Naval Research Laboratory
Stennis Space Center, MS 39529-5004

NRL/MR/7441--97-8073

Compressed Aeronautical Chart Access Software

PERRY B. WISCHOW
MAURA C. LOHRENZ

Mapping, Charting, and Geodesy Branch
Marine Geosciences Division

July 24, 1998

REPORT DOCUMENTATION PAGE

Form Approved
OBM No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 24, 1998	Final

4. TITLE AND SUBTITLE

Compressed Aeronautical Chart Access Software

5. FUNDING NUMBERS

Job Order No. 574562500
Program Element No. DMA
Project No.
Task No.
Accession No. DN154123

6. AUTHOR(S)

Perry B. Wischow and Maura C. Lohrenz

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

Naval Research Laboratory
Marine Geosciences Division
Stennis Space Center, MS 39529-5004

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/MR/7441--97-8073

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

National Imagery and Mapping Agency
8613 Lee Hwy.
Fairfax, VA 22031-2137

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Compressed Aeronautical Chart (CAC) data base is a global library of compressed, scanned, aeronautical charts that support Navy and Marine Corps aircraft moving-map displays and mission planning systems. The source for the CAC library is the National Imagery and Mapping Agency (NIMA) standard ARC (equal Arc-second Raster Chart) Digitized Raster Graphics (ADRG) image data set. ADRG is compressed and transformed into CAC via vector quantization and color compression techniques. The Map Data Formatting Facility (MDFF) of the Naval Research Laboratory, Stennis Space Center, MS (NRLSSC), produced the CAC library from April 1989 until September 1995, when NRLSSC transitioned the CAC Production System to NIMA.

This port is a programmer's reference for accessing the CAC library via NRL-developed CAC Access Software, which is a user-callable suite of utility programs. The CAC Access Software was written in ANSI C and is currently running under the following operating systems: Open VMS, Unix, MS-DOS, Windows 3.1, Windows 95, and Macintosh.

14. SUBJECT TERMS

digital maps, optical storage, data bases, data compression, aeronautical charts, mission planning

15. NUMBER OF PAGES

32

16. PRICE CODE

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

Compressed Aeronautical Chart Access Software

Perry B. Wischow and Maura C. Lohrenz

Contents

Introduction

The Compressed Aeronautical Chart (CAC) database is a global library of compressed, scanned, aeronautical charts that support Navy and Marine Corps aircraft moving-map displays and mission planning systems. The source for the CAC library is the National Imagery and Mapping Agency (NIMA) standard ARC (equal Arc-second Raster Chart) Digitized Raster Graphics (ADRG) image data set. ADRG is compressed and transformed into CAC via vector quantization and color compression techniques. The Map Data Formatting Facility (MDFF) of the Naval Research Laboratory, Stennis Space Center, MS (NRLSSC), produced the CAC library from April, 1989, until September, 1995, when NRLSSC transitioned the CAC Production System to NIMA.

Table 1. Available CAC scales and chart series

Scale	*Chart Series*
1:5M	Global Navigation Chart (GNC)
1:2M	Jet Navigation Chart (JNC)
1:1M	Operational Navigation Chart (ONC)
1:500k	Tactical Pilotage Chart (TPC)
1:250k	Joint Operational Graphics (JOG)
1:100k	Topographic Line Map (TLM) - 100
1:50k	TLM-50

NIMA distributes CAC installments on Compact Disk Read-Only Memory (CDROM). Each CDROM contains data at one of seven available chart scales, from 1:5M (M=million) to 1:50k (k=thousand), as listed in table 1. As of 1995, there were 34 CDROMs in the CAC library (table 3). For a more recent listing, the reader is referred to NIMA Customer Support.

Table 2. TS geographic zones

Zone ID	*Zone name*	*Southern latitude*	*Northern latitude*
0	South Polar	90.00 S	51.69 S
1	S. Temperate	51.69 S	31.38 S
2	Equatorial	31.38 S	31.38 N
3	N. Temperate	31.38 N	51.69 N
4	North Polar	51.69 N	90.00 N

CAC data is structured according to the Tessellated Spheroid (TS) map projection system. TS divides the world into five zones (table 2). Each zone is divided into rows and columns of segments, and each segment represents approximately 2 in × 2 in of paper chart. The geographic coverage of a segment is dependent on the chart scale and the zone in which the segment is located. Lohrenz, et al. (1993) describes the TS projection system in more detail, and Lohrenz and Ryan (1990) documents the CAC file structure. The reader is advised to become familiar with these reports prior to using the CAC Aceess Software.

This report is a programmer's reference for accessing the CAC library via NRL-developed CAC Access Software, which is a user-callable suite of utility programs. The CAC Access Software was written in ANSI C and is currently running under the following operating systems: OpenVMS, Unix, MS-DOS, Windows 3.1, Windows 95 and Macintosh. Appendices A, B, and C of this report contain the entry point descriptions for the High-Level, Low-Level and Miscellaneous Access Routines, respectively. Appendix D contains the High-Level Function Calling Order.

Table 3. CAC library installments as of September, 1995.

MDFF Library #	DMA Stock #	ed.	Geographic Coverage	Date
		GNC (1:5M scale)		
CD-1995-A-MAP6-00033	ACNxxGNCxx01	1	Worldwide coverage	09/95
		JNC (1:2M scale)		
CD-1991-B-MAP5-10006	ACNxxJNCxx01	1	N/S Am., Greenland, Australia, USSR	10/91
CD-1992-A-MAP5-00008	ACNxxJNCxx02	1	Europe, USSR, Africa, China, Japan	03/92
		ONC (1:1,000,000 scale)		
CD-1994-A-MAP4-00032	ACNxxONCxx01	1	Eurasia	9/94
CD-1991-A-MAP4-00010	ACNxxONCxx02	1	N/S Am., Greenland, Iceland, Arctic O.	08/92
CD-1994-A-MAP4-00031	ACNxxONCxx03	1	India, Indonesia, S.Pacific	08/94
CD-1994-A-MAP4-00030	ACNxxONCxx04	1	Africa, Saudi Arabia	07/94
CD-1994-A-MAP4-00029	ACNxxONCxx05	1	S.America, Australia, So. Pacific	05/94
		JOG (1:250,000 scale)		
CD-1991-A-MAP2-10007	ACNxx1501A09	1	Western U.S.	11/91
CD-1992-A-MAP2-00009	ACNxx1501A10	1	Eastern U.S.	04/92
CD-1993-A-MAP2-00016	ACNxx1501A16	1	Sea of Japan	03/93
CD-1993-A-MAP2-00017	ACNxx1501A2021	1	S. China Sea	03/93
CD-1993-A-MAP2-00018	ACNxx1501A27	1	Somalia/Ethiopia	04/93
CD-1993-A-MAP2-00019	ACNxx1501A12	1	Western Mediterranean	05/93
CD-1993-A-MAP2-00020	ACNxx1501A19	1	Saudi Arabia	05/93
CD-1993-A-MAP2-00022	ACNxx1501A23	1	Caribbean	07/93
CD-1993-A-MAP2-00023	ACNxx1501A13	1	Black Sea & Caspian Sea	09/93
CD-1993-A-MAP2-00024	ACNxx1501A22	1	Central America	09/93
CD-1994-A-MAP2-00026	ACNxx1501A04	1	U.K. & Baltic	02/94
CD-1993-A-MAP2-00027	ACNxx1501A08	1	Alaska	02/94
CD-1994-A-MAP2-00028	ACNxx1501A14	1	Afghanistan & NE Iran	05/94
		TPC (1:500,000 scale)		
CD-1995-C-MAP3-10001	ACNxxTPCxx0710	3	Update of U.S., Caribbean, and Panama	09/95
CD-1991-B-MAP3-10002	ACNxxTPCxx0309A	2	Update of Desert Storm	06/91
CD-1991-A-MAP3-10003	ACNxxTPCxx0512A	1	W. Pacific Rim & Hawaii	03/91
CD-1991-A-MAP3-10004	ACNxxTPCxx0506	1	No. Pacific (incl. Alaska & NE USSR)	04/91
CD-1991-A-MAP3-10005	ACNxxTPCxx0208A	1	Med., Europe, Scandinavia, Iceland	06/91
CD-1992-A-MAP3-00011	ACNxxTPCxx0411A	1	India and China	10/92
CD-1992-A-MAP3-00012	ACNxxTPCxx0304	1	W. Russia and E. Mongolia	11/92
CD-1992-A-MAP3-00013	ACNxxTPCxx0405	1	Siberia	01/93
CD-1992-A-MAP3-00014	ACNxxTPCxx1216	1	Australia, E. Indonesia	01/93
CD-1992-A-MAP3-00015	ACNxxTPCxx1014	1	S. America	01/93
CD-1993-A-MAP3-00021	ACNxxTPCxx0809	1	N. Central Africa	07/93
CD-1993-A-MAP3-00025	ACNxxTPCxx0915	1	Southern Africa and Madagascar	10/93

CAC Access Software

The CAC Access Software is made up of twelve files, including four C programs (*.c) and eight include files (*.h), as listed in table 3. The CAC Access Software is designed to allow a programmer both high-level and, if necessary, low-level access to a CAC CDROM. The high-level routines will be sufficient for most applications, but a user may require low-level access for more advanced applications that manipulate the image data.

Table 3. CAC Access Software files.

Programs	*Description*	*Support files*
cac_hlev.c	High level CAC access routines	
cac_llev.c	Low level CAC access routines	
cac_misc.c	Routines to access non-image CAC data files (e.g., audit trail files)	areadrc.h, areasorc.h, cd_header.h, dr_header.h, sg_header.h, pa.h
	C language structures and definitions; required by all four high-level programs	cac_inc.h
	Definitions for TS projection system; required by all four high-level programs	m4_const.h

High-Level Access Routines

The high-level CAC access routines, which are contained in the file `cac_hlev.c`, consist of four entry points: *cac_init, cac_inq_palette, cac_get_ll,* and *cac_get_rc*. These routines initialize the software, read the appropriate color palette, and retrieve the compressed CAC data for either a specific geographic point (latitude and longitude) or segment (row and column). The high-level routines will suffice for most user applications, such as displaying CAC data. Examples of the recommended calling sequences for these high-level routines are provided in the files `main_rc.c`, and `main_ll.c` (listed in Appendix D). Appendix A documents the high-level routines in detail. They and listed here in the intended calling sequence:

- *cac_init*
  Initializes the CAC retrieval software by allocating the necessary memory for segment buffering, reading the `cd_id.dat` and `cd_covrg.dat` files from the ID directory on the specified device, and initializing the internal CAC data structures based on the contents of the `cd_id.dat` and `cd_covrg.dat` files.

- *cac_inq_palette*
  Reads and returns the "day", "night" or "mono" color palette that corresponds to the retrieved CAC segment data. Each CAC color palette file includes three separate palettes: the day palette is appropriate for daytime flight, the night palette is used for nighttime flight, and the mono palette uses only gray shades. See Lohrenz, et al., for more information about CAC color palettes.

- *cac_get_ll*
  Retrieves the pixel value specified by latitude and longitude. Also retrieves the entire decompressed segment, if required.
- *cac_get_rc*
  Retrieves the entire decompressed segment specified by a TS row, column and zone.

Both *cac_get_ll* and *cac_get_rc* return the palette identifier for the color palette required to display the data. If the user requests an invalid latitude and longitude coordinate from *cac_get_ll* or an invalid row and column from *cac_get_rc* the function will return an error status.

Low-Level Access Routines

The low-level CAC access routines, which are contained in the file `cac_llev.c`, consist of 23 entry points. These low-level routines are used by the high-level software presented in the previous section. In addition, programmers that need to manipulate CAC data for more advanced applications will utilize the low-level access routines. Appendix B documents these routines in detail. They are listed here in alphabetical order, since the calling sequence may vary between applications. All floating point numbers are "double" in C.

- *cac_free*
  Frees memory that was allocated by the low-level access routines. This function is called by the high-level routine *cac_init* to free memory associated with the buffering of segment data.
- *decode_key*
  Decodes a TS keyname into its row and column components. A TS keyname is an encoded form of the TS row and column and is used to generate the filename for the TS segment of interest (see Lohrenz, et al., 1993, for more information about TS keynames and filenames). The inverse function is *encode_key*.
- *decompress_segment*
  Reads the compressed CAC segment and its codebook, then decompresses the segment.
- *double_to_si*
  Converts a floating-point number to a scaled integer. (Note: this function reduces the precision of the data). The inverse function is *si_to_double*.
- *encode_key*
  Encodes a segment row number and column number into a keyname. The inverse function is *decode_key*.
- *eq2pol*
  Converts equatorial zone latitude and longitude coordinates into polar zone latitude and longitude coordinates (see Lohrenz, et al., 1993, for more information about TS polar and non-polar coordinate systems). The inverse function is *pol2eq*.
- *get_decompressed_pixel*
  Retrieves the specified pixel from a compressed segment without decompressing the entire segment. This is done by computing the location of the compressed data byte in a two-dimensional array of compressed data bytes, given the pixel's x and y coordinates.

- *get_segment_name*
  Builds the CAC compressed segment path name from the current palette area directory name, row and column components, and TS zone of the requested segment.
- *latlon_calc*
  Converts a segment's row and column values to a latitude and longitude coordinate. The TS zone specification is required for the correct handling of overlap areas. The inverse function is *rc_calc*.
- *load_legend_data*
  Reads the specified CAC legend data's header, palette, and image files. Also returns a pointer to the beginning of the legend image, along with the red, green, and blue (RGB) buffers and the size in rows and columns of the legend image itself.
- *pol2eq*
  Converts polar zone latitude and longitude coordinates to equatorial zone latitude and longitude coordinates. The inverse function is *eq2pol*.
- *rc_calc*
  Converts a latitude and longitude coordinate to a segment's row and column values. The zone specification is required for the correct handling of overlap areas. The inverse function is *latlon_calc*.
- *read_cd_covrg*
  Reads the `cd_covrg.dat` file from the ID directory on the CAC CDROM. The `cd_covrg.dat` file contains the approximate rectangular coverages for each palette area (PA) on the CAC CDROM and the PA's associated TS zone number. (Exception: zone numbers were not included in the `cd_covrg.dat` file for early CAC CDROMs. The first CAC CDROM that did include zone numbers in its `cd_covrg.dat` file was MDFF library #CD-1991-A-MAP3-10004. The structure "no_zone_cacs," in `cac_llev.c`, contains the PA and zone number associations for all CACs produced prior to CD-1991-A-MAP3-10004). The `cd_id.dat` must be read first (see *read_cd_id*) to correctly process the `cd_covrg.dat` file.
- *read_cd_id*
  Reads the `cd_id.dat` file from the ID directory on the CAC CDROM.
- *read_compressed_segment*
  Reads the compressed segment and its codebook, and buffers them into an array of segments. The number of segments that can be buffered is controlled by the argument passed to the high-level routine *cac_init*. Buffering the segments reduces the overhead involved in re-reading an often-used segment.
- *read_palette*
  Reads and returns the "day", "night" or "mono" color palette for the retrieved CAC segment data.
- *read_entire_palette*
  Reads the entire CAC color palette, including the day, night, and mono components. This routine is used when an application requires the entire palette (i.e., during a copy operation).

- *read_pa_coverage*
  Reads the scaled integer latitude and longitude coordinates from the current PA's `coverage.dat` file, and returns them as floating-point numbers.
- *remap_palette*
  Remaps a CAC color palette (240 entries) to an algebraic palette (216 entries) to allow the CAC data to be displayed without color flicker. The color flicker is caused by an application using put the entire systems color palette. The result returned is an array of indices that point to the algebraic color that is closest to the CAC palette color specified.
- *si_convert*
  Converts a latitude or longitude value from an ASCII string to a scaled integer. The format of the string is **s**`DDDMMSS.SS` where:

  s = sign of the latitude or longitude (+ or –; must be present in the string);
  `DDD` = degree portion (000 - 090 for latitude, or 000 - 180 for longitude);
  `MM` = minutes portion (00 - 59);
  `SS.SS` = seconds portion (00.00 - 59.99).
- *si_to_double*
  Converts a scaled integer to a floating-point number. The inverse function is *double_to_si*.
- *spdec*
  Decompresses a CAC compressed segment. Due to the peculiarities of MSDOS, this routine has two different modes (one for MSDOS, and one for VMS and Unix).

Miscellaneous Access Routines

The miscellaneous CAC access routines, which are contained in the file `cac_misc.c`, consist of five entry points. These routines are used to access the audit trail data on a CAC CDROM. The audit trail provides a path back to the original paper charts used to create the ADRG CDROM. For more information about specific ADRG files referenced in this section, refer to NIMA (1989). Appendix C documents these routines in more detail.

- *read_areadrc*
  Reads the specified `areadrc.dat` file. This file contains a list of the CAC CDROM path names of the Distribution Rectangle (DR) files for each ADRG source CDROM in a particular scale and zone. This list can be used to locate the ADRG DR information for a particular area of the CAC CDROM.
- *read_areasorc*
  Reads the specified `areasorc.dat` file. This file contains a list of the CAC CDROM path names of the Source Graphic files for each DR from a source ADRG CDROM in a particular scale and zone. This list can be used to locate the ADRG source information for a particular area of the CAC CDROM.
- *read_cdheader*
  Reads the specified CD `header.dat` file. This file contains various information about a specific ADRG source CDROM.

- *read_drheader*
  Reads the specified DR `header.dat` file. This file contains information about a specific DR for an ADRG source CDROM.
- *read_sgheader*
  Reads the specified `sgghed.dat` file. The *gg* in the filename is the source graphics number (01 - 99). The Source Graphics file contains information about the original paper chart that was scanned into the ADRG CDROM.

Acknowledgements

This work was funded by the National Imagery and Mapping Agency (NIMA). The authors thank the program managers at NIMA (Richard Glass and Pat Corkery) for supporting this project. We also thank our fellow MDFF team members at NRLSSC for their hard work and dedication to the CAC Processing System and the original CAC library: Marlin Gendron, Michelle Mehaffey, Stephanie Myrick, and Michael Trenchard.

References

Lohrenz, Maura C., M.E. Trenchard, S.A. Myrick, P.B. Wischow, L.M. Riedlinger (1993). The Navy Tessellated Spheroid Map Projection System: A Comprehensive Definition. NRL/FR/7441—92-9408. Naval Research Laboratory, Stennis Space Center, MS. August.

Lohrenz Maura C., J.E. Ryan (1990). The Navy Standard Compressed Aeronautical Chart Database. NOARL Report 8. Naval Research Laboratory, Stennis Space Center, MS. July.

National Imagery and Mapping Agency (1989). Product Specifications for ARC Digitized Raster Graphics (ADRG), 1st edition. DMA Report PS/2DF/100, April.

Appendix A: Entry Point Descriptions for High-Level Access Routines

Note: `byte` is `typedefed` as `unsigned char`.

cac_init

short ***cac_init*** (char *cac_device [ ],
int *num_buffers*)

cac_device: Name of device that CAC CDROM is loaded on.
(char[ ], passed)

num_buffers: Number of segments that can be buffered at a time.
(int, passed)

Returns: 1: Normal.
-1: Error reading CD_ID.DAT.
-2: Error reading CD_COVRG.DAT.
-3: Error: CDROM is not a valid CAC CDROM.

---

cac_inq_palette

short ***cac_inq_palette*** (char *type*,
short *palid*,
short **size*,
byte *red [ ]*,
byte *green [ ]*,
byte *blue [ ]*)

type: Type of palette to load (DAY, NIGHT, or MONO)
(char, passed)

palid: Palette identification. This is a four digit number identifying the color palette to use for the selected segment.
(int, passed)

size: Size of the color palette returned.
(short *, returned)

red: Array of size *size* containing the RED component of the color palette.
(byte [ ], returned)

green: Array of size *size* containing the GREEN component of the color palette.
(byte [ ], returned)

blue: Array of size *size* containing the BLUE component of the color palette.
(byte [ ], returned)

Returns: 1: Normal.
-1: Error opening PALETTE.DAT file.
-2: Error reading PALETTE.DAT file.

---

cac_get_ll

short ***cac_get_ll*** (double *lon*,
double *lat*,
short **palid*,
short **color*)

lon: Longitude of requested pixel.
(double, passed)

lat: Latitude of requested pixel.
(double, passed)

palid: Palette identification of pixel at the specified *lat* and *lon*.
(short *, returned)

color: Pixel value at specified *lat* and *lon*. This is the index into the color palette.
(short *, returned)

Returns: 1: Normal
-1: Error: specified (*lat, lon*) point does not fall within bounds specified by CD_COVRG.DAT file. I.e., the specified data is not on the CDROM.

---

cac_get_rc

short ***cac_get_rc*** (long *row*,
long *col*,
short *map_zone*,
short **palid*)

row: Row of requested segment.
(long, passed)

col: Column of requested segment.
(long, passed)

map_zone: TS map zone that the requested segment is in. This is used to allow specifying segments in zone overlap areas.
(short , passed)

palid: Palette identification of segment at row/col.
(short *, returned)

Returns: 1: Normal
-1: Error: specified map zone is not on this CDROM.
-2: Error: specified segment at row/col is not on this CDROM.

Appendix B: Entry Point Descriptions for Low-Level Access Routines

cac_free

void ***cac_free*** (void)

Returns: None.

---

decode_key

void ***decode_key*** (char *keyname [ ],
long **row*,
long **col*)

keyname: Keyname to decode.
(char [ ], passed)

row: Row number to encode.
(long *, returned)

col: Column number to encode.
(long *, returned)

Returns: None.

---

decompress_segment

short ***decompress_segment*** (char *pa_path [ ],
unsigned char **decomp_seg*)

pa_path: Complete file specification of CAC segment file to decompress.
(char [ ], passed)

decomp_seg: Pointer to array containing the decompressed segment data.
(unsigned char *, returned)

Returns: 1: Normal
-1: Error reading the compressed segment file.

---

double_to_si

long ***double_to_si*** (double *value*)

value: Double precision number to convert to a Scaled integer.
(double, passed)

Returns: Encoded scaled integer as a signed long.

---

encode_key

void ***encode_key*** (long **row*,
long **col*,
char *keyname [ ]*)

row: Row number to encode.
(long *, passed)

col: Column number to encode.
(long *, passed)

keyname: Resultant encoded keyname.
(char [ ], returned)

Returns: None.

---

eq2pol

void ***eq2pol*** (double **latin*,
double **longin*,
double **latout*,
double **longout*,
short **zone*)

latin: Equatorial latitude to convert.
(double *, passed)

longin: Equatorial longitude to convert.
(double *, passed)

latout: Polar latitude.
(double *, returned)

longout: Polar longitude.
(double *, returned)

zone: Polar zone to use in the conversion.
(short *, passed)

Returns: None.

---

get_decompressed_pixel

short ***get_decompressed_pixel*** (short *y*,
short *x*)

y: Y coordinate of a pixel in the compressed segment.
(short , passed)

x: X coordinate of a pixel in the compressed segment.
(short, passed)

Returns: Short integer corresponding to the X and Y coordinates of the requested pixel.

---

get_segment_name

void ***get_segment_name*** (char *pa_path [ ]*,
long *row*,
long *col*,
short *zone*,
char *seg_path [ ]*)

pa_path: Path to the palette area subdirectory.
(char [ ], passed)
Note the "." character at the end of VMS path names, and the "/ " or " \" in Unix or MS-DOS filenames. The following are sample paths to the same palette area subdirectory on VMS, Unix, and MS-DOS systems:

VMS: **CDROM:[MAP3]PA012901.**
Unix: **/cdrom/map3/pa012901/**
MSDOS: **D:\map3\pa012901**

row: Row number of segment to decompress.
(long, passed)

col: Column number of segment to decompress.
(long, passed)

zone: Tesselated Sphere zone number cooresponding to *pa_path.*
(short, passed)

seg_path: Complete path specification for requested segment.
(char [ ], returned)
E.g., VMS: **CDROM:[MAP3.PA012901.R000015]12345678.214**
Unix: **/cdrom/map3/pa012901/r000015/12345678.214**
MSDOS: **D:\map3\pa012901\r000015\12345678.214**

Returns: None.

---

latlon_calc

```
void latlon_calc (short  *zone,
                  short  *scale,
                  long   *row,
                  long   *col,
                  double *lat,
                  double *lon)
```

zone: Zone to use in the conversion to latitude/longitude.
(short *, passed)

scale: Scale to use in the conversion to latitude/longitude.
(short *, passed)

row: Tesselated sphere row number to convert.
(long *,passed)

col: Tesselated sphere column number to convert.
(long *,passed)

lat: Latitude based on scale, zone, row and column.
(double *, returned)

lon: Longitude based on scale, zone, row and column.
(double *, returned)

Returns: None.

---

load_legend_data

void ***load_legend_data*** (char *legend_path [ ]*,
unsigned char ***legend_ptr*,
unsigned char *rbuf [ ]*,
unsigned char *gbuf [ ]*,
unsigned char *bbuf [ ]*,
unsigned long ****legend_x*,
unsigned long ****legend_y*)

legend_path: File specification of the directory containing the legend data.
(char [ ], passed)

legend_ptr: Pointer to the beginning of the array containing the legend image data.
(unsigned char **, returned)

rbuf: Red component of the legend image's palette.
(unsigned char [ ], returned)

gbuf: Blue component of the legend image's palette.
(unsigned char [ ], returned)

bbuf: Green component of the legend image's palette.
(unsigned char [ ], returned)

legend_x: Size of the legend image in the "x" direction (columns).
(unsigned long *, returned)

legend_y: Size of the legend image in the "y" direction (rows).
(unsigned long *, returned)

Returns: 1: Normal
-1: Error opening legend header file.
-2: Error reading legend header file.
-3: Error opening legend image file.
-4: Error reading legend image file.

---

pol2eq

void ***pol2eq*** (double **latin*,
double **longin*,
double **latout*,
double **longout*)

latin: Polar latitude to convert.
(double *, passed)

longin: Polar longitude to convert.
(double *, passed)

latout: Equatorial latitude.
(double *, returned)

longout: Equatorial longitude.
(double *, returned)

Returns: None.

---

rc_calc

void ***rc_calc*** (double **lat*,
double **lon*,
short **scale*,
short **zone*,
long **row*,
long **col*)

lat: Latitude to convert.
(double *, passed)

lon: Longitude to convert.
(double *, passed)

scale: Scale to use in the conversion to row/column.
(short *, passed)

zone: Zone to use in the conversion to row/column.
(short *, passed)

col: Tesselated sphere column number based on specified scale and zone.
(long *,returned)

row: Tesselated sphere row number based on specified scale and zone.
(long *,returned)

Returns: None

---

read_cd_covrg

short ***read_cd_covrg*** (char *path[ ]*,
char *pa_nums[MAX_PAS][8]*,
short **num_pas*,
double *pa_latlon[MAX_PAS][4]*,
char *pa_zones[MAX_PAS]*)

path: Complete path specification to the CDROM's CD_COVRG.DAT file.
(char [ ], passed)

pa_nums: Two-dimensional array of palette area names from the CD_COVRG.DAT file. Each palette area name is eight bytes. The maximum number of possible palette areas on one CDROM is MAX_PAS (see CAC_INC.H).
(char [ ][8], returned)

num_pas: The number of palette areas on the CDROM.
(short *, returned)

pa_latlon: Two dimensional array of approximate coverages of each palette area on the CDROM. The order of the latitude/longitude data in the array is as follows:
[*][0] = West longitude
[*][1] = East longitude
[*][2] = South latitude
[*][3] = North latitude
(double [ ][4], returned)

pa_zones: Array of TS zone numbers corresponding to *pa_nums* above.
(char [ ], returned)

Returns: 1: Normal
-1: Error opening CD_COVRG.DAT file.
-2: Error reading the number of palette areas from CD_COVRG.DAT file.
-3: Error reading a palette area name from CD_COVRG.DAT file.
-4: Error reading a palette area lat/lon set from CD_COVRG.DAT file.

---

read_cd_id

short ***read_cd_id*** (char *path[ ]*,
char *data[ ]*)

path: Complete path specification to the CDROMs CD_ID.DAT file.
(char [ ], passed)

data: Contents of specified CD_ID.DAT file.
(char [ ], returned, requires twenty bytes)

Returns: 1: Normal
-1: Error opening CD_ID.DAT file.
-2: Error reading CD_ID.DAT file .

---

read_compressed_segment

short ***read_compressed_segment*** (char *pa_path[ ],*
unsigned char **codebook,*
unsigned char **compseg)*

pa_path: Complete file specification of CAC segment file of interest.
(char [ ], passed)

codebook: Codebook to decompress segment (codebook requires 1024 bytes of memory).
(unsigned char *, returned)

compseg: Compressed segment to be decompressed (compressed segment requires 16384 bytes of memory).
(unsigned char *, returned)

Returns: 1: Normal
-1: Error opening compressed CAC segment file.
-2: Error reading compressed CAC segment codebook.
-3: Error reading compressed CAC segment data.

---

read_pa_coverage

short ***read_pa_coverage*** (char *name[ ],*
double **minlon,*
double **maxlon,*
double **minlat,*
double **maxlat)*

name: PA coverage filename (full path).
(char [ ], passed)
E.g., VMS: **CDROM:[MAP3.PA012901]COVERAGE.DAT**
Unix: **/cdrom/map3/pa012901/coverage.dat**
MSDOS: **D:\map3\pa012901\coverage.dat**

minlon: Minimum longitude coordinate.
(double *, returned)

maxlon: Maximum longitude coordinate.
(double *, returned)

minlat: Minimum latitude coordinate.
(double *, returned)

maxlat: Maximum latitude coordinate.
(double *, returned)

Returns: 1: Normal
-1: Error opening PA COVERAGE.DAT file.
-2: Error reading PA COVERAGE.DAT file .

---

read_palette

short ***read_palette*** (char *type*,
char *path[ ]*,
unsigned char *red[ ]*,
unsigned char *green[ ]*,
unsigned char *blue[ ]*)

type: Color palette type (day, night, mono).
(char, passed)

path: Complete file specification of CAC color palette.
(char [ ], passed)

red: Red component of color palette (requires at least 256 bytes).
(unsigned char *, returned)

green: Green component of color palette (requires at least 256 bytes).
(unsigned char *, returned)

blue: Blue component of color palette (requires at least 256 bytes).
(unsigned char *, returned)

Returns: 1: Normal
-1: Error opening CAC color palette.
-2: Error reading CAC color palette.

---

remap_palette

unsigned char ***remap_palette** (unsigned char *red[ ]*,
unsigned char *green[ ]*,
unsigned char *blue[ ]*)

red: Red component of color palette to remap.
(unsigned char *, returned)

green: Green component of color palette to remap.
(unsigned char *, returned)

blue: Blue component of color palette to remap.
(unsigned char *, returned)

Returns: Pointer to an array containing indices that represent the mapping of the specified CAC color map entries to the nearest color in an algebraic color map.

---

si_convert

long ***si_convert*** (char value[ ],
short type)

value: Character string of latitude or longitude to convert to a scaled integer number. The sign of the latitude and longitude value (i.e., "+" or "-") must be present. (char [ ], passed)

type: Denotes whether *value* is a longitude or latitude (0 = longitude, 1 = latitude). (enum{longitude,latitude}, passed)

Returns: The scaled integer of *value* is returned as a signed long.

---

si_to_double

double ***si_to_double*** (long *si*)

si: Scaled integer to be converted to a double. (long, passed)

Returns: Double equivalent of decoded scaled integer.

VMS and Unix usage:

void ***spdec*** (unsigned char *inptr[16384]*,
unsigned char *spcbptr[16384]*,
unsigned char *outptr[65536]*)

inptr: Compressed segment (assumed to be 16384 bytes) to be decompressed.
(unsigned char [ ], passed)

spcbptr: Codebook (assumed to be 1024 bytes) to decompress the segment.
(unsigned char [ ], passed)

outptr: Decompressed segment (requires 65536 bytes).
(unsigned char [ ], returned)

Returns: None.

MSDOS usage:

void ***spdec*** (unsigned char far *inptr[16384]*,
unsigned char far *spcbptr[16384]*,
unsigned char far *outptr[65536]*)

inptr: Compressed segment (assumed to be 16384 bytes) to be decompressed.
(unsigned char far [ ], passed)

spcbptr: Codebook (assumed to be 1024 bytes) to decompress the segment.
(unsigned char far [ ], passed)

outptr: Decompressed segment (requires 65536 bytes).
(unsigned char far [ ], returned)

Returns: None.

Appendix C: Entry Point Descriptions for Miscellaneous Access Routines

read_areadrc

Must include file `areadrc.h`.

short ***read_areadrc*** (char *path[ ]*,
struct areadrc **areadrc*,
short **numpas*)

path: Complete file specification of the AREADRC.DAT file.
(char [ ], passed)

areadrc: Structure to contain data read from AREADRC.DAT file.
(struct areadrc *, returned)

numpas: Number of PA areas (zones) in the AREADRC.DAT file.
(short *, returned)

Returns: 1: Normal
-1: Error opening AREADRC.DAT file.
-2: Error reading AREADRC.DAT file.

---

read_areasorc

Must include file `areasorc.h`.

short ***read_areasorc*** (char *path[ ]*,
struct areasorc **areasorc*,
short **numpas*)

path: Complete file specification of the AREASORC.DAT file.
(char [ ], passed)

areasorc: Structure to contain data read from AREASORC.DAT file.
(struct areasorc *, returned)

numpas: Number of PA areas (zones) in the AREASORC.DAT file.
(short *, returned)

Returns: 1: Normal
-1: Error opening AREASORC.DAT file.
-2: Error reading AREASORC.DAT file.

---

read_cdheader (Must include file `cd_header.h`).

short ***read_cdheader*** (char *path[ ]*,
struct cdheader **cdheader*,
short **numpas*)

path: Complete file specification for the CD HEADER.DAT file.
(char [ ], passed)

cdheader: Structure to contain data read from CD HEADER.DAT file.
(struct cdheader *, returned)

Returns: 1: Normal
-1: Error opening CD HEADER.DAT file.
-2: Error reading CD HEADER.DAT file.

---

read_drheader (Must include file `dr_header.h`).

short ***read_drheader*** (char *path[ ]*,
struct drheader **drheader*,
short **numpas*)

path: Complete file specification of the DR HEADER.DAT file.
(char [ ], passed)

drheader: Structure to contain data read from DR HEADER.DAT file.
(struct drheader *, returned)

Returns: 1: Normal
-1: Error opening DR HEADER.DAT file.
2: Error reading DR HEADER.DAT file.

---

read_sgheader (Must include file `sg_header.h`).

short ***read_sgheader*** (char *path[ ]*,
struct sgheader **sgheader*,
short **numpas*)

path: Complete file specification of the SG HEADER.DAT file.
(char [ ], passed)

sgheader: Structure to contain data read from SG HEADER.DAT file.
(struct sgheader *, returned)

Returns: 1: Normal
-1: Error opening SG HEADER.DAT file.
-2: Error reading SG HEADER.DAT file .

Appendix D: High Level Function Calling Order

main_ll.c

```
#include "cac_inc.h"
#include "m4_const.h"

int main (unsigned int argc, char *argv[ ])

{
        short size, i;
        int status;
        double lat,lon;
        short color;
        short palid, prev_palid=0;
        char  debug=0;
        static unsigned char decomp_seg[256][256]; /* Decompressed CAC segment */
        unsigned char red[256],                    /* Selected color palette */
                      green[256],
                      blue[256];

/* argv[1]: CDROM device name */
/* argv[2]: Number of segments to buffer */
        cac_init (argv[1], atoi(argv[2]));

        while (TRUE)
        {
                printf("Lat,Lon (separated by a comma): ");
                scanf ("%lf,%lf", &lat,&lon);
                status = cac_get_ll (lon, lat, &palid, &color);
                if (status == 1)
                {
                        printf ("Lat: %6.2lf Lon: %7.2lf\n", lat, lon);
                        printf ("Row: %6d Col: %6d\n", cac.row, cac.col);
                        printf (" Palette ID: %d\n", palid);
                        printf ("Pixel color: %d\n", color);
                        if (prev_palid != palid)
                        {
                                status = cac_inq_palette (day, palid, &size, red, green, blue);
                                printf ("Palette loaded for PA#: %d\n",palid);
                                prev_palid = palid;
```

```
                        if (debug)
                                for (i=0; i<size; i++)
                                        printf("%x  %x  %x\n", red[i],green[i],blue[i]);
                }
        }
        else
                printf ("Position NOT found on CAC...\n");

    } /* End "while (TRUE)" */

}
```

main_rc.c

```
#include "cac_inc.h"
#include "m4_const.h"

int main (unsigned int argc, char *argv[ ])

{
        short size, i;
        unsigned char  red[256],                /* Selected color palette */
                       green[256],
                       blue[256];
        int status;
        long  row,col;
        short color, map_zone;
        short palid, prev_palid=0;
        char debug=0;
        static unsigned char decomp_seg[256][256]; /* Decompressed CAC segment */

/* argv[1]: CDROM device name */
/* argv[2]: Number of segments to buffer */
        cac_init (argv[1], atoi(argv[2]));

        while (TRUE)
        {
                printf("Row,Column,Zone (separated by commas): ");
                scanf ("%ld,%ld,%d", &row, &col, &map_zone);
                status = cac_get_rc (row, col, map_zone, &palid, (unsigned char *)decomp_seg);
                if (status)
                {
                        printf ( "Row: %6ld Col: %6ld\n", row, col);
                        printf (" Palette ID: %d\n", palid);
                        printf (" cac.row: %6ld cac.col: %6ld\n", cac.row, cac.col);
                        if (prev_palid != palid)
                        {
                                status = cac_inq_palette (day, palid, &size, red, green, blue);
                                printf ("Palette loaded for PA#: %d\n",palid);
                                prev_palid = palid;
```

```
                        if (debug)
                                for (i=0;i<size;i++)
                                        printf ("%x  %x  %x\n",red[i],green[i],blue[i]);
                        }
                }
                else
                        printf ("Position NOT found on CAC...\n");

        } /* End "while (TRUE)" */
}
```