### Initialization

CS-MAP needs to be initialized. Initialization consists of providing CS-MAP with the directory in which the dictionary files reside. This is accomplished by calling the CS\_altdr function. This function takes a single argument, a character string which is the path to the appropriate directory. Calling CS\_altdr with a NULL pointer as an argument will cause the value of the environmental variable named CS\_MAP\_DIR to be used as the data directory. CS\_altdr returns and integer zero if the initialization was successful, -1 if not.

This was not a requirement in the past, thus the rather strange name for this initialization function.

**Important Note 1**: Whenever CS-MAP needs to go to disk to find something, that something often needs to reside in the directory specified by this function call.

**Important Note 2**: Failing to call this function successfully prior to using CS-MAP almost always results in a memory addressing fault.

## High Level Interface

Functions are provided which can convert a coordinate from one coordinate system to another with a single function call. This set of functions was originally developed specifically for the application programmer who is coding in BASIC, FORTRAN, APL, or other language (other than C or Pascal) which can make simple function calls. It does not use structure pointers of any sort. Since the affect on performance is small (about a 20% reduction), it is now the recommended interface for most applications.

Most of the functions described in this section use CSbcclu and CSbdclu to cache coordinate system and datum conversion definitions. Therefore the performance penalty of these functions is reduced to a search of a linked list for the coordinate system names involved. These cache functions are smart enough to keep the most recently accessed items at the front of the list to further minimize the performance penalty.

The information presented in this section is intended only to associate a function name with a specific capability. Refer to Chapter 4 of the CS-MAP documentation for detailed information and prototypes for all functions referred to in this section.

**Basic Coordinate Conversion -- CS\_cnvrt**

Given a coordinate as an array of three doubles, and the names of two coordinate systems as two character arrays, CS\_cnvrt converts the coordinate from one system to another. It's that simple. Where cartesian coordinates are provided and returned, the X coordinate is the first element of the array, the Y coordinate is the second, and the Z is the third element of the array. Where geographic coordinates are provided, the first element in the array must contain the longitude, the second the latitude**,** and the third element must contain the height. In either case, the manner in which the values are interpreted depends upon the coordinate systems involved. For example, if the source coordinate system definition specifies the unit to be meters, the X, Y, and Z coordinates are considered to be in meters. Similarly, if the target coordinate system is defined as a latitude and longitude system with an angular unit of grads, the returned latitude and longitude coordinates will be in units of grads.

The status value returned by CS\_cnvrt informs the calling application of the validity of the results. A zero return value indicates that the requested conversion was completed without complication and the results now occupy the coordinate array. A negative status return value indicates a hard error occurred and that the contents of the coordinate array remain unchanged. A positive, non-zero return status indicates that the conversion was performed, but an abnormality was encountered during the conversion. In this case, the results returned in the coordinate array may not be exactly what the user expects.

In all cases of a negative status return, the values in the provided coordinate array will remain unchanged. Taking the absolute value of the returned status value will often produce the CS-MAP error code for the specific condition causing the hard error. The numeric error code which defines the specific cause of the problem will also be stored in the cs\_Error global variable, and a textual description of the error condition can be obtained by calling the CS\_errmsg function before calling any other CS-MAP function. Typically, when applications detect a negative status return, the application informs the user using the textual description obtained from CS\_errmsg and terminates the current operation.

CS\_cnvrtreturns a positive non-zero status value whenever it encounters something suspicious, but not something that precludes a conversion. Positive non-zero return values are usually caused by coordinate systems and coordinates which are incompatible, or specific values which are singularity points for the projection(s) involved. A common cause of a positive non-zero return value is the conversion of a point at either pole. CS-MAP will return a positive non-zero value in these cases as longitude is undefined at the poles, and reversing the calculation is unlikely to reproduce the initial value. Another common cause of a positive non-zero status return is providing, say, UTM coordinates when the source coordinate system is given as "LL". UTM coordinates, usually, will not be in the normal range of geographic coordinates and CS-MAP will consider this to be suspicious. A positive return value will also be returned if, for example, it is requested to convert a geographic coordinate in Europe from NAD27 to NAD83.

When a positive non-zero return value from CS\_cnvrt is encountered, the typical application issues a warning message to the user and continues. These abnormal, but not necessarily fatal, conditions are often the result the user desires. It should be left the user to decide. For performance reasons, CS-MAP does not automatically generate a textual message for these conditions. However, application programs can analyze the returned status value in order to present a more specific warning message to the end user.

## High Performance Interface

The High Performance Interface to the Coordinate System Mapping Package consists of thirteen functions. By virtue of the data structures described above, use of these functions is independent of the actual coordinate systems, projections, or datums in use. This represents the most efficient means to use CS-MAP to convert coordinates from one coordinate system to another. It also insulates your applications from most changes which could be made to the CS-MAP in the future. This basic API has not changed since 1992. This interface requires the use of structure pointers and, therefore, may not be appropriate for use with some languages. Therefore, use this interface wherever high performance is a top priority and the application is written in a language which can handle pointers such as C, C++, or Pascal.

These functions make use of the Coordinate System Dictionary, the Datum Dictionary, the Ellipsoid Dictionary, and the functions which access them. This need not be of concern to the application programmer using the High Performance Interface as it all goes on "behind the scenes".

In this chapter, our intent is to associate function names with capabilities and features. Refer to Chapter 4 for full details and prototypes of the functions introduced here.

### Coordinate System to Coordinate System

In order to convert from one coordinate system to another, one simply obtains, from the CS\_csloc function, a definition of the two coordinate systems of concern. The inverse function, CS\_cs2ll, is used to convert the source coordinates to latitude and longitude and the forward function, CS\_ll2cs, is used to convert to the target coordinate system. The sample code segment shown is, for example, all the code necessary to convert a file of NAD27 based UTM Zone 13 (**UTM27-13**) coordinates to NAD27 based Colorado State Plane, Southern Zone (**CO-S**). To change the conversion to use other coordinate systems, only the names provided to the CS\_csloc function need be changed. Of course, these strings are rarely hard coded as has been done in this example.

int input, output;

double xy [2], ll [2];

struct cs\_Csprm\_ \*utm, \*co\_s;

utm = CS\_csloc ("UTM27-13");

co\_s = CS\_csloc ("CO-S");

while (read (input,xy,sizeof (xy)) != 0)

{

CS\_cs2ll (utm,ll,xy);

CS\_ll2cs (co\_s,xy,ll);

write (output,xy,sizeof (xy));

}

CS\_free (utm);

CS\_free (co\_s);

### The LL Coordinate System

Many products, such as our own Tralaine, will use the above scheme to provide the ability to convert from any coordinate system to any another. This scheme is completely general, supporting any combination of coordinate systems. Sometimes, however, it is desirable to convert from or to geographic coordinates. The **LL** coordinate system and the Unity projection accommodate this within the general scheme of things described above. That is, the **LL** coordinate system is simply a coordinate system in which the coordinates are latitudes and longitudes, and the Unity projection is simply a set of conversion functions which do little other than possible units conversion.

Therefore, supplying a coordinate system name of **LL**, for example, for either the input or output coordinate system will produce the desired results without the application program having to know about this specific situation. (Please note that **LL** is a cartographically referenced coordinate system. Coordinate systems such as **LL27** and **LL83** are usually used in practice.)

Latitude and longitude coordinates in different units or referenced to a prime meridian other than Greenwich are possible by defining different **LL** type coordinate systems. These definitions, all based on the Unity pseudo-projection, can include a units specification and a specification of a prime meridian other than zero (i.e. Greenwich).

### Adding Datum Conversions to the Interface

Datum conversions can be added to the basic scheme described above by adding calls to the datum conversion functions. Refer to the code given below for an example, paying special attention to the emphasized code. Once the two coordinate system definitions have been initialized, they are passed to CS\_dtcsu. By examining both the source and target coordinate system definitions, CS\_dtcsu is able to determine which, if any, datum transformation techniques need to be applied to accomplish the desired conversion. CS\_dtcsu will select one or more datum conversions as necessary to accomplish the desired conversion. For example, to convert from NAD27 to WGS72, three conversions are actually setup: 1)from NAD27 to NAD83 via the NADCON technique, 2)NAD83 to WGS84 (which is currently a null conversion), and finally 3)WGS84 to WGS72 using a hard coded formula. CS\_dtcsu assures that all preparations necessary for these conversions are initialized, and saves the results in the cs\_Dtcprm\_ structure to which it returns a pointer.

In the actual coordinate conversion loop, CS\_dtcvt is called for each coordinate once its geographic form has been obtained from CS\_cs2ll. Note that if CS\_dtcsu determined that no datum conversion was required, the information contained in the cs\_Dtcprm\_ structure which it returns causes CS\_dtcvt to simply copy the source geographic coordinates to the target array. Finally, when the conversion process is complete, CS\_dtcls is used to release any system resources which were allocated for the datum conversion and which are no longer needed.

int input, output;

double xy [2], ll [2];

struct cs\_Csprm\_ \*utm, \*co83\_s;

struct cs\_Dtcprm\_ \*dtc\_ptr;

.

.

utm = CS\_csloc ("UTM27-13");

co83\_s = CS\_csloc ("CO83-S");

dtc\_prm = CS\_dtcsu (utm,co83\_s,dat\_err,blk\_err);

while (read (input,xy,sizeof (xy)) != 0)

{

CS\_cs2ll (utm,ll,xy);

CS\_dtcvt (dtc\_prm,ll,ll);

CS\_ll2cs (co83\_s,xy,ll);

write (output,xy,sizeof (xy));

}

CS\_dtcls (dtc\_prm);

CS\_free (utm);

CS\_free (co\_s);

Sample Code Segment

### CS\_errmsg ERRor MeSsaGe

Sub CS\_errmsg (ByVal my\_bufr As String,ByVal bufr\_size As Integer)

procedure CS\_errmsg (msg\_bufr :PChar;bufr\_size :Integer);

void CS\_errmsg (char msg\_bufr,int bufr\_size);

CS\_errmsg returns to the calling function a null terminated string which describes the last error condition detected by the CS\_MAP library. The result is returned in the buffer pointed to by the msg\_bufr argument, which is assumed to be bufr\_size bytes long. The message is returned in one character per byte ANSI code characters.

CS\_errmsg will return the null string if called before any error condition is detected.

BUGS

After returning an error message to the user, CS\_errmsg should reset itself to the null string preventing the same error message from being returned a second time. It should, but is doesn't.

### CS\_altdr ALTernate DiRectory

Function CS\_altdr (ByVal new\_dir As String) As Integer

function CS\_altdr (alt\_dr :PChar):Integer;

int CS\_altdr (Const char alt\_dir);

Normally, all functions in the Coordinate System Mapping Package library expect to find data files in the C:\MAPPING directory as defined in CSdata. CS\_altdr can be used to specify an alternate directory at run time; that indicated by the alt\_dir argument. CS\_altdr returns zero if a coordinate system dictionary was indeed found in the directory provided; otherwise, it returns -1.

Calling CS\_altdr with the NULL pointer as its argument instructs CS\_altdr to use the value of the environmental variable CS\_MAP\_DIR as the location of the CS-MAP data files. Again a zero is returned if this was successful, -1 if not. (The string defining the name of the environmental variable name is defined in the **cs\_map.h** header file.)

Calling CS\_altdr with the alt\_dir argument pointing to the null string instructs CS\_altdr to use the current directory on the current drive as the location of CS-MAP data files. Again a zero is returned if this selection produces a directory which contains a Coordinate System Dictionary File. Otherwise -1 is returned.

Notice, that using the return status as a guide, several attempts at locating the CS-MAP data directory can be made in any application.

The name of the directory which is searched for all data files is maintained in a global character array cs\_Dir, which is defined in the CSdata module. The cs\_Dir array must, initially, contain a null terminated string, the last non-null character of which must be the directory separator character. The global character pointer cs\_DirP (also defined in CSdata) is expected to point to the terminating null character of the string in cs\_Dir. Under this scheme, Coordinate System Mapping Package data files are accessed as follows:

extern char cs\_Dir [];

extern char \*cs\_DirP;

.

.

strcpy (cs\_DirP,"file\_name");

fd = open (cs\_Dir,O\_MODE);

.

.

Achieving this particular setup is relatively easy using CS\_stcpy:

cs\_DirP = CS\_stcpy (cs\_Dir,"C:\\MAPPING\\");

BUGS

The purpose of this function is to insulate the library from system implementation issues. Without a function of this nature, all applications using CS-MAP would have to implement a specific directory on a specific drive. Not very pleasant. There does not appear to be a nice clean solution to this problem.

### CS\_cnvrt generalized CoNVeRT function

Function CS\_cnvrt (ByVal src\_cs As String,ByVal trg\_cs As String,

ByRef coord As Double) As Integer

function CS\_cnvrt (src\_cs,trg\_cs :PChar;var coord :double) :Integer;

int CS\_cnvrt (Const char \*src\_cs,Const char \*trg\_cs,double coord [3]);

CS\_cnvrt is in essence a High Level Interface to the CS\_MAP library. Using this single function, one can convert coordinates from any defined system to any other. Simply provide the key name of the source system via the src\_cs argument, and the key name of the destination coordinate system via the trg\_cs argument, and CS\_cnvrt will cause the coordinate in the array given by the coord argument is converted from the source system to the destination system. CS\_cnvrt returns zero if the conversion completed successfully without incident. Otherwise, a CS-MAP error code value is returned (see cs\_map.h).

CS\_cnvrt relies on a cache of coordinate systems, and for each conversion linearly searches the cache for the definitions of the two coordinate system definitions, and the datum conversion definition, it needs to perform its function. Thus, the performance penalty incurred from using this High Level Interface is not as great as one may think.

Currently, the third element of the coord argument is unused; but may be used in the future.

### CS\_cnvrt3D 3D generalized CoNVeRT function

Function CS\_cnvrt3D (ByVal src\_cs As String,ByVal dst\_cs As String,

ByRef coord As Double) As Integer

function CS\_cnvrt3D (src\_cs,dst\_cs :PChar; var coord :Double) :Integer

int CS\_cnvrt3D (Const char \*src\_cs,Const char \*dst\_cs,double coord [3]);

CS\_cnvrt3D is in essence a High Level Interface with regard to three dimensional conversions. Using this single function, one can convert three dimensional coordinates from any defined system to any other. Simply provide the key name of the source system via the src\_cs argument, and the key name of the destination coordinate system via the dst\_cs argument, and CS\_cnvrt3D will cause the coordinate in the array given by the coord argument to be converted from the source system to the destination system. CS\_cnvrt3D returns a zero if the conversion completed successfully without incident. Otherwise, a CS\_MAP error code value is returned.

CS\_cnvrt3D relies on a cache of coordinate systems, and for each conversion linearly searches the cache for the definitions of the two coordinate system definitions, and the datum conversion definition, it needs to perform its function. Thus, the performance penalty incurred from using this High Level Interface is not as great as one may think.

Use CS\_cnvrt3D only when converting data maintained in a three dimensional database. Note that if the application is able to supply the returned Z value during an inverse calculation, the inverted result may not match the original values.

### CS\_recvr RECoVeR resources

Sub CS\_recvr

procedure CS\_recvr;

void CS\_rcvr (void);

CS\_rcvr will release all system resources allocated by use of the single function user interface functions CS\_cnvrt, CS\_cnvrg, and CS\_scale. It essentially frees up the coordinate system cache and the datum conversion cache established by these functions to enhance performance.

**CS\_csloc Coordinate System LOCate and initialize**

struct cs\_Csprm\_ \*CS\_csloc (Const char \*cs\_nam);

struct cs\_Csprm\_ \*Cscsloc1 (Const struct cs\_Csdef\_ \*cs\_ptr);

struct cs\_Csprm\_ \*Cscsloc2 (Const struct cs\_Csdef\_ \*csPtr,

Const struct cs\_Dtdef\_ \*dtPtr,

Const struct cs\_Eldef\_ \*elPtr);

struct cs\_Csprm\_ \*CScsloc (Const struct cs\_Csdef\_ \*csPtr,

Const struct cs\_Datum\_ \*dtPtr);

CS\_csloc locates the coordinate system definition indicated by cs\_nam and returns a pointer to a malloc'ed, coordinate system parameter structure initialized for the specified coordinate system. The return value is the argument required by CS\_cs2ll, CS\_ll2cs, CS\_csscl, and CS\_cscnv. When no longer needed, the memory pointed to by the returned pointer should be released using CS\_free.

CS\_csloc accesses the definition dictionaries as is necessary to accomplish its task. The alternative functions enable applications to create coordinate system parameter structures using definitions that may have been obtained from sources other than the dictionaries. For example, certain applications may store definitions in vehicles other than the dictionaries, and then desire to construct a coordinate system parameter structure from these definitions.

Note that Cscsloc1 does not need to access the coordinate system dictionary as the coordinate system definition is provided by the cs\_ptr argument. However, it will need to access the datum and ellipsoid dictionaries to resolve datum and ellipsoid references. Cscsloc2 is completely independent of all dictionaries as all three definitions must be provided. CScsloc is simply a basic function is encapsulates the basic functions of CS\_csloc and its alternatives, and thus prevents duplication of large amounts of code.

ERRORS

CS\_csloc, CScsloc1, CScsloc2, and CScsloc return a NULL pointer and set cs\_Error through the use of CS\_erpt if any of the following conditions occur:

|  |  |
| --- | --- |
| cs\_UNKWN\_PROJ | The projection specified in the coordinate system definition is unknown to the system. |

CS\_csloc uses the following functions that detect a majority of the exceptional conditions that may occur:

|  |  |
| --- | --- |
| CS\_csdef | Locates and fetches the coordinate system definition from the Coordinate System Dictionary. |
| CS\_dtloc | Locates and fetches the datum definition from the Datum Dictionary. |
| CS\_eldef | Locates and fetches the ellipsoid definition from the Ellipsoid Dictionary. |

CScsloc1 uses the following functions that detect a majority of the exceptional conditions that may occur:

|  |  |
| --- | --- |
| CS\_dtloc | Locates and fetches the datum definition from the Datum Dictionary. |
| CS\_eldef | Locates and fetches the ellipsoid definition from the Ellipsoid Dictionary. |

### CS\_cs2ll Coordinate System TO Latitude/Longitude

void CS\_cs2ll (Const struct cs\_Csprm\_ \*csprm,double ll [2],Const double xy [2]);

Given the definition of the coordinate system, csprm, such as returned by CS\_csloc, CS\_cs2ll will convert the coordinates xy to latitude and longitude, returning the results in ll. The ll and xy arguments may point to the same array.

In the array arguments, the X coordinate and the longitude occupy the first element, the Y coordinate and the latitude the second element. West longitudes and south latitudes are negative. The returned values are in degrees.

### CS\_ll2cs Latitude/Longitude TO Coordinate System

void CS\_ll2cs (Const struct cs\_Csprm\_ \*csprm,double xy [2],Const double ll [2]);

Given the definition of the coordinate system, csprm, such as returned by CS\_csloc, CS\_ll2cs will convert the latitude and longitude given by ll to X and Y coordinates, returning the results in xy. The ll and xy arguments may point to the same array.

In the arrays, the X coordinate and the longitude occupy the first element, the Y coordinate and the latitude the second element. The latitude and longitude must be given in degrees where negative values are used to indicate west longitude and south latitude.

**CS\_dtcsu DaTum Conversion Set Up**

struct cs\_Dtcprm\_ \*CS\_dtcsu (Const struct cs\_Csprm\_ \*src\_cs,

Const struct cs\_Crprm\_ \*dest\_cs,

int dat\_err,

int blk\_err);

CS\_dtcsu, CS\_dtcvt, and CS\_dtcls, are designed to provide a generic application interface for datum conversion. The objective is to enable application programmers to incorporate datum conversion capabilities into applications with a minimum of impact. Therefore, application programmers use CS\_dtcsu to set up a datum conversion and CS\_dtcvt to perform the actual conversions independently of the number or type of datum conversions that may or may not be supported. CS\_dtcls provides a means of recovering any system resources that may be allocated by the activation of a datum conversion.

Application programmers use CS\_dtcsu to initiate a datum conversion process. Src\_cs points to the coordinate system definition of the source data that is to be converted while dest\_cs points to the coordinate system definition for the results. CS\_dtcsu examines the datum references in these coordinate systems, initializes the appropriate datum shift conversion, and returns a pointer to a malloc'ed datum conversion parameter block. The returned pointer is a required argument for the CS\_dtcvt function.

As is often the case, should the source and destination coordinate systems share the same datum, the null datum conversion is activated. That is, source latitudes and longitudes are copied directly to the destination array without modification.

The dat\_err argument is used to indicate the desired disposition of certain errors that are encountered during the setup of the datum conversion. The error disposition control afforded by dat\_err applies only to errors indicating that an unsupported datum conversion was requested. System errors, such as physical I/O or insufficient memory for example, are always treated as fatal errors and a NULL pointer is returned.

The following values for dat\_err are recognized:

|  |  |
| --- | --- |
| cs\_DTCFLG\_DAT\_I | Ignore unsupported datum conversion request errors and, in the event of such an error, **silently** activate the null conversion. |
| cs\_DTCFLG\_DAT\_W | In the event of an unsupported datum conversion request error, report the condition as a warning to CS\_erpt (cs\_DTC\_DAT\_W) and activate the null conversion. In this case, the user is notified, but data processing continues. |
| cs\_DTCFLG\_DAT\_F | In the event of any error, report the condition as a fatal error to CS\_erpt (cs\_DTC\_DAT\_F) and return the NULL pointer. |

The blk\_err argument is used to indicate the desired disposition of certain errors that are encountered during the conversion of individual coordinate values. The error disposition control afforded by blk\_err applies only to errors indicating that the required data for the geographic region containing the coordinate to be converted is not available. System errors, such as physical I/O or insufficient memory for example, are always treated as fatal errors.

The following values for blk\_err are recognized:

|  |  |
| --- | --- |
| cs\_DTCFLG\_BLK\_I | Ignore datum conversion errors caused by data availability problems and silently use the null conversion for the specific coordinate that could not be converted and cause CS\_dtcvt to return a zero value. |
| cs\_DTCFLG\_BLK\_W | In the event a datum conversion fails due to data availability, report a warning through CS\_erpt (cs\_DTC\_BLK\_W), convert the coordinate using the null conversion, and cause a CS\_dtcvt to return a positive non-zero value for the specific coordinate that could not be converted. The warning message is issued for each coordinate that could not be converted. |
| cs\_DTCFLG\_BLK\_1 | In the event a datum conversion fails due to data availability, cause CS\_dtcvt to return a positive non-zero value for the specific coordinate that could not be converted. That such an error has been reported is recorded in the datum parameter block and this is used to suppress repeated reporting of the error with regard to the same block. |
| cs\_DTCFLG\_BLK\_F | Report a fatal condition through CS\_erpt (cs\_DTC\_BLK\_F), convert the coordinate using the null conversion, and cause CS\_dtcvt to return a negative non-zero value to indicate that the expected conversion did not take place. |

Special Cases

Three special cases have been coded into this function. Normally, the geographic coodinates of the source datum are converted to WGS84 values, and the resulting WGS84 values are then converted to the target datum.

There are three cases where this genberal technique proved to be unsatisfactory. In these three cases, CS\_dtcsu has been expressly coded to look at the source and target datums, and implement direct conversions where appropriate. Note, that in each case, a specific Geodetic Data Catalog file is also involved. Thus, if the required Geodetic Data Catalog file is not present, all of the special processing is disabled.

The following table defines the special cases:

|  |  |  |  |
| --- | --- | --- | --- |
| Source Datum | Target Datum | Geodetic Data Catalog | Description |
| NAD27 | ATS77 | Nad27ToAts77.gdc | Converts directly from NAD27 to ATS77 using the very special TRANSFORM algorithm. |
| ATS77 | CSRS | Ats77ToCsrs.gdc | Converts directly as direct NTv2 format files are generally available. |
| NAD27 | CSRS | Nad27ToCsrs.gdc | Converts directly as direct NTv2 format files are generally available. |

ERRORS

Should the requested datum conversion requested be unsupported, CS\_dtcsu will perform as indicated by the dat\_err argument. Should the initialization of a supported datum conversion fail due to a system error, the NULL pointer will be returned and cs\_Error set to indicate the nature of failure. Should a datum conversion for which appropriate code is present fail because a required data file is not present, the failure is treated as an unsupported datum conversion request.

### CS\_dtcvt DaTum ConVerT

int CS\_dtcvt (struct cs\_Dtcprm\_ \*dtc\_ptr,Const double src\_ll [2],double dest\_ll [2]);

CS\_dtcvt performs the datum conversion indicated by dtc\_ptr returning in the array pointed to by dest\_ll the result of converting the latitude and longitude values pointed to by src\_ll. Src\_ll and dest\_ll may point to the same array. Latitude and longitude values must be given in degrees, where negative values indicate south and west. The longitude is carried in the first element of the array and the latitude is carried in the second element. The dtc\_ptr argument is that which is returned by CS\_dtcsu.

ERRORS

Should a system error occur during the conversion (e.g. a physical I/O error or insufficient memory) CS\_dtcvt returns a negative non-zero value and sets cs\_Error to indicate the cause of the failure.

Conversion failures caused by a lack of data covering the specific coordinate to be converted are handled as indicated by the blk\_err element of the cs\_Dtcprm\_ structure pointed to by the dtc\_ptr argument. The blk\_err element is set by CS\_dtcsu to the value of its blk\_err argument prior to returning dtc\_ptr. Refer to CS\_dtcsu for a detailed description of how such errors are handled.

In all cases, the null conversion is always performed before any other processing is attempted.

EXAMPLE

This function, and its companion CS\_dtcsu have been designed such that the following sequence of code is all that is necessary to perform a complete coordinate conversion, including a datum conversion (error handling omitted):

#define XX 0

#define YY 1

struct cs\_Csprm \*src\_cs, \*dest\_cs;

struct cs\_Dtcprm\_ \*dtc\_ptr;

double src\_xy [2], ll [2], dest\_xy [2];

.

.

src\_cs = CS\_csloc (src\_name);

dest\_cs = CS\_csloc (dest\_name);

dtc\_ptr = CS\_dtcsu (src\_cs,dest\_cs,cs\_DTCFLG\_DAT\_F,cs\_DTCDLG\_BLK\_1);

.

.

while (TRUE)

{

.

.

src\_xy [XX] = ???;

src\_xy [YY] = ???;

CS\_cs2ll (src\_cs,ll,src\_xy);

CS\_dtcvt (dtc\_ptr,ll,ll);

CS\_ll2cs (dest\_cs,dest\_xy,ll);

??? = dest\_xy [XX];

??? = dest\_xy [YY];

.

.

}

CS\_free (src\_cs);

CS\_free (dest\_cs);

CS\_dtcls (dtc\_ptr);

Notice, that adding the datum conversion to a simple cartographic conversion requires only the insertion of three lines of code (error handling aside) to the simple High Performance Interface described elsewhere in this manual.

### CS\_dtcls DaTum conversion CLoSe

void CS\_dtcls (struct cs\_Dtcprm\_ \*dtc\_ptr);

Initializing a datum conversion can use file descriptors (handles) and allocate memory from the heap. Applications may need to recover these system resources for other use prior to exiting. CS\_dtcls will release all system resources allocated to the datum conversion indicated by the dtc\_ptr argument (as returned by CS\_dtcsu). This function is, essentially, the inverse of CS\_dtcsu.