strcpy(err, ERR_MSG_MALLOC);
B  106:       return ERR_STAT_MALLOC;
B  107: }
B  108:
B  109: request->begday = bday;
B  110: request->endday = eday;
B  111: request->begtim = btim;
B  112: request->endtim = etim;
B  113: request->begss  = bss;
B  114: request->endss  = ess;
B  115:
B  116: /*
B  117:  * malloc space to return a linked
B  118:  * list of files meeting CRITERIA
B  119:  */
B  120:
B  121: satisfy = (FILELIST *)malloc(sizeof(FILELIST));
B  122:
B  123: if (satisfy == NULL)
B  124: {
B  125:       (void)strcpy(err, ERR_MSG_MALLOC);
B  126:       return ERR_STAT_MALLOC;
B  127: }
B  128:
B  129: istat = SelectMugImages((char *)dir, bpos, epos, request, &satisfy, err);
B  130:
B  131: if (istat == FAILURE)
B  132: {
B  133:       (void)strcpy(control->errormsg, err);
B  134:       control->returncode = ERR_STAT_SELECT;
B  135:       return (control->returncode);
B  136: }
B  137:
B  138: /*
```

```
B  139:  * no images were found
B  140:  */
B  141:
B  142: if (satisfy == NULL)
B  143: {
B  144:       (void)strcpy(control->errormsg, ERR_MSG_NOIMG);
B  145:       control->returncode = ERR_STAT_NOIMG;
B  146:       return (control->returncode);
B  147: }
B  148:
B  149: /*
B  150:  * loop through the MUG files
B  151:  */
B  152:
B  153: while (satisfy != NULL)
B  154: {
B  155:       /*
B  156:        * read the directories
B  157:        */
B  158:
B  159:       sprintf(dbg, "reading file %s", satisfy->name);
B  160:       m0sxtrce_(dbg, strlen(dbg));
B  161:
B  162:       istat = ReadMugDir(satisfy->name, aradir, err);
B  163:
B  164:       if (istat != FAILURE)
B  165:       {
B  166:             /*
B  167:              * build the comment cards
B  168:              */
B  169:
B  170:             sprintf(dbg, "success reading file %s", satisfy->name);
B  171:             m0sxtrce_(dbg, strlen(dbg));
B  172:
B  173:             istat = ReadMugCards(satisfy->name, aradir, cards, err);
B  174:
B  175:             if (istat == FAILURE)
B  176:             {
B  177:                   sprintf(dbg, "CARD ERR: %s", err);
B  178:                   m0sxtrce_(dbg, strlen(dbg));
B  179:             }
B  180:
B  181:       }
B  182:       else
B  183:       {
B  184:             sprintf(dbg, "failure reading file %s", satisfy->name);
B  185:             m0sxtrce_(dbg, strlen(dbg));
B  186:
B  187:             sprintf(dbg, "ERR: %s", err);
B  188:             m0sxtrce_(dbg, strlen(dbg));
B  189:
B  190:             continue;
B  191:       }
B  192:
B  193:       /*
B  194:        * transform to comm protocol
B  195:        */
B  196:
B  197:       nbytes = (IMG_DIR_LEN * 4) + 4 + (aradir[63] * MAX_CARD_LEN);
B  198:
B  199:       control->reply_length += nbytes;
B  200:
B  201:       swbyt4_(&nbytes, &one);
B  202:
B  203:       sprintf(dbg, "sending %d bytes", nbytes);
B  204:       m0sxtrce_(dbg, strlen(dbg));
B  205:
B  206:       m0sxsend_(&four, &nbytes);
B  207:
B  208:       /*
B  209:        * send four bytes of dummy
B  210:        */
```

```
B  211:
B  212:         m0sxsend_(&four, &dummy);
B  213:
B  214:         aradir[0] = satisfy->pos;
B  215:
B  216:         /*
B  217:          * send the directory
B  218:          */
B  219:
B  220:         (void)SendDir(aradir);
B  221:
B  222:         /*
B  223:          * send the comment cards
B  224:          */
B  225:
B  226:         if (aradir[63] != 0)
B  227:               (void)SendCards(aradir, cards);
B  228:
B  229:         /*
B  230:          * increment the pointer in the list of files
B  231:          */
B  232:
B  233:         satisfy = satisfy->next;
B  234: }
B  235:
B  236: return SUCCESS;
B  237:
B  238: }
```

```
C     1: #include      <dirent.h>
C     2: #include      <stdio.h>
C     3: #include      <stdlib.h>
C     4: #include      <strings.h>
C     5: #include      "mug.h"
C     6:
C     7: char      dbg[MAX_ERR_LEN]; /* debug message                    */
C     8:
C     9: /
*********************************************************************/
C    10:
C    11: int
C    12: SelectMugImages(char *dir, int bpos, int epos,
C    13:              CRITERIA *request, FILELIST **satisfy, char *err)
C    14: {
C    15:
C    16: char  fullname[MAX_NAME_LEN];/* fully specified name of a file    */
C    17: char  name[MAX_NAME_LEN]; /* name of a file                        */
C    18:
C    19: int   ALL;               /* tag to select all images in dir       */
C    20: int   aradir[IMG_DIR_LEN];/* mcidas area directory                 */
C    21: int   curpos=0;          /* initial position number               */
C    22: int   istat;             /* function return status                */
C    23: int   one=1;             /* the number one (1)                    */
C    24: int   pos=0;             /* stored position number                */
C    25: int   strmatch;          /* string comparison variable            */
C    26: int   valid_type;        /* flag for defining a valid format      */
C    27:
C    28: DIR *dirlist=NULL;       /* directory listing of files to test */
C    29: struct dirent *dirfile;  /* struct containing file from readdir*/
C    30:
C    31: FILELIST *head=NULL;      /* top of list of files in directory  */
C    32: FILELIST *cur=NULL;       /* top of list of files in directory  */
C    33:
C    34: FILELIST *rawlist=NULL;   /* top of list of files in directory  */
C    35:
C    36: FILELIST *test=NULL;      /* top of list of files to test       */
C    37: FILELIST *timlist=NULL;   /* top of list of time ordered files  */
C    38:
C    39: /*
C    40:  * the integer representation of 'ALL ' means
C    41:  * get all the images in dir
C    42:  */
C    43:
C    44: ALL = lit_("ALL ");
C    45: swbyt4_(&ALL, &one);
C    46:
C    47: /*
C    48:  * read contents of the directory
C    49:  */
C    50:
C    51: dirlist = opendir(dir);
C    52:
C    53: if (dirlist == NULL)
C    54: {
C    55:       (void)strcpy(err, ERR_MSG_DIR);
C    56:       return ERR_STAT_DIR;
C    57: }
C    58:
C    59: while ((dirfile = readdir(dirlist)) != NULL)
C    60: {
C    61:       /*
C    62:        * ignore special cases of filenames "." and ".."
C    63:        */
C    64:
C    65:       strmatch = strcmp(dirfile->d_name, ".");
```

```
C   66:        if (strmatch == 0) continue;
C   67:
C   68:        strmatch = strcmp(dirfile->d_name, "..");
C   69:        if (strmatch == 0) continue;
C   70:
C   71:        /*
C   72:         * make a fully specified name
C   73:         */
C   74:
C   75:        (void)strcpy(fullname, dir);
C   76:        (void)strcat(fullname, "/");
C   77:
C   78:        (void)strcat(fullname, dirfile->d_name);
C   79:        (void)strcat(fullname, "\0");
C   80:
C   81:        /*
C   82:         * determine the validity of the format
C   83:         */
C   84:
C   85:        valid_type = IsMugFormat(fullname);
C   86:
C   87:        if (valid_type == FAILURE)
C   88:              continue;
C   89:
C   90:        /*
C   91:         * increment the position
C   92:         */
C   93:
C   94:        curpos++;
C   95:
C   96:        /*
C   97:         * push the file onto the stack
C   98:         */
C   99:
C  100:        istat = PushFileByName(fullname, curpos, &rawlist);
C  101:
C  102:        if (istat == FAILURE)
C  103:        {
C  104:              (void)strcpy(err, ERR_MSG_MALLOC);
C  105:              return ERR_STAT_MALLOC;
C  106:        }
C  107:
C  108: #ifdef STDOUT_DEBUG
C  109:       cur = rawlist;
C  110:       (void)printf("--------------- \n");
C  111:       while (cur != NULL)
C  112:       {
C  113:             (void)printf("pos: %d file: %s \n", cur->pos, cur->name);
C  114:             cur = cur->next;
C  115:       }
C  116: #endif
C  117:
C  118: }
C  119:
C  120: (void)closedir (dirlist);
C  121:
C  122: /*
C  123:  * renumber the positions in the list
C  124:  * to reflect the alphabetical sorting
C  125:  */
C  126:
C  127: curpos = 0;
C  128: cur = rawlist;
C  129: while (cur != NULL)
C  130: {
C  131:
C  132:       curpos++;
C  133:
C  134:       /*
C  135:        * test the file against the CRITERIA
C  136:        */
C  137:
```

```
C  138:        istat = TestMugImage(cur->name, request, aradir, err);
C  139:
C  140:        if (istat == FAILURE)
C  141:        {
C  142:              cur = cur->next;
C  143:              continue;
C  144:        }
C  145:
C  146:        /*
C  147:         * push this file onto the stack
C  148:         */
C  149:
C  150:        istat = PushFileByName(cur->name, curpos, &head);
C  151:
C  152:        if (istat == FAILURE)
C  153:        {
C  154:              (void)strcpy(err, ERR_MSG_MALLOC);
C  155:              return ERR_STAT_MALLOC;
C  156:        }
C  157:
C  158:        cur = cur->next;
C  159: }
C  160:
C  161: #ifdef STDOUT_DEBUG
C  162:       cur = head;
C  163:       (void)printf("----SORTED----- \n");
C  164:       while (cur != NULL)
C  165:       {
C  166:             (void)printf("pos: %d file: %s \n", cur->pos, cur->name);
C  167:             cur = cur->next;
C  168:       }
C  169:       (void)printf("------END------ \n");
C  170: #endif
C  171:
C  172: free(cur);
C  173: free(rawlist);
C  174:
C  175: /*
C  176:  * begin pulling the need files off and putting them
C  177:  * in a new list that will be returned
C  178:  */
C  179:
C  180: /*
C  181:  * ALL files requested
C  182:  */
C  183:
C  184: if (bpos == ALL)
C  185: {
C  186:       *satisfy = head;
C  187: }
C  188:
C  189: /*
C  190:  * absolute file positions requested
C  191:  */
C  192:
C  193: else if (bpos > 0)
C  194: {
C  195:
C  196:       while (head != NULL)
C  197:       {
C  198:             /*
C  199:                * get name and pos off of stack
C  200:               */
C  201:
C  202:             istat = PopFile(name, &pos, &head);
C  203:
C  204:             /*
C  205:              * only use files that fit the
C  206:              * requested positions
C  207:              */
C  208:
C  209:             if (pos < bpos || pos > epos) continue;
```

```
C  210:
C  211:                /*
C  212:                 * put a file on the list to be returned
C  213:                 */
C  214:
C  215:                istat = PushFileByName(name, pos, &test);
C  216:
C  217:                if (istat == FAILURE)
C  218:                {
C  219:                      (void)strcpy(err, ERR_MSG_MALLOC);
C  220:                      return ERR_STAT_MALLOC;
C  221:                }
C  222:          }
C  223:
C  224:          *satisfy = test;
C  225: }
C  226:
C  227: /*
C  228:  * time relative file positions requested
C  229:  */
C  230:
C  231: else if (bpos <= 0)
C  232: {
C  233:          cur = head;
C  234:          while (cur != NULL)
C  235:          {
C  236:                /*
C  237:                   * push the file onto stack ordered by
C  238:                 * time
C  239:                  */
C  240:
C  241:               istat = PushMugFileByTime(cur->name, cur->pos, &test);
C  242:
C  243:                cur = cur->next;
C  244:
C  245:          }
C  246:
C  247:          curpos = 1;
C  248:          while (test != NULL)
C  249:          {
C  250:                /*
C  251:                   * decrement time dependent position
C  252:                  */
C  253:
C  254:                curpos--;
C  255:
C  256:                /*
C  257:                   * get name and pos off of stack
C  258:                  */
C  259:
C  260:                istat = PopFile(name, &pos, &test);
C  261:
C  262:                /*
C  263:                 * only use files that fit the
C  264:                 * requested positions
C  265:                 */
C  266:
C  267:                if (curpos > bpos || curpos < epos) continue;
C  268:
C  269:                /*
C  270:                 * put a file on the list to be returned
C  271:                 */
C  272:
C  273:                istat = PushMugFileByTime(name, pos, &timlist);
C  274:
C  275:                if (istat == FAILURE)
C  276:                {
C  277:                      (void)strcpy(err, ERR_MSG_MALLOC);
C  278:                      return ERR_STAT_MALLOC;
C  279:                }
C  280:          }
C  281:
```

```
C  282:        *satisfy = timlist;
C  283: }
C  284:
C  285: return SUCCESS;
C  286:
C  287: }
C  288:
C  289: /*********************************************************/
C  290:
C  291: int
C  292: PushMugFileByTime(char *name, int pos, FILELIST **list)
C  293:
C  294: /*
C  295:  * add a name and pos to a linked list sorted
C  296:  * in a chronological sense on time with the
C  297:  * most recent image at the head of the list
C  298:  *
C  299:  * name - filename to add to list
C  300:  * pos  - position number of name
C  301:  * list - list to which to add item
C  302:  *
C  303:  * success 1
C  304:  * failure 0
C  305:  *
C  306:  */
C  307:
C  308: {
C  309:
C  310: char      err[MAX_ERR_LEN];   /* error message              */
C  311:
C  312: int      aradir[IMG_DIR_LEN]; /* mcidas area directory      */
C  313: int      istat;               /* funtion return status      */
C  314: int      time;                /* image time                 */
C  315:
C  316: FILELIST *after;
C  317: FILELIST *insert;
C  318:
C  319: FILELIST *new=NULL;
C  320:
C  321: /*
C  322:  * malloc a new node
C  323:  */
C  324:
C  325: new = (FILELIST *)malloc(sizeof(FILELIST));
C  326:
C  327: if (new == NULL)
C  328: {
C  329:      return FAILURE;
C  330: }
C  331:
C  332: /*
C  333:  * fill the new struct with values
C  334:  */
C  335:
C  336: (void)strcpy(new->name, name);
C  337: new->pos = pos;
C  338:
C  339: /*
C  340:  * compute an image time in seconds since 1/1/72
C  341:  */
C  342:
C  343: istat = ReadMugDir(name, aradir, err);
C  344: time = sksecs_(&aradir[3], &aradir[4]);
C  345:
C  346: new->time = time;
C  347:
C  348: /*
C  349:  * this is the first element in the list
C  350:  */
C  351:
C  352: if ((*list) == NULL)
C  353: {
```

```
C  354:        new->next = NULL;
C  355:        *list = new;
C  356:        return SUCCESS;
C  357: }
C  358:
C  359: /*
C  360:  * insert the new element at the head
C  361:  * if the time since 1/1/72 is greater than the
C  362:  * current time, meaning this image is newer,
C  363:  * we insert here
C  364:  */
C  365:
C  366: if ((*list)->time < time)
C  367: {
C  368:        new->next = *list;
C  369:        *list = new;
C  370:        return SUCCESS;
C  371: }
C  372:
C  373: /*
C  374:  * insert the new element at
C  375:  * the appropriate place in the list
C  376:  */
C  377:
C  378: insert = *list;
C  379:
C  380: while (1)
C  381: {
C  382:        after = insert->next;
C  383:
C  384:        /*
C  385:         * end of list?
C  386:         */
C  387:
C  388:        if (after == NULL)
C  389:              break;
C  390:
C  391:        /*
C  392:         * if the time since 1/1/72 is greater than the
C  393:         * current time, meaning this image is newer,
C  394:         * we insert here
C  395:         */
C  396:
C  397:        if (after->time < time)
C  398:              break;
C  399:
C  400:        insert = after;
C  401: }
C  402:
C  403:
C  404: insert->next = new;
C  405: new->next = after;
C  406:
C  407: return SUCCESS;
C  408:
C  409: }
C  410:
C  411: /
*************************************************************************/
C  412:
C  413: int
C  414: IsMugFormat(char *name)
C  415: /*
C  416:  * determine if a file is the MUG training course format
C  417:  *
C  418:  * name - filename to test
C  419:  *
C  420:  * success 1
C  421:  * failure 0
C  422:  *
C  423:  */
C  424:
```

```
C  425: {
C  426:
C  427: int      valid;            /* flag for a valid file format      */
C  428:
C  429:       /*
C  430:        * the format test is based only
C  431:        * on the name of the file. this makes
C  432:        * the name of the file extrememly important
C  433:        */
C  434:
C  435:       valid = strcmp(name, BASE_FILENAME);
C  436:
C  437:       if (valid == 0)
C  438:       {
C  439:            return FAILURE;
C  440:       }
C  441:       else
C  442:       {
C  443:            return SUCCESS;
C  444:       }
C  445:
C  446: }
C  447:
C  448: /
*************************************************************************/
C  449:
C  450: int
C  451: ReadMugDir(char *name, int *aradir, char *err)
C  452: /*
C  453:  * read a MUG training course file and create a
C  454:  * corresponding mcidas area directory
C  455:  *
C  456:  * NOTE:      The indices for aradir in this function
C  457:  *            are zero-based.
C  458:  *
C  459:  * name   - filename to read
C  460:  * aradir - mcidas area directory for "name"
C  461:  *
C  462:  * success 1
C  463:  * failure 0
C  464:  *
C  465:  */
C  466:
C  467: {
C  468:
C  469: FILE      *fd;                  /* file descriptor for image    */
C  470:
C  471: const char      *argdum;        /* dummy for arg fetchers       */
C  472:
C  473: char      line[MAX_LINE_LEN];   /* max byte length of data line */
C  474:
C  475: int      argh=0;                /* handle for arg fetchers      */
C  476: int      callen=512;            /* byte length of cal block     */
C  477: int      i;                     /* loop variable                */
C  478: int      imgtime;               /* nominal image time, hhmmss   */
C  479: int      istat;                 /* function return status       */
C  480: int      julday;                /* julian date of image, yyddd  */
C  481: int      navlen=512;            /* byte length of nav block     */
C  482: int      nelems;                /* number of elems in image     */
C  483: int      nlines;                /* number of lines in image     */
C  484: int      parsed_len;            /* byte length of arg block     */
C  485:
C  486: /*
C  487:  * initialize array
C  488:  */
C  489:
C  490: for(i=0; i<=IMG_DIR_LEN-1; i++)
C  491: {
C  492:       aradir[i] = 0;
C  493: }
C  494:
C  495: /*
```

```
C  496:  * fixed value mcwords
C  497:  */
C  498:
C  499: aradir[0]  = 0;                          /* always 0            */
C  500: aradir[1]  = 4;                          /* always 4            */
C  501: aradir[2]  = 0;                          /* mcidas ss number    */
C  502: aradir[5]  = 1;                          /* upper left line     */
C  503: aradir[6]  = 1;                          /* upper left element  */
C  504: aradir[10] = 1;                          /* bytes per pixel     */
C  505: aradir[11] = 1;                          /* line resolution     */
C  506: aradir[12] = 1;                          /* element resolution  */
C  507: aradir[13] = 1;                          /* number of bands     */
C  508: aradir[18] = 1;                          /* bandmap             */
C  509:
C  510: /*
C  511:  * byte offsets
C  512:  */
C  513:
C  514: aradir[34] = IMG_DIR_LEN * 4;            /* byte offset to nav  */
C  515: aradir[62] = aradir[34] + navlen;        /* byte offset to cal  */
C  516: aradir[33] = aradir[62] + callen;        /* byte offset to dat  */
C  517:
C  518: /*
C  519:  * source and cal type
C  520:  */
C  521:
C  522: aradir[51] = lit_("VISR");               /* source type         */
C  523: aradir[52] = lit_("BRIT");               /* cal type            */
C  524:
C  525: /*
C  526:  * open the file and get a file descriptor
C  527:  */
C  528:
C  529: fd = fopen(name, "r");
C  530:
C  531: /*
C  532:  * did we get a valid file desriptor?
C  533:  */
C  534:
C  535: if (fd == FAILURE)
C  536: {
C  537:       return FAILURE;
C  538: }
C  539:
C  540: /*
C  541:  * get first line of file and make it
C  542:  * available to the arg fetching routines
C  543:  */
C  544:
C  545: (void)fgets(line, sizeof(line), fd);
C  546:
C  547: argh  = Mcargparse(line, NULL, &parsed_len);
C  548:
C  549: /*
C  550:  * number of lines (mcword 9)
C  551:  */
C  552:
C  553: istat = Mcargint(argh, "NR", 1, 0, 999, -999, &nlines, &argdum);
C  554:
C  555: aradir[8] = nlines;
C  556:
C  557: /*
C  558:  * number of elements (mcword 10)
C  559:  */
C  560:
C  561: istat = Mcargint(argh, "NC", 1, 0, 999, -999, &nelems, &argdum);
C  562:
C  563: aradir[9] = nelems;
C  564:
C  565: /*
C  566:  * get first line of file and make it
C  567:  * available to the arg fetching routines
```

```
C  568:  */
C  569:
C  570: (void)fgets(line, sizeof(line), fd);
C  571:
C  572: istat = Mcargfree(argh);
C  573:
C  574: argh  = Mcargparse(line, NULL, &parsed_len);
C  575:
C  576: /*
C  577:  * julian date, yyddd (mcword 4)
C  578:  */
C  579:
C  580: istat = Mcargint(argh, "VD.AY", 1, 0, 999, -999, &julday, &argdum);
C  581:
C  582: aradir[3] = julday;
C  583:
C  584: /*
C  585:  * image time, hhmmss (mcword 5)
C  586:  */
C  587:
C  588: istat = Mcargint(argh, "VT.IME", 1, 0, 999, -999, &imgtime, &argdum);
C  589:
C  590: aradir[4] = imgtime;
C  591:
C  592: (void)close((int)fd);
C  593:
C  594: istat = Mcargfree(argh);
C  595:
C  596: return SUCCESS;
C  597:
C  598: }
C  599:
C  600: /
**********************************************************************/
C  601:
C  602: int
C  603: MugNavImgToEarth(       char *name, float line, float elem,
C  604:                         float *lat, float *lon, char *err)
C  605: /*
C  606:  * convert MUG image line/elem into lat/lon
C  607:  *
C  608:  * name - filename to read
C  609:  * line - input image line
C  610:  * elem - input image element
C  611:  * lat  - output latitude of line/elem
C  612:  * lon  - output longitude of line/elem
C  613:  * err  - error string returned
C  614:  *
C  615:  * success 1
C  616:  * failure 0
C  617:  *
C  618:  */
C  619:
C  620: {
C  621:
C  622: static char    lastname[MAX_NAME_LEN];  /* filename last time in func.*/
C  623:
C  624: float      dum;                        /* dummy variable          */
C  625:
C  626: int      istat;                        /* function status         */
C  627: int      len;                          /* byte len of navcod      */
C  628: int      llflag;                       /* variable for nv1ini     */
C  629: int      navcod[MAX_NAV_LEN];          /* mcidas nav codicil      */
C  630: int      one=1;                        /* the number one (1)      */
C  631: int      two=2;                        /* the number two (2)      */
C  632:
C  633: if (strcmp(lastname, name) != 0)
C  634: {
C  635:       /*
C  636:        * get a mcidas nav codicil
C  637:        */
C  638:
```

```
C  639:        istat = ReadMugNavCod(name, navcod, 1, 1, &len);
C  640:
C  641:        if (istat == FAILURE)
C  642:        {
C  643:              (void)strcpy(err, ERR_MSG_NONAV);
C  644:              return FAILURE;
C  645:        }
C  646:
C  647:        /*
C  648:         * initialize mcidas nav transforms
C  649:         */
C  650:
C  651:        istat = nvprep_(&one, navcod);
C  652:
C  653:        if (istat != 0)
C  654:        {
C  655:              (void)strcpy(err, ERR_MSG_MCIDAS_NAV);
C  656:              return FAILURE;
C  657:        }
C  658:
C  659:        llflag = lit_("LL  ");
C  660:        istat = nv1ini_(&two, &llflag);
C  661:
C  662:        if (istat != 0)
C  663:        {
C  664:              (void)strcpy(err, ERR_MSG_MCIDAS_NAV);
C  665:              return FAILURE;
C  666:        }
C  667:
C  668:        (void)strcpy(lastname, name);
C  669: }
C  670:
C  671: istat = nv1sae_(&line, &elem, &dum, lat, lon, &dum);
C  672:
C  673: if (istat != 0)
C  674: {
C  675:        (void)strcpy(err, ERR_MSG_IMG_TO_EARTH);
C  676:        return FAILURE;
C  677: }
C  678:
C  679: /*
C  680:  * convert longitude to the mcidas west positive convention
C  681:  */
C  682:
C  683: #ifndef WEST_POSITIVE
C  684:        (*lon) = (*lon) *= -1;
C  685: #endif
C  686:
C  687: return SUCCESS;
C  688:
C  689: }
C  690:
C  691: /
**************************************************************************/
C  692:
C  693: int
C  694: MugNavEarthToImg(      char *name, float lat, float lon,
C  695:                    float *line, float *elem, char *err)
C  696: /*
C  697:  * convert MUG image lat/lon into line/elem
C  698:  *
C  699:  * name - filename to read
C  700:  * lat  - input latitude
C  701:  * lon  - input longitude
C  702:  * line - output image line of lat/lon
C  703:  * elem - output image element of lat/lon
C  704:  * err  - error string returned
C  705:  *
C  706:  * success 1
C  707:  * failure 0
C  708:  *
C  709:  */
```

```
C  710:
C  711: {
C  712:
C  713: static char  lastname[MAX_NAME_LEN];   /* filename last time in func.*/
C  714:
C  715: float      dum;                        /* dummy variable          */
C  716:
C  717: int       istat;                      /* function status         */
C  718: int       len;                        /* byte len of navcod      */
C  719: int       llflag;                     /* variable for nv1ini     */
C  720: int       navcod[MAX_NAV_LEN];        /* mcidas nav codicil      */
C  721: int       one=1;                      /* the number one (1)      */
C  722: int       two=2;                      /* the number two (2)      */
C  723:
C  724: if (strcmp(lastname, name) != 0)
C  725: {
C  726:        /*
C  727:         * get a mcidas nav codicil
C  728:         */
C  729:
C  730:        istat = ReadMugNavCod(name, navcod, 1, 1, &len);
C  731:
C  732:        if (istat == FAILURE)
C  733:        {
C  734:              (void)strcpy(err, ERR_MSG_NONAV);
C  735:              return FAILURE;
C  736:        }
C  737:
C  738:        /*
C  739:         * initialize mcidas nav transforms
C  740:         */
C  741:
C  742:        istat = nvprep_(&one, navcod);
C  743:
C  744:        if (istat != 0)
C  745:        {
C  746:              (void)strcpy(err, ERR_MSG_MCIDAS_NAV);
C  747:              return FAILURE;
C  748:        }
C  749:
C  750:        llflag = lit_("LL  ");
C  751:        istat = nv1ini_(&two, &llflag);
C  752:
C  753:        if (istat != 0)
C  754:        {
C  755:              (void)strcpy(err, ERR_MSG_MCIDAS_NAV);
C  756:              return FAILURE;
C  757:        }
C  758:
C  759:        (void)strcpy(lastname, name);
C  760: }
C  761:
C  762: /*
C  763:  * convert longitude to the mcidas west positive convention
C  764:  */
C  765:
C  766: #ifndef WEST_POSITIVE
C  767:       lon = lon *= -1;
C  768: #endif
C  769:
C  770: istat = nv1eas_(&lat, &lon, &dum, line, elem, &dum);
C  771:
C  772: if (istat != 0)
C  773: {
C  774:       (void)strcpy(err, ERR_MSG_EARTH_TO_IMG);
C  775:       return FAILURE;
C  776: }
C  777:
C  778: return SUCCESS;
C  779:
C  780: }
C  781:
```

```
C  782: /
**********************************************************************/
C  783:
C  784: int
C  785: ReadMugRes(char *name, float line, float elem, float *resx, float *resy,char *err)
C  786:
C  787: /*
C  788:  * calculate MUG image resolution at center point
C  789:  *
C  790:  * name - filename to read
C  791:  * line - input image line
C  792:  * elem - input image element
C  793:  * resx - output x-resolution at center of image (km)
C  794:  * resy - output y-resolution at center of image (km)
C  795:  * err  - error string returned
C  796:  *
C  797:  * success 1
C  798:  * failure 0
C  799:  *
C  800:  */
C  801:
C  802: {
C  803:
C  804: static char lastname[MAX_NAME_LEN]; /* filename last time in func.    */
C  805:
C  806: int      istat;                    /* function status                */
C  807:
C  808: double      azimuth;               /* directional azimuth (unused)*/
C  809: double      range1;                /* range from center down         */
C  810: double      range2;                /* range from down 1 line to
C  811:                                       over 1 element                 */
C  812:
C  813: float      dum;                   /* dummy variable                 */
C  814: float      elem_p1;               /* element plus one               */
C  815: float      lat;                   /* lat to measure distance        */
C  816: float      lat2;                  /* lat to measure distance        */
C  817: float      line_p1;               /* line plus one                  */
C  818: float      lon;                   /* lon to measure distance        */
C  819: float      lon2;                  /* lon to measure distance        */
C  820:
C  821: int      len;                     /* byte len of navcod             */
C  822: int      llflag;                  /* variable for nv1ini            */
C  823: int      navcod[MAX_NAV_LEN];     /* mcidas nav codicil             */
C  824: int      one=1;                   /* the number one (1)             */
C  825: int      two=2;                   /* the number two (2)             */
C  826:
C  827: if (strcmp(lastname, name) != 0)
C  828: {
C  829:       /*
C  830:        * get a mcidas nav codicil
C  831:        */
C  832:
C  833:       istat = ReadMugNavCod(name, navcod, 1, 1, &len);
C  834:
C  835:       if (istat == FAILURE)
C  836:       {
C  837:             (void)strcpy(err, ERR_MSG_NONAV);
C  838:             return FAILURE;
C  839:       }
C  840:
C  841:       /*
C  842:        * initialize mcidas nav transforms
C  843:        */
C  844:
C  845:       istat = nvprep_(&one, navcod);
C  846:
C  847:       if (istat != 0)
C  848:       {
C  849:             (void)strcpy(err, ERR_MSG_MCIDAS_NAV);
C  850:             return FAILURE;
C  851:       }
C  852:
```

```
C  853:        llflag = lit_("LL  ");
C  854:        istat = nv1ini_(&two, &llflag);
C  855:
C  856:        if (istat != 0)
C  857:        {
C  858:              (void)strcpy(err, ERR_MSG_MCIDAS_NAV);
C  859:              return FAILURE;
C  860:        }
C  861:
C  862:        (void)strcpy(lastname, name);
C  863: }
C  864:
C  865: /*
C  866:  * convert longitude to the mcidas west positive convention
C  867:  */
C  868:
C  869: #ifndef WEST_POSITIVE
C  870:        lon = lon *= -1;
C  871: #endif
C  872:
C  873: istat = nv1sae_(&line, &elem, &dum, &lat, &lon, &dum);
C  874:
C  875: if (istat != 0)
C  876: {
C  877:        (void)strcpy(err, ERR_MSG_IMG_TO_EARTH);
C  878:        return FAILURE;
C  879: }
C  880:
C  881: /*
C  882:  * move down one line and find the lat/lon
C  883:  */
C  884:
C  885: line_p1 = line + 1.0;
C  886:
C  887: istat = nv1sae_(&line_p1, &elem, &dum, &lat2, &lon2, &dum);
C  888:
C  889: if (istat != 0)
C  890: {
C  891:        (void)strcpy(err, ERR_MSG_IMG_TO_EARTH);
C  892:        return FAILURE;
C  893: }
C  894:
C  895: /*
C  896:  * find range from center to down one line
C  897:  */
C  898:
C  899: istat = lltora_(&lat, &lon, &lat2, &lon2, &range1, &azimuth);
C  900:
C  901: if (istat != 0)
C  902: {
C  903:        (void)strcpy(err, ERR_MSG_MCIDAS_NAV);
C  904:        return FAILURE;
C  905: }
C  906:
C  907: /*
C  908:  * move over one element and find the lat/lon
C  909:  */
C  910:
C  911: lat = lat2;
C  912: lon = lon2;
C  913:
C  914: elem_p1 = elem + 1.0;
C  915:
C  916: istat = nv1sae_(&line_p1, &elem_p1, &dum, &lat2, &lon2, &dum);
C  917:
C  918: if (istat != 0)
C  919: {
C  920:        (void)strcpy(err, ERR_MSG_IMG_TO_EARTH);
C  921:        return FAILURE;
C  922: }
C  923:
C  924: /*
```

```
C  925:  * find range from center to down one line
C  926:  */
C  927:
C  928: istat = lltora_(&lat, &lon, &lat2, &lon2, &range2, &azimuth);
C  929:
C  930: if (istat != 0)
C  931: {
C  932:       (void)strcpy(err, ERR_MSG_MCIDAS_NAV);
C  933:       return FAILURE;
C  934: }
C  935:
C  936: /*
C  937:  * res is average of the 2 ranges (resx = resy)
C  938:  */
C  939:
C  940: *resy = (float)(range1 + range2) / 2.0;
C  941: *resx = *resy;
C  942:
C  943: #ifdef DEBUG
C  944:       (void)sprintf(dbg, "res: %f", *resy);
C  945:       m0sxtrce_(dbg, strlen(dbg));
C  946: #endif
C  947:
C  948: return SUCCESS;
C  949:
C  950: }
C  951:
C  952: /*********************************************************/
C  953:
C  954: int
C  955: TestMugImage(char *name, CRITERIA *request, int *aradir, char *err)
C  956:
C  957: /*
C  958:  * test an image against the specification
C  959:  * criteria detailed in struct request
C  960:  *
C  961:  * NOTE: No tests will be performed for CRITERIA values of -1
C  962:  *
C  963:  * name    - filename to add to list
C  964:  * request - CRITERIA to test name
C  965:  * aradir  - mcidas area directory returned for name
C  966:  * err     - error string from ReadImgDir
C  967:  *
C  968:  * success 1
C  969:  * failure 0
C  970:  *
C  971:  */
C  972:
C  973: {
C  974:
C  975: int      imgday;                 /* day of image                */
C  976: int      imgss;                  /* satellite id for image      */
C  977: int      imgtim;                 /* time of image               */
C  978: int      istat;                  /* function status             */
C  979:
C  980: /*
C  981:  * get a mcidas area directory for this image
C  982:  */
C  983:
C  984: istat = ReadMugDir(name, aradir, err);
C  985:
C  986: if (istat == FAILURE)
C  987: {
C  988:       return FAILURE;
C  989: }
C  990:
C  991: /*
C  992:  * test beginning and ending time
C  993:  */
C  994:
C  995:
C  996: if (request->begtim >= 0)
```

```
C  997: {
C  998:
C  999:        imgtim = aradir[4];
C 1000:
C 1001:        if (imgtim < request->begtim || imgtim > request->endtim)
C 1002:        {
C 1003:              (void)strcpy(err, "Image time does not meet search criteria \n");
C 1004:              return FAILURE;
C 1005:        }
C 1006:
C 1007: }
C 1008:
C 1009: /*
C 1010:  * test beginning and ending day
C 1011:  */
C 1012:
C 1013: if (request->begday >= 0)
C 1014: {
C 1015:
C 1016:        imgday = aradir[3];
C 1017:
C 1018:        if (imgday < request->begday || imgday > request->endday)
C 1019:        {
C 1020:            (void)strcpy(err, "Image day does not meet search criteria \n");
C 1021:              return FAILURE;
C 1022:        }
C 1023:
C 1024: }
C 1025:
C 1026: /*
C 1027:  * test beginning and ending mcidas ss number
C 1028:  */
C 1029:
C 1030: if (request->begss >= 0)
C 1031: {
C 1032:
C 1033:        imgss = aradir[2];
C 1034:
C 1035:        if (imgss < request->begss || imgss > request->endss)
C 1036:        {
C 1037:              (void)strcpy(err, "Image satellite id does not meet search criteria \n");
C 1038:              return FAILURE;
C 1039:        }
C 1040: }
C 1041:
C 1042: return SUCCESS;
C 1043:
C 1044: }
C 1045:
C 1046: /
**********************************************************************/
C 1047:
C 1048: int
C 1049: ReadMugLine(char *name, READPARM *read, int band, short *buf, char *err)
C 1050: /*
C 1051:  * read a line of a MUG Training Course image
C 1052:  *
C 1053:  * name    - filename to read
C 1054:  * read    - READPARM struct containing read specs
C 1055:  * band    - band number of elements to read
C 1056:  * buf     - buffer containing image data
C 1057:  * err     - error string to return
C 1058:  *
C 1059:  * success 1
C 1060:  * failure 0
C 1061:  *
C 1062:  */
C 1063:
C 1064: {
C 1065:
C 1066: const char      *argdum;          /* dummy for arg fetchers    */
C 1067:
```

```
C 1068: char      line[MAX_LINE_LEN];       /* max byte length of data line    */
C 1069:
C 1070: static char   lastname[MAX_NAME_LEN];/* filename last time in func.     */
C 1071:
C 1072: static int      lastline=0;         /* last line of file read          */
C 1073:
C 1074: static FILE      *fd;              /* file descriptor                 */
C 1075:
C 1076: int      argh;                     /* handle for arg fetchers         */
C 1077: int      begele;                   /* fisrt element to read           */
C 1078: int      beglin;                   /* fisrt line to read              */
C 1079: int      bufsiz;                   /* max # of lines to buffer        */
C 1080: int      des_len;                  /* byte size of dest pixel         */
C 1081: int      i;                        /* loop variable                   */
C 1082: int      index;                    /* index into data array           */
C 1083: int      istat;                    /* function return status          */
C 1084: int      maxele;                   /* max number of elem in image     */
C 1085: int      maxlin;                   /* max number of lines in image    */
C 1086: int      numband;                  /* number of bands in image        */
C 1087: int      numele;                   /* number of elements to read      */
C 1088: int      numlin;                   /* number of lines to read         */
C 1089: int      parsed_len;               /* byte length of arg block        */
C 1090: int      src_len;                  /* byte size of source pixel       */
C 1091:
C 1092:
C 1093: double      *val;                  /* data values on a line           */
C 1094:
C 1095: if (strcmp(lastname, name) != 0)
C 1096: {
C 1097:       /*
C 1098:        * get a file descriptor
C 1099:        */
C 1100:
C 1101:       fd = fopen(name, "r");
C 1102:
C 1103:       /*
C 1104:        * did we get a valid file desriptor?
C 1105:        */
C 1106:
C 1107:       if (fd == NULL)
C 1108:       {
C 1109:             return FAILURE;
C 1110:       }
C 1111:
C 1112:       /*
C 1113:        * malloc space for the intermediate read
C 1114:        */
C 1115:
C 1116:       val = (double *)malloc(read->numele * sizeof(double));
C 1117:
C 1118:       if (val == NULL)
C 1119:       {
C 1120:             (void)strcpy(err, ERR_MSG_MALLOC);
C 1121:             return ERR_STAT_MALLOC;
C 1122:       }
C 1123:
C 1124:       /*
C 1125:        * read the first 3 lines of header
C 1126:        */
C 1127:
C 1128:       (void)fgets(line, sizeof(line), fd);
C 1129:       (void)fgets(line, sizeof(line), fd);
C 1130:       (void)fgets(line, sizeof(line), fd);
C 1131:
C 1132:       (void)strcpy(lastname, name);
C 1133: }
C 1134:
C 1135: /*
C 1136:  * extract needed parameters
C 1137:  * from the struct (this makes
C 1138:  * the function easier to read)
C 1139:  */
```

```
C 1140:
C 1141: begele  = read->begele;
C 1142: beglin  = read->beglin;
C 1143: bufsiz  = read->bufsiz;
C 1144: des_len = read->des_len;
C 1145: maxele  = read->maxele;
C 1146: maxlin  = read->maxlin;
C 1147: numband = read->numband;
C 1148: numele  = read->numele;
C 1149: numlin  = read->numlin;
C 1150: src_len = read->src_len;
C 1151:
C 1152: /*
C 1153:  * read any unwanted lines. this will
C 1154:  * only happen when a blowdown is requested
C 1155:  */
C 1156:
C 1157: for(i=1; i<beglin-lastline; i++)
C 1158: {
C 1159:       (void)fgets(line, sizeof(line), fd);
C 1160: }
C 1161:
C 1162: lastline = beglin;
C 1163:
C 1164: /*
C 1165:  * get next line of file and make it
C 1166:  * available to the arg fetching routines
C 1167:  */
C 1168:
C 1169: (void)fgets(line, sizeof(line), fd);
C 1170:
C 1171: argh = Mcargparse(line, NULL, &parsed_len);
C 1172:
C 1173: #ifdef DEBUG
C 1174:       (void)sprintf(dbg, "begele=%d numele=%d, argh=%d", begele, numele, argh);
C 1175:       m0sxtrce_(dbg, strlen(dbg));
C 1176: #endif
C 1177:
C 1178: index = 0;
C 1179: for (i=begele; i<=begele+numele-1; i++)
C 1180: {
C 1181:       istat = Mcargdbl(argh, "", i, (double)0,
C 1182:                  (double)999, (double)-999, &val[index], &argdum);
C 1183:
C 1184:       index++;
C 1185: }
C 1186:
C 1187: /*
C 1188:  * "calibrate" the line
C 1189:  */
C 1190:
C 1191: for (i=0; i<=index-1; i++)
C 1192: {
C 1193:       buf[i] = 255 - (val[i] - 75);
C 1194:
C 1195: #ifdef DEBUG
C 1196:       (void)sprintf(dbg, "buf[%d] = %d", i, buf[i]);
C 1197:       m0sxtrce_(dbg, strlen(dbg));
C 1198: #endif
C 1199: }
C 1200:
C 1201: (void)mpixel_(&index, &src_len, &des_len, (void *)buf);
C 1202:
C 1203: istat = Mcargfree(argh);
C 1204:
C 1205: return SUCCESS;
C 1206:
C 1207: }
C 1208:
C 1209: /
*************************************************************************/
C 1210:
```

```
C 1211: int
C 1212: ReadMugCalCod(char *name, int *cod, int *len)
C 1213:
C 1214: /*
C 1215:  * read a mcidas calibration codicil
C 1216:  *
C 1217:  * name - filename of image
C 1218:  * cod  - output mcidas cal codicil
C 1219:  * len  - byte length of output mcidas cal codicil
C 1220:  *
C 1221:  * success 1
C 1222:  * failure 0
C 1223:  *
C 1224:  */
C 1225:
C 1226: {
C 1227:
C 1228: int      rc;                 /* return code                          */
C 1229:
C 1230: int       i;
C 1231:
C 1232: /*
C 1233:  * get cal codicil
C 1234:  */
C 1235:
C 1236: *len=MAX_CAL_LEN;
C 1237: for(i=0; i<=(*len)-1; i++)
C 1238:       cod[i] = 0;
C 1239:
C 1240: rc = SUCCESS;
C 1241:
C 1242: return (rc);
C 1243:
C 1244: }
C 1245:
C 1246: /
**************************************************************************/
C 1247:
C 1248: int
C 1249: ReadMugNavCod(char *name, int *cod, int pt_line, int pt_elem, int *len)
C 1250:
C 1251: /*
C 1252:  * read a mcidas navigation codicil
C 1253:  *
C 1254:  * name    - filename of image
C 1255:  * cod     - output mcidas nav codicil
C 1256:  * pt_line - line number of upper left latitude
C 1257:  * pt_elem - elem number of upper left longitude
C 1258:  * len     - byte length of output mcidas nav codicil
C 1259:  *
C 1260:  * success 1
C 1261:  * failure 0
C 1262:  *
C 1263:  */
C 1264:
C 1265: {
C 1266:
C 1267: const char      *argdum;        /* dummy for arg fetchers          */
C 1268:
C 1269: char      line[MAX_LINE_LEN];   /* max byte length of data line    */
C 1270:
C 1271: static FILE      *fd;          /* file descriptor                 */
C 1272:
C 1273: int       argh=0;             /* handle for arg fetchers         */
C 1274: int       i;                  /* loop variable                   */
C 1275: int       istat;              /* function return status          */
C 1276: int       parsed_len;         /* byte length of arg block        */
C 1277: int       rc;                 /* return code                     */
C 1278:
C 1279: double      latinc;           /* degree spacing of latitude      */
C 1280: double      loninc;           /* degree spacing of longitude     */
C 1281: double      ullat;            /* upper left latitude             */
```

```
C 1282: double      ullon;              /* upper left longitude         */
C 1283:
C 1284: /*
C 1285:  * open the file and get a file descriptor
C 1286:  */
C 1287:
C 1288: fd = fopen(name, "r");
C 1289:
C 1290: /*
C 1291:  * did we get a valid file desriptor?
C 1292:  */
C 1293:
C 1294: if (fd == FAILURE)
C 1295: {
C 1296:       return FAILURE;
C 1297: }
C 1298:
C 1299: /*
C 1300:  * get first line of file and make it
C 1301:  * available to the arg fetching routines
C 1302:  */
C 1303:
C 1304: (void)fgets(line, sizeof(line), fd);
C 1305:
C 1306: argh  = Mcargparse(line, NULL, &parsed_len);
C 1307:
C 1308: /*
C 1309:  * number of lines (mcword 9)
C 1310:  */
C 1311:
C 1312: istat = Mcargdbl(argh, "LAT", 2, (double)0, (double)999, (double)-999,
C 1313:                                        &ullat, &argdum);
C 1314:
C 1315: istat = Mcargdbl(argh, "LON", 2, (double)0, (double)999, (double)-999,
C 1316:                                        &ullon, &argdum);
C 1317:
C 1318: istat = Mcargdbl(argh, "LAI.NC", 1, (double)0, (double)999, (double)-999,
C 1319:                                        &latinc, &argdum);
C 1320:
C 1321: istat = Mcargdbl(argh, "LOI.NC", 1, (double)0, (double)999, (double)-999,
C 1322:                                        &loninc, &argdum);
C 1323:
C 1324: /*
C 1325:  * build a rectilinear nav codicil
C 1326:  */
C 1327:
C 1328: *len=MAX_NAV_LEN;
C 1329:
C 1330: /*
C 1331:  * initialize to zero
C 1332:  */
C 1333:
C 1334: for(i=0; i<=(*len)-1; i++)
C 1335:       cod[i] = 0;
C 1336:
C 1337: cod[0]  = lit_("RECT");
C 1338: cod[1]  = pt_line;
C 1339: cod[2]  = (int)(ullat  * 10000.0);
C 1340: cod[3]  = pt_elem;
C 1341: cod[4]  = (int)(ullon  * 10000.0);
C 1342: cod[5]  = (int)(latinc * 10000.0);
C 1343: cod[6]  = (int)(loninc * 10000.0);
C 1344: cod[7]  = EARTH_RAD_METERS;
C 1345: cod[8]  = EARTH_ECCENTRICITY;
C 1346: cod[9]  = 0;
C 1347:
C 1348: #ifdef WEST_POSITIVE
C 1349:       cod[10] = 1;
C 1350: #else
C 1351:       cod[10] = -1;
C 1352: #endif
C 1353:
```

```
C 1354: #ifdef DEBUG
C 1355:       for(i=0; i<=10; i++)
C 1356:       {
C 1357:             (void)sprintf(dbg, "cod[%d] = %d", i+1, cod[i]);
C 1358:             m0sxtrce_(dbg, strlen(dbg));
C 1359:       }
C 1360: #endif
C 1361:
C 1362: rc = SUCCESS;
C 1363:
C 1364: istat = Mcargfree(argh);
C 1365:
C 1366: return (rc);
C 1367:
C 1368: }
C 1369:
C 1370: /*********************************************************/
C 1371:
C 1372: int
C 1373: ReadMugCards(char *name, int *aradir, int *cards, char *err)
C 1374:
C 1375: /*
C 1376:  * build the required ADDE comment cards
C 1377:  * containing the center point and res information
C 1378:  *
C 1379:  * name   - filename to inspect
C 1380:  * aradir - mcidas area directory for name
C 1381:  * cards  - mcidas comment cards returned as integers
C 1382:  * err    - error string from ReadImgDir
C 1383:  *
C 1384:  * success 1
C 1385:  * failure 0
C 1386:  *
C 1387:  */
C 1388:
C 1389: {
C 1390:
C 1391: char    one_card[MAX_CARD_LEN-1]; /* a single comment card      */
C 1392:
C 1393: float      elem;                  /* center element of image    */
C 1394: float      lat;                   /* center latitude of image   */
C 1395: float      line;                  /* center line of image       */
C 1396: float      lon;                   /* center longitude of image  */
C 1397: float      resx;                  /* elem res (km) at center    */
C 1398: float      resy;                  /* line res (km) at center    */
C 1399:
C 1400: int      add;                     /* function return for AddCard*/
C 1401: int      istat;                   /* function return status     */
C 1402: int      num_cards=0;             /* number of cards built      */
C 1403:
C 1404: /*
C 1405:  * get center line/element of image
C 1406:  */
C 1407:
C 1408: elem = aradir[9] / 2.0;
C 1409: line = aradir[8] / 2.0;
C 1410:
C 1411: /*
C 1412:  * get center lat/lon of image
C 1413:  */
C 1414:
C 1415: istat = MugNavImgToEarth(name, line, elem, &lat, &lon, err);
C 1416:
C 1417: if (istat != FAILURE)
C 1418: {
C 1419:       /*
C 1420:        * build cards for center point
C 1421:        */
C 1422:
C 1423:       (void)sprintf(one_card, CEN_LAT_CARD, lat);
C 1424:       num_cards++;
C 1425:
```

```
C 1426:         add = AddCard(one_card, MAX_CARD_LEN, cards, num_cards);
C 1427:
C 1428:         if (add == FAILURE)
C 1429:         {
C 1430:               (void)strcpy(err, ERR_MSG_MALLOC);
C 1431:               return FAILURE;
C 1432:         }
C 1433:
C 1434:         (void)sprintf(one_card, CEN_LON_CARD, lon);
C 1435:         num_cards++;
C 1436:
C 1437:         add = AddCard(one_card, MAX_CARD_LEN, cards, num_cards);
C 1438:
C 1439:         if (add == FAILURE)
C 1440:         {
C 1441:               (void)strcpy(err, ERR_MSG_MALLOC);
C 1442:               return FAILURE;
C 1443:         }
C 1444: }
C 1445: else
C 1446: {
C 1447:         /*
C 1448:          * center point failure
C 1449:          */
C 1450:
C 1451:         return FAILURE;
C 1452: }
C 1453:
C 1454: /*
C 1455:  * get the resolution at the center point of the image
C 1456:  */
C 1457:
C 1458: istat = ReadMugRes(name, line, elem, &resx, &resy, err);
C 1459:
C 1460: if (istat != FAILURE)
C 1461: {
C 1462:         /*
C 1463:          * build cards for the resolution
C 1464:          */
C 1465:
C 1466:         (void)sprintf(one_card, RES_X_CARD, (int)(resx+0.5));
C 1467:         num_cards++;
C 1468:
C 1469:         add = AddCard(one_card, MAX_CARD_LEN, cards, num_cards);
C 1470:
C 1471:         if (add == FAILURE)
C 1472:         {
C 1473:               (void)strcpy(err, ERR_MSG_MALLOC);
C 1474:               return FAILURE;
C 1475:         }
C 1476:
C 1477:         (void)sprintf(one_card, RES_Y_CARD, (int)(resy+0.5));
C 1478:         num_cards++;
C 1479:
C 1480:         add = AddCard(one_card, MAX_CARD_LEN, cards, num_cards);
C 1481:
C 1482:         if (add == FAILURE)
C 1483:         {
C 1484:               (void)strcpy(err, ERR_MSG_MALLOC);
C 1485:               return FAILURE;
C 1486:         }
C 1487:
C 1488: }
C 1489: else
C 1490: {
C 1491:         /*
C 1492:          * resolution failure
C 1493:          */
C 1494:
C 1495:         return FAILURE;
C 1496: }
C 1497:
```

```
C 1498:
C 1499: /*
C 1500:  * get the valid unit type for this image
C 1501:  */
C 1502:
C 1503: istat = ReadUnits(aradir, cards, &num_cards, err);
C 1504:
C 1505: /*
C 1506:  * change mcword 64 in the area
C 1507:  * directory to relect the number
C 1508:  * of comment cards
C 1509:  */
C 1510:
C 1511: aradir[63] = aradir[63] + num_cards;
C 1512:
C 1513: return SUCCESS;
C 1514:
C 1515: }
```

```
D    1: /*      mug.h
D    2:  *
D    3:  *      This is the main include file for the 1995 MUG
D    4:  *      Training Course ADDE server.
D    5:  *
D    6:  *      It contains data structures that are used by
D    7:  *      the image servers.
D    8:  */
D    9:
D   10: /*
D   11:  *      include files
D   12:  */
D   13:
D   14: #include      <unistd.h>
D   15: #include      "mcidas.h"          /* McIDAS include file    */
D   16: #include      "servacct.h"        /* ADDE include file      */
D   17:
D   18: /*
D   19:  *      return status
D   20:  */
D   21:
D   22: #define FAILURE      0
D   23: #define SUCCESS      1
D   24:
D   25: /*
D   26:  *      error handling
D   27:  */
D   28:
D   29: #define ERR_STAT_DIR            -2
D   30: #define ERR_MSG_DIR         "Unable to read contents of directory \n"
D   31:
D   32: #define ERR_STAT_MALLOC         -1
D   33: #define ERR_MSG_MALLOC      "Unable to allocate memory \n"
D   34:
D   35: #define ERR_STAT_BADLINE        -41
D   36: #define ERR_MSG_BADLINE     "Unable to read image line %d \n"
D   37:
D   38: #define ERR_STAT_BLANK_IMG      -47
D   39: #define ERR_MSG_BLANK_IMG   "There's nothing to see here. \n"
D   40:
D   41: #define ERR_STAT_NOCAL          -40
D   42: #define ERR_MSG_NOCAL       "Unable to initialize calibration \n"
D   43:
D   44: #define ERR_STAT_NOIMG          -51
D   45: #define ERR_MSG_NOIMG       "There are no images out here. \n"
D   46:
D   47: #define ERR_STAT_NONAV          -31
D   48: #define ERR_MSG_NONAV       "Unable to initialize navigation \n"
D   49:
D   50: #define ERR_STAT_SELECT         -50
D   51:
D   52: #define ERR_MSG_EARTH_TO_IMG "Unable to transform lat/lon to line/element \n"
D   53: #define ERR_MSG_FORMAT        "Unable to read this data format \n"
D   54: #define ERR_MSG_IMG_TO_EARTH "Unable to transform line/element to lat/lon \n"
D   55: #define ERR_MSG_MCIDAS_NAV   "Error in McIDAS navigation \n"
D   56: #define ERR_MSG_UNKNOWN_SAT  "Calibration units unknown for this satellite \n"
D   57:
D   58: /*
D   59:  *      max lengths
D   60:  */
D   61:
D   62: #define IMG_DIR_LEN     64    /* len of image directory           */
D   63: #define MAX_CAL_LEN     512   /* byte len of calibration codicil  */
D   64: #define MAX_CARD_LEN    80    /* len of a single comment card     */
D   65: #define MAX_DIR_LEN     1024  /* len of directory where data lives */
D   66: #define MAX_ERRMSG_LEN 72     /* byte len of err msg back to client*/
```

```
D   67: #define MAX_ERR_LEN     256   /* len of error string                */
D   68: #define MAX_LINE_LEN    10000 /* max. len of a line in the image file   */
D   69: #define MAX_NAME_LEN    1024  /* max. len of a image path and name  */
D   70: #define MAX_NAV_LEN     512   /* byte len of calibration codicil    */
D   71: #define MAX_NUM_BAND    32    /* max. number of bands in an image   */
D   72: #define MAX_NUM_CARDS   250   /* max. number of mcidas comment cards*/
D   73:
D   74: /*
D   75:  *      default values
D   76:  */
D   77:
D   78: #define DEF_NUM_ELEMS     640 /* default num of elements to send     */
D   79: #define DEF_NUM_LINES     480 /* default num of lines to send        */
D   80: #define READ_BUFFER_SIZE 1    /* # of image lines to buffer on read  */
D   81: #define RES_AT_CENTER     1   /* resolution in km at center of image */
D   82:
D   83: #define BASE_FILENAME  "MRF" /* MUG format files name convention   */
D   84:
D   85: /*
D   86:  *      nav parameters
D   87:  */
D   88:
D   89: #define EARTH_RAD_METERS   6378388  /* earth radius in meters     */
D   90: #define EARTH_ECCENTRICITY 81992.0  /* earth eccentricity * 1e6   */
D   91:
D   92: /*
D   93:  *      longitude convention
D   94:  *            comment this line out if a west negative
D   95:  *            convention is desired
D   96:  */
D   97:
D   98: #define WEST_POSITIVE      1
D   99:
D  100: /*
D  101:  *      comment card strings
D  102:  */
D  103:
D  104: #define CEN_LAT_CARD    "Center latitude = %f"
D  105: #define CEN_LON_CARD    "Center longitude = %f"
D  106: #define RES_X_CARD      "Longitude resolution (km) = %d"
D  107: #define RES_Y_CARD      "Latitude resolution (km) = %d"
D  108: #define VALID_UNIT_CARD "Valid calibration unit for band %d = %s \"%s\""
D  109:
D  110: /*
D  111:  *      CRITERIA holds the search parameters for
D  112:  *      image selection
D  113:  */
D  114:
D  115: typedef struct CRITERIA_
D  116: {
D  117:       int      begday;
D  118:       int      endday;
D  119:       int      begtim;
D  120:       int      endtim;
D  121:       int      begss;
D  122:       int      endss;
D  123: } CRITERIA;
D  124:
D  125: /*
D  126:  *      FILELIST holds the a list of files
D  127:  *      to be compared to the parameters in CRITERIA
D  128:  */
D  129:
D  130: typedef struct FILELIST_
D  131: {
D  132:       char      name[MAX_NAME_LEN];
D  133:       int      pos;
D  134:       int      time;
D  135:       struct      FILELIST_      *next;
D  136: } FILELIST;
D  137:
D  138: /*
```

```
D  139:  *       READPARM holds the specifications
D  140:  *       needed to read a line of image data
D  141:  */
D  142:
D  143: typedef struct READPARM_
D  144: {
D  145:       char      src_type[4];
D  146:       char      des_unit[4];
D  147:       char      src_unit[4];
D  148:
D  149:       int       begele;
D  150:       int       beglin;
D  151:       int       bufsiz;
D  152:       int       des_len;
D  153:       int       elem_res;
D  154:       int       line_res;
D  155:       int       maxele;
D  156:       int       maxlin;
D  157:       int       minele;
D  158:       int       minlin;
D  159:       int       numband;
D  160:       int       numele;
D  161:       int       numlin;
D  162:       int       src_len;
D  163:       int       ul_elem;
D  164:       int       ul_line;
D  165: } READPARM;
D  166:
D  167: /*
D  168:  *       mcidas prototypes
D  169:  */
D  170:
D  171: char *clit_(int *);
D  172:
D  173: int lit_(char *);
D  174: int lltora_(float *, float *, float *, float *, double *, double *);
D  175: int m0resred_(char *, int *, int *, int *);
D  176: int m0sxsend_(int *, int *);
D  177: int m0sxtrce_(char *, int);
D  178: int nv1eas_(float *, float *, float *, float *, float *, float *);
D  179: int nv1ini_(int *, int *);
D  180: int nvprep_(int *, int *);
D  181: int nv1sae_(float *, float *, float *, float *, float *, float *);
D  182: int sksecs_(int *, int *);
D  183:
D  184: /*
D  185:  *       MUG interface prototypes
D  186:  */
D  187:
D  188: int AddCard(char *, int, int *, int);
D  189: int CheckImgBounds(READPARM *, int, int);
D  190: int IsMugFormat(char *);
D  191: int MugNavEarthToImg(char *, float, float, float *, float *, char *);
D  192: int MugNavImgToEarth(char *, float, float, float *, float *, char *);
D  193: int PopFile(char *, int *, FILELIST **);
D  194: int PushFile(char *, int, FILELIST **);
D  195: int PushFileByName(char *, int, FILELIST **);
D  196: int PushMugFileByTime(char *, int, FILELIST **);
D  197: int ReadMugCards(char *, int *, int *, char *);
D  198: int ReadMugCalCod(char *, int *, int *);
D  199: int ReadMugRes(char *, float, float, float *, float *, char *);
D  200: int ReadMugDir(char *, int *, char *);
D  201: int ReadMugLine(char *, READPARM *, int, short *, char *);
D  202: int ReadMugNavCod(char *, int *, int, int, int *);
D  203: int ReadUnits(int *, int *, int *, char *);
D  204: int SelectMugImages(char *, int, int, CRITERIA *, FILELIST **, char *);
D  205: int SendCards(int *, int *);
D  206: int SendCod(int *, int);
D  207: int SendDir(int *);
D  208: int SendLine(short *, int);
D  209: int SendZeros(short *, int);
D  210: int TestMugImage(char *, CRITERIA *, int *, char *);
```

```
E    1:
E    2:
E    3: /*
E    4:
E    5:         subserv.c       j.benson      10/94
E    6:
E    7:
E    8:         This is called by exec from subserv_.c
E    9:
E   10:         It reconstructs the server environment, and continues the service
E   11:
E   12:         Note:  the varible Sname is not defined in this file
E   13:         it should be defined in the -DSname=value field on the
E   14:         compile.  value will be interpreted as the name of the
E   15:         application function that will perform the work. Eg aget_
E   16:
E   17:
E   18: */
E   19:
E   20: #include <memory.h>
E   21: #include <string.h>
E   22: #include <stdlib.h>
E   23:
E   24: #include "mcidas.h"
E   25: #include "servacct.h"
E   26:
E   27: /*  turn off trace */
E   28: extern int trace_;
E   29:
E   30:
E   31: #ifdef  __EMX__     /*  f2c compatibility for os/2  */
E   32: int xargc;
E   33: char **xargv;
E   34: #endif  /*   __EMX__  */
E   35:
E   36:
E   37: void main( int argc, char *argv[])
E   38: {
E   39:     servacct request_block;     /*  control block for transaction */
E   40:
E   41:     char       dbg[180];
E   42:
E   43:     int       istat;
E   44:     int       parsed_len;
E   45:
E   46:
E   47: #ifdef  __EMX__     /*  f2c compatibility for os/2  */
E   48:     xargc=argc;
E   49:     xargv=argv;
E   50: #endif /* __EMX__  */
E   51:
E   52:     /*
E   53:         We do not validate the transaction user and project.
E   54:         That should have been done by the main server previously    */
E   55:
E   56:     trace_ = 0;
E   57:
E   58: #ifdef DEBUG
E   59:     trace_ = 1;
E   60: #endif
E   61:
E   62:       (void)strcpy(dbg, "starting subserv");
E   63:       m0sxtrce_(dbg, strlen(dbg));
E   64:
E   65:
E   66:     /*  initialize McIDAS environment */
```

```
E   67:      {
E   68:          int i;
E   69:          initblok_( &i );
E   70:          if ( i != 0 ) return;
E   71:      }
E   72:
E   73:      /*  construct request block */
E   74:      memset( &request_block, 0, sizeof(request_block) );
E   75:
E   76:      * (int *) &request_block.server_address = atoi( argv[1] );
E   77:      request_block.server_port              = atoi( argv[2] );
E   78:      * (int *) request_block.user            = atoi( argv[3] );
E   79:      request_block.project                  = atoi( argv[4] );
E   80:      * (int *) request_block.transaction     = atoi( argv[5] );
E   81:      request_block.input_length             = atoi( argv[6] );
E   82:
E   83:      /*  put at least part of the request into the rb */
E   84:      strncpy( request_block.text, argv[7],
E   85:               strlen(argv[7]) < sizeof(request_block.text) ?
E   86:               strlen(argv[7]) : sizeof(request_block.text)  );
E   87:
E   88:      /*  initialize log fields  */
E   89:      m0sxlogi_( &request_block );
E   90:
E   91:      /*  parse the command  */
E   92:
E   93:      istat = Mcargfree(0);
E   94:      istat = M0cmdput( M0cmdparse(argv[7], &parsed_len));
E   95:
E   96:      /*  do the guts  */
E   97:      Sname( &request_block );
E   98:
E   99:      /*  termination */
E  100:      m0sxdone_( &request_block );
E  101:
E  102: }
```

```
F    1: /*      servutil.h
F    2:  *
F    3:  *      This is the main include file for the McIDAS
F    4:  *      ADDE server utilities.
F    5:  *
F    6:  */
F    7:
F    8: /*
F    9:  *      include files
F   10:  */
F   11:
F   12: #include      <stdio.h>
F   13: #include      <stdlib.h>
F   14: #include      <strings.h>
F   15: #include      "mcidas.h"
F   16:
F   17: /*
F   18:  *      return status
F   19:  */
F   20:
F   21: #define FAILURE      0
F   22: #define SUCCESS      1
F   23:
F   24: /*
F   25:  *      error handling
F   26:  */
F   27:
F   28: #define ERR_STAT_DIR        -2
F   29: #define ERR_MSG_DIR     "Unable to read contents of directory \n"
F   30:
F   31: #define ERR_STAT_MALLOC     -1
F   32: #define ERR_MSG_MALLOC  "Unable to allocate memory \n"
F   33:
F   34: #define ERR_MSG_UNKNOWN_SAT    "Calibration units unknown for this satellite \n"
F   35:
F   36: /*
F   37:  *      max lengths
F   38:  */
F   39:
F   40: #define IMG_DIR_LEN    64   /* len of image directory              */
F   41: #define MAX_CARD_LEN   80   /* len of  a single comment card       */
F   42: #define MAX_DIR_LEN    1024 /* len of directory where data lives   */
F   43: #define MAX_ERRMSG_LEN 72   /* byte len of err msg back to client */
F   44: #define MAX_ERR_LEN    256  /* len of error string                 */
F   45: #define MAX_NAME_LEN   1024 /* max. len of a image path and name   */
F   46: #define MAX_NUM_BAND   32   /* max. number of bands in an image    */
F   47: #define MAX_NUM_CARDS  250  /* max. number of mcidas comment cards*/
F   48:
F   49: /*
F   50:  *      comment card strings
F   51:  */
F   52:
F   53: #define VALID_UNIT_CARD "Valid calibration unit for band %d = %s \"%s\""
F   54:
F   55: /*
F   56:  *      CRITERIA holds the search parameters for
F   57:  *      image selection
F   58:  */
F   59:
F   60: typedef struct CRITERIA_
F   61: {
F   62:       int      begband;
F   63:       int      endband;
F   64:       int      begday;
F   65:       int      endday;
```

```
F   66:        int       begtim;
F   67:        int       endtim;
F   68:        int       begss;
F   69:        int       endss;
F   70: } CRITERIA;
F   71:
F   72: /*
F   73:  *      FILELIST holds the a list of files
F   74:  *      to be compared to the parameters in CRITERIA
F   75:  */
F   76:
F   77: typedef struct FILELIST_
F   78: {
F   79:        char       name[MAX_NAME_LEN];
F   80:        int       pos;
F   81:        int       time;
F   82:        struct       FILELIST_       *next;
F   83: } FILELIST;
F   84:
F   85: /*
F   86:  *      READPARM holds the specifications
F   87:  *      needed to read a line of image data
F   88:  */
F   89:
F   90: typedef struct READPARM_
F   91: {
F   92:        char       src_type[4];
F   93:        char       des_unit[4];
F   94:        char       src_unit[4];
F   95:
F   96:        int       begele;
F   97:        int       beglin;
F   98:        int       bufsiz;
F   99:        int       des_len;
F  100:        int       elem_res;
F  101:        int       line_res;
F  102:        int       maxele;
F  103:        int       maxlin;
F  104:        int       minele;
F  105:        int       minlin;
F  106:        int       numband;
F  107:        int       numele;
F  108:        int       numlin;
F  109:        int       src_len;
F  110:        int       ul_elem;
F  111:        int       ul_line;
F  112: } READPARM;
F  113:
F  114: /*
F  115:  *      mcidas routines
F  116:  */
F  117:
F  118: int m0sxsend_(int *, int *);
F  119: int m0sxtrce_(char *, int);
F  120:
F  121: /*
F  122:  *      server utility interface prototypes
F  123:  */
F  124:
F  125: int AddCard(char *, int, int *, int);
F  126: int CheckImgBounds(READPARM *, int, int);
F  127: int PopFile(char *, int *, FILELIST **);
F  128: int PushFile(char *, int, FILELIST **);
F  129: int PushFileByName(char *, int, FILELIST **);
F  130: int ReadCodLengths(char *, int *, int *);
F  131: int ReadUnits(int *, int *, int *, char *);
F  132: int SendCards(int *, int *);
F  133: int SendCod(int *, int);
F  134: int SendDir(int *);
F  135: int SendLine(int *, int);
F  136: int SendZeros(int *, int);
```

```
G    1: C ? MUGAREA -- Manipulate area data recevied by server for MUG demo
G    2: C ?    MUGAREA source pos1 oper pos2 destination
G    3: C ? Parameters:
G    4: C ?    source  | source dataset name, contains areas
G    5: C ?    pos1    | position of 1st area to use (default, most recent)
G    6: C ?    oper    | mathematical operation ADD, SUB, AVG are valid entries
G    7: C ?    pos2    | position of 2nd area to use (default, next most recent)
G    8: C ?    destination.pos | destination dataset and position
G    9: C ? Keywords:
G   10: C ?    SIZE = nlines neles  ! Size of area to get from server
G   11: C ? ----------
G   12:
G   13:       SUBROUTINE MAIN0
G   14:       IMPLICIT NONE
G   15: C
G   16: C
G   17: C-------external functions
G   18:
G   19:       character*12 cfr     ! left-justified integer->string
G   20:       integer iftok        ! make a character string an integer
G   21:       integer isdgch       ! is the character string digits characters?
G   22:       integer mcacal       ! get the calibration
G   23:       integer mcacou       ! write the comment cards
G   24:       integer mcacrd       ! read the comment cards
G   25:       integer mcaget       ! start server for lines of data
G   26:       integer mcalin       ! get a line of data from pipe
G   27:       integer mcanav       ! get the navigation
G   28:       integer mcaout       ! output the area using ADDE
G   29:       integer mcapfx       ! get the prefix
G   30:       integer mcaput       ! put the area
G   31:       integer mcasort      ! get sort conditions to pass to server
G   32:       integer mccmdint     ! get integer from command line
G   33:       integer mccmdkey     ! validate command line entries
G   34:       integer mccmdnum     ! number of entries with keyword
G   35:       integer mccmdstr     ! get string from command line
G   36:       integer mcdsnum      ! get number of positions in dataset
G   37: C
G   38: C--------parameter
G   39:
G   40:       integer MAXCARD
G   41:       parameter (MAXCARD = 500) ! max number of comment cards
G   42:       integer NLINMAX
G   43:       parameter (NLINMAX=1000)  ! max number of lines
G   44:       integer NELEMAX
G   45:       parameter (NELEMAX=1000)  ! max number of elements
G   46: C
G   47: C------- local variables
G   48: C
G   49:       character*12  cbday     ! begin day
G   50:       character*12  ceday     ! end day
G   51:       character*12  cposd     ! position of destination dataset
G   52:       character*12  oper_str  ! operation to perform on areas1&2
G   53:       character*40  sorts1(20)! sort strings to pass to server, 1st position
G   54:       character*40  sorts2(20)! sort strings to pass to server, 2nd position
G   55:       character*40  psorts(20)! sort string for putting
G   56:       character*40  dname     ! destination dataset name
G   57:       character*40  sname     ! store source name in here
G   58:       character*40  name      ! source dataset name, destination storage
G   59:
G   60:       integer buffer1(nelemax)  ! buffer for ADDE read/write
G   61:       integer buffer2(nelemax)  ! buffer for ADDE read/write
G   62:       integer buffer3(nelemax)  ! buffer for ADDE read/write
G   63:       integer cards1(MAXCARD)   ! area1 comment cards
G   64:       integer cards2(MAXCARD)   ! area2 comment cards
G   65:       integer cards3(MAXCARD)   ! area3 comment cards
G   66:       integer dir1(64)          ! area directory for 1st area
```

```
G    67:       integer dir2(64)          ! area directory for 2nd area
G    68:       integer dir3(64)          ! area directory for 3rd area
G    69:       integer dpos              ! destination dataset position
G    70:       integer h1                ! handle for mcaget
G    71:       integer h2                ! handle for mcaget
G    72:       integer h3                ! handle for mcaput
G    73:       integer i                 ! loop index
G    74:       integer ical1(10000)      ! cal block, area 1
G    75:       integer ical2(10000)      ! cal block, area 2
G    76:       integer ical3(10000)      ! cal block, area 3
G    77:       integer iele              ! number of elements to fetch
G    78:       integer ilin              ! number of lines to fetch
G    79:       integer inav1(1024)       ! nav block, area 1
G    80:       integer inav2(1024)       ! nav block, area 2
G    81:       integer inav3(1024)       ! nav block, area 3
G    82:       integer iread             ! number of directories read
G    83:       integer iret              ! return code
G    84:       integer iret1             ! return code
G    85:       integer iret2             ! return code
G    86:       integer ix                ! loop bound
G    87:       integer j                 ! loop bound
G    88:       integer jy                ! loop bound
G    89:       integer npos              ! number of positions in dataset
G    90:       integer pos1              ! 1st dataset position
G    91:       integer pos2              ! 2nd dataset position
G    92:       integer prefix1(250)      ! prefix for area 1
G    93:       integer prefix2(250)      ! prefix for area 2
G    94:       integer prefix3(250)      ! prefix for area 3
G    95:       integer neles             ! number of elements
G    96:       integer nlines            ! number of lines
G    97:       integer nsorts1           ! number of sorts conditions, 1st posn
G    98:       integer nsorts2           ! number of sorts conditions, 2nd posn
G    99:       integer nsortsp           ! number of put sort strings
G   100:      integer xdir1(65)          ! expanded area directory for 1st area
G   101:      integer xdir2(65)          ! expanded area directory for 2nd area
G   102:      integer xdir3(65)          ! expanded area directory for 3rd area
G   103:
G   104:       integer numkeys           ! number of valid keyword entries
G   105:       parameter(NUMKEYS=10)
G   106:       character*12 key_words(NUMKEYS) !  valid keywords
G   107:       data key_words/'AUX','BAN.D','CAL','DAY','LOC.ATE','MAG.NIFY',
G   108:      &               'SIZ.E','SU','TIM.E','DPO.SITION'/
G   109:
G   110:        nsorts1 = 0
G   111:        nsorts2 = 0
G   112:        nsortsp = 0
G   113: C
G   114: C-------check to see if keywords on command line are valid
G   115:
G   116:       if (mccmdkey(numkeys,key_words) .lt. 0) then
G   117:
G   118:            call edest('Ambiguous, illegal, or invalid keywords',0)
G   119:            goto 2000
G   120:       endif
G   121: C
G   122: C--------read in the source dataset name
G   123:
G   124:       if (mccmdstr(' ',1,'0',sname)  .lt.  0)  goto 2000
G   125:
G   126:       if (sname  .eq. '0') then
G   127:         call edest('Source dataset must be entered',0)
G   128:         goto 2000
G   129:       endif
G   130:
G   131:       if (isdgch(sname) .eq. 1) then
G   132:         call edest('Invalid source dataset name: '//sname,0)
G   133:         goto 2000
G   134:       endif
G   135: C
G   136: C-------read in the destination dataset name
G   137: C
G   138:       if (mccmdstr(' ',5,'0',name) .lt. 0)  goto 2000
```

```
G  139:
G  140:       if (name .eq. '0') then
G  141:         call edest('Destination dataset must be entered',0)
G  142:         goto 2000
G  143:       endif
G  144:
G  145:       if (isdgch(name) .eq. 1) then
G  146:         call edest('Invalid destination dataset name: '//name,0)
G  147:         goto 2000
G  148:       endif
G  149:
G  150: C
G  151: C-------Separate the dataset name from the position
G  152: C
G  153:       dname = name(1:index(name,'.')-1)
G  154:       cposd = name(index(name,'.')+1:)
G  155: C
G  156: C------cposd must be positive integer, do not allow anything else..
G  157: C
G  158:       if (cposd(1:1) .lt. '1' .or. cposd(1:1) .gt. '9') then
G  159:
G  160:         call edest('Invalid position specified in destination '//
G  161:      &           'dataset parameter.',0)
G  162:         call edest('Must be a positive integer',0)
G  163:         goto 2000
G  164:
G  165:        endif
G  166: C
G  167: C-------make sure that cposd is a number
G  168: C
G  169:       if (isdgch(cposd) .ne. 1) then
G  170:
G  171:         call edest('Invalid position specified in destination '//
G  172:      &           'dataset parameter.',0)
G  173:         call edest('Must be a positive integer',0)
G  174:         goto 2000
G  175:
G  176:        endif
G  177:
G  178:       dpos = iftok(cposd)
G  179: C
G  180: C------read in the bounds of the destination dataset name
G  181: C
G  182:       npos = mcdsnum(dname,'AREA')
G  183:         if (npos  .lt.  0) then
G  184:           if (npos .eq. -1)  then
G  185:             call edest('Unable to resolve dataset dname= '//dname,0)
G  186:             goto 2000
G  187:           elseif (npos .eq. -2) then
G  188:             call edest('Unable to resolve dataset dname= '//dname,0)
G  189:             goto 2000
G  190:           endif
G  191:           call edest('Bad mcdsnum return code: '//cfr(npos),0)
G  192:           goto 2000
G  193:         endif
G  194:
G  195:
G  196:       if (dpos .gt. npos) then
G  197:
G  198:          call edest('Destination dataset position ('//cposd//')'//
G  199:      &           ' exceeds dataset limits ('//cfr(npos)//')',0)
G  200:          goto 2000
G  201:
G  202:       endif
G  203: C
G  204: C------read in the bounds for the source dataset sname
G  205: C
G  206:
G  207:       npos = 8
G  208:         if (npos  .lt.  0) then
G  209:           if (npos .eq. -1)  then
G  210:             call edest('Unable to resolve dataset sname= '//sname,0)
```

```
G  211:              goto 2000
G  212:            elseif (npos .eq. -2) then
G  213:              call edest('Unable to resolve dataset sname= '//sname,0)
G  214:              goto 2000
G  215:            endif
G  216:            call edest('Bad mcdsnum return code: '//cfr(npos),0)
G  217:            goto 2000
G  218:          endif
G  219: C
G  220: C-------Read in the first position
G  221: C
G  222:          if (mccmdint(' ',2,'First Dataset Position',0,
G  223:         &              -npos,npos,pos1)  .lt.  0)  goto 2000
G  224: C
G  225: C-------Read in the second position
G  226: C
G  227:          if (mccmdint(' ',4,'Second dataset position',-1,
G  228:         &              -npos,npos,pos2) .lt.  0)  goto 2000
G  229: C
G  230: C------read in the operation to perform
G  231: C
G  232:          if (mccmdstr(' ',3,'ADD',oper_str) .lt. 0) then
G  233:            call edest('Invalid operator read in from command line:'
G  234:         &                  //oper_str,0)
G  235:            call edest('Use ADD, SUB, or AVG',0)
G  236:            goto 2000
G  237:          endif
G  238: C
G  239: C-------make sure operation entered is valid
G  240: C
G  241:          if (oper_str(1:3) .ne. 'ADD' .and. oper_str(1:3) .ne. 'SUB'
G  242:         &            .and. oper_str(1:3) .ne. 'AVG') then
G  243:
G  244:            call edest('Invalid operator:'//oper_str,0)
G  245:            call edest('Use ADD, SUB, or AVG',0)
G  246:            goto 2000
G  247:          endif
G  248: C
G  249: C-----read in the DAY requested
G  250: C
G  251:          if (mccmdnum('DAY')  .gt.  0)  then
G  252:
G  253:              if (mccmdstr('DAY',1,'X',cbday) .lt. 0)  goto 2000
G  254:              if (mccmdstr('DAY',2,cbday,ceday) .lt. 0)  goto 2000
G  255:
G  256:            nsorts1 = nsorts1 + 1
G  257:            sorts1(nsorts1) = 'DAY '//cbday//ceday
G  258:            nsorts2 = nsorts2 + 1
G  259:            sorts2(nsorts2) = 'DAY '//cbday//ceday
G  260:          endif
G  261:
G  262:          nsorts1 = 0
G  263:          nsorts2 = 0
G  264:
G  265:          if (mcasort(nsorts1,sorts1,1) .lt. 0) then
G  266:            call edest('Failed to return standard sort parms',0)
G  267:            call edest('for first position',0)
G  268:            goto 2000
G  269:          endif
G  270:
G  271:          if (mcasort(nsorts2,sorts2,1) .lt. 0) then
G  272:            call edest('Failed to return standard sort parms',0)
G  273:            call edest('for second position',0)
G  274:            goto 2000
G  275:          endif
G  276: C
G  277: C------add position to the sort strings
G  278: C
G  279:          nsorts1 = nsorts1 + 1
G  280:          sorts1(nsorts1) = 'POS '//cfr(pos1)
G  281:          nsorts2 = nsorts2 + 1
G  282:          sorts2(nsorts2) = 'POS '//cfr(pos2)
```

```
G  283: C
G  284: C-------get the Size
G  285: C
G  286:         if (mccmdint('SIZ.E',1,'Number of lines',145,1,1000,ilin)
G  287:       &                    .lt.  0)  goto 2000
G  288:         if (mccmdint('SIZ.E',2,'Number of eles',288,1,1000,iele)
G  289:       &                    .lt.  0)  goto 2000
G  290:
G  291:         nsorts1 = nsorts1 + 1
G  292:         sorts1(nsorts1) = 'SIZE '//cfr(ilin)//cfr(iele)
G  293:         nsorts2 = nsorts2 + 1
G  294:         sorts2(nsorts2) = 'SIZE '//cfr(ilin)//cfr(iele)
G  295: C
G  296: C----make the call to mcaget to start the serving of position 1
G  297: C
G  298:         name = sname
G  299:         iret = mcaget(name,nsorts1,sorts1,'TEMP','I4',
G  300:       &                    NELEMAX*4,1,dir1,h1)
G  301:         if (iret  .lt.  0) then
G  302:
G  303:          call edest('Fail in mcaget for position 1',0)
G  304:          goto 2000
G  305:         endif
G  306: C
G  307: C----make the call to mcaget to start the serving of position 2
G  308: C
G  309:         name = sname
G  310:         iret = mcaget(name,nsorts2,sorts2,'TEMP','I4',
G  311:       &                    NELEMAX*4,1,dir2,h2)
G  312:         if (iret  .lt.  0) then
G  313:
G  314:          call edest('Fail in mcaget',0)
G  315:          goto 2000
G  316:         endif
G  317: C
G  318: C--------Make sure the two areas are of identical size and shape
G  319: C
G  320:         do 600 ix = 7,15
G  321:
G  322:          if (dir1(ix) .ne. dir2(ix)) then
G  323:            call edest('Area mismatch at position '//cfr(ix),0)
G  324:            call edest('Area 1 value: '//cfr(dir1(ix)),0)
G  325:            call edest('Area 2 value: '//cfr(dir2(ix)),0)
G  326:            goto 2000
G  327:          endif
G  328:  600        continue
G  329:
G  330:         dir1(11) = 1
G  331: C
G  332: C----get the nav and cal blocks
G  333: C
G  334:         if (mcanav(h1,inav1) .ne. 0)  goto 2000
G  335:
G  336:         if (mcacal(h1,ical1) .ne. 0)  goto 2000
G  337:
G  338:         if (mcanav(h2,inav2) .ne. 0)  goto 2000
G  339:
G  340:         if (mcacal(h2,ical2) .ne. 0)  goto 2000
G  341:
G  342:         iread = 1
G  343: C
G  344: C------start the server transaction to put
G  345: C
G  346:         psorts(1) = 'POS '//cposd
G  347:         iret = mcaput(dname,1,psorts,dir1,inav1,ical1)
G  348:         if (iret .ne. 0)  then
G  349:
G  350:          call edest('Error in setting up put',0)
G  351:          goto 2000
G  352:
G  353:         endif
G  354: C
```

```
G  355: C------read in the lines of imagery from the two areas
G  356: C
G  357:  1000      continue
G  358:
G  359:         iret1 = mcalin(h1,buffer1)
G  360:
G  361:        if (iret1  .lt. 0)  then
G  362:           call edest('Fail in mcalin for area1 at line '//
G  363:      &                    cfr(iread),0)
G  364:           goto 2000
G  365:         endif
G  366:
G  367:        if (mcapfx(h1, prefix1) .lt. 0) then
G  368:         call edest('Prefix Read in area 1 has failed',0)
G  369:         goto 2000
G  370:        endif
G  371:
G  372:         iret2 =  mcalin(h2,buffer2)
G  373:
G  374:        if (iret2  .lt. 0)  then
G  375:           call edest('Fail in mcalin for area2 at line '//
G  376:      &                    cfr(iread),0)
G  377:           goto 2000
G  378:          endif
G  379:
G  380:       if (mcapfx(h2, prefix2) .lt. 0) then
G  381:         call edest('Prefix Read in area 2 has failed',0)
G  382:         goto 2000
G  383:       endif
G  384:
G  385:       if (iret1 .eq. 0 .and. iret2 .eq. 0) then
G  386: C
G  387: C------got something from pipe that is area data, perform operation
G  388: C
G  389:         do 1200 ix = 1, dir1(10)
G  390:
G  391:           if (oper_str(1:3) .eq. 'ADD')  then
G  392:                  buffer3(ix) = min0(buffer1(ix) + buffer2(ix),255)
G  393:           elseif (oper_str(1:3) .eq. 'SUB')   then
G  394:                  buffer3(ix) = max0(buffer1(ix) - buffer2(ix),0)
G  395:           elseif (oper_str(1:3) .eq. 'AVG')  then
G  396:                  buffer3(ix) = (buffer1(ix) + buffer2(ix))/2.
G  397:           else
G  398:             call edest('Bad operator specified',0)
G  399:             goto 2000
G  400:           endif
G  401:
G  402:  1200          continue
G  403: C
G  404: C-------put the buffer
G  405: C
G  406:         call pack(dir1(10),buffer3,buffer3)
G  407:
G  408:           if (mcaout(buffer3)  .lt. 0)  then
G  409:            call edest('Write failed ',0)
G  410:            goto 2000
G  411:         endif
G  412:
G  413:         iread = iread + 1
G  414:         goto 1000
G  415:
G  416:       elseif ( (iret1 .eq. 0 .and. iret2 .eq. 1) .or.
G  417:      &          (iret1 .eq. 1 .and. iret2 .eq. 0)  ) then
G  418:
G  419:           call edest('Pipes 1 and 2 did not empty at the same time',0)
G  420:       else
G  421:           if (mcacrd(h1,cards1)  .ne. 0)  then
G  422:             call edest('Read of comment cards from area 1 failed',0)
G  423:             goto 2000
G  424:           endif
G  425:           if (mcacrd(h2,cards2)  .ne. 0)  then
G  426:             call edest('Read of comment cards from area 2 failed',0)
```

```
G  427:              goto 2000
G  428:            endif
G  429: C
G  430: C------now write the comment cards
G  431: C
G  432:             if (dir1(64) .gt. 0) then
G  433:
G  434:               if (mcacou (cards1)  .ne. 0) then
G  435:                   call edest('Failed to write comment cards',0)
G  436:                   goto 2000
G  437:               endif
G  438:
G  439:             endif
G  440:         endif
G  441:
G  442:         call sdest('MUGAREA:  Done',0)
G  443:
G  444:   900    format(1x,i3,1x,2(i12,1x))
G  445:   2000 return
G  446:           end
```

Development Environment for McIDAS-X and -OS2

Presented by

Tom Whittaker

McIDAS Development Team Manager

Session 3

McIDAS Developer/Operator Training

October 23-25, 1995

Table of Contents

Setting up the environment for McIDAS-X 3-1
- Setting up your user environment 3-2
- The McIDAS-X library 3-3
- Testing your code 3-6
- Making HELPs 3-7

Setting up the environment for McIDAS-OS2 3-8
- Managing the software for locally developed code 3-8
- Required software for local development 3-9
- Considerations for functions and subroutines 3-9
- Compiling and linking 3-10
- Building HELP files 3-11
- Dynamic Link Libraries (DLLs) 3-12

Setting up the environment for McIDAS-X

To create a development environment for McIDAS-X, you must first set up a proper user environment. Verify that you or your system administrator has properly installed the McIDAS software in the user account named *mcidas*.The instructions are in Chapter 1 of the *McIDAS-X Users Guide*. Once McIDAS is installed, the *mcidas* account will have two sets of directories:

- package directories
- installation directories

Package directories

Each version of McIDAS-X and other McIDAS packages, such as McIDAS-XCD, builds its own set of directories. The names of the directories depend on the package name and version number. For example, the McIDAS-X 2.1 package directories and their contents are listed below.

Directory	Contents
~mcidas/mcidas2.1/src	McIDAS-X 2.1 source files, help files and binaries
~mcidas/mcidas2.1/data	McIDAS-X 2.1 data files

Installation directories

The installation directories and their contents are listed below.

Directory	Contents
~mcidas/admin	administrative files
~mcidas/bin	program executables
~mcidas/data	data files
~mcidas/help	help files
~mcidas/inc	include files
~mcidas/lib	libraries

Your user account

Verify that your account is set up as a normal McIDAS user, following the steps below. See Chapter 2 of the *McIDAS-X Users Guide* for additional information.

1. Login to your account.

2. Modify the environment variable PATH in your .profile (ksh) or .cshrc (csh) files. Insert $HOME/mcidas/bin, the directory with your site's locally developed code (for example, ~mclocal/mcidas/bin), and ~mcidas/bin in your PATH, in that order.

3. You may also need to modify your PATH if it does not contain all of the required directories or have them in the correct order, as listed below.

Operating system	**Modifications**
AIX 3.2.5 and 3.2.5p	none
HP-UX 9.0.3 and 9.0.5	add /usr/bin/X11
IRIX 5.2 and 5.3	none
SunOS 4.1.3	add /usr/lang and /usr/openwin/bin
Solaris 2.3 and 2.4	add /opt/SUNWspro/bin, /usr/ccs/bin, and /usr/openwin/bin; if your PATH contains /usr/ucb, it must follow /opt/SUNWspro/bin, /usr/ccs/bin, and /usr/bin

4. Logout and login again for the changes to take effect.

Your development environment

Before doing local development, you must create the following directories.

$HOME/mcidas
$HOME/mcidas/bin
$HOME/mcidas/data
$HOME/mcidas/src
$HOME/mcidas/help
$HOME/mcidas/lib

Every McIDAS-X upgrade provides a new library of functions, which is placed in the file ~mcidas/lib/libmcidas.a. As a developer, you can link to this library if you are only developing commands. If you are developing functions and subroutines, you can either copy this library to your $HOME/mcidas/lib directory, or create and use your own development (usually temporary) library for your function and subroutine object code.

There are two ways to compile your code: using the **fx** script, which invokes the **mccomp** and **mcar** scripts to do the compiles and linking, or using the latter routines directly. **mccomp** and **mcar** are provided to keep system-dependent compiler and library options transparent to the developer.

If you develop subroutines or functions, as opposed to just commands, it is better to use **mccomp** and **mcar** directly because you can put your object code into a separate library and link from there. The **fx** script, on the other hand, puts all object code into the McIDAS library, which forces you to copy libmcidas.a into your $HOME/mcidas/lib each time you start a different project.

If you use the **fx** script for compiling, follow these steps:

1. Make a fresh link to the main.o file in the *mcidas* account.

```
cd $home/mcidas/src
rm main.o
ln -s ~mcidas/lib/main.o .
```

2. Make a fresh link to the libmcidas.a file in the *mcidas* account. If you have library routines to put in libmcidas.a, use `cp` instead of `ln -s` below.

```
cd $HOME/mcidas/lib
rm libmcidas.a
ln -s ~mcidas/lib/libmcidas.a .
```

3. Make fresh links from $HOME/mcidas/src to the include files in ~mcidas/inc.

4. The **fx** script uses the scripts **mccomp** and **mcar** to do the compiles and library updates. They are located in ~mcidas/bin. If you need to modify the compiler options, make a private version of **mccomp** in your $HOME/mcidas/bin directory. Before modifying the archive options, make a private version of **mcar** in your $HOME/mcidas/bin directory.

5. On IRIX, there is a system-supplied command called **fx** in /usr/bin. To use the McIDAS-X **fx** command, you must have the mcidas bin directories (such as ~mcidas/bin and $HOME/mcidas/bin) before /bin and /usr/bin in your PATH.

6. Recompile your locally developed software using the new **fx** script and mcidas library. You may encounter problems when compiling. For example, using Fortran direct I/O with RECL= is not supported since it is not portable. On some platforms, RECL=7 means the record length is 7 bytes; on other platforms, it means 7 words. Use the McIDAS LW I/O or LB I/O instead.

We recommend using a makefile for compiling. For example:

```
# Sample makefile for local McIDAS development...September, 1995

#-----------------------------------------------------------------
# First section are things you'll set up for every project...
#-----------------------------------------------------------------

# The list of .pgm's to compile; comment out if not used.

MCFPSRC = run.pgm \
          tvanot.pgm

# The list of .for's to compile;  comment out if not used.

MCFFSRC = runaix.for

# The list of .c's to compile; comment out if not used.

# MCCFSRC =

# The list of .dlm's for compile: comment out if not used.

# MCDLMSRC =

# Derive the object file names; comment out any not used...

MCFPBIN = $(MCFPSRC:.pgm=.mx)
MCFFOBJ = $(MCFFSRC:.for=.o)
# MCCFOBJ = $(MCCFSRC:.c=.o)
# MCDLMOBJ = $(MCCDLMSRC:.dlm=.o)

# The name of the development library; if not using development
#   library (that is, there is no MCFFSRC or MCCFSRC), be sure
#   to set MYMCLIBL =
MYMCLIB      = dev
MYMCLIBA     = lib$(MYMCLIB).a
MYMCLIBL     = -l$(MYMCLIB)

# MYMCLIBL     =

#-----------------------------------------------------------------
# Now define the McIDAS environment and the developer's home
# directory.
#-----------------------------------------------------------------

# McIDAS root directory and associated subdirectories

MCROOTDIR    = /home/mcidas
MCINCDIR     = $(MCROOTDIR)/inc
MCLIBDIR     = $(MCROOTDIR)/lib
MCBINDIR     = $(MCROOTDIR)/bin

# the suffix of the name of the McIDAS library
```

```
MCLIB         = mcidas

# developer's root directory and associated subdirectories

MYROOTDIR     = $(HOME)
MYMCBINDIR    = $(MYROOTDIR)/mcidas/bin

# the location of the core version of main.o

MAIN_O        = $(MCLIBDIR)/main.o

#-----------------------------------------------------------------
# Set the DEBUG flags for the compiler.
#-----------------------------------------------------------------

DEBUG         =

#-----------------------------------------------------------------
# L_OBJ can force certain object files to be used during the
# link instead of getting it from a library; This macro can
# also be modified on the command line of the make call
#-----------------------------------------------------------------

L_OBJ         =

#-----------------------------------------------------------------
# Now define the location of includes, libraries, and compile
# scripts.
#-----------------------------------------------------------------

# the list of include file arguments

INCARGS       = -I. -I$(MCINCDIR)

# the list of library file arguments

LIBARGS       = -L. -L$(MCLIBDIR) $(MYMCLIBL) -l$(MCLIB) -lX11

# the compile, link and library archive commands for McIDAS

COMPCMD       = $(MCBINDIR)/mccomp
LINKCMD       = $(MCBINDIR)/mccomp
LIBARC        = $(MCBINDIR)/mcar
CONVDLM       = $(MCBINDIR)/convdlm

#-----------------------------------------------------------------
# Create the suffix rules for compiles, etc.
#   ( the "$*" refers to the root name of the prerequisite file )
#-----------------------------------------------------------------

.SUFFIXES: .o .mx .for .c .pgm

# compile the .c's

.c.o:
$(COMPCMD) $(INCARGS) $(DEBUG) -c $*.c
$(LIBARC) $(MYMCLIBA) $*.o

# compile the .for's

.for.o:
$(COMPCMD) $(INCARGS) $(DEBUG) -c $*.for
$(LIBARC) $(MYMCLIBA) $*.o

# compile the .dlm's

.dlm.o:
$(CONVDLM) $*.dlm
$(COMPCMD) $(INCARGS) $(DEBUG) -c $*1.f
$(COMPCMD) $(INCARGS) $(DEBUG) -c $*2.f
```

```
$(COMPCMD) $(INCARGS) $(DEBUG) -c $*3.f
$(LIBARC) $(MYMCLIBA) $*1.o $*2.o $*3.o

# compile and link the .pgms and copy the resulting .mx to
# ~/mcidas/bin

.pgm.mx:
$(COMPCMD) $(INCARGS) $(DEBUG) -c $*.pgm
$(COMPCMD) $(MAIN_O) $(L_OBJ) $*.o $(LIBARGS) -o $*.mx
cp $*.mx $(MYMCBINDIR)

install.all: $(MCFPBIN)

#-----------------------------------------------------------------
# The .o files are dependent on the .for and .c source code.
# The .mx's are dependent on the .pgm source, and the other .o files.
#-----------------------------------------------------------------

$(MCFFOBJ): $(MCFFSRC)
# $(MCCFOBJ): $(MCCFSRC)
$(MCFPBIN): $(MCFPSRC) $(MCFFOBJ) $(MCCFOBJ)

#-----------------------------------------------------------------
# Clean out the residue...
#-----------------------------------------------------------------
clean:
rm $(MCFPBIN) $(MCFFOBJ) $(MCCFOBJ) $(MYMCLIBA)

# end-of-sample-makefile
```

Testing your code

Keep your testing environments separate from all user environments. The primary points to keep in mind are the following:

- get a good data sample; for example, the dataset provided with the *McIDAS-X Learning Guide*
- verify that the account is set up like a user account
- set your MCPATH environment variable appropriately
- use different names for the source and executables if you change core McIDAS code
- be aware that we will never issue a core McIDAS command beginning with the letter **y**

If your changes will become part of your routine operations, then you should be prepared to recompile, link, and retest with each upgrade that you install. Using makefiles will help this effort.

Create McIDAS-style HELPs (lines prefixed by C ? in the source code) for all locally developed commands using the template below. Note that macro commands, which have the **.mac** extension, use the double-quote (") instead of C as the comment character.

```
C ? NAME -- Describe the purpose of this command
C ?    NAME FUNCT1 parm1 parm2 <keywords> "quote
C ?    NAME FUNCT2 parm1 parm2 <keywords>
C ? Parameters:
C ?    FUNCT1   | describe the purpose of this function switch/option
C ?    FUNCT2   | describe the purpose of this function switch/option
C ?    parm1    | describe this parameter (def=default value)
C ?    parm2    | describe this parameter (def=default value)
C ?    "quote   | describe the contents of the quote string
C ? Keywords:
C ?    KEYNAME=    | describe values (def=default)
C ?    KEY2=YES    | describe effect (def=default)
C ?    KEYS= first second  | describe values (def=default)
C ? Remarks:
C ?    Add remarks, from most to least important. Use complete
C ?    sentences. If there are multiple remarks, separate them
C ?    with a single blank line, as below.
C ?
C ?    Always end the help section with a line of 10 dashes, as below.
C ? ----------

      SUBROUTINE MAIN0
      IMPLICIT NONE
C --- symbolic constants & shared data
C --- external functions
C --- local variables
C --- initialized variables
```

To produce a McIDAS help file from your source code, do the following:

1. Change to the help directory.

   Type: **cd $HOME/mcidas/help**

2. Make help files from your code in your $HOME/mcidas/src directory by entering the two command lines below.

   Type: **mcmkhelp $HOME/mcidas/src/*.pgm**
   mcmkhelp $HOME/mcidas/src/*.mac

3. Have all users who run locally developed commands add the appropriate directories to the MCPATH setting. For example, users who run commands in the mclocal account must add ~mclocal/mcidas/help to their MCPATH. Users who run commands in their own account, must add $HOME/mcidas/help to their MCPATH.

Setting up the environment for McIDAS-OS2

The *McIDAS-OS2 Users Guide* discusses the user environment, while the old *McIDAS Applications Programming Manual* provides details of the developer environment. In both cases, the directory tree and all developer settings are established during the installation, except as noted below.

The directory tree established after installing both the McIDAS-OS2 and the Development software is shown below.

Directory	**Contents**
\mcidas\tools	libraries, scripts, editor
\mcidas\source	core source code: do not change
\mcidas\working	local source code
\mcidas\code	core executable code: do not change
\mcidas\user\code	local executable code
\mcidas\data	data of all kinds
\mcidas\help	.HLP files

Managing the software for locally developed code

Keep the source code for all locally developed code in the \mcidas\working subdirectory. The McIDAS upgrade procedures do not affect the contents of this directory. Also, begin the names of your .PGM files with the letter **y**.

Do not change the LIBMC.LIB as delivered with each upgrade; by default, the **f.cmd** uses MCUSER.LIB for local subroutines and functions. Delete this library between projects.

Finally, use **make** or create compile scripts to deal with projects having several things to compile and/or link. You will generally need to recompile and link your local code after installing each upgrade. For example:

```
REM Rebuild the "YPROG" command
SETLOCAL
CD \MCIDAS\TOOLS
DEL MCUSER.LIB
CALL F MYSUB1 CLI
CALL F MYSUB2 LI
CALL F YPROG L
```

Required software for local development

McIDAS-OS2 is built using the **emx/gcc** compilers and the **f2c** Fortran translator. You may obtain a copy of these from SSEC either on diskette or via anonymous ftp, following the instructions in the upgrade information. Once you obtain the .ZIP file and unpack its contents, a directory tree under \EMX\ is created. You should read the information in \EMX\DOC\ and \EMX\BOOK\. Use the **view** command in OS/2 to read the .INF files.

The \EMX directory contains a script called **setemx.cmd**. If you run this script, the environment is set up such that you can compile your code within that session only. If you do a large amount of local development, take the **set** commands from this script and, after substituting the correct drive value, put them in the \CONFIG.SYS file: Do NOT put any other statements from the **setemx.cmd** file into \CONFIG.SYS.

Considerations for functions and subroutines

The IBM linker, LINK386, restricts the searching of libraries to a prescribed order; once a library is completely searched, the next one in sequence is searched. In the **f.cmd** compile script, MCUSER.LIB is always searched before LIBMC.LIB.

If you create a subprogram that is called only by a routine in the LIBMC.LIB (this would only happen if you intended to replace something in LIBMC.LIB), your routine will not be linked because the object code is put into the MCUSER.LIB by **f.cmd**. In this case, copy the LIBMC.LIB into MCUSER.LIB before compiling. In the example recompile script above, change the line:

```
DEL MCUSER.LIB
```

to:

```
COPY LIBMC.LIB MCUSER.LIB
```

The **f.cmd** compile script is provided with McIDAS-OS2 in the \MCIDAS\TOOLS directory when you install the Development software. This script, written in REXX, provides all the commands you will need to compile and link your code. The general form of the command is:

```
F <name> <option>
```

where: `<name>` is the name of the source file without an extension
`<option>` describes the type of compile to do and the type of source code

Entering F without <name> <option> produces this help information:

```
   ------ McIDAS compile script -----
F <name> <function: L,LI,NM,MAC,NO,C,CLI>
F <name>  DL <defname> <dllname>

Specify <name> with NO extensions!
Extensions supplied are:
      L -> .PGM     (Fortran McIDAS commands)
     LI -> .FOR     (Compile and LIB Fortran subprograms)
     DL -> .DLM     (Dynamic load modules)
     NM -> .FP      (Non-McIDAS programs; put .EXE into \MCIDAS\TOOLS)
    NMC -> .FP      (Non-McIDAS programs; put .EXE into \MCIDAS\CODE)
    MAC -> .MAC     (McIDAS macros)
     NO -> .FOR     (Compile Fortran code only - no LIB step)
      C -> .C       (Compile C code only - no LIB step)
    CLI -> .C       (Compile C code and insert into library)
     CP -> .C       (C language non-McIDAS commands)
     CL -> .C       (C language McIDAS commands)
```

This will search for <name> first in the current directory, then in \mcidas\working, and finally in \mcidas\source.

The environment variables are shown below:

Variable	Function
useworking=NO	suppresses the normal search of \mcidas\working
mcout=c:\directory	writes .exe files into \directory
mapout=c:\directory	writes MAP files into \directory

The McIDAS-OS2 HELP command uses the files placed in the \mcidas\help\ directory. The files are the names of McIDAS-OS2 commands followed by a .HLP extension. For example, IMGDISP.HLP is the HELP information for the IMGDISP command. If you follow the standards previously described for McIDAS-X for formatting the help section of your locally developed commands, you can use the **mkhelp** program in \mcidas\tools\ to extract this information and place it in the proper directory.

The command syntax for **mkhelp** is shown below. You must run it from the OS/2 command prompt, not within McIDAS-OS2.

```
mkhelp <filename> <options> <output_directory>

where <filename> is the name of the source code file for the
                 McIDAS-OS2 command you want to extract the
                 HELP information from.  You may also specify
                 a wildcard with an extention (eg., *.PGM) to
                 do all the files with this extension.

      <options> may be LIST if you just want the extracted
                        HELP information listed to the screen
                   the name of a 'template' file containing
                        lines of text that are to be included
                        in the HELP (we recommend that you
                        include a message like "This code
                        was developed at NXYZ")
                   X if you want neither of the above options
       <output_directory> to specify the directory where the
                   .HLP file(s) will be written. The default
                   is \mcidas\help\.
```

If you run this command with no arguments, the following one-line help message is displayed:

```
MKHELP  <filename>  <templatefile | LIST | X>  <output directory>
```

Navigation and calibration routines are handled in McIDAS-OS2 as Dynamic Link Library modules (DLLs). They are not actually linked into your program when it's compiled, but rather remain in a special file that can be loaded into memory and then linked into your program after it begins to run. OS/2 makes extensive use of this feature; McIDAS-OS2 uses it not only for navigation and calibration, but also for all graphics displays and part of the MDX command group.

You must make three DLLs for each navigation or calibration module and one for each graphics driver (platform). You must explicitly perform each of these, as the **f.cmd** does not. For example, to compile a navigation module for type ABCD, you would run these commands:

```
F NVXABCD DL NAVLIB NV1ABCD
F NVXABCD DL NAVLIB NV2ABCD
F NVXABCD DL NAVLIB NV3ABCD
```

The NAVLIB parameter defines the definitions file to use for creating navigation library DLL routines. The final parameter is the name of the created .DLL file. You must replace the X in the root name (for example, **nvxgvar**) with the digits 1-3 (for example, **nv1gvar**, **nv2gvar**, **nv3gvar**) as illustrated.

A similar process is used for calibration:

```
F KBXABCD DL CALLIB KB1ABCD
F KBXABCD DL CALLIB KB2ABCD
F KBXABCD DL CALLIB KB3ABCD
```

CALLIB designates the definitions file for calibration.

Graphics are compiled using just one F.CMD per display type:

```
F GRADVCn DL GRALIB  GRADVCn
```

where *n* is the number assigned to the particular display type.

Writing GUIs for McIDAS using Tcl/Tk

Presented by

Susan Gorski *- McIDAS Applications Programmer*

David Glowacki *- McIDAS Systems Programmer*

Session 4

McIDAS Developer/Operator Training

October 23-25, 1995

Table of Contents

Overview 4-1

Terminology 4-2

Tcl syntax and structure 4-3

Variables 4-5

Tk syntax 4-7
- Terms 4-7
- Examples 4-8
- Error reporting 4-8

Packing in Tk 4-9

Using external procs 4-10

Widgets 4-11

Useful procs 4-16
- Find a User Common value 4-16
- Get today's date 4-16
- Get the current time 4-17
- List the U.S. states 4-18
- List all available areas 4-19
- Set standard options 4-21
- Run a McIDAS command; send output to listbox 4-21
- Run a McIDAS command; send output to text window . . . 4-23

Considerations for building a GUI 4-26

Sample GUI code 4-27

Solution set 4-38

Resources 4-44

Overview

McIDAS user interfaces in the past have included: a command window, a soft tablet, a hard tablet and the F Key menu system. When we began developing a Graphical User Interface for McIDAS, we assessed different underlying packages and chose Tcl/Tk.

Tcl (Tool command language) and the Tk (Tool kit) are two software packages developed by John Ousterhout at the UC-Berkeley. Together they make up a system for developing Graphical User Interface (GUI) applications. Tcl is an interpretive scripting language with variables and control structures; it contains the control portion of a GUI. Tk is a tool kit of graphical widgets that can be accessed from a Tcl script. Tk contains the visual portion of a GUI.

The ease of programming in Tcl/Tk allows for rapid McIDAS GUI development. With very little effort, a GUI can be prototyped for a look-and-feel evaluation before any functionality is added. Because Tcl/Tk source code is freely available, you can obtain the development package at no cost, making it more accessible. Tcl/Tk code makes it easy for you to create your own GUI, and alter the GUIs we create.

The McIDAS GUI is a series of intuitive menus and command GUIs for running McIDAS commands. It displays defaults, offers easy selection of options and limits commands to decrease complexity. One such GUI is Merlin, a freely distributed package that uses McIDAS and Tcl/Tk as its base. Because Merlin has no command line, all requests for data display are through the GUI. To use MERLIN as a base for other projects with more specific applications, we also developed *sidecars*.

This training session provides the information you need to write your own GUI using Tcl/Tk.

Terminology

The terms below are used throughout this section.

GUI	Graphical User Interface
pack	Tk command for arranging slave widgets inside a master widget
proc	submodule of Tcl code that allows the code to be reused in different scripts
progressive disclosure	giving users only as many choices as they need to make a simple command, but allowing expanded functionality as needed
sidecar	a set of GUI applications developed for a project that can be run alongside MERLIN
Tcl	Tool command language; the scripting language that contains the control portion of a GUI developed in Tcl/Tk
tclsh	Tcl shell application
Tk	Tool kit; a.package of graphical widgets that contain the visual portion of a Tcl/Tk GUI
unpack	the *pack forget* command in Tk; when a widget is unpacked, it is not displayed in the master widget
widget	basic building block for a GUI in Tk; a window with a particular appearance and behavior
wish	a windowing shell application that includes all of tclsh and the commands defined by Tk

Tcl syntax and structure

Tcl is the control portion of the Tcl/Tk package. Below is a brief description of its syntax and structure.

Variable - can hold numeric, string or list value

```
set a 1
set a "string"
set a "a {2 3}"

set b $a
```

Array - variable with a string index

```
set a(0) "spoon"
set a(fork) "knife"

set b(1) $a(fork)
```

Control structures

```
while <boolean> <body>
if <boolean> [then] <body1> [ else <body2> ]
foreach <variable> <list> <body>
for <loop_init> <boolean> <loop_incr> <body>
switch <flags> <value> <pattern1> <body1> [ <pattern2> <body2> ...]
break
continue
return <string>
```

Variable creation/deletion

```
set <name> <value>
unset <name> [ <name> ... ]
```

Variable manipulation

```
append <name> <value> [ <value> ... ]
incr <name> [ <increment> ]
```

Numeric expression-related commands

```
expr <expression>
```

String-related commands

```
format, regexp, regsub, scan, string, subst
```

Array-related commands

```
array
```

List-related commands

```
concat, join, lappend, lindex, linsert, list,
llength, lrange, lreplace, lsearch, lsort, split
```

File-related commands

```
cd, close, eof, file, flush, gets, glob, open,
puts, pwd, read, seek, tell
```

Tcl function-related commands

```
catch, error, global, proc, rename, return, uplevel, upvar
```

Process-related commands

```
eval, exec, exit, pid, source
```

Miscellaneous commands

```
auto_mkindex, history, info, time, trace, unknown
```

Variables

In Tcl/Tk, words are grouped together using double quotes or curly brackets, depending on when the variables inside the groups are substituted. Variables inside double quotes are substituted; variables inside curly brackets are not substituted.

For example:

```
set a "test"
puts "this is a $a"
```

will print out *this is a test.*

However,

```
set a "test"
puts {this is a $a}
```

will print out *this is a $a.*

You can use this to your advantage when running Tcl code triggered by an event. Because Tcl/Tk code is interpretive, the script is run first, all variables are substituted (except those in brackets), and all widgets are created. When the GUI is displayed, all code at the lowest level is already interpreted, and wish now sits and waits for a user event. When a user performs an action in a window, an event is generated. Some events are captured and cause some branch of Tcl code to be interpreted at that time. For example, when a user presses the left mouse button over a button widget, an event causes the widget's *-command* option to run.

The example below sets the color variable to red. When the button .setit is pressed, the variable is set to blue. The button .colorbutton prints the value of the color variable. Since the value of $color is substituted when this script is run (because the command is in quotes, not brackets), the output is always red, even if the .setit button is pressed before the .colorbutton button.

```
set color red
button .setit -command "set color blue"
button .colorbutton -command "puts $color"
```

The example below outputs the current value of the color variable. Putting the event code in brackets reflects the current state of the variables at the time the button is pressed, instead of the values at startup. So another widget can change the value of a variable, and that value is always current. Pressing the .colorbutton outputs red. Pressing the .setit button, followed by the .colorbutton outputs blue.

```
set color red
button .setit -command "set color blue"
button .colorbutton -command {set swatch $color}
```

All widgets have a defined set of actions that trigger events. Additionally, you can use the *bind* command to add trigger events to a widget definition, allowing different actions to cause events.

Tk syntax

To create a Tk widget, use the command below:

```
widget_type widget_name -option1 value1 -option2 value2 \
       -option3 value3
```

Terms

Each term in the command is defined below.

`widget_type`	one of the valid Tk widget types, for example: label, button, frame
`widget_name`	name of the widget; all references to this widget use this name. The hierarchy of the widget is contained in its name. For example, .frame.button is a widget named button inside the frame .frame. Precede all widget names with a period. Also use the period to separate fields of widgets in a hierarchy.
`-option#`	option name; options can be general to all widgets, for example: -background -foreground -relief. All widget types have options specific to that widget; you can find these on the man pages. Most options have valid synonyms to shorten the option name; for example, -background shortens to -bg.
`value#`	value assigned to the option; relates to the class of widget. A color option like -foreground has valid values of #rXgXbX or a color name.
`\`	continuation character indicating the command is continued on the next line. The command is more readable if the next line is indented from the first.

Examples

The command below makes a frame named *.showme*, which appears raised and has a red background color.

```
frame .showme -relief raised -bg red
```

The command below makes a button named *.showme.now*, implying that *now* is a child of the .showme widget. The button's background is green, the foreground is yellow, the words *Press Me* appear on the button, and, when pressed, the button prints *Now what?* via the Tcl puts command.

```
button .showme.now -command {puts "Now what?"} -bg green -fg yellow \
  -text "Press Me"
```

Error reporting

Syntax errors are reported as they are encountered. Because Tcl/Tk is interpretive, this may occur immediately after the program is run, or as part of a context that will be interpreted when an event is invoked. If the errors occur when the command starts, you will probably see the errors reported in the originating window. If an error occurs when running the GUI, you will get a pop-up window reporting the error, and giving the option to quit or view the stack.

Packing in Tk

Use the *pack* command to manage the widget layout. Widgets will not appear on the screen until they are managed by the geometry manager via a pack. You can arrange the widgets in a frame vertically or horizontally. They can appear in any order, fill the frame, or have space padding them within the frame. If you don't specify the widget's size, it can expand to accommodate the attributes.

Use the steps below to create a frame and then pack it.

1. Make a frame called .a.

```
frame .a -relief raised -borderwidth 4
```

2. Make two buttons called a.b and a.c, which are children of .a.

```
button .a.b -text "button1" -command "puts $junk"
button .a.c -text "button2" -command "puts $test"
```

3. Pack the two buttons horizontally.

```
pack .a.b .a.c -side left
```

4. Pack the buttons vertically.

```
pack .a.b .a.c -side top
```

5. Pack the buttons with a 2 millimeter pad on all sides.

```
pack .a.b .a.c -side left -padx 2m -pady 2m
```

6. Fill the available space in the frame.

```
pack .a.b .a.c -side top -fill x -fill y
```

Using external procs

If you create a proc that will be used in more than one GUI, add that proc to your proc library by running a script containing the following lines:

```
#! /usr/local/bin/wish -f
auto_mkindex . myproc.tcl
exit
```

Or, run *wish* and, at the prompt, enter these two lines:

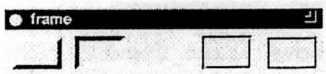
```
auto_mkindex . myproc.tcl
exit
```

This adds a reference to your proc to the file TclIndex in your current directory. When a GUI is run, it will look in that file for the location of the external proc.

Widgets

The widgets used in the McIDAS GUI are described below.

Frame



The frame widget is the GUI's basic building block. Use it to group other widgets and to build nested layouts. In a frame, you can arrange widgets vertically or horizontally.

Possible use: arrange a label widget to the left of an entry widget.

```
frame .a -width 15m -height 10m -relief raised -borderwidth 1.4m
frame .b -width 1.5c -height 1c -relief sunken -borderwidth .14c
frame .c -width 0.59i -height .39i -relief flat -borderwidth .04i
frame .d -width 42p -height 29p -relief groove -borderwidth 3p
frame .e -width 0.59i -height 10m -relief ridge -borderwidth 4
pack .a .b .c .d .e -side left -padx 2m -pady 2m
```

Button

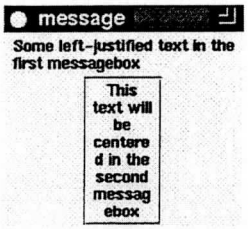
The button widget, also called a command button, is a rectangular box to press to perform an action. The label on the button can be text or a bitmap.

Possible use: bring up a window to query the user when the button is pressed.

```
frame .a -borderwidth 4
pack .a -side bottom

button .a.b1 -text QUIT -command {destroy .}
button .a.b2 -text PRINT \
       -command {puts "Pushed Print button"}
button .a.b3 -text NOTHING
pack .a.b1 .a.b2 .a.b3 -side right
```

Message

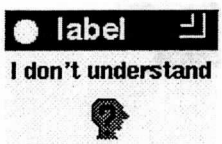


The message widget displays multi-line text strings. Text appears exactly as typed, including spaces and carriage returns. Enter text as one long line so it displays appropriately if the widget is resized.

Possible use: describe the GUI's purpose in a line at the top of the GUI.

```
set text1 "Some left-justified text in the first messagebox"
set text2 "This text will be centered in the second messagebox"
message .a -width 5c -justify left -text $text1
message .b -aspect 50 -justify center -relief groove -text $text2
pack .a .b -side top
```

Label

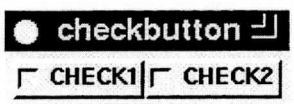


The label widget displays a label, which can be text or a bitmap. No action is performed.

Possible use: display a label opposite an entry widget to describe what the user can type into it.

```
label .a -bitmap questhead
label .b -text "I don't understand"
pack .b .a
```

Checkbutton



The checkbutton widget is a toggle switch for a variable. The variable is set when the button is pressed. Checkbuttons act independently of each other.

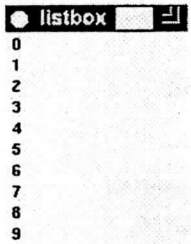
Possible use: describe the attributes of a display where the attributes can be off or on.

```
frame .a -borderwidth 4
pack .a -side bottom

checkbutton .a.c1 -text CHECK1 -variable chk1 \
        -command {puts "chk1=$chk1, chk2=$chk2"}
checkbutton .a.c2 -text CHECK2 -variable chk2 \
        -command {puts "chk1=$chk1, chk2=$chk2"}
pack .a.c1 .a.c2 -side left

puts "chk1=$chk1, chk2=$chk2"
```

Listbox



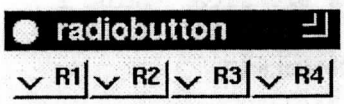
The listbox widget gives the user a list of options. You can configure the widget so users choose one or a range of options, or double-click on a value. Each option in the list is separated by commas and appears on a separate line. To view long lists, tie the list to a vertical scrollbar.

Possible use: give user a list of states to choose from.

```
listbox .a
for {set i 0} {$i < 10} {incr i} {
    .a insert end $i
}
pack .a

bind .a <Double-Button-2> {
    puts [selection get]
}
```

Radiobutton



The radiobutton toggle, when toggled on, toggles another button off. Radiobuttons are grouped and interconnected. Although they are attached to the same variable, each has its own value. When a button is pressed, the variable is assigned the value attached to the button.

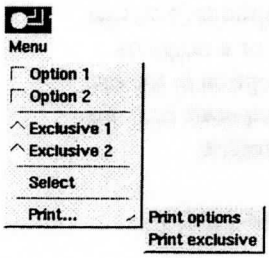
Possible use: query the user on a color attribute, where only one color can be chosen.

```
frame .a -borderwidth 4
pack .a -side bottom

radiobutton .a.r1 -text R1 -variable shared -value r1 \
        -command {puts "shared=$shared"}
radiobutton .a.r2 -text R2 -variable shared -value r2 \
        -command {puts "shared=$shared"}
radiobutton .a.r3 -text R3 -variable shared -value r3 \
        -command {puts "shared=$shared"}
radiobutton .a.r4 -text R4 -variable shared -value r4 \
        -command {puts "shared=$shared"}
pack .a.r1 .a.r2 .a.r3 .a.r4 -side left

puts "shared=$shared"
```

Menubutton



The menubutton widget, when pressed, presents the user with a menu of buttons. This widget supports: radiobuttons, checkbuttons, command buttons, and cascade buttons.

Radiobuttons and checkbuttons are defined in this section. Command buttons run a block of code. When pressed, cascade buttons will cascade down another set of buttons. You can place separators between sets of buttons to logically differentiate subsets. Use a menubutton to implement progressive disclosure.

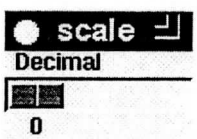
Possible use: let a user choose map boundaries from a predefined set of boundaries or a list of states.

```
menubutton .a -text Menu -menu .a.b
pack .a

menu .a.b
.a.b add checkbutton -label "Option 1" -variable option1
.a.b add checkbutton -label "Option 2" -variable option2
.a.b add separator
.a.b add radiobutton -label "Exclusive 1" -variable exclusive -value 1
.a.b add radiobutton -label "Exclusive 2" -variable exclusive -value 2
.a.b add separator
.a.b add command -label "Select" -command {puts "Selected"}
.a.b add separator
.a.b add cascade -label "Print..." -menu .a.b.c

menu .a.b.c
.a.b.c add command -label "Print options" \
      -command {puts "option1 $option1, option2 $option2"}
.a.b.c add command -label "Print exclusive" \
      -command {puts "exclusive $exclusive"}
```

Scale

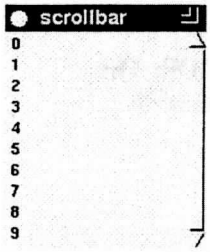


The scale widget, also called a *slider*, is a button that can be moved from end to end to choose a value from a spectrum of values. This widget works well to select from a small, evenly spaced set of numerical values.

Possible use: set the color level on a scale of 0 255.

```
scale .a -label Decimal -orient horizontal -from 0 -to 9 -command dumpScale
pack .a
proc dumpScale value {puts "Value is $value"}
```

Scrollbar



The scrollbar widget is a slider that you can attach to another widget. When the slider on the scrollbar is moved, the contents of the other widget are also scrolled.

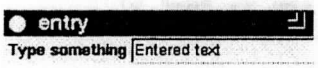
Possible use: attach to a small text window, so the user can scroll with the scrollbar to see more information.

```
listbox .a -yscrollcommand ".b set"
for {set i 0} {$i < 10} {incr i} {
    .a insert end $i
}
pack .a -side left

bind .a <Double-Button-2>{
    puts [selection get]
}

scrollbar .b -command ".a yview"
pack .b -side right -fill y
```

Entry



The entry widget lets the user type and edit a one-line text string. Use the -textvariable option to display a variable in the entry widget. The variable is updated whenever the user edits the entry widget.

Possible use: display a field containing the name of a file to save to disk. The user can edit the file name before running an operation on the file.

```
label .a -text "Type something"
entry .b -textvariable text -relief sunken
bind .b "<Return>" {puts "Entered \"$text\""; .b delete 0 end}
pack .a .b -side left
```

Useful procs

The procs below are ones that we found useful when creating GUIs. The numbers on the left are for reference only and are not part of the code.

Find a User Common value

This proc will find a specific value in User Common.

```
 1.  #
 2.  #       UCval index
 3.  #
 4.  #       get value from User Common at index
 5.  #
 6.  #       Arguments:
 7.  #               index - index of value to obtain
 8.  #
 9.  #               return- value at index in UC
10.  #
11.  proc UCval {index} {
12.
13.         global uc
14.
15.  #      open pipe and run UCU PEEK command
16.         set ucupeek [open "|echo UCU PEEK $index | ucu.mx $uc" r]
17.  #      read output from th command
18.         gets $ucupeek line
19.  #      close the pipe
20.         catch {close $ucupeek}
21.  #      scan output line for value (get 4th value on line
22.         set value [lindex $line 4]
23.  #      return the value.
24.         return $value
25.  }
```

Get today's date

This proc will give you today's Julian date.

```
 1.  #
 2.  #       Today
 3.  #
 4.  #       get todays Julian date from McIDAS
 5.  #
 6.  #       Arguments:
 7.  #
 8.  #               return- todays Julian Date
 9.  #
10.
11.  proc Today {}  {
```

```
12.
13.           global uc
14.
15.   # start up a McIDAS command and set the output pipe to "date"
16.           set date [open "|echo TL Y | tl.mx $uc" r]
17.   # get the first line of output from $date and store in "line"
18.           gets $date line
19.   # close the pipe
20.           catch {close $date}
21.   # pick the second value off the line and set that to the return value
22.           set value [lindex $line 2]
23.   # return $value as the return value
24.           return $value
25.
26.   }
```

Get the current time

To designate the current time, which is the default for the TIME parameter, use the Now proc.

```
 1.   #
 2.   #        Now
 3.   #
 4.   #        Get the current time
 5.   #
 6.   #        Arguments:
 7.   #
 8.   #                return- Current time
 9.   #
10.
11.   proc Now {}  {
12.
13.           global uc
14.
15.   # Start up McIDAS command and pipe data to "date"
16.           set date [open "|echo TL H | tl.mx $uc" r]
17.   # first line of output goes to line
18.           gets $date line
19.   # close the pipe
20.           catch {close $date}
21.   # value gets set to second value in output string
22.           set value [lindex $line 2]
23.   # get the first two digits out of the string (pull the HH out of HHMMSS)
24.           set val [string range $value 0 1]
25.   # return the HH value
26.           return $val
27.
28.   }
```

List the U.S. states

To present the user with a list of U.S. states, tie the command portion of a button to the getstate proc. Getstate sets the variable stateid to the state the user selects.

```
proc getstate {} {

#    If a toplevel window known as states already exists, kill it.
#    This would happen if the user pushed the button twice, we only
#    want one widget.
     catch {destroy .states}

#    the toplevel widget is called .states
     toplevel .states

#    define the title and icon title

     wm title .states "Image Listing"
     wm iconname .states "Images"

#    define the minimum size allowed.
     wm minsize .states 1 1

#    define User Common pointer as global from calling routine - this allows
#    us to access McIDAS commands from outside of the command window
     global uc

#    make a message widget describing this widget.
     message .states.msg -font -Adobe-times-medium-r-normal--*-180* \
         -aspect 300 -relief raised\
             -text "Select Image by double clicking left mouse button.

Press Dismiss to exit. "

#    Make a frame which will hold the listbox information.
     frame .states.frame -borderwidth 10

#    Make a scrollbar so that we can scroll down in the states.
     scrollbar .states.frame.scroll -relief sunken \
             -command ".states.frame.list yview"

#    Make a listbox which will attach the scrollbar to this listbox.
#    The listbox will be 25 characters wide and will display 10 states
#    at a time.
     listbox .states.frame.list -yscroll ".states.frame.scroll set"\
             -relief sunken \
             -geometry 25x10 -setgrid 1

#    pack the scrollbar and the list into the frame
     pack append .states.frame .states.frame.scroll {right filly} \
             .states.frame.list {left expand fill}

#    insert all of the states into the listbox. Each quoted field will be
#    on a separate line.
     .states.frame.list insert 0 "Alabama (AL)" "Alaska (AK)" "Arizona (AZ)"\
     "Arkansas (AR)" "California (CA)" "Colorado (CO)" "Connecticut (CT)" \
     "Delaware (DE)" "Florida (FL)" "Georgia (GA)" "Hawaii (HI)" "Idaho (ID)"\
     "Illinois (IL)" "Indiana (IN)" "Iowa (IA)" "Kansas (KS)" "Kentucky (KY)"\
     "Louisiana (LA)" "Maine (ME)" "Maryland (MD)" "Massachusetts (MA)"\
     "Michigan (MI)" "Minnesota (MN)" "Mississippi (MS)" "Missouri (MO)"\
     "Montana (MT)" "Nebraska (NE)" "Nevada (NV)" "New Hampshire (NH)"\
     "New Jersey (NJ)" "New Mexico (NM)" "New York (NY)" "North Carolina (NC)"\
     "North Dakota (ND)" "Ohio (OH)" "Oklahoma (OK)" "Oregon (OR)"\
     "Pennsylvania (PA)" "Rhode Island (RI)" "South Carolina (SC)" \
```

```
60.       "South Dakota (SD)" "Tennessee (TN)" "Texas (TX)" "Utah (UT)" \
61.       "Vermont (VT)" "Virginia (VI)" "Washington (WA)" "West Virginia (WV)" \
62.       "Wisconsin (WI)" "Wyoming (WY)"
63.  #    bind the double key press to an action
64.       bind .states.frame.list <Double-1> \
65.           {set line [selection get]
66.            set ind [string first "(" $line]
67.            set state [string range $line [expr $ind +1] [expr $ind +2]]
68.                  stateid {set stid $state; liststa}
69.            destroy .states
70.       }
71.
72.  #    Make a button for the users to exit without making a selection
73.       button .states.ok -text Dismiss -command "destroy .states"
74.
75.  #    pack up the message, frame and button
76.       pack .states.msg -side top -fill x
77.       pack .states.frame -side top
78.       pack .states.ok -side bottom
79.  }
```

List all available areas

To prompt the user with a list of available areas, call listarea from a command in a button. This proc lists the areas and sets the variable area to the area the user selects.

```
 1.  proc ListArea {area} {
 2.
 3.  #    destroys any widgets called .listarea that are displayed - in case
 4.  #    the user hit the button twice.
 5.       catch {destroy .listarea}
 6.
 7.  #    we are going to change the bitmap to the hourglass because this can
 8.  #    take a while, and then the user knows that we are working on it.
 9.  #    get the label value, set it to the hourglass, and force to the screen
10.       global blabel
11.       $blabel configure -bitmap @~mcidas/gui0.1/hourglass.xbm
12.       update
13.
14.  #    define the toplevel widget as .listarea
15.       toplevel .listarea
16.
17.  #    define the window title
18.       wm title .listarea "Image Listing"
19.
20.  #    define the icon title
21.       wm iconname .listarea "Images"
22.
23.  #    do not allow resizing past a set size
24.       wm minsize .listarea 1 1
25.
26.  #    allow access to user common
27.       global uc
28.
29.  #    set the message to some instructions for the user
30.       message .listarea.msg -font -Adobe-times-medium-r-normal--*-180* \
31.           -aspect 300 -relief raised\
32.               -text "Select Image by double clicking left mouse button.
33.
34.  Press Dismiss to exit. "
35.
```

```
36.  #    define a frame to hold the output
37.       frame .listarea.frame -borderwidth 10
38.
39.  #    make a scrollbar which will attach to the listbox
40.       scrollbar .listarea.frame.scroll -relief sunken \
41.                 -command ".listarea.frame.list yview"
42.
43.  #    make a listbox to hold the output.  The listbox will be 80 characters
44.  #    wide and 10 lines long.
45.       listbox .listarea.frame.list -yscroll ".listarea.frame.scroll set" \
46.                 -relief sunken \
47.                 -geometry 80x10 -setgrid 1
48.
49.  #    pack the scrollbar and listbox into the frame
50.       pack append .listarea.frame .listarea.frame.scroll {right filly} \
51.                 .listarea.frame.list {left expand fill}
52.
53.  #    initialize localarea
54.       set localarea " "
55.
56.  #    run the McIDAS command routing output to lalist pipe
57.       set lalist [open "|echo LISARA 1 9999 | lisara.mx $uc" r]
58.
59.  #    define a label for above the listbox as a title bar
60.       label .listarea.label1  -width 79
61.
62.  #    read the first line of output (always a title line),
63.  #    and set this to label
64.       gets $lalist lab1
65.       .listarea.label1 configure -text $lab1
66.
67.  #    get the second line which is all underline characters, and toss it
68.       gets $lalist lab2
69.
70.  #    get all the lines and display them in the listbox
71.       while {[gets $lalist line] > -1} {
72.               .listarea.frame.list insert end $line
73.       }
74.
75.  #    set the label back to the SSEC logo and flush to the screen
76.       $blabel configure -bitmap @~mcidas/gui0.1/sseclogo.xbm
77.       update
78.
79.  #    close the pipe
80.       catch {close $lalist}
81.
82.  #    add the action that an item is selected when they double click on
83.  #    the line, and parse the line for the area number
84.       bind .listarea.frame.list <Double-1> \
85.           {set areal [selection get]
86.            set localarea [string range $areal 2 5]
87.            set area $localarea
88.            destroy .listarea
89.       }
90.       button .listarea.ok -text Dismiss -command "destroy .listarea"
91.       pack .listarea.msg -side top -fill x
92.       pack .listarea.label1 -side top -anchor w
93.       pack .listarea.frame -side top
94.       pack .listarea.ok -side bottom
95.  }
```

Set standard options

The setoptions proc sets the default colors for the mandatory entry widgets, the font used in the message section, and any other standard options to the values specified in the gui.options file.

```
proc setoptions { } {


#       name two special SSEC options
#               mand is the color level of mandatory entry widgets
#               messfont is the font of the message widget
        global mand
        global messfont

#       All of the other options from the gui.options file are
#       set in this loop, these will be the defaults for the entire
#       gui, unless a value is set to override them.
        set options [open "gui.options" r]
        while {[gets $options optline] >1} {
                set name [lindex $optline 0]
                set value [lindex $optline 1]
                if {$name == "*mandatoryfield"} {
                        set mand $value
                } else {if {$name == "*messageFont"} {
                        set messfont $value
                        } else {
                        option add $name $value
                        }
                }
        }
}
```

Run a McIDAS command; send the output to a listbox

```
#
#       RouteComm command num_lines_header
#
#       Run a McIDAS command , sending output to a listbox
#
#       Arguments:
#               command - command string to enter
#


proc RouteComm {command headsize} {

#       If there is a .command widget, destroy it.
        catch {destroy .command}

#       access the SSEC Logo and change it to an hourglass, forcing
#       display change to the screen with and "update".
        global blabel
        $blabel configure -bitmap @~mcidas/gui0.1/hourglass.xbm
        update

#       Define the toplevel widget as .command.
        toplevel .command

#       Find the first blank in the string
        set blank [string first " " $command]
```

```
26.  #       Parse out the command name
27.          set name [string range $command 0 [expr $blank - 1 ] ]
28.
29.  #       we need a pointer to User common to run McIDAS command
30.          global uc
31.
32.  #       we need the terminal number in order to write to the text window
33.          global term
34.  #
35.  #       Make the window title the same as the McIDAS command
36.  #       this isn't great because you get windows titled 'SL' which
37.  #       isn't very informative, but we are keeping the proc general.
38.          wm title .command [format "%s Output" $name]
39.          wm minsize .command 1 1
40.
41.  #       make a frame for the listbox and scrollbar
42.          name .command.frame
43.
44.  #       make a scrollbar for scrolling through the output
45.          scrollbar .command.frame.scroll -relief sunken \
46.                  -command ".command.frame.listbox yview"
47.
48.  #       make a listbox to recieve output
49.          listbox .command.frame.listbox \
50.                  -yscroll ".command.frame.scroll set" \
51.                  -relief sunken -geometry 80x15
52.
53.  #       pack the scroll and the list into the frame
54.          pack append .command.frame \
55.                  .command.frame.scroll {right filly} \
56.                  .command.frame.listbox {left expand fill}
57.
58.  #       convert the command name to lowercase
59.          set lowname [string tolower $name]
60.
61.  #       add a ".mx" to make the executable name
62.          set executable [format "%s.mx" $lowname]
63.
64.  #       start up the McIDAS command, routing output to the execout pipe
65.          set execout [open "|echo $command | $executable $uc" r]
66.
67.  #       run through the header lines, and put them into a label
68.          for {set i 0} { $i < $headsize} {incr i} {
69.                  gets $execout lab($i)
70.                  set length [string length $lab($i)]
71.                  label .command.label$i -text $lab($i) -width $length
72.                  pack .command.label$i -side top -anchor w
73.          }
74.  #
75.  #       read through the rest of the pipe and write each line into
76.  #       the listbox
77.          while {[gets $execout line] > -1} {
78.                  .command.frame.listbox insert end $line
79.          }
80.
81.  #       return the label bitmap to SSEC logo
82.          $blabel configure -bitmap @~mcidas/gui0.1/sseclogo.xbm
83.
84.  #       force the screen to display the change
85.          update
86.
87.  #       close the pipe
88.          catch {close $execout}
89.
90.          button .command.dismiss -text Dismiss -command "destroy .command"
91.          pack .command.frame     -side top -fill x
92.          pack .command.dismiss   -side bottom
```

Run a McIDAS command; send the output to the text window

```
#
#       RunCommand command
#
#       Run a McIDAS command , sending output to the text window.
#
#       Arguments:
#               command - command string to enter
#
proc RunCommand {command} {

#       We need uc to fire off the McIDAS command which needs to point
#       to user common since we aren't running from a command line.
        global uc

#       We need the terminal number so that we can route output to the text
#       window
        global term

#       Blabel is the SSEC logo at the bottom of the GUI.  We need it
#       so that we can change it to an hourglass when we run a command.
        global blabel

#       Set up the pipe to the  McIDAS text window
        set MCPIPE [format "McText%sW0" $term]
        set pipeid [open $MCPIPE w]

#       change the bitmap to an hourglass.
        $blabel configure -bitmap @~mcidas/gui0.1/hourglass.xbm

#       force the changes to the screen.
        update
#

#       echo the command to the text window
        puts $pipeid $command flush $pipeid

#       Find the first blank in the command line.
        set blank [string first " " $command]

#       set name to the first character up to the first blank
        set name [string range $command 0 [expr $blank - 1 ] ]

#       Change the name to lowercase
        set lowname [string tolower $name]

#       make the executable name (with a .mx on the command)
        set executable [format "%s.mx" $lowname]

#       fork off an exec of the command
        catch {exec $executable $uc << $command > $MCPIPE}

#       close the pipe.
        catch {close $pipeid}

#       Return the label to the SSEC logo
        $blabel configure -bitmap @~mcidas/gui0.1/sseclogo.xbm

#       flush display changes to the screen.
        update
}

```

Skeleton of a McIDAS GUI

To customize, add widgets inside of .frame and alter the **bold** text to refect your application.

```
#!/usr/local/bin/wish -f

# THIS IS SSEC PROPRIETARY SOFTWARE - ITS USE IS RESTRICTED.

# set path to access library procs
set auto_path "~mcidas/gui0.2 $auto_path"

# set default font and colors from gui.options file
setoptions

# set up to access McIDAS session
set term [lindex $argv 0]
set file [format "UCTERM.00%s" $term]
if {$term > 9} {set file [format "UCTERM.0%s" $term]}
set odline [open "|od -d $file +6" r]
gets $odline line
set uc [lindex $line 1]
close $odline

#
#
#       Skeleton for GUI for McIDAS commands
#
#
#

# set the window title
wm title . "NAME OF APPLICATION HERE"

# Make a frame to contain the help button in the  upper right corner
frame .top
button .top.help -text "Help" -command mkhelp
pack .top.help -side right

# set the message area at the top of the gui.  Give a brief explanation
# of the command.  The font will be the default font from gui.options. The
# width of the ZZ
message .msg -width 500 -relief raised\
-font $messfont -borderwidth 1 -text " *** Brief explanation of the
command***. To execute press OK button. To exit without performing an
action, press Dismiss Button."

# Make a frame which will contain the body of the GUI, place all of
# the widgets here.

frame .frame  -borderwidth 10

# make a frame which will hold the OK and cancel buttons and the SSEC logo
frame .bottom  -relief raised

# a button to execute the command.  All variables will be evaluated as the
# value they are when the button is pressed.

button .bottom.ok -text OK -width 20 \
        -command {

#               set command to the uppercase McIDAS command
#               which you will be executing
                   set command " *****COMMAND****"

#               runcommand executes the command and routes
#               the output to the text window
                   RunCommand $command
                                }
```

```
127.  # a button to cancel the command without executing anything
128.  button .bottom.cancel -text Dismiss -command {destroy .} -width 20
129.
130.  # a ssec logo between the OK and the cancel buttons
131.  label .bottom.label -bitmap @~mcidas/gui0.2/sseclogo.xbm
132.
133.  # set a global variable 'blabel' to the label widget, so that we can
134.  # change the bitmap of the label to an hourglass when the command is
135.  # running.
136.  set blabel .bottom.label
137.
138.  # Pack the OK, & cancel buttons and label into the bottom frame, padding
139.  # 10 pixels around the button (x and y direction), order will be in order
140.  # specified OK, label, cancel (left to right)
141.  pack .bottom.ok -side left -padx 10 -pady 10
142.  pack .bottom.label -side left -padx 10 -pady 10
143.  pack .bottom.cancel -side left -padx 10 -pady 10
144.
145.
146.  #
147.  # Pack the message, the command and the OK buttons into the base widget
148.  #
149.  pack .top -side top -fill x
150.  pack .msg -side top -fill x
151.  pack .frame -side top -fill x
152.  pack .bottom -side bottom -fill x
153.
154.  # set the minimum size for the gui
155.  wm minsize . 10 10
156.
157.
158.  # Proc for help.
159.  proc mkHelp {} {
160.
161.  # name the toplevel widget
162.          toplevel .help
163.
164.  #set the title (change 'command name' to your gui name)
165.          wm title .help "Help - command name"
166.
167.  #set the minimum size for the help window
168.          wm minsize .help 10 10
169.
170.  # make a text widget, to hold the help text, note height should be changed
171.  # to be the number of lines of help text.
172.          text .help.frame -relief raised -height 10
173.
174.  # set the text widget to contain the actual text
175.          .help.frame insert 0.0 {
176.
177.          Put all of the help you want to document between the curly brackets!
178.  }
179.
180.  # make a button to remove the window when pressed
181.          button .help.ok -text "      OK      "\
182.                  -relief raised -borderwidth 2 -command {destroy .help}
183.
184.  # pack the text widget and the OK button into the window
185.          pack .help.frame     -side top
186.          pack .help.ok        -side bottom
187.
188.  }
```

Considerations for building a GUI

It is a good idea whenever possible to make the procs that you build reusable. Just like code in other languages, modularization is a good rule.

Build GUIs with the user in mind. Set reasonable defaults so the user will only have to type or choose minimum information. Don't include rarely used options or keywords from McIDAS commands in your GUI. You may want to make one command into several GUIs if the functionality of the command is complex. Conversely, you may want to combine the functionality of several commands into one GUI. For example, you can include an option to display a map over an image immediately after it is displayed.

Sample GUI code

```
1.  #!/usr/local/bin/wish -f
2.  # The above line needs to be the first line in the script to specify which
3.  # scripting language we will use, and where it's located.
4.
5.  #
6.  #       gu.gui
7.  #
8.
9.
10. # THIS IS SSEC PROPRIETARY SOFTWARE - ITS USE IS RESTRICTED.
11.
12. #
13. #       Set path to point to the ~mcidas/gui0.2 directory as well
14. #       as the Tcl/Tk options set on installation of Tcl/Tk.
15. #
16. set auto_path "~mcidas/gui0.2 $auto_path"
17.
18. #
19. #       call the proc to set the default options for this widget
20. #
21. setoptions
22.
23. #       Set up access to User Common. This is needed to call McIDAS
24. #       commands outside of the command line. The variable $term is
25. #       picked up as the first argument of the call to the GUI.
26. set term [lindex $argv 0]
27.
28. #
29. #       Based on the term number, we choose the file to read.
30. set file [format "UCTERM.00%s" $term]
31. if {$term > 9} {set file [format "UCTERM.0%s" $term]}
32.
33. #       Read the file, which will contain the UC block attached to this
34. #       terminal number, this is set to $uc variable.
35. set odline [open "|od -d $file +6" r]
36. gets $odline line
37. set uc [lindex $line 1]
38. close $odline
39.
40. #
41. #
42. #       GUI for the Graphics utility application
43. #
44. #
45. #       Set the title which appears in the window manager
46. #       frame surrounding the GUI.
47. wm title . "Edit Graphics Colors"
48.
49. #       Set up a button which allows the user to call up the Help window.
50. #       The button will be made in the frame ".top"
51. frame .top
52. button .top.help -text "Help" -command "Help"
53.
54. #
55. #       Pack the help button on the right side of the .top frame.
56. pack .top.help -side right
57.
58. #       define a message area, with a width of 500 pixels. The border
59. #       of the message area will be raised with a width of 1. The font
60. #       was set in the setoptions proc (external).
61. #       The -text field specifies the text to appear inside of the widget.
62. message .msg -width 500 -relief raised\
63. -font $messfont -borderwidth 1 -text "Utility for altering the graphics color table.
```

```
64.
65.  The graphics color table can be modified, saved and restored.
66.
67.  To run press OK button. To exit without performing an action, press Dismiss button."
68.
69.  #       Create a frame which will hold all the command options.
70.  frame .frame  -borderwidth 10
71.
72.  #       Create a frame which will hold the options which the user can select
73.  #       to be prompted with the widgets specific to that option.
74.  frame .frame.option
75.
76.  #       Make a button, which will prompt the user with a series of
77.  #       radio buttons when pressed.
78.  menubutton .frame.option.menu -text "Option" -relief raised \
79.          -menu .frame.option.menu.choices
80.
81.  #       Define the menu which appears when the menubutton is pressed.
82.  menu .frame.option.menu.choices
83.
84.  #       "add" all of the radio buttons to the menu. Define the label which
85.  #       will appear with the button, the variable which will be set when
86.  #       the button is pressed, the value the variable will be set to, and
87.  #       the command which will be performed when the button is pressed.
88.  #       Note, in this menu, when an option is changed, the old option
89.  #       will be removed from the screen with the "pack forget", and the
90.  #       new options widgets will be presented with a "pack".
91.
92.  #       Radio button for "MAKE" option
93.  .frame.option.menu.choices add radio -variable option \
94.          -label "Edit Color" -value MAKE -command "
95.          pack forget .frame.restore
96.          pack forget .frame.save
97.
98.  #       Radio button for "RESTORE" option
99.  .frame.option.menu.choices add radio -variable option \
100.         -label "Restore Table" -value REST -command "
101.         pack forget .frame.save
102.         pack forget .frame.make
103.         pack .frame.restore -side top -fill x
104.
105. #       Radio button for "SAVE" option
106. .frame.option.menu.choices add radio -variable option \
107.         -label "Save Table" -value SAVE -command "
108.         pack forget .frame.restore
109.         pack forget .frame.make
110.         pack .frame.save -side top -fill x
111.
112. #       Radio button for "RESET" option
113. .frame.option.menu.choices add radio -variable option \
114.         -label "Reset" -value RESET  -command "
115.         pack forget .frame.restore
116.         pack forget .frame.make
117.         pack forget .frame.save"
118.
119.
120. #       define an entry widget which will hold the value which was
121. #       set by the radiobutton chosen above.  Note the "textvariable"
122. #       is the same as the "variable" in the radiobuttons above.
123. #       The current value of the variable will be displayed in the
124. #       entry widget, even as the variable changes values by the menu
125. #       above. The widget will be sunken and will be 5 characters wide.
126. #
127. entry .frame.option.entry -relief sunken -width 5 -textvariable option
128.
129. #       Pack the menubutton and entry button left to right
130. pack .frame.option.menu -side left
131. pack .frame.option.entry -side right
132.
133. #       Pack the option frame into ".frame".
134. pack .frame.option -side top -fill x
```

```
135.  #       Restore option - consists of a button, and an entry widget.
136.  #       Note that this frame is not packed at this time.  It will not
137.  #       be packed until the RESTORE option is selected on the radio
138.  #       buttons above.
139.
140.  frame .frame.restore
141.
142.  #       Button which when pressed runs the proc "Getgra". The text
143.  #       is "Graphics file name".
144.  button .frame.restore.button -text "Graphics File Name" -command Getgra
145.
146.  #       Entry widget to show the value selected by the user in getgra
147.  #       proc. The variable associated with this entry is gratab.
148.  entry .frame.restore.entry -textvariable gratab -relief sunken -width 12
149.
150.  #       Pack the button on the left and the entry on the right.
151.  pack .frame.restore.button -side left
152.  pack .frame.restore.entry -side right
153.
154.  # save option
155.
156.  frame .frame.save
157.  label .frame.save.label -text "Graphics File Name"
158.  entry .frame.save.entry -textvariable gratab -relief sunken -width 12
159.  pack .frame.save.label -side left
160.  pack .frame.save.entry -side right
161.
162   #       make option
163.
164.  #       set the variables red, green, blue and clev to zero.
165.  set red 0
166.  set green 0
167.  set blue 0
168.  set clev 0
169.
170.  #       Create a frame for the MAKE option.
171.  frame .frame.make
172.
173.  #       Create a color frame within the make option.
174.  frame .frame.make.color
175.
176.  #       Create a button which when pressed will run the getcolor proc.
177.  button .frame.make.color.button -text "Color Level" -command "getcolor "
178.
179.  #       Create an entry widget which will display the results of the
180.  #       getcolor proc.
181.  entry .frame.make.color.entry -textvariable clev -relief sunken -width 3
182.
183.  #       red, green and blue (which are dereferenced with a $), were
184.  #       returned from the getcolor proc.  The values are put into
185.  #       a format which can be used to specify a color for the
186.  #       swatch below.
187.  set color [format "#%02x%02x%02x" $red $green $blue]
188.
189.  #       Make a frame which is nothing more than a color swatch - a
190.  #       rectangle of specified color.
191.  frame .frame.make.color.swatch -width 200 -height 20 -bg $color
192.
193.  #       Pack the button on the left, the swatch to the right of it, and
194.  #       the entry widget on the right.
195.  pack .frame.make.color.button -side left
196.  pack .frame.make.color.swatch -side left -padx 20
197.  pack .frame.make.color.entry -side right
198.
199.  #       Pack the color frame into the make frame.
200.  pack .frame.make.color -side top -fill x
201.
202.  #       Now we make a frame which will contain radio buttons allowing
203.  #       the user to specify how they will select the new color (by
204.  #       name or by sliders)
```

```
frame .frame.make.how

#       Make a radiobutton which allows the user to select a named color.
#       variable is "how", value will be set to "byname" if chosen, the
#       label associated with the button is "Select Named Color", and
#       when radiobutton is selected the "name" frame will be displayed.
#       Since the users are specifying a named color, the red, green,
#       and blue sliders will not be needed, and thus will be "unpacked".

radiobutton .frame.make.how.byname -variable how -value byname \
        -text "Select Named Color" -command {
                pack .frame.make.name -side top -fill x
                pack forget .frame.make.red
                pack forget .frame.make.green
                pack forget .frame.make.blue
                }

#       Now, make one to select the color by intensity levels.
#       Here the the variable will be set to "byval", and
#       the "name" frame will be "unpacked", as the red, blue
#       and green frames are brought into view.  We also set the
#       the values of the variable associated with the sliders
#       to the values of the chosen color level to start.

radiobutton .frame.make.how.byval -variable how -value byval \
        -text "Select Color Intensities" -command {
                pack forget .frame.make.name
                pack .frame.make.red -side top -fill x
                pack .frame.make.green -side top -fill x
                pack .frame.make.blue -side top -fill x
                .frame.make.red set $red
                .frame.make.green set $green
                .frame.make.blue set $blue
                }

#       Pack byname and byval side by side.
pack .frame.make.how.byname -side left
pack .frame.make.how.byval -side left

#       Pack the how frame into the MAKE frame.
pack .frame.make.how -side top -fill x -pady 10


#       Frame which will contain the color name to change the level to.
frame .frame.make.name

#       Button which runs GetNamedColor proc
button .frame.make.name.button -text "Color Name" -command GetNamedColor

#       Entry widget which contains the value set in the GetNamedColor
#       proc to the colorname variable.
entry .frame.make.name.entry -textvariable colorname -width 12 -relief sunken

#       Pack the button and entry widgets side by side.
pack .frame.make.name.button -side left
pack .frame.make.name.entry -side right

#       Make a scale to set the red pigment of the color.
#       The scale will be horizontally oriented, will contain values
#       from 0 to 255, and will be 256 pixels long. The value will be
#       displayed as it changes, and the color of the slider is red.
#       As the slider is moved, the SetColor proc will be called to
#       update the color of the swatch with the changing values.
#       When a command is called from a scale widget, the current value
#       of the widget is appended to the proc call, so the proc actually
#       has 2 arguments, although only one is specified in the -command
#       option. That value is not actually used in the proc, but
#       because it will appear in the calling sequence, it must be
#       enumerated in the argument list. This is a subtle trick of
#       Tk to be aware of!
```

```
scale .frame.make.red -from 0 -to 255 -label "Red Intensity" -length 256\
        -showvalue YES -orient horizontal -sliderforeground red\
        -command {SetColor "num"}

#       Green scale, which is largely the same as the red scale.
scale .frame.make.green -from 0 -to 255 -label "Green Intensity"\
        -length 256 -showvalue YES -orient horizontal\
        -sliderforeground green -command {SetColor "num"}

#       Blue scale, just like the red and green one - only it's blue!
scale .frame.make.blue -from 0 -to 255 -label "Blue Intensity" -length 256\
        -showvalue YES -orient horizontal -sliderforeground blue\
        -command {SetColor "num"}

#       Set the values of the sliders to the red, green and blue values
#       which are associated with the color level, this will cause the
#       the slider to move to this value.
.frame.make.red set $red
.frame.make.green set $green
.frame.make.blue set $blue

#       This frame will be the bottom portion of the gui. It will hold
#       the OK, and cancel buttons, and a bitmap of the SSEC logo.
frame .bottom  -relief raised

#       OK button executes commands based on option.
#       because the command is in {}s it will be evaluated when the
#       button is pressed instead of when the GUI is started.
#       The command is built based on the option which was chosen
#       by the user.

button .bottom.ok -text OK -width 20 \
        -command {
                    if {$option == "REST"} {set command "GU REST $gratab"}
                    if {$option == "SAVE"} {set command "GU SAVE $gratab"}
                    if {$option == "RESET"} {set command "GU REST"}
                    if {$option == "MAKE"} {
                        if {$show == "byname"} {
                                set col $colorname}
                        if {$show == "byval"} {
                                set col [format "%s %s %s" $blue $green $red]}
                        set command "GU MAKE $clev $col"
                    }

#       This is a proc we created to execute a McIDAS command.
                     RunCommand $command

                  }

#       cancel button in case user decides to bail out, it will destroy
#       the window.

button .bottom.cancel -text Dismiss -command {destroy .} -width 20

#       label containing the SSEC logo.  It will change to an hourglass
#       bitmap when RunCommand is executed. RunCommand uses the
#       variable blabel to change the value of the bitmap.

label .bottom.label -bitmap @~mcidas/gui0.2/sseclogo.xbm
set blabel .bottom.label

#       Pack the ok button, bitmap label, and cancel buttons side by side,
#       with a 10 pixel padding on all sides.

pack .bottom.ok -side left -padx 10 -pady 10
pack .bottom.label -side left -padx 10 -pady 10
pack .bottom.cancel -side right -padx 10 -pady 10

#       Pack the top of the frame into the top of the main window.
#       The frame will be expanded to fit into the window.
```

```
345.  pack .top -side top -fill x
346.
347.  #       Next, pack the message, expanding to fit
348.  pack .msg -side top -fill x
349.
350.  #       Now pack the frame which contains the command widgets.
351.  pack .frame -side top -fill x
352.
353.  #       OK, now we can pack the bottom frame of the widget
354.  pack .bottom -side bottom -fill x
355.
356.  #       Set the minimum dimensions allowd for the underlying window
357.  #       during interactive resizing.
358.  wm minsize . 10 10
359.
360.  # Help proc - display the help associated with this GUI
361.
362.  proc Help {} {
363.
364.
365.  #       define the toplevel window
366.          toplevel .help
367.
368.  #       Set the title bar
369.          wm title .help "Help - Edit Graphics Colors"
370.
371.  #       set the minimum size
372.          wm minsize .help 10 10
373.
374.  #       create a text widget.  The border is raised, it is 16 lines long,
375.  #       the body of the widget is inserted in another step.
376.  #
377.          text .help.text -relief raised -height 16
378.
379.  #       insert the lines of text
380.          .help.text insert 0.0 {
381.
382.  Option          Select the option to perform on the graphics color table
383.
384.          Edit Color      Select the color to be edited, and the color name,
385.                          or color intensities to set it to.
386.
387.          Save Table      Save the current frames graphics color table to a
388.                          named file.
389.
390.          Restore Table   Restore a previously stored graphics color table
391.                          to the current frame.
392.
393.          Reset           Reset the color table to the standard color table.
394.  }
395.
396.  #       make a button which will dismiss the window
397.          button .help.ok -text "      OK      " \
398.                  -relief raised -borderwidth 2 -command {destroy .help}
399.
400.  #       Pack the text and OK frame into the toplevel frame.
401.          pack .help.text -side top
402.          pack .help.ok   -side bottom
403.
404.  }
405.
406.  #       Getgra lists graphics files which can be selected for restoring
407.  #       the graphics table
408.
409.  proc Getgra {} {
410.
411.  #       gratab is a global variable, it is also the variable attacted
412.  #       to an entry widget above.  It must be a global variable to be
413.  #       updated in the entry widget of another frame when it changes
414.  #       within a proc.
```

```
415.          global gratab
416.
417.  #       If there is already a window called .listgra - destroy it.
418.          catch {destroy .listgra}
419.
420.  #       define the toplevel widget
421.          toplevel .listgra
422.
423.  #       title, icon name and minimum size of the window created inside
424.  #       this proc
425.          wm title .listgra "Graphics Files Listing"
426.          wm iconname .listgra "Graphics"
427.          wm minsize .listgra 1 1
428.
429.  #       uc needs to be global, so we can access it here, we need it
430.  #       because we'll be running a McIDAS command.
431.
432.          global uc
433.          message .listgra.msg -font -Adobe-times-medium-r-normal--*-180* \
434.          -aspect 300 -relief raised\
435.            -text "Select Graphics File by double clicking left mouse button.
436.
437.  Press Dismiss to exit. "
438.
439.  #       Make a frame which will contain the listbox and scrollbar
440.          frame .listgra.frame -borderwidth 10
441.
442.  #       Make a scrollbar.  The command is to change the view of
443.  #       the listbox
444.          scrollbar .listgra.frame.scroll \
445.                  -relief sunken -command ".listgra.frame.list yview"
446.
447.  #       Make a listbox.  Attach the scrollbar to it.  The listbox will
448.  #       appear sunken, and will be 80x10 characters.
449.          listbox .listgra.frame.list -yscroll ".listgra.frame.scroll set" \
450.          -relief sunken -geometry 80x10 -setgrid 1
451.
452.          pack .listgra.frame.scroll -side right -fill y
453.          pack .listgra.frame.list -side left -expand yes -fill y
454.
455.  #       set a temporary variable to " ", so Tcl knows it's a character value
456.          set temp " "
457.
458.  #       Fire off a "GU LIST" command to McIDAS.
459.          set gralist [open "|echo GU LIST | gu.mx $uc" r]
460.
461.  #       grab the first line off the output to make a title bar
462.          global lab1
463.          label .listgra.label1  -width 79
464.          gets $gralist lab1
465.
466.  #       Grab the second line off the output and toss it.
467.          .listgra.label1 configure -text $lab1
468.          gets $gralist lab2
469.
470.  #       Get each line, of output, and insert it into the listbox.
471.          while {[gets $gralist line] > -1} {
472.                  .listgra.frame.list insert end $line
473.          }
474.
475.  #       Close the pipe for the McIDAS - failure to do so will leave
476.  #       defunct processes around.
477.          catch {close $gralist}
478.
479.  #
480.  #       Set the action within {} to what occurs when a double click of
481.  #       the button takes place.  Within {}s we set "thisval" to the
482.  #       line which was selected, we grab characters 4-15 of that line
483.  #       we set the global variable gratab to this value, and it will
484.  #       appear in the entry widget.  Then we destroy the widget.
```

```
485.          bind .listgra.frame.list <Double-1> \
486.                  {set thisval  [selection get]
487.                  set temp [string range $thisval 4 15]
488.                  set gratab $temp
489.                  destroy .listgra
490.          }
491.
492.  #       We make a button for the user to exit without making a selection.
493.          button .listgra.ok -text Dismiss -command "destroy .listgra"
494.
495.  #       Pack the message,  title bar, widget frame, and OK button into
496.  #       the toplevel window top to bottom.
497.          pack .listgra.msg -side top -fill x
498.          pack .listgra.label1 -side top -anchor w
499.          pack .listgra.frame -side top
500.          pack .listgra.ok -side bottom
501.
502.  }
503.
504.  #       Works exactly as does the getgra proc above, but doesn't execute
505.  #       a McIDAS command. Instead it reads in McIDAS MCRGB.TXT file.
506.
507.  proc GetNamedColor {} {
508.          catch {destroy .colors}
509.          toplevel .colors
510.          wm title .colors "Image Listing"
511.          wm iconname .colors "Images"
512.          wm minsize .colors 1 1
513.          global uc
514.          message .colors.msg -font -Adobe-times-medium-r-normal--*-180* \
515.                  -aspect 300 -relief raised\
516.                  -text "Select Image by double clicking left mouse button.
517.
518.  Press Dismiss to exit. "
519.          frame .colors.frame -borderwidth 10
520.          scrollbar .colors.frame.scroll -relief sunken \
521.                  -command ".colors.frame.list yview"
522.          listbox .colors.frame.list -yscroll ".colors.frame.scroll set" \
523.                  -relief sunken -geometry 20x10 -setgrid 1
524.          pack .colors.frame.scroll -side right -fill y
525.          pack .colors.frame.list -side left -expand yes -fill y
526.          set collist [open "MCRGB.TXT" r]
527.
528.  #       These variables need to be global because we access this info
529.  #       in the main GUI.
530.          global i
531.          global reds
532.          global greens
533.          global names
534.          global blues
535.          set i 0
536.
537.  #       Read in the contents of MCRGB.TEXT (collist), and store the
538.  #       values in the names, reds, greens, and blues array
539.          while {[gets $collist line] > -1} {
540.                  .colors.frame.list insert end [lindex $line 3]
541.                  set names($i)  [lindex $line 3]
542.                  set reds($i) [lindex $line 0]
543.                  set greens($i) [lindex $line 1]
544.                  set blues($i) [lindex $line 2]
545.                  incr i
546.          }
547.
548.  #       close the pipe
549.          catch {close $collist}
550.          global colorname
551.          global lowname
552.
553.  #       set a double click event to do the following between {}.
554.  #       Events within {} include getting, the line, getting the
```

```
555.  #       colorname from the line, making it lowercase, and calling
556.  #       SetColor proc to change the swatch to the selected color.
557.          bind .colors.frame.list <Double-1> \
558.                  {set color [selection get]
559.                  set cname [lindex $color 0]
560.                  set colorname [string trim $cname]
561.                  set lowname [string tolower $colorname]
562.                  SetColor "name" 0
563.                  destroy .colors
564.          }
565.
566.  #       Pack all of the widgets into the toplevel window.
567.          button .colors.ok -text Dismiss -command "destroy .colors"
568.          pack .colors.msg -side top -fill x
569.          pack .colors.frame -side top
570.          pack .colors.ok -side bottom
571.  }
572.
573.
574.  #       The SetColor proc sets the color of the swatch in the MAKE
575.  #       option to the color chosen by the user
576.  proc SetColor {how junk} {
577.
578.  #       We need access to all of these variables set in other procs.
579.          global lowname
580.          global colorname
581.          global red
582.          global green
583.          global blue
584.          global i
585.          global reds
586.          global greens
587.          global blues
588.          set j 0
589.
590.  #       If they chose a named color, figure out which color they picked,
591.  #       and set red, green, and blue to that value.
592.          if {$how == "name"} {
593.                  while { $j < $i} {
594.                          global names
595.                          if {$colorname == $names($j)}{
596.                                  set red $reds($j)
597.                                  set green $greens($j)
598.                                  set blue $blues($j)
599.                          }
600.                          incr j
601.                  }
602.
603.  #       Set color variable to format #RRGGBB where RR GG BB are the
604.  #       hex representation of the Red Green and Blue color levels.
605.
606.                  set color [format "#%02x%02x%02x" $red $green $blue]
607.
608.  #       Set the swatch to chosen color
609.                  .frame.make.color.swatch config -bg $color
610.
611.  #       If they chose a color by color levels, get those values off of
612.  #       the scale widgets
613.          } else {
614.                  set red [.frame.make.red get]
615.                  set blue [.frame.make.blue get]
616.                  set green [.frame.make.green get]
617.
618.  #       Set color variable to format #RRGGBB where RR GG BB are the
619.  #       hex representation of the Red Green and Blue color levels.
620.                  set color [format "#%02x%02x%02x" $red $green $blue]
621.
622.  #       Set the swatch to chosen color
623.                  .frame.make.color.swatch config -bg $color
624.          }
```

```
}

proc getcolor {} {
#
#               getcol - display McIDAS graphics colors, allowing the
#                        the user to choose one for their application.
#

#       These are the global variables we set or use inside of this proc.
        global uc
        global lev
        global clev
        global red
        global green
        global blue

#       If there is already a colors widget out there - destroy it.
        catch {destroy .colors}
        toplevel .colors
        wm title .colors "Color "
        wm minsize .colors  1 1
        message .colors.msg -font -Adobe-times-medium-r-normal--*-180* \
                -relief sunken \
                -text "Select Color Level by clicking on the color"

#       Make a frame which will hold the buttons.
        frame .colors.frame

#       Start up a McIDAS command to list out the current RGB levels
#       of the graphics color levels.
        set guout [open "|echo GU TAB | gu.mx $uc" r]

#       Retrieve the number of Graphics levels of this session from
#       user common.
        set maxlev [UCval 500]

#       Go through all of the output from the GU output.
        while {[gets $guout guline] > -1} {

#       Determine which color level this is.
                set level [string range $guline 3 4]
                if {$level <= 9} { set level [string range $guline 4 4]}

#       grab the RGB values for this level
                set ired [string range $guline 35 37]
                set igreen [string range $guline 30 32]
                set iblue [string range $guline 24 26]

#       make sure its a good level
                if {$level >= 0 && $level <= $maxlev} {

#       Set rgb variable to #RRGGBB format
                        set rgb [format "#%02x%02x%02x" $ired $igreen $iblue]

#       Make a frame for each level - name it lev1, lev2, lev3, etc.
                        frame .colors.frame.lev$level

#       Make a button which has a bitmap label which is a box, the color
#       of the RGB levels we retrieved for this level. The $level, ired,
#       igreen, and iblue will be dereferenced now.  The swatch will
#       be set to this RGB level when this button is pressed.
                        button .colors.frame.lev$level.button \
                                -bitmap @~mcidas/gui0.2/box.xbm \
                                -foreground $rgb \
                                -command "set lev $level ;set clev $level;
                                        set red $ired
                                        set green $igreen
                                        set blue $iblue
                .frame.make.color.swatch config -bg $rgb
                                        destroy .colors"\
```

```
695.                                    -activebackground gray\
696.                                    -activeforeground $rgb
697.  #       Make a label which has the level number as the text
698.                          label .colors.frame.lev$level.label -text $level
699.
700.  #       Pack the button, and label into each levels frame
701.                          pack append .colors.frame.lev$level\
702.                              .colors.frame.lev$level.label {left}\
703.                              .colors.frame.lev$level.button {right}
704.
705.  #       Pack each levels frame into the large frame
706.                          pack append .colors.frame \
707.                              .colors.frame.lev$level{top fill}
708.                  }
709.          }
710.
711.  #       close the pipe
712.          catch {close $guout}
713.
714.  #       button to dismiss the GUI without taking action.
715.          button .colors.dismiss -command "destroy .colors" -text "Dismiss"
716.          pack append .colors \
717.                  .colors.msg     {top fill} \
718.                  .colors.frame   {top fill} \
719.                  .colors.dismiss {top fill}
720.  }
```

Solution set

Below is a solution set for the in-class assignment.

```
 0: #!/usr/local/bin/wish -f
 1:
 2: # THIS IS SSEC PROPRIETARY SOFTWARE - ITS USE IS RESTRICTED.
 3:
 4: # set path to access library procs
 5: set auto_path " ~/mcidas/data ~mcidas/gui0.2 $auto_path"
 6:
 7: # set default font and colors from gui.options file
 8: setoptions
 9:
10: # set up to access McIDAS session
11: set term [lindex $argv 0]
12: set file [format "UCTERM.00%s" $term]
13: if {$term > 9} {set file [format "UCTERM.0%s" $term]}
14: set odline [open "|od -d $file +6" r]
15: gets $odline line
16: set uc [lindex $line 1]
17: close $odline
18:
19: #
20: #
21: #       GUI for MUG demo to run MUGAREA command
22: #
23: #
24: #
25:
26: # set the window title
27: wm title . "MUG Demo Command"
28:
29: # Make a frame to contain the help button in the  upper right corner
30: frame .top
31: button .top.help -text "Help" -command mkHelp
32: pack .top.help -side right
33:
34:
35: # set the message area at the top of the gui.  Give a brief
36: # explanation of the command.  The font will be the default font
37: # from gui.options. The width of the ZZ
38: message .msg -width 500 -relief raised\
39: -font $messfont -borderwidth 1 -text "Retrieve 2 McIDAS GRIDS in an
40: AREA format via ADDE, and combine them with arithmetic function,
41: sending result to a McIDAS AREA formatted file. To execute press OK
42: button. To exit without performing an action, press Dismiss Button."
43:
44: # Make a frame which will contain the body of the GUI, place all of
45: # the widgets here.
46:
47: frame .frame  -borderwidth 10
48:
49:
50: # set the default source dataset name
51: set source_ds "MUG/MRFT"
52:
53: # frame to hold the source dataset name widgets
54: frame .frame.source
55:
56: #label describing the entry widget value to the right
57: label .frame.source.label -text "Source Dataset Name"
58:
59: #entry widget holding the source dataset name
60: entry .frame.source.entry -textvariable source_ds -width 20 -relief sunken
```

```
61:
62: #pack widgets left and right
63: pack .frame.source.label -side left
64: pack .frame.source.entry -side right
65:
66: #pack the source dataset frame into main frame
67: pack .frame.source -side top -fill x
68:
69:
70: # frame to hold the position widgets
71: frame .frame.pos1
72:
73: # button which will run the  list_adde_image proc
74: button .frame.pos1.button -text "Source Position 1"\
75:         -command {set which first;list_adde_image}
76:
77: # entry which will hold the results of the position selection
78: entry .frame.pos1.entry -textvariable pos1 -width 5 -relief sunken
79:
80: #pack the widgets left and right
81: pack .frame.pos1.button -side left
82: pack .frame.pos1.entry -side right
83:
84: #install the position 1 frame in the GUI
85: pack .frame.pos1 -side top -fill x
86:
87: frame .frame.pos2
88:
89: # frame to hold the position widgets
90:
91:
92: # button which will run the  list_adde_image proc
93: button .frame.pos2.button -text "Source Position 2"\
94:         -command {set which second;list_adde_image}
95:
96: # entry which will hold the results of the position selection
97: entry .frame.pos2.entry -textvariable pos2 -width 5 -relief sunken
98:
99: #pack the widgets left and right
100: pack .frame.pos2.button -side left
101: pack .frame.pos2.entry -side right
102:
103: #install the position 2 frame in the GUI
104: pack .frame.pos2 -side top -fill x
105:
106: # frame to hold the options
107: frame .frame.option
108:
109: # radiobutton with text of ADD which will set variable option to ADD
110: radiobutton .frame.option.add -text "Add" \
111:         -variable option -value ADD -width 15
112:
113: # radiobutton with text of Subtract which will set variable option to SUBTRACT
114: radiobutton .frame.option.subtract -text "Subtract"\
115:         -variable option -value SUBTRACT -width 15
116:
117: # radiobutton with text of Average which will set variable option to AVERAGE
118: radiobutton .frame.option.average -text "Average" \
119:         -variable option -value AVERAGE -width 15
120:
121: # pack all of the radiobuttons onto a line
122: pack .frame.option.add .frame.option.subtract  \
123:         .frame.option.average -side left
124:
125: # install the options on the GUI
126: pack .frame.option -side top -fill x
127:
128: # frame for destination dataset name
129: frame .frame.dest
130:
131: # label describing entry field on the right
132: label .frame.dest.label -text "Destination Dataset Name"
```

```
133:
134: # entry field holding the value of the destination dataset name
135: entry .frame.dest.entry -textvariable dest -width 20 -relief sunken
136:
137: # pack the label and entry, left to right
138: pack .frame.dest.label -side left
139: pack .frame.dest.entry -side right
140:
141: # install the destination name widgets
142: pack .frame.dest -side top -fill x
143:
144:
145: # frame for destination position number
146: frame .frame.dpos
147:
148: # label describing entry
149: label .frame.dpos.label -text "Destination Position"
150:
151: # entry holding value of the destination position
152: entry .frame.dpos.entry -textvariable dpos -width 5 -relief sunken
153:
154: # pack the label & entry left and right
155: pack .frame.dpos.label -side left
156: pack .frame.dpos.entry -side right
157:
158: # install the destination position number widgets
159: pack .frame.dpos -side top -fill x
160:
161:
162: # make a frame which will hold the OK and cancel buttons and the SSEC logo
163: frame .bottom  -relief raised
164:
165: # a button to execute the command.  All variables will be evaluated as the
166: #value they are when the button is pressed.
167:
168: button .bottom.ok -text OK -width 20 \
169:         -command {
170:
171: #                set command to the uppercase McIDAS command
172: #                which you will be executing
173:                  set dsname [format "%s.%s" $dest $dpos]
174:                  set command "MUGAREA $source_ds $pos1 $option $pos2 $dsname"
175:
176: #                runcommand executes the command and routes
177: #                the output to the text window
178:                     RunCommand $command
179:                                }
180:
181: # a button to cancel the command without executing anything
182: button .bottom.cancel -text Dismiss -command {destroy .} -width 20
183:
184: # a ssec logo between the OK and the cancel buttons
185: label .bottom.label -bitmap @~mcidas/gui0.2/sseclogo.xbm
186:
187: # set a global variable 'blabel' to the label widget, so that we can
188: # change the bitmap of the label to an hourglass when the command is
189: # running.
190: set blabel .bottom.label
191:
192: # Pack the OK, & cancel buttons and label into the bottom frame,
193: # padding 10 pixels around the button (x and y direction), order
194: # will be in order specified OK, label, cancel (left to right)
195:
196: pack .bottom.ok -side left -padx 10 -pady 10
197: pack .bottom.label -side left -padx 10 -pady 10
198: pack .bottom.cancel -side left -padx 10 -pady 10
199:
200:
201: #
202: # Pack the message, command and OK buttons into the base widget
203: #
204: pack .top -side top -fill x
```

```
205: pack .msg -side top -fill x
206: pack .frame -side top -fill x
207: pack .bottom -side bottom -fill x
208:
209: # set the minimum size for the gui
210: wm minsize . 10 10
211:
212:
213: # Proc for help.
214: proc mkHelp {} {
215:
216: # name the toplevel widget
217:         toplevel .help
218:
219: #set the title (change 'command name' to your gui name)
220:         wm title .help "Help - command name"
221:
222: #set the minimum size for the help window
223:         wm minsize .help 10 10
224:
225: # make a text widget, to hold the help text, note height should
226: # be changed to be the number of lines of help text.
227:         text .help.frame -relief raised -height 15
228:
229: # set the text widget to contain the actual text
230:         .help.frame insert 0.0 {
231:
232: Source Dataset Name      Name of source dataset which contains grids
233:                          served as ADDE images
234:
235: Source Position 1        position of one of the grids to perform option on
236:
237: Source Position 2        position of other grids to perform option on
238:
239: Option                   Add, Subtract or Average option
240:
241: Destination Dataset name Name of dataset to write resultant image to
242:
243: Destination Position     Position of destination image in dataset
244: }
245:
246: # make a button to remove the window when pressed
247:         button .help.ok -text "      OK      " -relief raised \
248:                 -borderwidth 2 -command {destroy .help}
249:
250: # pack the text widget and the OK button into the window
251:         pack .help.frame -side top
252:         pack .help.ok  -side bottom
253:
254: }
255:
256:
257:
258:
259: # mkListArea w
260: #
261: # Create a top-level window containing a listbox showing a bunch of
262: # McIDAS Images
263: #
264: # Arguments:
265: #    w -Name to use for new top-level window.
266:
267:
268: # THIS IS SSEC PROPRIETARY SOFTWARE - ITS USE IS RESTRICTED.
269:
270: proc list_adde_image {} {
271:
272: # destroy a .listarea widget if it exists
273:     catch {destroy .listarea}
274:
275: #   label of the main widget, set it to the hourglass while we get
276: #   listing
```

```
277:     global blabel
278:     $blabel configure -bitmap @~mcidas/gui0.1/hourglass.xbm
279:     update
280:
281:
282: # name the toplevel widget .listarea
283:     toplevel .listarea
284:
285: #  set the characteristics of the window
286:     wm title .listarea "Image Listing"
287:     wm iconname .listarea "Images"
288:     wm minsize .listarea 1 1
289:
290: # bring in uc and which as global variables - we'll use them later
291:     global uc
292:     global which
293:
294: # make a message widget to hold the instructions
295:     message .listarea.msg -font -Adobe-times-medium-r-normal--*-180* \
296:         -aspect 300 -relief raised\
297:             -text "Select Image by double clicking left mouse button.
298:         Press Dismiss to exit. "
299:
300: # make a frame to hold the workings of the widget
301:         frame .listarea.frame -borderwidth 10
302:
303: # make a scrollbar to allow scrolling of the listing
304:         scrollbar .listarea.frame.scroll -relief sunken \
305:                 -command ".listarea.frame.list yview"
306:
307: # make a listbox to hold the image listings
308:         listbox .listarea.frame.list -yscroll ".listarea.frame.scroll set" \
309:                 -relief sunken \
310:                 -geometry 80x10 -setgrid 1
311:
312: # pack the scrollbar on the right of the listbox
313:         pack append .listarea.frame .listarea.frame.scroll {right filly} \
314:                 .listarea.frame.list {left expand fill}
315:
316:         set localarea " "
317:
318: # set lalist to be the pipe from the IMGLIST command
319:         set lalist [open "|echo IMGLIST MUG/MRFT.ALL | imglist.mx $uc" r]
320:
321: # set the label to the first line of output
322:         global lab1
323:         label .listarea.label1  -width 79
324:         gets $lalist lab1
325:         .listarea.label1 configure -text $lab1
326:
327: # get the next 3 lines of output and discard them
328:         gets $lalist lab2
329:         gets $lalist lab2
330:         gets $lalist lab2
331:
332: # retrieve all of the rest of the lines and display them in the listbox
333:         while {[gets $lalist line] > -1} {
334:                 .listarea.frame.list insert end $line
335:         }
336:
337: # set the label to the SSEC logo again
338:         $blabel configure -bitmap @~mcidas/gui0.1/sseclogo.xbm
339:         update
340:
341: # close the pipe
342:         catch {close $lalist}
343:
344: # bind the double left button click
345:         bind .listarea.frame.list <Double-1> \
346:
347: # get the selected line
348:                 {set areal [selection get]
```

```
349:
350: #set  a variable to position 2 - 5 of the line
351:                 set localarea [string range $areal 2 5]
352:
353: # set pos 1 if they are to set the first position
354:                 if { $which == "first" } {
355:                         set pos1 $localarea
356:                 } else {
357:                         set pos2 $localarea
358:                 }
359:                 destroy .listarea
360:         }
361:
362: # make a button to get rid of the widget
363:         button .listarea.ok -text Dismiss -command "destroy .listarea"
364:
365: # pack the widgets into the window
366:         pack .listarea.msg -side top -fill x
367:         pack .listarea.label1 -side top -anchor w
368:         pack .listarea.frame -side top
369:         pack .listarea.ok -side bottom
370: }
```

Resources

Tcl/Tk books

Tcl and the Tk Toolkit
John K. Ousterhout
Addison Wesley
ISBN 0-201-63337-X

Practical Programming in Tcl and Tk
Brent B. Welch
Prentice Hall
ISBN 0-13-182007-9

Internet resources

World Wide Web site
http://www.sunlabs.com/research/tcl

Tcl/Tk distribution sites
ftp://ftp.cs.berkeley.edu/ucb/tcl/
ftp://ftp.smli.com/pub/tcl/

Tcl/Tk extensions, programs, utilities
ftp://ftp.aud.alcatel.com/
ftp://ftp.neosoft.com/pub/tcl/

List of Frequestly Asked Questions
ftp://rtfm.mit.edu/pub/usenet-by-group/comp.lang.tcl/

Usenet newsgroup
comp.lang.tcl

McIDAS GUI ftp site
ftp.ssec.wisc.edu - email *gui@ssec.wisc.edu* for ftp instructions and password.

Alternatives to Tcl

Tk for perl
Tkinter for Python

McIDAS Navigation and Calibration Subsystems

Presented by

Dave Santek

McIDAS Applications Project Leader

Session 5

McIDAS Developer/Operator Training

October 23-25, 1995

Table of Contents

Overview 5-1
- Terminology 5-1
- Source file naming conventions 5-2
- Applications interfaces 5-2
- Components 5-3
- API 5-4
- Internal function names 5-4

Under the hood on OS/2 5-5

Under the hood on Unix 5-6

Guidelines for writing a module 5-8

References 5-10

Overview

A redesign of the McIDAS Area file format took place in the mid-1980s to accommodate the ever increasing types of remotely sensed data. A generalized method for storing, navigating, and calibrating these data was developed.

The design allows the addition of new data sets from a variety of platforms (satellite, aircraft, radar, etc.) with no changes to existing software or file format. This is done by defining a file format (Areas) that can accommodate multibanded, multibyte data along with a variety of ancillary data. In addition, it is recognized that two basic functions for working with remotely sensed data needs to be handled in software: navigation and calibration. This is done by defining an API (subroutine names, calling sequence, and functionality) that all navigation and calibration modules will adhere to, and a mechanism to access the appropriate module at application run time.

Terminology

The terms defined below are used in this section.

ancillary data	additional information needed to identify, quantitate and manipulate image data, including the directory, navigation and calibration blocks
calibration	the process of converting data values received from the satellite to a useful, physical quantity such as temperature, radiance, or albedo
DLL	Dynamic Link Library; the library of routines used in dynamic linking
dynamic linking	subprograms loaded at run time
navigation	the process of transforming row/column in the image file to lat/lon and vice versa
slot number	1, 2 or 3 to allow for simultaneous use of up to three navigation and calibration modules
static linking	subprograms included at compile/link time

Source file naming conventions

Use the naming conventions below.

	Calibration	**Navigation**
MVS:	MUKBX*nam*	MUNVX*nam*
OS/2 and Unix:	KBX*name*.DLM	NVX*name*.DLM

where *name* represents the name of the data format, such as GVAR, TIRO, AAA, etc. In MVS, the naming convention only allows three characters; so in all but one case, the four character names are truncated to three. The one exception is the GVAR navigation, which is in MUNVXGVR.

Applications interfaces

The McIDAS applications interface to the navigation and calibration modules is totally independent of the modules themselves. This division allows new modules to be written without affecting the applications. A few of these interfaces are described below.

For navigation, a call to **nvset** will load and initialize the appropriate navigation module. The API functions (discussed in the API section) can then be called to perform the navigation transformations. Other applications will make use of higher level functions (**itvll** and **illtv**), hiding some of the details.

For calibration, a call to **araopt** (set Area options) will set up a calibration module only if the physical quantity, which is the UNIT option in **araopt**, is different than that stored in the image file, which is word 53 of the Area directory. Calls to **redara** (read a line of data) will make use of a calibration module only if needed. Most applications do not call the API functions directly; two notable exceptions are IMGPROBE and D.

Below **nvset** and **araopt** are the subroutines that provide the interface to the dynamic navigation and calibration modules; they are **nvprep** and **kbprep**, respectively. **nvprep** and **kbprep** build the name of the module to load, based on the slot number and type of data.

The slot number (1, 2 or 3) allows for the simultaneous use of up to three different navigation and calibration modules. The maximum of three was an arbitrary decision. Up to this point, no applications have made use of more than two navigation modules. Being able to remap a satellite image into a different projection illustrates this need. For example, to remap a GOES satellite image into a Mercator projection, the application would load the GOES navigation into slot 1 and the Mercator projection into slot 2.

There have been problems, though, with a limit of three calibration modules on MVS, where up to six Area files can be read at one time. In the ADDE (Abstract Data Distribution Environment) this is not an issue, since every file read is provided by a different server.

The data type (GVAR, TIRO, AAA, etc.) is also needed to construct the DLL to load. For calibration, the data type is stored in word 52 of the Area directory; for navigation, it is stored in the first word of the navigation block. For GOES-8, the type is GVAR for both navigation and calibration. Thus, the name constructed, using slot number 1, for navigation is NV1GVAR. It will load the module nv1gvar.dll.

API

The navigation and calibration modules are not accessed through their internal function names (as seen in the next section), but rather through the names below, allowing more than one module to be used simultaneously.

The numeric is the *slot number*. To further describe the remap scenario presented in components, the application would be able to do navigation transformations in the Mercator projection by calling NV2EAS and NV2SAE, and in the GOES projection by calling NV1EAS and NV1SAE.

Navigation:	nv1ini, nv1eas, nv1sae, nv1opt nv2ini, nv2eas, nv2sae, nv2opt nv3ini, nv3eas, nv3sae, nv3opt
Calibration:	kb1ini, kb1cal, kb1opt kb2ini, kb2cal, kb2opt kb3ini, kb3cal, kb3opt

Internal function names

Each navigation or calibration module has the following internal function names. The calling sequence and function of these routines are identical across all data types. This is the key to adding new data types or navigation algorithms transparent to the applications.

	Entry points	**Description**
Navigation:	nvxini	initialization
	nvxeas	earth-to-satellite transformation
	nvxsae	satellite-to-earth transformation
	nvxopt	additional operations (satellite sub-point, for example)
Calibration:	kbxini	initialization
	kbxcal	calibration for an array of data values
	kbxopt	additional operations (what type of calibration is possible, for example)

Now let's go 'Under the Hood' to look at how the routines in the DLLs are actually accessed for OS/2 and Unix.

Under the hood on OS/2

This section details the processes well below the API that affect the dynamic linking for the navigation and calibration modules. This information is not needed to write a navigation or calibration module.

A call to **gethdl**, in os2glue.c, loads the DLL (nv1gvar.dll, for example) through the system call **DosLoadModule**, and returns a file handle. To access the individual routines in the DLL, the addresses are obtained by calling **getadr** (in os2glue.c), with the file handle, which calls **DosQueryProcAddr**. In addition to the API routines listed above, the routine **dllsub** (in dllsub.for) is included in the DLLs to set up the McIDAS environment (access to User Common and the LW file subsystem). The list of entry points in the DLLs is contained in navlib.def and callib.def. **dllsub** is called from **nvprep** and **kbprep**.

COMMON/NVXCOM/ stores the addresses of the API routines. An additional layer was imposed to map the API (**nv1ini**, etc.) calls to the appropriate entry point (**nvxini**, etc.) in the DLL. The API routines are stored in .for files of the same name (nv1ini.for, for example). When an API routine is called, it in turn calls one of the **icall***n* routines (in os2glue.c), passing the address of the entry point in the DLL along with all the parameters. The **icall***n* is a mechanism to call a subroutine by address, rather than by name. The code for **nv1eas** and **icall6** is listed below.

Extracted from nv1eas.for:

```
INTEGER FUNCTION NV1EAS(X1,X2,X3,X4,X5,X6)
COMMON/NVXCOM/IHANDL(3),NVADDR(3,4)
NV1EAS=ICALL6(NVADDR(1,3),X1,X2,X3,X4,X5,X6)
RETURN
END
```

Extracted from os2glue.c:

```
long
icall6_(Fint (**isub)(), void *a, void *b, void *c, void *d, void *e,  void *f)
{
   return( (**isub)( a, b, c, d, e, f) );
}
```

Under the hood on Unix

This section details how dynamic linking is simulated for McIDAS-X. This information may be useful when debugging a navigation or calibration module.

When McIDAS was ported to Unix, a standard dynamic loading feature was not identified. Even shared libraries were implemented differently; some requiring relinking when the shared library was updated. It has been five years since we first did an investigation and we have begun to look into it again.

As an interim measure, we have decided to simulate dynamic linking. This is done by compiling the .DLM files as subroutines, storing them in the library, and then statically linking to the applications. Since the entry point names are the same in each navigation or calibration module, the names must be changed to store them in the library. A preprocessor program, **convdlm** (in convdlm.fp), automatically modifies the entry point names in a unique way to avoid duplication of names. Below is a skeleton of a calibration module (kbxtest.dlm), along with the output from **convdlm**, kbxtest1.f.

kbxtest.dlm

```
 1.     INTEGER FUNCTION KBXINI (CIN,COUT,IOPT)
 2.
 3.     COMMON/M0TEST/JTYPE,ISOU,IDES,JOPT
 4.
 5.     KBXINI=0
 6.     RETURN
 7.     END
 8.
 9.     INTEGER FUNCTION KBXCAL(CALB,IDIR,NVAL,IBAND,IBUF)
10.
11.     COMMON/M0TEST/JTYPE,ISOU,IDES,JOPT
12.
13.     CALL MAKTAB(ITAB,ISCAL(1),ISCAL(2),ISCAL(3),ISCAL(4))
14.     CALL MPIXTB(NVAL,ISOU,IDES,IBUF,ITAB)
15.
16.     KBXCAL=0
17.     RETURN
18.     END
19.
20.
21.     INTEGER FUNCTION KBXOPT(CFUNC,IIN,IOUT)
22.
23.     COMMON/M0TEST/JTYPE,ISOU,IDES,JOPT
24.
25.     RETURN
26.     END
27.
28.     SUBROUTINE MAKTAB(ITAB,INLO,INHI,IBLO,IBHI)
29.     RETURN
30.     END
```

kbxtest1.f

```
 1.    INTEGER FUNCTION KB1INItest
 2.  + (CIN,COUT,IOPT)
 3.
 4.    COMMON/M0TESTtestkb1/
 5.  + JTYPE,ISOU,IDES,JOPT
 6.
 7.    KB1INItest
 8.  + =0
 9.    RETURN
10.    END
11.
12.    INTEGER FUNCTION KB1CALtest
13.  + CALB,IDIR,NVAL,IBAND,IBUF)
14.
15.    COMMON/M0TESTtestkb1/
16.  + JTYPE,ISOU,IDES,JOPT
17.
18.    CALL MAKTABtestkb1
19.  + (ITAB,ISCAL(1),ISCAL(2),ISCAL(3),ISCAL(4))
20.    CALL MPIXTB(NVAL,ISOU,IDES,IBUF,ITAB)
21.
22.    KB1CALtest
23.  + =0
24.    RETURN
25.    END
26.
27.    INTEGER FUNCTION KB1OPTtest
28.  + (CFUNC,IIN,IOUT)
29.
30.    COMMON/M0TESTtestkb1/
31.  + JTYPE,ISOU,IDES,JOPT
32.
33.    RETURN
34.    END
35.
36.    SUBROUTINE MAKTABtestkb1
37.  + (ITAB,INLO,INHI,IBLO,IBHI)
38.    RETURN
39.    END
```

All subroutine, function, and common block names are modified. The function **kbxini** becomes **kb1initest**. The common block **common/m0test/** becomes **common/m0testtestkb1/**. Note that **maktab** is modified because it is inline, but the call to **mpixtb** is not.

Guidelines for writing a module

Here are some things to keep in mind when writing a navigation or calibration module. Most of the restrictions are related to preprocessing the routine with **convdlm**.

- Write the module in Fortran. At this time, **convdlm** only runs against Fortran. To use a C module on Unix, you would have to modify the names manually. C modules work fine in OS/2.
- Write the module in uppercase. When **convdlm** was written, all the modules were in uppercase; navigation and calibration modules originated from the mainframe and ported to OS/2. The only recognized comment line begins with a C.
- Do not use the words SUBROUTINE, FUNCTION or COMMON in comment lines or message lines (such as DDEST). Also, in these lines, do not enter the *name* of any subroutine, function, or common block in uppercase.
- Do not use the Fortran ENTRY statement; **convdlm** does not recognize or handle it correctly.
- Do not imbed a function call within another function call if both functions are in the module. **convdlm** will not be able to handle the expansion and will print an error message and then exit. For example, if SUBROUTINE ASUB(K) and FUNCTION BFUNC(J) are both in the .DLM, the following is illegal:

```
CALL ASUB( BFUNC(10))
```

- Do not allow routines that expect character variables to be passed in (KBXINI, KBXOPT, for example), to declare the variables as CHARACTER*(*). The length of the variable is not passed along. So, in KBXINI and KBXOPT, the lengths are known and are so declared as CHARACTER*4.
- Do not output text (SDEST, Fortran WRITE, etc.). This causes problems with ADDE servers that send data through standard output. You can imbed DDEST calls for debugging, but they will only be output with non-ADDE commands.

- Run the process **convdlm** manually if you want to examine **.f** files, as they are automatically deleted during compiling. When compiling .DLMs on Unix, **convdlm** reads the .DLM file and outputs three .f files: kbxtest.dlm becomes kbxtest1.f, kbxtest2.f, and kbxtest3.f. These files are compiled, so any compiler warnings and/or errors refer to these files, which have different line numbers for the statements than the .DLMs. To run **convdlm**, use: **convdlm** *filename*
  for example: **convdlm kbxtest.dlm**.

References

McIDAS Programmer's Manual
Preliminary issue of Chapters 4, 5 and 6
October 1995

Dengel, Russ and Dave Santek (1986): *A Generalized Method for Storing and Processing Digital Satellite Data. Preprints*, 2nd International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology.

Designing and Implementing Calibration Modules

Presented by

Dave Santek

McIDAS Applications Project Leader

Session 6

McIDAS Developer/Operator Training

October 23-25, 1995

Table of Contents

Overview 6-1
Terminology 6-1

Calibration module design 6-2
Module requirements 6-2
Storing parameters 6-3
Performance: computation vs. look-up tables. 6-5

Structure of a calibration module 6-7

Integrating calibration modules into McIDAS 6-9
McIDAS-OS2 6-9
McIDAS-X 6-9

Sample program 6-10

References. 6-14

Overview

The McIDAS calibration subsystem is designed to be extensible, so that new data or calibration techniques for existing data types can be included. Thus, you can define calibration modules that will allow McIDAS applications to view the data that you prescribe.

This training session presents the issues that you should resolve when designing a calibration module. It also provides a sample module. It is assumed that you have a basic understanding of the McIDAS Area file format and a conceptual understanding of calibration and how it relates to remotely sensed data.

Terminology

The terms defined below are used in this section.

band	spectral band; band 4 for the GOES-8 Imager is 10.7 microns (infrared); see Appendix G of the *McIDAS-X Users Guide*
data value	1-, 2-, or 4-byte quantity; sometimes called *pixel*
physical quantity	radiance, temperature, albedo, etc.; sometimes called *unit*
sensor source number	SS number stored in word 3 of the Area directory; 70 is the GOES-8 Imager; see Appendix H of the *McIDAS-X Users Guide*
sensor type	up to four characters indicating the sensor; it is stored in word 52 of the Area directory; GVAR is the sensor on GOES-8
slot	the number 1, 2 or 3 to allow for loading of up to three dynamic modules
unit	measurement standard, such as Celsius or meters

Calibration module design

All calibration modules are built using the same framework. They must conform to the McIDAS convention for functionality, names of functions, and types of arguments. This standardization allows applications to make use of these modules in a generic, yet powerful way. It is usually not necessary for applications to have any private knowledge of the data with which it is working; the calibration interface provides a means to acquire certain aspects that are common to all.

Although the framework is rigid, you must resolve certain implementation issues. Two key issues, which are described later, are the storing of calibration parameters and performance.

Module requirements

Each calibration module has identical function names and interface, and performs similar operations. The actual algorithm for calibrating the data is hidden from the application, regardless of the calibration type.

All calibration modules contain the following three functions:

Function	Description
kbxini	initializes and verifies the requested calibration
kbxcal	calibrates the data
kbxopt	provides additional operations, which are usually queries from the application

The structure of these functions is designed so that any required ancillary data is passed in as arguments or handled by **kbxopt**. An exception is the access to the calibration block. The calibration subsystem was created about 10 years ago, before the use of calibration blocks. Currently, calibration blocks are handled by reading the disk from within the module. We are investigating changing this, due to the new ADDE paradigm, which presents a different view of the data to the applications.

All the functions either return the requested data or a bad status. They should never terminate, but rather rely on the exception handling of the application.

Most of the current McIDAS calibration modules store parameters in these locations:

- Area directory
- calibration block
- line prefix
- LW file
- none

Area directory

The MSAT (Meteosat) calibration parameters are stored in the Area directory. Although the entire image only has three constant parameters, they do change twice per day. The Area directory is not the recommended storage location for your parameters because of the lack of available words in the directory. In retrospect, the parameters should have been placed in a calibration block.

Calibration block

The AAA (GOES-7 Mode AAA), QTIR (Quick AVHRR), PRD (Product), and GVAR (GOES-8, etc.) calibration parameters are stored in the calibration block. This block consists of 128 words and is the most used location for storing calibration parameters. Although the length of this block will be changed to unrestricted, it will remain at 128 words on the mainframe. These values are used for the entire image file. Sections are defined for AAA and GVAR relating to the parameters for the different bands.

Normally, integers or scaled integers are stored so that moving the data to different platforms, such as Unix or OS/2, is not a problem. However, there are potential problems when storing character data. For example, when accessing files that are not native to the platform, byte flipping may have to take place on the integers but not the character data. The problem arises when deciding how to flip the bytes, since a schema for storing calibration parameters is not defined. Currently, 4-byte words are tested to determine if all the bytes are printable characters. This does not always work for large or scaled integers. Be aware of this situation when developing new calibration modules, if problems seem to be platform-dependent. A software solution to this problem is in progress.

Line prefix

VAS (GOES-5 Sounder), AAA (GOES Mode AAA), and TIRO (AVHRR) calibration parameters are stored in the prefix part of the data line. This is the preferred location for image data, where the calibration parameters can change throughout the image. For AAA data, two different channels may alternate through the image; TIRO calibration has a different set of parameters every five lines.

The prefix can be up to 1000 bytes, and is divided into four sections:

- validity code
- documentation
- calibration
- level

You can define both the documentation and calibration sections for specific data. However, you should use only the calibration section for storing calibration parameters; the documentation section is not guaranteed to move with some of the copy commands.

LW file

The GMS (Japanese satellite) and VAS calibration parameters are stored in LW files. The use of these files is decreasing as the definition of the calibration block becomes less restricted. With the previous limit of 128 words in the calibration block, storing large look-up tables in an LW file was preferable to including it as DATA statements in the code.

For VAS calibration, the line-to-line variability in the calibration coefficients required that the look-up tables be pre-generated for better performance. The file VASTBLS, which accounts for all possible look-up tables, is over 6 megabytes.

GMS data is received from the satellite as 1-byte values. For the IR, each value corresponds to a temperature; for the visible, an albedo. Because this table is fixed for GMS-3, and the calibration block is restricted to 128 words, an LW file is the best place to store the calibration information. On the workstation, changes are underway to allow for calibration blocks of any size. This is needed for GMS-5, since the table is no longer fixed.

None

Some calibration algorithms, such as VISR (GOES 1-byte data) and WSI (WSI Radar), do not require additional information, or the amount is fixed and relatively small.

The VISR calibration was designed for the original GOES satellites, which transmitted its data as 1-byte values. For the visible (even sensor source numbers), the RAW value is the same as the BRIT, and no conversion is necessary. For the IR (odd sensor source numbers), an option for TEMP (temperature) is available.

WSI radar images need only 16 values to represent the data, base map, labels, etc. These images are handled in the code without an external data structure.

Performance: computation vs. look-up tables

Most calibration modules in McIDAS generate look-up tables to convert the stored data to some output physical quantity. This is usually preferred over performing the computation for every input data value. For example, a standard McIDAS image frame is about 300,000 pixels. Since most input data is 8 or 10 bits per value, performing 256 or 1024 computations to create a look-up table is much more efficient than doing 300,000.

The table below shows the functions used extensively in calibration modules.

Routine	**Description**
movb	moves bytes
movc	moves bytes
movw	moves 4-byte words
movpix	moves bytes with sampling and offsets
movblk	moves blocks of bytes with sampling and offsets
mpixel	in place data expand/pack
mpixtb	in place data expand/pack with lookup table
maaatb	AAA specific mpixtb
mavhtb	AVHRR specific mpixtb
mvastb	VAS specific mpixtb
mgvatb	GVAR specific mpixtb

The first three (**movb, movc, movw**) move data from one buffer to another. **movpix** includes parameters for sampling bytes and moving into a buffer in reverse order. **movblk** has the same functions as **movpix**, but also operates on blocks of bytes.

mpixel expands and packs the data values in place. If the source of the data is 1 byte, but the application expects it as 4 bytes, **mpixel** will do it without the need of an additional buffer. **mpixtb** adds the feature of passing the data through a look-up table.

The last four functions are special implementations of **mpixtb**. This was done for performance and memory considerations, such as AAA data sent by the satellite as 10-bit data, but stored on disk as 15 bits. Rather than generating a look-up table of 32,768 values (for 15 bits), a table of 1024 values is made for the 10-bit data. **maaatb** does the bit shifting to take the raw 15-bit data to 10-bit, and then passes it into the look-up table.

Structure of a calibration module

The sample calibration module, on pages 6-10 through 6-13, illustrates the structure of a calibration module. This calibration will accept input data, from 0 to 255, and return values modified by a sine curve. To use this module with existing 1-byte data, run the following McIDAS command:

CA *Area* **STYPE=SIN CTYPE=RAW**

After compiling KBXSIN.DLM and any appropriate applications, McIDAS applications can be run against the data.

The required three functions are present:

- kbxini **[1-72]**
- kbxcal **[74-129]**
- kbxopt **[132-205]**

An additional subroutine, **maktab**, generates the look-up table.

kbxini takes it input, usually from **araopt**, and copies it to a local buffer **[53]**. It then verifies that the calibration requested is valid **[59-60]**.

kbxcal is usually not called directly from the application; it is called when required by **redara**. It takes as input the prefix of the line, the Area directory, the number of values to calibrate, the band (if required), and the buffer containing the data. The calibrated data is returned through this same buffer. **kbxcal** checks to see if the look-up table was generated **[118]**. If not, a call is made to **maktab [119]**. **mpixtb** completes the calibration **[125]** by taking the data, passing it through the look-up table, and expanding or packing the bytes.

kbxopt contains additional operations used by applications to query information about the calibration. The KEYS option **[182-187]** passes in a frame directory block to the calibration module; the number and list of physical quantities are returned. This option was written for the D and IMGPROBE programs, which list out the stored data value converted to appropriate quantities. Because the information returned by the KEYS option was incomplete, the option, INFO **[191-202]**, was added to provide scale factors and units. The input for INFO is: band number, sensor source number and calibration type.

An important feature to note is that most calibration modules contain code to handle stretch tables generated by the SU command. By calling **kbxopt** with BRKP as the option and the name of the stretch table, the calibration module computes a modified brightness value based on the table. You can usually identify the sections of code where this is done by finding the CALTYP variable, which is held in COMMON/BRKPNT. In ADDE, this function is done in the client application instead of the calibration module. This code will be removed from the calibration modules at some future date.

Integrating calibration modules into McIDAS

When the calibration module is coded, you must incorporate it into McIDAS for testing and use by placing the source code in the proper directory and running the appropriate McIDAS tools.

McIDAS-OS2

Copy the calibration module source code (our example is KBXSIN.DLM) into the \mcidas\working directory and run the following three commands from the OS/2 command line.

```
F   KBXSIN   DL   CALLIB   KB1SIN
F   KBXSIN   DL   CALLIB   KB2SIN
F   KBXSIN   DL   CALLIB   KB3SIN
```

These commands invoke the F.CMD script and produce KB1SIN.DLL, KB2SIN.DLL, and KB3SIN.DLL in \mcidas\user\code. The calibration of type SIN is immediately available; it is not necessary to recompile applications.

McIDAS-X

Integrating a user-developed calibration module into McIDAS-X is more complicated because of the lack of dynamic linking. Copy the source KBXSIN.DLM into the mcidas/working directory and run the following procedure from the Unix prompt.

```
fx  kbxsin dl
```

This procedure generates, compiles, and files in the user library the three source files kbxsin1.f, kbxsin2.f, and kbxsin3.f, each with unique generated names for all function calls and common blocks. Then, you must generate a new kbprep.for with explicit references to the new calibration type by running the two commands below at the Unix prompt.

```
cal_init ~mcidas/mcidas2.1/src/kb*.dlm  kbx*.dlm > kbprep.for
fx kbprep li
```

You must then recompile all applications that use calibration before they can access the new type. Note that ~mcidas/mcidas2.1/src is the directory where the core McIDAS-X source can be found for version 2.1. Adjust this accordingly for other versions of McIDAS-X.

Sample program

The sample calibration module, KBXSIN.DLM is provided below.

```
1:       INTEGER FUNCTION KBXINI(CIN,COUT,IOPT)
2:
3: *$ Name:
4: *$       kbxini - Initialize for sine modified calibration
5: *$
6: *$ Interface:
7: *$       integer function
8: *$       kbxini( character*4 cin, character*4 cout, integer iopt(*))
9: *$
10: *$ Input:
11: *$       cin     - input physical quantity ('TEMP', 'BRIT', 'RAW', etc.)
12: *$       cout    - output physical quantity
13: *$       iopt    -
14: *$                    iopt(1)     precision of stored data (1, 2 or 4 bytes)
15: *$                    iopt(2)     spacing of output data (1, 2 or 4 bytes)
16: *$                    iopt(3-5)   filled by araopt but should not be used
17: *$
18: *$ Input and Output:
19: *$       none
20: *$
21: *$ Output:
22: *$       none
23: *$
24: *$ Return values:
25: *$        0          - success
26: *$       -1          - unit conversion not possible
27: *$
28: *$ Remarks:
29: *$        This calibration module will only accept values from 0 to 255
30: *$        and will return values modified by a sine curve. There is no
31: *$        check for input data out of range.
32: *$
33: *$ Categories:
34: *$         calibration
35:
36:       CHARACTER*4 CIN
37:       CHARACTER*4 COUT
38:       INTEGER IOPT(*)
39:
40:       INCLUDE 'areaparm.inc'
41:
42:       INTEGER JTYPE
43:       INTEGER ISOU
44:       INTEGER IDES
45:       INTEGER JOPT(NUMAREAOPTIONS)
46: C
47: C--- Store information needed in other functions
48: C
49:       COMMON/M0SIN/JTYPE,ISOU,IDES,JOPT
50: C
51: C--- Copy what araopt sent in
52: C
53:       CALL MOVW(NUMAREAOPTIONS,IOPT,JOPT)
54:
55:       JTYPE=0
56:       ISOU=IOPT(1)          ! length in bytes of input data
57:       IDES=IOPT(2)          ! length in bytes to ouput data
58:
59:       IF(CIN.EQ.'RAW'.AND.COUT.EQ.'SIN')  JTYPE=1
60:       IF(CIN.EQ.'RAW'.AND.COUT.EQ.'BRIT') JTYPE=2
61: C
```

```
C--- If not one of the 2 cases above are true, error
C
      IF(JTYPE.EQ.0) GO TO 900

      KBXINI=0
      RETURN

900 CONTINUE
      KBXINI = -1
      RETURN
      END

      INTEGER FUNCTION KBXCAL(PREFIX,IDIR,NVAL,IBAND,IBUF)
*$ Name:
*$       kbxcal - Calibrate data
*$
*$ Interface:
*$       integer function
*$       kbxcal( integer prefix(*), integer idir(*), integer nval,
*$                  integer iband, integer ibuf(*))
*$
*$ Input:
*$       prefix - prefix part of image line              (not needed)
*$       idir   - Area directory                         (not needed)
*$       nval   - number of values to calibrate
*$       iband  - band number                            (not needed)
*$
*$ Input and Output:
*$       ibuf - buffer containing data
*$
*$ Output:
*$       none
*$
*$ Return values:
*$        0          - success
*$       -1          - error                             (not needed)
*$
*$ Categories:
*$         calibration

      INTEGER PREFIX(*)

      INCLUDE 'areaparm.inc'

      INTEGER JTYPE
      INTEGER ISOU
      INTEGER IDES
      INTEGER JOPT(NUMAREAOPTIONS)

      INTEGER ITAB(256)

      COMMON/M0SIN/JTYPE,ISOU,IDES,JOPT
      DATA IFLAG/0/
C
C--- If the calibration type changes, remake the lookup table
C
      IF( JTYPE .NE. IFLAG) THEN
         CALL MAKTAB(JTYPE, ITAB)
         IFLAG = JTYPE
      ENDIF
C
C--- Pass the data IBUF through the lookup table ITAB
C
      CALL MPIXTB(NVAL,ISOU,IDES,IBUF,ITAB)

      KBXCAL=0
      RETURN
      END


      INTEGER FUNCTION KBXOPT(CFUNC,IIN,IOUT)
*$ Name:
```

```
134: *$       kbxopt - Additional operations
135: *$
136: *$ Interface:
137: *$       integer function
138: *$       kbxopt( character*4 cfunc, integer iin(*), integer iout(*))
139: *$
140: *$ Input:
141: *$       cfunc  - function ('INFO', 'KEYS')
142: *$       iin    - for cfunc 'KEYS', iin contains frame directory block
143: *$                for cfunc 'INFO'
144: *$                     iin(1)    - band number
145: *$                     iin(2)    - sensor source number
146: *$                     iin(3)    - calibration type ('GVAR', for example)
147: *$
148: *$ Input and Output:
149: *$       none
150: *$
151: *$ Output:
152: *$       iout   - for cfunc 'KEYS'
153: *$                  iout(1)   - number of physical quantities ('TEMP', etc.)
154: *$                  iout(2-n) - list of physical quantities
155: *$       iout   - for cfunc 'INFO'
156: *$                  iout(1)   - number of physical quantities ('TEMP', etc.)
157: *$                  iout(2-n) - list of physical quantities, units,
158: *$                              and scale factors
159: *$
160: *$ Return values:
161: *$        0          - success
162: *$       -1          - invalid function
163: *$
164: *$ Categories:
165: *$         calibration
166:
167:       CHARACTER*4 CFUNC
168:
169:       INTEGER IIN(*)
170:       INTEGER IOUT(*)
171:
172:       INCLUDE 'areaparm.inc'
173:
174:       INTEGER JTYPE
175:       INTEGER ISOU
176:       INTEGER IDES
177:       INTEGER JOPT(NUMAREAOPTIONS)
178:       COMMON/M0SIN/JTYPE,ISOU,IDES,JOPT
179: C
180: C--- KEYS option
181: C
182:       IF( CFUNC .EQ. 'KEYS') THEN
183:          IOUT(1) = 3              ! Number of types
184:          IOUT(2) = LIT('RAW ')    ! Physical quantities
185:          IOUT(3) = LIT('SIN ')
186:          IOUT(4) = LIT('BRIT')
187:       ENDIF
188: C
189: C--- INFO option
190: C
191:       IF( CFUNC .EQ. 'INFO') THEN
192:          IOUT(1) = 3              ! Number of types
193:          IOUT(2) = LIT('RAW ')    ! Physical quantities
194:          IOUT(3) = LIT('SIN ')
195:          IOUT(4) = LIT('BRIT')
196:          IOUT(5) = LIT('    ')    ! Units
197:          IOUT(6) = LIT('none')
198:          IOUT(7) = LIT('    ')
199:          IOUT(8) = 1              ! Scale factors
200:          IOUT(9) = 1000
201:          IOUT(10)= 1
202:       ENDIF
203:
204:       RETURN
205:       END
```

```
206:
207:
208:       SUBROUTINE MAKTAB(JTYPE, ITAB)
209: *$ Name:
210: *$       maktab - Make lookup table for sine modified calibration
211: *$
212: *$ Interface:
213: *$       subroutine
214: *$       maktab( integer jtype, integer itab(*))
215: *$
216: *$ Input:
217: *$       jtype  - calibration type
218: *$                1    - sine
219: *$                2    - grayscale
220: *$
221: *$ Input and Output:
222: *$       none
223: *$
224: *$ Output:
225: *$       itab   - lookup table of 256 values
226: *$
227: *$ Remarks:
228: *$       This routine makes a lookup table by computing the sine
229: *$       for all possible values from 0 to 255 (the range of the
230: *$       input data). Rather than computing the sine directly on
231: *$       the values 0 to 255, it is initially scaled to 0 to 10
232: *$       which is approximately 3 sine waves (3 * PI = 10)
233: *$
234: *$ Categories:
235: *$         calibration
236:
237:       INTEGER ITAB(*)
238:
239:       REAL SINVAL
240:       REAL X
241: C
242: C---
243:
244:       DO 100 I = 1, 256
245:
246:          X = I - 1                  ! X goes from 0 to 255
247:          X = 10. * (X / 255.)       ! Normalize and scale to 3 sine waves
248:          SINVAL = SIN(X)
249: C
250: C--- Output sine value
251: C
252:          IF (JTYPE .EQ. 1) THEN
253:             ITAB(I) = NINT(SINVAL * 1000.)        ! Scale sine by 1000
254: C
255: C--- Output gray scale value
256: C
257:          ELSE IF (JTYPE .EQ. 2) THEN
258:             ITAB(I) = NINT( 127. + 128 * SINVAL) ! Scale to 0 to 255
259:          ENDIF
260:
261: 100   CONTINUE
262:
263:
264:       RETURN
265:       END$
```

References

McIDAS Programmer's Manual
Preliminary issue of Chapters 4, 5 and 6
October 1995

McIDAS Applications Programming Manual
Issued February 1988; Revised November 1993

Dengel, Russ and Dave Santek (1986): *A Generalized Method for Storing and Processing Digital Satellite Data.* Preprints, 2nd International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology.

Designing and Implementing Navigation Modules

Presented by
Robert Merrill
McIDAS Applications Programmer

Session 7
McIDAS Developer/Operator Training
October 23-25, 1995

Table of Contents

Overview 7-1

Navigation module requirements 7-2

Navigation module design 7-3

Navigation algorithm 7-4
- Coordinate conventions 7-4
- Navigation example - the tangent cone projection 7-5
- Algorithm development 7-7
- Additional operations 7-9

Implementing navigation modules 7-10
- Coding conventions in McIDAS-X 7-10
- Sample code description 7-11

Integrating navigation modules into McIDAS 7-13
- McIDAS-OS2 7-13
- McIDAS-X 7-13

Sample navigation module 7-14

Sample application 7-25

Overview

The McIDAS navigation subsystem allows you to extend the list of available map projections for data remapping and display by writing new navigation modules. These user-defined navigation modules can provide exactly the right map projection for your data. You can remap imagery directly into the new projection using the REMAP command. Then a blank image with the new navigation type can serve as a background for the grids and point source data that you want to view in the custom projection.

Navigation modules perform these services:

- convert image coordinates to earth coordinates
- convert earth coordinates to image coordinates
- provide special services associated with a particular navigation type

Because all navigation modules provide these services, a ready-made design framework, including naming and argument conventions, is already in place. Constructing a new navigation module, especially for a map projection, is simpler than it may appear. The difficult part is the navigation algorithms because information about the transform equations is often incomplete. Implementing most map projections will be straightforward if you understand the projection in question, recognize the additional relationships needed, and derive them.

Modules for satellite navigation are more difficult because they involve prediction of the satellite's position in time, operations in both celestial and terrestrial coordinates, three-dimensional vector operations, and possibly an iterative solver for the inverse (earth to image) transform. The overall design principles and approach are the same, however.

This training session will describe the design and implementation of NVXTANC, a navigation module for the tangent cone projection. Although it is a relatively simple navigation module, it has many features common to more complex modules, such as those used for satellite navigation.

Navigation module requirements

All navigation modules must satisfy the following requirements.

- They must validate their inputs.
- They must never crash, regardless of the inputs.
- They must return either a good transformation or an error status.

All navigation modules, including the ones you will write, must conform exactly to the McIDAS conventions for function naming and arguments, both in position and type. These conventions are shown in the *Sample navigation module* section of this training session.

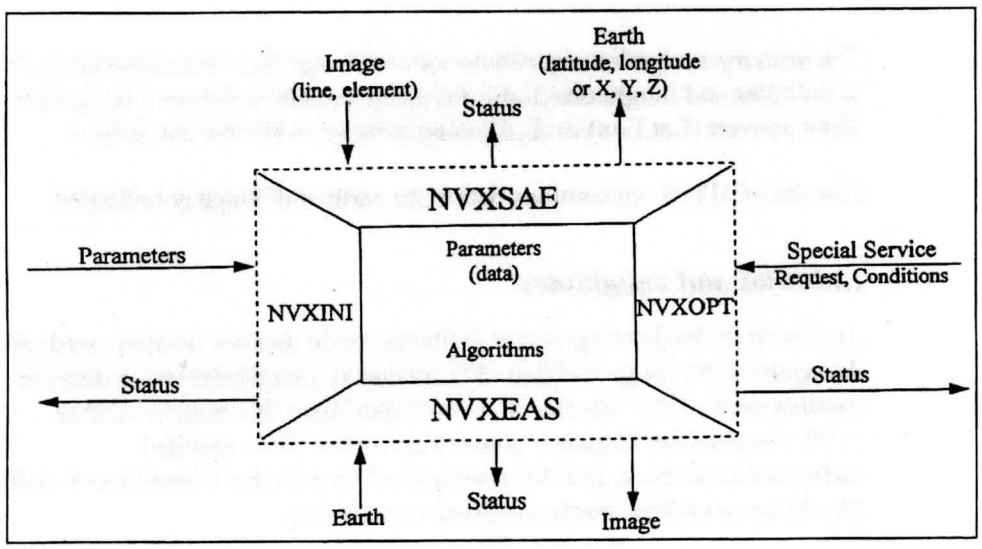
The system can only select the right navigation type at run time based on the first word of the navigation block or *codicil*. The selection is done by a call to **nvprep**. In MCIDAS-OS2, **nvprep** links the correct navigation module at run time. In MCIDAS-X, **nvprep** uses an arithmetic IF statement to run the correct navigation module. Either way, the calling sequences for all navigation services must be identical.

A navigation module should never crash, regardless of its inputs, or return an erroneous solution. If the input cannot be transformed, it must return an error status. The application is then responsible for handling the error properly. To ensure this is done, write your navigation modules so that if an error occurs, the module returns outrageous transform results and an error status. During development, this will help you uncover bugs in the applications that use the modules.

Navigation module design

The interface requirements determine the navigation module design. McIDAS navigation modules are *polymorphic*. This means they have identical entry points and perform similar operations, regardless of navigation type. The type of transformation (the algorithm) and the particular instance of that transformation (parameters) are hidden in the module. The drawing below shows how a navigation module encapsulates its algorithm and parameters while presenting a familiar polymorphic face to the applications.

Navigation module components and interface



All navigation modules contain the four functions below.

Function	Description
nvxini	initializes the navigation module
nvxsae	converts image to earth coordinates; used by the E command
nvxeas	converts earth to image coordinates; used by the PCE command
nvxopt	performs other special services besides image-earth transforms

The actual navigation algorithms (equations) are coded in the module. The application initializes the module by passing it parameters, which are stored in the module between service calls. They may be altered only by the application via a call to **nvxini** to reinitialize the module. **nvxini** and the three navigation services all return a status.

Navigation algorithm

Published map projections typically contain only equations to transform latitude and longitude onto a Cartesian system on the projection surface. You must understand the projection's properties and often must also derive an inverse transformation and equations to convert between McIDAS earth and image coordinates and earth and projection coordinates, respectively, used by the projection.

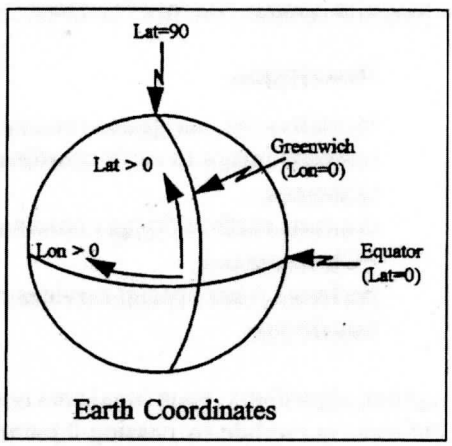
Coordinate conventions

The primary navigation algorithms convert image lines and elements (L,E) to latitudes and longitudes (Lat,Lon) using **nvxsae** or forward navigation. They convert (Lat,Lon) to (L,E) using **nvxeas** or inverse navigation.

Use the McIDAS conventions below for earth and image coordinates.

Latitudes and Longitudes

As shown in the drawing below, latitudes are in degrees positive north of the equator; the range is -90 to +90, inclusive. Longitudes are in degrees positive west of the Greenwich (prime) meridian; the range is -180 to +180. Because the longitude convention differs from standard cartographic practice, and the treatment of the date line is ambiguous, you should be careful to handle longitude consistently.

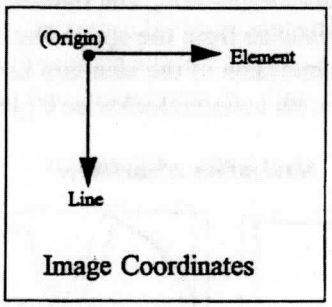


Earth Coordinates

Image coordinates

Image coordinates are in lines and elements, as shown below, and may contain a fractional component. It is convenient to think of (L,E) as defining a right-handed coordinate system with its origin in the upper-left of the image as displayed on the screen.

Image coordinates are the basic coordinate system for data stored in both AREA files and the TV display. Area and TV coordinates can be converted to or from image coordinates using information in the area directory or frame directory, respectively. Navigation modules all operate directly on image coordinates.



Navigation example - the tangent cone projection

As an example, we will implement a navigation module for a tangent cone projection (Saucier 1983)[1]. All we are given is the following:

- a pair of equations for R and θ, which are the polar coordinates on the developed projection surface, in terms of ψ and λ
- a third equation for map scale σ as a function of ψ

This is typical. The cartographers who develop map projections speak their own language and use their own symbols and conventions. As a McIDAS navigation module developer, you must bridge the gaps. It is generally easier to work in the variables of the projection and convert to and from McIDAS earth coordinates as a pre- or post-processing step.

As shown in the drawings on the next page, the projection's earth coordinates are ψ and λ, or colatitude and longitude. Colatitude is measured in radians, beginning from 0 at the pole of the projection, which is the apex of the tangent cone. The longitude is also in radians with 0 at the prime meridian, but is positive to the east.

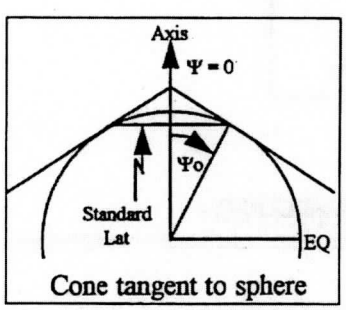
---

[1] Saucier, W. J. 1983: *Principles of meteorological analysis*. Dover Publications, Inc., New York. 438 pp.

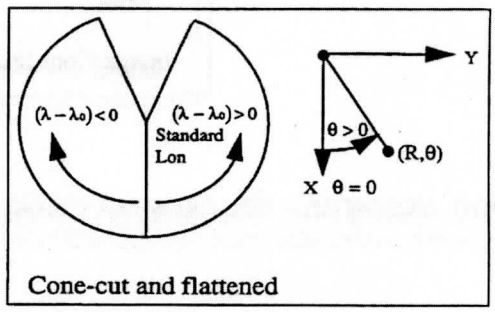
The colatitude at which the cone is tangent to the Earth is ψ_0, or the standard colatitude. At this colatitude, distance on the imaginary tangent cone exactly matches distance on Earth. Elsewhere, distances on the cone are larger than on the Earth by a factor σ, which is a function of ψ only.

This is the mathematical price paid for representing the curved earth on a flat surface. This particular projection is *conformal*, meaning the scale σ at a point is independent of direction. This is not true of all projections.

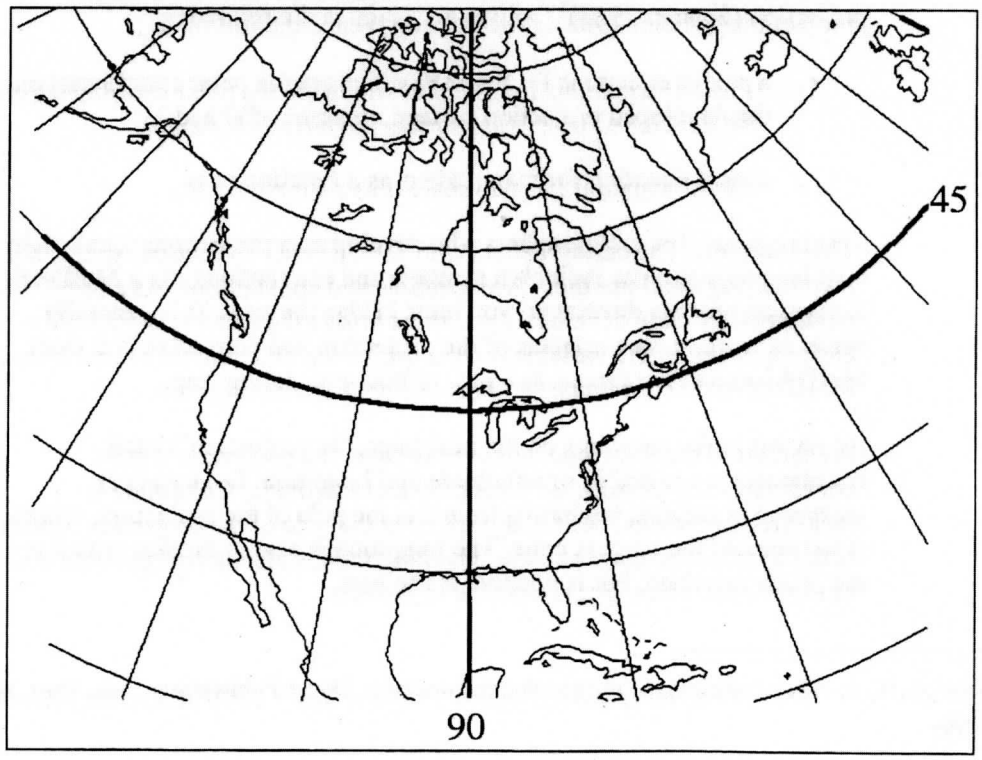
The projection converts earth coordinates to projection coordinates R and θ on the developed surface. Think of the projection as being made by removing the tangent cone from the Earth, cutting it along the longitude opposite the standard longitude λ_0, and flattening it. The radius R of a point on the cone is its distance from the apex. The bearing θ of the point is the angle between the meridian of the standard longitude λ_0 and a meridian through the point, with counterclockwise θ positive.



Cone tangent to sphere



Cone-cut and flattened



Derived tangent cone navigation transforms use the following symbols.

Symbol	Meaning
a	Earth radius
E	image element
E_0	image element location of the North Pole
L	image line
L_0	image line location of the North Pole
Lat	latitude; McIDAS convention
Lon	longitude; McIDAS convention
m	map scale, in km per pixel at standard latitude
R	radius on the projection surface, in km
λ	longitude; projection convention
λ_0	standard longitude; projection convention
θ	bearing from standard longitude (and element axis) on the projection surface
ψ	colatitude
ψ_0	standard colatitude

Note the limiting cases and singularities

The equations for R and θ are shown below.

$$R = a \tan\psi_0 \left[\left(\tan\frac{\psi}{2} \right) / \left(\tan\frac{\psi_0}{2} \right) \right]^{\cos\psi_0} \quad (1)$$

$$\theta = \cos\psi_0 (\lambda - \lambda_0) \quad (2)$$

Inspection reveals that a divide-by-zero will occur for ψ_0 of 0. For ψ_0 of $\pi/2$, the leading $\tan\psi_0$ in the expression for R will tend to infinity, leading to another singularity. You must recognize and reject these choices of standard colatitude (North Pole and equator, respectively).

For $\psi_0 > \pi/2$, which is a standard colatitude in the Southern Hemisphere, the leading $\tan\psi_0$ term in the expression for R is negative, leading to negative radii. It is unclear if this would be consistent for a tangent cone with its apex over the South Pole. You would also have to reconsider the sign convention for longitude (λ) for the Southern Hemisphere. For a fully general navigation module, you should resolve these difficulties to support Southern Hemisphere tangent cones. However, in this training session, we will exclude $\psi_0 > \pi/2$.

You should also note that the valid range of θ is not $-\pi < \theta \leq \pi$ but $-\pi \cos\psi_0 < \theta \leq \pi \cos\psi_0$. This means that points in the pie-wedge region where the cone was cut and flattened are not navigable and should be rejected as arguments to inverse navigation. Also note that as $\psi \rightarrow \pi$, $R \rightarrow \infty$, meaning the South Pole is unnavigable in practice and should be excluded. The limits on inputs are summarized below.

Parameters

$0 < \psi_0 < \pi/2$	standard colatitude is confined to the Northern Hemisphere
$-\pi/2 < \lambda_0 \leq \pi/2$	standard longitude must be a legal value

Earth coordinates

$0 \leq \psi < \pi$	all colatitudes are navigable except the South Pole
$-\pi < \lambda \leq \pi$	all longitudes are navigable

Image coordinates

$R \geq 0$	no valid earth location maps to -R
$-2\pi\cos(\psi_0) < \theta \leq 2\pi\cos(\psi_0)$	area in the split region of the flattened cone is not navigable; exclude it as an input

Derive the inverse transform

The next step is to derive the forward transform (R,θ) to (ψ,λ) from the inverse transform provided by algebraic manipulation of (1) and (2), yielding the two equations below.

$$\psi = 2\tan^{-1}\left[\tan\frac{\psi_0}{2}\left(\frac{R}{a\ \tan\psi_0}\right)^{\frac{1}{\cos\psi_0}}\right] \qquad (3)$$

and

$$\lambda = \lambda_0 + \frac{\theta}{\cos\psi_0} \qquad (4)$$

Extend transforms to McIDAS API units

Now write the equations for the conversions between McIDAS and the projection earth coordinates. They are straightforward, involving a degree-radian conversion, the definition of colatitude and latitude, and the differing convention for positive longitude as shown below.

$$\psi = \frac{\pi}{2} - \frac{\pi}{180}\mathrm{Lat} \qquad \lambda = -\frac{\pi}{180}\mathrm{Lon} \tag{5}$$

and

$$\mathrm{Lat} = 90 - \frac{180}{\pi}\psi \qquad \mathrm{Lon} = -\frac{180}{\pi}\lambda \tag{6}$$

To understand the conversion between (R,θ) and screen coordinates, imagine the flattened cone superimposed on your image coordinate system with the standard longitude parallel to the left edge of the screen. You will want to specify the location of the pole on the image (L_0,E_0) and the map scale *m* (distance on the cone in km per pixel). Although the convention that the standard longitude λ_0 lies along a constant element value could be relaxed by adding another parameter for rotation, this has not been done. Simple trigonometry yields these conversions between (R,θ) and (L,E).

$$L = L_0 + \frac{R\cos\theta}{m} \qquad E = E_0 + \frac{R\sin\theta}{m} \tag{7}$$

and

$$R = m\sqrt{(L - L_0)^2 + (E - E_0)^2} \tag{8}$$

$$\theta = \tan^{-1}[(E - E_0)/(L - L_0)]$$

Additional operations

Optional services can involve the computation of anything about the projection that an application needs. The example here is the computation of the scale factor at any point on the Earth. The expression for σ gives the ratio of the map scale at a given ψ to that at ψ_0, so the actual map scale M(ψ) is just *m*σ(ψ), where *m* is km per pixel specified at ψ_0.

$$M(\psi) = m\sigma(\psi) = m\left(\frac{\sin\psi_0}{\sin\psi}\right)\left[\frac{\tan\psi}{\tan\psi_0}\right]^{\cos\psi_0} \tag{9}$$

Implementing navigation modules

To implement your navigation module, you should know the special coding conventions for McIDAS-X and the architecture of the module.

Coding conventions in McIDAS-X

Navigation modules must conform to the following coding conventions to build properly in MCIDAS-X.

- The name declaration for each function must be uppercase: INTEGER FUNCTION NVXINI(...
- The word COMMON in all common block definitions must be in uppercase, and no more than one space must occur between it and the / at the beginning of the common block name. The use of unnamed (blank) common is generally a risky practice; it is not allowed here.
- The words FUNCTION, COMMON, and any of the interface function names NVXINI, NVXSAE, NVXEAS, and NVXOPT should not occur in comments.

These restrictions are necessary because dynamic linking is not presently supported in MCIDAS-X. To emulate dynamic linking to allow more than one navigation module (slot) to be used by an application at a time (as for REMAP), the scripts that build navigation for MCIDAS-X generate three distinct source modules, each with unique interface function names.

Common blocks are also renamed to avoid collisions between modules. The coding conventions above make it possible for the scripts to recognize those names that need to be modified.

Two sample source files are provided at the end of this document:

- sample navigation module
- sample application

Each is described below. Numbers in bold type refer to the numbered lines in the examples.

Sample navigation module

The sample navigation module contains the four routines **nvxini**, **nvxsae**, **nvxeas**, and **nvxopt** required for all navigation modules.

When called from an application via **nvprep** to initialize the module (first argument *option* value of 1), **nvxini** validates the navigation type **[150]**. It then converts the parameters to floating point **[154-158]**, checks their validity, and converts them from McIDAS to projection form **[166-212]**. If successful, it stores them and some intermediate quantities needed by the navigation transform **[199-202]** in the common block **[118-120]** and sets the initialization flag **[246]**. **nvxini** is also used by applications to select the form of the Earth coordinates, latitude-longitude or Cartesian, by specifying a value of 2 for argument *option* and a value of LL or XYZ for the second argument *param*. Lines **224-235** interpret these inputs and either set or clear the flag Latlon, also stored in the common block.

Forward navigation (**nvxsae**) first verifies that the navigation module is initialized **[368-371]**. It then converts the incoming image coordinates into projection R and θ **[376-380]**. Range checking is then applied **[385-388]** according to the conditions derived earlier (see the *Algorithm development* section). The actual navigation transform is very short **[393-397]**. The earth coordinates are then made to conform to the McIDAS convention **[403-405]** and, if necessary, converted from latitude-longitude to Cartesian coordinates using McIDAS library routine **nllxyz [408]**.

Inverse navigation (**nvxeas**) follows a similar pattern. The module state is checked **[539-542]**, incoming earth coordinates are converted to McIDAS latitude and longitude, if required by the current navigation option **[548-555]**, and the range is checked **[560-573]**. When this is done, the latitude and longitude are converted from McIDAS to projection convention **[575-590]**, the transform is applied **[598-599]**, and image coordinates are computed **[609-610]**.

nvxopt first verifies that the module is initialized **[720-723]**. It then examines the name of the special service request **[729 and 749]**. At present, only SCAL is supported; other options return an error status **[753-754]**. As in **nvxsae** and **nvxeas**, the SCAL option in **nvxopt** involves an input range check **[734-740]**, conversion from McIDAS to projection form of Earth coordinates **[742]**, and algorithm evaluation **[746]**.

Sample application

The sample application is the source code for the McIDAS command MAKTANC, which creates an area with tangent cone navigation. Areas produced with MAKTANC can be used by the REMAP command. The tangent cone map shown earlier was prepared using MAKTANC.

The sample application shows how a navigation block is prepared and inserted into an area. Most of the code fetches user input from the command line and prepares a consistent set of navigation parameters. Five parameters (line and element of pole, standard latitude, standard longitude, and scale) are required. These can either be entered directly using the POLE, SLAT, SLON, and SSCALE keywords (see the help section of the code sample), or computed from the latitude, longitude, and scale at the center of the area being created. Lines **222-247** create the area directory. Lines **253-258** then fill the block with the navigation type and parameters, and lines **260-262** insert the block into the area. Note that the order and scaling of these parameters exactly matches that in the tangent cone initialization module **nvxini**. See lines **17-22** of the sample navigation module.

Integrating navigation modules into McIDAS

When the navigation module is coded, you must incorporate it into McIDAS for testing and use by placing the source code in the proper directory and running the appropriate McIDAS tools.

McIDAS-OS2

Copy the navigation module source code (our example is NVXTANC.DLM) into the \mcidas\working directory and run the following three commands from the OS/2 command line.

```
F   NVXTANC   DL   NAVLIB   NV1TANC
F   NVXTANC   DL   NAVLIB   NV2TANC
F   NVXTANC   DL   NAVLIB   NV3TANC
```

These commands invoke the F.CMD script and produce NV1TANC.DLL, NV2TANC.DLL, and NV3TANC.DLL in \mcidas\user\code. The navigation for TANC codicils is immediately available; it is not necessary to recompile applications.

McIDAS-X

Integrating a user-developed navigation module into McIDAS-X is more complicated because of the lack of dynamic linking. Copy the source NVXTANC.DLM into the mcidas/working directory and run the following procedure from the Unix prompt.

```
fx  nvxtanc dl
```

This procedure generates, compiles, and files in the user library the three source files nvxtanc1.f, nvxtanc2.f, and nvxtanc3.f, each with unique generated names for all function calls and common blocks. Then, you must generate a new nvprep.for with explicit references to the new navigation type by running the two commands below at the Unix prompt.

```
nav_init ~mcidas/mcidas2.1/src/nv*.dlm  nvx*.dlm > nvprep.for
fx nvprep li
```

You must then recompile all applications that use navigation before they can access the new type. Note that ~mcidas/mcidas2.1/src is the directory where the core McIDAS-X source can be found for version 2.1. Adjust this accordingly for other versions of McIDAS-X.

Sample navigation module

The sample navigation module, NVXTANC.DLM is provided below.

```
0001: C   THIS IS SSEC PROPRIETARY SOFTWARE - ITS USE IS RESTRICTED.
0002:
0003: C *** McIDAS Revision History ***
0004: C *** McIDAS Revision History ***
0005:
0006: *$ Name:
0007: *$      nvxini - Initialize navigation for tangent cone projection
0008: *$
0009: *$ Interface:
0010: *$      integer function
0011: *$      nvxini(integer option, integer param(*))
0012: *$
0013: *$ Input:
0014: *$      option  - 1 to set or change projection parameters
0015: *$      option  - 2 set output option
0016: *$      param   - For option 1:
0017: *$                param( 1) = 'TANC'
0018: *$                param( 2) = image line    of pole*10000
0019: *$                param( 3) = image element of pole*10000
0020: *$                param( 4) = km per pixel         *10000
0021: *$                param( 6) = standard latitude    *10000
0022: *$                param( 7) = standard longitude   *10000
0023: *$                for option 2:
0024: *$                param( 1) = 'LL' or 'XYZ'
0025: *$
0026: *$ Input and Output:
0027: *$      none
0028: *$
0029: *$ Output:
0030: *$      none
0031: *$
0032: *$ Return values:
0033: *$       0      - success
0034: *$      -3      - invalid or inconsistent navigation parameters
0035: *$      -4      - invalid navigation parameter type
0036: *$      -5      - invalid nvxini() option
0037: *$
0038: *$ Remarks:
0039: *$          Latitudes and longitudes are in degrees, West positive.
0040: *$      Projection parameters must be in the following ranges:
0041: *$              0. <  standard latitude  <  90.
0042: *$           -180. <= standard longitude < 180.
0043: *$              0. <  scale
0044: *$      Accuracy may suffer near the standard latitude limits.
0045: *$
0046: *$          The projection algorithm is adapted from that in
0047: *$      Saucier, W. J. 1989: Principles of meteorological analysis.
0048: *$        Dover Publications, Inc. 433 pp.
0049: *$
0050: *$ Categories:
0051: *$      navigation
0052:
0053: C      // CODING CONVENTION note: function declarations and common
0054: C      // block declarations are all capitalized to be recognizeable
0055: C      // to script 'convdlm;' this is necessary for a correct build
0056: C      // in MCIDAS-X. For the same reason, one must avoid referring
0057: C      // to function or common block names in upper case elsewhere
0058:
0059:       INTEGER FUNCTION NVXINI(option,param)
0060:
0061:
```

```
0062:       implicit          NONE
0063:
0064:
0065: C     // Interface variables (formal arguments)
0066:
0067:       integer           option          ! initialization option
0068:       integer           param(*)        ! navigation parameters or
0069: C                                       ! output coordinate type
0070:
0071: C     // Local variable definitions
0072:
0073:       character*4       navtyp          ! codicil type
0074:       character*4       outcoord        ! output coordinate type
0075:       real              lat0            ! standard latitude
0076:       character*80      cbuf            ! text output buffer
0077:
0078:
0079:
0080: C     ////////////////////////////////////////////////////////////
0081: C     Common block variables and declaration.
0082:
0083: C     ALL CODE BETWEEN THE '//////' SEPARATORS MUST BE
0084: C     DUPLICATED EXACTLY IN EACH NAVIGATION ROUTINE
0085:
0086: C     (A more maintenance-safe version would use ENTRY points
0087: C     rather than separate functions for the navigation API
0088: C     but entry points cannot be processed by 'convdlm.')
0089:
0090: C     // Common block contents: projection parameters
0091:
0092:       real              Lin0            ! image line of pole
0093:       real              Ele0            ! image element of pole
0094:       real              Scale           ! km per unit image
0095: C                                       ! coordinate (pixel)
0096:       real              Lon0            ! standard longitude
0097:       real              Colat0          ! standard colatitude
0098:
0099: C     // Common block contents: pre-computed intermediate values
0100:
0101:       real              Coscl           ! cosine(Colat0)
0102:       real              Tancl           ! tangent(Colat0)
0103:       real              Tancl2          ! tangent(Colat0/2)
0104:       real              Mxtheta         ! limit of angle from std.
0105: C                                       ! lon on projection surface
0106:
0107: C     // Common block contents: constants
0108:
0109:       real              D2R             ! degrees to radians factor
0110:       real              Pi
0111:       real              Badreal         ! returned when navigation
0112: C                                       ! cannot be done
0113:       real              Erad            ! Earth radius
0114:       logical           Init            ! initialized flag
0115:       logical           Latlon          ! .TRUE. for lat/lon I/O
0116:
0117:
0118:       COMMON/TANC/ Lin0, Ele0, Scale, Lon0, Colat0,
0119:      &  Coscl, Tancl, Tancl2, Mxtheta,
0120:      &  D2R, Pi, Badreal, Erad, Init, Latlon
0121:
0122: C     End of common block variables and declaration.
0123: C     ////////////////////////////////////////////////////////////
0124:
0125:
0126:
0127:
0128: C     // Begin initialization process by setting constants.
0129:
0130:       Erad = 6370.                    ! This value of Erad is ok
0131: C                                     ! for "low-precision" nav
0132: C                                     ! where spherical Earth is
0133: C                                     ! adequate (Saucier, p. 32)
```

```
0134:       Pi         = acos(-1.)
0135:       D2R        = Pi / 180.
0136:
0137:       Badreal    = -1.E10                ! obvious unreasonable value
0138: C                                        ! for nav transform result
0139:
0140: C     // Process initialization options. Only one, initialize
0141: C     // navigation parameters, is supported in this demo version,
0142: C     // but a 'hook' is left for an additional option to set the
0143: C     // output coordinate to something other than lat/lon
0144:
0145:       if( option.eq.1 ) then
0146:
0147:          call DDEST('nvxini(tanc) option=1',0)
0148:
0149:          call movwc(param(1),navtyp)
0150:          if( navtyp.eq.'TANC') then
0151:
0152: C           // Unpack tangent cone projection parameters
0153:
0154:             Lin0         = param(2) / 10000.
0155:             Ele0         = param(3) / 10000.
0156:             Scale        = param(4) / 10000.
0157:             lat0         = param(5) / 10000.
0158:             Lon0         = param(6) / 10000.
0159:
0160:             write(cbuf,'('' nvxini: lat0, Lon0 '',2F12.4)')
0161:      *        lat0, Lon0
0162:             call DDEST(cbuf,0)
0163:
0164: C           // apply range checking
0165:
0166:             if(Scale.le.0. ) then
0167:                call DDEST('nvxini(tanc) scale is negative',0)
0168:                Init      = .FALSE.
0169:                NVXINI    = -3
0170:                return
0171:             end if
0172:
0173:             if(lat0.le.0. .or. lat0.ge.90. ) then
0174:                call DDEST('nvxini(tanc) std lat out of range',0)
0175:                Init      = .FALSE.
0176:                NVXINI    = -3
0177:                return
0178:             end if
0179:
0180:             if(Lon0.le.-180. .or. Lon0.gt.180. ) then
0181:                call DDEST('nvxini(tanc) std lon out of range',0)
0182:                Init      = .FALSE.
0183:                NVXINI    = -3
0184:                return
0185:             end if
0186:
0187: C           // convert degrees to radians and latitude to colat.
0188: C           // Account for McIDAS longitude convention
0189:
0190:             Lon0         = -Lon0 * D2R
0191:             Colat0       =  Pi/2. - D2R*lat0
0192:
0193:             write(cbuf,'('' nvxini:  Colat0, Lon_0 '',2F12.4)')
0194:      *        Colat0, Lon0
0195:             call DDEST(cbuf,0)
0196:
0197: C           // Compute intermediate quantities
0198:
0199:             Coscl        = cos(Colat0)
0200:             Tancl        = tan(Colat0)
0201:             Tancl2       = tan(Colat0/2.)
0202:             Mxtheta      = Pi*Coscl
0203:
0204:             write(cbuf,'('' nvxini: Coscl,   Tancl'', 2F7.4)')
0205:      *        Coscl, Tancl
```

```
0206:              call DDEST(cbuf,0)
0207:
0208:              write(cbuf,'('' nvxini: Tancl2, Mxtheta '', 2F7.4)')
0209:       *        tancl2, Mxtheta
0210:              call DDEST(cbuf,0)
0211:
0212:              Latlon      = .TRUE.
0213:
0214:
0215:           else           ! option=1 but type not 'TANC'
0216:
0217:              call DDEST('nvxini(tanc) parameter type bad',0)
0218:              Init        = .FALSE.
0219:              NVXINI      = -4
0220:              return
0221:
0222:           end if
0223:
0224:        else if ( option .eq. 2) then
0225:
0226:           call movwc(param(1),outcoord)
0227:           if( outcoord.eq.'LL' ) then
0228:               Latlon = .TRUE.
0229:           else if( outcoord.eq.'XYZ') then
0230:               Latlon = .FALSE.
0231:           else
0232:               call DDEST('opt=2 coord '//outcoord//' unknown',0)
0233:               Init   = .FALSE.
0234:               NVXINI = -5
0235:           end if
0236:
0237:        else              ! option not 1 or 2
0238:
0239:           call DDEST('nvxini(tanc) unknown option ',option)
0240:           NVXINI =  -4
0241:           return
0242:
0243:        end if
0244:
0245:        NVXINI     = 0
0246:        Init       = .TRUE.
0247:
0248:        return
0249:        end
0250:
0251:
0252:
0253:
0254: *$ Name:
0255: *$      nvxsae - Compute earth coordinates from image coordinates
0256: *$
0257: *$ Interface:
0258: *$      integer function
0259: *$      nvxsae( real lin, real ele, real dummy,
0260: *$              real e1,  real  e2, real    e3 )
0261: *$
0262: *$ Input:
0263: *$      lin      - image line
0264: *$      ele      - image element
0265: *$      dummy    - (unused)
0266: *$
0267: *$ Input and Output:
0268: *$      none
0269: *$
0270: *$ Output:
0271: *$      e1       - latitude  or x
0272: *$      e2       - longitude or y
0273: *$      e3       - height    or z
0274: *$
0275: *$ Return values:
0276: *$       0       - success
0277: *$      -1       - input data physically valid but not navigable
```

```
0278: *$                  given the specified projection
0279: *$       -6       - module not initialized
0280: *$
0281: *$ Remarks:
0282: *$           The navigation module must first be initialized with
0283: *$       a call to nvxini(). The output form (lat,lon) or (x,y,z)
0284: *$       depends on the last call to nvxini() with option 2.
0285: *$
0286: *$ Categories:
0287: *$       navigation
0288:
0289:       INTEGER FUNCTION NVXSAE( lin, ele, dummy, e1, e2, e3 )
0290:
0291:       implicit          NONE
0292:
0293: C     // Interface variables (formal arguments)
0294:
0295:       real              lin      ! image line to navigate
0296:       real              ele      ! image element to navigate
0297:       real              dummy    ! (unused argument)
0298:       real              e1       ! Earth coordinate 1
0299:       real              e2       ! Earth coordinate 2
0300:       real              e3       ! Earth coordinate 3
0301:
0302: C     // Local variables
0303:
0304:       real              lat      ! latitude  (McIDAS convention)
0305:       real              lon      ! longitude (McIDAS convention)
0306:       real              hgt      ! height
0307:       real              dx       ! zonal displacement from pole
0308: C                               ! on projection surface
0309:       real              dy       ! meridional displacement from pole
0310:       real              radius   ! distance from pole on projection
0311:       real              theta    ! angle from standard longitude on
0312: C                               ! projection surface
0313:       real              colat    ! colatitude of navigated point
0314:
0315:
0316: C     ////////////////////////////////////////////////////////////
0317: C     Common block variables and declaration.
0318:
0319: C     ALL CODE BETWEEN THE '//////' SEPARATORS MUST BE
0320: C     DUPLICATED EXACTLY IN EACH NAVIGATION ROUTINE
0321:
0322: C     (A more maintenance-safe version would use ENTRY points
0323: C     rather than separate functions for the navigation API
0324: C     but entry points cannot be processed by 'convdlm.')
0325:
0326: C     // Common block contents: projection parameters
0327:
0328:       real              Lin0             ! image line of pole
0329:       real              Ele0             ! image element of pole
0330:       real              Scale            ! km per unit image
0331: C                                        ! coordinate (pixel)
0332:       real              Lon0             ! standard longitude
0333:       real              Colat0           ! standard colatitude
0334:
0335: C     // Common block contents: pre-computed intermediate values
0336:
0337:       real              Coscl            ! cosine(Colat0)
0338:       real              Tancl            ! tangent(Colat0)
0339:       real              Tancl2           ! tangent(Colat0/2)
0340:       real              Mxtheta          ! limit of angle from std.
0341: C                                        ! lon on projection surface
0342:
0343: C     // Common block contents: constants
0344:
0345:       real              D2R              ! degrees to radians factor
0346:       real              Pi
0347:       real              Badreal          ! returned when navigation
0348: C                                        ! cannot be done
0349:       real              Erad             ! Earth radius
```

```
0350:       logical           Init            ! initialized flag
0351:       logical           Latlon          ! .TRUE. for lat/lon I/O
0352:
0353:
0354:       COMMON/TANC/ Lin0, Ele0, Scale, Lon0, Colat0,
0355:      &  Coscl, Tancl, Tancl2, Mxtheta,
0356:      &  D2R, Pi, Badreal, Erad, Init, Latlon
0357:
0358: C     End of common block variables and declaration.
0359: C     ////////////////////////////////////////////////////////////////
0360:
0361:       e1        = Badreal
0362:       e2        = Badreal
0363:       e3        = Badreal
0364:
0365:
0366: C     // verify initialized module
0367:
0368:       if(.not.Init) then
0369:           NVXSAE = -6
0370:           return
0371:       end if
0372:
0373:
0374: C     // Compute radius and bearing from pole
0375:
0376:       dx        = Scale*(lin-Lin0)
0377:       dy        = Scale*(ele-Ele0)
0378:
0379:       radius    = sqrt(dx*dx+dy*dy)
0380:       theta     = atan2(dy,dx)
0381:
0382:
0383: C     // Range check theta to determine if point is navigable
0384:
0385:       if ( theta.le.-Mxtheta .or. theta.gt.Mxtheta ) then
0386:          NVXSAE = -1
0387:          return
0388:       end if
0389:
0390: C     // Forward navigation: compute longitude and colatitude
0391: C     // from radius and theta
0392:
0393:       lon       = Lon0 + theta/Coscl
0394:       if(lon.le.-Pi) lon = lon + 2.d0*Pi
0395:       if(lon.gt. Pi) lon = lon - 2.d0*Pi
0396:
0397:       colat = 2.*atan( Tancl2 * (radius/(Erad*Tancl))**(1./Coscl))
0398:
0399: C     // Rescale to McIDAS convention (degrees, West positive).
0400: C     // Apply conversion to Cartesian coordinates if 'XYZ' set
0401: C     // as output form. Set return code for success.
0402:
0403:       lon       = -lon/D2R
0404:       lat       = 90. - colat/D2R
0405:       hgt       = 0.
0406:
0407:       if(.not.Latlon) then
0408:           call nllxyz(lat,lon,e1,e2,e3)
0409:       else
0410:           e1 = lat
0411:           e2 = lon
0412:           e3 = 0.
0413:       end if
0414:
0415:       NVXSAE    = 0
0416:
0417:       return
0418:       end
0419:
0420:
0421:
```

```
0422:
0423: *$ Name:
0424: *$      nvxeas - Compute image coordinates from earth coordinates
0425: *$
0426: *$ Interface:
0427: *$      integer function
0428: *$      nvxeas( real e1, real e2, real e3,
0429: *$              real lin, real ele, real dummy)
0430: *$
0431: *$ Input:
0432: *$      e1       - latitude  or x
0433: *$      e2       - longitude or y
0434: *$      e3       - height    or z
0435: *$
0436: *$ Input and Output:
0437: *$      none
0438: *$
0439: *$ Output:
0440: *$      lin      - image line
0441: *$      ele      - image element
0442: *$      dummy    - (unused)
0443: *$
0444: *$ Return values:
0445: *$       0       - success
0446: *$      -1       - input data physically valid but not navigable
0447: *$                 given the specified projection
0448: *$      -2       - input data exceed physical limits
0449: *$      -6       - module not initialized
0450: *$
0451: *$ Remarks:
0452: *$           The navigation module must first be initialized with
0453: *$      a call to nvxini(). The input form (lat,lon) or (x,y,z)
0454: *$      depends on the last call to nvxini() with option 2.
0455: *$          Input longitude may be in the range -360 to +360;
0456: *$      values outside this range will not be denavigated.
0457: *$      Height (hgt) is ignored.
0458: *$
0459: *$ Categories:
0460: *$      navigation
0461:
0462:       INTEGER FUNCTION NVXEAS( e1, e2, e3, lin, ele, dummy)
0463:
0464:       implicit          NONE
0465:
0466: C     // Interface variables (formal arguments)
0467:
0468:       real              e1       ! Earth coordinate 1
0469:       real              e2       ! Earth coordinate 2
0470:       real              e3       ! Earth coordinate 3
0471:       real              lin      ! image line to navigate
0472:       real              ele      ! image element to navigate
0473:       real              dummy    ! (unused argument)
0474:
0475: C     // Local variables
0476:
0477:       real              lat      ! latitude  (McIDAS convention)
0478:       real              lon      ! longitude (McIDAS convention)
0479:       real              hgt      ! height
0480:       real              in_lon   ! input longitude (radians,
0481: C                               ! East positive)
0482:       real              colat    ! colatitude
0483:       real              radius   ! distance from pole on projection
0484:       real              theta    ! angle from standard longitude on
0485: C                               ! projection surface
0486:
0487: C     //////////////////////////////////////////////////////////////
0488: C     Common block variables and declaration.
0489:
0490: C     ALL CODE BETWEEN THE '//////' SEPARATORS MUST BE
0491: C     DUPLICATED EXACTLY IN EACH NAVIGATION ROUTINE
0492:
0493: C     (A more maintenance-safe version would use ENTRY points
```

```
0494: C     rather than separate functions for the navigation API
0495: C     but entry points cannot be processed by 'convdlm.')
0496:
0497: C     // Common block contents: projection parameters
0498:
0499:       real              Lin0            ! image line of pole
0500:       real              Ele0            ! image element of pole
0501:       real              Scale           ! km per unit image
0502: C                                       ! coordinate (pixel)
0503:       real              Lon0            ! standard longitude
0504:       real              Colat0          ! standard colatitude
0505:
0506: C     // Common block contents: pre-computed intermediate values
0507:
0508:       real              Coscl           ! cosine(Colat0)
0509:       real              Tancl           ! tangent(Colat0)
0510:       real              Tancl2          ! tangent(Colat0/2)
0511:       real              Mxtheta         ! limit of angle from std.
0512: C                                       ! lon on projection surface
0513:
0514: C     // Common block contents: constants
0515:
0516:       real              D2R             ! degrees to radians factor
0517:       real              Pi
0518:       real              Badreal         ! returned when navigation
0519: C                                       ! cannot be done
0520:       real              Erad            ! Earth radius
0521:       logical           Init            ! initialized flag
0522:       logical           Latlon          ! .TRUE. for lat/lon I/O
0523:
0524:
0525:       COMMON/TANC/ Lin0, Ele0, Scale, Lon0, Colat0,
0526:      &  Coscl, Tancl, Tancl2, Mxtheta,
0527:      &  D2R, Pi, Badreal, Erad, Init, Latlon
0528:
0529: C     End of common block variables and declaration.
0530: C     ////////////////////////////////////////////////////////////////
0531:
0532:
0533:       lin       = Badreal
0534:       ele       = Badreal
0535:       dummy     = Badreal
0536:
0537: C     // verify that module is initialized
0538:
0539:       if(.not.init) then
0540:           NVXEAS = -6
0541:           return
0542:       end if
0543:
0544: C     // Preprocess input values. If mode is 'XYZ' first convert
0545: C     // from Cartesian to lat/lon. If mode is 'LL' just transcribe
0546: C     // from arguments.
0547:
0548:       if(Latlon) then
0549:           lat = e1
0550:           lon = e2
0551:           hgt = e3
0552:       else
0553:           call nxyzll( e1, e2, e3, lat, lon)
0554:           hgt = 0.
0555:       end if
0556:
0557: C     // check that input values are physically possible and
0558: C     // then convert to radians and East positive
0559:
0560:       if ( lat.lt.-90. .or. lat.gt.90. ) then
0561:          NVXEAS = -2
0562:          return
0563:       end if
0564:
0565:       if( lon.le.-360..or.lon.gt.360.) then
```

```
0566:             NVXEAS = -2
0567:             return
0568:         end if
0569:
0570:         if( lat.eq.-90. .or. lat.eq.90. ) then
0571:             NVXEAS = -1
0572:             return
0573:         end if
0574:
0575:         colat      = Pi/2. - D2R*lat
0576:         in_lon     = -D2R*lon
0577:
0578: C       // map longitude into range -Pi to Pi
0579:
0580:         if(in_lon.le.-Pi) in_lon = in_lon + 2.*Pi
0581:         if(in_lon.gt. Pi) in_lon = in_lon - 2.*Pi
0582:
0583:
0584: C       // Now trap South Pole. Though a valid latitude,
0585: C       // tan(colat/2) -> infinity there so it is not navigable
0586:
0587:         if ( colat.eq.Pi ) then
0588:           NVXEAS  = -1
0589:           return
0590:         end if
0591:
0592:
0593: C       // Compute radius and theta of point on projection surface.
0594: C       // Theta is tricky; you have to compute offset relative
0595: C       // to standard longitude, force that into -pi to +pi range,
0596: C       // and THEN scale by cos(Colat0)
0597:
0598:         radius     = Erad * Tancl *( tan(colat/2.)/Tancl2 ) ** Coscl
0599:         theta      = in_lon-Lon0
0600:
0601:         if(theta.le.-Pi) theta = theta + 2.*Pi
0602:         if(theta.gt. Pi) theta = theta - 2.*Pi
0603:
0604:         theta      = Coscl * theta
0605:
0606:
0607: C       // Compute line and element
0608:
0609:         lin        = Lin0 + radius*cos(theta)/Scale
0610:         ele        = Ele0 + radius*sin(theta)/Scale
0611:         dummy      = 0.
0612:
0613:         NVXEAS     = 0
0614:
0615:         return
0616:         end
0617:
0618:
0619:
0620: *$ Name:
0621: *$       nvxopt - Perform supplemental navigation operations
0622: *$
0623: *$ Interface:
0624: *$       integer function
0625: *$       nvxopt(integer option, real xin(*),
0626: *$         real xout(*) )
0627: *$ Input:
0628: *$       option  - 'SCAL' compute projection scale
0629: *$       xin(1)  - latitude
0630: *$
0631: *$ Input and Output:
0632: *$       none
0633: *$
0634: *$ Output:
0635: *$       xout(1) - km per pixel at given latitude
0636: *$
0637: *$ Return values:
```

```
0638: *$       0       - success
0639: *$      -1       - input latitude physically valid, but projection
0640: *$                 undefined or scale infinite there
0641: *$      -2       - input latitude exceeds physical limits
0642: *$      -5       - unrecognized option
0643: *$      -6       - module not initialized
0644: *$
0645: *$ Remarks:
0646: *$          The navigation module must first be initialized by
0647: *$      a call to nvxini().  Latitude is in degrees, north positive,
0648: *$      and must lie between -90. and +90.
0649: *$
0650: *$ Categories:
0651: *$      navigation
0652:
0653:       INTEGER FUNCTION NVXOPT( option, xin, xout)
0654:
0655:       implicit          NONE
0656:
0657: C     // Interface variables (formal arguments)
0658:
0659:       integer           option  ! special service name (character
0660: C                               ! stored as integer)
0661:       real              xin(*)  ! input vector
0662:       real              xout(*) ! output vector
0663:
0664: C     // Local variables
0665:
0666:       character*4       copt    ! special service (character form)
0667:       real              colat   ! input colatitude
0668:
0669:
0670: C     ////////////////////////////////////////////////////////////
0671: C     Common block variables and declaration.
0672:
0673: C     ALL CODE BETWEEN THE '//////' SEPARATORS MUST BE
0674: C     DUPLICATED EXACTLY IN EACH NAVIGATION ROUTINE
0675:
0676: C     (A more maintenance-safe version would use ENTRY points
0677: C     rather than separate functions for the navigation API
0678: C     but entry points cannot be processed by 'convdlm.')
0679:
0680: C     // Common block contents: projection parameters
0681:
0682:       real              Lin0            ! image line of pole
0683:       real              Ele0            ! image element of pole
0684:       real              Scale           ! km per unit image
0685: C                                       ! coordinate (pixel)
0686:       real              Lon0            ! standard longitude
0687:       real              Colat0          ! standard colatitude
0688:
0689: C     // Common block contents: pre-computed intermediate values
0690:
0691:       real              Coscl           ! cosine(Colat0)
0692:       real              Tancl           ! tangent(Colat0)
0693:       real              Tancl2          ! tangent(Colat0/2)
0694:       real              Mxtheta         ! limit of angle from std.
0695: C                                       ! lon on projection surface
0696:
0697: C     // Common block contents: constants
0698:
0699:       real              D2R             ! degrees to radians factor
0700:       real              Pi
0701:       real              Badreal         ! returned when navigation
0702: C                                       ! cannot be done
0703:       real              Erad            ! Earth radius
0704:       logical           Init            ! initialized flag
0705:       logical           Latlon          ! .TRUE. for lat/lon I/O
0706:
0707:
0708:       COMMON/TANC/ Lin0, Ele0, Scale, Lon0, Colat0,
0709:      &  Coscl, Tancl, Tancl2, Mxtheta,
```

```
0710:      &  D2R, Pi, Badreal, Erad, Init, Latlon
0711:
0712: C     End of common block variables and declaration.
0713: C     ////////////////////////////////////////////////////////////////
0714:
0715:
0716:       xout(1)    = Badreal
0717:
0718: C     // verify initialized module
0719:
0720:       if(.not.init) then
0721:           NVXOPT = -6
0722:           return
0723:       end if
0724:
0725: C     // Extract and interpret the option
0726:
0727:       call movwc(option,copt)
0728:
0729:       if(copt.eq.'SCAL') then
0730:
0731: C        //  Compute colatitude and make sure it is
0732: C        //  physically possible and navigable
0733:
0734:          if ( xin(1).gt.90. .or. xin(1).lt.-90. ) then
0735:              NVXOPT = -2
0736:              return
0737:          else if ( xin(1).eq.90. .or. xin(1).eq.-90. ) then
0738:              NVXOPT = -1
0739:              return
0740:          end if
0741:
0742:          colat    = Pi/2. - D2R*xin(1)
0743:
0744: C        // Now compute actual scale for this colatitude
0745:
0746:          xout(1) = scale
0747:      *            *(sin(Colat0)*(tan(colat/2.)/Tancl2)**Coscl)/sin(colat)
0748:
0749: C     else if(copt.eq.'????')
0750: C        // Add code for additional options here
0751:
0752:       else
0753:          NVXOPT = -5
0754:          return
0755:       end if
0756:
0757:       NVXOPT = 0
0758:
0759:       return
0760:       end
```

Sample application

A sample application, MAKTANC.PGM, which creates an area with attached TANC navigation, is provided below.

```
0001: C   THIS IS SSEC PROPRIETARY SOFTWARE - ITS USE IS RESTRICTED.
0002:
0003: C *** McIDAS Revision History ***
0004: C *** McIDAS Revision History ***
0005:
0006: C ? MAKTANC -- Create test area with tangent cone navigation
0007: C ?    MAKTANC area lat lon <keywords>
0008: C ? Parameters:
0009: C ?    area    | area number to create
0010: C ?    lat     | latitude at center  (def compute from POLE=)
0011: C ?    lon     | longitude at center (def compute from POLE=)
0012: C ? Keywords:
0013: C ?    POLE    | lin ele     Image location of North Pole
0014: C ?            | (def compute from lat,lon)
0015: C ?    SLAT    | standard latitude  (def=lat)
0016: C ?    SLON    | standard longitude (def=lon)
0017: C ?    SSCALE  | scale (km per pixel) at standard latitude
0018: C ?    --or--  |
0019: C ?    LSCALE  | scale (km per pixel) at center latitude
0020: C ? Remarks:
0021: C ?       The standard latitude must be between 0 and 90, exclusive.
0022: C ? ----------
0023:
0024:         subroutine main0
0025:
0026:         implicit        NONE
0027:
0028: C       // Parameters
0029:
0030:         integer         NKEYS, MXANUM, NVWDS, DIRSIZ
0031:         parameter       ( NKEYS=5, MXANUM=9999, NVWDS=128 )
0032:         parameter       ( DIRSIZ=64 )
0033:         integer         PFXLEN, CALLEN, LEVLEN
0034:         parameter ( PFXLEN=0, CALLEN=0, LEVLEN=0 )
0035:         double precision        PI, D2R, A
0036:         parameter       ( A=6370.D0 )
0037:
0038: C       // Local variables
0039:
0040:         character*12    keys(NKEYS)     ! keywords
0041:         logical         cpoint          ! center point specified ?
0042:         integer         slat_type       ! standard latitude  source
0043:         integer         slon_type       ! standard longitude source
0044:         integer         aranum          ! area number
0045:         integer         nvblk(NVWDS)    ! navigation block
0046:         integer         adir(DIRSIZ)    ! area directory
0047:         integer         i               ! loop index
0048:         double precision        clat    ! center latitude
0049:         double precision        clon    ! center longitude
0050:         double precision        slat    ! standard latitude
0051:         double precision        slon    ! standard longitude
0052:         double precision        sscale  ! scale at std latitude
0053:         double precision        lscale  ! scale at center latitude
0054:         double precision        ratio   ! scale ratio center:std
0055:         double precision        lin_0   ! image line    of pole
0056:         double precision        ele_0   ! image element of pole
0057:         double precision        lin_c   ! line of center
0058:         double precision        ele_c   ! element of center
0059:
0060:
```

```
0061:          double precision        psi_0    ! standard colatitude
0062:          double precision        psi      ! colatitude
0063:          double precision        lam_0    ! standard longitude
0064:          double precision        lam      ! standard longitude
0065:          double precision        radius   ! radius on projection
0066:          double precision        theta    ! bearing on projection
0067:
0068:
0069:
0070:
0071: C        // External functions
0072:
0073:          character*12    cfi             ! integer to string
0074:          integer         luc             ! User Common peek
0075:          integer         lit             ! integer from character*4
0076:          integer         mccmdkey        ! validate keywords
0077:          integer         mccmdnum        ! # values with keyword
0078:          integer         mccmddll        ! fetch latitude/longitude
0079:          integer         mccmddbl        ! fetch double precision
0080:          integer         mccmdint        ! fetche integer
0081:
0082: C        // Initialization ---------------------------------------
0083:
0084:          data keys/'P.OLE','SLA.T','SLO.N','SS.CALE','LS.CALE'/
0085:
0086:
0087:          PI      = dacos(-1.d0)
0088:          D2R     = PI/180.d0
0089:
0090:
0091: C        // Validate key words -----------------------------------
0092:
0093:          if( mccmdkey(NKEYS, keys) .lt. 0 ) return
0094:
0095:          if( mccmdnum(keys(4)).gt.0.and.mccmdnum(keys(5)).gt.0) then
0096:              call edest('Please use either SSCALE or LSCALE '//
0097:       &         'to set map scale',0)
0098:              return
0099:          end if
0100:
0101:
0102: C        // Fetch command line arguments -------------------------
0103:
0104: C        // If area already exists, shut down now. If not, create
0105: C        // an area with the specified number
0106:
0107:          if( mccmdint(' ', 1, 'Number of area to create',
0108:       $    1, 1, MXANUM, aranum).ne.1100) then
0109:              call edest('You must specify area number to create',0)
0110:              return
0111:          end if
0112:
0113:          call readd( aranum, adir )
0114:          if(adir(1).eq.0) then
0115:              call edest('Area '//cfi(aranum)//' already exists',0)
0116:              call edest('Please delete it or use another number',0)
0117:              return
0118:          end if
0119:
0120: C        // Fetch center latitude and longitude, if available
0121:
0122:          if( ( mccmddll(' ', 2, 'Center latitude',
0123:       $      0.d0,  -90.d0,  90.d0, clat).gt.1000 ) .and.
0124:       $      ( mccmddll(' ', 3, 'Center longitude',
0125:       $      0.d0, -180.d0, 180.d0, clon).gt.1000 ) ) then
0126:
0127:              cpoint = .TRUE.
0128:          else
0129:              cpoint = .FALSE.
0130:          end if
0131:
0132:
```

```
0133: C       // Fetch standard latitude and longitude. If center point
0134: C       // was entered, use it as default. Otherwise, require that
0135: C       // a value be entered. Determine this by comparing 'cpoint'
0136: C       // flag and the return codes from mccmddll(). If cpoint is
0137: C       // not set, actual values (not the defaults) must have been
0138: C       // fetched for SLAT and SLON
0139:
0140:         slat_type = mccmddll(keys(2), 1, 'Standard latitude',
0141:      $     clat, 0.d0, 90.0d0, slat)
0142:         slon_type = mccmddll(keys(3), 1, 'Standard longitude',
0143:      $     clon, -180.d0, 180.d0, slon)
0144:
0145:         if( .not.cpoint .and.
0146:      $    (slat_type.ne.1550 .or. slon_type.ne.1550) ) then
0147:             call edest('You must specify either a center point'//
0148:      $        ' or SLAT= and SLON=',0)
0149:             return
0150:         end if
0151:         if( slat.le.0.d0 .or. slat.ge.90.d0 ) then
0152:             call edest('SLAT must be BETWEEN 0 and 90',0)
0153:             if(cpoint) then
0154:                 call edest('If you want a center point outside',0)
0155:                 call edest('this range, you must use SLAT=, too',0)
0156:             end if
0157:             return
0158:         end if
0159:
0160:         psi_0 = PI/2.d0 - D2R*slat
0161:
0162:
0163: C       // Fetch scale. A default is available for the absolute
0164: C       // (standard latitude) scale. If local scale is specified,
0165:
0166:         if(mccmddbl(keys(4), 1, 'Scale (km/pixel at std lat)',
0167:      $        1.d0, 0.001d0, 1000.d0, sscale).lt.0) return
0168:         if(cpoint .and. mccmddbl(keys(5), 1,
0169:      $    'Scale at center latitude', 1.d0, 0.001d0, 1000.d0,
0170:      $     lscale).eq.1200) then
0171:
0172: C           // Compute sscale from lscale, using standard latitude
0173:
0174:             psi = PI/2.d0 - D2R*clat
0175:             ratio = (tan(psi/2.d0)/tan(psi_0/2.d0))**cos(psi_0) *
0176:      $              sin(psi_0) / sin(psi)
0177:             sscale = lscale*ratio
0178:
0179:         end if
0180:
0181:
0182: C       // If a center point was not specified, fetch the location
0183: C       // of the pole. Otherwise, compute it
0184:
0185:         if( .not.cpoint ) then
0186:
0187:             if( mccmddbl(keys(1), 1, 'Image line of pole',0.d0,
0188:      $        1.d0, 0.d0, lin_0) .lt.0 ) return
0189:             if( mccmddbl(keys(1), 2, 'Image element of pole',0.d0,
0190:      $        1.d0, 0.d0, ele_0) .lt.0 ) return
0191:
0192:         else
0193:
0194:             psi = PI/2.d0 - D2R*clat
0195:             lam =         - D2R*clon
0196:             lam_0  =      - D2R*slon
0197:
0198:
0199:             radius = A * tan(psi_0) *
0200:      $               (tan(psi/2.d0)/tan(psi_0/2.d0))**cos(psi_0)
0201:             theta  = cos(psi_0)*(lam-lam_0)
0202:
0203:
0204:             lin_c  = dble(luc(11)) / 2.d0
```

```
0205:             ele_c  = dble(luc(12)) / 2.d0
0206:
0207:
0208:             lin_0  = lin_c - radius*cos(theta)/sscale
0209:             ele_0  = ele_c - radius*sin(theta)/sscale
0210:
0211:         end if
0212:
0213:
0214: C       // Now make the area ------------------------------------
0215:
0216: C       // Fill in the area directory and create the area
0217:
0218:         do i=1,DIRSIZ
0219:             adir(i) = 0
0220:         end do
0221:
0222:         adir( 1) =  0                     ! existence flag
0223:         adir( 2) =  4                     ! format (interleaved)
0224:         adir( 3) =  1                     ! satellite ID (test area)
0225:         call getday(adir(4))              ! today's date
0226:         adir( 5) =  0                     ! hour 0
0227:         adir( 6) =  1                     ! image line    upper left
0228:         adir( 7) =  1                     ! image element upper left
0229:         adir( 9) =  luc(11)               ! # lines     (TV size)
0230:         adir(10) =  luc(12)               ! # elements (TV size)
0231:         adir(11) =  1                     ! bytes per element
0232:         adir(12) =  1                     ! line    resolution
0233:         adir(13) =  1                     ! element resolution
0234:         adir(14) =  1                     ! number of bands
0235:         adir(15) =  0                     ! y-z prefix (0 for image)
0236:         adir(16) =  luc(1)                ! project number logged on
0237:         call getday(adir(17))             ! creation date
0238:         call gettim(adir(18))             ! creation time
0239:         adir(19) =  1                     ! set band 1 (least bit)
0240:         adir(34) = 4*DIRSIZ + 4*NVWDS     ! nav block end (bytes)
0241:         adir(35) = 4*DIRSIZ               ! nav block start (bytes)
0242:         adir(36) =  0                     ! validity code
0243:         adir(49) = PFXLEN                 ! prefix length, bytes
0244:         adir(50) = CALLEN                 ! prefix cal length, bytes
0245:         adir(51) = LEVLEN                 ! prefix lev length, bytes
0246:         adir(52) = lit('VISR')            ! sensor type
0247:         adir(53) = lit('BRIT')            ! calibration type
0248:
0249:         call makara ( aranum, adir )
0250:
0251: C       // Create and insert the navigation block
0252:
0253:         nvblk( 1) = lit('TANC')           ! Tangent Cone nav type
0254:         nvblk( 2) = nint(10000*lin_0)     ! image line    of pole
0255:         nvblk( 3) = nint(10000*ele_0)     ! image element of pole
0256:         nvblk( 4) = nint(10000*sscale)    ! scale km/pixel at slat
0257:         nvblk( 5) = nint(10000*slat)      ! standard latitude
0258:         nvblk( 6) = nint(10000*slon)      ! standard longitude
0259:
0260:         call araput(aranum,4*DIRSIZ,4*NVWDS,nvblk)
0261:         call stamp(aranum)
0262:         call clsara(aranum)
0263:
0264:
0265:         call sdest('MAKTANC done!',0)
0266:
0267:         return
0268:         end
```

Developing Local Decoders in McIDAS-XCD

Presented by

John Pyeatt

MUG Program Manager

Session 8

McIDAS Developer/Operator Training

October 23-25, 1995

Table of Contents

Overview 8-1
Terminology 8-1
Ingestors currently supported in McIDAS-XCD 8-2
Decoders currently supported in McIDAS-XCD 8-2
Decoders currently under development 8-3

What does a decoder typically do? 8-4

Questions to ask before writing a decoder 8-7

Other McIDAS-XCD subsystems 8-18
Gross error checking system 8-18
Configuration file system 8-19
Decoder status display 8-20
Station monitoring 8-21
DECTEST 8-21

Compiling and linking 8-22

Integrating a local decoder in McIDAS-XCD 8-23
Building the decoder and putting the data in MD files 8-23
Automating the decoder process 8-27

Developing local decoders for your site 8-29

Useful functions in McIDAS-XCD 8-30

Sample decoder 8-33

Overview

This training session will provide McIDAS software developers with useful information for writing and maintaining locally developed decoders in conjunction with the McIDAS-XCD software package.

The Nested Grid Model's trajectory forecast reports are used as examples for writing data decoders. You will find a sample decoder for this data type at the end of this section. Values placed in brackets '[ ]' throughout this section reference specific lines of source code provided in the sample decoder.

Terminology

The following terms are used throughout this section.

data block	text data containing a WMO header and a data section
data monitor	McIDAS command that periodically tests to see if new data has arrived that may be important for a specific decoder
data section	portion of text containing independent observations
DDS	Domestic Data Service
decoder	software that parses data from one format into a common format for use by another process such as a plotter or lister, or software that further manipulates data
HRS	High Resolution data Service
IDS	International Data Service
ingestor	process that listens to data received by a communications port and reformats the information for further processing
NGM	National Meteorological Center Nested Grid Model
observation	one complete, independent report about the state of a given group of fields in a specific time or time range

parsing	decomposition of an observation into its most elementary parts
PPS	Public Products Service
STARTXCD	mother task to the entire McIDAS-XCD ingestor/ decoder package
status display	X Windows application that informs an operator of the current state of the McIDAS-XCD decoder/ ingestor system
token	smallest entity to which an observation may be parsed
WMO header	WMO formatted string containing a product code, product number, originating station and day/time stamp; for example: FOUS14 KWBC 231200

Ingestors currently supported in McIDAS-XCD

- National Weather Service Family of Services: DDS, IDS, PPS, HRS
- National Weather Service AFOS
- AFGWC (Tinker AFB)

Decoders currently supported in McIDAS-XCD

- Surface Hourly SAO and METAR
- RAOB (TEMP): TTAA, TTBB, TTDD, PPAA and PPBB
- Ship and Buoy
- FOUS14
- SYNOP
- AIREP/PIREP
- POES Navigation
- Severe Weather Watch
- GRIB (for HRS circuit)

Decoders currently under development

- Severe Weather by County Federal Information Processing Standards (FIPS)
  - Flood Watch
  - Severe Thunderstorm Watch
  - Tornado Watch
  - Flood Warning
  - Severe Thunderstorm Warning
  - Tornado Warning
- New SAO/METAR Format
- New Aerodrome Forecast (TAF) format
- Expanded data support for GRIB format

What does a decoder typically do?

A decoder goes through approximately seven steps when processing a data type. These steps may include the following:

1. isolating the observation
2. extracting the observation's location
3. extracting the observation time
4. parsing the observation into small enough components so that the information can be interpreted by the software
5. decoding the parsed reports
6. passing the decoded information to the appropriate subsystem for availability to the user
7. writing information to the status display

Each step is described below.

Isolating an observation

When writing decoders that process text data, one of the most difficult tasks is to isolate an individual observation from the data block. Many data types contain observations from several locations in one data block. Determining where one observation ends and the next begins can be a frustrating task. Ideally, a starting character or termination character separates observations.

A starting character is usually the record separator (Hex 1E). If you scan through a data block and encounter a hex 1E, you can be quite certain the characters that follow are a new observation.

If no record separator exists at the beginning of an observation, look for a character indicating the end of the observation. This character may vary depending on the data type. SAO or METAR reports will most likely have an equals sign (=) at the end. Terminal forecasts may be followed by two periods (..).

Be careful. Many overseas reports don't have separation characters and you cannot ignore this problem. If some type of contingent plan is not in the code, you may find erroneous values in your final dataset. For example, several times during its development for the McIDAS-XCD package, the SYNOP decoder produced reports of 400 inches of snow in parts of Iraq. This occurred because the decoder was not prepared for observations coming in without record separators or termination characters.

Extracting the observation's location

Once you isolate an observation, you must determine its origin, such as station ID for METAR reports or station block number for SYNOP reports. This is simple if certain flags are present in the report to help you isolate the section of the data containing the station origin. It may get more complicated with reports that are more free-format.

When you have the origin, you can scan a list of known stations looking for latitude, longitude and elevation information. For some data types, you must determine the latitude and longitude of the observation directly from the report. This is straightforward for reports such as DRIBUs or AIREPs, but can be difficult for a PIREP, which contains no definitive reporting format (or worse, multiple reporting formats).

Once you find the station ID, the decoder can determine if it should continue processing. If you don't have enough meta-data about the station, (latitude, longitude, state, etc.) there is no need to continue processing. In this case, you should file this station in the new stations list with the ID monitoring system **[A565]** and move on to the next report.

Extracting the observation time

Extracting the observation time is an easy task. The format is rigid, with simple checks to perform **[A434-A464]**. The only difficulty is when you must use the time report to validate the date of the observation. For example, if a real-time decoder is processing DRIBU reports at 1845 UTC today and an observation comes in with a reporting time of 2130 UTC, your decoder must be robust enough to recognize that the report is really from yesterday. You can use the McIDAS-XCD function **mcfiltim** to simplify this logic.

The observation time is often used to determine where in a data structure an observation should be filed. For example, in MD files, the time determines the appropriate row number for filing data.

Parsing the observation

Once the location and time are known, the decoder breaks the report into individual tokens based on the reporting format. The method of parsing varies among data types. Typically, the three formats are: tabular, formatted and plain text. These will be discussed later in this session.

Decoding the parsed reports

Decoding occurs once the raw text is parsed. This is where all the real work takes place. It is a simple process for tabular data **[A1099-A1167]**. The only time tabular reports are problematic is when a signal goes bad. Problems with formatted or plain text reports are usually due to human error, such as typos, incorrectly placed data, or meteorologically incorrect values. A decoder must be smart enough not to abort, and to recover successfully from reporting errors. The gross error checking system, described later, is the final place to resolve problems in reports **[A705-A709]**.

Passing the decoded information to the appropriate subsystem

After the information in an observation is fully interpreted by the decoder, it is usually stored in some type of disk file format that users can access. In McIDAS, the most common storage format is MD or grid files. After decoding the report, the software must perform a transformation on the data to put it in a format that is acceptable for the data file structure being used. This transformation may include converting units of the report, changing values from floating point to scaled integers, or converting character strings into string literals **[A671-A716]**.

Writing information to the status display

The final step performed by a decoder is to update the status display **[A792-A800]**. This display provides the mechanism for operations to monitor the system. Information of interest to the operations staff might include: the time data was last processed by this decoder; the last MD file, row and column written to; the last grid file and grid number written to; or other information unique to this data type.

Questions to ask before writing a decoder

Before you write your decoders, answer the questions below.

What are the characteristics of the data?

1. Is the format of the data to be decoded binary or text?
2. If text, is its format tabular, formatted or plain?
3. Does the information come in continually or sporadically?

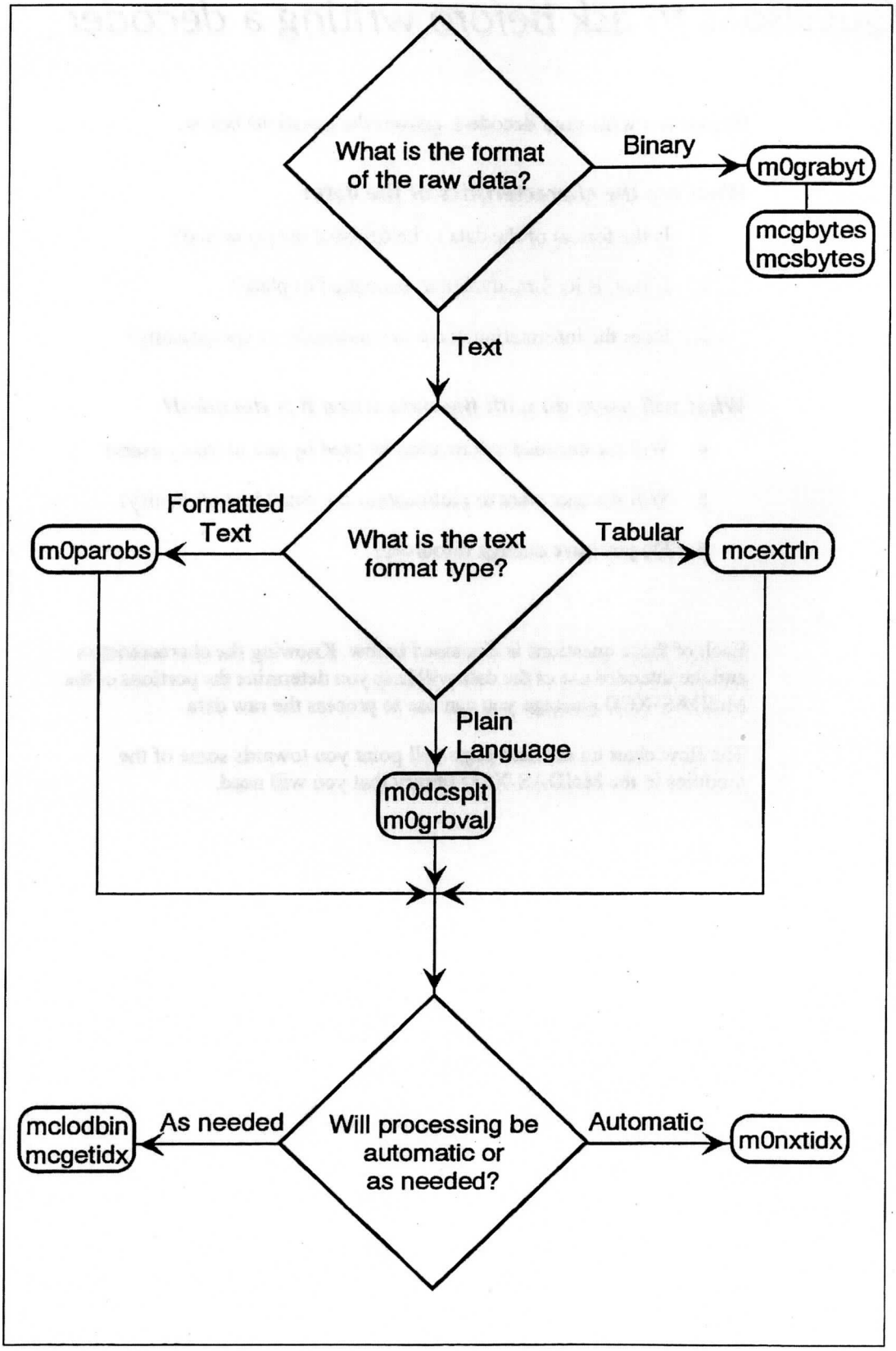
What will users do with the data when it is decoded?

4. Will the decoded information be used by one or many users?
5. Will the user want to plot/contour the data? List it? Notify?
6. Do you have enough resources?

Each of these questions is discussed below. Knowing the characteristics and the intended use of the data will help you determine the portions of the McIDAS-XCD package you can use to process the raw data.

The flow chart on the next page will point you towards some of the modules in the McIDAS-XCD library that you will need.

McIDAS-XCD library modules for acquiring and decoding raw data



Is the format of the data to be decoded binary or text?

Binary

Binary data is the easiest format to decode. A binary format has very rigid standards, so there is little chance for human error. If something is wrong with the signal, it is usually not subtle. The most important concern when writing decoders for binary data is to fireproof the decoder from a bad signal. Fireproofing includes protection against array overwrites and obvious invalid meteorological values.

If the data is a binary stream received through a communications port, you must activate the INGEBIN ingestor for the signal so McIDAS-XCD can decode it. To activate a binary data ingestor in McIDAS-XCD you must perform the three steps below.

1. Create a configuration file containing the information specific for the circuit. This information includes the baud rate, the communications port used, and the destination file in which to store the data. If you use the configuration file for the HRS circuit (HRS.CFG) as a template, you will modify the values for IBAUD, OBAUD, PORT and SPOOL.

2. Add the ingestor to the list of ingestors that McIDAS-XCD monitors. From the McIDAS command window, enter the command below, replacing *name* with the circuit name and *confil* with the configuration file containing the communications information.

   CIRCUIT ADD *name* **CONFIG=***confil* **INGESTOR=INGEBIN**

3. Activate the ingestor. From the McIDAS command window, enter the command below.

   CIRCUIT SET *name* **ACTIVE**

   Every 30 seconds, STARTXCD checks to see if processes were activated, inactivated or aborted. Since the new circuit was just activated, STARTXCD will start up the ingestor for you and the ingestor will file the data into the spool file designated in the configuration file.

Once the data is placed in the spool file by INGEBIN, you can access it with the McIDAS-XCD function **m0grabyt**. This is a very useful interface for processing real-time binary data.

If the signal contains packed data, use the McIDAS-XCD functions **Mcupackbit** and **mcgbytes** to extract packed values.

Text

If the data is an ASCII stream, decoding gets more complicated. The interfaces for text data in McIDAS-XCD expect the data to begin with a WMO header followed by *n* lines of text.

The form for the WMO header is: *ppcc## orig ddhhmm*
where: *pp* product code
cc country code
numeric code
orig station origin
dd day of the month
hhmm time stamp

For example: FOUS14 KWBC 241725

For this training session, assume the data is already filed in the appropriate raw format.

Tabular text

If the text is a computer generated table, such as FOUS14, it is relatively easy to decode. Use the McIDAS-XCD function **mcextrln** to extract information **[A1103-A1104]**, as shown below.

```
                          1         2         3
column number:   12345678901234567890123456789012345678
sample string = '68          79         SNOW      45.3'

character*12 string
double precision fltval
integer intval

ok = mcextrln(line,1,2,string,strsta,intval,intsta,fltval,fltsta)
intval returns 68
fltval returns 68.0
string returns '68'
all statuses return success

ok = mcextrln(line,33,36,string,strsta,intval,intsta,fltval,fltsta)
string returns 'SNOW'
strsta returns success
intsta and fltsta return failure

ok = mcextrln(line,32,35,string,strsta,intval,intsta,fltval,fltsta)
fltval returns 45.3
string returns '45.3'
intsta returns failure
```

Below is an example of tabular text.

```
FOUS51 KWBC 181200                                                95230 1615
TRAJECTORY FCST
          181200Z   181800Z   190000Z   190600Z          191200Z
         LATLONPPP LATLONPPP LATLONPPP LATLONPPP      TEMP DEWPT   K
 BIL 700 446169672 445148642 446126631 449107659       2.7 -22.3  -9
     850 466129813 465108826 462093842 456086841      10.0  -4.0
     SFC 466128819 465107832 462092848 456085846 854   8.8  -3.0
 RAP 700 443148572 445115609 447083640 445054671       6.6 -19.7 -10
     850 476112810 473083834 467060846 455040855      11.2  -3.1
     SFC 478104877 475077895 468057905 456038910 902  12.3    .2
 BIS 700 483169547 480136575 480098605 477053650       2.3 -27.4 -14
     850 495141739 494109777 493076795 483039823      10.4  -5.1
     SFC 490122865 492090900 491065914 482034930 944  14.9   1.1
```

Formatted text

If the text is formatted, such as SYNOP or RAOB reports, the McIDAS-XCD function **m0parobs** will parse the observation into tokens that can then be broken down into meteorological data. The source line contains the following:

```
'72645 11/// AUTO 10244 20222'

parameter (maxgrp = 100)
character*8 creprt(maxgrp)
integer report(maxgrp)

ok = m0parobs(msg,28,maxgrp,report,creprt,ngroup)
```

ngroup returns 5

Group #	creprt	report
1	'72645'	72645
2	'11///'	11000
3	'AUTO'	-1
4	'10244'	10244
5	'20222'	20222

When **m0parobs** is finished, you can begin extracting parameters. For example, if you determine that the fourth group is the temperature field in Celsius, the extracting code will be:

```
if (creprt(4)(3:5) .ne. '///') temp = float(mod(report(4),1000)) / 10.0
```

Below is an example of formatted text.

```
SXUS23 KWBC 182200 RRA                                              95230 2244
CMAN 18224
BLIA2 46/// /2307 10114 40171 92230 333 91207 555 11006 22006=
POTA2 46/// /2308 10123 30150 40164 92230 333 91211 555 11008 22008=
```

Plain text

Plain text, such as SAO reports, is the most difficult data to decode. Plain text reports do not have a rigid format or data indication flags. These decoders are the most complicated due to the nature of the format. The McIDAS-XCD functions **m0dcsplt** and **m0grbval** can help. **m0dcsplt** tokenizes a string into its basic components and **m0grbval** gives you the values in the tokenized string **[A408-A447]**.

Example:

```
'MSN SA 1155 CLR 10 207/75/65'

include 'xcd.inc'
integer terms(2)
data terms/59,61/   ! termination characters '=' and ';'

numgrp = 0
ok = m0dcsplt(80,numgrp,2,terms)
```

numgrp returns 15

Group #	Value	Type	Numch	Point
1	MSN	ACHAR	3	1
2	blank	ABLANK	1	4
3	SA	ACHAR	2	5
4	blank	ABLANK	1	7
5	1155	ADIGIT	4	8
6	blank	ABLANK	1	12
7	CLR	ACHAR	3	13
8	blank	ABLANK	1	16
9	10	ADIGIT	2	17
10	blank	ABLANK	1	19
11	207	ADIGIT	3	20
12	/	ASLASH	1	23
13	75	ADIGIT	2	24
14	/	ASLASH	1	26
15	65	ADIGIT	2	27

Other Type values include the following.

Type	Description
-1	terminator character
APUNC	, . : ;
PLSMNS	+ -
ARECSP	record separator (0x1e)
AXCLAM	!
AAMPER	@
ALOGNT	^
AOTHER	any other character

Below is an example of plain text.

```
SAUS80 KWBC 182200                                                    95230 2151
ILM SA 2150 50 SCT 250 -SCT 7 112/89/74/1507/986/ MDT CU W
 LGA SA 2150 250 SCT 13 132/89/60/0512/992/ CU N
 LND SA 2150 300 -BKN 70 147/78/18/3212/015/ CU N-NE
 MFE SA 2150 40 SCT E250 BKN 10 85/75/0911/986/TCU N-E
 MIA SA 2150 30 SCT M50 BKN 250 OVC 7 120/90/75/2210/988/ CB N MOVG S
 MSN SA 2150 28 SCT M36 BKN 7 144/85/73/1208/997
 OFK SA 2150 CLR 15 073/96/72/1712/980/ CB DSNT W-NW MOVG NE
```

Retrieving Text Data

Data Block Index

1 - circuit source
2 - number of bytes in data block
3 - starting location in *.XCD file
4 - time stamp (hhmmss)
5 - WMO header ex. 'FOUS'
6 - WMO product number
7 - WMO station origin ex. 'KWBC'
8 - AFOS/AWIPS product code
9 - AFOS/AWIPS station origin
10 - AFOS/PIL station origin
16 - FAA catalog number

FO95300.IDX

224

	1	2	3	4	5	6	7	8	9	10	16
160	DDS	240	10240	062104	FOUS	14	KWBC				
176	DDS	320	11280	062109	FOUS	78	KWBC				
192	DDS	320	13260	062109	FOUS	78	KWBC				
208	DDS	880	16000	062145	FOAK	78	KWBC				
224	DDS	540	17020	072356	FOUS	30	KWBC	QPF	ERD		

mcgetidx

mOnxtidx

Interface for accessing sporadic data

Does the information come in continually or sporadically?

If data is continuous, such as surface hourlies, RAOBs, and MDRs, you should consider processing it in an automatic mode. This method requires no user intervention to ensure data is decoded.

To process data automatically, you must create data monitors similar those distributed in McIDAS-XCD. Using **m0nxtidx [C344-C347]**, these data monitors scan a predefined set of index files, checking to see if they have been updated by the ingestors. If they have, the data monitor wakes up, loads the new index block, and, if it is one the data monitor is interested in, calls an appropriate decoder**[C433-C439]**. One data monitor can run many decoders. If you plan to have numerous local decoders, SSEC recommends imbedding them in your own data monitors and activating them with the DECINFO command.

If your data is sporadic, it may be enough to run the decoder on an as-needed basis. For example, to plot projected hurricane storm tracks, which are not high volume reports, you can access the raw text directly each time a plot is requested instead of creating a special filing format just for hurricane tracks.

The interface for accessing sporadic data is very different from automatic processing. In the automatic mode, a data monitor, using **m0nxtidx**, scans to see if new data has arrived. In the as-needed mode, the application, using **mcgetidx [B39]**, starts with the most recent data and works backward in time. Each call to this function grabs the next oldest index block for the data you request. See the diagram on the adjacent page. Once you retrieve the new index block, you can load the text and call your decoder.

The structure of the index block returned by **mcgetidx** and **m0nxtidx** is as follows:

Position	Description
1	circuit source
2	number of bytes in the data block
3	starting location in the .XCD file
4	time stamp, hhmmss
5	WMO header, for example FOUS
6	WMO product number
7	WMO station origin, for example KWBC
8	AFOS/AWIPS product code, when available
9	AFOS/AWIPS station origin, when available
10	AFOS/PIL station origin, when available
16	FAA catalog number, when available

Once you have the appropriate index block, you load the data into a character array buffer with **mclddatb** and continue decoding.

Note: Binary data should be processed in the automatic mode.

Will the decoded information be used by one or many users?

If only one user will access the data, you may not need to create a data monitor for specific requests. Processing the data in an as-needed mode should be sufficient. If multiple users will access the data, consider the automatic mode to prevent redundant processing and wasting of resources.

What will the user do with the data?

If users want to plot or contour data, the data should be decoded and placed in McIDAS-supported data formats such as MD **[A783]** or grid files. Then users can display or list it with many of the core McIDAS commands.

When you build a decoder, do not tie that decoder to a specific filing format. The decoder should extract the information from the data stream into a known memory structure. Then you can write an ancillary routine that converts the memory structure to the format for the file system. Thus, if you decide to file a certain type of data in a different file structure, the only new software you have to write is a routine converting the memory structure to the new file format.

If timeliness of text retrieval is important for observational reports, you may want to decode the observation and file it in the McIDAS-XCD Rapid Access (RA) text format. This file structure allows the user to formulate the request for individual stations or groups of stations and retrieve the data quickly. Putting raw text into this format is a two-step process.

1. Register the filing format with the BILDTEXT command, which sets up the pointer files for observational data. Enter the type of data you will store, the length of time to store it, and the format of the station ID. BILDTEXT initializes the file based on this input.

2. In the decoder, open the RA file using **mctxtopn [A367]**. Fill in two data blocks with pertinent information about the observation, and call **mctxtwrt [A740-A773]** to write the data to an RA file. An example is provided in the program at the end of this section.

Once the data is in an RA file, you can use the command OBLIST to access the data quickly. Typically, you will build a macro on top of OBLIST to access the data type. The McIDAS-XCD commands SAO, RAOB, FT and TAF are examples of macros that call OBLIST.

Sometimes, the only thing a user needs to know from a data type is if a particular event took place. In this case, nothing needs to be filed, although an alert should be sent to the user. As an example, operations should be notified when an administrative message is received.

Do you have enough resources?

A tremendous amount of data passes through a McIDAS-XCD server workstation. In a typical ingest set up with DDS, PPS and IDS, you can expect about 100 Mb per day before the data is put in a special format such as MD or grid files. When you add McIDAS data files to this list, you quickly expand the need for disk storage for these machines.

To calculate the amount of disk space (in bytes) needed for an MD file, use this formula:

$$(NR * NC * NDKEYS + NC * NCKEYS + NR * NRKEYS) * 4$$

where:	*NR*	number of rows in the MD file
	NC	number of columns in the MD file
	NDKEYS	number of keys in the MD file's data section
	NCKEYS	number of keys in the MD file's column header
	NRKEYS	number of keys in the MD file's row header

To calculate the amount of disk space needed for grids, use this formula:

$$((NR * NC * 4) + 256) * NG$$

where:	*NR*	number of rows in the grid
	NC	number of columns in the grid
	NG	number of grids in the file

The HRS circuit transmits another 70 to 90 Mb per day. Once you decode and file the HRS signal, the amount of needed disk space grows rapidly, depending on which grid formats are decoded. Some grids sent by NCEP, for example, may be up to 160 Kb per grid, and approximately 500 grids are sent in this format.

SSEC recommends that your McIDAS-XCD workstation has a minimum of 2 Gb of hard disk for the typical ingest/decode configuration.

Other McIDAS-XCD subsystems

Many of the processes done in decoders are the same, regardless of the data type. This section describes some of the common subsystems in McIDAS-XCD that you can integrate into your site's local decoder system.

Gross error checking system

McIDAS-XCD performs gross error checks, discarding values that are not meteorologically possible. The McIDAS-XCD functions **mcgrsscl** and **m0grserr** perform this task **[A708-A709]**. Below is a table of the current parameters and thresholds used for gross error checking.

Parameter	Name	Units	Level	Min. value	Max. value
Pressure	P	mb	All	0	1100
Sea level pressure	SLP	mb	All	800	1100
Precipitation	PCP	M	All	0	2
Snow depth	SNOW	M	All	0	25
Sea surface temp	SST	K	All	265	315
Temperature	T	K	All	188	340
Dew point	TD	K	All	188	310
Wind direction	DIR	Deg	All	0	360
Wind speed	SPD	MPS	All	0	200
Zonal wind	U	MPS	All	0	200
Meridional wind	V	MPS	All	0	200
Visibility	VIS	M	All	0	300000
Cloud height	CLD	M	All	0	27,000
Height	Z	M	MSL	-500	500
Height	Z	M	SFC	-500	no maximum
Height	Z	M	1000	-500	500
Height	Z	M	925	0	1000
Height	Z	M	850	1000	2000
Height	Z	M	700	2500	4000
Height	Z	M	500	4000	7000
Height	Z	M	400	6000	9000
Height	Z	M	300	8000	11000
Height	Z	M	250	9000	13000
Height	Z	M	200	10000	14000
Height	Z	M	150	11000	15000
Height	Z	M	100	12000	18000
Height	Z	M	TRO	0	no maximum

Some McIDAS-XCD subsystems require numerous settings for proper configuration. Because it is impractical to enter these fields from a McIDAS command line, a configuration script language and interface was developed. Users can write scripts with a text editor and the subsystem will get the information it needs from the text file instead of the command line. This mechanism is currently used for configuring communications ports and defining settings for decoders.

The syntax of the configuration script language is simple. It is designed to get specific values based on keywords and positional parameters. The McIDAS-XCD routines used to interface with this subsystem are:

Function	Description	Language
Mcgtcfgstr	retrieves a value as a string from the file	C
mcgtcstr	retrieves a value as a string from the file	FORTRAN
Mcgtcfgint	retrieves a value as an integer from the file	C
mcgtcint	retrieves a value as an integer from a file	FORTRAN
Mcgtcdbl	retrieves a value as a double from a file	C
mcgtcdbl	retrieves a value as a double from a file	FORTRAN

The contents of a sample file are shown below.

```
#       Cross reference list
#       The cross reference list allows values to be accessed with
#       multiple keyword names

:FLAGS[01]MD
:FLAGS[02]NR
:FLAGS[03]NC

MD=101              # first MD file in real-time range
NR=24               # number of rows to make for MD file
NC=500              # number of columns to make for MD file
WMO=FOUS FOUE       # list of WMO headers to decode
MINPRD=51           # minimum WMO product number to decode
MAXPRD=57           # maximum WMO product number to decode
DTIME=17.0          # number of hours to scan back in time to locate data
```

Below is sample code used to extract information.

```
integer intval
double precision fltval
character*80 error , string
character*12 file

file   = 'example.cfg'
ok     = mcgtcstr (file , 'WMO' , 1 , string , error)
           string will contain 'FOUS'
ok     = mcgtcdbl (file , 'DTIME' , 1 , fltval , error)
           fltval will contain 17.0
ok     = mcgtcint(file , 'FLAGS[01]' , 1 , intval , error)
           intval will contain 101
```

The McIDAS-XCD decoders have a useful interface into the configuration scripts through the function **m0dcinfo [C187-C195]**.

The McIDAS-XCD status display monitors the status of the ingestors and decoders. Operators can check the status display by running the STAT command in McIDAS, or *statdisp* from the Unix prompt. The functions for reading from and writing to the status display are **m0rsdcd [A375]** and **m0wsdcd [A800]**, respectively. The positions in the memory structure written to the status display are found in xcd.inc, and are shown below.

```
                                        Gridf  Grid
## Decoder Time    Begptr  Lasptr  MD     Row    Col    Text          Index
   (1)     (2)      (3)     (4)    (5)    (6)    (7)    (8)           (9)
```

##	Flag name	Description
1	BBTASK	Name of the decoder, 8 characters maximum
2	BBTIME	Time stamp, hhmmss
3	BBBPTR	Beginning index location being processed, usually set by m0nxtidx
4	BBLPTR	Last index location to process, usually set by m0nxtidx
5	BBMD	MD file number being processed
5	BBGRDF	Grid file number being processed
6	BBROW	MD file row number being written to
6	BBGRID	Grid number being written to
7	BBCOL	MD file column number being written to
8	BBTEXT	General text, 12 characters maximum
9	BBDXNM	Index file being processed, usually set by m0nxtidx
	BBON	Status flag indicating the decoder is on
	BBDAY	Julian day stamp

The information in the status display is stored on disk. McIDAS-XCD decoders read from/write to the file ~oper/mcidas/data/DECOSTAT.DAT. If you write local decoders that use the status display, SSEC recommends writing your status to a different file so you won't have to modify the file with each upgrade. To display local decoder status, assign a file name to the environment variable **XCD_disp_file** and start a second version of *statdisp*.

The structure for writing data to the status display is provided in the include file xcd.inc. Portions of this information may be written to the status display by the index block access routine **m0nxtidx,** but most information is provided by the decoder.

The typical way to write data to the status display is to fill the array *bullbd* with the appropriate values and to call the writing routine **m0wsdcd [A792-A800]**, as shown in the example below.

```
INCLUDE 'xcd.inc'
IMPLICIT INTEGER (A-Z)

md=102
row=3
col=21
decnum=4
.
.
.
bullbd(BBON)= 1
bullbd(BBDAY)= yyddd
bullbd(BBTIME)= now
bullbd(BBMD)= md
bullbd(BBROW)= row
bullbd(BBCOL)= col

call movcw('TRJDEC' , bullbd(BBTASK))
call movcw('TRAJECTORY',bullbd(BBTEXT))
ok = m0wsdcd ('LOCALSTA.DAT','DEC',decnum,bullbd,BBSIZE,jstat)
```

Once the call to m0wsdcd is complete, the status display looks like this:

```
## Decoder   Time  Begptr  Endptr  MD   Row  Col  Text
 4 TRJDEC  063423  160000  160160  102    3   21  TRAJECTORY
```

Station monitoring

The station tables delivered with McIDAS-XCD don't always contain all stations available for a data type. They may also contain obsolete stations no longer reporting a certain data type. McIDAS-XCD lets you monitor incoming stations in a decoder with the **m0idnew** function **[A565]**. This function keeps a running count of the number of times a station reports, and files the last day and time a station reported.

Use the command IDMON to display the station status. To add a station to a decoder type, enter: **IDU EDIT** *stdid* **SWITCH=YES DEC=***decname* replacing *stdid* with the station to modify and *decname* with the name of the decoder to add to the station.

If you add a station to a decoder that is currently filing in an RA file (*rafile*), you must add that station separately with the BILDTEXT ADD command. For example: **BILDTEXT ADD** *stdid rafile*.

DECTEST

The command DECTEST is included in the McIDAS-XCD package. As a developer, you can link your text decoder into DECTEST during development to simulate how the decoder will work in a real-time data monitor. It is a good way to expose problems while they are easy to locate.

Compiling and linking

To build decoders in the McIDAS-XCD environment, you typically write decoders as functions placed in a library and call these decoders from the McIDAS command line or scheduler. Decoder software is usually built using the McIDAS library archive script *mcar* and the McIDAS compilation script *mccomp*.

To compile the decoder function, **m0trjdec.for**, and place the object in the library libmylib.a, perform these two steps.

1. Compile the function **m0trjdec.for**.

```
mccomp -I. -I/~mcidas/inc -c m0trjdec.for
```

2. Put the object code created in step 1 in the library libmylib.a.

```
mcar libmylib.a m0trjdec.o
```

Now, compile the data monitor, dmlocal.pgm; link it with the appropriate libraries; and create the McIDAS executable, dmlocal.mx.

3. Compile the data monitor, dmlocal.pgm.

```
mccomp -I. -I~mcidas/inc -c dmlocal.pgm
```

4. Link the data monitor to the appropriate libraries and generate the McIDAS executable dmlocal.mx.

```
mccomp ~mcidas/lib/main.o dmlocal.o -L. -L/~mcidas/lib
  -L/~oper/mcidas/lib -lmylib -lxcdcli -lxcd -lmcidas -lX11
  -o dmlocal.mx
```

Integrating a local decoder in McIDAS-XCD

This section provides an example of how to decode and file NGM Trajectory Forecasts. These forecasts are sent twice daily in conjunction with the NGM model runs. This is not a high volume dataset, as the NCEP supplies trajectory forecasts for only about 40 stations. The reports are sent in WMO headers FOUS50-57. The decoder provided at the end of this section can be implemented to run either as-needed or automatically. The steps for making the decoder run automatically are in the last part of the exercise.

The raw data format is as follows:

```
FOUS51 KWBC 181200
TRAJECTORY FCST
          181200Z   181800Z   190000Z   190600Z          191200Z
         LATLONPPP LATLONPPP LATLONPPP LATLONPPP      TEMP DEWPT   K
 BIL 700 446169672 445148642 446126631 449107659       2.7 -22.3  -9
     850 466129813 465108826 462093842 456086841      10.0  -4.0
     SFC 466128819 465107832 462092848 456085846 854   8.8  -3.0
 LND 700 424119703 423100681 421087693 420087694       7.1  -8.1   8
     SFC 441081807 434071814 428067813 423077781 767   9.7    .8
```

Part 1: Building the decoder and putting the data into MD files

Perform the six steps below to build the pieces of the decoder and put the data into MD files.

1. Build the schema template and register the schema. Since this is point source data at fixed locations, place the location-dependent information in the column header and the time-dependent information in the row header.

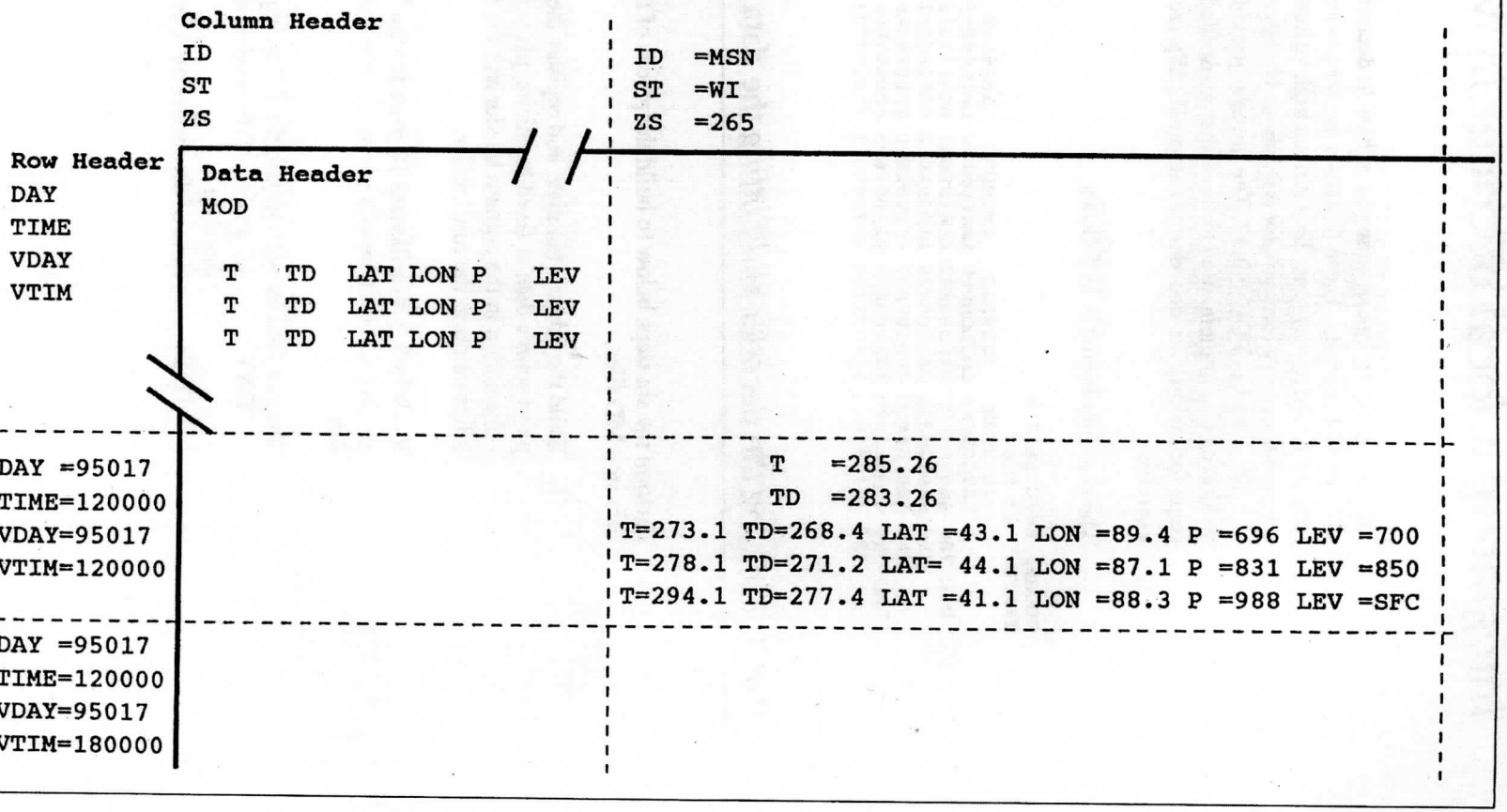
   By default, you will want 10 rows for the MD file because there are five forecast periods per model run and two model runs per day.

   The *row header* will contain the following:

DAY	Julian day of the model run that generated this data
TIME	time of the model run that generated this

```
                                  171200Z    171800Z    180000Z    180600Z      181200Z
                                 LATLONPPP  LATLONPPP  LATLONPPP  LATLONPPP    TEMP DEWPT    K
              MSN   700   432894696
                    850   441871831
                    SFC   411883988
```

Trajectory MD file structure



Sample MD file structure

You will need one column for each station reporting trajectories. The *column header* will contain the following:

ID	station ID
ST	state
ZS	station elevation

The *data header* will contain the following:

MOD	modification flag
3 REPEAT groups (SFC, 850 and 700)	
T	temperature at the station
TD	dew point at the station
LAT	latitude of the parcel
LON	longitude of the parcel
P	pressure of the parcel
LEV	pressure level of the parcel (SFC, 850 or 700)

Once the schema is designed appropriately it is registered with the LSCHE command.

See the sample trajectory MD file structure on the adjacent page.

2. Write a decoder that parses out the information into a known structure. Remember to adhere to the standard calling sequence **[A0-A47]** used by the core decoders if you want to test the decoder using DECTEST. If the decoder will run in automatic mode, use the configuration script interface of the function **m0dcinfo**. The standard calling sequence looks like this:

   integer function m0trjdec(wmohdr, block, nlines, circit, julday, timdec, flags, cflags)

   The input values are defined below.

wmohdr	c*(*)	line containing the WMO header for example: FOUS51 KWBC 231200
block	c*(*)	array containing the data portion
nlines	integer	number of lines in *block* that this data section uses
circit	c*(*)	data source; seldom used
julday	integer	Julian day for which the data is valid, yyddd
timdec	integer	time of the data filing, hhmmss
flags	integer	array containing the integer values for this decoder; typically they will be: 1 - error output flag (1 = active) 2 - ID monitoring key 3 - decoder number for the status display 4 - base MD or grid file range 5 - number of rows in the MD file 6 - number of columns in the MD file 7 - rapid text filing flag

cflags	c*(*)	array containing the characters strings for this decoder; typically they will be: 1 - error file name 2 - old station ID file name 3 - new station ID file name 4 - ID table name to use 5 - master station list to use 6 - pointer file for the RA format

3. Create the MD file, if it does not exist, and initialize its row and column headers **[A809-1014]**. Building the row headers is easy **[A979-A1000]** because all the information is in the decoding process. Building the column headers is more difficult because the only information you have is the station ID, and you also need the state and station elevation.

   The McIDAS-XCD function **m0bildid [A959]** builds a station ID table from parameters that you suggest. Since you need the station ID, state and station elevation for the column header, send your request to **m0bildid** to build a file with these parameters **[A1002-A1009]**. Once the ID table is built, you can build the column headers for the MD file.

4. When the data is decoded, find the column number in the MD file where the station will be filed. Use the function **m0loccol** to find the appropriate column number; you must provide the station ID and MD file number **[A637-A640]**. This function may appear cumbersome, but it was designed to work with an unlimited number of MD files simultaneously. This is useful when you have multiple decoders activated by the same data monitor.

5. Convert the known decoded trajectory structure to the format that MDO expects and file the data in the MD file. The conversion process may include unit conversions (Celsius to Kelvin), type conversion (floating point to scaled integer), or some type of symbol conversion.

6. Write the output to the decoder status display using **m0wsdcd [A792-A800]**, and process the next observation to decode.

Part 2: Automating the decoder process

Use the three steps below to make the decoder/filer an automatic process and connect it to McIDAS-XCD.

1. Create a data monitor. The easiest way to do this is to start with an existing one. If you use **dmmisc.pgm** as the template for **dmlocal.pgm [C0-C540]** and you use the standard decoder calling sequence enumerated in step 2 below, you will only have to change about 10 lines of code to install your decoder.

   For these trajectory products you will change:
   numdec to 1 **[C47]**
   task to DMLOCAL **[C53]**
   decnam to TRJDEC **[C157]**

   The actual call to the decoder will look like this **[C433-C439]**:

```
  if (decnam(dec) .eq. 'TRJDEC')then
        decok = m0trjdec(cblk(1),cblk(2),
&               nlines-1,circit,yyddd,time,flags(1,dec),cflags(1,dec))
  endif
```

2. Set up a configuration file to tell the system where to look for data, the MD file range in which to put the data, the number of rows and columns in the MD file, and the name of the station ID table.

   Assume that you want the data filed in MD files 101 to 110 and the MD files are 10 rows by 600 columns. The data comes in as WMO headers FOUS50-57. Your configuration file, TRJDEC.CFG, might look like the one below.

```
# TRJDEC.CFG            - Configuration file for the Trajectory forecasts
# ------------------ Cross reference List -----------------
: FLAGS[01]   ERRORFLG
: FLAGS[02]   IDMONFLG
: FLAGS[03]   DISPLAYNUM
: FLAGS[04]   MDF
: FLAGS[05]   NROWS
: FLAGS[06]   NCOLS

: CFLAGS[01] ERRORFILE
: CFLAGS[02] OLDIDFILE
: CFLAGS[03] NEWIDFILE
: CFLAGS[04] IDTABLE
: CFLAGS[05] MASTERFILE
#   -------------------- End Of Cross Reference List -------------------
# -              You can modify any of the fields below                -
# decoder description
DESCRIPTION="FOUS50-57 Decoder"

# which indices to search for this decoder
INDEX=FO

# which specific WMO headers to activate the decoder for
WMO=FOUS
MINPRD=50
MAXPRD=57

# which specific station origins to activate the decoder for
ORIGIN=KWBC

ERRORFLG=0                  # error output flag set to 1 to activate
ERRORFILE=FO50DEC.ERR       # error file name
IDMONFLG=0                  # station id monitoring activation flag
                            #    set to 1 or 3 to monitor new stations
                            #    set to 2 or 3 to monitor old stations
OLDIDFILE=OLDFO50.IDM       # old station id file used for monitoring
NEWIDFILE=NEWFO50.IDM       # new station id file used for monitoring
DISPLAYNUM=1                # decoder number on status display
MDF=101                     # first real-time MD file number to use for decoder
NROWS=10                    # number of rows to make for MD file
NCOLS=600                   # number of columns to make for MD file
IDTABLE=FO50DEC.IDT         # ID file to build when creating MD file
MASTERFILE=MASTERID.DAT     # master ID table file to use to build IDTABLE
```

3. Configure the system so STARTXCD activates the data monitor DMLOCAL and starts the decoder TRJDEC. Enter the three commands below from the McIDAS command line, logged on as oper.

   DECINFO ADD DMLOCAL DEC=TRJDEC

   DECINFO EDIT DMLOCAL TRJDEC ACTIVE CONFIG=TRJDEC.CFG

   DECINFO SET DMLOCAL ACTIVE

   The next time STARTXCD samples the system configuration, it will see that DMLOCAL is active and will try to start it.

Developing local decoders for your site

Below is a list of local decoders that you may want to develop at your site.

Binary

BUFR format
Lightning

Text

Description	WMO Header	Text Format
Hurricane forecast positions	WT	Plain
Model output text	FO	Tabular
BATHY (bathythermal obs)	SO	Formatted
WAVEOB (spectral wave obs)	SX	Formatted
C-MAN (automated pier obs)	SX	Formatted
WINTEM (aviation wind/temp forecast)	FB	Tabular
ARFOR(area forecast for aviation)	FA	Tabular
SARAD(satellite clear radiance obs)	TR	Formatted
SATEM (satellite remote upper air soundings)	TH	Formatted
Frontal analysis	AS	Formatted
Forecasted frontal analysis	FS	Formatted
Winds aloft forecast	FD	Tabular
Satellite derived cloud information	TB	Tabular
Daily climate summaries	CS	Plain
Dropsonde	UZ	Formatted
SFLOC (Report of geographic location of atmospherics)	SF	Formatted
River and rainfall observations	SR	Plain
Aircraft recon data	UR	Formatted
Coded city forecasts	FP	Formatted
Convective outlook	AC	Plain
Coded upper air forecasts	FU	Formatted

Useful functions in McIDAS-XCD

The library functions described below are provided in the McIDAS-XCD package in two separate libraries. Functions called by the McIDAS-XCD user commands reside in the library ~mcidas/lib/libxcdcli.a. The source code associated with libxcdcli.a is located in ~mcidas/xcd1.X/src. Functions specific to the server portion of the package are in ~oper/mcidas/lib/libxcd.a. The source code associated with libxcd.a is located in ~oper/mcidas/xcd1.*X*/src.

Ingestors

Source file	Library	Description
m0filblk.for	libxcd.a	files a text data block in the native McIDAS-XCD format
M0pt_utils.c	libxcd.a	group of functions that configures and reads from communications ports

Data retrieval

Source file	Library	Description
mcfndgrd.for	libxcdcli.a	returns a list of grids in a grid file given search conditions
mcgetidx.for	libxcdcli.a	retrieves the index block directory given search conditions
mclddatb.for	libxcdcli.a	loads the raw text of a data block
mclodbin.for	libxcdcli.a	loads the list of text index files
mctxtopn.for	libxcdcli.a	opens an RA text file
mctxtred.for	libxcdcli.a	reads an observation from an RA file given search conditions
Mcrtgdfile.c	libxcdcli.a	retrieves the list of real-time grid files given search conditions
Mcrtmodels.c	libxcdcli.a	retrieves the list of real-time grid files
M0getsplbyt.c	libxcd.a	retrieves bytes from a spool file
m0splnam.for	libxcdcli.a	creates the *.XCD file name for a given day and source
m0txtget.for	libxcdcli.a	retrieves an observation from an RA file
m0txtput.for	libxcd.a	writes an observation to an RA file

Station IDs

Source file	Library	Description
mcclsest.for	libxcdcli.a	returns a list of stations interactively from the cursor
mcid2idn.for	libxcdcli.a	converts station ID to station block number
mclodids.for	libxcdcli.a	loads a list of station IDs from a group list
m0bildid.for	libxcdcli.a	builds a station ID table based on selection criteria
m0idnew.for	libxcd.a	monitors new and old station acquisition tables

String parsing

Source file	Library	Description
mcextrln.for	libxcd.a	retrieves parameters from a fixed format string
Mcgtpstrg.c	libxcd.a	parses a formatted string that contains field separators
mcscnblk.for	libxcdcli.a	scans an entire data block looking for string matches
m0dcsplt.for	libxcdcli.a	tokenizes a string into its basic components
m0grbval.for	libxcdcli.a	retrieves a component of a m0dcspltED tokenized string
m0parobs.for	libxcd.a	parses a SYNOPTIC style text string

Source file	Library	Description
mccidflt.for	libxcdcli.a	returns a reasonable default contour interval for a given parameter/level
mcfiltim.for	libxcd.a	returns the nominal time of an observation
Mcibmfloat.c	libxcd.a	converts IBM floating point representation to machine native format
Mcisbitset.c	libxcd.a	tells whether a bit in a buffer is set
Mcpackbit.c	libxcd.a	packs bits into a buffer
Mcupackbit.c	libxcd.a	unpacks bits from a buffer
mcydd2ch.for	libxcd.a	converts Julian day to a variety of formats
m0af2wmo.for	libxcd.a	converts AFOS PILs to WMO headers
m0grserr.forl	ibxcd.a	performs gross error checks on known meteorological parameters
m0isrung_.c	libxcd.a	tells whether a Unix PID is active
m0rsdcd.for	libxcd.a	reads from a status display
m0wsdcd.for	libxcd.a	writes to a status display
M0unixuser.c	libxcd.a	returns the user ID of the session

Sample decoder

```
A    0: c $ Name:
A    1: c $ m0trjdec - decode trajectory forecasts from FOUS51-57 into
A    2: c $                 TRAJ schema MD file
A    3: c $
A    4: c $ Interface:
A    5: c $ integer function
A    6: c $     m0trjdec(character*(*) wmo, character*(*) cblk(*),
A    7: c $              integer nlines,integer julday,integer timdec,
A    8: c $              character*(*) circit, integer flags(*),
A    9: c $              character*(*) cflags(*))
A   10: c $
A   11: c $ Input:
A   12: c $  wmo      wmo header of data block
A   13: c $  cblk     array containing raw text data
A   14: c $  nlines      number of lines in cblk used
A   15: c $  julday      julian day of the data
A   16: c $  timdec      time stamp from the data (hhmmss)
A   17: c $  circit      source of the data
A   18: c $  flags       array of integer flags
A   19: c $                        3  - decoder display number
A   20: c $                        4  - base md file in range
A   21: c $                        5  - number of rows for the file
A   22: c $                        6  - number of columns
A   23: c $   cflags  array of character strings
A   24: c $              3  - file name to use to contain
A   25: c $                      the list of new stations
A   26: c $              5  - station id file to use
A   27: c $
A   28: c $ Input and Output:
A   29: c $      none
A   30: c $
A   31: c $ Output:
A   32: c $      none
A   33: c $
A   34: c $ Return values:
A   35: c $   0 - success
A   36: c $
A   37: c $ Remarks:
A   38: c $ The RA text file for this file format is initialized by
A   39: c $ the keyin.
A   40: c $
A   41: c $ BILDTEXT INIT FO50.RAP FO50.RAT 600 3 C4 5 12 80 FOUS14 X 1
A   42: c $
A   43: c $ The resultant MD file schema used is the TRAJ schema.
A   44: c $
A   45: c $ This decoder is intended as an example only.  It was built
A   46: c $ to demonstrate many of the features at the disposal of the
A   47: c $ developer in writing local decoders.
A   48:
A   49:      integer function m0trjdec(wmo,cblk,nlines,julday,timdec,
A   50:       &                              circit,flags,cflags)
A   51:      implicit none
A   52:      include 'xcd.inc'
A   53:
A   54: c---      maxlev   - maximum number of trajectory levels
A   55: c---      maxper   - maximum number of forecast periods
A   56: c---      maxsta   - maximum number of stations
A   57: c---      maxids   - maximum number of stations in this data block
A   58: c---      ndkeys   - number of data keys in md file
A   59:
A   60:      integer maxlev
A   61:      integer maxper
A   62:      integer maxsta
```

```
A   63:      integer maxids
A   64:      integer ndkeys
A   65:      parameter (maxlev = 3 , maxper = 5 , maxsta = 600)
A   66:      parameter (maxids = 30 , ndkeys = 19)
A   67:
A   68:      character*(*) cblk(*) , cflags(*) , circit , wmo
A   69:      integer flags(*)
A   70:
A   71: c---      daylst  - array containing recent days of the month
A   72: c---      yyddd   - julian days associated with recent days of
A   73: c---                the month
A   74:      integer daylst(4)
A   75:      integer yyddd(4)
A   76:
A   77: c---      idinfo  - return array from m0loccol containing station
A   78: c---                information
A   79:      integer idinfo(6)
A   80:
A   81: c---      vday    - list of forecast valid julian days
A   82:      integer vday(maxper)
A   83:
A   84: c---      vtime   - list of forecast valid times (hhmmss)
A   85:      integer vtime(maxper)
A   86:
A   87: c---      pre     - list of pressure levels of parcels
A   88:      integer pre(maxper , maxlev , maxids)
A   89:
A   90: c---      lat     - list of latitudes of parcels
A   91: c---      lon     - list of longitudes of parcels
A   92:      integer lat(maxper , maxlev , maxids)
A   93:      integer lon(maxper , maxlev , maxids)
A   94:
A   95: c---      linloc  - location in data block where each ob is found
A   96: c---                for level and station
A   97:      integer linloc(maxlev , maxids)
A   98:
A   99: c---      bcol    - beginning column for a forecast
A  100: c---      ecol    - ending column for a forecast
A  101:      integer bcol(maxper)
A  102:      integer ecol(maxper)
A  103:
A  104: c---      tlat    - temporary latitude array for storing
A  105: c---                maxper latitudes for one level
A  106: c---      tlon    - temporary longitude array for storing
A  107: c---                maxper longitude for one level
A  108: c---      tpre    - temporary pressure array for storing
A  109: c---                maxper pressure for one level
A  110:      integer tlat(maxper)
A  111:      integer tlon(maxper)
A  112:      integer tpre(maxper)
A  113:
A  114: c---      staton  - list of all possible stations that could
A  115: c---                file this type of report
A  116: c---      slat    - list of latitudes of possible stations that
A  117: c---                could file this type of report
A  118: c---      slon    - list of longitudes of possible stations that
A  119: c---                could file this type of report
A  120:      integer slat(maxsta)
A  121:      integer slon(maxsta)
A  122:      integer staton(maxsta)
A  123:
A  124: c---      line    - temporary array used for cracking raw text
A  125:      integer line(20)
A  126:
A  127: c---      mdbase  - base md file number
A  128: c---      mdrow   - md row number to write to
A  129: c---      mdcol   - md column number to write to
A  130: c---      nr      - number of rows to make for md file
A  131: c---      nc      - number of columns to make for md file
A  132: c---      record  - array containing decoded data section for
A  133: c---                writing to md file
```

```
A  134:      integer mdbase
A  135:      integer mdrow
A  136:      integer mdcol
A  137:      integer nr
A  138:      integer nc
A  139:      integer record(ndkeys)
A  140:
A  141: c---      mdlocs  - array containing md file positions for all
A  142: c---                of the parameters in the data section
A  143:      integer mdlocs(ndkeys)
A  144:
A  145: c---      mdidtb  - array containing list of stations in column
A  146: c---                headers for m0loccol
A  147:      integer mdidtb(maxsta)
A  148:
A  149: c---      blkdom  - block domain used for RA
A  150: c---      timdom  - time domain used for RA
A  151: c---      ptrhed  - pointer header for RA
A  152: c---      idtab   - station id table for RA
A  153:      integer blkdom(4)
A  154:      integer timdom(8)
A  155:      integer ptrhed(THSIZE)
A  156:      integer idtab(maxsta)
A  157:
A  158: c---      t       - array containing all temperature forecasts for
A  159: c---                the final period
A  160: c---      td      - array containing all dew point forecasts for
A  161: c---                the final period
A  162:      double precision t(maxlev,maxids)
A  163:      double precision td(maxlev,maxids)
A  164:
A  165: c---      tt      - temporary temperature storage
A  166: c---      ttd     - temporary dew point storage
A  167:      double precision tt
A  168:      double precision ttd
A  169:
A  170: c---      cvalue  - temporary string value used in mcgrbval
A  171:      character*12 cvalue
A  172:
A  173: c---      stafil  - file containing list of all possible stations
A  174:      character*12 stafil
A  175:
A  176: c---      cid     - array of decoded station ids
A  177:      character*12 cid(maxids)
A  178:
A  179: c---      ctemp   - temporary string
A  180:      character*12 ctemp
A  181:
A  182: c---      ctemp4  - temporary string
A  183:      character*4 ctemp4
A  184:
A  185: c---      cline   - temporary string used for debug messages
A  186:      character*80 cline
A  187:
A  188: c---      ptrfil  - pointer file for RA
A  189:      character*12 ptrfil
A  190:
A  191: c---      lvls    - array containing observation levels for each
A  192: c---                station
A  193:      character*4 lvls(maxlev,maxids)
A  194:
A  195: c---      tlvl    - temporary storage variable for observation level
A  196:      character*4 tlvl
A  197:
A  198: c---      clit    - function declaration
A  199: c---      lit     - function declaration
A  200: c---      incday  - function declaration
A  201: c---      dectrj  - function declaration
A  202: c---      makmdf  - function declaration
A  203: c---      mdo     - function declaration
A  204: c---      mctxtopn- function declaration
```

```
A  205: c---      mcgrsscl- function declaration
A  206: c---      mclodids- function declaration
A  207: c---      m0dcsplt- function declaration
A  208: c---      m0loccol- function declaration
A  209: c---      mctxtwrt- function declaration
A  210:
A  211:      character*4 clit
A  212:      integer lit
A  213:      integer incday
A  214:      integer mctxtopn
A  215:      integer mcgrsscl
A  216:      integer mclodids
A  217:      integer m0dcsplt
A  218:      integer dectrj
A  219:      integer makmdf
A  220:      integer m0loccol
A  221:      integer mctxtwrt
A  222:      integer mdo
A  223:
A  224: c---      nlines  - number of lines used in cblk
A  225:      integer nlines
A  226:
A  227: c---      julday  - julian day the data represents
A  228: c---      timdec  - current time
A  229:      integer julday
A  230:      integer timdec
A  231:
A  232: c---      frstcl  - flag indicating this is the first call to dectrj
A  233:      integer frstcl
A  234:
A  235: c---      opnrap  - return value from mctxtopn
A  236:      integer opnrap
A  237:
A  238: c---      i,j,k   - loop counters
A  239:      integer i
A  240:      integer j
A  241:      integer k
A  242:
A  243: c---      decnum  - decoder number to write to on status display
A  244:      integer decnum
A  245:
A  246: c---      month   - month number for julday
A  247: c---      year    - year for julday
A  248:
A  249:      integer month
A  250:      integer year
A  251:
A  252: c---      numrap  - number of stations stored in RA file
A  253:      integer numrap
A  254:
A  255: c---      stat    - status returned from m0wsdcd and m0rsdcd
A  256:
A  257:      integer stat
A  258:
A  259: c    ok    - function return value
A  260:      integer ok
A  261:
A  262: c---      numlod  - number of stations loaded from m0loccol
A  263:      integer numlod
A  264:
A  265: c---      linnum  - internal line number counter
A  266:      integer linnum
A  267:
A  268: c---      numgrp  - number of groups tokanized by m0dcsplt
A  269:      integer numgrp
A  270:
A  271: c---      dum     - dummy variable
A  272: c---      dum1    - dummy variable
A  273: c---      dum2    - dummy variable
A  274: c---      dum3    - dummy variable
A  275: c---      dum4    - dummy variable
```

```
A  276: c---      dum5     - dummy variable
A  277:      integer dum
A  278:      integer dum1
A  279:      integer dum2
A  280:      integer dum3
A  281:      integer dum4
A  282:      integer dum5
A  283:
A  284: c---      value    - value returned from m0grbval
A  285:      integer value
A  286:
A  287: c---      group    - group counter from m0dcsplt
A  288:      integer group
A  289:
A  290: c---      numper   - number of forecast periods
A  291:      integer numper
A  292:
A  293: c--       tday     - temporary day storage
A  294:      integer tday
A  295:
A  296: c---      numid    - counter for number of stations decoded
A  297:      integer numid
A  298:
A  299: c---      nostat   - flag indicating if a station id had been found
A  300:      integer nostat
A  301:
A  302: c---      clat     - station latitude
A  303: c---      clon     - station longitude
A  304:      integer clat
A  305:      integer clon
A  306:
A  307: c---      lev      - pointer to the level 1-SFC 2-850 3-700
A  308:      integer lev
A  309:
A  310: c---      id       - station id
A  311:      integer id
A  312:
A  313: c---      mdfile   - mdfile number to make
A  314:      integer mdfile
A  315:
A  316: c---      mdlist   - list of md files to use for m0loccol
A  317:      integer mdlist
A  318:
A  319: c---      pt       - pointer describing where in 'record' array to
A  320: c---                 store decoded data
A  321:      integer pt
A  322:
A  323: c---      ln       - line number to use for filing
A  324:      integer ln
A  325:
A  326: c---      pres     - type value used for storing in RA file
A  327:      integer pres
A  328:
A  329:      data bcol/10,20,30,40,50/
A  330:      data ecol/18,28,38,48,67/
A  331:      data frstcl/0/ , opnrap/-1/
A  332:
A  333: c---      generate the md key locations for the data section of
A  334: c---      the observation
A  335:
A  336:      do 1 i = 1 , ndkeys
A  337:         mdlocs(i) = i + 7
A  338: 1    continue
A  339:
A  340: c---      assign values for decoder number, base md file, number
A  341: c---      of rows and columns to make md file based on input from
A  342: c---      data monitor
A  343:
A  344:      decnum = flags(3)
A  345:      mdbase = flags(4)
A  346:      nr     = flags(5)
A  347:      nc     = flags(6)
```

```
A  348:
A  349: c---      calculate the day of the month to julian day conversions
A  350: c---      this is necessary becaues the report retrieved from the
A  351: c---      raw text is day of month but we need julain day for filing.
A  352: c---      the 4 values stored will be value from julday and the
A  353: c---      3 following days.
A  354:
A  355:      yyddd(1) = julday
A  356:      do 5 i = 1 , 4
A  357:         call yddmy(yyddd(i),daylst(i),month,year)
A  358:         if (i .lt. 4)yyddd(i+1) = incday(yyddd(i),1)
A  359: 5    continue
A  360:
A  361: c---      initialize the arrays for the RA files if this is the
A  362: c---      first call to the decoder
A  363:
A  364:      ptrfil = cflags(6)
A  365:      if (frstcl .eq. 0)then
A  366:         frstcl = 1
A  367:         opnrap = mctxtopn(ptrfil , ptrhed , maxsta , idtab, numrap)
A  368:      endif
A  369:      m0trjdec = 0
A  370:
A  371:      stafil = cflags(5)
A  372:
A  373: c---      get the values currently stored in the status display
A  374:
A  375:      call m0rsdcd(' ',decnum,'DECO',bullbd,BBSIZE,stat)
A  376:
A  377: c---      acquire the list of stations/lat/lons possible for
A  378: c---      this data type , this is used to determine station lat/lon
A  379:
A  380:      ok = mclodids (stafil,maxsta,staton,slat,slon,numlod)
A  381:      call ddest('number of stations loaded ',numlod)
A  382:
A  383: c--- ****************************************************************
A  384: c--- *    the actual decoding begins here                           *
A  385: c--- ****************************************************************
A  386:
A  387: c---      we will look for a line that contains 6 digits
A  388: c---      immediately followed by a single character 'Z'. this will
A  389: c---      be the valid forecast times of the trajectory data.
A  390: c---      ex.  141200Z
A  391:
A  392:      linnum = 0
A  393:
A  394: 200 continue
A  395:
A  396: c---        grab one line at a time until you reach the end of
A  397: c---        the data block
A  398:
A  399:        linnum = linnum + 1
A  400:        if (linnum .le. nlines)then
A  401:
A  402: c---           tokanize the line into decodable groups
A  403:
A  404:           call ddest(cblk(linnum),0)
A  405:           numgrp = 0
A  406:           call movc(80,cblk,(linnum-1)*80,line,0)
A  407:           call crack(80,line,msg)
A  408:           ok = m0dcsplt(80,numgrp,0,dum)
A  409:
A  410: c---           print debug messages showing how the string was
A  411: c---           tokanized otherwise loop 210 is frivolous
A  412:
A  413:           do 210 i = 1 , numgrp
A  414:             call m0grbval(cvalue , value , i)
A  415:             write(cline,FMT='(4(a2,i5,1x),a12)')
A  416:      &            'gp',i,'pt=',point(i),'tp=',type(i),
A  417:      &            'nc=',numch(i),cvalue
A  418: c           call ddest(cline,0)
A  419: 210       continue
```

```
A  420:
A  421: c---          this is where you actually scan for 6 digits
A  422: c---          followed immediately by the character Z.
A  423:
A  424:          group  = 0
A  425:          numper = 0
A  426: 220      continue
A  427:
A  428: c---          scan through the tokanized groups of the line until
A  429: c---          you have no more groups to check for this line
A  430:
A  431:          group = group + 1
A  432:          if (group .le. numgrp)then
A  433:
A  434: c---          if this group is NOT 6 digits immediately followed
A  435: c---          by a single character, it cannot be the valid
A  436: c---          forecast time, so go check the next group
A  437:
A  438:               if (type(group) .ne. ADIGIT .or.
A  439:      &            numch(group) .ne. 6 .or.
A  440:      &            type(group+1) .ne. ACHAR .or.
A  441:      &            numch(group+1) .ne. 1)goto 220
A  442:
A  443: c---          if you have made it to here, you know you have
A  444: c---          a forecast period
A  445:
A  446:               numper = numper + 1
A  447:               call m0grbval(cvalue , value , group)
A  448:
A  449: c---          extract the valid date ,and convert to julian
A  450:
A  451:          tday = value / 10000
A  452:          do 230 j = 1 , 4
A  453:            if (daylst(j) .eq. tday)then
A  454:              tday = yyddd(j)
A  455:              goto 235
A  456:            endif
A  457: 230      continue
A  458: 235      continue
A  459:
A  460:               vday(numper) = tday
A  461:
A  462: c---          extract the valid time and convert to hhmmss
A  463:
A  464:               vtime(numper) = mod(value , 10000) * 100
A  465:
A  466: c---          if the number of periods is less than maxper,
A  467: c---          continue extracting periods from the line
A  468:
A  469:               if (numper .lt. maxper)goto 220
A  470:
A  471: c---          if you have made it to here, you have all the
A  472: c---          periods, so go to the next point in the decoding
A  473:
A  474:          goto 290
A  475:
A  476:       endif
A  477:
A  478: c---          if you have made it to here, you haven't found the
A  479: c---          forecast periods yet. go back and grab a new line.
A  480:
A  481:          goto 200
A  482:
A  483:       else
A  484:          call sdest('unable to find valid forecast labels',0)
A  485:          goto 999
A  486:       endif
A  487:
A  488: 290 continue
A  489:
A  490: c---     if you have made it to here, you have all the forecast
A  491: c---     dates and times in 'vday' and 'vtime'.
```

```
A  492:
A  493: c---      loop 295 is printing the valid times that were decoded
A  494:
A  495:      do 295 i = 1 , maxper
A  496:         write(cline,FMT='(a4,i2,1x,a5,i5,1x,a5,i6.6)')
A  497:       &      'per=',i,'vday=',vday(i),'vtim=',vtime(i)
A  498:         call ddest(cline,0)
A  499: 295 continue
A  500:
A  501: c---      now that we have the forecast information, we will scan
A  502: c---      each line for decodable data. Since this format only
A  503: c---      sends the station id once regardless the number of levels
A  504: c---      forecasted we will assume that lines that do not contain
A  505: c---      station ids have the same station id as the previous line
A  506:
A  507:      numid  = 0
A  508:      nostat = 0
A  509:      linnum = linnum + 1
A  510: 300 continue
A  511:
A  512: c---      loop through the remaining number of lines until you
A  513: c---      reach the end of the data block
A  514:
A  515:      linnum = linnum + 1
A  516:      if (linnum .le. nlines)then
A  517:
A  518: c---      initialize the output arrays to missing value codes
A  519:
A  520:      call zmiss(maxper , tlat , MISS)
A  521:      call zmiss(maxper , tlon , MISS)
A  522:      call zmiss(maxper , tpre , MISS)
A  523:      tt  = -9999.d0
A  524:      ttd = -9999.d0
A  525:
A  526: c---      decode the current line
A  527:
A  528:      ok = dectrj(cblk(linnum),ctemp,tlat,tlon,tpre,tlvl,tt,ttd)
A  529:
A  530: c---      if we haven't found a station id yet and this line
A  531: c---      does not include a station id then go grab the next
A  532: c---      line and start over.
A  533:
A  534:        if (nostat .eq. 1 .and. ok .ne. 1)goto 300
A  535:        nostat = 0
A  536:
A  537: c---      if the decoding detected a station id you must find
A  538: c---      the lat/lon of the station
A  539:
A  540:     if (ok .eq. 1)then
A  541:
A  542:        do 315 i = 1 , numlod
A  543:
A  544: c---      if a station match is found, assign the appropriate
A  545: c---      lat/lon, increment the number of stations found, and
A  546: c---      reset the level counter for the new station.
A  547:
A  548:         ctemp4 = clit(staton(i))
A  549:         if (ctemp(1:3) .eq. ctemp4(1:3))then
A  550:             numid = numid + 1
A  551:             cid(numid) = ctemp
A  552:             clat       = slat(i)
A  553:             clon       = slon(i)
A  554:             lev        = 0
A  555:             goto 316
A  556:         endif
A  557:
A  558: 315 continue
A  559:
A  560: c---      if the station was not found and the flag is set
A  561: c---      at the station to the monitor list
A  562:
```

```
A  563:          if (flags(2) .eq. 1)then
A  564:             id = lit(ctemp4)
A  565:             call m0idnew(id,cflags(3),1,'NEW ')
A  566:          endif
A  567:          nostat = 1
A  568:          goto 300
A  569:
A  570:      endif
A  571:
A  572: c---      if you have made it to here you have successfully decoded
A  573: c---      the entire observation.  clat/clon contain the station
A  574: c---      lat/lons you are currently processing
A  575:
A  576: 316 continue
A  577:
A  578: c---      move the station lat/lon into the final forecast period's
A  579: c---      slot in the temporary array
A  580:
A  581:       tlat(5) = clat
A  582:       tlon(5) = clon
A  583:
A  584: c---      increment the level number being viewed and moved the
A  585: c---      data from the temporay arrays into the permanent ones
A  586: c---      the value for lev will be used to indicate where in the
A  587: c---      repeat section of the md file to store this report
A  588:
A  589:       if (tlvl .eq. 'SFC')then
A  590:          lev = 1
A  591:          elseif (tlvl .eq. '850')then
A  592:          lev = 2
A  593:          else
A  594:          lev = 3
A  595:          endif
A  596:
A  597: c---      moved the data just decoded from temporary into permanent
A  598: c---      storage arrays.
A  599:
A  600:       t(lev,numid)       = tt
A  601:       td(lev,numid)      = ttd
A  602:       linloc(lev,numid) = linnum
A  603:       lvls(lev,numid)    = tlvl
A  604:       call movw(maxper,tlat,lat(1,lev,numid))
A  605:       call movw(maxper,tlon,lon(1,lev,numid))
A  606:       call movw(maxper,tpre,pre(1,lev,numid))
A  607:
A  608: c---      go back up looking for more stations
A  609:
A  610:       goto 300
A  611:
A  612:      endif
A  613:
A  614: c---      if you have made it to here all of the important arrays
A  615: c---      have been filled so you can commence filing the reports
A  616:
A  617: c---      first check to see if the MD file you will be filing to
A  618: c---      exists. if it doesn't build the row/column headers
A  619:
A  620:       mdfile = mdbase + mod(julday,10)
A  621:       if (mdfile .eq. mdbase)mdfile = mdfile + 10
A  622:
A  623:       ok = makmdf(mdfile,nr,nc,julday)
A  624:
A  625: c---   ************************************************************
A  626: c---   * file decoded reports in MD and RA files                   *
A  627: c---   ************************************************************
A  628:
A  629: c---      loop through all of the stations that were decoded
A  630:
A  631:      do 400 i = 1 , numid
A  632:         write(cline,FMT='(a3,a4)')'id=',cid(i)
A  633:         call ddest(cline,0)
A  634:
```

```
A  635: c---          locate the appropriate column number to file data in
A  636:
A  637:         id    = lit(cid(i))
A  638:         mdcol = m0loccol (id , mdfile , 1 , 0 , 0 , 0 ,
A  639:        &                  0 , 1 , mdlist , maxsta , mdidtb , dum1 ,
A  640:        &                  dum2 , dum3 , dum4 , dum5 , idinfo)
A  641:
A  642: c---          if the station was not found go to the next station
A  643:
A  644:         if (mdcol .le. 0)then
A  645:            goto 400
A  646:         endif
A  647:
A  648: c---          figure out the appropriate row number
A  649: c---          if the first VTIM is 12Z that means this is the
A  650: c---          output from the 12z model run so begin filing
A  651: c---          after the 5 00Z rows
A  652:
A  653:         mdrow = 0
A  654:         if (vtime(1) .gt. 0)mdrow = 5
A  655:
A  656:         record(1) = 0
A  657:         do 440 k = 1 , maxper
A  658:
A  659:            mdrow = mdrow + 1
A  660:
A  661: c---          loop through the forecast periods calculating the
A  662: c---          appropriate row number and filling up the record
A  663: c---          array for the md file.  This section also files
A  664: c---          in the RA file if appropriate.
A  665:
A  666:            do 420 j = 1 , maxlev
A  667:
A  668:
A  669: c---             fill the md data output array
A  670:
A  671: c---       the storage format for the output array 'record' is:
A  672: c---       1 - MOD flag
A  673: c---       2 - temperature from SFC (only reported for last period)
A  674: c---       3 - dew point from SFC (only reported for last period)
A  675: c---       4 - latitude from SFC
A  676: c---       5 - longitude from SFC
A  677: c---       6 - pressure level SFC
A  678: c---       7 - ending pressure SFC
A  679: c---       8 - temperature from 850
A  680: c---       9 - dew point from 850
A  681: c---       10- latitude from 850
A  682: c---       11- longitude from 850
A  683: c---       12- pressure level 850
A  684: c---       13- ending pressure 850
A  685: c---       14- temperature from 700
A  686: c---       15- dew point from 700
A  687: c---       16- latitude from 700
A  688: c---       17- longitude from 700
A  689: c---       18- pressure level 700
A  690: c---       19- ending pressure 700
A  691:
A  692: c---          convert the temp and dewpoint to kelvin and put
A  693: c---          in the appropriate record location for the final
A  694: c---          forecast period.
A  695:
A  696:            pt = 1 + (j - 1) * 6
A  697:            record(pt+1) = MISS
A  698:            record(pt+2) = MISS
A  699:            if (k .eq. maxper)then
A  700:               t(j,i)     = 273.16 + t(j,i)
A  701:               td(j,i)    = 273.16 + td(j,i)
A  702:               record(pt+1) = int(t(j,i)  * 100.d0)
A  703:               record(pt+2) = int(td(j,i) * 100.d0)
A  704:
A  705: c---                perform gross error checks on the temperature
A  706: c---                and dew point
```

```
A  707:
A  708:                ok = mcgrsscl(record(pt+1),2,MISS,'T   ','K   ',0)
A  709:                ok = mcgrsscl(record(pt+2),2,MISS,'TD  ','K   ',0)
A  710:
A  711:             endif
A  712:
A  713:             record(pt+3)= lat(k,j,i)
A  714:             record(pt+4)= lon(k,j,i)
A  715:             record(pt+5)= pre(k,j,i)
A  716:             record(pt+6)= lit(lvls(j,i))
A  717:
A  718: c---              if the RA text file was successfully opened
A  719: c---              file the portion of the observation having
A  720: c---              to do with this station, level, and forecast
A  721: c---              period
A  722:
A  723:             if (opnrap .ge. 0)then
A  724:
A  725: c---              build the domain within the data block that
A  726: c---              should be filed
A  727: c---              since this is a bit of a flaky output, we will
A  728: c---              build a temporary string which is what we will
A  729: c---              actually file
A  730:
A  731:                ln    = linloc(j,i)
A  732:                cline = cid(i)(1:4)//cblk(ln)(6:9)
A  733:       &                          //cblk(ln)(bcol(k):ecol(k))
A  734:
A  735: c---              blkdom contains the list of what section of the
A  736: c---              raw observation stored in 'cline' is to be
A  737: c---              included.  This includes row/col information,
A  738: c---              not to be confused with the MD row and col
A  739:
A  740:                blkdom(1) = 1
A  741:                blkdom(2) = blkdom(1)
A  742:                blkdom(3) = 1
A  743:                blkdom(4) = 80
A  744:
A  745: c---              build the time domain information.
A  746: c---              we store 1001 for SFC level.
A  747:
A  748:                pres = 1001
A  749:                if (lvls(j,i) .eq. '700')then
A  750:                   pres = 700
A  751:                elseif (lvls(j,i) .eq. '850')then
A  752:                   pres = 850
A  753:                endif
A  754:
A  755: c---              timdom stores pertinent meta-data about the
A  756: c---              observation. What type of report (SFC, 850, 700)
A  757: c---              valid day/time
A  758:
A  759:                timdom(1) = pres
A  760:                timdom(2) = 1
A  761:                timdom(3) = vday(k)
A  762:                timdom(4) = vtime(k)
A  763:                timdom(5) = timdom(4)
A  764:                timdom(6) = timdom(3)
A  765:                timdom(7) = timdom(4)
A  766:                timdom(8) = timdom(5)
A  767:
A  768:                id = lit(cid(i))
A  769:
A  770: c---              write the observation to the RA file
A  771:
A  772:                ok = mctxtwrt(ptrfil,ptrhed,cline,blkdom,
A  773:      &                       id,timdom,maxsta,numrap,idtab)
A  774:                call ddest(cline(1:60),ok)
A  775:
A  776:             endif
A  777:
A  778: 420     continue
```

```
A  779:
A  780: c---          at this point the entire data record should be
A  781: c---          filled so we can write the output to the md file
A  782:
A  783:        ok = mdo(mdfile,mdrow,mdcol,ndkeys,mdlocs,record)
A  784:
A  785:        write(cline,FMT='(a4,i2,1x,2(a2,i4,1x))')
A  786:     &      'mdo=',ok,'r=',mdrow,'c=',mdcol
A  787:        call ddest(cline,0)
A  788:
A  789: c---          update the buffer for the status display and output
A  790: c---          it to the window
A  791:
A  792:        bullbd(BBON)    = 1
A  793:        call gettim(bullbd(BBTIME))
A  794:        call getday(bullbd(BBDAY))
A  795:        bullbd(BBMD)    = mdfile
A  796:        bullbd(BBROW)   = mdrow
A  797:        bullbd(BBCOL)   = mdcol
A  798:        call movcw('TRAJECT ',bullbd(BBTASK))
A  799:        call movcw('        ',bullbd(BBTEXT))
A  800:        call m0wsdcd(' ',decnum,'DECO',bullbd,BBSIZE,stat)
A  801:
A  802: 440  continue
A  803:
A  804: 400 continue
A  805: 999 continue
A  806:     return
A  807:     end
A  808:
A  809: c $ makmdf - make the md file with the necessary row and colum
A  810: c $           headers
A  811: c $
A  812: c $ integer function makmdf(integer mdfile , integer nr ,
A  813: c $                           integer nc , integer julday)
A  814: c $
A  815: c $ input:
A  816: c $   mdfile  md file number to build
A  817: c $   nr      number of rows to make md file
A  818: c $   nc      number of columns to make md file
A  819: c $   julday  julian day md file is valid for
A  820: c $
A  821: c $ return values:
A  822: c $   0 - md file already exists
A  823: c $   1 - md file made successfully
A  824: c $  <0 - error while making md file
A  825:
A  826:     integer function makmdf(mdfile , nr , nc , julday)
A  827:
A  828:     implicit none
A  829:     include 'xcd.inc'
A  830:
A  831: c---      nkeys   - number of keys to get from master station list
A  832: c---      maxsta  - maximum number of stations that can be stored
A  833: c---                in the column headers
A  834:
A  835:     integer nkeys
A  836:     integer maxsta
A  837:     parameter (nkeys = 3 , maxsta = 600)
A  838:
A  839: c---      nrkeys  - number of keys in the row header
A  840: c---      nckeys  - number of keys in the column header
A  841:     integer nrkeys
A  842:     integer nckeys
A  843:     parameter (nrkeys = 4 , nckeys = 3)
A  844:
A  845: c---      mdfile  - md file to be built
A  846: c---      nr      - number of rows to make the md file
A  847: c---      nc      - number of columns to make the md file
A  848:     integer mdfile
A  849:     integer nr
A  850:     integer nc
```

```
A  851:
A  852: c---      julday  - julian day this md file represents
A  853: c---      record  - storage array for output to the md file
A  854:      integer julday
A  855:      integer record(nrkeys+nckeys)
A  856:
A  857: c---      title   - array containing the title for the md file
A  858:      integer title(8)
A  859:
A  860: c---      rklocs  - row key locations in the md file
A  861: c---      cklocs  - columns key locations in the md file
A  862:      integer rklocs(nrkeys)
A  863:      integer cklocs(nckeys)
A  864:
A  865: c---      filnam  - temp string containing lw filename of md file
A  866:      character*12 filnam
A  867:
A  868: c---      idfile  - id file to use to build station list and
A  869: c---                column headers from
A  870:      character*12 idfile
A  871:
A  872: c---      ckeys   - list of keys to get from station id list
A  873:      character*4 ckeys(nkeys)
A  874:
A  875: c---      cdate   - string to contain date for title of md file
A  876:      character*40 cdate
A  877:
A  878: c---      cttl    - string containing md file title
A  879:      character*40 cttl
A  880:
A  881: c---      filtmp  - integer array name of md file
A  882:      integer filtmp(3)
A  883:
A  884: c---      finc    - forecast increment between md file rows
A  885:      integer finc
A  886:
A  887: c---      ids     - array containing complete list of station ids
A  888: c---                to include in column headers
A  889: c---      ele     - array containing complete list of elevations
A  890: c---                to include in column headers
A  891: c---      st      - array containing complete list of states to
A  892: c---                include in column headers
A  893:      integer ids(maxsta)
A  894:      integer ele(maxsta)
A  895:      integer st(maxsta)
A  896:
A  897: c---      lwfile  - function declaration
A  898: c---      lit     - function declaration
A  899: c---      mcydd2ch- function declaration
A  900: c---      mdmake  - function declaration
A  901: c---      m0bildid- function declaration
A  902: c---      mdo     - function declaration
A  903: c---      mcinchr - function declaration
A  904: c---      lwi     - function declaration
A  905:      integer lwfile
A  906:      integer lit
A  907:      integer mcydd2ch
A  908:      integer mdmake
A  909:      integer m0bildid
A  910:      integer mdo
A  911:      integer mcinchr
A  912:      integer lwi
A  913:
A  914: c---      schema  - md schema name
A  915: c---      ok      - function return value
A  916: c---      nsta    - number of stations returned from m0bildid
A  917: c---      start   - starting word in station id table to begin
A  918: c---                loading station information
A  919: c---      i       - loop counter
A  920:      integer schema
A  921:      integer ok
A  922:      integer nsta
```

```
A  923:      integer start
A  924:      integer i
A  925:
A  926:      equivalence (filnam , filtmp)
A  927:      data rklocs/1,2,3,4/ , cklocs/5,6,7/
A  928:
A  929: c---      build the file name
A  930:
A  931:      call mdname(mdfile , filtmp)
A  932:
A  933: c---      check to see if the file already exists
A  934:
A  935:      makmdf = 0
A  936:      if (lwfile(filnam) .eq. 1)goto 999
A  937:
A  938: c---      if you have made it to here, the file doesn't exist so
A  939: c---      create the md file with row and column headers
A  940:
A  941:      makmdf = -1
A  942:
A  943: c---      build the md file title and make the md file
A  944:      schema = lit('TRAJ')
A  945:      ok     = mcydd2ch(julday , 4 , cdate)
A  946:      cttl   = 'NGM Traj Fcst: '//cdate
A  947:      call movcw(cttl , title)
A  948:      ok = mdmake (mdfile , schema , 0 , nr , nc , julday , title)
A  949:      if (ok .lt. 0)goto 999
A  950:
A  951: c---      first we must build a station list. For this exercise
A  952: c---      we will use the station ids used for the FOUS14 decoder
A  953:
A  954:      idfile   = 'FOUS51.IDT'
A  955:      ckeys(1) = 'CID1'
A  956:      ckeys(2) = 'ST  '
A  957:      ckeys(3) = 'ELE1'
A  958:
A  959:      nsta = m0bildid(' ',1,idfile,ckeys,nkeys,'DDS ',0,'FOUS14',
A  960:       &                    1,1,5)
A  961:
A  962: c---      check for errors in building station id table
A  963:
A  964:      if (nsta .le. 0)goto 999
A  965:
A  966:      makmdf = makmdf - 1
A  967:      if (nsta .gt. maxsta)goto 999
A  968:
A  969: c---      load the station list into the appropriate arrays
A  970:
A  971:      makmdf = makmdf - 1
A  972:      start = 1024
A  973:      if (lwi(idfile,              start , nsta , ids) .lt. 0)goto 999
A  974:      if (lwi(idfile, 1 * nsta + start , nsta , st)  .lt. 0)goto 999
A  975:      if (lwi(idfile, 2 * nsta + start , nsta , ele) .lt. 0)goto 999
A  976:
A  977: c---      build the row headers
A  978:
A  979:      record(1) = julday
A  980:      record(2) = 0
A  981:      finc      = 0
A  982:      do 10 i = 1 , nr
A  983:
A  984: c---      if i = 6 that means that we are starting to build the
A  985: c---      row headers for the 12Z model run
A  986:
A  987:       if (i .eq. 6)then
A  988:          record(2) = 120000
A  989:          finc      = 0
A  990:       endif
A  991:
A  992: c---      increment the forecast time
A  993:
A  994:       ok = mcinchr(record(1),record(2),finc,record(3),record(4))
```

```
A  995:
A  996: c---     write to the row header
A  997:
A  998:       ok = mdo(mdfile , i , 0 , nrkeys , rklocs , record)
A  999:       finc = finc + 6
A 1000: 10 continue
A 1001:
A 1002: c---     build the column headers
A 1003:
A 1004:       do 100 i = 1 , nsta
A 1005:          record(1) = ids(i)
A 1006:          record(2) = st(i)
A 1007:          record(3) = ele(i)
A 1008:          ok = mdo(mdfile , 0 , i , nckeys , cklocs , record)
A 1009: 100 continue
A 1010:
A 1011:      makmdf = 1
A 1012: 999 continue
A 1013:      return
A 1014:      end
A 1015:
A 1016: c $ dectrj - decodes a line of the trajectory forecast
A 1017: c $
A 1018: c $ integer function dectrj(line,cid,lat,lon,
A 1019: c $                                pre,level,t,td)
A 1020: c $ input:
A 1021: c $   line    c(*)- line to be decoded
A 1022: c $ output:
A 1023: c $   cid     c(*)- station id
A 1024: c $   lat     i(*)- array of decoded latitudes
A 1025: c $   lon     i(*)- array of decoded longitudes
A 1026: c $   pre     i(*)- array of presures
A 1027: c $   t       dp  - final temperature
A 1028: c $   td      dp  - final dewpoint
A 1029: c $ return values:
A 1030: c $   0  - success, no station id on this line
A 1031: c $   1  - success, station id on this line
A 1032:
A 1033:      integer function dectrj(line,cid,lat,lon,pre,level,t,td)
A 1034:      implicit none
A 1035:
A 1036: c---     numper  - number of periods in the for aaaoooppp
A 1037:      integer numper
A 1038:      parameter (numper = 4)
A 1039:
A 1040: c---     line    - line to decode
A 1041: c---     cid     - station id decoded (if found)
A 1042: c---     level   - level decoded (SFC, 850, 700)
A 1043:
A 1044:      character*(*) line
A 1045:      character*(*) cid
A 1046:      character*(*) level
A 1047:
A 1048: c---     fltval  - temporary floating point variable
A 1049: c---     t       - temperature decoded from report
A 1050: c---     td      - dew point decoded from report
A 1051:
A 1052:      double precision fltval
A 1053:      double precision t
A 1054:      double precision td
A 1055:
A 1056: c---     ctemp   - temporary character string
A 1057:      character*12 ctemp
A 1058:
A 1059: c---     lat     - array of decoded latitudes
A 1060: c---     lon     - array of decoded longitudes
A 1061: c---     pre     - array of decoded pressures
A 1062:      integer lat(*)
A 1063:      integer lon(*)
A 1064:      integer pre(*)
A 1065:
```

```
A 1066: c---      latcol  - column number where latitudes begin
A 1067: c---      loncol  - column number where longitudes begin
A 1068: c---      pcol    - column number where pressures begin
A 1069:
A 1070:      integer latcol(numper)
A 1071:      integer loncol(numper)
A 1072:      integer pcol(numper)
A 1073:
A 1074:      integer ok
A 1075:      integer strsta
A 1076:      integer intval
A 1077:      integer intsta
A 1078:      integer fltsta
A 1079:      integer mcextrln
A 1080:      integer i
A 1081:
A 1082:      data latcol/10,20,30,40/
A 1083:      data loncol/13,23,33,43/
A 1084:      data pcol  /16,26,36,46/
A 1085:
A 1086:      dectrj = 0
A 1087:
A 1088: c---      extract the station id if it exists from column 2-4
A 1089:
A 1090:      ok = mcextrln(line,2,4,cid,strsta,intval,intsta,fltval,fltsta)
A 1091:
A 1092: c---      if the line contains a station id, return 1
A 1093:
A 1094:      if (cid .ne. ' ')dectrj = 1
A 1095:
A 1096: c---      loop 10 scans through the line decoding the first 4
A 1097: c---      forecast periods
A 1098:
A 1099:      do 10 i = 1 , numper
A 1100:
A 1101: c---      extract the latitude
A 1102:
A 1103:      ok = mcextrln(line,latcol(i),latcol(i)+2,ctemp,strsta,
A 1104:     &                  intval,intsta,fltval,fltsta)
A 1105:      lat(i) = intval * 1000
A 1106:
A 1107: c---      extract the longitude, we may have to convert it
A 1108: c---      if the value extracted is less than 600 it implies
A 1109: c---      that the value is actually preceeded by 100.
A 1110: c---      example:
A 1111: c---      if the value is 765 it implies a value of 76.5
A 1112: c---      if the value is 116 that implies a value of 111.6
A 1113:
A 1114:      ok = mcextrln(line,loncol(i),loncol(i)+2,ctemp,strsta,
A 1115:     &                  intval,intsta,fltval,fltsta)
A 1116:
A 1117:      if (intval .ge. 600)then
A 1118:         lon(i) = intval * 1000
A 1119:      else
A 1120:         lon(i) = 1000000 + (intval * 1000)
A 1121:      endif
A 1122:
A 1123: c---      extract the pressure
A 1124:
A 1125:      ok = mcextrln(line,pcol(i),pcol(i)+2,ctemp,strsta,
A 1126:     &                  intval,intsta,fltval,fltsta)
A 1127:
A 1128: c---      if the value extracted is less that 100 assume that
A 1129: c---      the data is actually for a level above 1000mb
A 1130:
A 1131:      if (intval .lt. 100)intval = intval + 1000
A 1132:      pre(i) = intval
A 1133:
A 1134: 10 continue
A 1135:
A 1136: c---      now decode the final state of the parcel. put the
A 1137: c---      results in position 5 of the lat, lon, and pre arrays.
```

```
A 1138:
A 1139: c---      extract the ending parcel level
A 1140:
A 1141:      ok = mcextrln(line,6,8,ctemp,strsta,
A 1142:     &                  intval,intsta,fltval,fltsta)
A 1143:
A 1144:      level = ctemp
A 1145:
A 1146: c---      if this is not the SFC level put this value in pre(5)
A 1147: c---      otherwise read the pressure value from SFC location
A 1148:
A 1149:      if (ctemp .eq. 'SFC')then
A 1150:
A 1151:         ok = mcextrln(line,50,52,ctemp,strsta,intval,intsta,
A 1152:       &                    fltval,fltsta)
A 1153:
A 1154:      endif
A 1155:
A 1156:      pre(5) = intval
A 1157:
A 1158: c---      extract the temperature
A 1159:
A 1160:      ok = mcextrln(line,53,57,ctemp,strsta,intval,intsta,
A 1161:     &                  t,fltsta)
A 1162:
A 1163: c---      extract the dewpoint
A 1164:
A 1165:      ok = mcextrln(line,59,63,ctemp,strsta,intval,intsta,
A 1166:     &                  td,fltsta)
A 1167:
A 1168: 999 continue
A 1169:      return
A 1170:      end
```

Source code used to run m0trjdec from the command line

```
B     0: c $ trjdec - demonstration command for trajectory decoder
B     1: c $ this command runs the trajectory forecast decoder m0trjdec
B     2: c $ to file information in an MD file format and RA format
B     3: c $ for quick access.
B     4:
B     5:      subroutine main0
B     6:      implicit integer (a-z)
B     7:      parameter (maxlin = 200 , idxsiz = 16)
B     8:      integer idxblk(idxsiz) , flags(8)
B     9:      character*4 header , clit , src , origin , cindex , wmo
B    10:      character*80 cline , file , cblk(maxlin)
B    11:      character*12 cflags(8)
B    12:      double precision dtime
B    13:      data cindex/'FO'/
B    14:      data header/'FOUS'/
B    15:      data minprd/50/ , maxprd/57/
B    16:      data dtime/13.d0/
B    17:
B    18:      call getday(today)
B    19:      call gettim(now)
B    20:      yyddd      = ikwp('DAY',1,today)
B    21:      time       = ikwphr('TIME',1,now)
B    22:
B    23: c---           set the appropriate values for the cflags array
B    24:
B    25:      cflags(3) = 'NEWFO50.IDM'
B    26:      cflags(5) = 'FO14DEC.IDT'
B    27:      cflags(6) = 'FO50.RAP'
B    28:      flags(2)  = 1
B    29:      flags(3)  = 12
B    30:      flags(4)  = 100
B    31:      flags(5)  = 10
B    32:      flags(6)  = 600
B    33:      flag      = 0
```

```
B    34:      found     = 0
B    35: 100 continue
B    36:
B    37: c---           get the next available index block
B    38:
B    39:       gidx = mcgetidx(yyddd,time,dtime,1,cindex,idxblk,flag)
B    40:
B    41: c---           if there was an error or we have no more data
B    42:
B    43:       if (gidx .le. 0)goto 999
B    44:
B    45:       src    = clit(idxblk(1))
B    46:       wmo    = clit(idxblk(5))
B    47:       origin = clit(idxblk(7))
B    48:
B    49:        write(cline,FMT='(a4,1x,i5,1x,i9,1x,i6,1x,a4,1x,i2,1x,a4)')
B    50:     &     src,idxblk(2),idxblk(3),idxblk(4),wmo,idxblk(6),origin
B    51:       call ddest(cline,0)
B    52:
B    53: c---           make certain the wmo header products match
B    54:
B    55:       if (wmo .ne. header)goto 100
B    56:       if (idxblk(6) .lt. minprd .or.
B    57:     &         idxblk(6) .gt. maxprd)goto 100
B    58:
B    59:       found = found + 1
B    60:
B    61: c---           if you make it to here, you know you have the data you
B    62: c---           are interested in, so load the text and decode it
B    63:
B    64:       call m0splnam(idxblk(3) , src , yyddd , file , ptr)
B    65:       ok = mclddatb(file , ptr , idxblk(2),maxlin*80,80,cblk)
B    66:       nlines = (idxblk(2) -1) / 80 + 1
B    67:
B    68:       do 200 i = 1 , nlines
B    69:          call ddest(cblk(i),0)
B    70: 200   continue
B    71:       call ddest('----------',0)
B    72:
B    73:       ok = m0trjdec(cblk(1),cblk(2),nlines-1,yyddd,time,'     ',
B    74:     &                    flags,cflags)
B    75:
B    76:       if (found .lt. (maxprd-minprd+1))goto 100
B    77:
B    78: 999 continue
B    79:       call edest('done',0)
B    80:       return
B    81:       end
```

Source code used to run m0trjdec from a data monitor, dmlocal.pgm

```
C     0: C   THIS IS SSEC PROPRIETARY SOFTWARE - ITS USE IS RESTRICTED.
C     1:
C     2: C ***McIDAS Revision History ***
C     3: C ***McIDAS Revision History ***
C     4:
C     5:      SUBROUTINE main0
C     6:
C     7:      IMPLICIT INTEGER (a-z)
C     8:      INCLUDE 'xcd.inc'
C     9:
C    10: c   parameter definitions:
C    11: c      maxlin - the maximum number of lines that can be decoded
C    12: c               by one decoder call
C    13: c      numdec - the number of decoders that are called in this
C    14: c               particular processing data monitor
C    15: c      maxidx - the maximum number of different indices that can
C    16: c               be used by each decoder.  this value is wired
```

```
C   17: c                into the data structure for decinfo.dat and
C   18: c                cannot be altered.
C   19: c       mxiflg - the maximum number of integer decoding flags
C   20: c                that can be set for a particular decoder.  this
C   21: c                value is wired into the data structure for
C   22: c                decinfo.dat and cannot be altered.
C   23: c       mxcflg - the maximum number of character decoding flags
C   24: c                that can be set for a particular decoder.  this
C   25: c                value is wired into the data structure for
C   26: c                decinfo.dat and cannot be altered.
C   27: c       maxwmo - the maximum number of wmo headers
C   28: c                that can be set for a particular decoder.  this
C   29: c                value is wired into the data structure for
C   30: c                decinfo.dat and cannot be altered.
C   31: c       maxorg - the maximum number of station origins
C   32: c                that can be set for a particular decoder.
C   33: c       byprln - number of bytes per line
C   34:
C   35:
C   36: c  +-------------------------------------------------------+
C   37: c  | If you build your own decoding task using this one as a |
C   38: c  | template, the only things that have to be changed are  |
C   39: c  | 'numdec' - the number of decoders this task processes  |
C   40: c  | 'task'   - the data monitor name that is running       |
C   41: c  | 'decnam'   the decoders to be processed for this    task |
C   42: c  |                                                        |
C   43: c  | And the actual decoder calls themselves                |
C   44: c  +-------------------------------------------------------+
C   45:
C   46:      PARAMETER (idxsiz = 16)
C   47       PARAMETER (maxlin = 1000 , numdec = 1 , byprln = 80)
C   48:      PARAMETER (maxidx = 8 , mxiflg = 16 , mxcflg = 8 , maxwmo = 20)
C   49:      PARAMETER (maxorg = 128)
C   50:      PARAMETER (maxbyt = maxlin * byprln)
C   51:
C   52:      CHARACTER*12 task , decnam(numdec) , cuser
C   53:      PARAMETER (task = 'DMLOCAL')
C   54:
C   55: c---           wmo is a list of the maxwmo character portions
C   56: c---           of the acceptable headers for each of the
C   57: c---           numdec decoders
C   58:
C   59:      CHARACTER*4  wmo(maxwmo,numdec)
C   60:
C   61: c---           orglst is a list of the maxorg station origins
C   62: c---           that are acceptable for each of the numdec
C   63: c---           decoders
C   64:
C   65:      CHARACTER*4  orglst(maxorg,numdec)
C   66:
C   67: c---           namidx is a list of the maxidx indices that are
C   68: c---           to be processed for the numdec decoders
C   69:
C   70:      CHARACTER*4  namidx(maxidx,numdec)
C   71:
C   72: c---           descrp is a list of the decoder titles
C   73:
C   74:      CHARACTER*80 descrp(numdec)
C   75:
C   76: c---           cflags is a list of the mxcflg character
C   77: c---           string flags used by the numdec decoders
C   78:
C   79:      CHARACTER*12 cflags(mxcflg,numdec)
C   80:
C   81: c---           cidxfl is a list of the mxcflg index file names
C   82: c---           used by the numdec decoders
C   83:
C   84:      CHARACTER*12 cidxfl(maxidx,numdec)
C   85:
C   86: c---           cblk is the character array that will contain
C   87: c---           the data to be decoded
C   88:
```

```
C   89:      CHARACTER*80 cblk(maxlin)
C   90:
C   91: c---           begptr and lasptr are the beginning and ending
C   92: c---           index pointer locations processed for each of
C   93: c---           the maxidx indices for each numdec decoders.
C   94: c---           begptr is initialized to 0 if the task is just
C   95: c---           starting or if the day has changed. lasptr is
C   96: c---           initialized to -1.
C   97: c---           calflg is the flag indicating initial
C   98: c---           processing procedures, see m0nxtidx
C   99:
C  100:      INTEGER begptr(maxidx,numdec) , lasptr(maxidx,numdec)
C  101:      INTEGER calflg(maxidx,numdec)
C  102:
C  103: c---           flags is a list of mxiflg integer flags used
C  104: c---           by the numdec decoders
C  105:
C  106:      INTEGER flags(mxiflg,numdec)
C  107:
C  108: c---           numwmo is the number of wmo headers defined
C  109: c---           for the numdec decoders
C  110:
C  111:      INTEGER numwmo(numdec)
C  112:
C  113: c---           numorg is the number of station origins
C  114: c---           defined for the numdec decoders
C  115:
C  116:      INTEGER numorg(numdec)
C  117:
C  118: c---           numidx is the number of indices that are
C  119: c---           actually defined for the numdec decoders
C  120:
C  121:      INTEGER numidx(maxidx)
C  122:
C  123: c---           decsta is the decoder status flag for the
C  124: c---           numdec decoders
C  125:
C  126:      INTEGER decsta(numdec)
C  127:
C  128: c---           minprd and maxprd are the minimum and
C  129: c---           maximum product numbers that are to be
C  130: c---           decoded for the numdec decoders
C  131:
C  132:      INTEGER minprd(maxwmo,numdec)
C  133:      INTEGER maxprd(maxwmo,numdec)
C  134:
C  135: c---           init is an initialization flag set to 0
C  136: c---           any time the day changes for a specific
C  137: c---           index/decoder pair
C  138:
C  139:      INTEGER init(maxidx,numdec)
C  140:
C  141: c---           idxblk is the index directory for the
C  142: c---           data block being processed
C  143:
C  144:      INTEGER idxblk(idxsiz)
C  145:
C  146:      CHARACTER*12 cfu , chead , cspool , cbull1 , cbull2 , cdnum ,
C  147:     &             cerror , ctemp , cfj
C  148:
C  149:      CHARACTER*80 cline
C  150:
C  151:      CHARACTER*4  circit , cwmo , corgin , clit , afos , afostn ,
C  152:     &             afoorg
C  153:
C  154:      LOGICAL difday
C  155:      DATA stlnot/0/
C  156:
C  157:      DATA decnam /'TRJDEC      '/
C  158:
C  159: c---           setting a debugging keyword
C  160:
```

```
C  161:      dbgflg = ikwp('DEB',1,0)
C  162:
C  163: c---          if mxstrt is set to -1 then this decoding
C  164: c---          task will run indifinitely.  otherwise
C  165: c---          it will end normally after the routine
C  166: c---          sleep has been called mxstrt times
C  167:
C  168:      mxstrt = ikwp('RESTART' , 1 , 500)
C  169:
C  170:      call getday(yyddd)
C  171:
C  172:      cerror = task
C  173:      len = min0(nchars(cerror,ib,ie),8)
C  174:      cerror = cerror(1:len)//'.ERR'
C  175:
C  176: c---          make certain the correct unix login is used
C  177:
C  178:      ok = m0oprchk(cuser,1)
C  179:      if (ok .eq. 0)goto 2000
C  180:
C  181:      numoff = 0
C  182:
C  183:      do 100 dec = 1 , numdec
C  184:
C  185: c---          get the decoder configuration information
C  186:
C  187:       active = m0dcinfo(FDCINF , task , decnam(dec) ,
C  188:     &               tsksta , decsta(dec) ,
C  189:     &               maxidx , numidx(dec) , namidx(1,dec) ,
C  190:     &               mxcflg , cflags(1,dec) ,
C  191:     &               mxiflg , flags(1,dec) ,
C  192:     &               maxwmo , numwmo(dec) , wmo(1,dec) ,
C  193:     &               minprd(1,dec) , maxprd(1,dec) ,
C  194:     &               maxorg , numorg(dec),orglst(1,dec),descrp(dec))
C  195:
C  196:       if (tsksta .lt. 0)then
C  197:          call edest('Data Monitor '//task//'is inactive '//cerror,
C  198:     &          tsksta)
C  199:          goto 2000
C  200:       endif
C  201:
C  202: c---          output any errors generated by m0dcinfo
C  203:
C  204:       if (active .eq. -1)then
C  205:          call edest('Unable to find '//task//' and '//decnam(dec)
C  206:     &              //' in '//FDCINF,0)
C  207:          goto 2000
C  208:       endif
C  209:
C  210: c---          if no indices are defined, the decoder does
C  211: c---          not know where to look for data, so
C  212: c---          exit out
C  213:
C  214:      if (numidx(dec) .le. 0)then
C  215:         call edest('No indices defined for '//decnam(dec),0)
C  216:         goto 2000
C  217:      endif
C  218:
C  219: c---          if the decoder is currently labeled inactive
C  220: c---          notify the starter and exit out if no decoders
C  221: c---          are labeled active
C  222:
C  223:      if (decsta(dec) .lt. 0)then
C  224:
C  225:          numoff = numoff + 1
C  226:          call edest(decnam(dec)//' is labeled inactive:  '
C  227:     &              //descrp(dec),0)
C  228:          if (numoff .eq. numdec)goto 2000
C  229:
C  230:      elseif (decsta(dec) .eq. 0)then
C  231:
C  232:          call edest('No decoder found in '//task//' called '
```

```
C  233:     &                    //decnam(dec),0)
C  234:           goto 2000
C  235:
C  236:       else
C  237:           call sdest(task//' Starting: '//descrp(dec),0)
C  238:       endif
C  239:
C  240: c---           initialize the things that are index/decoder
C  241: c---           dependent
C  242:
C  243:       do 110 idx = 1 , numidx(dec)
C  244:
C  245:           calflg(idx,dec) = -2
C  246:           begptr(idx,dec) = 0
C  247:           lasptr(idx,dec) = -1
C  248:           init(idx,dec)   = 0
C  249:           chead = namidx(idx,dec)
C  250:           call m0idxnam(chead,yyddd,'     ',cidxfl(idx,dec))
C  251:
C  252: 110   continue
C  253:
C  254: c---           add startup information to the error
C  255: c---           message file
C  256:
C  257:       call m0rsdcd('  ',flags(3,dec),'DECO',bullbd,BBSIZE,kstat)
C  258:       cbull1 = cfu(bullbd(BBBPTR))
C  259:       cbull2 = cfu(bullbd(BBLPTR))
C  260:       cdnum  = cfu(flags(3,dec))
C  261:       cline  = 'Started '//decnam(dec)//cbull1(1:10)
C  262:     &          //cbull2(1:10)//cdnum
C  263:       call mcermess(1,cerror,cline)
C  264:
C  265: 100   continue
C  266:
C  267: c---           difday - flag set to true if the data-day
C  268: c---           currently being processed is differant from
C  269: c---           the system day. this was required to insure
C  270: c---           that all the text from one spool file is
C  271: c---           processed before going on to the next spool
C  272: c---           file.
C  273:
C  274:       difday = .false.
C  275:
C  276: c---           statement #5 is only accessed if no new data
C  277: c---           has come in in a while or if the task has been
C  278: c---           asleep.
C  279:
C  280: c---           rstart is a counter used to determine when
C  281: c---           this task is to restart
C  282:
C  283:       rstart = 0
C  284:
C  285: 5     continue
C  286:
C  287:           call getday(curday)
C  288:
C  289: c---           if the system day is differant from the data-day
C  290: c---           set the difday flag to true
C  291:
C  292:           if (yyddd .ne. curday)difday = .true.
C  293:
C  294: c---           loop 200 scans through each of the defined
C  295: c---           decoders
C  296:
C  297:           do 200 dec = 1 , numdec
C  298:
C  299:             if (mod(dbgflg,2) .eq. 1)then
C  300:
C  301:           call gettim(time)
C  302:           ctemp      = cfj(time)
C  303:           cline      = task//' processing for '//decnam(dec)//' '
C  304:     &                  //ctemp(7:12)
```

```
C  305:           ctemp      = cfu(yyddd)
C  306:           cline(49:) = ctemp
C  307:           ctemp      = cfu(curday)
C  308:           cline(56:) = ctemp
C  309:           call sdest(cline,0)
C  310:
C  311:       endif
C  312:
C  313: c---          if the decoder is labeled as active
C  314:
C  315:       if (decsta(dec) .ge. 0)then
C  316:
C  317: c---          loop 300 scans through each of the defined
C  318: c---          indices used by each decoder
C  319:
C  320:       do 300 idx = 1 , numidx(dec)
C  321:
C  322:           call gettim(time)
C  323:
C  324: c---          if the task has just started, check to make
C  325: c---          certain that the index lw file exists
C  326: c---          (i.e. data of the type you are interested has
C  327: c---          been ingested).  if not go to the next index file
C  328:
C  329:           if (init(idx,dec) .eq. 0)then
C  330:
C  331:             lwf = lwfile(cidxfl(idx,dec))
C  332:             if (lwf .eq. 0)goto 300
C  333:             init(idx,dec) = 1
C  334:
C  335:           endif
C  336:
C  337: c---          statement #400 starts the main processing
C  338: c---          loop. this section gets the next 4 word
C  339: c---          index block, and if it is a new block,
C  340: c---          processes the data to completion
C  341:
C  342: 400       continue
C  343:
C  344:             next   = m0nxtidx(cidxfl(idx,dec),yyddd,time,
C  345:      &                        idxblk,cspool,ptr,flags(3,dec),' ',
C  346:      &                        begptr(idx,dec),lasptr(idx,dec),
C  347:      &                        calflg(idx,dec))
C  348:
C  349:             circit = clit(idxblk(1))
C  350:             nbytes = idxblk(2)
C  351:             nlines = nbytes / byprln
C  352:             tmstmp = idxblk(4)
C  353:             cwmo   = clit(idxblk(5))
C  354:             prodct = idxblk(6)
C  355:             corgin = clit(idxblk(7))
C  356:             afos   = ' '
C  357:             if (idxblk(8) .ne. MISS)afos   = clit(idxblk(8))
C  358:             afostn = ' '
C  359:             if (idxblk(9) .ne. MISS)afostn = clit(idxblk(9))
C  360:             afoorg = ' '
C  361:             if (idxblk(10) .ne. MISS)afoorg = clit(idxblk(10))
C  362:             faa    = 0
C  363:             if (idxblk(16) .ne. MISS)faa = idxblk(16)
C  364:
C  365: c---          if next .lt. 0 goto 300 for and look at the
C  366: c---          next index/decoder
C  367:
C  368:             if (mod(dbgflg/2,2) .eq. 1)then
C  369:
C  370:               write(cline,490)task,cidxfl(idx,dec),
C  371:      &              begptr(idx,dec),lasptr(idx,dec),
C  372:      &              cspool,nbytes,cwmo,prodct,corgin,next
C  373: 490              format(2(a12,1x),2(i6,1x),a12,1x,i6,1x,
C  374:      &                     a4,1x,i2,1x,a4,1x,i3)
C  375:
C  376:               call sdest(cline,0)
```

```
C  377:
C  378:            endif
C  379:
C  380:            if (next .lt. 0)goto 300
C  381:
C  382: c---          if a new data block was found , reset
C  383: c---          the processing counter to 0
C  384:
C  385:            stlnot = 0
C  386:
C  387: c---          now we will do a check on
C  388: c---          this data block to be sure we really want to
C  389: c---          load it.
C  390:
C  391: c---          if specific wmo headers are to be decoded
C  392: c---          check to make certain that this might be
C  393: c---          a correct block.
C  394:
C  395:            if (numwmo(dec) .gt. 0 .or. numorg(dec) .gt. 0)then
C  396:
C  397: c---          compare the wmo header, (ex. 'fous') and
C  398: c---          the product number with the list of
C  399: c---          acceptable values
C  400:
C  401:              ok = m0hedchk(cwmo , prodct , corgin ,
C  402:     &                      1 , 1 , 1 ,
C  403:     &                      1 , numwmo(dec) , wmo(1,dec) ,
C  404:     &                      minprd(1,dec) , maxprd(1,dec) ,
C  405:     &                      numorg(dec),orglst(1,dec))
C  406:
C  407: c---          if this is not a correct header go grab the
C  408: c---          next index block
C  409:
C  410:              if (ok .lt. 0)goto 400
C  411:
C  412:            endif
C  413:
C  414: c---          if the index block is ok, load the data block
C  415: c---          to cblk and do actual decoding
C  416:
C  417:            ok = mclddatb(cspool,ptr,nbytes,maxbyt,byprln,cblk)
C  418:
C  419: c---          if the data block was successfully loaded,
C  420: c---          decode the data
C  421:
C  422:            if (ok .gt. 0)then
C  423:
C  424:              if (mod(dbgflg/4,2) .eq. 1)then
C  425:
C  426:                do 250 ln = 1 , nlines
C  427:                  call sdest(task//' '//cblk(ln)(1:65),0)
C  428: 250              continue
C  429:                call sdest('-------------------------',0)
C  430:
C  431:              endif
C  432:
C  433:              if (decnam(dec) .eq. 'TRJDEC ')then
C  434:
C  435:                 decok = m0trjdec(cblk(1),cblk(2),nlines-1,
C  436:     &                           circit,yyddd,time,
C  437:     &                           flags(1,dec),cflags(1,dec))
C  438:
C  439:              endif
C  440:
C  441: c---          go back up and see if any more index blocks
C  442: c---          can be processed before moving onto the next
C  443: c---          index
C  444:
C  445:              goto 400
C  446:
C  447:            endif
C  448:
```

```
C  449: 300      continue
C  450:
C  451:        endif
C  452:
C  453: 200 continue
C  454:
C  455: 700 continue
C  456:
C  457: c---          statement #700 checks the following:
C  458: c---             1)  if the system day is differant from
C  459: c---                 the data-day update the data-day to
C  460: c---                 the system day and build the new index
C  461: c---                 file name.
C  462: c---             2)  increment the counter stlnot by 1.  if
C  463: c---                 stlnot is .gt. 1 that means that m0nxtidx
C  464: c---                 has been called twice in a row without
C  465: c---                 receiving any new data.  if this occurs,
C  466: c---                 set stlnot back to zero and go to sleep
C  467: c---             3)  else go back to the top and continue
C  468: c---                 processing
C  469:
C  470:     if (difday .and. stlnot .gt. 0)then
C  471:
C  472:       yyddd  = curday
C  473:       difday = .false.
C  474:
C  475: c---          reinitialize all of the index/decoder
C  476: c---          dependent information for the new day
C  477:
C  478:       do 710 dec = 1 , numdec
C  479:
C  480:          do 720 idx = 1 , numidx(dec)
C  481:
C  482:             init(idx,dec)    = 0
C  483:             calflg(idx,dec) = -1
C  484:             begptr(idx,dec) = 0
C  485:             lasptr(idx,dec) = -1
C  486:             chead = namidx(idx,dec)
C  487:             call m0idxnam(chead,yyddd,circit,cidxfl(idx,dec))
C  488:
C  489: 720      continue
C  490:
C  491: 710   continue
C  492:
C  493:     endif
C  494:
C  495:     stlnot = stlnot + 1
C  496:
C  497: c---          if stlnot is .gt. 1 that means that the
C  498: c---          cycle of loops 200 and 300 has been through
C  499: c---          2 complete times without finding anything
C  500: c---          new to decode so go to sleep
C  501:
C  502:     if (stlnot .gt. 1)then
C  503:
C  504:       stlnot = 0
C  505:       goto 1000
C  506:
C  507:     endif
C  508:
C  509:     goto 5
C  510:
C  511: c---          statement 1000 puts system to sleep for
C  512: c---          about 30 seconds
C  513:
C  514: 1000 continue
C  515:
C  516: c---          check to see if the system is shutting down
C  517:
C  518:     if (luc(194) .ne. 0)goto 2000
C  519:
C  520: c---          check to see if we should end the command
```

```
C  521: c---          instead of sleeping
C  522:
C  523:      if (mxstrt .gt. 0 .and. rstart .ge. mxstrt) goto 2000
C  524:
C  525:      rstart = rstart + 1
C  526:
C  527:      if (mod(dbgflg,2) .eq. 1)call sdest(task//' is sleeping',0)
C  528:
C  529:      call sleep(30000)
C  530:
C  531: c---          after waking up.  go back to the top and try
C  532: c---          processing again.
C  533:
C  534:      goto 5
C  535:2000 continue
C  536:
C  537:      call mcermess(1,cerror,task//' - Done')
C  538:      call edest('Done',0)
C  539:      return
C  540:      end
```

Moving to a Distributed System

Presented by
Dee Wade
McIDAS Operations Manager

Session 9
McIDAS Developer/Operator Training
October 23-25, 1995

Table of Contents

Overview. 9-1
SSEC's McIDAS system 9-1
Problems encountered. 9-3
Future goals. 9-5

Overview

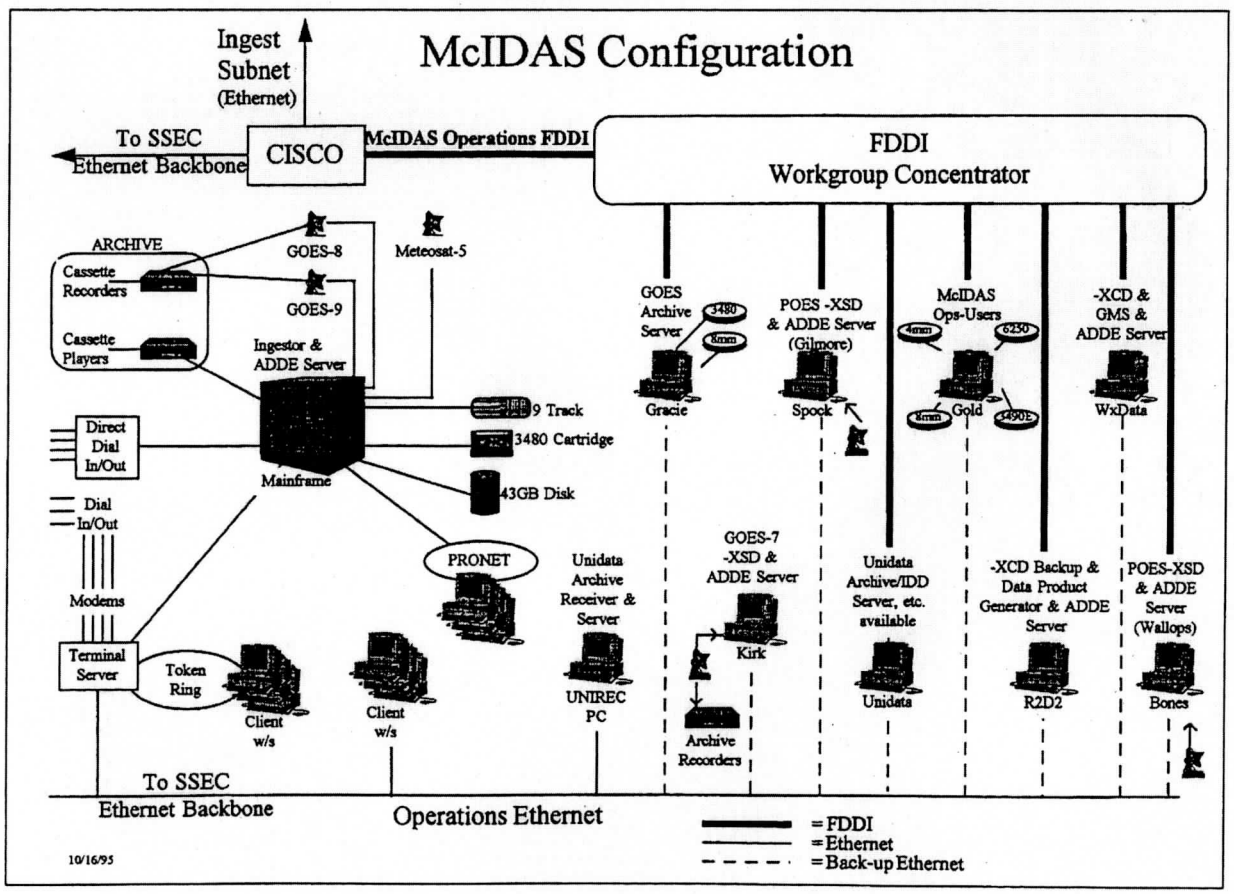
This session will update you on SSEC's status on moving to a distributed system. It will also explain some of the problems SSEC has encountered during our transition and the goals we have set for the near future.

SSEC's McIDAS system

The SSEC McIDAS system currently consists of the following:

- an IBM 4381-T92 mainframe that ingests and serves satellite data from GOES-8, GOES-9 and Meteosat-5, and receives conventional data from the Unix -XCD relay workstation
- several Unix workstations that handle a variety of tasks

The diagram below shows this configuration. The mainframe provides users with applications that have not been ported to Unix or OS/2.



The Unix McIDAS network consists of the following workstations:

Workstations	SSEC names
two IBM RISC/6000 3BTs	*spock* and *bones*
one HP 725/75	*kirk*
two SUN SPARCstation 10s	*wxdata* and *r2d2*
one IBM RISC/6000 58H	*gold*

The two 3BTs run McIDAS-XSD, ingesting and serving the POES relay data: GAC, LAC and HRPT.

The HP receives and serves GOES-7 data.

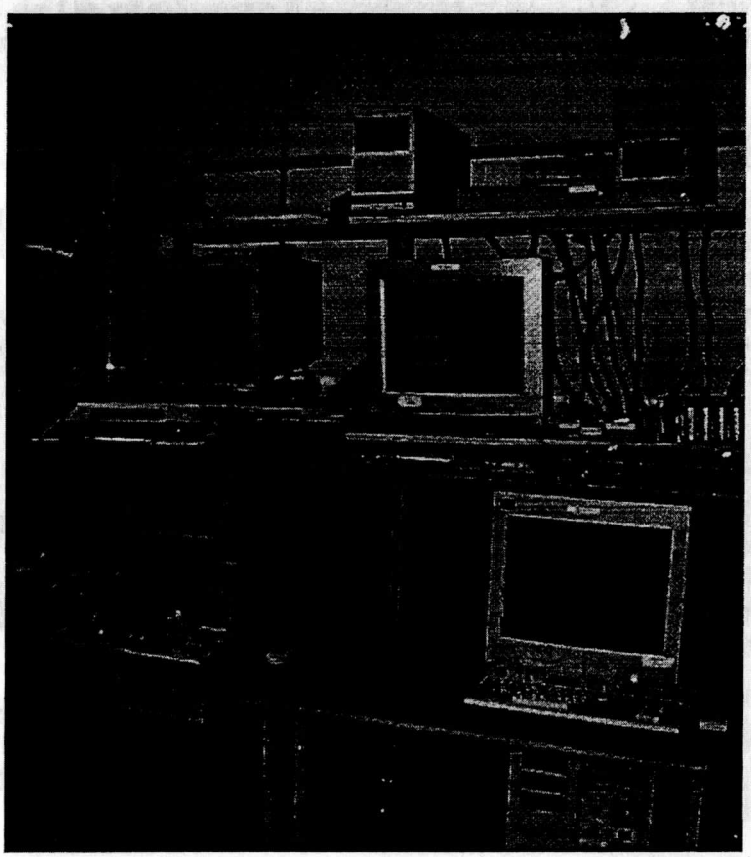
The SUN workstations run McIDAS-XCD. *wxdata* is the primary -XCD workstation; it also receives and serves GMS data from Australia. *r2d2* is a hot spare for -XCD, and is also used for product generation and serving.

The RISC/6000 58H serves in-house users who don't have the computing power on their desks. It is also the main source for tape output. *gold* has two 4mm DAT drives, an 8mm Exabyte drive, a 9trk drive, and a 3490E drive. All Unix tape processing is done on this workstation.

We quickly found that we didn't have enough disk space. All operation's Unix workstations now have 9 GB disk drives.

Limited floor space

As we began moving towards a distributed system, one of the first problems we encountered was where to put the workstations. Because limited floor space didn't allow us to add desks or tables, we had to build vertically. Complicating the search for appropriate furniture was the requirement that we handle multiple vendor hardware. Not only did the furniture have to be flexible, since the size and shape of the workstations and monitors varied from vendor to vendor, but the shelves had to support a great deal of weight. We chose Ergotron because of their pricing and flexibility. The picture below shows the Ergotron units.



Each unit holds six workstations: three on the bottom shelf and three on the middle shelf. The top shelf is used for peripherals. So far, this has worked well, but we don't have six workstations on a unit yet. This will limit the operator's workspace.

Operator training

Another problem we encountered was the training of our operations staff. The Unix environment is very different from the mainframe. The staff was not familiar with the Unix software or the new hardware. We will continue our training until the McIDAS operators are completely familiar with the new environment.

Increased staff

Operating a distributed Unix environment has forced an increase in the operations staff numbers. We added two people with strong Unix backgrounds to assist the McIDAS operators by writing scripts and programs, and performing some administrative duties. Unix system administrators are also needed to set up and maintain the workstations and network. SSEC's system administrator is responsible for all Unix systems in the Center, not just those in the McIDAS area. It quickly became apparent that McIDAS Operations needed its own system administrator.

Root access

In the McIDAS mainframe environment, the user *oper* often had power that other users did not. In a Unix environment, the user *root* has complete control over the system. This is usually reserved for the system administrator. We decided that the operator should not run as *root*. We use *root* only when necessary: when setting up accounts, for example. Each site must decide who has *root* access.

Shared files and peripherals

Multiple workstations must share common files and the use of peripherals. Operators should not be required to maintain a list of user initials and project numbers on each workstation, but every workstation needs access to this data. Each workstation should not require a separate tape drive for backups or saving data.

To solve these problems, we decided to NFS mount shared files, locate all tape drives on one workstation, and connect the workstations with an FDDI interface (100Mb/sec vs. 10Mb/sec of ethernet). We used the same approach with printing.

Have the option to turn off the mainframe

Our goal is to make all mainframe functions available on Unix by the end of 1996. We can then turn off our mainframe if we choose. This means we must have reliable -XSD GVAR and Meteosat ingestors. Both of these satellite types are currently under development. Since SSEC maintains the GOES archive, we must also be able to process the archive data via -XSD. Finally, we must resolve our problems with tape processing, which Unix does not readily address, and the porting of user commands.

Operate from a consolidated console

A distributed system means the operator must monitor multiple workstations. Operators need a consolidated console that provides a condensed status of processing on all workstations and sends warning messages when a workstation has trouble. The operator must also be able to get in-depth information about each workstation. We will be working on this console in 1996.

Produce the Distributed Operations Manual

We hope to have the first version of the *Distributed Operations Manual* available in the spring of 1996. Because we are still learning to run a distributed system, this will be a living document. If you have suggestions about what should be in this manual, please let me know.

I can be reached at (608) 263-0527 or via e-mail at *deew@ssec.wisc.edu*

McIDAS-XSD Operations

Presented by

Dana Davis *- McIDAS Operator*

Jerrold Robaidek *- Operations Programmer*

Session 10

McIDAS Developer/Operator Training

October 23-25, 1995

Table of Contents

Overview 10-1
Terminology 10-1

McIDAS-XSD ingest system 10-2
Satellite signals that McIDAS-XSD ingests 10-2
Satellite signal ingestors under development 10-2
McIDAS-XSD vs. -MVS ingestors 10-3

McIDAS-XSD data flow 10-5
Image data flow 10-5
Control processes and information flow 10-6

McIDAS-XSD setup 10-12
Initializing schedule files 10-12
Modifying your startup environment 10-13
Obtaining navigation 10-13
Setting up satellite schedules 10-14
Setting up event schedules 10-15

Starting and stopping McIDAS-XSD 10-16
Geostationary satellite ingest 10-16
Polar orbiter ingest 10-17
Stopping McIDAS-XSD 10-17

Miscellaneous utilities 10-18

Troubleshooting 10-21

Overview

This training session will provide basic information about McIDAS-XSD (McIDAS-**X** Satellite **D**ata). After this session, you should be able to:

- start and stop McIDAS-XSD
- recognize and use the control and display windows
- make modifications to your configuration file, xsd.cfg
- recognize aborted processes and restart them
- set up an event schedule

Terminology

The following terms are used throughout this section.

cron	Unix daemon that runs commands at specified times
crontab	Unix command for submitting entries to the cron daemon
daemon	a Unix process that operates continuously and unattended to perform a service
DDS	Domestic Data Service
full resolution buffer	a portion of shared memory where the ingd process stores full resolution data to make it available for other processes
PDL	Processor Data Load
rcp	remote copy protocol
SAS	Satellite-data Acquisition System
sector buffer	the portion of shared memory where the ingd process gets data
shared memory	a block of memory accessible to more than one process
$XSD	Unix environment variable set as the home directory path of McIDAS-XSD

McIDAS-XSD ingest system

The McIDAS-XSD ingest system allows Unix-based workstations to directly receive and process satellite data. With McIDAS-XSD, you can schedule the creation of real-time areas, schedule post processing of those areas, control the Satellite-data Acquisition System (SAS), and monitor system operation.

Satellite signals that McIDAS-XSD ingests

McIDAS-XSD ingests the five types of satellite data below. SSEC currently ingests only GOES-7 and POES-Relay data with McIDAS-XSD.

- GOES AAA
- POES-Relay
- POES-Flyover
- GMS
- DMSP

Satellite signal ingestors under development

- METEOSAT
- GVAR

Besides being designed for Unix-based workstations, McIDAS-XSD has several other differences from the McIDAS-MVS ingestor.

Ingest process

On the -MVS ingestor, the ingestor card does most of the work, including averaging, sampling and resolution reduction. McIDAS-XSD has several software processes to accomplish these tasks.

Operational commands

Many of the McIDAS-MVS operational commands for generating satellite schedules are also available in McIDAS-XSD. However there are important differences. The satellite scheduler commands in McIDAS-MVS (SSK*x* and GEO*x*) are entered at the McIDAS command line. The McIDAS-XSD scheduler commands (pol*x* and geo*x*) are entered at the Unix prompt.

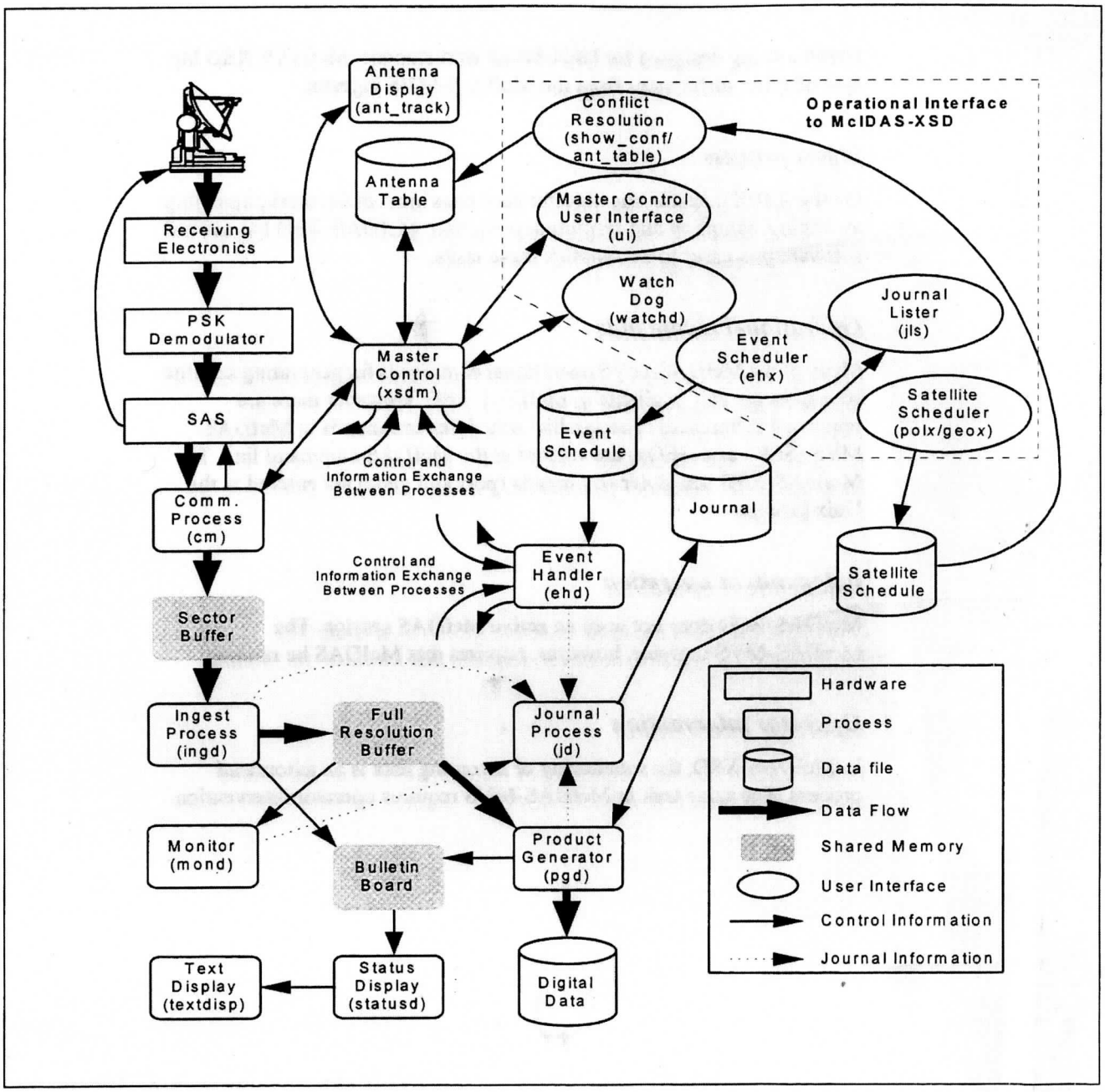
Independent operation

McIDAS-XSD does not need an active McIDAS session. The McIDAS-MVS ingestor, however, requires that McIDAS be running.

Operator intervention

In McIDAS-XSD, the monitoring of incoming data is an automated process. The same task in McIDAS-MVS requires operator intervention.

McIDAS-XSD Ingestor Flow Chart



McIDAS-XSD data flow

McIDAS-XSD uses several processes when ingesting satellite data. The data used, and the information exchanged by these processes follow many paths. This section examines these processes and the flow of data and information through McIDAS-XSD, as shown in the flow chart on the adjacent page.

Image data flow

Image data from the satellite is received by the antenna and passed to the receiving electronics and PSK demodulator. From the demodulator, it is passed to the SSEC ingestor card, or SAS (Satellite-data Acquisition System).

The SAS is a stand-alone unit that collects satellite data and provides it to another system upon request via an ethernet connection. The SSEC ingest card, along with the software device driver, provides that same data to a RS6000-based ingest system. Because the board is inserted into a Micro-channel slot on the RS6000, an external communication link is not required.

The communication process (**cm**) receives the data from the SAS or SSEC ingestor card and informs **xsdm**. **xsdm** then directs **cm** where in the sector buffer section of shared memory to put the data. The ingest process (**ingd**) takes the data from the sector buffer and creates a full resolution buffer section of shared memory. The full resolution buffer contains full resolution image data in a format that enables the product generator to use the data. The product generator (**pgd**) gets the data from the full resolution buffer and creates products (digital areas) on disk according to the requests in the satellite scheduler.

xsdm

The **xsdm** process is the master control for the entire McIDAS-XSD system. This process is started at the Unix prompt by typing **xsdm**. The following functions are performed by **xsdm**:

- reserves the use of the SAS
- manages all program-to-program communication
- allocates ingestor resources
- sends antenna positioning information to the SAS for polar HRPT flyover ingests
- starts all other McIDAS-XSD processes

cm

The communications process, **cm**, is the first process started by **xsdm.** It establishes communication pathways that allow **xsdm** to pass commands and to receive the satellite signals from the SAS. When the SAS begins sending data, a second **cm** process is started. This **cm** process receives the satellite signal and writes the data to the sector buffer.

Information is also exchanged by **xsdm** and **cm** to control the rest of the McIDAS-XSD system. The SAS receives antenna table information and control information from **cm**, and returns the antenna position and SAS status information. The SAS controls the antenna movements by sending antenna position information to the antenna.

ehd

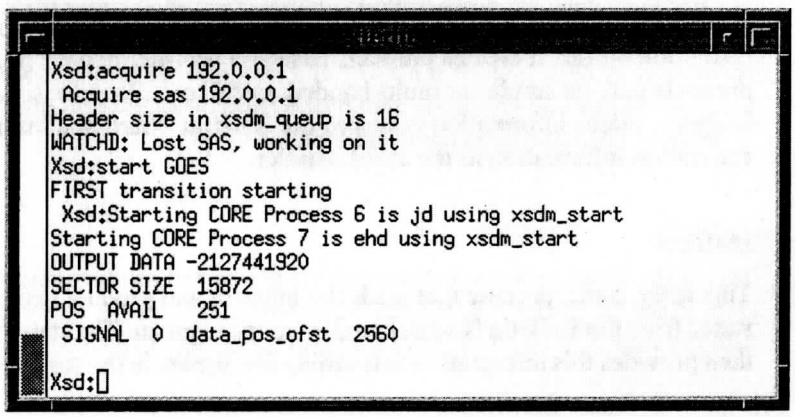
The event handler process is **ehd**. It receives messages from other processes and broadcasts the messages throughout the McIDAS-XSD system. The event handler also evaluates user-defined expression and command pair entries from the event scheduler. If an expression evaluates as true, **ehd** runs the command. The user can create these expression and command pairs with the **ehe** command.

The ingest process sends messages to the event handler for these events:

Event	Message contents
image begin and end time	image date and time, sensor source number and type, scan number, scan date and time, start block, satellite mode
navigation codicil filed	scan count, slot where the data directory is filed, scan date and time, image date and time, sensor source number and type
calibration codicil filed	scan count, scan date and time, image date and time, slot of the data directory, bandmap, sensor source number and type
PDL	satellite mode, scan date and time, image date and time, pdlscans, pdllats, pdlbands, pdlspins (if DWELL), and pdlnum

ui

The process, **ui**, is the user interface process for **xsdm**. This process sends user-entered commands back to **xsdm.** All processes in McIDAS-XSD use the **ui** window for error and other text output.



The user interface window is an xterm window with an Xsd: prompt, as shown above. From this prompt, all processes in McIDAS-XSD can be started or stopped. This is also where other **ui** commands are entered, such as **acquire** *SAS-address* or **start** *satellite.*

watchd

The **watchd** process is known as the watch dog process. Information regarding the status of all other McIDAS-XSD processes (except **xsdm**) is exchanged between **watchd** and **xsdm**. If a process fails, **watchd** notifies the operator by displaying a yellow popup window on the screen.

Once data is detected, the following processes are started:

- **ingd**
- **pgd**
- **statusd**
- **textdisp**
- **mond**
- **ant_track**
- **jd**

ingd

The ingestor process, known as **ingd**, reads the raw data from the sector buffer, and generates full resolution data. This data is later used by other processes in McIDAS-XSD. Status information is sent by **ingd** to the Bulletin Board shared memory segment.

pgd

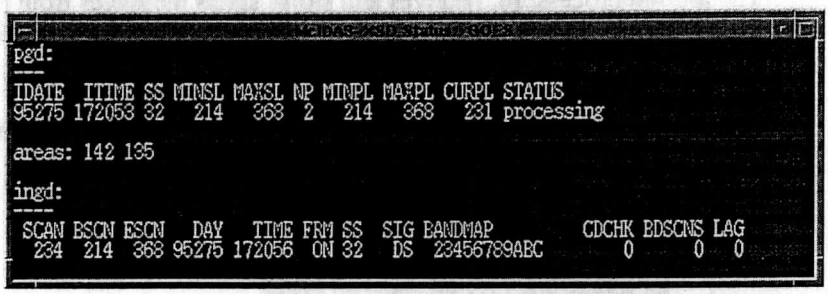
The product generator process, **pgd**, generates McIDAS areas from the full resolution buffer. It creates products based on user-defined entries. Area products may be single- or multi-banded, or full or reduced resolution imagery. Status information is sent to the Bulletin Board and product generation information to the event handler.

statusd

This is the status process that reads the ingestor and product generator status from the Bulletin Board shared memory segment. The status process then provides this information to **textdisp** for display in the status window.

textdisp

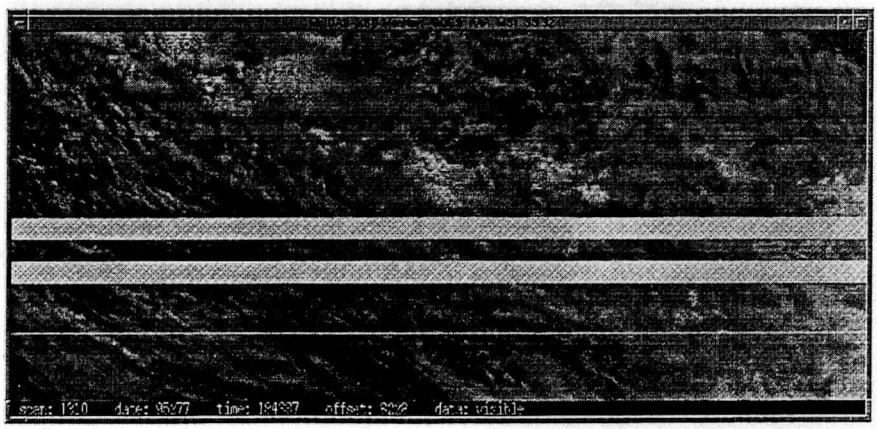
This process creates and displays status messages. The McIDAS-XSD Status window, shown below, is generated by the textdisp process. It lists information about the product generator (**pgd**) and ingest process (**ingd**). Status information from **ingd** and **pgd** are obtained by **statusd** from the Bulletin Board shared memory segment and passed to **textdisp**.



mond

The monitor process, **mond**, creates a window and displays data from the full resolution shared memory buffer. Other image information can also be found in this window such as date, time and element offset.

The McIDAS-XSD Monitor window, shown below, displays a portion of the current image being ingested. If the ingest process flags a bad line of data, a color line (shown as a gray line below) is displayed on the window. The text line at the bottom of the window displays the current scan line, date, time, and the type of data being displayed.



Note: For geostationary satellites, clicking the left mouse button on the McIDAS-XSD monitor window toggles the display between visible and infrared.

ant_track

This process displays the current position of an antenna that is tracking polar orbiting satellites.

jd

The journal process, **jd**, receives and stores status messages from other McIDAS-XSD processes. It writes all journal messages that it receives to the journal files, which can be listed with the **jls** command.

The ingestor process sends journal messages when the following events occur:

- an image begins for all ingestor types
- an image ends for all ingestor types
- navigation is filed; GMS and geostationary satellites only
- calibration is filed
- a beta is filed; geostationary satellites only
- a piece of common doc is bad for any ingestor type
- a bad scan line is detected for any ingestor type

The event handler, **ehd**, sends a journal message when the following events occur:

- ehd starts or shuts down
- ehd receives a messages
- ehd updates its list of entries
- ehd receives a message in an incorrect syntax

The product generator process, **pgd**, sends a journal message each time the following events occur:

- the pgd process starts or stops
- the product ends earlier then scheduled
- the percentage of good data in the product is less than the minimum required to retain the product
- the beginning and end of each product

The monitor process, **mond**, sends a journal message each time it starts a new image.

The journal process writes those messages to journal files in the $OPROOT directory. Within that directory, journal files are broken into subdirectories depending on the process sending the message and the type of ingestor that is running. For example, the disk file $OPROOT/GOES/db/ingd contains journal messages from the ingd process of the GOES ingestor.

McIDAS-XSD setup

The McIDAS-XSD setup consists of the following procedures, each of which is described below.

- initializing the schedule files
- modifying the startup environment
- obtaining navigation
- setting up the satellite schedules and event schedules

Initializing schedule files

Before McIDAS-XSD can ingest data, the software has to be told what type of data it will ingest, and when to ingest it. This information is kept in the satellite scheduler files, which must be initialized with the **files** command before they can be edited or added to using the **geoe** and **pole** commands. The **files** command initializes the satellite scheduler data files by deleting their contents. Run this command once at installation to initialize these files. ***Use this command with caution!*** It deletes all entries and windows. It is usually run only at installation, when major ingestor/scheduler software changes occur, or if the satellite scheduler files are accidentally deleted or become corrupt.

Another command that must be run at installation is **popdbs**. This command creates a binary version of the satellite scheduler database file (SATDBS). You must run **popdbs** before requests for satellite data can be made with the **geoe** and **pole** commands. You must also run it after any changes are made to the scheduler database file. These changes are usually provided in software upgrades.

sites.dat

The scheduler command **polx** will not work with earth coordinates until your antenna location is defined in the sites.dat file in the $XSD/lib/pfdata directory. Below is an example of a sites.dat file:

```
SSEC     43.07      89.41
sentry   -1000.0    -1000.0
```

The first line contains the name of the site and the location of the site's antenna. The second line denotes the end of the file.

The attributes of the processes started by **xsdm** are determined by the McIDAS-XSD system defaults. To change these attributes, you must create or modify the text file xsd.cfg, which is located in the $XSD directory. The xsd.cfg contains a set of user defined arguments that **xsdm** uses when it starts the other McIDAS-XSD processes. An example of an xsd.cfg file is shown below:

```
#
#MOND_POES="-s $DATA -k $OUT -w 640 -h 500 -I 30 -b 2 -f"
MOND_POES="-s $DATA -k $OUT -w 640 -h 350 -I 160"
MOND_GOES="-s $DATA -k $OUT -w 1000 -h 350 -I 25"
```

This file overrides the default for **mond** and starts **mond** with a window size as defined by the **w** and **h** flags. The symbol # denotes comment lines and is ignored when the file is read.

Obtaining navigation

In addition to the commands described above, you must obtain navigation for the satellites.

POES

POES navigation is sent via DDS under the header TBUS. The TIRDEC decoder in the miscellaneous data monitor **dmmisc** (provided in McIDAS-XCD), decodes POES navigation and files it into the file SYSNAV1 on a McIDAS-XCD machine. SYSNAV1 is then transferred via rcp to the McIDAS-XSD machine and put into the directory $XSD/lib/mcdata.

DMSP

DMSP navigation is sent as part of the DMSP data stream. The DMSP ingestor decodes and files the navigation into the proper system navigation files.

GMS

GMS navigation is sent as part of the GMS data stream. The ingestor puts navigation into the GMS DOC areas. The event handler then runs an entry containing the McIDAS-X utility GMSGRD, which reads the navigation from the GMS DOC areas and files landmarks into the appropriate system navigation file. The McIDAS-X utility, NVUP, which is run from the crontab, reads those landmarks and updates the navigation.

GOES

GOES navigation is sent as part of the GOES data stream. The GOES ingestor decodes and files the navigation into the proper system navigation files.

PRED

PRED navigation predictions are done for all types of data to ensure that the current days navigation always exists in $XSD/lib/mcdata/SYSNAV1.

Setting up satellite schedules

The satellite scheduler commands determine ingest windows and products. The command syntax, keywords, flags, and results are very similar to those for the McIDAS-MVS commands SSK*x* and GEO*x*. The scheduler commands for polar and geostationary satellites are defined below.

Commands	**Description**
pole	creates satellite scheduler windows, entries and named sectors for polar orbiting satellites.
polu	modifies satellite scheduler windows, entries and named sectors for polar orbiting satellites
poll	lists satellite scheduler windows for polar orbiting satellites
geoe	creates satellite scheduler windows, entries and named sectors for geostationary satellites.
geou	modifies satellite scheduler windows, entries and named sectors for geostationary satellites
geol	lists satellite scheduler windows for geostationary satellites

ehe

Use **ehe** to create, edit, activate, or deactivate an event handler entry. Each entry contains a Boolean expression and a Unix command. As the event handler receives messages from other McIDAS-XSD processes, it checks the message fields against entry expressions. If the expression is true, the event handler runs the specified command. Utilities used by **ehe** can be found in *Miscellaneous utilities* later in this section. The format is as follows:

```
$prog{field} operator matchstring
```

For example: `$ingd {event} eq "image end"`

Use command **ehls** to list the event scheduler; use command **ehrm** to delete entries from the event scheduler.

show_conf and ant_table

show_conf and **ant_table** resolve antenna conflicts for polar orbiting satellites and send the antenna table to the **xsdm**, which in turn uses that table to control the antenna movements. **ant_table** and **show_conf** read all the polar orbiter windows in the satellite schedule and determine which subentries apply to the time interval given, and if any of those subentries conflict with each other. If no conflicts exist, the antenna table is sent to **xsdm**. If conflicts do exist, use **show_conf** to resolve the conflicts via a series of graphical user interfaces (GUIs). If you use **ant_table** with the **-d** flag, it will resolve any conflicts by choosing the longest of the conflicting orbits; **ant_table** will not send the antenna table to **xsdm** if conflicts exist.

Starting and stopping McIDAS-XSD

This section describes how to start and stop geostationary satellite ingests and polar orbiter ingests.

Geostationary satellite ingest

When all the necessary files are initialized and the satellite scheduler files are updated, you can begin to ingest data. Use the steps below to start McIDAS-XSD.

1. From an xterm window, logon to the account where McIDAS-XSD is installed.

2. From the Unix prompt, start the master control process.

   Type: **xsdm**

   The xsdm process starts **cm**, **watchd** and **ui**. The Unix prompt in the xterm window is replaced with the xsdm-User Interface prompt, Xsd:.

3. From the Xsd: prompt, reserve the Satellite-data Acquisition System (SAS).

   Type: **acquire** *address*

   where *address* is the IP address of the SAS to be used. Some sites allow only numeric IP addresses. Contact your Unix system administrator for more information.

4. From the Xsd: prompt, specify the type of satellite data to ingest.

   Type: **start** *satellite*

   where *satellite* is the type of satellite data to ingest, GMS or GOES, for example. Enter the satellite name in uppercase. This entry tells **xsdm** to allow data flow from the SAS when data is available. Currently, McIDAS-XSD can process only one type of geostationary satellite data at a time.

When **xsdm** detects data, it starts **ingd**, **jd**, **pgd**, **statusd**, and **mond** using the satellite type's default flags or user defined flags from the configuration file xsd.cfg; see the *Modifying your startup environment* section above.

To enter McIDAS-XSD satellite schedule or event handler entries, or Unix crontab entries, use another xterm window logged on to the account where McIDAS-XSD is installed.

Polar orbiter ingest

Starting a polar orbiter ingest is similar to the geostationary ingest. However, before you start McIDAS-XSD and before any scheduling is done, verify that current navigation parameters exist; otherwise, the antenna tracking will not function properly. See the *McIDAS-XSD Users Guide* (9/95) for a complete description.

Stopping McIDAS-XSD

To stop a geostationary or polar orbiter ingest system, you must stop the master control process. From the Xsd: prompt,

Type: **quit xsdm**

This command stops all McIDAS-XSD processes and releases the SAS.

Miscellaneous utilities

This section describes some of the miscellaneous Unix and McIDAS utilities.

jls

jls lists journal records according to specified sort parameters. The search keys available for the **ehd**, **ingd**, and **pgd** processes can be found in the *McIDAS-XSD Users Guide*. **jls** is a Unix utility and must be issued from a Unix prompt.

mccommand

mccommand creates an environment so that McIDAS commands can run without an active McIDAS-X session. **mccommand** requires a connection to an X Window server on which you have privileges. Use the Unix **export** command to set your display environment variable, so the server's output is sent to your display. For example, **export DISPLAY**=*unix*:0

mccommand is ideal for scheduling McIDAS commands to run via the event handler or UNIX crontab entries. When scheduling commands, you may want redirect text output to a file, which can serve as a log of errors and successful commands run by **mccommand**. The command format is:

mccommand *COMMAND*

where *COMMAND* is the complete McIDAS-X command, in uppercase.

To enter multiple McIDAS commands on one command line, separate the commands with a backslash (\) and a semicolon (;). Use this format:

mccommand *FIRST CMD*\;*SECOND CMD*

There are many McIDAS-X utilities available for post processing of ingested areas. On McIDAS-MVS, these utilities were run from McIDAS with the event scheduler. McIDAS-XSD runs these commands via **mccommand**. These commands are set up with the event scheduler command **ehe**. The information below describes the McIDAS-XSD utilities likely to be run.

DMSPCAL

You must run the DMSPCAL command on an SSMI area before you can display the area. DMSPCAL does not change the data values, but it does record calibration information to the CAL portion of the area prefix. You can enter DMSPCAL manually, but the event handler is normally used to automate this process. Use the command format below for scheduling DMSPCAL. Enter the entire command on one line, as shown.

```
ehe -e '$pgd{event} eq "product end" && $pgd{sig} eq "ssmi"' -c 'mccommand DMSPCAL pgd{area}'
```

GMSGRD and GMSNVUP

GMSGRD and GMSNVUP update the navigation for GMS images. GMSGRD files grid locations (landmarks) from the GMS DOC areas into the specified navigation file. GMSNVUP is a macro that uses NVUP to upgrade the current navigation with the landmarks created by GMSGRD. GMSNVUP also uses PRED to predict navigation for the next day.

IMGFLIP

IMGFLIP flips polar orbiter images from left to right or top to bottom, so the northwest corner of the image is in the upper-left. You can use this command with all DMSP sensors and the POES AVHRR sensor. You cannot flip POES HIRS and MSU images.

ORBPLOT

ORBPLOT displays the orbital tracks of polar orbiting satellites. It gets the navigation information defining the orbital tracks from two different sources. If you use the AREA keyword, the navigation stored with the specified area generates the tracks. Otherwise, the navigation parameters are taken from a system navigation file, specified with the NAVF keyword.

By default, ORBPLOT plots orbital tracks over the displayed image or map. If the frame contains no navigation, a Mercator map of the world is displayed before plotting the orbital tracks. Use the MAP keyword to specify a different map, or use MAP=DEF and the LAT and LON keywords to define a custom map.

PRED

PRED files predicted navigation based on a master navigation entry. It is usually run from the crontab once per day.

TIRCAL

TIRCAL calibrates POES AVHRR images. It is scheduled to run at the end of each image via the event scheduler. Use the command format below for scheduling TIRCAL:

```
ehe -e '$pgd{event} eq "product end" && $pgd{bandmap} ne "<6>" && $pgd{satmode} eq
"POES"' -c 'mccommand TIRCAL $pgd{area}'
```

TIPTI

TIPTI creates HIRS and MSU products from the raw data in the POES TIP areas.

Troubleshooting

If you encounter any of the problems below, use the suggested solutions to fix them. Although restarting McIDAS-XSD may fix a problem, do a restart only after investigating all other avenues.

pgd has stopped

Problem:	The watchd process (watchdog) reports that **pgd** has stopped. After restarting **pgd,** it stops again.
Possible solution:	The file system that **pgd** writes to could be full. Check the file systems from the Unix prompt with **df**. Delete any unneeded files, such as old areas or debug files.

Navigation is lost

Problem:	Loss of navigation
Possible solutions:	McIDAS-XSD may have been down for more than 24 hours and PRED was not run. You may need to run PRED. Or, TBUS may be old. Check the TBUS time on a McIDAS-XCD workstation. If TBUS is more than two days old, contact your TBUS supplier.

Ingestion or processes are hung

Problem:	McIDAS-XSD is not ingesting or processes seem to be hung. After trying a restart, McIDAS-XSD still does not come up.
Possible solution:	Duplicate processes may have been started. Multiple processes are usually a result of **xsdm** being stopped by means other than **quit xsdm** at the Xsd: prompt. Stop/delete the extra processes from the Unix prompt. Restart McIDAS-XSD.

McIDAS-XCD Operations

Presented by

Chad Johnson

Operations Programmer

Session 11

McIDAS Developer/Operator Training

October 23-25, 1995

Table of Contents

Overview 11-1
Terminology 11-1
McIDAS-XCD ingestors and data monitors 11-2

Bouncing ingestors and data monitors 11-3

Updating station changes 11-4
Station dictionaries 11-5
Site-specific ID tables 11-5

Setting up real-time data locations 11-6

Customizing the GRIB decoder 11-9

Status display 11-12

The DATARECV program 11-14

Allocating disk space 11-15

Deleting text, MD and grid files 11-16

Archiving data 11-17

Recovering archived data 11-19

Troubleshooting 11-20

Overview

This training session will provide McIDAS operators with the information they need to set up and maintain McIDAS-XCD. It describes the conventional data ingestors and monitors, and the status display. It also explains how to perform the following procedures:

- bounce ingestors and data monitors
- update station tables
- set up real-time data locations
- customize the GRIB decoder
- allocate disk space
- delete data files
- archive data
- troubleshoot

Terminology

The terms below are used throughout this section.

circuit file	file containing text data blocks received from a text ingestor; the file has the extension .XCD
data block	text data containing a WMO header
data monitor	a McIDAS program that periodically checks newly ingested data to determine if a specific decoder should be called
DDS	Domestic Data Service
decoder	software that parses data from one format into a common format for use by another process such as a plotter or lister
GRIB	GRIdded Binary message format accepted by the World Meteorological Organization for the distribution of gridded data

HRS	High Resolution Service
IDS	International Data Service
index file	a file, written by ingestors, that contains pointers to data blocks in circuit files; index files have the extension .IDX
ingestor	process that listens to data received by a communications port and reformats the information for further processing
NGM	National Meteorological Center Nested Grid Model
NWS	National Weather Service
PPS	Public Products Service
STARTXCD	mother task of the entire McIDAS-XCD ingestor/decoder package
status display	X Windows application that displays the current state of the McIDAS-XCD ingestor/decoder system

McIDAS-XCD ingestors and data monitors

The McIDAS-XCD ingestors read data from a communications port and reformat that data for later processing. It has two types of ingestors:

- text
- binary

The text ingestor reads ASCII data from the communications port and files that data in a circuit-specific file. For each block of text data written, additional information is written to an index file specific to the WMO category to which that block belongs. The binary ingestor reads data from a circuit and files that data into a circular spool file for later processing.

The -XCD data monitors traverse through the index files or binary spool files as data is filed by the ingestors. When new data arrives that a decoder is interested in, that block of data is passed to the decoder for processing. The decoder then parses the data and converts it into McIDAS file format.

The McIDAS-XCD ingestors and data monitors are started by the XCD master process, STARTXCD. Never start the ingestors or data monitors from the McIDAS command line.

Bouncing ingestors and data monitors

You will rarely need to bounce a McIDAS-XCD ingestor or data monitor. Ingestors run continuously. The only time you may need to bounce them is for communications port changes, such as baud rate or port connection changes.

Data monitors are designed to periodically restart with no intervention so that any changes made since the last restart will become effective. For example, if you make a station change to MASTERID.DAT, such as adding a RAOB station with the IDU command, it becomes effective the next time the decoder restarts.

You must bounce a data monitor manually if either of these events occur:

- a decoder within a data monitor is activated or inactivated; for example, if you activate the TIROS Navigation decoder and want the change to take effect immediately
- a configuration file is modified and must take effect immediately; for example, if you activate the station ID monitoring system or move real-time MD files

STARTXCD continuously monitors the state of -XCD. To bounce an -XCD ingestor or data monitor, simply stop the version of the process currently running just as you would stop any McIDAS command. STARTXCD samples the system every 30 seconds and will restart any process it finds no longer running.

Updating station changes

With the modernization currently taking place in the weather service, stations are constantly being moved, commissioned, decommissioned, and augmented.

When a station's status is changed, the NWS typically sends a message under WMO header NOUS41 containing relevant information about the station change. Because this method is not 100% reliable, you should also check the NWS gopher server, which was recently established to help disseminate modernization information. The address is: gopher.cominfo.nws.noaa.gov (140.90.5.206).

McIDAS-XCD can monitor some incoming stations for you, since most McIDAS-XCD decoders contain software to do this. If you keep statistics on stations that you are expecting to receive and those that are currently not in your database, you can inactivate stations that are no longer reporting.

To activate station monitoring, you must set the IDMONFLAG= value in the configuration file for that decoder. The configuration files reside in ~oper/mcidas/data. The IDMONFLAG value is a number between 0 and 3, as described below.

Value	Description
IDMONFLAG=0	no station monitoring; this is the default when McIDAS-XCD is installed
IDMONFLAG=1	monitors new stations
IDMONFLAG=2	monitors old stations
IDMONFLAG=3	monitors both old and new stations

Use the McIDAS-XCD command, IDMON, to display the information collected with station monitoring.

Station dictionaries

You can find station ID information in these two files: IDMSL and MASTERID.DAT

IDMSL is the Master Station Catalog, which is maintained by Tinker Air Force Base, and is distributed with McIDAS-X. IDMSL is used by McIDAS commands that convert station IDs to lat/lon groups, such as MSL, PC L, and DF.

MASTERID.DAT is used by the McIDAS-XCD decoders. It contains both geographical information about stations and the types of observations each station reports. MASTERID.DAT resides in ~mcidas/data and is replaced with each McIDAS-XCD upgrade. The site administrator uses the McIDAS-XCD utility, IDU ADD/EDIT, to keep MASTERID.DAT up-to-date. IDU is specifically designed to manipulate the individual entries in MASTERID.DAT.

MASTERID.DAT and IDMSL are updated monthly and are available to you via the MUG BBS (Bulletin Board System). If you don't want to replace your version of the file MASTERID.DAT, a list of changes made each month and cumulative since the last McIDAS-XCD upgrade is provided. Instructions for upgrading your MASTERID.DAT and IDMSL are also available on the BBS.

If you know of stations whose status is incorrect in MASTERID.DAT, contact the McIDAS Help Desk and we will integrate those changes in the next release if they are beneficial to the entire McIDAS community. Until the stations are updated in the core system, you can add them yourself. Keep a copy of the IDU commands that you run locally in a batch file. Then if you need to reload the core station list, you can run a batch command to reimplement the station changes.

Site-specific ID tables

If you have numerous, specific station changes at your site, you should maintain a batch file of IDU commands with these changes. When you upgrade to a new version of MASTERID.DAT, you can simply run this batch file to add your site-specific stations.

You can also maintain this station table separately from the core MASTERID.DAT in LOCALID.DAT. To do this, write to an alternate file with the FILE= keyword of the McIDAS-XCD IDU command. Then modify the MASTERFILE= keyword in the configuration file for your local decoder to use this station ID table.

Setting up real-time data locations

When you install McIDAS-XCD, real-time data is filed in these locations:

Data type	MD/Grid files	Files	Config. file
SAO/METAR	1 - 10	SAOMETAR.RAP	ISFCDEC.CFG
RAOB	11 - 30	RAOB.RAP	IRABDEC.CFG
Ship/Buoy	31 - 40		ISHPDEC.CFG
FOUS14	41 - 50		FO14DEC.CFG
SYNOPTIC	51 - 60	SYNOPTIC.RAP	SYNDEC.CFG
PIREP/AIREP	61 - 70		PIRDEC.CFG
FTs/TAFs		TERMFCST.RAP	TERDEC.CFG
Watch Box		WXWATCH.DAT	WBXDEC.CFG
TIROS Nav		SYSNAV1	TIRDEC.CFG
Real-time Grids	5001 - 5300		RTMODELS.CFG

MD filing

Do not alter the MD file locations at this time. Most of the applications that display real-time conventional data call a subroutine that has not yet been modified to allow more flexibility.

Grid filing

The miscellaneous grid file group (5001-5010) stores output from models that send minimal data; for example, the Wind Wave Forecast Model (WWFM). The grid files are divided by model: ETA, NGM, MRF, MAPS. The file RTMODELS.CFG, located in ~mcidas/data, determines the schemes used for filing gridded data. The three basic filing schemes are described below.

- All grids from a model are filed in one grid file regardless of model run and valid forecast times. Although this scheme is the simplest to understand, the search time for specific grids may be excessive if many products are generated by the model.

- Everything from the model is stored in one grid file per model run time per day. For example, all the NGM data from the 12 Z model run is stored in grid files 6001-6010, based on Julian day.

- Grids are filed based on model, model run time and model forecast validation time. For example, all the MRF data from the 12 Z model run with forecast times of 00hr through 24hr are stored in grid files 5071-5080, based on Julian day. This scheme is the most difficult for a user trying to find a specific grid file, but search times for individual grids are shorter.

Software that uses the appropriate API for locating real-time grid files will make whichever method you choose irrelevant to the user.

A complete description of RTMODELS.CFG can be found in Appendix D of the *McIDAS-XCD Installation and Users Guide*.

Example

Below is an example of an RTMODELS.CFG file and a description of how the various models are filed.

```
# Positional parameter descriptions
#
#   position     description
#      1         grid filing format
#                   0  - everything from the model is stored in one
#                        grid file per model run time
#                   1  - grids are filed based on model run time and
#                        valid forecast time
#                   2  - all the grids from a model run are filed in
#                        the same grid file regardless of run time
#                        or forecast time
#                   3  - same as 1 except no grids are assumed beyond
#                        the max forecast time (parameter 5)
#      2          first grid file in the range for this model
#      3          interval between model runs (hhmmss)
#      4          which forecast period interval to use to separate
#                 forecast grids
#      5          maximum forecast time, after which all grids are
#                 stored in the same grid file

SCRATCH=5001
MAPS= 0 1001 30000
MRF=  1 1101 120000 240000 480000
NGM=  2 1201
```

In the above example, any model not specified in RTMODELS.CFG is filed in the range specified by the SCRATCH= keyword. For example, ETA grids are filed in a grid file between 5001 and 5010 based on the Julian day of the model run time.

MAPS grids are filed only by model run time beginning with grid file 1001. See the table below.

Grid file	Run time	Forecast range
1001-1010	00 Z	all forecast times
1011-1020	03 Z	all forecast times
1021-1030	06 Z	all forecast times
1031-1040	09 Z	all forecast times
1041-1050	12 Z	all forecast times
1051-1060	15 Z	all forecast times
1061-1070	18 Z	all forecast times
1071-1080	21 Z	all forecast times

MRF grids are filed based on model run time and model verification time. In this example, the forecast period interval separating the forecast grids is 24 hours, and the maximum forecast time is 48 hours. This means that forecast grids in 24-hour intervals are filed in one grid file. All grids with a validation time greater than 48 hours are filed in another grid file. The interval between model run times determines how many groups of these model verification time grid files are created. In this example, the interval between model run times is 12. This means that two of these groups will be created. See the table below.

Grid file	Run time	Forecast range
1101-1010	00 Z	00hr <= Forecast Time <= 24hr
1111-1120	00 Z	24hr < Forecast Time <= 48hr
1121-1130	00 Z	>48hr Forecast Time
1131-1140	12 Z	00hr <= Forecast Time <= 24hr
1141-1150	12 Z	24hr < Forecast Time <= 48hr
1151-1160	12 Z	>48hr Forecast Time

In the RTMODELS.CFG example above, all NGM grids are filed in one range of grid files beginning with grid file 1201. Although this scheme is the easiest to understand, it results in large search times and is very difficult to locate grids of interest.

Customizing the GRIB decoder

McIDAS-XCD version 1.1 contains the first release of the GRIB decoder. You can configure this decoder to discard grids that are not of interest to you. This is advantageous because certain models are sent in multiple projections and there is no need to store this duplicate information. It can also limit the number of grids filed if disk space is a concern.

You can configure the GRIB decoder to discard individual grids based on the following:

- model name
- model run time
- forecast time
- projection
- level
- parameter

This information is stored in the file NOGRIB.CFG in the directory ~oper/mcidas/data.

Below is the format of the entries in the configuration file.

```
A | B | C | D | E | F | G | H | I | J | K
```

where:

A	model
B	model run time minimum
C	model run time maximum
D	model forecast validation hour minimum
E	model forecast validation hour maximum
F	grid projection minimum
G	grid projection maximum
H	grid level minimum
I	grid level maximum
J	grid parameter minimum
K	grid parameter maximum

The entries placed in NOGRIB.CFG are based on values sent in the Product Definition Section (PDS) of the GRIB message. The information in the PDS includes the model that generated the grid, the forecast period this grid covers, and the parameter type and units of the grid. The values used as parameters in NOGRIB.CFG are the same values sent in the PDS.

The first value to an entry in NOGRIB.CFG is the model number, which is the only required value. The values sent are as follows:

PDS number	Model	McIDAS name
39	Nested Grid Model	NGM
64	Regional Optimal Interpolation	ROI
77	Spectral Model, Aviation Run	MRF
78	Medium Range Forecast Model	MRF
83	80 km ETA	ETA
84	40 km ETA	ETA
85	30 km ETA	ETA
86	Mesoscale Atmos. Prediction Sys	MAPS

For example, the entry to discard all output from the MAPS model will look like this:

```
86| -1| -1| -1| -1| -1| -1| -1| -1| -1| -1
```

The second and third values are the model run times to discard. If you are not interested in the MAPS model runs from 03 Z through 09 Z, your entry will look like this:

```
86|  3|  9| -1| -1| -1| -1| -1| -1| -1| -1
```

The fourth and fifth values are the forecast valid times to discard. If you aren't interested in the MRF forecast times beyond 48 hours, your entries should look like this:

```
77| -1| -1| 49|999| -1| -1| -1| -1| -1| -1
78| -1| -1| 49|999| -1| -1| -1| -1| -1| -1
```

The sixth and seventh values relate to the projections in which the models are sent. Some models are sent in more than one projection. It is usually redundant to store more than one projection of the same data. The default configuration for the GRIB decoder when McIDAS-XCD is installed is to discard duplicate projections of the NGM, ROI and MRF.

Below is a table of common projections, their associated models, and the PDS values to enter to discard them.

Projection	Description	Models	PDS values
Mercator	2.5x5.0 global	MRF	21 - 24
Mercator	5.0x5.0 global	MRF	25 - 26
Mercator	1.25x1.25 global thinned	MRF	37 - 44
Mercator	1,25x2.5 North America	NGM, ROI	50
Polar St.	North America	ETA,ROI, NGM, MRF	211

For example, to discard all the 1.25x1.25 Mercator of the MRF, the entry will look like this:

```
77 | -1| -1| -1| -1| 21| 24| -1| -1|-1| -1
```

Use parameters eight and nine to discard particular levels. For example, to discard all data at 900 mb, and levels 350 through 150 mb for the MAPS model, your entries in NOGRIB.CFG will look like this:

```
86 | -1| -1| -1| -1| -1| -1| 900| 900| -1| -1
86 | -1| -1| -1| -1| -1| -1| 150| 350| -1| -1
```

The final two values on the NOGRIB.CFG configuration line are the parameters to discard. Below is a table of the most common values transmitted. You can find a complete list in the file gbtbpds001.2v2 in ~oper/mcidas/data.

Parameter	**McIDAS name**	**PDS value**
Pressure	P	1
Pressure reduced to MSL	P	2
Geopotential Height	Z	7
Temperature	T	11
u-component wind	U	33
v-component wind	V	34
Relative Humidity	RH	52

To discard all relative humidity grids from the ETA model, your entry will look like this:

```
83| -1| -1| -1| -1| -1| -1| -1| -1| 52| 52
```

A complete description of NOGRIB.CFG and the discarding of grids is described in Appendix D of the *McIDAS-XCD Installation and Users Guide*.

Status display

The status display (statdisp) monitors each decoder and ingestor. The program statdisp reads from a file and displays the output as shown below.

McIDAS-XCD Status (wxdata)

Thu Oct 5 17:44:59 1995

##	Src	Ingestor	Time	Byte	Index	Index file	Origin	Wmo	Product
1	DDS	INGETEXT	174403	54111840	11840	FX95278.IDX	KZSE	FXUS	45
3	IDS	INGETEXT	174347	17808880	109552	SM95278.IDX	CWTO	SNVD	17
4	HRS	INGEBIN	174459	6654641		HRS.SPL			

##	Decoder	Time	Begptr	Lasptr	Gridf MD	Grid Row	Col	Text	Index
1	SAODEC	174448	288592	288592	8	53	1154		SA95278
2	RABDEC	174445	59680	59680	28	5	4474		UJ95278
3	SYNDEC	174449	109584	109584	58	6			SM95278
4	SHPDEC	174449	109584	109584	38	18	253		SM95278
5	F14DEC	174436	14672	14672	48	20	238		FO95278
6	WBXDEC	174436	4960	4960				Watch # 1014	WW95278
7	PIRDEC	174436	17056	17056	68	18	133		UA95278
8	TERDEC	174436	46336	46336					FT95278
9	TIRDEC	174436	1632	1632				TIROS 11	TB95278
10	GRIB	173311			5158	141	0	HHPE96 KWBC	051200

The default status display file is DECOSTAT.DAT, which resides in ~oper/mcidas/data. If you want your local decoders to write to their own status display, set the environment variable XCD_disp_file. Then statdisp will use that file to determine what to update. For example, if your local decoders write to their own status display file structure called ~local/mcidas/data/LOCSTAT.DAT, you can force statdisp to read that file by typing the following from the Unix command line.

```
export XCD_disp_file=~local/mcidas/data/LOCSTAT.DAT
statdisp &
```

The contents of the file can only be changed by the ingestors and decoders. The ingestors change the contents of the file any time data is received from a communications port. For the text ingestors, this occurs after a complete data block is received. For the binary ingestors, this occurs after it reads from the port 20 times. The decoders update DECOSTAT.DAT any time the .IDX file that they are indexing through is updated.

Every five seconds, statdisp checks the contents of DECOSTAT.DAT. If the status for any decoder or ingestor has not changed in five minutes, the color of the output turns red. If a line for a decoder turns red, it means that no data for the WMO product identifier that the decoder is interested in was filed in the last five minutes. Except for the GRIB decoder, it is a rare occurrence for a decoder line on the status display to turn red.

An ingestor line turning red means data was not received from the communications port within the last five minutes. DDS, IDS and PPS almost never change to the warning color. If they do, all of them will probably change, indicating a fundamental problem with either the source or your hardware. The ingestor for HRS often changes colors between 10 Z and 12 Z, and again between 22 Z and 00 Z. These are usually quiet periods when no gridded data is sent.

To view the status display on a workstation other than the one processing conventional data, you can set the DISPLAY environment variable to the appropriate X-server, and then start statdisp by entering the following lines from the Unix command line.

```
export DISPLAY=outfield.ssec.wisc.edu:0
statdisp &
```

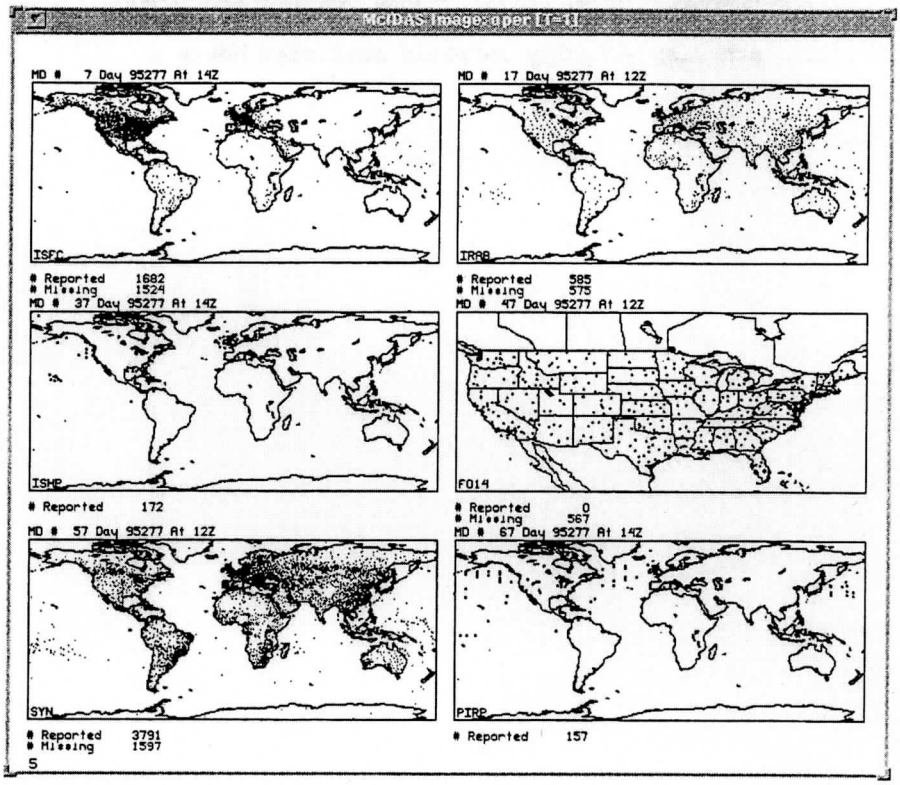
Alternatively, you can use the -display command line option:

```
statdisp -display outfield.ssec.wisc.edu:0 &
```

The DATARECV program

You can use the McIDAS-XCD program DATARECV to graphically display which stations are expected to be decoded for a data type, and which stations were decoded the previous hour. A sample display is shown below.

Although you can't discern the color of the dots below, the white dots on your display indicate the stations with their decoder turned on, and that data was received for those stations. Red dots indicate stations with their decoder turned on, but data was not filed by that decoder for those stations.



Allocating disk space

Below is a list of all dynamic files used in McIDAS-XCD and the approximate disk space they consume per day, in MB.

Data type	Format	Disk space per day
DD+	TEXT	75
ID	TEXT	25
HRS	SPOOL	16
SFC	MD	20
IRAB/IRSG	MD	6
ISHP	MD	3
FO14	MD	2
SYN	MD	7
PIRP	MD	6
ETA	GRID	52
NGM*	GRID	50
MRF*	GRID	350
MAPS	GRID	140
Other grids	GRID	1
Total		751 MB

* if you save all the projections

Deleting text, MD and grid files

Up to 751 MB per day is a considerable amount of data and could fill your file system to capacity within a matter of days. These two McIDAS programs can help you remove old data files:

- DELWXT removes text and index files
- QRTMDG removes MD or grid files

They are designed to run from the McIDAS scheduler to delete files older than a specified number of days. For example, to keep four days worth of data online and remove all grid, MD, and text data older than that, enter the following three lines in the McIDAS schedule.

```
#Y 00:05:00 999999 00100:00:00 "DELWXT 4
#Y 00:05:00 999999 00100:00:00 "QRTMDG MD md1 mdn 4
#Y 00:05:00 999999 00100:00:00 "QRTMDG GRID grid1 gridn 4
```

Archiving data

The most complete way to archive data is to save the text (*.XCD) and index (*.IDX) files, and the MD and grid files.

Raw text

Use the steps below to archive raw text.

1. Collect the necessary files into an archive file. The resulting file will be approximately 100 Mb for the Family of Services text circuits.

   ```
   tar -cvf TEXT93017.tar IDXALIAS.DAT ??93017.IDX ??930170.XCD
   ```

2. Compress the file. The resulting .Z file will be about 19 Mb.

   ```
   compress TEXT93017.tar
   ```

3. Write the compressed tar file to tape, replacing `tape-device` with the unix file name of your tape drive.

   ```
   tar -cvf tape-device TEXT93017.tar.Z
   ```

Real-time MD files

Use the steps below to archive real-time MD files.

1. Collect the necessary files into an archive file. The resulting file will be approximately 45 Mb for the standard McIDAS-XCD real-time files.

   ```
   tar -cvf MD93017.tar MDXX00?7
   ```

2. Compress the file. The resulting .Z file will be about 2.5 MB.

   ```
   compress MD93107.tar
   ```

3. Write the compressed tar file to tape, replacing `tape-device` with the unix file name of your tape drive.

   ```
   tar -cvf tape-device MD93017.tar.Z
   ```

Real-time grid files

Use the steps below to archive real-time grid files.

1. Collect the necessary files into an archive file. The resulting file will be approximately 400 Mb for the standard McIDAS-XCD real-time files.

   ```
   tar -cvf GRID93017.tar GRID5[0-3]???
   ```

2. Compress the file. The resulting .Z file will be about 200 Mb.

   ```
   compress GRID93107.tar
   ```

3. Write the compressed tar file to tape, replacing `tape-device` with the unix file name of your tape drive.

   ```
   tar -cvf tape-device GRID93017.tar.Z
   ```

Recovering archived data

Use the steps below to recover archived data files.

1. Change the directory to an appropriate directory in which to recover the data.

2. Extract the archived grid files.

   ```
   zcat GRID93017.tar.Z | tar -xvf -
   ```

   This command uncompresses the file and extracts the tar file.

3. Extract the archived text files

   ```
   zcat TEXT93017.tar.Z | tar -xvf -
   ```

4. Extract the archived MD files

   ```
   zcat MD93017.tar.Z | tar -xvf -
   ```

Troubleshooting

Below are some typical problems you may encounter and their solutions.

No real-time data

Symptoms: The user reports no real-time data, or the status display is red for the ingestors.

Problem: The file system is full.

The antenna has an obstruction or there is a problem with the antenna hardware.

The source provider is experiencing a problem.

Solution: Check the status of the file system on the workstation with the Unix command **df**. If the file system is full, clean the file system. You can use the McIDAS-XCD programs QRTMDG and DELWXT to delete older text, MD files and grid files. Do not delete any files for the current day.

Check for an obstruction in the antenna and verify that all receiving hardware is working properly.

Contact your source provider to see if they are having a problem with the broadcast.

No grid data

Symptom: The GRIB decoder is not filing grids.

Problem: The decoder can't find RTMODELS.CFG, which contains information about the grid files to search.

Solution: The file RTMODELS.CFG should reside in ~mcidas/data when McIDAS-XCD is installed correctly. Either the decoder can't reach the file or it is missing. If it's missing, either recreate the file or copy a new version from ~mcidas/xcd1.1/data/RTMODELS.CFG, if your site is using the default configuration for real-time grid file locations.

Garbled or missing data

Symptom: Text data is missing or text output is garbled. If decoding grids, grids are missing.

Problem: More than one ingestor is trying to read the same circuit, resulting in garbled or missing data. Or, there could be an obstruction in the receiving antenna.

Solution: Check the process status of the system to see how many ingetext and ingebin processes are running. There should be only one ingetext process running for each text circuit, and one ingebin process running for each binary circuit. If this isn't the case, stop all McIDAS-XCD processes that are running.

Do a ps to get the PID of all the -XCD processes; for example: **ps -u | grep oper**. Use the kill -9 *PID* command to stop the -XCD processes in the following order.

```
startxcd.mx
all ingetext.mx
all ingebin.mx
all dm*.mx
```

Then restart -XCD with the McIDAS command STARTXCD.

If the above process doesn't work, check for an obstruction in the receiving antenna.

TIROS data lacks navigation

Symptom: Users report that TIROS data does not contain navigation.

Problem: The McIDAS-XCD decoder TIRDEC may not be running or is not able to write the SYSNAV1 file.

Solution: Verify that the TIRDEC decoder is turned on and the decoder configuration file is correct. If you are ingesting TIROS with McIDAS-XSD, verify that your -XSD ingestor can retrieve the SYSNAV1 file from the -XCD ingestor.

McIDAS Operations on a Distributed System

Presented by
Chad Johnson
Operations Programmer

Session 12
McIDAS Developer/Operator Training
October 23-25, 1995

Table of Contents

Overview 12-1
Terminology 12-1
The McIDAS distributed system 12-2

The ADDE server 12-3
ADDE server transaction logging 12-3
ADDE server security 12-4
ADDE server installation 12-5
ADDE server configuration 12-6
ADDE server startup 12-11

McIDAS system support 12-12
File system limitations 12-12
NFS (Network File System) 12-13
Off-line data storage and tape devices 12-14
Bringing a backup ingestor and server online 12-15
Frequently asked questions 12-16

Overview

This training session will provide McIDAS site administrators and operators with information about McIDAS' role in a distributed data system. It includes the following topics:

- installing and configuring ADDE (Abstract Data Distributed Environment) servers
- providing server security
- choosing dataset names
- following the life cycle of the server as it fulfills a request

This session will also present the following topics, which should be of interest to Unix system administrators:

- file system limitations of the supported platforms
- NFS (Network File System) and where it might be useful
- off-line data storage and tape devices in a distributed system
- how to bring a backup ingestor/server online

Terminology

The terms below are defined as they apply to a McIDAS distributed system. Some of these definitions may not be applicable to other distributed systems.

client	workstation receiving and displaying data; initiates the requests
daemon	background process that periodically wakes up, checking to see if it should do something
dataset	collection of one or more files with a common format
dataset name	name used by ADDE to reference a dataset; consists of the group name and a descriptor

descriptor	name used to reference a dataset, separated by a slash (/).
file system	method for cataloging files in a computer system
group	name used to reference a collection of descriptors; used by the client to determine which server to query for data requests
inetd	Unix system daemon that listens to various network ports
mcserv	ADDE server program started by **inetd** when a network connection is made to a port
server	the machine that stores and supplies data in response to clients' requests
transaction logging	record keeping done by ADDE servers for each transaction

The McIDAS distributed system

A distributed system is a computing system in which the storing and serving of data are distributed across multiple workstations. For example, users may be working at a workstation, yet the data they are using may exist on other workstations. Users request the data from the data storage machines. The request is processed and the data is sent to the user.

The McIDAS distributed system is similar, but differs from the generic distributed system in two ways:

- the ingest of data may also be done on the data storage and serving machines
- client and server software have specific data abstractions they support, such as the McIDAS Area file in ADDE

ADDE has been part of the McIDAS-X software package since the June 1995 upgrade.

The ADDE server

This section provides information about the ADDE server, including the following topics:

- transaction logging
- security
- installation and configuration
- startup

ADDE server transaction logging

ADDE servers can provide logging for each request they serve. You can initiate transaction logging for the following reasons:

- accounting
- server usage statistics
- dataset usage statistics

When server transaction logging is turned on, each transaction is logged as a record to the file SERVER.LOG. This file resides in ~mcadde/mcidas/data or the first writable directory in the environment variable MCPATH. Some of the parameters available in each log record are: IP of the server and client, date and time of the transaction, user initials, project number, return code, dataset name, bytes sent and received, CPU time, and error messages.

The SERVER.LOG file is a continuously growing file. To prevent this file from becoming very large, schedule a move of this file to a different file. You can schedule the move via the McIDAS scheduler or the Unix cron scheduler. SERVER.LOG is created again when the next transaction record is written.

You should consider these two forms of security before setting up your ADDE server:

- file security, which attempts to prevent file deletion or modification by unwanted sources
- access security, which attempts to prevent unauthorized users from accessing your data

File security

The Unix operating system allows multiple users to share the same workstation's processing power and storage space. In this type of working environment, it is logical to have a system that prevents users from being able to modify and overwrite system and other user's files. In the Unix operating system, each file is owned by a user and a group, with a set of associated permission flags. The owner can set read, write, and execution privileges for each class: user, group and all other users on the system. It is this that gives the ADDE servers a method of file protection.

To prevent users from writing to an ADDE dataset on your server, you should set up another user account named *mcadde.* The ADDE server will then be configured to run from this account. The name of the account you choose doesn't matter, as long as it is different from the account of your data ingestion/storage.

The *oper* account is typically used for ingest and data storage. The permission of the data files in this account should be set to allow reading for all users, but writing by only *oper.* This will prevent the server from overwriting your real-time data.

Access security

Access security is available with ADDE by turning on transaction logging, as described on the previous page.

Access security uses the three pieces of information below, which are sent to the server with each data request:

- user initials
- project number
- IP address of the client machine

The list of valid user initials, project numbers, and IP addresses is contained in three text files: SERVER.USR, SERVER.PRJ and SERVER.IP, respectively. These files are created by the operators and should exist in the ~mcadde/mcidas/data directory of each server workstation. The format of the file is one line per parameter; comments begin with a *. Below is an example of a SERVER.PRJ file.

```
* DDE VALID PROJECT NUMBER LIST
1000
1001
1002
1003
1004
1006
```

Access validation occurs if any of the above files exist. It is not mandatory that all files be present. For example, if only the SERVER.USR file exists, only requests made by users listed in that file are allowed. If all files exist, validation will be checked for all three parameters, and only requests made by valid users with valid project numbers from a valid workstation are allowed. If none of the above files are present, no validation will take place and all requests are allowed.

ADDE server installation

The ADDE server/client software package was included in the standard distribution and installation of the June 1995 McIDAS-X upgrade. To collect transaction logs or use the method of access security mentioned above with the ADDE server, you must perform additional steps when installing McIDAS-X 2.1. You must compile McIDAS-X 2.1 with the DDE_ACCOUNTING compile flag turned on. The steps to accomplish this are provided below.

1. Logon to the workstation as user *mcidas*.

2. Do not run the installation script **mcidas2.1*version*#.sh**.

3. Extract the tar file manually.

   Type: **zcat mcidas2.1*xx*.tar.Z | tar -xvf -**

   This will extract the tar file and create the mcidas2.1*version#* directory. This directory contains the src and data directories.

4. Change the directory to the McIDAS source directory.

   Type: **cd $HOME/mcidas2.1*xx*/src**

5. At the shell prompt, export the following two variables.

   Type: **export McINST_ROOT=$HOME**
   export McIDAS_ROOT=$HOME

6. Build McIDAS. This will build the ADDE servers to perform transaction logging.

   Type: **make INCARGS="-DDDE_ACCOUNTING"**

7. Have all your McIDAS users exit their McIDAS sessions.

8. Install the binaries that will perform transaction logging.

   Type: **make install**

9. Have all McIDAS users start their McIDAS sessions.

The ADDE servers will now write to the LW file, SERVER.LOG, in the ~mcadde/mcidas/data directory as transactions are logged. This is a binary file and is not viewable by any text editor. However, McIDAS Operations has software to view it. Contact McIDAS Operations if you are interested in any of these utilities.

ADDE server configuration

ADDE servers will typically run on the same workstation performing the ingest. For example, you may have multiple workstations ingesting data from multiple satellite sources. Each of these workstations must be configured to run an ADDE server.

Naming the account

When setting up a workstation as an ADDE remote server, you must create a remote server account on the workstation. Consider the following guidelines when deciding on a name for the remote server account.

- Don't use the *mcidas* account. You can't run McIDAS-X from the *mcidas* account or other SSEC-supplied software packages that run under it.

- Don't use any accounts that run McIDAS-XCD and McIDAS-XSD packages which provide conventional or satellite data that you may want to make available to users. For example, don't use the *oper* account.

- Don't use accounts that may hold data you want to provide to users via the ADDE server.

- Dedicate this account to the administration of the ADDE remote server. For example, use it only for acquiring and naming data for ADDE clients.

We recommend naming this new account *mcadde*. This account must be configured as a McIDAS-X user account; for example, setting the appropriate PATH and directories to run McIDAS-X.

Configuring the workstation

You must configure the Unix workstation to accept port connections from ADDE clients and to run the server to fulfill these requests. Included with the June '95 McIDAS-X upgrade is the script **mcinet***version#***.sh**, which will configure the system daemon **inetd** to do this.

The script **mcinet***version#***.sh** configures two system files:

- /etc/services

- /etc/inetd.conf

The file /etc/services specifies the names of services available through the Internet and the protocol of these services, and assigns a port number that each service will connect. Below is an example of the /etc/services file.

```
# UNIX specific services
#
# these are NOT officially assigned
#
nfsd        2049/udp    nfs    # NFS server daemon
mcserv      500/tcp             # McIDAS ADDE port
xcd_rlyc1   502/tcp             # XCD core data stream
```

The ADDE service is named mcserv and is defined as a TCP service that connects to port 500.

The /etc/inetd.conf files contains a list of programs that **inetd** will start when it receives an Internet request on that port. For example:

```
mcserv      stream tcp      nowait          mcadde
     /home/mcidas/bin/mcservsh          mcservsh -H /home/mcoper
```

You must configure the ADDE server account so the servers can locate the data files. The ADDE servers use the MCPATH environment variable and the REDIRECT table to locate the McIDAS data files. For example, if you run a McIDAS-XSD ingestor on this same system, you will add the path to the data files in the environment variable MCPATH or add them to the REDIRECT table of the *mcadde* user.

Assigning dataset names

To assign dataset names, perform the steps below.

1. Verify that McIDAS of the server account is able to locate the data files you want to serve. You can do this by adding the data directory to the MCPATH environment variable, or adding it to the REDIRECT table.

2. Choose the dataset name for this dataset.

3. Start a McIDAS session under the ADDE server account and add this dataset using the McIDAS command DSSERVE.

4. Notify your users that you have a new dataset available. Give them the name of the dataset and the host name or IP address of the server workstation.

For example:

Suppose areas 1000 through 1024 are GOES-8 CONUS visible areas and the files reside in /home/oper/mcidas/data on the workstation foo.ssec.wisc.edu. Verify that the McIDAS server account can locate areas 1000 through 1024 in the /home/oper/mcidas/data directory by starting McIDAS in the server account.

Type: **DMAP AREA**

If the paths listed for AREAs 1000 through 1024 are different from /home/oper/mcidas/data, you need to add the directory to the MCPATH environment variable.

In this example, we will assign the dataset name to *EAST/CONUSV*. Start a McIDAS session in the server account and assign this dataset to a group of areas by entering the command below.

DSSERVE ADD EAST/CONUSV AREA 1000 1024 "GOES-8 visible CONUS

The ADDE server is ready to serve AREAs 1000 through 1024 as dataset EAST/CONUSV. Notify your users that GOES-8 visible CONUS data is available from the server foo.ssec.wisc.edu as dataset EAST/CONUSV. They should add this dataset to their client routing tables with the following command.

Type: **DATALOC ADD EAST "foo.ssec.wisc.edu**

Selecting a dataset name

All ADDE commands use dataset names (in a group/descriptor format) that map to datasets on a server. It is easier for users to locate the data if you follow a logical convention when assigning group and descriptor names. There are three tiers in the hierarchical naming scheme for datasets.

- type
- group
- descriptor

Type is the top tier and can be either image, grid, or point.

Group is the next tier in the naming scheme. A group name can be used only once under each type. Groups are defined by the operator when assigning dataset names to a dataset; for example, a range of McIDAS areas.

Descriptors are the bottom tier in the naming scheme. They further classify or describe the dataset. Descriptors are defined by the operator when assigning dataset names to a dataset.

For example:

Group/descriptor	Data format		Comment
WEST/ALL	AREA 101	150	ALL DATA FROM WEST SATELLITE
WEST/CONUSV	AREA 101	104	CONTINENTAL US; 1KM; VIS; GOES-7
WEST/DS	AREA 131	136	DWELL SOUNDING; 8KM; GOES-7
WEST/DSV	AREA 141	142	DWELL SOUNDING; 1KM; GOES-7
WEST/FDIR	AREA 109	112	FULL DISK; BAND 8; 4KM; GOES-7
WEST/FDMSI	AREA 121	128	FULL DISK; MSI; 8KM; GOES-7
WEST/FDV	AREA 105	108	FULL DISK; VISIBLE; 4KM; GOES-7

The table above shows datasets that are being served by a server. All the datasets with the group name WEST reference AREA files ingested from the GOES-7 satellite. There are multiple WEST groups, each with a different descriptor name. Each unique group/descriptor pair represents a defined dataset on the server. Users locate the data using that name.

Defining group names

Below are some hints to help you select group names when defining your dataset names.

Group names are used by the client to locate data. You cannot use a group name that is already used to describe data served from another workstation. You can use the same group name for different types served from the same server.

Group names should describe the data source. For example, if you are serving GOES-8 data, you can use EAST as the group name in all your dataset names referencing data for the east satellite on that server.

Choose group names that do not change often. For example, don't use satellite names such as GOES8 as group names.

Defining descriptor names

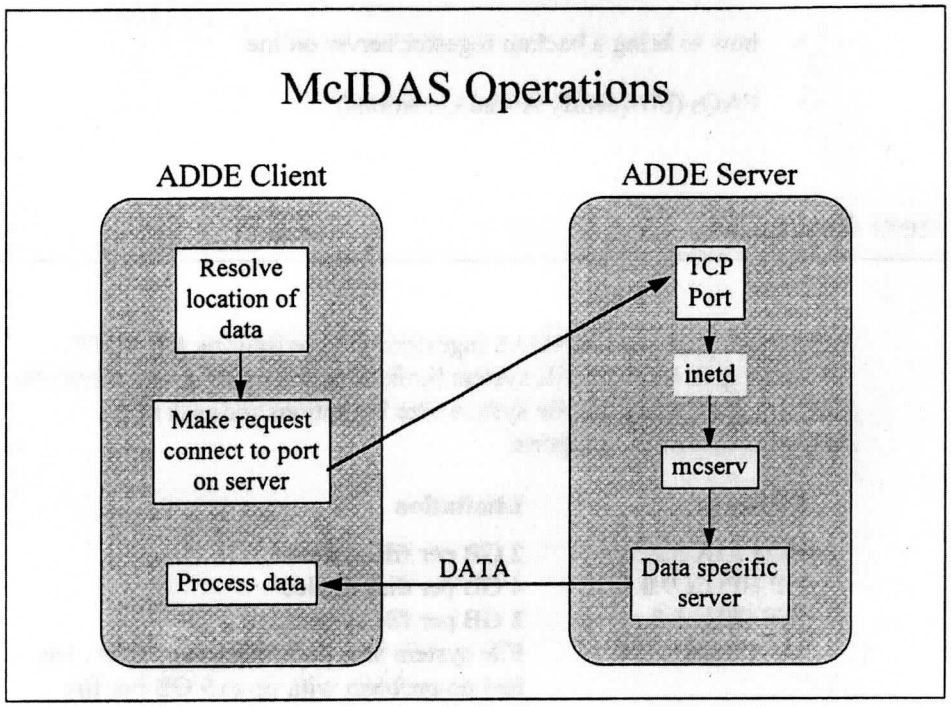
Below are some hints to help you select descriptor names when defining your dataset names.

When assigning descriptor names for gridded data types, the name should describe the time of the grid, or the model run time and verification time if the grid is model output.

Descriptors should describe the geographical coverage of the data. For example, a visible image area that covers a region in the Northern Hemisphere could have a descriptor name such as NHEMV.

Make the names as short as possible. Some users may make aliases for the dataset names, but all users may not. In this case, the users must remember and type in the names you have chosen.

A user on a workstation starts an ADDE client process by entering the McIDAS command IMGCOPY. This client process looks in the DATALOC table to determine the IP address of the server machine that is serving the data requested. The client attempts to make a connection to port 500 of the server machine. If the connection is successful, the **inetd** daemon on the server machine wakes up and determines which service is defined for port 500. **inetd** then starts the program **mcservsh**, which examines the request and activates the appropriate data server for finishing the request.



There are several data servers, and each data server knows only about the data type it serves. For instance, the data server **agetserv** only knows how to subsect and serve McIDAS AREAs; the **ggetserv** server only understands how to serve McIDAS GRIDs. Below is a list of data servers and the services they provide.

Data server	Service
adirserv	sends McIDAS area directory listings
agetserv	sends McIDAS areas from a server
aputserv	receives McIDAS areas from a client
gdirserv	sends McIDAS grid directory listings
ggetserv	sends McIDAS grids from a server
gputserv	receives McIDAS areas from a client

McIDAS system support

This section discusses topics of interest to site administrators who must set up a McIDAS distributed system. The topics discussed include:

- file system limitations
- NFS (Network File System)
- off-line data storage
- how to bring a backup ingestor/server online
- FAQs (Frequently Asked Questions)

File system limitations

When setting up your McIDAS ingestors and configuring the ADDE server, keep in mind any file system limitations that exist for that particular platform. Below are the file system size limitations for each of the platforms that SSEC supports:

Platform	Limitation
IBM AIX 3.2.5	2 GB per file system
HP HPUX 9.0.5	4 GB per disk device
SGI IRIX 5.3	8 GB per file system
SUN Solaris 2.4	File system size limit unknown; SSEC has had no problem with up to 9 GB per file system.

HPUX has a limitation of 4 GB per device. If you purchase a 9 GB disk drive for an HP system, you will only be able to utilize 4 GB (less than half) of the total storage capacity of that drive. The remaining 5 GB cannot be partitioned and mounted on additional file systems.

NFS is a protocol and program set that allows file transfers to occur over a network. It allows one workstation to share file systems with other workstations on the network, rather than keeping separate copies of these files on each workstation. When NFS is used, this shared file system will appear in the local file system as though it really does exist there, although the files actually reside in the file system on a remote workstation.

Some examples of where NFS may be used in the McIDAS distributed system include the following:

- If you're sharing the -XCD data directory, users can mount the data directory on their system.
- If you're performing access restrictions, you can share server initial, project, or IP address files; for example, SERVER.USR, SERVER.PRJ, SERVER.IP.
- If you're performing transaction logging, you can mount the transaction log files from multiple workstations on one workstation.

To use NFS, you must **share** or **export** the file system containing the sharable files. When you share a file system, you allow this file system to be remotely mounted on a remote host. You have the option to allow only certain hosts to mount this file system, or you can give read-only permission for connecting hosts. A shared file system can be thought of as being served by an NFS server. To gain use of this shared file system on the external host, users must **mount** the shared file system on their local workstations. As with most system configurations, you must be *root* to share or mount remote file systems.

NFS relies on the RPC (Remote Procedure Call) subsystem. If you have trouble sharing or mounting shared file systems, check to make sure RPC is running on your systems.

Type: **ps -ef | grep rpc**

You should see a line containing the string "rpc.bind" in the output. If you don't see it in the process listing, RPC isn't running. There typically is an administrative script to start the RPC subsystem under the /etc directory. The name and location of this script varies among platforms; contact your system administrator for details.

You can connect many different tape devices to Unix workstations for off-line data storage; for example: DAT-4mm, Exabyte-8mm, 9-Track, 3480, 3490E, 3590. The media formats that your site supports will depend on numerous factors, which may include:

- previous experience
- reliability
- the media on which the majority of your data is stored
- future trends in off-line storage

In a distributed system it is inevitable that the files you would like to write to tape are on a different workstation than the tape devices. Listed below are three possible solutions to this problem.

- Ftp or rcp (remote copy) the files to the system with the tape device.
- NFS mount the source file system to a file system on the workstation containing the tape drive; the data is then accessible via a file system on the machine with the tape drive.
- Execute a remote tar write across the network; for example, to write the files to a remote device,

  Type: **tar -cvf - | rsh *host* dd of=*device***

  To read the files from a remote device type,

  Type: **rsh -n dd if=*device* | tar -xvf -**

  where ***host*** is the name of the remote system with the tape device, and ***device*** is the name of the device on the remote system you want to write.

Streaming tape devices, such as 8mm and 4mm devices, are very sensitive to having a steady stream of data. When writing across a network to these types of devices, network latency may cause tape write or read failures. To prevent this, you should have 8 Mbits per second bandwidth available on your network when writing to a streaming tape device. Due to the burst nature of network traffic, you can't assume that the required bandwidth necessary for the device will be present at the moment data is written to tape. Options one and two above are not recommended.

It is not recommended that you use any hardware compression that your drive manufacturer supports. Different hardware vendors may not support the same compression algorithm, resulting in an unreadable tape when transferring it to another tape drive.

Bringing a backup ingestor and server online

Your backup system must be configured exactly as your primary system. Items that must be configured on the backup system include the following:

- If this is a hot spare for immediate switchover, verify that you have a second signal feed for the source you intend to back up.
- The McIDAS-XSD or -XCD software must be installed and configured to ingest the same data source as the primary ingestor.
- If this is a backup for McIDAS-XCD, configure the decoders just as they are for the primary machine.
- If this is a McIDAS-XSD ingestor, the satellite scheduler windows and McIDAS scheduler entries must be entered exactly as they are on the primary system.
- The ADDE server should be configured to use the same dataset names as the real-time system to reference data files. This makes the switchover to the hot spare easiest for the user.

Notify users that you are switching to a different workstation to serve data. Inform them of the following:

- the name of the backup workstation
- any changes to the dataset names

If the dataset names were not changed, the users need only run the McIDAS command **DATALOC** to change their routing table to point to the backup workstation for that group. Users should be able to access the data just as they had before with no change required to any scheduled commands.

If the dataset names were modified for some reason, the users must run the McIDAS command **DATALOC** to add the new group to their routing table. They must also modify their scheduled data retrievals to use the new dataset names.

What is the effect of changing a machine's IP address?

Changing the IP address should have no effect on users, provided they reference that machine by the host name and not the IP address. When an IP address changes on a system, the local name server must be updated. This change will bubble up to the other name servers on the internet system, although this may take a few hours to occur. If users have difficulty accessing this machine, have the system administrator flush the cache of their name server.

If the system is an ADDE server, have all ADDE users who have IP numbers in their dataloc tables, change the IP address of this server with the McIDAS command DATALOC. Also, have all ADDE users run this McIDAS command: **DATALOC HOST**

ADDE client software does not look up the address for the server when making each request. This command refreshes the internal name/IP lookup table for servers.

How do I know who's accessing the server at any given time?

Use the Unix utility, **netstat**, to display the network status of the workstation. Appended to the end of this document is a shell script that you can use to display a continually updated status of ADDE server connections.