User's Guide

CS-MAP User's Guide

Printed on 20 March, 2009

Contents

CHAPTER 1

Chapter 1 -- Distribution/Release Notes

This chapter contains Distribution Notes for new users of CS_MAP and Release Notes for previous users. The Release Notes which describe recent changes in the library follow the Distribution Notes which describe the distribution and how to build CS-MAP in some of the more common build environments.

Current Manual Status

AT the current time, this compilation of documentation for the Open SOurce distribution shows many ofthe tell-tale signs of a document maintained by different people using different tools over a period of twenty years. Thus you will enocunter several distracting formatting issues, and be displeased by the lack of a comprehensive index. The presentation quality of the document will improve over time.

Technical content of this document was current with release 11.11 of CS-MAP. The original Open SOurce distribution is actually deemed to be release 12. Therefore, a significant amount of writing and editing which needs to be completed to bring this document up to date. Some of the areas in which this document is out of date are:

Name Mapping

The original name mapping functions have been replaced with an entirely new scheme which is driven by an external data file for maintenance convenience.

NTF to RGF93 Datum Conversion

There now exists a Geodetic Data Catalog file which controls access to the various grid shift files in use. Particularly, NTv2 formatted grid shift files for local municipalities in France are now supported.

DHDN To ETRF89 Datum Conversion

A new Geodetic Data Catalog file is now supported to define access to the German BeTA2007.gdb grid shift file.

ED50 To ETRF89 Datum Conversion

A new Geodetic Data Catalog file is now supported to define access to the Spanish and Portuguese (and quite likely others in the future) datum grid shift files.

Geocentric Datum Transformation Technique

The Three Parameter Datum Transformation technique has been deprecated and replaced by a new technique known as Geocentric Translation.

Category Dictionary

A Category DIctionary has been added which is a more flexible version of the origina CS-MAP group concept. While a definition canonly belog to one group, a definition can belong to several categories.

Danish System 34

Due to distribution permission issues, it was determined that the polynomial coefficients for the System 34 coordinate conversions could not be open sourced. Thus, to properly incorporate Syatem 34 capabilities into your application, you will need to obtain a copy of these coefficients for yourself. Refer to the source module named *CSsys34KMS.c* for details.

CSV File Support

Implemented as a C++ object, there now exists rather substantial support for reading and writing data file in the CSV (comma separated value) format.

EPSG Support

Still a work inprogress, there now exists substantial support for accessing data provided by the EPSG Parameter Dataset. As this dataset is traditionally distributed in Microsoft Access format, this facility relies on the conversion of all EPSG data tables to .CSV format, and uses the new CSV FIle SUpport object to access them.

WKT Flavor Support

Using the new Name Mapper facility, CS-MAP's ability to handle various flavors of WKT has been imporved. There is lots yet to be done, but a non-trivial improvement in the accuracy and number of flavors supported.

So, there remains much to do to bring this documentup to the standard desired by the CS-MAP contributor team. For now, it's important to provide potential users with the basicinformation necessary to get started using CS-MAP.

The Distribution

Detailed instructions on how to obtain the distribution are available at
`http://trac.osgeo.org/csmap/wiki/HowToGetTheSourceCode`

The distribution includes many somewhat voluminous grid data files known to OSGeo as being in the public domain. There are several such grid data files which are not in the public domain and which must be obtained from the source on an individual user basis. Simple registration is all that is required in many cases, license fees are rarely required. Mostly, the issuing authority just wants to keep track of who is using the data in order to adhere to ISO quality control standards.

The Canadian National Transformation data file is, perhaps, the most widely used example of a grid data file which OSGeo is not permitted to distribute. You and your clients will need to obtain this file from the Canadian government. A license fee is no longer required, but Geomatics Canada still needs to know who is using the data file. Contact:

www.geod.nrcan.gc.ca

Since the distribution cannot include a copy of the Canadian National Transformation file, the test cases for this transformation are commented out in the provided TEST.DAT file. After obtaining the Canadian National Transformation file, you will probably want to uncomment these test cases from the test file.

Typically, a README.txt file is placed in the folder in which an undistributable grid data files would normally reside. THis text file will usually provide information as to how to obtain a copy of the data file. When a CS-MAP error message which indicates that a file is missing is encountered, check to see if there is not a README.txt file in the folder in which CS-MAP was looking for the file and examine its contents.

The following sub-topics described the directory structure of the distribution.

Include Directory

You don't need to be a genius to figure out that all header files are installed into this sub-directory. What might surprise you is that there is only one real header file: *cs_map.h*. While this file is quite large, the precompiled header feature of most modern compilers make this approach most convenient.

Also, you never have to guess in which header file a specific item is defined in. They're all defined in *cs_map.h*. Neither do you have to wonder which files must be included into your application; *cs_map.h* is the one. There are other include files, but are those required by the rather strange environment used for MFC development.

The *cs_map.h* header file will specifically include two files. The files, and exactly where they are included, are described below. These provide a means by which users can incorporate their own features without having to modify *cs_map.h* after each new release.

cs_clientBeg.h -- The inclusion of this file occurs immediately after the check for a previous include of *cs_map.h*, but before the cs_map.h file does anything else. An excellent place to place defines which control the environment of the compilation.

cs_clientEnd.h -- The inclusion of this file occurs immediately before the #endif which terminates the multiple inclusion protection. That is, it is included after everything else in *cs_map.h*.

Library Source Code

The directory named *Source* will contain all source code to the library proper. The source code components of CS-MAP are normally compiled and the resulting objects used to construct an object module library. Source code to dictionary compilers and test programs are provided elsewhere. Make files for building the library are provided in this folder.

- *Library.mak* can used used to build the object library in the Linux environment.
- *Library.nmk* can use used int the Microsoft WIndows environment using the *nmake* facility.

Each of the make files includes a list of all the modules that belong in the library. This list represents most of the drudgery of creating a make file. Adjust the actual rules as necessary for your platform/compiler. Notice that leaving the manifest constant **__MFC__** undefined will cause all MFC related code to be skipped during the compilation process. Obviously, if compiling in an environment other than Windows 32/64, be sure to leave the **__MFC__** constant undefined.

Dictionary Source Code and Data

The directory named *Dictionaries* will contain the source code to the dictionary compiler, and the data files which this compiler compiles to produce the binary form of the Category Dictionary, the Coordinate System Dictionary, the Datum Dictionary, the Ellipsoid Dictionary, and the Multiple Regression Transformation data files. Be sure to compile these dictionaries with the **/t** option if you intend to run the CS-MAP test program. Refer to Chapter 3 of this manual for more information on the dictionary compiler program.

Make files for building the compiler are provided in this folder.

- *Compiler.mak* can used used to build the compiler in the Linux environment.
- *Compiler.nmk* can use used to build the compiler in the Microsoft WIndows environment using the *nmake* facility.

The distribution also uses this directory to convey sample Geodetic Data Catalogs which you will definitely want to inspect and perhaps modify. Geodetic data files which OSGeo believes to be in the public domain are deposited in sub-folders of this folder in a specific hierarchy. This hierarchy is consistent with the provided Geodetic Data Catalog (.gdc) files. There is no specific requirement for the location of the geodetic data files other than their location must be consistent with the specifications int he Geodetic Data Catalog files.

Also please note that the *OSTN97.TXT* and *OSTN02.TXT* data files must also reside in the primary data directory. Again, by design, there is only one file for each of these transformations, and the implementation of a Geodetic Data Catalog file was skipped. Also, due to the rather strange nature of these files, most of the features of a Geodetic Data Catalog file do not apply anyway.

Test Source Code and Data

The directory named *Test* will contain the source code to the CS-MAP test program and the supporting data file, *TEST.DAT*. Compilation and linking of this program will obviously require the inclusion of the header file and library. Make files for building the test program are provided in this folder.

- *Test.mak* can used used to build the test program in the Linux environment.
- *Test.nmk* can use used int the Microsoft WIndows environment using the *nmake* facility.

In order to execute the entire test sequence, you will need to have compiled the dictionaries with the **/t** option. This causes the retention of the test coordinate systems (not normally distributed with an application) in the dictionary files. The test program will also expect to have access to the NADCON and HPGN data files provided in the sub-directories of the *Dictionaries* directory. When executing the test program, use the **/d** option to indicate the location of the directory in which the dictionaries, NADCON, and HPGN data files reside; e.g. `/d..\Dictionaries`. Refer to Chapter 3 of this manual for more detailed information on the test program.

This program performs a fairly substantial test of most all features and capabilities of CS-MAP. This program should be used each time CS-MAP is used in a new configuration or compiled with a different compiler.

Note that a large number of the tests encoded in the TEST.DAT file are commented out as they rely on the existence of specific geodetic data files which OSGeo cannot distribute. For example, since OSGeo cannot distribute the Canadian National Transformation, all test dependent upon that data file are commented out. Upon obtaining geodetic data files which Mentor Software cannot distribute, you should consider un-commenting the tests related to such files.

Documentation Directory

The directory named *Documentation* is where you will find a copy of this documentation. Depending upon the format, it may consist of a single file, or a directory containing multiple files. A prinable version may also be present.

Additionally, this directory will contain the source code to the help file which is provided for use with the MFC based components of CS-MAP. The source code files include an *.rtf* file, a contents file (*.cnt*), and several screen shots in *.bmp* format. Again, a make file compatible with Visual C++ Version 6 (or later) *nmake* is provided; it is named *help.nmk*. (As the help file is not generally usable in the Linux environment, no Linux compatible make file is provided.) Note, that the CS-MAP MFC based functions expect to find the help file in the same directory as the mapping data files. Of course, a function exists which enables application programmers to override this default location (*CS_setHelpPath*). If for any reason, the MFC based functions cannot locate the help file, the help button on all dialogs will be grayed out.

Please note that the help file is designed for distribution with your application. It does not mention OSGeo, CS-MAP, or the original developers of CS-MAP. It uses a rather generic term "coordinate conversion system" to refer to that which it is describing.

Data Directory

Several data files used in the construction of CS-MAP's Name Mapper facility are included in the *Data* directory. It is envisioned that these files will be replaced by a more convenient and controllable means in the near future.

Make Procedure

Building CS-MAP on Windows and Linux

The CS-MAP distribution will produce a series of five directories (described in detail in the previous topics):

- Include: Contains all header files referenced the source code in the Source directory.
- Source: Contains all the source code for the CS-MAP library itself.
- Dictionaries: Contains the coordinate system dictionaries in source form, and the source code for a compiler which will convert the dictionary source to the operational binary form.
- Test: Contains the source code for a console type test program and the test data which it uses.
- Data: Contains a series of data files used to construct name mapping files.

Building the entire product is a series of five steps:

1. Build the CS-MAP library.
2. Build the dictionary compiler.
3. Run the dictionary compiler.

4 Build the console test program.

5 Execute the console test program.

After installation, and before building, it will be best to obtain a copy of the Canadian National Transformation file (*NTV2_0.gsb*) and copy it to the *Dictionaries/Canada* directory. This data file may not be distributed by OSGeo. Geomatics Canada reserves the right to distribute this file and maintain a list of those using it. Therefore, since we do not distribute the file as part of this open source distribution, we recommend strongly that you simply obtain a copy, even if for testing purposes only. Chances are very good you already have a copy of this file on your system already. If not, you can obtain one (no fee) at:

http://www.geod.nrcan.gc.ca

The *TEST.DAT* data file in the Test directory contains several hundred test points which are directly related to the above mentioned grid shift data file. To prevent confusion and unnecessary technical support, tests related to the Canadian National Transformation data file are commented out in the distribution. After obtaining a copy of the above mentioned data file, these test should be "uncommented" back in, so that the test program will test this feature.

OK. Now for building on your system:

For Windows:

1> Build the CS-MAP Library:

- Make the 'Source' directory your current working directory.
- Use the MSVC set variables script to set the environmental variables correctly.
- Use the 'nmake' command and supply it with the 'Library.nmk' make file. E.g. 'nmake /fLibrary.nmk'

2> Build the Dictionary Compiler (CS_comp)

- Make the 'Dictionaries' directory your current working directory.
- Use the MSVC set variables script to set the environmental variables correctly.
- Use the 'nmake' command and supply it with the 'Compiler.nmk' make file. E.g. nmake /fCompiler.nmk'

3> Run the Dictionary Compiler

- Make the 'Dictionaries' directory your current working directory.
- Execute the 'CS_comp' program. E.g. CS_Comp . .'
- Note that the first argument to this command is the folder containing the dictionary source, the second argument is the directory to which the binary dictionary files are to be written.

4> Build the Console Test program (CS_Test)

- Make the 'Test' directory your current working directory.
- Use the MSVC set variables script to set the environmental variables correctly.
- Use the 'nmake' command and supply it with the 'Test.nmk' make file. E.g. 'nmake /fTest.nmk'

5> Execute the console test program

- Make the 'Test' directory your current working directory.
- Execute the 'CS_Test' program. E.g. 'CS_Test /d..\Dictionaries'
- Note that the /d argument is the directory which the test program is to look to for the dictionaries and related data files.

For Linux:

1> Build the CS-MAP Library:

- Make the 'Source' directory your current working directory.
- Use the 'make' command and supply it with the 'Library.mak' make file. E.g. 'make -fLibrary.mak'

2> Build the Dictionary Compiler (CS_Comp)

- Make the 'Dictionaries' directory your current working directory.
- Use the 'make' command and supply it with the 'Compiler.mak' make file. E.g. 'make -fCompiler.mak'

3> Run the Dictionary Compiler

- Make the 'Dictionaries' directory your current working directory.
- Execute the 'CS_Comp' program. E.g. './CS_Comp . .'
- Note that the first argument is the directory containing the dictionary source, the second argument is the directory to which the binary dictionary files are written.

4> Build the Console Test program (CS_Test)

- Make the 'Test' directory your current working directory.
- Use the 'make' command and supply it with the 'Test.mak' make file. E.g. 'make -fTest.mak'

5> Execute the console test program

- Make the 'Test' directory your current working directory.
- Execute the 'CS_Test' program. E.g. './CS_Test -d../Dictionaries'
- Note that the /d argument is the directory which the test program is to look to for the dictionaries and related data files.

MS VC++ 2005 (Version 8):

The CS-MAP Open Source distribution will deposit in the primary directory a Microsoft Visual C++ Version 8.0 (VC2005) solution file. This file references project files in the Source, Dictionaries, and Test directories. This solution file and its related project files can be used to manufacture the library, dictionary compiler, and the test module. No provisions have been made for executing the dictionary compiler or the test module.

Release Notes for CS-MAP 12

This section, and its future sub-sections will describe the recent changes made to CS-MAP for the benefit of users of previous major releases. Recent changes to the CS-MAP trunk will be described in this topic. As major releases are created into a branch of the source code tree, these notes should be moved to a separate sub-topic.

CHAPTER 2

Chapter 2 -- Descriptions and Discussion

In Chapters 3 through 5 of this manual, you will find detailed information about the components of the CS-MAP library and the executable modules supplied in the OSGeo distribution for testing and maintenance purposes. The purpose of this section of the manual is to provide an overview of CS-MAP so that you will have an idea as to which specific program elements in the remainder of this manual you need to look at.

Therefore, in this section we give a broad overview of the structure of CS-MAP and, usually, a simple function name so that you can locate the specific information you need in Chapters 3, 4 and 5. It is not the intent of this section to duplicate the information contained in the remaining chapters.

Note that the first sub-section of this Chapter is titled Overview, and is expressly designed for Application Programmers who, like the author, don't usually read the manual until something doesn't work. Please take the five minutes necessary to read that section before attempting to add CS-MAP to your application.

Overview

As one programmer to another, I present this Overview Section as the manual for people who, like myself, don't read the manual (until something doesn't work). This section contains all of the information you'll need to get started quickly, and the specific information you'll need to stay out of trouble. Please read this section before attempting to use CS-MAP. Refer to the remainder of the manual as necessary.

Deferring the details to subsequent sections, it is helpful to consider CS-MAP as consisting of a Coordinate System Dictionary and a set of functions which use the information in the dictionary to accomplish the desired task. All coordinate systems used by CS-MAP reside in the dictionary and are given a name, which we refer to as a key name, much like we give names to files. CS-MAP, then, performs coordinate system conversion based on the names of the coordinate systems provided. This technique eliminates the need to have your users process through a long list of parameters which they (usually) don't understand whenever a conversion is necessary. All they need provide are the names of the appropriate coordinate systems.

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 an memory addressing fault.

High Level Interface

Given the name of the source and target coordinate systems, conversion of coordinate data from one to the other is as simple as a single function call. The following example would cause the coordinate in the array of three doubles named *coordinate* to be converted from NAD27 based UTM Zone 13 coordinates to NAD83 based Colorado State Plane, Central Zone, coordinates:

```
status = CS_cnvrt ("UTM27-13","CO83-C",coordinate);
```

It's that simple. The simplicity of this hides several important features. First, note that the datum shift implied by the coordinate systems, NAD27 to NAD83 in this case, is automatically applied. Second, you would hardly ever code an application with hard coded coordinate system names as was done in this example. Simple character arrays that are passed by argument, providing data entered by the user from a choice list, or providing data obtained from a database, whatever, all work just fine. These are just very simple, case insensitive, null terminated strings. Third, the application programmer has no need to know which projections, datums, and/or ellipsoids are involved. Fourth, the source and/or target coordinate systems could just as easily be Latitude and Longitude coordinates, based on any of several units, and referenced to any prime meridian.

Similarly, should your application need to know the grid scale factor at a specific point, all you need code is a function call similar to the following:

```
grid_scale = CS_scale ("CO83-C",coordinate);
```

Coordinate in this case *must* be geographic, i.e. the latitude and longitude of the point at which the grid scale is to be determined. Again, it is unlikely that the name of the coordinate system would be hard coded as was done in this example. Need the convergence angle? You have probably figured it out already:

```
convergence = CS_cnvrg ("CO83-C",coordinate);
```

Concerned about performance? Originally, the above interface was designed as a means by which applications written in languages other than 'C' could access CS-MAP; i.e. no pointers required. However, due to its design and the high speed processors with on-board caches which are common today, this interface provides amazing levels of performance. It is now the interface which we recommend for all simple applications.

Several other features are made available in what we call the High Level Interface; that is features which are available without the use of pointers. The details of all such features are provided in the topic named High Level Interface.

Coordinate System Dictionary

The coordinate system is the heart of the CS-MAP package. The CS-MAP distribution includes the definition of more than 5,000 coordinate systems. Each definition includes the projection to be used, the projection parameters, the datum or ellipsoid to which the coordinate system is referenced, and the unit to be used. Admittedly, many of the coordinate system definitions provided are very similar to others, differing only in the system unit or datum referenced. However, we believe your users will appreciate the simplicity of having only to remember and enter an easy to remember name or two to get what they need. The default feature described elsewhere in this manual can be used in those cases where this design becomes inconvenient.

Coordinate system definitions are stored in a file we refer to as the Coordinate System Dictionary. This file is a simple fixed length record file containing binary data. It is maintained in sorted order by coordinate system name and is accessed using a binary search technique. This provides portability to almost any environment without having to license any other software. Your application will need to provide the CS-MAP functions with the location of the directory in which the Coordinate System Dictionary resides (see *CS_altdr*). Once CS-MAP locates the Coordinate System Dictionary, it expects to find all most other data files in the same directory.

CS-MAP distributions include an ASCII file which defines all of the coordinate systems included in the distribution. This file is usually committed to version control and treated has a highly valuable source file. A compiler included with CS-MAP can convert this ASCII file into the binary form used by the system.

Important Note: The first four bytes of a binary coordinate system dictionary file is a "magic number". This sequence of bytes are used to identify that the file is indeed a coordinate system dictionary file. The revision level of the dictionary format is also encoded into this "magic number".

Other Interfaces

It is probably obvious to you that the interface described above can not be the most efficient interface possible. Applications which require the absolute highest performance, may want to use the High Performance Interface. This interface requires the knowledge of as many as ten functions. It also requires the use of pointers to structures and, therefore, is usable only from languages such as C, C++, or Pascal which can handle pointers appropriately. It does, however, provide the application programmer with the highest performance level possible, while still insulating your application from changes to the internals of CS-MAP. Using this interface, performance levels of 1,000,000 non-trivial conversions per second are routinely observed.

Applications which are hard coded around specific projections can use the Low Level Interface to obtain high performance solutions which do not require a Coordinate System Dictionary or, for that matter, any other supporting data. That is, at this level, the application programmer has access to the specific code for each projection. Similarly, applications can use the low level interface to access any of the 12 or so datum shift techniques supported by the library.

The High Performance and Low Level Interfaces are described in detail in the remaining chapters of this manual.

Other Dictionaries

Like coordinate systems, datums are many and varied. (The term datum is widely used to refer to a Horizontal Geodetic Reference System. Although this usage is not technically incorrect, we will go along with the crowd and use the simpler term.) While several datum conversion techniques are hard coded into CS-MAP, the actual definitions are not. As you might suspect, there exists a Datum Dictionary which contains the definition of all datums known to the system and provides a name by which they can be accessed (i.e. key name). Coordinate systems are referenced to a datum by including the datum key name in the coordinate system definition. Thus, when converting from one coordinate system to another, CS-MAP can automatically activate the appropriate datum shifts necessary by examining the datum references in the coordinate system definitions. While full support is provided, application programmers rarely, if ever, access the Datum Dictionary directly.

All datum definitions must include a reference to an ellipsoid; coordinate system definitions can include a reference to an ellipsoid. Again, there are many ellipsoid definitions and these are not hard coded into CS-MAP. You guessed it! There is an Ellipsoid Dictionary:

- that carries the definitions of all ellipsoids,
- that assigns each ellipsoid definition a name (i.e. a key name),
- to which CS-MAP provides a full set of access functions; and
- that the application programmer rarely, if ever, accesses directly.

Important Note: Datum and Ellipsoid dictionaries include a "magic number" as the first four bytes of the binary file. These "magic numbers" identify the type of the file in addition to the file format revision level.

Cartographic vs. Geodetic Referencing of Coordinate Systems

While coordinate system definitions can be referenced to a datum, they can also be referenced directly to an ellipsoid. In this manual, we will use the term *geodetically referenced* coordinate system to refer to the former case, and *cartographically referenced* coordinate system to refer to the latter. When both the source and target coordinate systems involved in a conversion are geodetically referenced, CS-MAP can automatically perform the necessary datum shift without any additional information required from the user, or any additional code required by the application programmer. When either or both coordinate system definitions are cartographically referenced, the datum shift feature is effectively disabled. That is, CS-MAP cannot calculate a datum shift it does not know both the source and target datums.

In many cases, the datums upon which coordinates are based are known. In these cases, coordinate system definitions are usually geodetically referenced and CS-MAP can, therefore, include the appropriate datum shift automatically whenever appropriate. In other cases, the datum upon which a coordinate system is based is not known and the coordinate system should be cartographically referenced to an ellipsoid. CS-MAP's automatic datum shift feature is then disabled whenever this coordinate system is involved in a coordinate conversion. Therefore, while the individual maintaining the coordinate system definitions may need to understand the distinctions between geodetic and cartographic references, the typical end user is not usually concerned.

Cartographically referenced coordinate systems can be a great convenience to both user and application programmer. For example, the *LL* coordinate system which you will find in the CS-MAP distribution is cartographically referenced to the WGS84 ellipsoid (not the WGS84 datum). This means that using the *LL* coordinate system as the target in any conversion produces geographic coordinates which are always based on the same datum as the source coordinate system. This is often very convenient indeed.

Latitudes and Longitudes

Coordinate systems can be defined to be geographic, i.e. consist of latitude and longitude coordinates. This is achieved through the use of a pseudo projection we call the Unity projection. The Unity Projection is simply a projection which does nothing, the results being geographic coordinates. Since the definitions of a latitude and longitude coordinate systems are included in the Coordinate System Dictionary, they can, and indeed do, have datum references, origin longitudes, and units specifications. Thus, several latitude and longitude definitions appear in the coordinate system dictionary. As with any other coordinate system definition, these definitions are either geodetically or cartographically referenced, have a unit specification, and have an origin longitude. In this context, the units are the angular units of measure to be used (e.g. degrees, minutes, seconds, grads, etc.) and the origin longitude is the prime meridian (i.e. Greenwich, Meridian of Paris, etc.). Thus, latitude and longitude coordinates are fully supported by the system, and the application programmer is not required to write any special code to process them.

The distribution Coordinate System Dictionary contains a geographic coordinate system simply named LL. This is a cartographically referenced latitude and longitude coordinate system which is referenced to the Greenwich Prime Meridian, using units of degrees. Thus, LL should be considered to be the generic latitude and longitude coordinate system, and can be used as such. For example, to obtain the grid scale factor of a coordinate system using cartesian coordinates (as opposed to the geographic coordinates required by the *CS_scale* function) one could simply write as follows:

```
status = CS_cnvrt ("CO83-C","LL",coordinate);
grid_scale = CS_scale ("CO83-C",coordinate);
```

Note, that due to the generic nature of the LL coordinate system (i.e. cartographically referenced, thereby disabling datum shifts), the intermediary latitude and longitude results will be always be based on the same datum as that of the source coordinate system. Also, the LL coordinate system is defined to match the specific definition of latitude and longitude used internally within CS-MAP. That is, in those few cases where CS-MAP specifically requires a coordinate in terms of latitude and longitude:

1. the coordinates must be in degrees,
2. referenced to the Greenwich Prime Meridian, and
3. west longitude and south latitude must be negative values.

Coordinate Arrays

As is common in the industry, cartesian coordinates are passed in arrays of doubles. The X coordinate is always the first element in the array, the Y coordinate being the second element, and the Z coordinate being the third element. When performing a two dimensional conversion, CS-MAP often ignores the Z coordinate. Obviously, when performing a three dimensional conversion, a valid Z coordinate is required. To simplify use of the library, all coordinate arrays should be dimensioned at three. Typically, a Z value of zero is provided when performing a two dimensional conversion.

In the case of cartesian coordinates, coordinates must be ordered as described above; specifically X, Y, and then Z. While it is common usage to refer to geographic coordinates as latitude and longitude, and it is also common to give the latitude first and the longitude second, CS-MAP requires that the first element of a geographic coordinate be the longitude, the second element be the latitude, and the third element the height. While negative values are usually used to indicate west longitude and south latitude, geographic coordinate systems can be defined where the opposite sign convention is used.

It is important to note that there is a significant difference between coordinates returned to the user which may just happen to be geographic, and the geographic coordinates required by certain CS-MAP functions. The conventions used in "user" coordinates are determined by the coordinate system definition. Thus, in a "user" geographic coordinate, as returned by *CS_cs2ll* for example, the prime meridian may be other than Greenwich and the unit other than degrees. Users can also define geographic coordinate systems where west longitude is positive and/or the order of the coordinates is swapped.

However, wherever a CS-MAP function specifically requires a geographic coordinate, the values provided must:

1. be given in degrees,
2. be referenced to the Greenwich Prime Meridian, and
3. west longitude and south latitude be given as negative values.

Selected Source Code

For historical reasons, most all global data definitions are coded in distinct modules. This has been very convenient over the years. Many of the features of CS-MAP can be adjusted to better fit into your application by tweaking one of these source modules. Four of these modules are worth mention in this overview: *CSdata*, *CSdataU*, *CSdataPJ*, and *CS_error*. In *CSdata* you will find the definition of many global constants used throughout CS-MAP. While you obviously don't want to change the definition of π, there are several other aspects of CS-MAP which are controlled by global variables defined in this module. Make the modifications which you need, recompile, and re-link your application.

CSdataU contains the unit table which CS-MAP uses. To add (or remove) a unit, simply modify the table and recompile.

CSdataPJ contains CS-MAP's projection table. To remove a projection (to reduce the size of your executable, for example), simply remove the projection's entry in the table an all object code references to the projection will be removed.

CS_error contains the text of all error messages.

All data which may be language related is contained in these four modules. To translate the entire system to another language, only these four modules, and perhaps the help file, need attention.

Naming Conventions

Originally, the names of global variables, manifest constants (i.e. defines), structures, and functions used in CS-MAP adhered to a very definitive naming convention. Much of the library still adheres to this convention. This convention was developed with three purposes in mind. First, and foremost, it is necessary to insure that the probability of a name collision with existing application code is kept to a minimum. Second, to enable programmers to quickly determine the type of entity being referenced by a name and to quickly determine where the definition of such can be found. Third, provide an efficient means by which the libraries and other components of CS-MAP can be efficiently maintained and manufactured.

In later developments of the library, such as the inclusion of some C++ elements and the planned porting of the entire library to C++, there are several modules no longer adhere to the original naming convention.

Name Collisions

All names whose scope extends outside the specific file in which is it is defined will start with the two character sequence *CS*. As described below, this initial sequence may be in upper or lower case. Additionally, every such name will contain at least one upper case character and at least one lower case character. In this way, the possibility of a CS-MAP global name being the same as a name already used in your application is virtually nil.

Function Names

All function names begin with an upper case *CS* sequence. If the function is expected to be accessed by modules outside of the CS-MAP library in normal use, the initial *CS* sequence is followed by an underscore character. The remainder of the function name follows, the first of which will be lower case if it is an alphabetic character.

Structure Tags

Structure tags begin with a lower case *cs* sequence. If the structure is expected to be accessed by modules outside of the CS-MAP library in normal use, the initial sequence is followed by an underscore character. The remainder of the structure tag follows, the first character of which will always be uppercase. Finally, the last character of all structure tags will be the underscore character.

Global Variable Names

Global variable names begin with a lower case *cs* sequence. If the global variable is expected to be accessed by modules outside of the CS-MAP library in normal use, the initial sequence is followed by an underscore character. The remainder of the global variable name follows, the first character of which will always be an upper case letter. A global variable name will never end with the underscore character. A global variable which is a definition of a structure, or a pointer to same, will usually have the same name as the structure tag, sans the trailing underscore.

Manifest Constants

Manifest constant names, e.g. include file *define*'s, begin with a lower case *cs* sequence. If the constant being defined is expected to be used by modules outside of the CS-MAP library, the initial sequence is followed by an underscore character. The remainder of the constant name follows and will be all upper case.

Naming Convention Examples

Table II shows several examples of the naming convention.

Name	Type	Comment
CS_csloc	Function, external	Name of a function expected to be called from outside of the CS-MAP library.
CSnad283	Function, internal	Name of a function not expected to be called from outside of the CS-MAP library.
cs_Csdef_	Structure tag, external	Structure tag name for a structure expected to be referenced by modules outside of the CS-MAP library.
csNaddir_	Structure tag, internal	Structure tag name for a structure not expected to be accessed outside of the CS-MAP library.

cs_Dir	Global Variable, external	Global variable name expected to be accessed outside of the CS-MAP library.
csErrlng	Global Variable, internal	Global variable name not expected to be accessed outside of the CS-MAP library.
cs_NO_MEM	Manifest Constant, external	Manifest constant expected to be referenced by modules outside of the CS-MAP library.
csGRF_MAX_ACTIVE	Manifest Constant, internal	Manifest constant not expected to be referenced by modules outside of the CS-MAP library.

Table II - Naming Convention Examples

Projection Code Names

Each of the thirty eight projections has a five character code name which is used in all structure tags and function names associated with the specific projection. Table III lists each projection, the five character code value, the structure tag name, and setup function name associated with each as examples of how this code value is used to identify the projection each is associated with. All code elements which are specifically related to specific projection are named in a similar manner.

Projection	Code	Structure Tag	Setup Function
Transverse Mercator	trmer	cs_Trmer_	CStrmerS
Lambert Conformal Conic	lmbrt	cs_Lmbrt_	CSlmbrtS
Hotine Oblique Mercator	oblqm	cs_Oblqm_	CSoblqmS
Alber's Equal Area	alber	cs_Alber_	CSalberS
Mercator	mrcat	cs_Mrcat_	CSmrcatS
Miller Cylindrical	millr	cs_Millr_	CSmillrS
Lambert Equidistant Azimuthal	azmed	cs_Azmed_	CSazmedS
Lambert Equal Area Azimuthal	azmea	cs_Azmea_	CSazmeaS
Polar Stereographic	pstro	cs_Pstro_	CSpstroS
Oblique Stereographic	ostro	cs_Ostro_	CSostroS
Snyder's Oblique Stereographic	sstro	cs_Sstro_	CSsstroS
Equidistant Conic	edcnc	cs_Edcnc_	CsedcncS
Sinusoidal	sinus	cs_Sinus_	CSsinusS
American Polyconic	plycn	cs_Plycn_	CSplycnS
Modified Polyconic	modpc	cs_Modpc_	CsmodpcS
Lambert Tangential	lmtan	cs_Lmtan_	CSlmtanS
Van der Grinten	vdgrn	cs_Vdgrn_	CSvdgrnS
Orthographic	ortho	cs_Ortho_	CSorthoS

Gnomonic	gnomc	cs_Gnomc_	CsgnomcS
Equidistant Cylindrical	edcyl	cs_Edcyl_	CSedcylS
Cassini	csini	cs_Csini_	CScsiniS
Modified Stereographic	mstro	cs_Mstro_	CSmstroS
New Zealand National Grid	nzlnd	cs_Nzlnd_	CSnzlndS
Robinson Cylindrical	robin	cs_Robin_	CSrobinS
Bonne	bonne	cs_Bonne_	CsbonneS
Equal Area (Authalic) Cylindrical, Normal Aspect	nacyl	cs_Nacyl_	CSnacylS
Equal Area (Authalic) Cylindrical, Transverse Aspect	tacyl	cs_Tacyl_	CStacylS
Mollweide	molwd	cs_Molwd_	CsmolwdS
Eckert IV	ekrt4	cs_Ekrt4_	CSekrt4S
Eckert VI	ekrt6	cs_Ekrt6_	CSekrt6S
Goode Homolosine	hmlsn	cs_Hmlsn_	CShmlsnS
Bipolar Oblique Conformal Conic	bpcnc	cs_Bpcnc_	CSbpcncS
Oblique Cylindrical	swiss	cs_Swiss_	CSswissS
Snyder Transverse Mercator	trmrs	cs_Trmrs_	CStrmrsS
Krovak Oblique Conformal Conic	krovk	cs_Krovk_	CSkrovkS
Non-georeferenced Coordinates	nerth	cs_Nerth_	CSnerthS
Danish System 34	sys34	cs_Sys34_	CSsys34S
Unity Pseudo Projection	unity	cs_Unity_	CSunityS

Table III - Projection Code Names and Usage Examples

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_cnvrt returns 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.

Grid Scale Factor -- CS_scale

Given a coordinate system name and a location in the form of a geographic coordinate, *CS_scale* will return the grid scale factor of the coordinate system at the specified location. *CS_scale* returns a negative one in the event of an error condition. In such cases, the cause of the error can be determined by examining `cs_Error` which will contain the CS-MAP numeric error code of the condition which caused the error. *CS_errmsg* can be used to obtain a textual description of the error condition. An error caused by the location being outside of the domain of the coordinate system will be indicated by a `cs_Error` value of zero. In this case, no textual description will be available.

Note that the coordinate provided as the second argument must be a geographic coordinate, i.e. latitude and longitude in degrees referenced to the Greenwich prime meridian. Longitude is the first element in the array, latitude is the second. (The third element is not currently used for grid scale calculations, but may be in the future.) As always for internal geographic coordinates, use negative values for west longitude and south latitude.

Convergence Angle -- CS_cnvrg

Given a coordinate system name and a location in the form of a geographic coordinate, *CS_cnvrg* will return the convergence angle of the coordinate system at the specified location, in degrees east of north. *CS_cnvrg* returns a negative 360 (i.e. -360) value in the event of an error condition. In such cases, the cause of the error can be determined by examining `cs_Error` which will contain the CS-MAP numeric error code of the condition which caused the error. *CS_errmsg* can then be used to obtain a textual description of the error condition. An error caused by the location being outside of the domain of the coordinate system will be indicated by a `cs_Error` value of zero. In this case, no textual description will be available.

Note that the coordinate provided as the second argument must be a geographic coordinate, i.e. latitude and longitude in degrees referenced to the Greenwich prime meridian. Longitude is the first element in the array, latitude is the second. (The third element is not used for convergence calculations.) As always, use negative values for west longitude and south latitude.

Data Directory -- CS_altdr

In order to operate correctly, CS-MAP needs to access to several data files, the most important of which is the Coordinate System Dictionary. *CS_altdr* can be used to provide CS-MAP with the path to the directory it should look in for all of its data files. The single argument should contain the full path to the directory containing all of CS-MAP's supporting data files. You can instruct *CS_altdr* to use the value of the `CS_MAP_DIR` environmental variable by setting the argument to the **NULL** pointer. Should the argument point to the null string, CS-MAP will consider the current directory on the current drive as the directory in which to search for data file.

In all cases, *CS_altdr* will return a -1 if a valid Coordinate System Dictionary file could not be located in the indicated directory, for whatever reason.

Important Note: Failure to successfully call this function prior to calling any other CS-MAP function is likely to cause a fatal addressing error and the host application to crash.

Recover System Resources -- CS_recvr

Use this function to recover any and all system resources, such as file descriptors and heap memory, which CS-MAP may have allocated due to calls to *CS_cnvrt*, *CS_scale*, and *CS_cnvrg* functions.

Get Error Message Text -- CS_errmsg

CS_errmsg returns in the buffer supplied by the calling module a null terminated string which is suitable for reporting the last error condition detected by CS-MAP. This function should be used only after any CS-MAP function returns a negative status. It should be called prior to any other CS-MAP function call.

Compute Azimuth and Distance -- CS_llazdd

CS_llazdd is a utility function which is a part of the High Level Interface. Use this function to compute the azimuth from one geographic coordinate to another. It also returns the distance between the two points. These calculations take full account of the ellipsoid, and ellipsoid parameters are part of the calling sequence.

Unit Lookup -- CS_unitlu

Use *CS_unitlu* function to obtain the conversion constant for any of the unit systems understood by CS-MAP. *CS_unitlu* will return a zero if the supplied unit name is not valid.

Coordinate System Name Verification – CS_csIsValid

Use the *CS_csIsValid* function to determine if a coordinate system key name is that of an existing coordinate system defined in the currently active Coordinate System Dictionary without any side affects.

Datum Name Verification – CS_dtIsValid

Use the *CS_dtIsValid* function to determine if a datum key name is that of an existing datum defined in the currently active Datum Dictionary without any side affects.

Ellipsoid Name Verification – CS_elIsValid

Use the *CS_elIsValid* function to determine if a ellipsoid key name is that of an existing ellipsoid defined in the currently active Ellipsoid Dictionary without any side affects.

Low Level Functions

While the use of the High Level or the High Performance Interfaces described above is highly recommended, certain applications may require the use of the lower level functions. The sub-sections of this section organize these functions into three major groups:

1. cartographic projection functions,
2. geodetic datum shift functions, and
3. general mapping/geodetic functions.

Cartographic Projections

For each projection supported, there exist eight, possibly nine, functions which comprise the full implementation of a projection. The brief description of these functions is given below. These descriptions will be important to those adding a new projection as well.

Those adding projections to the system only need to add an entry to the projection table `cs_Prjtab`. Doing so, you will need to add a pointer to the Definition Check and the Setup functions. To activate the proper parameter settings in the MFC dialog, you will also need to add the projection to the `cs_PrjprmMap` table. It is unlikely, but if your new projection uses a new type of parameter, you may need to add a new entry to the `cs_Prjprm` table. All of these tables are defined in the *CSdataPJ.c* module.

Definition Check Functions

For each projection there exists a function which verifies that a coordinate system definition adheres to the requirements of the projection. These functions are named in a manner similar to all other projection functions, but the distinguishing final character is Q. To prevent large scale duplication of code, each Q function checks only those elements of a coordinate system definition which are specific to the projection. The generalized check function, *CS_cschk*, checks those elements which are common to all coordinate systems (e.g. datum, ellipsoid, and units).

CS_csloc calls this function prior to calling the setup function, thus providing the setup function with data known to be valid for the given projection. This also implies that the pointer to the Q function for each projection must reside in the projection table.

Setup Function

The setup function for each projection has two basic responsibilities. It should perform all of the one-time calculations which can be performed independent of the specific coordinates which are to be converted and insert in the `cs_Csprm_` structure pointers to the nine functions required for coordinate conversion. It is the setup function which is the primary repository for all knowledge about a specific projection. Therefore, it is one of the five elements required in the projection table other than the name of the projection. (C++ users would use the term constructor for the setup function. The design of CS-MAP predates the availability of C++ compilers on personal computers.)

This function is always supplied with a pointer to a `cs_Csprm_` structure. This single argument supplies the setup function with the information required to perform the setup via the `csdef` and `datum` elements and the repository for the results of the setup by way of the `ll2cs`, `cs2ll`, `cs_csscl`, `cs_cscnv`, `cs_cssck`, `cs_cssch`, `llchk`, `xychk`, and `prj_prms` elements. Note that the `prj_prms` element of the `cs_Csprm_` structure is a union of all the pre-processed projection parameter structures, thus providing a repository for setup parameters regardless of the projection in use. Refer to Table IV (given in the next topic) for the names of the setup functions for all thirty eight projections currently supported.

Also note that in order to reduce the amount of duplicated code necessary to support the large number of projection variations now supported, the `prj_code` element of the `cs_Csprm_` structure must be filled in as well prior to calling the setup function.

Forward Functions

For each supported projection, there exists a forward function. It is responsible for converting latitudes and longitudes to the appropriate coordinates given a pointer to the projection parameters for the specific projection. A pointer to such function is inserted into the `ll2cs` element of the `cs_Csprm_` structure by the setup function. These functions require that they be given a pointer to the projection parameters calculated by the setup function, e.g. a pointer to a element of the `prj_prms` union in the `cs_Csprm_` structure. Refer to Table IV for the names of the forward functions for all thirty eight projections currently supported.

Setup Function	Forward Function	Inverse Function	Projection
CstrmerS	CStrmerF	CStrmerI	Transverse Mercator
CSlmbrtS	CSlmbrtF	CSlmbrtI	Lambert Conformal Conic
CsoblqmS	CSoblqmF	CSoblqmI	Hotine Oblique Mercator
CSalberS	CSalberF	CSalberI	Alber's Equal Area Conic
CSmrcatS	CSmrcatF	CSmrcatI	Mercator
CSmillrS	CSmillrF	CSmillrI	Miller Cylindrical
CsazmedS	CSazmedF	CSazmedI	Lambert Equidistant Azimuthal
CsazmeaS	CSazmeaF	CSazmeaI	Lambert Equal Area Azimuthal
CSpstroS	CSpstroF	CSpstroI	Polar Stereographic
CSostroS	CSostroF	CsostroI	Oblique Stereographic
CSsstroS	CSsstroF	CSsstroI	Snyder's Oblique Stereographic
CsedcncS	CSedcncF	CSedcncI	Equidistant Conic
CSsinusS	CSsinusF	CSsinusI	Sinusoidal
CSplycnS	CSplycnF	CSplycnI	American Polyconic
CsmodpcS	CSmodpcF	CSmodpcI	Modified Polyconic
CSlmtanS	CSlmtanF	CSlmtanI	Lambert Tangential
CSvdgrnS	CSvdgrnF	CSvdgrnI	Van der Grinten
CSorthoS	CSorthoF	CSorthoI	Orthographic
CsgnomcS	CSgnomcF	CSgnomcI	Gnomonic
CSedcylS	CSedcylF	CSedcylI	Equidistant Cylindrical
CScsiniS	CScsiniF	CScsiniI	Cassini
CSmstroS	CSmstroF	CSmstroI	Modified Stereographic
CSnzlndS	CSnzlndF	CSnzlndI	New Zealand National Grid
CSrobinS	CSrobinF	CSrobinI	Robinson

CSbonneS	CSbonneF	CSbonneI	Bonne
CSnacylS	CSnacylF	CSnacylI	Normal Aspect, Equal Area (Authalic) Cylindrical
CStacylS	CStacylF	CSnacylI	Transverse Aspect, Equal Area (Authalic) Cylindrical
CsmolwdS	CSmolwdF	CSmolwdI	Mollweide
CSekrt4S	CSekrt4F	CSekrt4I	Eckert IV
CSekrt6S	CSekrt6F	CSekrt6I	Eckert VII
CShmlsnS	CShmlsnF	CShmlsnI	Goode Homolosine
CSbpcncS	CSbpcncF	CSbpcncI	Bipolar Oblique Conformal Conic
CSswissS	CSswissF	CSswissI	Oblique Cylindrical
CStrmrsS	CStrmrsF	CStrmrsI	Transverse Mercator ala Snyder
CSkrovkS	CSkrovkF	CSkrovkI	Krovak Oblique Conformal Conic
CSnerthS	CSnerthF	CSnerthI	Non-georeferenced coordinate system

Table IV - Setup, Forward, and Inverse Function Names

CSsys34S	CSsys34F	CSsys34I	Danish System 34
CSunityS	CSunityF	CSunityI	Unity (pseudo projection)

Inverse Functions

Similarly, there exists for each projection an inverse function, responsible for converting coordinate system coordinates to latitudes and longitudes. A pointer to such function is inserted into the `cs2ll` element of the `cs_Csprm_` structure by the setup function, and these functions require a pointer to the setup parameters calculated by the setup function. Refer to Table IV for the names of the inverse functions for all thirty eight projections currently supported. Please note that the last character in the name of each of these functions is I (uppercase i).

Scale Functions

The Coordinate System Mapping Package also includes the ability to determine the grid scale factor of a coordinate system at any point. In many cases there is an analytical formula which produces the desired results. Since analytical formulas for the grid scale factor for all thirty eight projections could not be found, the grid scale factor is determined empirically for some projections using the latitude/longitude azimuth and distance calculation function *CS_llazdd*.

K Scale Function	H Scale Function	Convergence Function	Projection
CStrmerK	<none>	CStrmerC	Transverse Mercator
CSlmbrtK	<none>	CSlmbrtC	Lambert Conformal Conic
CSoblqmK	<none>	CSoblqmC	Hotine Oblique Mercator
CSalberK	CSalberH	CSalberC	Alber's Equal Area Conic
CSmrcatK	<none>	CSmrcatC	Mercator
CSmillrK	CSmillrH	CSmillrC	Miller Cylindrical
CSazmedK	CSazmedH	CSazmedC	Lambert Equidistant Azimuthal
CSazmeaK	CSazmeaH	CSazmeaC	Lambert Equal Area Azimuthal
CSpstroK	<none>	CSpstroC	Polar Stereographic
CSostroK	<none>	CSostroC	Oblique Stereographic
CSsstroK	<none>	CSsstroC	Snyder's Oblique Stereographic
CsedcncK	CSedcncH	CSedcncC	Equidistant Conic
CSsinusK	CSsinusH	CSsinusC	Sinusoidal
CSplycnK	CSplycnH	CSplycnC	American Polyconic
CsmodpcK	CSmodpcH	CSmodpcC	Modified Polyconic
CSlmtanK	CSlmtanH	CSlmtanC	Lambert Tangential
CSvdgrnK	CSvdgrnH	CSvdgrnC	Van der Grinten
CSorthoK	CSorthoH	CSorthoC	Orthographic
CsgnomcK	CSgnomcH	CSgnomcC	Gnomonic
CSedcylK	CSedcylH	CSedcylC	Equidistant Cylindrical
CScsiniK	CScsiniH	CScsiniC	Cassini
CSmstroK	<none>	CSmstroC	Modified Stereographic
CSnzlndK	<none>	CSnzlndC	New Zealand National Grid
CSrobinK	CSrobinH	CSrobinC	Robinson
CsbonneK	CSbonneH	CSbonneC	Bonne
CSnacylK	CSnacylH	CSnacylC	Normal Aspect, Equal Area (Authalic) Cylindrical
CStacylK	CStacylH	CSnacylC	Transverse Aspect, Equal Area (Authalic) Cylindrical

CsmolwdK	CSmolwdH	CSmolwdC	Mollweide
CSekrt4K	CSekrt4H	CSekrt4C	Eckert IV
CSekrt6K	CSekrt6H	CSekrt6C	Eckert VII
CShmlsnK	CShmlsnH	CShmlsnC	Goode Homolosine
CSbpcncK	<none>	CSbpcncC	Bipolar Oblique Conformal Conic
CSswissK	<none>	CSswissC	Swiss Oblique Cylindrical
CStrmrsK	<none>	CStrmrsC	Transverse Mercator ala Snyder
CSkrovkK	<none>	CSkrovkC	Krovak Oblique Conformal Conic
CSnerthK	<none>	CSnerthC	Non-georeferenced coordinate system; scale is always 1.0, convergence is always zero.

Table V - K Scale, H Scale, and Convergence Angle Function Names

CSsys34K	<none>	CSsys34C	Danish System 34 (believed to be conformal, but this is not a sure thing)
CSunityK	<none>	CSunityC	Unity

As mentioned above, for non-conformal projections, there are two scale factors, K and H. Therefore, for all thirty eight projections, there exists a *K* function, and for all non-conformal projections there exists an *H* function. A pointer to the appropriate function is inserted into the `cs_Csprm_` structure by the setup function and, as you might expect, each of these functions requires a pointer to the projection parameters as calculated by the setup function.

Refer to Table V for the names of the *K* scale functions, i.e. grid scale factor along a parallel, and the *H* scale functions (i.e. scale along a meridian) for all thirty five projections currently supported. The name of the *H* function is given as <none> for conformal projections. In these cases, the H scale factor is the same as the K scale factor.

Convergence Functions

For each projection there exists a function which computes the convergence angle for any point given a coordinate system definition. Refer to Table V for the names of the convergence angle functions for all thirty one projections currently supported. Again, analytical formulas for the convergence angle have not been located for all projections. Therefore, for some projections the convergence angle is determined empirically using the *CS_llazdd* function.

Geographic Limits Check Functions

For each projection, there exists a function which determines if a given geographic coordinate, great circle, or region is entirely within the mathematical domain of a coordinate system. These functions are named in a manner similar to all other projection functions, but the distinguishing final character is L.

For performance reasons, the actual conversion functions of CS-MAP check the coordinate data they are provided only to the extent necessary to prevent floating point exceptions. The geographic limits function of each projection can be used prior to a conversion to determine if a specific geographic coordinate, great circle defined by two geographic coordinates, or a closed region defined by four or more geographic coordinates is entirely within the mathematical domain of the projection. It is not unusual for applications to know the extents of the data set which is to be converted, and thus the extents only, not each individual coordinate, need be checked. This can provide significant performance advantages.

Cartographic Limits Check Functions

For each projection, there exists a function which determines if a given cartesian coordinate, line segment, or region is entirely within the mathematical domain of a coordinate system. These functions are named in a manner similar to all other projection functions, but the distinguishing final character is X.

For performance reasons, the actual conversion functions of CS-MAP check the coordinate data they are provided only to the extent necessary to prevent floating point exceptions. The cartesian limits function of each projection can be used prior to a conversion to determine if a specific cartesian coordinate, line defined by two cartesian coordinates, or a closed region defined by four or more cartesian coordinates are entirely within the mathematical domain of the projection. It is not unusual for applications to know the extents of the data set which is to be converted, and thus the extents only, not each individual coordinate, need be checked. This can provide significant performance advantages.

Geodetic Datum Shift Functions

The methods/functions associated with geodetic datum shifts are not nearly as well organized as those of cartographic projections. This is the result of many different governmental agencies solving the problem independently and relying on different data sets and calculation techniques. However, the basic functions involved in the most generalized techniques are described in the following sub-sections.

NADCON Emulation Functions

Four lower level functions can be used to perform NAD27 to NAD83 conversions. Use *CSnadInit* to initialize the system, and *CSnadCls* to release all resources absorbed by *CS_nadinit*.

Once *CSnadInit* has been called, *CSnad27ToNad83* can be called to convert geographic coordinates from NAD27 to NAD83. *CSnad83ToNad27* can be used to convert NAD83 to NAD27. Its that simple.

Datum Conversion Functions

The basic technique used for NAD83 and HARN described in previous sections is used for several other datums now defined worldwide. In the descriptions given below, you will see how this technique applies to AGD66; the Geodetic Datum of Australia of 1966. A similar pattern exists for the following datums and descriptions of these inidividual sets of functions will not be repeated in this section of the manual. In the future, there are likely to be a lot more of these.

Abbreviated Name	Full Name
AGD66	Australian Geodetic Datum of 1966
AGD84	Australian Geodetic Datum of 1984
GDA94	Geocentric Datum of Australia, 1994
NZGD49	New Zealand Geodetic Datum of 1949
NZGD2K	New Zealand Geocentric Datum of 2000
ATS77	Average Terrestrial System of 1977
CSRS	Canadian Spatial Reference System

Four lower level functions can be used to convert coordinates from AGD66 to GDA94. Use *CSagd66Init* to initialize the system, and *CSagd66Cls* to release all resources absorbed by *CSagd66Init*.

Once *CSagd66Init* has been successfully called, *CSagd66ToGda94* can be used to convert geographic coordinates from AGD66 to GDA94. Similarly, *CSgda94ToAgd66* can be used to convert geographic coordinates from GDA94 to AGD66.

General Utility Functions

Supporting the generalized coordinate conversions described in the previous sections, are several functions which perform calculations which are quite useful to the GIS/GPS/Mapping application programmer. Several (but probably not all) of these are described in the following sub-sections.

GEOID Height Functions

This facility enables applications to calculate and use, as appropriate, the geoid height (or geoid separation if you prefer) at locations for which data is available. Record the data files available, and desired to be used, in the Geodetic Data Catalog file named *GeoidHeight.gdc*.

This implementation emulates C++, but is written in ANSI compliant C. The functions are named: *CSnewGeoidHeight*, *CSdeleteGeoidHeight*, and *CScalcGeoidHeight;* and are defined in the module named *cs_GeoidHeight.c*. Code specific to geoid height file formats can be found in modules named: *CS_geoid96.c*, *CS_geoid99.c*, *CS_bynFile.c*, and *CS_osgm91.c*. Low level applications may wish to access the functions defined in these modules directly.

Note there is no inverse function, as it is unnecessary. To obtain orthometric height at a given location, add the geoid height returned by *CScalcGeoidHeight* to the ellipsoid height. To calculate the ellipsoid height, subtract the geoid height returned by *CScalcGeoidHeight* from the orthometric height.

Geocentric Coordinates

Converting between geographic and geocentric coordinates has been inside of CS-MAP for many years. However, this capability has been hidden inside of the datum conversion functions. In this release, this capability is now explicitly available in functions named *CS_llhToXyz* and *CS_xyzToLlh* which are defined in the *CS_dtCalc.c* module.

Note that each of these functions requires the definition of the ellipsoid in use, expressed as two separate `double` arguments: equatorial radius and eccentricity squared.

MGRS Implementation

Release 11.01 includes a series of new functions that provide the ability to generate MGRS designations from geographic coordinates, and vice versa. This implementation consists of 6 functions designed for application programmer use, and two supporting functions. The support functions may be of interest as they provide the ability to convert between geographic and UTM coordinates/zone number where the rather strange stuff which goes on in northern Europe (i.e. southern Norway and the Svaldberg Islands) is appropriately accounted for.

While written in 'C' to be consistent with the rest of CS-MAP, the implementation of the MGRS capability has a definite C++ structure to it. That is, there exists a structure definition, three constructors, a destructor, two public functions and two private functions (i.e. the supporting functions).

MGRS Constructors

Construction of a `cs_Mgrs_` object (i.e. allocation and initialization of a a `cs_Mgrs_` structure) requires knowledge of the ellipsoid definition to be used and if the alternative lettering sequence (i.e. Bessel) is to be used. Thus, the three constructors simply provide three different ways of specifying the ellipsoid which is to be used:

```
struct cs_Mgrs_ *CSnewMgrs (double e_rad,double e_sq,short bessel);
struct cs_Mgrs_ *CSnewMgrsE (const char *elKeyName,short bessel);
struct cs_Mgrs_ *CSnewMgrsD (const char *dtKeyName,short bessel);
```

where the **elKeyname** argument can be used to specify the ellipsoid by key name. Alternatively, application programmers can specify a datum name (the **dtKeyName** argument) and the calculations will be based on the ellipsoid upon which the datum is referenced. Of course, the application programmer can use the **e_rad** and **e_sq** version to specify the equatorial radius and square of eccentricity directly. In all cases, the **bessel** argument is zero to indicate the normal lettering scheme. +1 to indicate the alternative lettering scheme.

All constructors return the null pointer in the event of an error. Use *CS_errmsg* to obtain a string that describes the nature of the error.

MGRS Destructor

Use *CSdeleteMgrs* to delete a `cs_Mgrs_` object constructed by one of the ocnstructors. Currently, a call of *CS_free* will accomplish the same thing, but maybe not in the future.

```
void CSdeleteMgsr (struct cs_Mgrs_ * __This);
```

Like it's C++ equivalent, this function is smart enough not to attempt to free a null pointer.

MGRS Public Functions

Naturally enough, two conversion functions exists. Given a properly initialized `cs_Mgrs_` object (i.e. structure) and a geographic coordinate, *CScalcMgrsFromLl* will return the appropriate MGRS designation. *CScalcLlFromMgrs* reverses the process. In both cases, a return value of zero indicates success, non-zero indicates failure. Use CS_errmsg to obtain a description of the cause of failure.

```
int CScalcMgrsFromLl (struct cs_Mgrs_ *__This,char *result,int size,double latLng [2],int
prec);
int CScalcLlFromMgrs (struct cs_Mgrs_ *__This,double latLng [2],Const char *mgrsString);
```

In these prototypes, the first argument is a pointer to an initialized cs_Mgrs_ object obtained from one of the constructors described above. The **latLng** argument refers to an array of at least two doubles which contain the longitude and latitude (in that order) in degrees. The **prec** argument indicates the number of digits to be included in the resulting MGRS designation. Valid values range from 0 to 5. Note, that a value of 5 indicates that 5 easting, and 5 northing digits will be included in the resulting string. Of course, *CScalcMgrsFromLl* will never write more than **size** characters to the result array, and (assuming **size** is greater than zero) will cause result to be null terminated.

MGRS Private Functions

Two "private" functions, i.e. internal support functions, exist which convert geographic coordinates to UTM/UPS coordinates and zone number. These functions are aware of the missing/widened zones in the northern Europe region. They are also capable of switching between UTM and UPS (Universal Polar Stereographic) coordinates as appropriate. These functions use a **utmZone** variable which carries the UTM zone number where: 1) northern UTM zones are positive numbers between 1 and 60 inclusive, 2) southern UTM zones are negative numbers between –1 and –60 inclusive, 3) +61 refers to the North Pole UPS zone, 4) –61 refers to the South Pole UPS Zone,, and 5) the value zero is invalid and used to indicate an error condition.

```
int CScalcUtmUps (struct cs_Mgrs_ *__This,double utmUps [2],const double latLng [2]);
int CScalcLatLng (struct cs_Mgrs_ *__This,double latLng [2],const double utmUps [2],int
utmZone);
```

Given a geographic coordinate, *CScalcUtmUps* calculates the appropriate UTM/UPS coordinates and returns the appropriate **utmZone** value. Given a UTM/UPS coordinate and **utmZone** value, *CScalcLatLng* returns the appropriate geographic coordinate. *CScalcLatLng* returns zero on success, non-zero on failure. In all cases, the calculation is based on the ellipsoid used to construct the `cs_Mgrs_` argument to the function.

Forward/Inverse Functions

The term Forward/Inverse is a common way of referring to what is also known as the basic geodesy problem. That is, given a geodetic latitude/longitude, calculate the a new position given an azimuth an distance. This may sound pretty simple, but the calculation must be carried out on the ellipsoid. Thus the calculation is rather complex.

Forward refers to the calculation of a new geodetic position given an azimuth and a distance. Inverse refers to the calculation of an azimuth and distance given two geodetic positions.

CS_llazdd performs the forward calculation, and *CS_azddll* performs the inverse calculation.

Error Handling

Having it origins in the 'C' language, error reporting in the CS-MAP library is implemented using the return status methodology. That is, functions which can detect abnormal situations will return an integer status value to indicate the success of the operation intended, Status returns are of two types.

With the exceptions described below, all CS-MAP functions which can detect an abnormal situation will return a zero for success. A negative value will be returned for a failure which is considered fatal, and a non-zero positive value is returned for a warning or providing information about a remarkable condition.

While fatal errors can and do occur during the setup phase of a conversion/transformation combination, it is an important part of the design of CS-MAP that fatal errors cannot occur with conversion and transformation functions. Thus, any function involved in the creation and/or initialization of a conversion or transformation can be expected to return a negative status value. Functions which actually perform the conversion and/or transformation calculations can be expected to never return a negative status.

In the event of a negative status return, applications should immediately call the CS_errmsg function which will return an (8 bit character, English only) textual description of the cause of the problem. This description will often include, when appropriate, contextual information such as file names etc.

Status return values from calculation functions are always positive and non-zero values usually indicate that the coordinate to be converted is:

- Outside the useful range of the coordinate system being used,
- Outside the coverage area of a grid shift data file,
- Would have produced a domain error (i.e. log (-1))
- Or the coordinate is at either pole (which means any longitude is equivalent to any other).

In any case, the calculation function will indeed return a rational value. In many cases, this rational value will be produced by what is called a fallback technique.

In the experience of the developers of CS-MAP, once a conversion/transformation operation has been successfully constructed (i.e. setup), applications should at most simply count the number of non-zero positive status values returned by the calculation functions and report this number to the user (assuming it is non-zero) upon completion of the conversion. That is, a non-zero status return from any and all calculation functions should be considered as information only. Otherwise, your application will be bogged down and end user's can easily come to the conclusion that your application has crashed.

Also note, that to keep performance levels high, a non-zero positive status return value does not cause the generation of a descriptive error message and calling CS_errmsg after encountering a positive status return will produce misleading information, is anything at all.

Exception

There are several functions in CS-MAP whose function in life is to enumerate a list of entries in internal lists or dictionaries. These functions tend to return a positive 1 value to indicate success. A zero is returned to indicate that the end of the sequence has been encounters. (Of course, a negative return value indicates a fatal error of some sort. This convention was chosen so as to make obtaining such a list or enumeration rather easy to code in a robust manner:

```
int index;
char elpName [cs_KEEN_DEC};
.
.
.
index = 0;
while (CS_elEnum (index++,eleVate) > 0)
{
        /* Do something with this name */
        .
        .
        .
}
```

Data Structures

Discussions which follow refer to the primary data structures of CS-MAP. Twelve such data structures are described. These structure provide the basis for the operation of the Coordinate System Mapping Package. All structure definitions are found in the *cs_map.h* header file.

Ellipsoid Definition Structure

The Ellipsoid Definition structure, `cs_Eldef_`, carries the two principal data elements (among others) which define an ellipsoid for our purposes. These are the equatorial radius and the eccentricity of the ellipsoid. Among the other items contained in the structure is a key name, which is used to distinguish one ellipsoid definition from another.

Datum Definition Structure

The Datum Definition structure, `cs_Dtdef_`, carries the eight principal data elements (among others) which define a datum for our purposes. These are the key name of the ellipsoid definition upon which the datum is based, the X, Y, and Z components of the vector from the geocenter of the datum being defined to the geocenter of the WGS84 ellipsoid, the three rotation components of the transformation, and the scale component of the transformation. Among the other elements contained in this structure is a datum key name which is used to distinguish one datum definition from another.

Datum Composite Structure

The Datum Composite structure, `cs_Datum_`, carries the contents of both the Datum Definition structure and the Ellipsoid Definition structure in an composite form. This structure is never written to disk and is used internally as a programming convenience.

Coordinate System Definition Structure

The Coordinate System Definition structure, `cs_Csdef_`, carries all the elements required to define a coordinate system. Twenty four of these elements are referred to as projection parameters as their use depends upon the projection in use (which is one of the other data elements). Therefore, it is difficult to describe their use without delving into the specifics of each projection. (Refer to the descriptions of the functions associated with each projection in Chapter 4 for a description of parameter use for each projection.)

For our purposes here, let it be said that the `cs_Csdef_` structure is capable of carrying the definition of any coordinate system based on any one of the thirty eight projections supported by CS-MAP and (hopefully) any others which may be added in the future. This includes the parameters specific to the projection, the projection origin, the coordinate system units, the coordinate system scale, the false easting, the false northing, etc.

Preprocessed Projection Structures

There exists one structure for each of the thirty eight projections which carry the definition of a coordinate system based on the respective projection in a preprocessed form. That is, once the specific projection parameters applicable to a specific coordinate system are established, there are many calculations which can be performed independent of the specific coordinates to be converted. The results of these calculations are stored in these structures. The thirty five structure names are shown in Table III. It is the content of these structures which actually control the conversion of cartesian coordinates to and from latitudes and longitudes.

Coordinate System Parameter Structure

The Coordinate System Parameter Structure, `cs_Csprm_`, is used to carry a complete definition of a coordinate system and is the single structure used throughout the development of a coordinate conversion. It contains a copy of the `cs_Csdef_` structure of the coordinate system being used, a copy of the `cs_Datum_` structure which the coordinate system definition references, and the coordinate system in its pre-processed form as a union of the thirty eight pre-processed parameter structures described above. It also contains pointers to the functions which are capable of performing the forward and inverse coordinate conversions. Pointers are also included for grid scale and convergence angle functions. `cs_Csprm_` also includes information about the specific projection in use, as well as the limits of the coordinate system, both in cartesian and geographic form. While CS-MAP is designed such that the application should not need to know anything about the projection, there are instances (such as our own test program) where some knowledge of the projection in use, or its specific features, is helpful.

As a result, this single structure represents a complete definition of a coordinate system which can be easily passed around by pointer. Through the use of pointers to the appropriate coordinate conversion functions contain in this structure, modules which receive a pointer to this structure do not ever have to know exactly which projection is in use in order to perform coordinate conversions.

Projection Name Table Structure

The Projection Name Table Structure, `cs_Prjtab_`, is used solely to create a table of the projections known to the system. It primarily associates a name with a projection code and a setup function. To add new projections to the system, one need only create an entry in this table and reference the code which, of course, must also be written. You can also add additional names for existing projections by simply making additions to this table.

More importantly, to remove a projection from the system (in order to reduce the size of the text space within an executable, for example), one need simply remove (or comment out) the projection's entry in this table.

In developing a coordinate system parameter structure, the name of the projection is extracted from the coordinate system definition. This name is located in the projection table. The projection setup function associated with the selected named entry is then called and given a pointer to the union of pre-processed structures. The setup function initializes the union as if it were the pre-processed structure associated with the projection under construction. Of course, CS-MAP does all of this, mostly in the function named *CS_csloc*.

The Projection Name Table includes a fully descriptive name of each projection as well as a bit map of the features of the projection. Refer to *CSdataPJ* in Chapter 5 of this manual for full details.

Datum Shift Definition Structure

The Datum Shift Definition structure, `cs_Dtcprm_`, carries all of the information necessary to perform a datum shift on geographic coordinates (i.e. latitude and longitude). The information contained in this structure includes the definition of the source and target datums and a road map of the various conversions necessary to get, in the most accurate form, from the source datum to the target datum. This structure is allocated upon request given the definitions of the source and target coordinate systems. Once properly allocated, a pointer to this structure is all that the datum shift function needs in order to calculate datum shifts.

Thus, applications do not need to have any knowledge of what datums or how many different conversions are necessary to get from one datum to the next. In fact, quite often the conversion is the null conversion which implies that the source latitude and longitude are simply copied to the target array. Again, in this case, the application has no need to know of this situation.

The above is possible only because CS-MAP (to a large extent) requires that all datum definitions define how to convert a specific datum to/from WGS84. Thus, by "going through" WGS84 CS-MAP can convert any coordinate system/datum to any other. There are certain exceptions to this basic theme.

The Data Dictionaries

The Coordinate System Mapping Package includes the definition of more than 1,000 commonly used coordinate systems, more than 130 datum definitions, and 37 commonly referenced ellipsoids. These definitions are carried in the Coordinate System Dictionary (a file usually named *Coordsys*), the Datum Dictionary (a file usually named *Datums*), and the Ellipsoid Dictionary (a file usually named *Elipsoid*) respectively. These files are normally expected to reside in the *C:\MAPPING* directory (*/usr/MAPPING* under UNIX). The location of these files can be modified to suit your requirements at compile time (see *CSdata*) or at run time (see *CS_altdr*). The names of these files can be changed either at run time (see *CS_csfnm*, *CS_dtfnm*, and *CS_elfnm*) or compile time (see *cs_map.h*).

The Coordinate System Dictionary

The Coordinate System Dictionary is a fixed length record file of `cs_Csdef_` structures, maintained in sorted order by the `key_nm` element, i.e. the coordinate system key name. Entries in this file are located through the use of the binary search technique, therefore it is important that this file remain in sorted order. The file has a magic number in the first four bytes. This is a sequence of bytes which identify the file as being a Coordinate System Dictionary file and is defined by the **cs_CSDEF_MAGIC** manifest constant in *cs_map.h*. This value is checked each time the file is opened to make sure that the file is indeed a Coordinate System Dictionary and that it has not been seriously corrupted. The specific value of the magic number is changed each time the format of the `cs_Csdef_` structure is changed.

Functions are provided to access and maintain this file as a Coordinate System Dictionary. *CS_csopn* will open the file, verify its magic number, and return a file descriptor (or handle). *CS_csrd* and *CS_cswr* will perform sequential reads from and writes to a file of this type, handling encryption appropriately. *CS_cscmp* compares records in the file for sorting and searching purposes. *CS_csdef* will extract a particular record from the dictionary for you. *CS_csupd* will update an existing entry or add a new entry to the Coordinate System Dictionary, assuring that the file remains in sorted order. Finally, *CS_csdel* can be used to delete a record from the dictionary.

Coordinate system definitions are verified for validity before they are written to the Coordinate System Dictionary through the use of the *CS_cschk* function. *CS_csloc* (described below) also checks each definition before it is actually used to create the active form of a coordinate system.

The Datum Dictionary

The Datum Dictionary is a fixed length record file of `cs_Dtdef_` structures, maintained in sorted order by the `key_nm` element, i.e. the datum key name. Entries in this file are located through the use of the binary search technique, therefore it is important that this file remain in sorted order. The file has a magic number in the first four bytes of the file and is defined by the **cs_DTDEF_MAGIC** manifest constant in *cs_map.h*. This is a sequence of bytes which identify the file as being a Datum Dictionary file. This value is checked each time the file is opened to make sure that the file is indeed a Datum Dictionary and that it has not been seriously corrupted. The specific value of the magic number is changed each time the format of the `cs_Dtdef_` structure is changed.

Functions are provided to access and maintain this file as a Datum Dictionary. *CS_dtopn* will open the file, verify its magic number, and return a file descriptor (or handle). *CS_dtrd* and *CS_dtwr* will perform sequential reads from and writes to a file of this type, handling encryption appropriately. *CS_dtcmp* compares Datum Dictionary entries for sorting and searching purposes. *CS_dtdef* will extract a particular record from the dictionary for you. *CS_dtupd* will update an existing entry or add a new entry to the Datum Dictionary, assuring that the file remains in sorted order. Finally, *CS_dtdel* can be used to delete a record from the dictionary.

The Ellipsoid Dictionary

The Ellipsoid Dictionary is a fixed length record file of `cs_Eldef_` structures, maintained in sorted order by the `key_nm` element, i.e. the ellipsoid name. Entries in this file are located through the use of the binary search technique, therefore it is important that this file remain in sorted order. The file has a magic number in the first four bytes of the file and is defined by the **cs_ELDEF_MAGIC** manifest constant in *cs_map.h*. This is a sequence of bytes which identify the file as being a Ellipsoid Dictionary file. This value is checked each time the file is opened to make sure that the file is indeed an Ellipsoid Dictionary and that it has not been seriously corrupted. The specific value of the magic number is changed each time the format of the `cs_Eldef_` structure is changed.

Functions are provided to access and maintain this file as a Ellipsoid Dictionary. *CS_elopn* will open the file, verify its magic number, and return a file descriptor (or handle). *CS_elrd* and *CS_elwr* will perform sequential reads from and writes to a file of this type, handling encryption appropriately. *CS_elcmp* compares Ellipsoid Dictionary entries for sorting and searching purposes. *CS_eldef* will extract a particular record from the dictionary for you. *CS_elupd* will update an existing entry or add a new entry to the Ellipsoid Dictionary, assuring that the file remains in sorted order. Finally, *CS_eldel* can be used to delete a record from the dictionary.

Dictionary Encryption

The definitions of coordinate systems, datums, and ellipsoids can represent a significant investment on the part of the application developer. Under certain circumstances, a demonstration disk for example, the application developer may not wish to provide this information in a form from which this valuable data can be easily extracted. As a result, CS-MAP fully supports a means by which dictionary data can be encrypted. All CS-MAP functions will work equally as well with encrypted dictionaries as with normal versions. Dictionary compilers normally produce dictionaries in encrypted form. An option is provided to produce unencrypted dictionaries.

Dictionary Definition Protection

Dictionary entries are normally protected. That is, changes to coordinate system, datum, and ellipsoid definitions are controlled. This reduces technical support calls significantly. Application programmers can control the extent of protection, or turn it off altogether. How this system works is described below.

In normal operation, CS-MAP will not allow end users to change definitions distributed with the application. More specifically, definitions created by the Dictionary Compiler are marked as to be protected. End users can, and often do, create new coordinate system definitions. Therefore, rather than change a coordinate system as distributed with the application, users would typically create a new definition which is modified as necessary to achieve the desired results.

Definitions created by end users, and which remain unchanged for 60 days, are also protected. This is done under the assumption that a definition which remains unchanged for 60 days will have been used and judged satisfactory, and therefore should be preserved as a means of recording the actual definition used to produce the results.

Finally, CS-MAP normally requires that the key names for all user defined definitions contain the colon character. By adopting this convention, application updates can include coordinate system updates without the possibility of the update overwriting a valid and valuable user defined definition. (This, of course, assumes that the distribution will never contain a coordinate system definition with a key name containing a colon character.)

Application programmers can control to what degree the protection system is active by simply setting the value of either, or both, of two global variables, either at compile time (see *CSdata.c*) or at run time.

`char cs_Unique;` In the CS-MAP distribution, the value of this global variable is set to the colon. Set the value of this variable to the null character to turn the user definition key name protection feature described above off. You can select a character other than the colon by simply setting this variable to the desired character.

`short cs_Protect;` Use this variable to control the protect applied to dictionary definitions. A positive, non-zero value is the number of days associated with the user defined definition protection described above. For example, in the CS-MAP distribution, this value is set to 60, indicating that after a user defined definition remains unchanged for 60 days, it automatically becomes protected. Set `cs_Protect` to zero to maintain distribution coordinate system protection, but disable all user defined definition protection. Set `cs_Protect` to a negative value to disable all dictionary definition protection.

Programmer Note: Dictionary definitions include a `short` which controls the protection of the definition. If this value is set to zero, the dictionary entry is permanently unprotected. If this value is set to one, the entry is permanently protected. Otherwise, this value is set to the date at which the definition was last modified expressed as the number of days since January 1, 1990. Thus, changing the value of `cs_Protect` will change the protection of user defined definitions in a dynamic manner.

Byte Ordering

All three dictionary files described in this section contain data in binary form, thus byte ordering becomes a serious issue when using CS-MAP on different platforms. Beginning with Release 6.0 of CS-MAP, dictionaries are expected to be in little endian byte order regardless of the platform in use. A byte swapping function, *CS_bswap*, is called immediately after each read from any of these dictionaries, and immediately before the write of any data to any of the dictionaries. *CS_bswap* is programmed to automatically determine if byte swapping is required based on a compile time constant. Thus, a single copy of these dictionaries can be distributed for use on all platforms.

The term "all platforms" is perhaps misleading. *CS_bswap* will only swap between little endian and big endian byte orders. The rather odd byte orderings of some older DEC machines is not supported at the current time.

The automatic byte ordering feature is easy to disable if so desired. Refer to *CS_bswap* for more information.

The automatic byte ordering feature also applies to all other binary data files upon which CS-MAP relies. That is, automatic byte swapping is applied to all reads from NADCON database files, Multiple Regression data files, Canadian National Transformation files, and all HPGN database files. The choice of using little endian byte ordering was natural as these files are generally distributed by their sources in this form.

Dictionary Compiler

ASCII versions of the three dictionaries are provided in the distribution. The names of these files are *COORDSYS.ASC*, *DATUMS.ASC*, and *ELIPSOID.ASC*. Binary versions of the dictionary files can be produced by using the dictionary compiler program *CS_COMP*. This enables the definition files to be committed to version control procedures and the dictionaries remade as part of your product manufacturing process.

Originally, the ASCII files and the compilers were provided as a means of overcoming the byte order problem on different platforms. Now that CS-MAP has been modified to process little endian files on all platforms, this purpose is now obsolete. The version control purpose of these files remains valid.

Multiple Regression Datum Transformation Files

The compiler referred to above will also compile a fourth ASCII source file named *MREG.ASC* which is also supplied in the distribution. *MREG.ASC* contains in an ASCII, version controllable form, the definition of all multiple regression transformations known to the system. Compiling this file produces the *.MRT* files which CS-MAP accesses, as necessary, when performing datum conversions. Your application distribution should include the *.MRT* files produced by compiling the *MREG.ASC* file.

Default Datums, Ellipsoids, and Units

CS-MAP supports the concept of default datums, ellipsoids, and/or units. Defaults represent a convenient way to switch a coordinate system definition between different datums, ellipsoids, and/or units without having to change the Coordinate System Dictionary. How this feature applies to datums is described first; and this description is then extended to ellipsoids and units.

A datum reference in a coordinate system definition may be marked as "defaultable" by enclosing the name in square brackets. The actual datum name provided must be a valid datum reference as this reference will be used whenever the default feature is not active. This also implies that the default feature need not be active for the coordinate system definition to be valid and usable.

Upon activation of a coordinate system, regardless of the interface used, CS-MAP will check to see if the datum specified is "defaultable". If so, it examines the current default datum setting. If a valid default datum has been specified, the "defaultable" datum reference is replaced by the current default setting and coordinate system setup continues. If there is no current default setting, the "defaultable" datum is used as is. Thus, in the absence of a default specification, the coordinate system definition operates as defined. Whenever a default replacement is performed, the replaced datum name in the `cs_Csdef_` element of the `cs_Csprm_` structure will be enclosed in parenthesis to indicate that a default replacement has occurred.

Use the *CS_dtdflt* function to define a default datum. It will return the status and the name of any previous default. It will not allow an invalid default setting to be made. Calling *CS_dtdflt* with the **NULL** pointer as its argument can be used to determine if the datum default feature is active and, if so, what the current default setting is. Call *CS_dtdflt* with a pointer to the null string to disable the datum default feature. Until *CS_dtdflt* is called with a valid datum reference, the datum default feature remains disabled.

Cartographically referenced coordinate systems, and datum definitions, can contain "defaultable" ellipsoid references. Use *CS_eldflt* to enable and disable the "defaultable" ellipsoid feature.

Coordinate system unit specifications can also be "defaultable". Separate default values are maintained for linear and angular units. Use *CS_ludflt* to control the state of linear unit defaults, and *CS_audflt* to control the state of angular unit defaults.

You can completely ignore this concept of default datums, ellipsoids, and units by simply not calling (or even referencing) any of the default functions mentioned above. Default processing is off by default, and by never calling any of these functions, it never gets turned on. This is how most users deal with the default feature.

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.

The Functions

The thirteen functions which comprise the High Performance Interface are *CS_csloc*, *CS_dtcsu*, *CS_ll2cs*, *CS_dtcvt*, *CS_cs2ll*, *CS_dtcls*, *CS_csscl*, *CS_cnvrg*, *CS_cssch*, *CS_cssck*, *CS_llchk*, *CS_xychk*, and *CS_free*. The typical coordinate conversion application uses only seven of these function.

Refer to the major sections following this to see how the use of these functions is combined to produced generalized coordinate conversion capabilities.

Coordinate System Locate

Given the key name of a coordinate system defined in the Coordinate System Dictionary, *CS_csloc* returns a pointer to a fully initialized `cs_Csprm_` structure. This initialization includes all of the "one-time" calculations and the establishment of pointers to the appropriate coordinate conversion functions. This structure is *malloc*'ed from dynamic memory. Therefore, you may have several such definitions active at any given time. Also, you should release these structures when your application no longer has need of them; use *CS_free*.

Datum Conversion Setup

Given pointers to the both the source and target coordinate systems as returned by *CS_csloc*, *CS_dtcsu* returns a pointer to a *malloc*'ed `cs_Dtcprm_` structure which is a required argument to the *CS_dtcvt* function which actually calculates datum shifts. As its name implies, *CS_dtcsu* "sets up" a datum conversion by allocating and initializing a `cs_Dtcprm_` structure. Since datum conversions often require the use of file descriptors, grid cell caches, and the like, do not use *free* or *CS_free* to release the `cs_Dtcprm_` structure returned by *CS_dtcsu*. The sixth function of this interface, *CS_dtcls* must be used instead to prevent memory and file descriptor leaks in your application. Note, when appropriate, *CS_dtcsu* returns a pointer to a null datum conversion; a conversion which does nothing successfully and rapidly.

Coordinate System to Lat/Long Conversion

Given a pointer to an initialized `cs_Csprm_` structure which describes the coordinate system in use, the *CS_cs2ll* function will convert a cartesian coordinate to geographic form in terms of latitude and longitude in internal form. The resulting geographic coordinate will be based on the same datum as the coordinate system defined in the provided `cs_Csprm_` structure.

The conversion function pointers inserted into the *cs_Csprm_* structure by *CS_csloc* are used to select the proper code for the conversion. The application has no need to know what projection is in use or, for that matter, anything else about the coordinate system in use.

Datum Conversion

Given a pointer to an initialized datum conversion parameter structure, `cs_Dtcprm_`, as returned by *CS_dtcsu*, the *CS_dtcvt* function will convert the supplied geographic coordinates from the source datum to the target datum.

Lat/Long to Coordinate System Conversion

Given a pointer to an initialized `cs_Csprm_` structure, the *CS_ll2cs* function will convert a geographic coordinate specified in terms of latitude and longitude (in degrees) to the coordinates of the coordinate system defined by the `cs_Csprm_` structure.

The conversion function pointers inserted into the `cs_Csprm_` structure by *CS_csloc* are used to select the proper code for the conversion. The application has no need to know what projection is in use or, for that matter, anything else about the coordinate system in use.

Close Datum Conversion

Given a pointer to an initialized datum conversion parameter structure, as returned by the *CS_dtcsu* function, *CS_dtcls* will release all system resources allocated for the specific datum conversion. Note, that since several datum conversions may be initialized and operative at any given time, *CS_dtcls* does not necessarily release all resources associated with certain datum transformations until such time as the last datum conversion parameter block referencing such resources is closed.

Grid Scale Factor

Given a pointer to an initialized `cs_Csprm_` structure as returned by the *CS_csloc* function, *CS_csscl* will return the grid scale factor for the coordinate system at a location indicated by a geographic coordinate. It is important to note that the concept of Grid Scale as a single number applies only to coordinate systems based on conformal projections such as the Transverse Mercator, Lambert Conformal Conic, and the Oblique Mercator. Other projections, such as equal area projections, will have two such scale factors. In the case of equidistant projections, there are two such scale factors but one of them will usually be one. In the case on non-conformal projections, *CS_csscl* will return what the designers of CS-MAP consider the more interesting of the two scale factors for the specific projection involved.

Scale Along a Parallel

The two scale factors mentioned above consist of the scale along a parallel, often referred to as 'k', and the scale along a meridian, referred to as 'h'. Given a pointer to an initialized `cs_Csprm_` structure as returned by *CS_csloc*, *CS_cssck* will always return the scale factor along a parallel at the location provided.

The nomenclature referred to here is that used by J. P. Snyder in *Map Projections - A Working Manual*. Other authors use different symbology. In his later work, *Map Projections - A Reference Manual*, Snyder uses the more common notation. CS-MAP, whose origins date back to 1987, uses the original Snyder notation.

Scale Along a Meridian

Given a pointer to an initialized `cs_Csprm_` structure, *CS_cssch* returns the scale along a meridian, often referred to as 'h', at the location given by its second argument which must be a geographic coordinate.

Convergence Angle

Given a pointer to an initialized `cs_Csprm_` structure as returned by *CS_csloc*, *CS_cnvrg* returns the convergence angle of the coordinate system, at the location indicated by the second argument which must be a geographic coordinate. The return value is in degrees; positive is east of north.

Check Limits, Geographic

Given a pointer to an initialized `cs_Csprm_` structure as returned by *CS_csloc*, *CS_llchk* will determine if all coordinates in the list provided are within the mathematical domain of the coordinate system and/or with the useful range of the coordinate system. In the case where the provided list contains two or more points, the determination applies to each coordinate on each great circle arc formed by consecutive geographic coordinates. In those cases where the provided coordinate list consists of four or more geographic coordinates which define a closed region, the determination applies to all coordinates enclosed within the region, or which reside the provided boundary.

Check Limits, Cartesian

Given a pointer to an initialized `cs_Csprm_` structure as returned by *CS_csloc*, *CS_xychk* will determine if all coordinates in the list provided are within the mathematical domain of the coordinate system and/or with the useful range of the coordinate system. In the case where the provided list contains two or more points, the determination applies to each coordinate on each line segment formed by consecutive cartesian coordinates. In those cases where the provided coordinate list consists of four or more cartesian coordinates which define a closed region, the determination applies to all coordinates enclosed within the region, or which reside on the provided boundary.

Free Coordinate System Parameters

Use *CS_free* to free memory allocated by the any CS-MAP functions. This function is to be used, for example, to free the coordinate system parameter block returned by *CS_csloc*. It is important that *CS_free* be used as in certain environments (a Windows DLL for example), the heap used by the library is not necessarily the same as the heap used by the application. Thus, if CS-MAP allocated the memory, it is best if CS-MAP releases it also.

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 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 and prime meridian 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 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).

Multiple Conversions

Please note, that since coordinate system definitions (as returned by *CS_csloc*) reside in "heap" memory, there is no practical limit as to the number of definitions which can be active at any given time. Therefore, using the three functions described above, several different coordinate conversions can be active at the same time.

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

Geodetically Referenced Coordinate Systems

In the normal case, CS-MAP converts from one geodetically referenced system to another. In the sample code segment in the previous topic, for example, UTM27-13 is referenced to NAD27 and CO83-C is referenced to NAD83. The need for NAD27 to NAD83 conversion is unambiguous and performed automatically by CS-MAP's High Performance Interface without the application having to know of this situation. However, there are circumstances where a coordinate system cannot be referenced to a specific datum. For example, what datum should UTM zone 25 (middle of the Atlantic Ocean) be referenced to?

For various reasons, it is not always possible or convenient to reference a coordinate system to a specific datum. To handle such cases, CS-MAP supports the concept of a cartographically referenced coordinate system.

Cartographically Referenced Coordinate Systems

The `cs_Csdef_` structure, which carries the definition of all coordinate systems, has an ellipsoid key name element as well as a datum key name element. If the datum key name element is not the null string, the coordinate system is said to be geodetically referenced. If the datum key name element is the null string, the ellipsoid key name element must then carry the key name of an ellipsoid definition in the ellipsoid dictionary. In this case, the coordinate system is said to be cartographically referenced. (If both elements are not the null string, the ellipsoid key name field is ignored.)

The example shown previously in this section showed how conversions are performed between two geodetically referenced systems. If either of the two coordinate systems involved is cartographically referenced, or if both are cartographically referenced, *CS_dtcsu* simply returns the null datum conversion. Thus, for example, when the target coordinate system is cartographically referenced, the resulting coordinates are based on the source datum, whatever it may happen to be. Similarly, if the source is cartographically referenced and the target is geodetically referenced, there is an implied assumption that the source coordinates are based on the datum of the target. If both coordinate systems are cartographically referenced, we have no knowledge of the datums in either case and the conversion is strictly cartographic, hence the semantic convention adopted here.

Examination of the *COORDSYS.ASC* file will produce several cartographically referenced coordinate systems. Many of these are UTM zones in areas other than the US and Canada. (In the US and Canada, UTM zones can be reliably said to be based on either NAD27 or NAD83.) Perhaps the most important cartographically referenced coordinate system is that which is named LL. This enables us to use the LL coordinate system to convert generic Lat/Long's to any coordinate system, or convert any coordinate system to Lat/Long's based on the same datum as the source, what ever it might be.

Cartographic Projections

For each projection supported, there exist eight, possibly nine, functions which comprise the full implementation of a projection. The brief description of these functions is given below. These descriptions will be important to those adding a new projection as well.

Those adding projections to the system only need to add an entry to the projection table `cs_Prjtab`. Doing so, you will need to add a pointer to the Definition Check and the Setup functions. To activate the proper parameter settings in the MFC dialog, you will also need to add the projection to the `cs_PrjprmMap` table. It is unlikely, but if your new projection uses a new type of parameter, you may need to add a new entry to the `cs_Prjprm` table. All of these tables are defined in the *CSdataPJ.c* module.

Program Environments

Portability to a large variety of program environments is a major feature of CS-MAP. Thus, programmers accustomed to a single environment may consider some of the code and design of CS-MAP rather awkward or old fashioned. Nevertheless, by keeping things very basic and simple (e.g. binary searching a sorted fixed length record file), CS-MAP ports to just about any other environment without change and without requiring any additional software beyond the contents of a normal C runtime library.

However, simplicity is insufficient to cover the entire range of issues. In this section we discuss features and aspects of CS-MAP which are specifically intended for differing program environments.

Multi-Threaded Programming

Beginning with release 8.0, CS-MAP is fully compatible with multi-threaded Windows environments. Threads are different from processes in that not only do they share their parents code space, the data space is also shared. Since CS-MAP uses several global variables, this presents a problem.

This problem is addressed by declaring each of the several dynamically used (i.e. non-constant) global variables to be (using the Microsoft vernacular) *__declspec (thread)*. (In the Borland vernacular, its *__thread*.) This causes each new thread to have it's own copy of these variables; but the initialization of these variables upon starting a new thread is unclear. In any case, we do count on the operating system being able to give us a separate copy of all of these variables for each new thread instance which is started.

In order to insure that each of these variables is properly initialized, we have provided the function named *CS_init*. It should be called in each new thread just once, prior to any other CS-MAP function call in the thread. If the parent thread's value of the global *cs_Dir* and *cs_DirP* variables are valid, these values are preserved. If not they are initialized. In the case of several other variables, such as defaults, the application programmer may specify if these are to be inherited from the parent thread. Variables dealing with NAD27 to NAD83 datum conversions and the like are always initialized to the NULL state.

Thus, each thread will have its own set of NADCON data file buffers. In certain applications, this may be wasteful, but in most cases this provides the highest performance. Otherwise, we would be wasting considerable resources with resource locks etc.

UNIX users need not fear. All declarations and definitions in the CS-MAP code where the *__declspec (thread)* or *__thread* are appropriate are accomplished using the manifest constant Thread defined in the CS-MAP header file. This constant is defined to be nothing in most cases. Only in the event that compiler pre-defined constants indicate a multi-thread environment (i.e. _MT), is Thread defined to be something other than nothing. None of this applies if you are building a DLL. In a DLL, there is a single data segment, and multi-threading has no effect.

GUI Considerations

CS-MAP supports a graphic user interface based on Microsoft's MFC library. This is, admittedly, non portable but does provide useful product capability for a large percentage of our clients. Use of these functions requires that you include the *csmap.rc* file into your projects resources. This can be accomplished by adding an include "*csmap.rc*" statement to your project's .rc2 file. Please note that the use of these functions implies that your application activates the basic infrastructure of the MFC library. Accessing the functions in an isolated application does not work without special effort. Also note, that MFC is a multi-threaded environment. You will need to compile the CS-MAP library using the multi-threaded options to eliminate frustrations generated by the Microsoft linker.

The *CS_csEditor* function causes activation of the coordinate system editor. Similarly, the *CS_dtEditor* and *CS_ellEditor* functions cause the activation of the datum and ellipsoid editors respectively. In all three cases, the functions require a single input and return a single result. The input is the key name of the definition which is to be initially displayed. The result is the key name that was displayed when the user caused the dialog to exit. All other user activity is recorded in the appropriate dictionary.

The *CS_csBrowser* function activates a coordinate system browser and can be used to obtain a coordinate system key name from the user. The *CS_csDataDr* function displays a dialog which enables the user to specify the directory in which the mapping data files reside.

Customization

Some users will, no doubt, require modification to the basic capabilities of the Coordinate System Mapping Library as provided by OSGeo. The following sections describe how some of the more common requirements can be accomplished with a minimum of effort.

Tuning the Protection System

As distributed, CS-MAP will not allow users to modify or delete a dictionary definition which is produced by the dictionary compiler; i.e. a distribution definition. Further, CS-MAP will not permit modification or deletion of a user defined coordinate system after such definition has remained unaltered for 60 days. The behavior of this feature is controlled by the `cs_Protect` global variable, an *int* defined in the *CSdata* module. You can change the value of this variable in *CSdata* and recompile, or at run time.

Setting `cs_Protect` to a negative value disables all of the above described protection features. Setting `cs_Protect` to zero enables distribution coordinate system protection, but disables the user defined protection system. Setting `cs_Protect` to a positive value enables the user definition protection feature, and also specifies the number of days which must elapse from the last modification before the definition is protected.

Turning of Unique Names

As distributed, CS-MAP requires that a colon appear in all dictionary definition key names. By doing so, CS-MAP guarantees that the names of all user definitions will be different from any definition which may be contained in a future distribution of CS-MAP. You can disable this feature of CS-MAP by setting the global *char* variable named `cs_Unique` to the null character (i.e. '\0'). Alternatively, you can enable this feature using a different character by setting the value of `cs_Unique` to that character. `cs_Unique` is defined in the *CSdata* module and be set at compile time or run time.

Eliminating a Projection

If you do not have a need for all thirty eight supported projections, you can simply remove from the projection table (defined in *CSdataPJ*) the entry which references any projection which you do not need. Doing so will eliminate all references to the code for a specific projection and reduce the code size of your executable.

Data Dictionary Directory

The directory in which *CS_csdef*, *CS_dtdef*, and *CS_eldef* look for their respective dictionary files is defined in the *CSdata.c* module. You must use the *CS_altdr* function to initialize this variable to point to the directory which contains all data files. *CS_altdr* will use the value of an environmental variable when called with a NULL pointer as an argument. The name of this environmental variable, `CS_MAP_DIR` by default, is established in *cs_map.h* as a manifest constant.

Dictionary File Names

The names assigned to the three dictionary files are defined as manifest constants in the *cs_map.h* header file, declared and initialized in the *CSdata* module. They can also be modified at run time by using the *CS_csfnm*, *CS_dtfnm*, and *CS_elfnm* function.

Adding Units

The units which are recognized by the Coordinate System Mapping Package are defined in the *CSdataU* module. You can add or delete as necessary. Note that this table has provisions for an abbreviation in addition to the full name. Use the code as provided as an example of how to incorporate a new unit. Also note that the factor is the multiplier required to change the new unit to meters, or degrees, by multiplication depending upon the type of unit.

Language Translation

Textual descriptions of all error conditions are provided in the *CSerpt* module. All language oriented text is located either in the *cs_map.h* header file, one of the three data modules: *CSdata*, *CSdataU*, *CSdataPJ*, or in the ASCII form of the dictionary files (*COORDSYS.ASC*, *DATUMS.ASC*, and *ELIPSOID.ASC*). Language translations efforts need only address these eight files (MFC dialogs excepted).

CHAPTER 3

Chapter 3 -- Executables

This section describes the use and function of the three executable modules which are a part of the CS-MAP distribution. Note that the executables modules themselves are not provided. Source code which can be used to create the executable modules on your platform are included.

CS_COMP—Coordinate System COMPiler

```
CS_COMP [/c] [/b] [/s] [/t] [/w] source_file_dir output_dir
```

CS_COMP creates binary dictionary files from the ASCII source files provided with the CS-MAP distribution. This feature has been added to 1) eliminate problems with the byte order of binary data on platforms other than Intel and 2) provide a means by which coordinate system definitions can be committed to source control procedures. Release 6 (and later) of CS-MAP precludes, to a large extent, the first need described above. See Byte Ordering below.

On UNIX systems, you will need to use the UNIX option character (i.e. the dash) when specifying options.

Source code to *CS_COMP* and its components is provided to all licensees, and has been tested as a console application under Windows XP and Linux 3.2.2. As CS-MAP is intended for use in a large variety of systems, and for compilation and linking by users who may not have access to *lex* and *yacc*, the user interface and source file formats are very simple and basic.

By compiling all four source files at the same time, *CS_COMP* can now perform consistency checks between all four files. To simplify use, the actual file names are hard coded into the program. Thus, the source files are required to be named *Coordsys.asc*, *Datums.asc*, *Elipsoid.asc*, and *Mreg.asc* for the Coordinate System, Datum, Ellipsoid, and Multiple Regression Transformation data files respectively. *CS_COMP* expects these files to reside in the directory specified as the first positional argument on the command line.

Similarly, the names of the output files are fixed by the program, being the names specified in the *CSdata* module. These are the same names that the library in general searches for. The directory in which these files are written is specified as the second positional parameter on the command line.

Since *CS_COMP* performs a consistency check between all files, it expects to compile all four files. It is not possible to compile the files individually.

CS_COMP is coded for possible compilation as a Windows XP console application. Therefore, it requires an acknowledgment before it exits, enabling the operator to verify successful completion before the window disappears. For use in make files and/or batch files, you may wish to use the **/b** option to suppress the requirement for acknowledgment before exit.

CS_COMP normally produces encrypted output files. Use the **/c** option to cause unencrypted versions of the dictionaries to be produced for testing and/or debugging purposes.

Presence of the **/t** option instructs *CS_COMP* to include coordinates systems, datums, and ellipsoids in the Test group. Normally, these are neither compiled or distributed with your application.

Use the **/w** option to instruct *CS_COMP* to report any inconsistency in the data which may only be suspicious.

In the UNIX environment, the dash must be used as the option character.

Use the **/s** option to instruct *CS_COMP* to produce dictionary files in big endian byte order regardless of the system on which the program has been compiled. This forces a byte swap before output on little endian processors (e.g. Intel) and omits the byte swap on big endian processors (e.g. Sun).

Byte Ordering

CS_COMP calls *CS_bswap* immediately prior to writing any data to the dictionary file in order to effect, as necessary, a byte order switch to little endian (i.e. Intel/DOS) byte order. This is the same byte order expected by the CS-MAP library on all platforms. It is also the byte order which the data file from other sources, such as the NADCON LAS/LOS files, are distributed. Use of the **/s** options reverses this effect, causing big endian data files to be produced, regardless of the processor in use.

Source File Formats

Coordsys.asc

Records in the *COORDSYS.ASC* source file consist of a line of text. Blank lines are generally ignored. All characters following an un-escaped pound sign character ('#') are ignored. The pound sign character can be escaped by the backslash ('\\') or pound sign ('#') characters. Note that this is required to get a pound sign character into a coordinate system description as these descriptions are not quoted. Leading and trailing whitespace on all records is also ignored. Records to be processed must start with one of the 39 keywords described below. The colon separator character is considered to be part of the keyword. The value of the keyword must follow the keyword on the same line of text, white space may be used to separate the keyword from its value.

Each occurrence of the CS_NAME: keyword indicates the beginning of a new coordinate system definition and the end of any previous coordinate system definition. Otherwise, the order of the specifications is not important. However, to maintain your source in a comprehensible format, it is strongly recommended that each coordinate system definition begin with the *CS_NAME:* keyword and be terminated by one or more blank lines. No blank lines should appear within the definition itself. Further, while *CS_COMP* will sort the resulting binary Coordinate System Dictionary file, maintaining the coordinate system source file in sorted order makes it (relatively) easy to locate a definition should review or editing be required.

For projection specific information, refer to the projection descriptions in Chapter 7 of this guide. Note that rarely will any projection require the use of all 39 keyword specifications.

The 39 keywords and their values are:

CS_NAME: - Used to specify the key name of the coordinate system which is to be defined, 23 characters max. See *CS_nampp* for conventions concerning key names.

DESC_NM: - Used to specify the 63 character descriptive name of the coordinate system being defined.

DT_NAME: - Used to specify the datum key name to which a geodetically referenced coordinate system is to be referenced to. The ellipsoid used is a part of this datum definition. Presence of a valid datum key name here indicates that the coordinate system is geodetically referenced. Either a **DT_NAME:** specification, or an **EL_NAME:** specification must be provided for each coordinate system definition. If both **DT_NAME:** and **EL_NAME:** specifications are provided, the **EL_NAME:** specification is ignored by CS-MAP.

EL_NAME: - Used to specify the ellipsoid key name for a cartographically referenced coordinate system. A coordinate system is cartographically referenced to the ellipsoid named by this specification if, and only if, the datum key name specification is omitted. If both **DT_NAME:** and **EL_NAME:** specifications are provided, the **EL_NAME:** specification is ignored by CS-MAP.

ORG_LAT: - Used to specify the origin latitude of the coordinate system. This value may be in any form acceptable to *CS_atof*, but is always in degrees relative to the equator. Use positive numbers for north latitude, negative numbers for south latitude.

ORG_LNG: - Used to specify the origin longitude of the coordinate system. This value may in any form acceptable to *CS_atof*, but always in degrees and is always relative to the Greenwich prime meridian. Use positive numbers for east longitude, negative numbers for west longitude.

SCL_RED: - Used to specify the scale reduction which may apply to a coordinate system. This value is ignored by the many projections which do not support this feature. The value may be specified as a decimal number, e.g. 0.9996, or as a ratio, e.g. 1:2500. A value of 1.0 or greater is unusual, but is accepted.

ZERO_X: - Used to specify the minimum X value which is to be considered non-zero. X coordinate values whose absolute value is less than the value specified here will be converted to hard zeros. This is used to suppress coordinate output such as 4.3472E-07 which can be of value in certain applications. A value of 0.0 is assumed if no specification is made.

ZERO_Y: - Used to specify the minimum Y value which is to be considered non-zero. Y coordinate values whose absolute value is less than the value specified here will be converted to hard zeros. This is used to suppress coordinate output such as 4.3472E-07 which can be of value in certain applications. A value of 0.0 is assumed if no specification is made.

PARM1: thru PARM24: - Used to specify the value of as many as 24 parameters which are specific to the particular projection in use. The use of these items varies from one projection to another. The value is always a real number. Where a longitude is specified, it must be given in degrees, relative to Greenwich, where west longitude is negative. Where a latitude is expected, it must be given in degrees relative to the equator where north latitude is positive and south latitude is negative. When an azimuth is specified, it must be given in degrees east of north (i.e. west of north would be negative). In all cases, the values are processed by *CS_atof*, and any form acceptable to that function may be used.

X_OFF: - Used to specify the value of the false easting of the coordinate system. A value of 0.0 is assumed if no specification is made. Any form acceptable to *CS_atof* may be used, including the use of the comma as a thousands separator.

Y_OFF: - Used to specify the value of the false northing of the coordinate system. A value of 0.0 is assumed if no specification is made. Any form acceptable to *CS_atof* may be used, including the use of the comma as a thousands separator.

PROJ: - Used to specify the code name of the projection upon which the coordinate system is based. This code value must be the value assigned to the desired projection in the *CSdataPJ* module, i.e. the projection table. This value is a character string of two to eight characters. Since projections can now have several variations, this code value is not the same as the five character code used to generate function and structure tag names. You will need to refer to the *CSdataPJ* module to determine the code name for a specific projection type.

UNIT: - Used to specify the name of the system unit for the coordinate system being defined. In the case of a normal cartesian coordinate system, this must be one of the supported unit of length names as defined in the *CSdataU* module. In the case of the Unity projection, i.e. the coordinate system being defined is latitude and longitude or a variation thereof, the unit name must be one of those names defined as a unit of angular measure in the *CSdataU* module.

GROUP: - Used to classify coordinate systems into groups to make selection of a coordinate system from the 5,000+ provided a bit easier. The supported group codes are defined in the *CSdata* module.

SOURCE: - Used to specify the source of the information used to define this coordinate system, 63 characters maximum. The term Authority is often used to describe this information.

QUAD: - Used to indicate the quadrant of the cartesian coordinates produced by the coordinate system. Zero or 1 indicate the normal right handed cartesian system where X increases to the east, and Y increases to the north. Quadrants are numbered counterclockwise, therefore a value of 2 specifies a cartesian system where X increases to the west, while Y increases north. A value of 3 indicates that X increases to the west and Y increases to the south. A value of 4 indicates that X increases to the east and Y increases to the south. A negative value will cause the axes to be swapped after the appropriate quadrant is applied. A value of 1 is assumed if this specification is absent.

MIN_LNG: - This parameter is optional and can be used to specify the minimum longitude of the useful range of the coordinate system. The value is given in degrees relative to Greenwich in any form acceptable to *CS_atof*. Positive values indicate east longitude, while negative values indicate west longitude. Its value should be normalized between -360 and +360 and when used must be algebraically less than the MAX_LNG: parameter.

MAX_LNG: - This parameter is optional and can be used to specify the maximum longitude of the useful range of the coordinate system. The value is given in degrees relative to Greenwich in any form acceptable to *CS_atof*. Positive values indicate east longitude, while negative values indicate west longitude. Its value should be normalized between -360 and +360 and when used must be algebraically greater than the MIN_LNG: parameter.

MIN_LAT: - This parameter is optional and can be used to specify the minimum latitude of the useful range of the coordinate system. The value is given in degrees relative to the equator in nay form acceptable to *CS_atof*. Positive values indicate north latitude while negative values indicate south latitude. Its value should be normalized between -90 and +90 and when used must be algebraically less than the MAX_LAT: parameter.

MAX_LAT: - This parameter is optional and can be used to specify the maximum latitude of the useful range of the coordinate system. The value is given in degrees relative to the equator in nay form acceptable to *CS_atof*. Positive values indicate north latitude while negative values indicate south latitude. Its value should be normalized between -90 and +90 and when used must be algebraically greater than the MIN_LAT: parameter.

MIN_XX: - This parameter is optional and can be used to specify the minimum X coordinate value of the useful range of the coordinate system. The value is given in system units and may be provided in any form acceptable to *CS_atof*. Its value must be algebraically less than the MAX_XX: parameter.

MAX_XX: - This parameter is optional and can be used to specify the maximum X coordinate value of the useful range of the coordinate system. The value is given in system units and may be provided in any form acceptable to *CS_atof*. Its value must be algebraically greater than the MIN_XX: parameter.

MIN_YY: - This parameter is optional and can be used to specify the minimum Y coordinate value of the useful range of the coordinate system. The value is given in system units and may be provided in any form acceptable to *CS_atof*. Its value must be algebraically less than the MAX_YY: parameter.

MAX_YY: - This parameter is optional and can be used to specify the maximum Y coordinate value of the useful range of the coordinate system. The value is given in system units and may be provided in any form acceptable to *CS_atof*. Its value must be algebraically greater than the MIN_YY: parameter.

Other key words have been coded into the *CS_COMP* module but are reserved for future use by OSGeo contributors. They should not be used until their exact usage is defined in a future release.

Datums.asc

Records in the *DATUMS.ASC* source file consist of a line of text. Blank lines are ignored. All characters following an un-escaped pound sign character ('#') are ignored. The pound sign character can be escaped by the backslash ('\\') or pound sign ('#') characters. Note that this is required to get a pound sign character into a datum description as these descriptions are not quoted. Leading and

trailing whitespace on all records is also ignored. Records to be processed must start with one of the twelve keywords described below. The colon separator character is considered to be part of the keyword. The value of the keyword must follow the keyword on the same line of text, white space may be used to separate the keyword from its value.

Each occurrence of the **DT_NAME:** keyword indicates the beginning of a new datum definition and the end of any previous datum definition. Otherwise, the order of the specifications is not important. However, to maintain your source in a comprehensible format, it is strongly recommended that each coordinate system definition begin with the **DT_NAME:** keyword and be terminated by one or more blank lines. No blank lines should appear within the definition itself. Further, while *CS_COMP* will sort the resulting binary Datum Dictionary file, maintaining the datums source file in sorted order makes it (relatively) easy to locate a definition should review or editing be required.

Also note, that CS-MAP uses the Datum Key name as the base portion of a file name to access the Multiple Regression Transformation file. Therefore, a datum key name of more than 8 characters on a DOS based system may not work as expected.

The twelve keywords and their required values are:

DT_NAME: - Used to specify the key name of the datum which is to be defined, 23 characters max. Refer to *CS_nampp* for conventions concerning key names. Since datum key names are used to link to the Multiple Regression Transformation files, datum key names longer than 8 characters may present problems on systems using the DOS FAT-16 file system (i.e. file name limited to 8 characters).

DESC_NM: - Used to specify the 63 character descriptive name of the datum being defined.

ELLIPSOID: - Used to specify the key name of the ellipsoid definition upon which this datum is based. This name must be the key name of an entry in the Ellipsoid Dictionary.

USE: — This parameter is required and is used to specify the datum conversion technique to convert coordinates based on the datum being defined to WGS84 coordinates. There are, currently, ten valid values. They are:

1 MOLODENSKY - Use the Molodensky transformation to convert to WGS84.

2 GEOCENTRIC - Use the geocentric translation method to convert to WGS84.

3 BURSA - Use the Bursa/Wolfe Seven Parameter Transformation to convert to WGS84.

4 7PARAMETER - Use the Seven Parameter Transformation to convert to WGS84, default to Molodensky if the necessary parameters are not present.

5 MULREG - Use the Multiple Regression Transformation formulas. If such a definition is not available, default to the Bursa/Wolfe Seven Parameter Transformation.

6 NAD27 - Use the NADCON or Canadian National Transformation emulation as appropriate to convert to NAD83, and consider the result to be WGS84 coordinates.

7 NAD83 - Consider the coordinates to be WGS84 coordinates already, no shift is to be performed.

8 WGS84 - The coordinates are WGS84 coordinates already, no datum shift is required.

9 WGS72 - Use an internal formula to convert to WGS84.

10 HPGN - Use the NADCON algorithm, but use the HPGN data files, to shift the coordinates to NAD83, then consider the result to be WGS84 coordinates without any further datum shift.

DELTA_X: - The X component of the vector from the geocenter of this datum to the geocenter of the WGS-84 datum in meters.

DELTA_Y: - The Y component of the vector from the geocenter of this datum to the geocenter of the WGS-84 datum in meters.

DELTA_Z: - The Z component of the vector from the geocenter of this datum to the geocenter of the WGS-84 datum in meters.

BWSCALE: - The scale of the Bursa/Wolfe or Seven Parameter Transformation, given as parts per million as is ordinarily the case for this transformation. This value can be positive or negative. The actual resulting scale factor used in the transformation is 1.0 + (`BWSCALE` * 1.0E-06).

ROT_X: - The rotation about the X axis, given in seconds of arc. Positive values indicate clockwise rotation of the right handed Helmert coordinate system.

ROT_Y: - The rotation about the Y axis given in seconds of arc. Positive values indicate clockwise rotation of the right handed Helmert coordinate system.

ROT_Z: - The rotation about the Z axis given in seconds of arc. Positive values indicate clockwise rotation of the right handed Helmert coordinate system.

SOURCE: - The source of information from which this definition was developed.

Support of other keywords has been coded into the *CS_COMP* module; but use of these is reserved by OSGeo contributors for future use.

Elipsoid.asc

Records in the *ELIPSOID.ASC* source file consist of a line of text. Blank lines of text are ignored. All characters following an un-escaped pound sign character ('#') are ignored. The pound sign character can be escaped by the backslash ('\\') or pound sign ('#') characters. Note that this is required to get a pound sign character into an ellipsoid description as these descriptions are not quoted. Leading and trailing white space on all records is also ignored. Records to be processed must start with one of the five keywords described below. The colon separator character is considered to be part of the keyword. The value of the keyword must follow the keyword on the same line of text, white space may be used to separate the keyword from its value.

Each occurrence of the `EL_NAME:` keyword indicates the beginning of a new ellipsoid definition and the end of any previous ellipsoid definition. Otherwise, the order of the specifications is not important. However, to maintain your source in a comprehensible format, it is strongly recommended that each ellipsoid definition begin with the `EL_NAME:` keyword and be terminated by one or more blank lines. No blank lines should appear within the definition itself. Further, while *CS_COMP* will sort the resulting binary Ellipsoid Dictionary file, maintaining the ellipsoid definition source file in sorted order makes it (relatively) easy to locate a definition should review or editing be required.

The five keywords and their required values are:

EL_NAME: - Used to specify the key name of the ellipsoid which is to be defined, 23 characters max.

DESC_NM: - Used to specify the 63 character descriptive name of the ellipsoid being defined.

E_RAD: - Used to specify the equatorial radius of the ellipsoid being defined. This radius must be specified in meters.

P_RAD: - Used to specify the polar radius of the ellipsoid being defined. This radius must be specified in meters. Use the same value given for `E_RAD:` to define a spherical ellipsoid.

SOURCE: - The source of the information used to make this definition.

GROUP: - This keyword is used to mark ellipsoid definitions as being for testing only. Additional groups may be established in the future.

CS_COMP will calculate the eccentricity and flattening from the provided radii.

MReg.asc

Records in a *MREG.ASC* source file consist of a line of text. Blank lines are ignored. All characters following an un-escaped pound sign character ('#') are ignored. The pound sign character can be escaped by the backslash ('\\') or pound sign ('#') characters. Note that this is required to get a pound sign character into a datum description as these descriptions are not quoted. Leading and trailing white space on all records is also ignored. Records to be processed must start with one of the 12 keywords described below. The colon separator character is considered to be part of the keyword. The value of the keyword must follow the keyword on the same line of text, white space may be used to separate the keyword from its value. In certain cases, the actual keyword contains numeric qualifiers which indicate the specific power series term the keyword applies to.

Each occurrence of the `DATUM_NAME:` keyword indicates the beginning of a new multiple regression definition and the end of any previous multiple regression definition. The order of the specifications is not important. However, to maintain your source in a comprehensible format, it is strongly recommended that each multiple regression definition begin with the `DATUM_NAME:` keyword and be terminated by one or more blank lines. No blank lines should appear within the definition itself. Maintaining the multiple regression source file in sorted order makes it (relatively) easy to locate a definition should review or editing be required.

Please note that a test case is required for each datum. A binary multiple regression coefficient data file will not be written unless the provided coefficients satisfy the provided test case. In the terminology used in this module, `LAMBDA` refers to longitude, and `PHI` refers to latitude. In preparation for future enhancements to CS-MAP, the `HEIGHT` coefficients are also required. If such are not currently available, simply use values which will produce a zero change in height to get around the required coefficient check.

Values for the various keywords are given in a manner which is somewhat inconsistent with CS-MAP. However, the manner in which these values are specified is consistent with the conventions used in the source for most of this type of information: DMA TR-8350.2B.

The 12 keywords and their required values are:

DATUM_NAME: - Used to specify the 23 character name of the datum for which the following coefficients represent the multiple regression coefficients. This is the name, with the .MRT extension appended, which is given to the coefficient data file. The association of a datum in the Datum Dictionary and the multiple regression data file is established by this name.

TEST_LAMBDA: - This keyword must be followed by the longitude of the test point. This longitude must be given in degrees, minutes, and seconds form, where west longitudes are given as values greater than 180. This entry is processed by a "%d %d %lf" *sscanf* format specification.

TEST_PHI: - This keyword must be followed by the latitude of the test point. This latitude must be given in degrees, minutes, and seconds form. Use a negative value to indicate south latitude. This entry is processed by a "%d %d %lf" *sscanf* format specification.

DELTA_LAMBDA: - This keyword must be followed by a single real value which represents the amount of longitude shift, in seconds of arc, which is expected at the provided test point.

DELTA_PHI: - This keyword must be followed by a single real value which represents the amount of latitude shift, in seconds of arc, which is expected at the provided test point.

DELTA_HEIGHT: - This keyword must be followed by a single real value which represents the amount of elevation shift, in meters, which is expected at the provided test point.

LAMBDA_OFF: - This keyword must be followed by the longitude offset used to normalize the coefficient formula. This value must be given in decimal degrees, use negative values for west longitude.

PHI_OFF: - This keyword must be followed by the latitude offset used to normalize the coefficient formula. This value must be given in decimal degrees, use negative values for west longitude.

KK - This keyword must be followed by the scale factor which is used to normalize the regression formula. This value is unitless.

LAMBDA - This keyword is used to identify a longitude formula coefficient. The keyword itself must be followed by a Un and a Vn sequence which indicates which coefficient follows. For example,

```
LAMBDA U1 V2: 1.23456
```

indicates that 1.23456 is the coefficient for the longitude times latitude squared term in the regression formula. CS-MAP does not support terms with powers higher than 9.

PHI - This keyword is used to identify a latitude formula coefficient. The keyword itself must be followed by a Un and a z sequence which indicates which coefficient follows. For example,

```
PHI U2 V1: 1.23456
```

indicates that 1.23456 is the coefficient for the longitude squared times latitude term in the regression formula. CS-MAP does not support terms with powers higher than 9.

HEIGHT - This keyword is used to identify an elevation formula coefficient. The keyword itself must be followed by a Un and a z sequence which indicates which coefficient follows. For example,

```
HEIGHT U0 V0: 1.23456
```

indicates that 1.23456 is the constant term in the regression formula. CS-MAP does not support terms with powers higher than 9.

TEST -- TEST program

```
TEST [/t12345...] [/dmap_dir] [/pnn] [/s] [/v] [/b] [test_data_file_name]
```

TEST will exercise most (but not all) of the functions included in the Coordinate System Mapping Package library. It can be used to verify the correct operation of the library in different environments, especially useful after compiling the library with a new or different C compiler or on a different platform.

Since *TEST* relies on the test coordinate systems, datums, and ellipsoids, *CS_COMP* should be run with the **/t** option prior to using the *TEST* program.

The program requires no arguments and normally expects the Ellipsoid Dictionary, the Datum Dictionary, the Coordinate System Dictionary, all Multiple Regression Transformation files, and Geodetic Data Catalogs to reside in their default locations and the file *TEST.DAT* to reside in the current working directory when executed. As a convenience, however, if a file named *COORDSYS* exists in the same directory from which *TEST* was executed, *TEST* will look to that directory for all supporting data files (except the *TEST.DAT* file). Alternatively, you may use the **/d** option to specify the directory you want *TEST* to look to for all supporting data files.

TEST will write all diagnostic messages to the console screen. On Windows 95/98/NT systems, you will need to use the MS-DOS option character (i.e. the forward slash) when specifying options.

The provided test file, *TEST.DAT*, includes tests for the NAD27 to NAD83 datum conversion software. Therefore, the NADCON *CONUS* database file system must exist in the default data directory (see *CSdata*(5CS)) if these tests are to be successful. *TEST.DAT* also includes tests for the Canadian National Transformation. However, since OSGeo cannot distribute the data files associated with the Canadian National Transformation, these tests have been commented out. Canadian users may wish to uncomment these tests before using *TEST*.

The command line options can be used to modify the operation of the test procedure. There are, currently, 16 separate tests performed by *TEST*, and each is designated with a number or a letter. Normally, *TEST* performs each of the first fifteen tests twice; first in forward numeric order, and then in reverse numeric order. Use the **/t** option to specify the specific test(s) you would like performed, and the order in which they are to be performed; one test per character (120 maximum).

Individual Tests

The nature of the 16 tests are:

Test 1 - This test manipulates the Ellipsoid Dictionary using *CS_eldef*, *CS_elupd*, and *CS_eldel*.

Test 2 - This test manipulates the Datums Dictionary using *CS_dtdef*, *CS_dtupd*, and *CS_dtdel*.

Test 3 - This test manipulates the Coordinate System Dictionary using *CS_csdef*, *CS_csupd*, and *CS_csdel*.

Test 4 - This test reads the file named *TEST.DAT* in the current directory and performs all of the conversions indicated, comparing the calculated results with the expected results recorded in the file. Of course, all discrepancies are reported to the user. You may specify an alternate file name (and directory) on the command line as the only positional argument. Each supported projection (except the Equidistant Cylindrical) has at least one test from a source other than CS-MAP in *TEST.DAT*. *TEST.DAT* also includes tests of datum conversions.

Test 5 - Test 5 is a performance test. Normally, it records the amount of wall clock time necessary to make 300,000 conversions from "UTM27-13" to "CO83-C" using the High Performance Interface. Each conversion, therefore, includes an inverse Transverse Mercator, a NAD27 to NAD83 datum shift, and a forward Lambert Conformal Conic conversion. The test cycles through a list of 10 different coordinate pairs to add some reality to the test without distorting the numeric results with I/O time and/or system overhead. The elapsed time and the effective conversion rate are reported to the user. The **/p** option can be used to change the number of conversions in the test. For example, **/p45** would instruct *TEST* to perform 450,000 conversions whenever it performs test number 5.

Test 6 - This test exercises the sorting and binary search functions of CS-MAP well beyond what would be experienced in normal use. This is accomplished by sorting the Coordinate System Dictionary into reverse order, binary searching the result, and resorting back into normal order. Finally, the order of the result is verified to be correct.

Test 7 - This test exercises the *CS_csgrp* function and its supporting data.

Test 8 - For each coordinate system in the Coordinate System Dictionary, this test will cause a coordinate to be converted in both the forward and inverse direction, as well as calculate the grid scale factor and the convergence angle. This test does not verify the accuracy of the results, but simply assures that every calculation function is exercised at least once. This test is somewhat superfluous now that Test C is available.

Test 9 - This test exercises the functions which are used to calculate the power series solutions to the elliptical integrals used quite frequently in CS-MAP. It verifies the results against an outside source, and then compares forward and inverse calculations with each other. Please note that the external source for the correct values is limited in precision. Therefore the RMS discrepancy values may be alarmingly high. This is not the case, however, as indicated by the RMS discrepancies between forward and reverse calculations.

Test A - This test is identical to test 4 in every way except it uses the High Level Interface function *CS_cnvrt* function for all conversions, thus testing the caching system for coordinate systems and datum conversions.

Test B - Test B tests the grid scale and convergence angle functions of all non-azimuthal projections. It uses an empirical technique to determine the grid scale and convergence angle of several random points within the useful range of each coordinate system. Azimuthal projections are skipped as the grid scale functions in these cases usually return scale factors along and normal to radials from the origin. An empirical means of calculating these scale factors eludes us at the current time.

Test C - This test tries very hard to produce a floating point exception. Very regular and randomly generated coordinate values, both geographic and cartesian, are generated and passed to all functions for each coordinate system in the coordinate system dictionary. The coordinates are, in most cases, completely outlandish numbers. Reporting floating point exceptions is very difficult, however, varying from compiler to compiler, system to system. However, CS-MAP has passed this test many times, on four different compilers.

Test D - This test tests the forward and inverse functions of each projection against each other. That is, random geographic coordinates within the useful range of each coordinate system are converted to cartesian form using the forward function, and then back to geographic using the inverse function. The results are then compared.

Test E - This test performs all the functions of Test D; but in this case the useful range of each of the projections is reduced by about one half, and thus enabling the error tolerance to be substantially reduced.

Test F - Test F tests the *CS_atof* and *CS_ftoa* functions in two phases. First, the standard system function *sprintf* is used to check the operation of the *CS_atof* function. In the second phase, *CS_atof* is used to test the *CS_ftoa* function. Neither phase, currently, tests the operation of the degree, minute. and/or second processing.

Test G - Test G is the coordinate creep test. Creep is defined as the number of millimeters a coordinate moves after repeated conversions from cartesian to geographic and back. Test G starts with a specific cartesian coordinate well within the useful range of a projection (but certainly not the natural origin) and converts the coordinate to geographic and back to cartesian 1,000 times. The distance between the original coordinate and the final result is calculated in millimeters to arrive at the creep value. A creep value greater than 10 is considered a failure. As of release 8.01, only the four most used projections are tested.

Test S - This is not really a test, *per se*, but a request that the test program switch its mode with regard to byte ordering. Thus, on an little endian processor such as Intel, the initial S in a test sequence will cause all binary data files to be swapped to big endian byte ordering and the *CS_bswap* module adjusted to cause the necessary byte swaps for the program to function. Thus, all byte swapping mechanisms can be tested on a single processor. Since the byte swap algorithm is its own inverse, a second occurrence of Test S reverses the effect of a previous execution. Note, including an odd number of S specifications in the test specification will leave all dictionary files in a swapped condition.

Test V - This is not a test. Occurrence of a V in the test string simply toggles the verbose flag. Thus, the verbose flag can be turned off , or on, for specific tests individually.

Test Z - This is not a test. When *TEST* encounters this character in the test sequence, it simply starts the test sequence again from the beginning. Thus, an infinite loop can be established. Typically, a CONTROL-C is used to terminate the program at some point.

Test Data

The TEST.DAT file specifies the actual coordinates which are to be converted and the expected results. *TEST* ignores empty lines in the file and lines which begin with the pound sign (#) character. Other records are expected to contain 8 fields separated by commas. The eight fields are expected to contain:

1 Key name of the coordinate system of the coordinates specified in fields 2 and 3 of this line. That is, the name of the source coordinate system of the test conversion to be performed.

2 & 3 The X and then Y coordinates to be converted. These are expected in decimal form and are converted to binary using the *CS_atof* function of the CS-MAP library. Thus a variety of forms can be used.

4 Key name of the coordinate system to which the coordinates given in fields 2 & 3 are to be converted.

5 & 6 The expected X and Y coordinates of the conversion. Again, any form acceptable to *CS_atof* can be used.

7 & 8 The X and Y tolerance within which the converted values must agree with the expected values. If the converted values do not match the expected values within the specified tolerance, a diagnostic message is printed.

Most conversion examples provided in *TEST.DAT* were obtained from reputable sources other than CS_MAP itself. Comments in the *TEST.DAT* file itself will indicate those specific tests which have not been verified with sources outside other than CS-MAP. The tolerance values given in the provided *TEST.DAT* file should not be considered as an indication of the accuracy or precision of the CS-MAP library. Rather, these values usually indicate the accuracy and precision of the source data from which the examples were obtained. Occasionally, the tolerance values do indicate the accuracy of the CS-MAP library; comments in the *TEST.DAT* file indicate when this is the case.

Other Command Line Options

Finally, the **/v** option can be used to cause *TEST* to operate in verbose mode. In this mode, *TEST* will report its progress through each test. In order to enable use of *TEST* as a Windows NT console application, *TEST* normally requires an acknowledgment when all tests are complete. The **/b** option can be used to suppress this feature. The **/s** option can be used to instruct *TEST* to start out in big endian mode; useful when testing the automatic byte swapping feature. Use the **/d** option to provide *TEST* with the full path to the directory containing the binary dictionary files it is to use. In the absence of this option, *TEST* uses the directory encoded into the *CSdata* module; except that if a valid *COORDSYS* file exists in the directory from which *TEST* was executed, that directory is used (MS-DOS only). Use the **/p** option to indicate the length of the performance test (i.e. Test 5). The value provided is multiplied by 10,000 to obtain the number of conversions which are performed and timed in order to produce the conversion rate.

BUGS

Test C, the floating point exception test, does not exercise the datum conversion functions as yet.

mfcTEST -- MFC Dialog TEST

`mfcTEST`

mfcTest is designed to test the MFC based GUI interactive dialogs provided with CS-MAP. The program is a simple dialog box MFC program which contains a menu. The single menu entry provides a selection for each of the primary dialogs provided. Note also, that the Test dialog can be very convenient for testing projection and datum shift calculations in an interactive environment (assuming you're using Windows, of course).

This test program requires no arguments. All testing must be done in an interactive manner. Note, that each of the dialogs has a help button, and expects to find the help file in the same directory as the primary mapping data files. The help buttons are grayed out if the help file does not exist in this location.

Dictionary Differences Program

The source to Dictionary Difference program, `DictDiff`, is conveyed in a file named *CS_DictDiff.c*; the distribution places this file in the *Dictnary* directory. The main module calls functions named CS_csDiff, CS_dtDiff, and CS_elDiff which are defined in a module named CSdictDiff.c which the distribution deposits in the *Source* directory. These three functions are a part of the normal library build.

Messages which report differences refer to "was" and "is". That is, messages report the previous value and the new value for all detected changes.

`DictDiff` is a command line program and can used in virtually any environment that supports a C compiler. It compares the binary forms of dictionary files and reports all differences detected. It requires exactly two positional arguments. The first command line argument is the directory containing the "was" or previous dictionary files. The second argument is the directory of the "is" or current dictionary files. All messages are reported to `stdout`, i.e. the terminal.

In producing the differences, some tolerance numbers had to be chosen to delineate what a change consists of. You should examine the source code in file named *CSdiffDict.c* to verify that you are comfortable with the tolerance values that were chosen. The tolerance values are manipulated throughout the program, but the variable name `okValue` is consistent.

Chatper 4 -- Library Functions

This section includes a technical description of 500+ functions in the CS-MAP library. The descriptions are organized by the interface of which they are a part. An index is provided elsewhere in this document.

High Level Interface Functions

Functions described in this section are designed to be called from high level languages such as Visual Basic. Therefore, descriptions of most functions in this section also include a function declaration appropriate for use in Visual Basic and Delphi in addition to the standard C prototype.

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_atof Ascii TO Floating point

```
Function CS_atof (ByRef result As Double,ByVal value As String) As Long
function CS_atof (var result :double;value: PChar) :LongInt;
long CS_atof (double *result,Const char *value);
```

CS_atof will convert the ASCII, null terminated string provided by the **value** argument to double precision floating point form, returning this result in the location pointed to by the **result** argument. Obviously, the string provided by **value** is expected to be an ASCII representation of a numeric value.

CS_atof has several features built into it for handling numeric formats that are commonly used in mapping, specifically, large numbers, and values in degrees, minutes, and seconds format. Use of thousands separators is supported and, when present, their improper use is reported. Other than leading white space, spaces in the input value are interpreted to indicate degrees, minutes, and seconds format. Values can be entered using minutes only (a single space) or minutes and seconds (two spaces encountered). Use of either directional characters (i.e. N, S, E, W) or plus and hyphen characters for sign is also supported. *CS_atof* returns a long that carries a complete specification of the format used to enter the value, suitable for use by *CS_ftoa* for formatting the value for output. *CS_atof* will also correctly process scale factors entered as ratios, and this feature can be mixed with the thousands separator feature. Thus, scale reduction for state plane coordinate systems can be entered as "1:17,000."

The return value is a bitmap of information used to contain precision, formatting specifications, and error status values. The following descriptions refer to constants defined in the various header files. Construct a format specification by inclusively or'ing the desired options:

cs_ATOF_PRCMSK	The least significant five bits are used to indicate the number of digits found after the decimal point. The value is actually the number of digits plus one. (Zero is reserved to indicate automatic precision determination on output.) This constant is a mask that will mask out the precision value.
cs_ATOF_VALLNG	The value processed is acceptable for a longitude value.
cs_ATOF_VALLAT	The value processed is acceptable for a latitude value.
cs_ATOF_MINSEC	The value processed was in degrees, minutes, and seconds form.
cs_ATOF_MINUTE	The value processed was in degrees and minutes form.
cs_ATOF_EXPNT	The value processed was in scientific notation form.
cs_ATOF_COMMA	The value processed included thousands separators to the left of the decimal point.
cs_ATOF_DIRCHR	The value processed included directional characters to indicate sign.
cs_ATOF_XEAST	The directional characters used to indicate the sign came from the E and W set, as opposed to the N and S set.
cs_ATOF_MINSEC0	The value processed included leading zeros in the minutes or seconds fields.
cs_ATOF_DEG0	The value processed included leading zeros in the degrees field.
cs_ATOF_0BLNK	The value processed was the null string.
cs_ATOF_FORCE3	The value processed used minutes or minutes and seconds format, and 3 digits of degrees were encountered; implying a longitude value.
cs_ATOF_RATIO	The processed value was provided in the form of a ratio, e.g. 1:2500, to indicate a value such as, for example, 0.9996.

Bits defined by the following constants are set to indicate the associated error condition. The **cs_ATOF_FMTERR** bit is set if any error condition is detected and forces the return value to negative. In all such cases, *CS_atof* will report the error condition and a subsequent call to *CS_errmsg* will return an appropriate error message.

cs_ATOF_SECS60	What was interpreted to be the seconds field of the processed value produced a value greater than or equal to 60.
cs_ATOF_MINS60	What was interpreted to be the minutes field of the processed value produced a value greater than or equal to 60.
cs_ATOF_MLTPNT	More than one decimal point was encountered in the input value.
cs_ATOF_MLTSGN	More than one sign indication was encountered in the input value.
cs_ATOF_ERRCMA	Improper positioning of the thousands separator character was detected in the input value.
cs_ATOF_RATERR	A string that contained the ratio character, usually ':', did not conform to the normal convention for a ratio. Usually, the character immediately left of the colon was not a '1'.
cs_ATOF_FMTERR	A general format error, not covered by the above, was encountered in the input value.
cs_ATOF_ERRFLG	This bit is set, producing a negative return value, if any of the above error conditions are encountered during processing. Whenever this bit is set, the error condition will have been reported to *CS_erpt*, and a subsequent call to *CS_errmsg* will produce an appropriate error message.

CS_azddll LatLong Azimuth Distance calculator

```
Function CS_azddll (ByVal e_rad As Double, ByVal e_sq As Double,
                    ByRef ll_from As Double,
                    ByVal azimuth As Double,
                    ByVal distance As Double,
                    ByRef ll_to As Double) As Integer
function CS_azddll (e_rad,e_sq :Double; var ll_from :Double;
                    azimuth, distance :Double;
                    var ll_to :Double) :Integer;
int CS_azddll (double e_rad,double e_sq,double ll_from [3],
               double azimuth,
               double dist,
               double ll_to [3]);
```

CS_azddll calculates the latitude and longitude of a target point given an initial point, an azimuth from the initial point, and a distance. The initial point and the result are in degrees, where the longitude occupies the first element in the array and latitude the second element. The reference of the longitude is immaterial, as both (the initial point and the calculated point) will share the same reference whatever it is. Currently, the third element in each array is unused (i.e. un-referenced and unmodified). This may change in future releases. **e_rad** is the equatorial radius, and **e_sq** the eccentricity squared, of the ellipsoid to be used. The units of the radius are immaterial other than they must be the same as that used to specify the **distance**. **Azimuth** is the azimuth at the initial point given in degrees east of north. **distance** is the distance traveled in the same units as used to specify the equatorial radius of the ellipsoid. The result is returned in the array pointed to by the **ll_to** argument.

CS_azddll returns a zero to indicate success, -1 for failure. Failure of the internal Newton Rhapson iterative calculation is the only possible cause of failure. This can be caused by rather strange input values, specifically values that would produce results that are antipodal to the initial point.

The algorithm used is known as: "Solution of the geodetic inverse problem after T. Vincenty modified Rainsford's Method with Helmert's elliptical terms," whatever all that means. This algorithm is appropriate for any combination of points that are not antipodal.

CS_azsphr AZimuth on a SPHeRe

```
Function CS_azsphr (ByRef ll_1 As Double, ByRef ll_2 As Double) As Double
function CS_azsphr (var ll_1, ll_2 :Double) : Double;
double CS_azshpr (Const double ll0 [2],Const double ll1 [2]);
```

CS_azsphrl returns the azimuth, in degrees east of north, from the geographic location given by **ll0** to the geographic location given by **ll1**. The calculation assumes a spherical earth, so a radius and eccentricity is not required.

CS_cnvrg CoNVeRGence function

```
Function CS_cnvrg (ByVal cs_name As String, ByRef ll_ary As Double) As
Double
function CS_cnvrg (cs_name :PChar;var ll_ary :double) :Double;
double CS_cnvrg (Const char *cs_name,Const double ll_ary [2]);
```

CS_cnvrg returns the convergence angle of the coordinate system whose key name is provided by the **cs_name** argument, at the location provided by the **ll_ary** argument. The position provided by the **ll_ary** argument must be in longitude and latitude form, in degrees, where the first element of the array is the longitude and the second element of the array is the latitude. Use negative values for west longitude and south latitude. The returned value is in degrees, east of north.

CS_cnvrg uses the same cache of coordinate system definitions as does *CS_cnvrt*, therefore, the performance penalty of using this very simple function is not as great as one might expect.

ERRORS

CS_cnvrg will return a value of -360.0 (clearly a bogus value for a convergence angle) if an error is detected during the calculation. The most common cause of errors is an invalid coordinate system name.

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_csEnum Coordinate System ENUMerator

```
Function CS_csEnum (ByVal index As Integer,ByVal key_name As String,
                    ByVal size As Integer) As Integer
function CS_csEnum (index :Integer;key_name :Pchar;size :Integer) :Integer;
int CS_csEnum (int index,char *key_name,int size);
```

CS_csEnum is used to enumerate all coordinate systems in the Coordinate System Dictionary. *CS_csEnum* returns in the memory buffer pointer to by the **key_name** argument the key name of the **index**'th entry in the Coordinate System Dictionary. *CS_csEnum* will never write more than **size** bytes to the indicated location. **Index** is a zero based index; the index of the first entry in the Coordinate System Dictionary is zero.

CS_csEnum returns a positive 1 to indicate success. If **index** is too large, a zero is returned.

ERRORS

CS_csEnum will return a -1 and set `cs_Error` appropriately if any of the following conditions are encountered:

cs_CSDICT	The Coordinate System Dictionary could not be found or otherwise opened. See *CS_altdr*.
cs_IOERR	A physical I/O error occurred in accessing the Coordinate System Dictionary.
cs_CS_BAD_MAGIC	The file assumed to be the Coordinate System Dictionary by virtue of its name was not a Coordinate System Dictionary; it had an invalid magic number or was of an incompatible release level.
cs_INV_INDX	The index argument was negative.

CS_csIsValid Coordinate System key name Is Valid

```
Function CS_csIsValid (ByVal key_name As String) As Integer
function CS_csIsValid (key_name :PChar) :Integer;
int CS_csIsValid (Const char *key_name);
```

CS_csIsValid is used to validate coordinate system key names. *CS_csIsValid* returns a positive 1 if **key_name** is a valid coordinate system key name, a zero if not.

ERRORS

CS_csEnum will return a -1 and set `cs_Error` appropriately if any of the following conditions are encountered:

cs_CSDICT	The Coordinate System Dictionary could not be found or otherwise opened. (See *CS_altdr*.)
cs_IOERR	A physical I/O error occurred in accessing the Coordinate System Dictionary.
cs_CS_BAD_MAGIC	The file assumed to be the Coordinate System Dictionary by virtue of its name was not a Coordinate System Dictionary; it had an invalid magic number. This can also be caused by an incompatible release.

CS_csRangeEnum Coordinate System Useful Range Enumerator

```
Function CS_csRangeEnum (ByVal index As Integer,ByVal csKeyName As String,
                         ByVal size As Integer) As Integer
function CS_csRangeEnum (index :Integer;csKeyName :Pchar;size :Integer)
:Integer;
int CS_csRangeEnum (int index,char *csKeyName,int size);
```

CS_csRangeEnum is used to enumerate all coordinate systems which were located by the last call to the *CS_csRangeEnumSetup* function. Using these two functions, it is possible to obtain a list of only those coordinate systems whose useful range include a specific point. *CS_csRangeEnum* returns in the memory buffer pointer to by the **key_name** argument the key name of the **index**'th entry in the list generated by *CS_csRangeEnumSetup*. *CS_csRangeEnum* will never write more than **size** bytes to the indicated location. **Index** is a zero based index; the index of the first entry in the Coordinate System Dictionary is zero.

CS_csEnum returns a positive 1 to indicate success. If **index** is too large, a zero is returned. A negative value is returned for a serious error, such as failure to call CS_csRangeEnumSetup prior to calling this function.

CS_csRangeEnumSetup Coordinate System Range Enumeration Setup

```
Function CS_csRangeEnumSetup (ByVal longitude As DOuble,
                              ByVal latitude As Double) As Integer
function CS_csRangeEnumSetup (longitude, latitutde :Double) : Integer
int CS_csRangeEnumSetup (double longitude,double latitude);
```

Use this function to set the base location for subsequent *CS_csEnumRange* useage. That is, use this function to produce (internally) a list of all coordinate systems whose useful range includes the given location. Essentially, this function will generate the list, and the application programmer then uses the *CS_csEnumRange* function to enumerate the list.

The location is specified in geographical terms (i.e. latitude and longitude). These values must be in degrees, relative to Greenwich. Since datum differences are on the order of, at most, a few hundred meters, the datum upon which these coordinates are based is immaterial for the purpose of this function.

CS_csRangeEnumSetup will return a negative value in the event of a serious error, such as beung unable to access the Coordinate System Dictionary. Use *CS_errmsg* to get a textual description of the error which can be reported to the application user. Otherwise, *CS_csRangeEnumSetup* will return the number of coordinate systems located, which can be zero.

Finally, note that the *CS_recvr* function will recover all allocated resources, including the list of coordinate systems generated by the last call to this function.

CS_dtEnum DaTum ENUMerator

```
Function CS_dtEnum (ByVal index As Integer,ByVal key_name As String,
                    ByVal size As Integer) As Integer
function CS_dtEnum (index :Integer;key_name :Pchar;size :Integer) :Integer;
int CS_dtEnum (int index,char *key_name,int size);
```

CS_dtEnum is used to enumerate all datums in the Datum Dictionary. *CS_dtEnum* returns in the memory buffer pointer to by the **key_name** argument the key name of the **index**'th entry in the Datum Dictionary. *CS_dtEnum* will never write more than **size** bytes to the indicated location. **Index** is a zero based index; the index of the first entry in the Datum Dictionary is zero.

CS_dtEnum returns a positive 1 to indicate success. If **index** is too large, a zero is returned.

ERRORS

CS_dtEnum will return a -1 and set `cs_Error` appropriately if any of the following conditions are encountered:

cs_DTDICT	The Datum Dictionary could not be found or otherwise opened. (See *CS_altdr*.)
cs_IOERR	A physical I/O error occurred in accessing the Datum Dictionary.
cs_DT_BAD_MAGIC	The file assumed to be the Datum Dictionary by virtue of its name was not a Datum Dictionary; it had an invalid magic number. This can also be caused by a dictionary file of an incompatible release.
cs_INV_INDX	The index argument was negative.

CS_dtIsValid DaTum key name Is Valid

```
Function CS_dtIsValid (ByVal key_name As String) As Integer
function CS_dtIsValid (key_name :PChar) :Integer;
int CS_dtIsValid (Const char *key_name);
```

CS_dtIsValid is used to validate datum key names. *CS_dtIsValid* returns a positive 1 if **key_name** is a valid datum key name, a zero if not.

ERRORS

CS_dtIsValid will return a -1 and set `cs_Error` appropriately if any of the following conditions are encountered:

cs_DTDICT	The Datum Dictionary could not be found or otherwise opened. (See *CS_altdr*.)
cs_IOERR	A physical I/O error occurred in accessing the Datum Dictionary.
cs_DT_BAD_MAGIC	The file assumed to be the Datum Dictionary by virtue of its name was not a Datum Dictionary; it had an invalid magic number. This can also be caused by a dictionary file of an incompatible release.

CS_elEnum ELlipsoid ENUMerator

```
Function CS_elEnum (ByVal index As Integer,ByVal key_name As String,
                    ByVal size As Integer) As Integer
function CS_elEnum (index :Integer;key_name :Pchar;size :Integer) :Integer;
int CS_elEnum (int index,char *key_name,int size);
```

CS_elEnum is used to enumerate all ellipsoids in the Ellipsoid Dictionary. *CS_elEnum* returns in the memory buffer pointer to by the **key_name** argument the key name of the **index**'th entry in the Ellipsoid Dictionary. *CS_elEnum* will never write more than **size** bytes to the indicated location. **Index** is a zero based index; the index of the first entry in the Ellipsoid Dictionary is zero.

CS_elEnum returns a positive 1 to indicate success. If **index** is too large, a zero is returned.

ERRORS

CS_elEnum will return a -1 and set `cs_Error` appropriately if any of the following conditions are encountered:

cs_ELDICT	The Ellipsoid Dictionary could not be found or otherwise opened. (See *CS_altdr*.)
cs_IOERR	A physical I/O error occurred in accessing the Ellipsoid Dictionary.
cs_DT_BAD_MAGIC	The file assumed to be the Ellipsoid Dictionary by virtue of its name was not an Ellipsoid Dictionary; it had an invalid magic number. Note that dictionary magic numbers can be different for different releases.
cs_INV_INDX	The index argument was negative.

CS_ellsValid ELlipsoid key name Is Valid

```
Function CS_elIsValid (ByVal key_name As String) As Integer
function CS_elIsValid (key_name :PChar) :Integer;
int CS_elIsValid (Const char *key_name);
```

CS_elIsValid is used to validate ellipsoid key names. *CS_elIsValid* returns a positive 1 if **key_name** is a valid ellipsoid key name, a zero if not.

ERRORS

CS_elIsValid will return a -1 and set `cs_Error` appropriately if any of the following conditions are encountered:

cs_ELDICT	The Ellipsoid Dictionary could not be found or otherwise opened. (See *CS_altdr*.)
cs_IOERR	A physical I/O error occurred in accessing the Ellipsoid Dictionary.
cs_EL_BAD_MAGIC	The file assumed to be the Ellipsoid Dictionary by virtue of its name was not a Ellipsoid Dictionary; it had an invalid magic number. Note that magic numbers can be different for different releases.

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 it doesn't.

CS_erpt Error RePorT

```
extern int cs_Error,cs_Errno;
void CS_erpt (int err_num);
```

CS_erpt is called by all functions in the Coordinate System Mapping Package whenever an error condition is detected. The value of **err_num** indicates the specific error condition detected and must be one of the manifest constants defined in *cs_map.h*.

At the current time, *CS_erpt* does nothing other than set the value of global variable `cs_Error` to the supplied value of **err_num** and set the global variable of `cs_Errno` to the current value of the system's global variable `errno`.

It is expected that users will want to write their own *CS_erpt* function that will properly inform the operator of the nature of the problem encountered.

Each function in the Coordinate System Mapping Package is programmed to clean up after itself after return from *CS_erpt*. That is, upon return from *CS_erpt*, all memory *malloc*'ed by the function detecting the error is *free*'ed and any temporary file created by the function detecting the error is removed.

CS_fast FAST mode

```
Sub CS_fast (ByVal fast As Integer)
procedure CS_fast (fast :Integer);
void CS_fast (int fast);
```

CS_fast can be used to improve the performance of applications using the High Level Interface. When incorporated into a DLL, the High Level Interface normally verifies the veracity of each pointer argument provided by the application. This is convenient, of course, but also somewhat time consuming. Calling *CS_fast* with a non-zero value for the **fast** argument will disable this checking. It is recommended that calling *CS_fast* be added to your application only after it has been debugged. Fast mode can be turned off by calling *CS_fast* with argument **fast** set to zero.

CS_ftoa Floating point TO Ascii

```
Function CS_ftoa (ByVal buffer As String,ByVal size As Integer,ByVal value
As Double,
                  ByVal format As Long) As Long
function CS_ftoa (buffer :Pchar; size :Integer;value :Double;format
:Longint) :Longint;
long CS_ftoa (char *buffer,int size,double value,long format);
```

CS_ftoa formats the double precision floating point value provided by the **value** argument into ASCII form returning the result in the character array pointed to by the **buffer** argument. The result is always a null terminated string, and the length of the string is never longer than **size** - 1 characters. The format of the character string is controlled by the **format** argument. *CS_ftoa* returns a long that indicates the actual format used to format the value. The returned format specification may be different from the requested format if the buffer provided was not large enough, or if the requested format is not appropriate for the value provided.

CS_ftoa is intended to be a generalized formatting function that accommodates the formats commonly used in mapping. That is, large numbers and values in degrees minutes and seconds form. The somewhat awkward **format** argument is designed such that the value returned by *CS_atof* is suitable for use by *CS_ftoa*.

The original intent behind the design of the format specification was to enable users to indicate the desired format of output by simply entering a suitable value in the form they desire. The application would then use *CS_atof* to convert the value to binary form. If no errors occurred during the conversion, the returned long could then be used to format output. Experience will determine the success of this approach.

The **format** argument is a bitmap of information used to contain precision, formatting specifications, and error status values. The following descriptions refer to constants defined in the various header files. Construct a format specification by inclusively or'ing the desired options.

cs_ATOF_PRCMSK	The least significant five bits are used to indicate the number of digits to be produced after the decimal point. The value is actually the number of desired digits plus one. Zero indicates that the precision is to be calculated automatically. This constant is a mask that will mask out the precision value.
cs_ATOF_MINSEC	Output is to be in degrees, minutes, and seconds form.
cs_ATOF_MINUTE	Output is to be in degrees and minutes form.
cs_ATOF_EXPNT	This bit is set in the returned value if *CS_ftoa* had to resort to scientific notation in order to format the value in the space provided.
cs_ATOF_OVRFLW	This bit is set in the returned value if *CS_ftoa* could only output the overflow indication (i.e. *.*) in the space provided.
cs_ATOF_COMMA	Output is to include thousands separators to the left of the decimal point as appropriate.
cs_ATOF_DIRCHR	Output is to include directional characters to indicate the sign of the numbers rather than plus or minus signs.
cs_ATOF_XEAST	Meaningful only when the **cs_ATOF_DIRCHR** bit is set. Indicates that character set used to indicate positive or negative are E and W as opposed to N and S.
cs_ATOF_MINSEC0	Output is to include leading zeros in the minutes and seconds fields instead of leading spaces.
cs_ATOF_DEG0	Output is to include leading zeros in the degrees field rather than spaces.
cs_ATOF_0BLNK	Output a null string if the provided value is zero.
cs_ATOF_FORCE3	Used to force at least three character output in the degree field. Usually used when formatting a longitude.
cs_ATOF_RATIO	Output the result in a ratio format, e.g. 1:2500. Can be used in conjunction with **cs_ATOF_COMMA** to get something like 1:2,500.

CS_geoctrSetUp GEOCenTRic setup

```
Function CS_geoctrSetUp (ByVal ellipsoid As String) As Integer
function CS_geoctrSetUp (ellipsoid :PChar) :Integer;
int CS_geoctrSetUp (const char *ellipsoid);
```

Use this function to specify the ellipsoid definition that is to be used in geocentric coordinate calculations. The **ellipsoid** argument must be the key name of an ellipsoid defined in the ellipsoid dictionary. Zero is returned on success, -1 on error. Errors are usually caused by invalid ellipsoid names.

CS_geoctrGetXyz GEOCenTRic GET XYZ

```
Function CS_geoctrGetXyz (ByRef xyz As Double,ByRef llh As Double) As
Integer
function CS_geoctrGetXyz (var xyz, llh :Double) :Integer;
int CS_geoctrGetXyz (double xyz [3], double llh [3]);
```

Given the geographic coordinates of a point via the **llh** argument, *CS_geoctrGetXyz* returns the corresponding geographic coordinate in the array indicated by the **xyz** argument. Use the *CS_geoctrSetUp* to specify the ellipsoid that is to be used in the calculation. Note that the returned geocentric coordinates will be in meters, and the third element of the **llh** argument is considered to be the ellipsoidal height in meters. *CS_geoctrGetXyz* returns zero on success and –1 on failure. Failure can be caused by failing to specify an ellipsoid by calling *CS_geoctrSetUp*, or providing a bogus set of geographic coordinates.

CS_geoctrGetLlh GEOCenTRic GET LatLongHgt

```
Function CS_geoctrGetLlh (ByRef llh As Double,ByRef xyz As Double) As
Integer
function CS_geoctrGetLlh (var llh, xyz :Double) :Integer;
int CS_geoctrGetLlh (double llh [3], double xyz [3]);
```

Given the geocentric coordinates of a point via the **xyz** argument, *CS_geoctrGetLlh* returns the corresponding geographic coordinate in the array indicated by the **llh** argument. Use the *CS_geoctrSetUp* to specify the ellipsoid that is to be used in the calculation. Note that the geocentric coordinates must be in meters, and the height (i.e. the third element of the **llh** result) is the ellipsoidal height in meters. *CS_geoctrGetLlh* returns zero on success and –1 on failure. Failure can be caused by failing to specify an ellipsoid by calling *CS_geoctrSetUp*, or providing a bogus set of geocentric coordinates.

CS_getCountyFips Get County Federal Information Processing Standard code

```
Function CS_getCountyFips (ByVal stateFips As Integer,ByVal countyName As
String) As Integer
function CS_getDataDirectory (stateFips :Integer; countyName :PChar)
:Integer;
int CS_getCountyFips (int stateFips,Const char* countyName);
```

This function returns the Federal Information Processing Standard code value assigned to a county indicated by the **countyName** argument. This is appropriate for the US only. You can obtain the appropriate value for the **stateFips** argument by using *CS_getStateFips*. Note, that **countyName** must be the complete offical name of the county without any punctuation. The lookup procedure is NOT case sensitive. The function returns zero if the information provided by the two arguments fails to produce a county code.

CS_getDataDirectory GET DATA DIRECTORY

```
Function CS_getDataDirectory (ByVal data_dir As String,
                              ByVal dir_sz As Integer) As Integer
function CS_getDataDirectory (data_dir :PChar; dir_sz :Integer) :Integer;
int CS_getDataDirectory (char *data_dir,int dir_sz);
```

CS_getDataDirectory will return in the character array pointed to by the **data_dir** argument the full path to the directory it is searching for supporting data file. It will always return a null terminated string, but never write more than **dir_sz** characters to the array. *CS_getDataDirectory* will return **TRUE** if the directory returned does indeed contain a Coordinate System Dictionary file (i.e. a file named COORDSYS).

CS_getDatumOf Get Datum of a Coordinate System

```
Function CS_getDatumOf (ByVal csKeyName As String,ByVal datumName As
String,ByVal size As Integer) As Integer
function CS_getDatumOf (csKeyName, datumName  :PChar; size :Integer)
:Integer;
int CS_getDatumOf (Const char *csKeyName,char *datumName,int size);
```

Use this function to obtain the key name of the datum assigned to the coordinate system whose key name is provided by the **csKeyName** argument. The datum key name is returned in the string pointed to by the **datumName** argument. `CS_getDatumOf` will never write more than **size** characters to the **datumName** string. A zero value is returned for success, and -1 for failure. Failure is almost always caused by providing an invalid coordinate system key name. The string at **datumName** will be the empty string if the coordinate system referred to is cartographically referenced (i.e. referenced directly to an ellipsoid).

CS_getDescriptionOf Get Description of a Coordinate System

```
Function CS_getDescriptionOf (ByVal csKeyName As String,
                              ByVal description As String,
                              ByVal size As Integer) As Integer
function CS_getDescriptonOf (csKeyName, description :PChar; size :Integer)
:Integer;
int CS_getDescriptionOf (Const char *csKeyName,char *description,int size);
```

Use this function to obtain the description the coordinate system whose key name is provided by the **csKeyName** argument. The description is returned in the string pointed to by the **description** argument. `CS_getDescriptionOf` will never write more than **size** characters to the **datumName** string. A zero value is returned for success, and -1 for failure. Failure is almost always caused by

providing an invalid coordinate system key name. Note that description field of a coordinate system definition is limited to 63 characters, and **size** is typically 64 (to accommodate for the null terminating character used in C).

CS_getEllipsoidOf Get Ellipsoid Of a Coordinate System

```
Function CS_getEllipsoidOf (ByVal csKeyName As String,
                            ByVal ellipsoidName As String,
                            ByVal size As Integer) As Integer
function CS_getEllipsoidOf (csKeyName, ellipsoidName :PChar;
                            size :Integer) :Integer;
int CS_getEllipsoidOf (Const char *csKeyName,char *ellipsoidName,int size);
```

Use this function to obtain the ellipsoid referenced by the coordinate system whose key name is provided by the **csKeyName** argument. The ellipsoid key name is returned in the string pointed to by the **ellipsoidName** argument. CS_getEllipsoidOf will never write more than **size** characters to the **ellipsoidName** string. A zero value is returned for success, and -1 for failure. Failure is almost always caused by providing an invalid coordinate system key name. Note that key names are limited to 23 characters, and **size** is typically 24 (to accommodate for the null terminating character used in C).

This function usually returns the empty string as most coordinate systems are referenced to a datum rather than an ellipsoid. Use this function only in those cases where the CS_getDatumOf function returns the empty string, indicating a coordinate system which is cartographically referenced.

CS_getReferenceOf Get Reference Of a Coordinate System

```
Function CS_getReferenceOf (ByVal csKeyName As String,
                            ByVal reference As String,
                            ByVal size As Integer) As Integer
function CS_getReferenceOf (csKeyName, reference :PChar;
                            size :Integer) :Integer;
int CS_getReferenceOf (Const char *csKeyName,char *reference,int size);
```

Use this function to obtain an ASCII representation of what the coordinate system referenced by the **csKeyName** argument is referenced to. This operates correctly for both geodetic and cartographic references. The returned ASCII string will include an indication of the type of reference, and also the key name imvolved. The reference description is returned in the string pointed to by the **reference** argument. CS_getRefernceOf will never write more than **size** characters to the **reference** string. A zero value is returned for success, and -1 for failure. Failure is almost always caused by providing an invalid coordinate system key name. 32 is a customary value for the **size** argument. A geodetic reference looks something like: `Datum: WGS84`.

CS_getSourceOf Get Source Of Coordinate System

```
Function CS_getSourceOf (ByVal csKeyName As String,ByVal source As String,
                         ByVal size As Integer) As Integer
function CS_getSourceOf (csKeyName, source :PChar; size :Integer) :Integer;
int CS_getSourceOf (Const char *csKeyName,char *source,int size);
```

Use this function to obtain the source of information field of the coordinate system definition whose key name is provided by the **csKeyName** argument. The source information is returned in the string pointed to by the **source** argument. `CS_getSourceOf` will never write more than **size** characters to the **source** string. A zero value is returned for success, and -1 for failure. Failure is almost always caused by providing an invalid coordinate system key name. Note that source of information field of a coordinate system definition is limited to 63 characters, and **size** is typically 64 (to accommodate for the null terminating character used in C).

CS_getUnitsOf Get Units of a Coordinate System

```
Function CS_getUnitsOf (ByVal csKeyName As String,ByVal units As String,
                        ByVal size As Integer) As Integer
function CS_getUnitsOf (csKeyName, units :PChar; size :Integer) :Integer;
int CS_getUnitsOf (Const char *csKeyName,char *units,int size);
```

Use this function to obtain the key name of the units of the coordinate system definition whose key name is provided by the **csKeyName** argument. The unit key name is returned in the string pointed to by the **units** argument. `CS_getUnitsOf` will never write more than **size** characters to the **units** string. A zero value is returned for success, and -1 for failure. Failure is almost always caused by providing an invalid coordinate system key name. Note that unit key name field of a coordinate system definition is limited to 23 characters, and **size** is typically 24 (to accommodate for the null terminating character used in C).

CS_getElValues Get Ellipsoid Values

```
Function CS_getElValues (ByVal elKeyName As String,ByRef eRadius As Double,
                         ByRef eSquared As Double) As Integer
function CS_getElValues (elKeyName :PChar; var eRadius, eSquared :Double)
:Integer;
int CS_getElValues (Const char *elKeyName,double *eRadius,double
*eSquared);
```

Use this function to obtain the equatorial radius and the eccentricity squared values for the ellipsoid referenced by **elKeyName** argument. The appropriate values are returned in the double variables pointed to by the **eRadius** and **eSquared** arguments.. A zero value is returned for success, and -1 for failure. Failure is almost always caused by providing an invalid ellipsoid key name. Note that the value returned in **eRadius** is the equatorial radius and is always in meters. The value returned in the **eSquared** variable is unitless, and will be zero if the ellipsoid definition referenced by the **elKeyName** argument ia actually the definition of a sphere.

CS_getCurvatureAt get CURVATURE AT specified latitude

```
Function CS_getCurvatureAt (ByVal csKeyName As String,
                            ByVal latitude As Double) As Double
function CS_getCurvatureAt (csKeyName, source :PChar; latitude :double)
:Double;
double CS_getCurvatureAt (Const char *csKeyName,double latitude);
```

This function uses the ellipsoid underlying the coordinate system definition indicated by the **csKeyName** argument, and computes the Gaussian curvature at the specified **latitude**. The key name argument must be that of a coordinate system definition, and the latitude argument is specified in degrees.

The function returns a hard zero in the event of an error, which can be caused by providing an invalid coordinate system key name. The latitude argument is not checked and used as is, since only the sine of the latitude is necessary for the calculation (and all real values have, technically, a sine value).

CS_isgeo IS GEOgraphic

```
Function CS_isgeo (ByVal key_nm As String) As Integer
function CS_isgeo (key_nm :PChar) :Integer;
int *CS_isgeo (Const char *key_nm);
```

CS_isgeo will check the coordinate system definition with the key name indicated by the **key_nm** argument and return +1 (i.e. TRUE) if the coordinate system does return geographic coordinates. A zero is returned if the named coordinate system is not geographic.

CS_isgeo returns a negative value in the event of a hard error. The most frequent cause of a hard error is providing an invalid coordinate system name.

CS_llazdd Lat/Long to AZimuth and Distance calculator

```
Function CS_llazdd (ByVal e_rad As Double, ByVal e_sq As Double,
                    ByRef ll_from As Double,
                    ByRef ll_to As Double,
                    ByRef dist As Double) As Double
function CS_llazdd (e_rad,e_sq :Double;var ll_from,ll_to :Double;
                    var dist :Double) :Double;
double CS_llazdd (double e_rad,double e_sq,Const double ll_from [2],
                               Const double ll_to [2],
                               double *dist);
```

CS_llazdd returns the ellipsoidal azimuth and distance between two points on the surface of an ellipsoid specified in terms of latitude and longitude. **e_rad** specifies the equatorial radius and **e_sq** specifies the square of the eccentricity of the ellipsoid. The returned azimuth is calculated from the location specified by **ll_from** to that specified by **ll_to**, and the distance between the two points is returned at the location pointed to by **dist**. The units of the returned distance are the same as those used to specify the equatorial radius.

Latitude and longitude values are in degrees where the first element in each array is the longitude and the second element is the latitude. West longitude and south latitude are negative.

The algorithm used is known as: "Solution of the geodetic inverse problem after T. Vincenty modified Rainsford's Method with Helmert's elliptical terms."

ERRORS

CS_llazdd makes no checks for possible errors. The algorithm used is appropriate for any combination of points that are not antipodal. That is, the points used must not be exactly opposite each other, i.e. on the endpoints of a straight line that passes through the center of the earth.

CS_llFromMgrs calculate Lat/Long FROM MGRS

```
Function CS_mgrsFromLl (ByRef latLng As Double, ByVal mgrs As String) As
Integer
function CS_mgrsFromLl (var latLng :Double; mgrs :PChar) :Integer;
double CS_mgrsFromLl (double latLng [2],const char *mgrs);
```

CS_llFromMgrs returns in the array indicated by the **latLng** argument the geographic coordinate equivalent of the MGRS (Military Grid Reference System) string provided by the **mgrs** argument. This function is aware of the poles and the rather strange stuff that happens in the northern Europe.

CS_llFromMgrs returns a zero for success, and –1 for failure. Failure can be caused by failing to call the *CS_mgrsSetUp* prior to calling *CS_llFromMgrs* or providing an invalid MGRS string.

CS_mgrsFromLl calculate MGRS FROM Lat/Long

```
Function CS_mgrsFromLl (ByVal mgrs As String, ByRef latLng As Double,
                        ByVal precision As Integer) As Integer
function CS_mgrsFromLl (mgrs :PChar; var latLng :Double;precision :Integer)
:Integer;
double CS_mgrsFromLl (char *mgrs,double latLng [2],int precision);
```

CS_mgrsFromLl returns the MGRS (Military Grid Reference System) equivalent of the geographic position provided by the **latLng** argument in the character array (string) indicated by the **mgrs** argument. The precision of the result is controlled by the **precision** argument that must have a value between 1 and 5 (inclusive). The result array is assumed to be at least 16 bytes in length. The **latLng** argument must adhere to the convention established for internal coordinates. This function is aware of the poles and the rather strange stuff that happens in the northern Europe.

CS_mgrsFromLl returns a zero for success, and –1 for failure. Failure can be caused by failing to call the *CS_mgrsSetUp* prior to calling *CS_mgrsFromLl* or providing an invalid geographic coordinate.

CS_mgrsSetUp MGRS SETUP

```
Function CS_mgrsSetUp (ByVal ellipsoid As String, ByVal bessel As Integer)
As Integer
function CS_mgrsSetUp (ellipsoid :PChar; bessel :Integer) :Integer;
double CS_mgrsSetUp (const char* ellipsoid, int bessel);
```

Use the CS_mgrsSetUp to specify the ellipsoid that is to be used in the MGRS (Military Grid Reference System) calculations. Use the **ellipsoid** argument to provide the key name of the ellipsoid definition that is to be used. There are two alphabetic code sequences used with MGSR. A zero value for the **bessel** argument causes the normal code sequence to be used, a value of +1 indicates that the code sequence associated with the Bessel ellipsoid is to be used.

CS_mgrsFromLl returns a zero for success, and –1 for failure. Failure is usually caused by a invalid ellipsoid name.

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_scale grid SCALE factor function

```
Function CS_scale (ByVal cs_name As String,ByRef ll As Double) As Double
function CS_scale (cs_name :PChar;var ll :Double) :Double;
double CS_scale (Const char *cs_name,double ll [2]);
```

CS_scale returns the grid scale factor of the coordinate system whose key name is provided by the **cs_name** argument, at the location provided by the **ll** argument. The position provided by the **ll** argument must be in longitude and latitude form, in degrees, where the first element of the array is the longitude and the second element of the array is the latitude. Use negative values for west longitude and south latitude. The returned value is the grid scale factor.

CS_scale uses the same cache of coordinate system definitions as does *CS_cnvrt*, therefore, the performance penalty of using this very simple function is not as great as one might expect.

In the case of a conformal projection, the K and H scale factors are the same; there is no ambiguity. For non-conformal projections, however, the K and H functions are not the same. In these cases, this function will return the more interesting of the two factors. For example, for the Equidistant Conic, the K factor is always 1.0, and this function would return the H factor for this projection.

ERRORS

CS_scale will return a negative one (i.e. -1.0) if an error occurs. Providing an invalid coordinate system name is the most common source of error.

CS_scalh grid SCALE factor(H) function

```
Function CS_scalh (ByVal cs_name As String,ByRef ll As Double) As Double
function CS_scalh (cs_name :PChar;var ll :Double) :Double;
double CS_scalh (Const char *cs_name,double ll [2]);
```

CS_scalh returns the grid scale factor along a meridian of the coordinate system whose key name is provided by the **cs_name** argument, at the location provided by the **ll** argument. The position provided by the **ll** argument must be in longitude and latitude form, in degrees, where the first element of the array is the longitude and the second element of the array is the latitude. Use negative values for west longitude and south latitude. The returned value is the grid scale factor.

CS_scalh uses the same cache of coordinate system definitions as does *CS_cnvrt*, therefore, the performance penalty of using this very simple function is not as great as one might expect.

ERRORS

CS_scalh will return a negative one (i.e. -1.0) if an error occurs. Providing an invalid coordinate system name is the most common source of error.

CS_scalk grid SCALE factor(K) function

```
Function CS_scalk (ByVal cs_name As String,ByRef ll As Double) As Double
function CS_scalk (cs_name :PChar;var ll :Double) :Double;
double CS_scalk (Const char *cs_name,double ll [2]);
```

CS_scalk returns the grid scale factor along a parallel of the coordinate system whose key name is provided by the **cs_name** argument, at the location provided by the **ll** argument. The position provided by the **ll** argument must be in longitude and latitude form, in degrees, where the first element of the array is the longitude and the second element of the array is the latitude. Use negative values for west longitude and south latitude. The returned value is the grid scale factor.

CS_scalk uses the same cache of coordinate system definitions as does *CS_cnvrt*, therefore, the performance penalty of using this very simple function is not as great as one might expect.

ERRORS

CS_scalk will return a negative one (i.e. -1.0) if an error occurs. Providing an invalid coordinate system name is the most common source of error.

CS_setHelpPath SET HELP PATH

```
Function CS_setHelpPath (ByVal helpPath As String) As Integer
function CS_setHelpPath (helpPath :PChar) :Integer;
int CS_setHelpPath (const char *helpPath);
```

Use the *CS_setHelpPath* function to set the directory that you desire to have CS-MAP search when seeking the MFC dialog help file. The **helpPath** argument must point to a null terminated string that carries the full path to the desired directory.

CS_setHelpPath returns +1 (i.e. TRUE) if a properly named file exists in the indicated directory. Zero (i.e. FALSE) is returned if such a file does not exist.

CS_spZoneNbrMap State Plane ZONE NumBeR MAPper

```
Function CS_spzone (ByValue csKeyName As String,ByVal is83 As Integer) As
Integer
function CS_spzone (csKeyName :PChar; is83 :Integer) :Integer
int CS_spzone (char *csKeyName,int is83);
```

CS_spZoneNbrMap examines the character array provided by the **csKeyName** argument and if it determines that the array contains a valid state plane zone number specification, the contents of the array is replaced with the appropriate corresponding coordinate system key name. If the **is83** parameter is non-zero, the zone number is interpreted as a NAD83 zone number. Otherwise, the zone number is interpreted as a NAD27 zone number.

If the original content of the character array pointed to be the **csKeyName** argument is not a valid state plane zone number, the contents of the array remains unchanged.

CS_spZoneNbrMap returns 0 if a substitution was made. A positive one is returned if a substitution was not made because the value passed was not considered to be a valid state plane zone number. Minus one is returned if the original passed value is close to a state plane zone number (i.e. consisted of three or four digits), but did not match a valid state plane zone number.

CS_unEnum UNits ENUMerator

```
Function CS_unEnum (ByVal index As Integer,ByVal type As Integer,
                    ByVal key_name As String,
                    ByVal name_sz As Integer) As Integer
function CS_unEnum (index, type :Integer;key_name :Pchar;name_sz :Integer)
:Integer;
int CS_unEnum (int index,int type,char *key_name,int nm_size);
```

CS_unEnum is used to enumerate all units of a specific type in the CS-MAP units table. *CS_unEnum* returns in the memory buffer pointer to by the **key_name** argument the name of the **index**'th entry in the unit table of the type specified by the *type* argument. *CS_unEnum* will never write more than **nm_size** bytes to the indicated location. **Index** is a zero based index; the index of the first entry in the unit table is zero.

Currently, only two types of units supported, length and angular measure. Manifest constants defined in the *cs_map.h* header file are used to distinguish the desired type. These are **cs_UTYP_LEN**, for linear units, and **cs_UTYP_ANG**, for angular units. The **type** argument must be one of these values.

CS_unEnum returns a positive 1 to indicate success. If **index** is too large, a zero is returned.

ERRORS

CS_unEnum will return a -1 and set `cs_Error` appropriately if any of the following conditions are encountered:

cs_INV_INDX	The index argument was negative.

BUGS

If called with an invalid type, *CS_unEnum* should probably return an error condition, but it doesn't. Calling *CS_unEnum* with an invalid type causes a return value of zero for all positive values of the **index** argument.

CS_unitlu UNIT Look Up

```
Function CS_unitlu (ByVal type As Integer,ByVal unit_nm As String) As
Double
function CS_unitlu (type :Integer;unit_nm :PChar) :Double;
double CS_unitlu (short type,Const char *unit_nm);
```

Given the type of measurement, either length or angular, as specified by the **type** argument and the unit name as specified by the **unit_nm** argument, *CS_unitlu* will return a double which represents the multiplier required to convert a value in the unit system indicated by **unit_nm** to units of meters or degrees.

Currently, only two types of units supported, length and angular measure. Manifest constants defined in the cs_map.h header file are used to distinguish the desired type. These are **cs_UTYP_LEN** and **cs_UTYP_ANG**. The **type** argument must be one of these values.

unit_nm must be a null terminated string matching one of the supported units as defined in *CSdataU*. *CS_unitlu* returns zero in the event the provided unit name is not known. **unit_nm** may be one of the supported abbreviations for any of the units defined in the unit table.

For example, to convert a value in feet to meters, one could code:

```
double CS_unitlu ();
{
     meters = feet * CS_unitlu (cs_UTYP_LEN,"FOOT");
}
```

Or to convert degrees to grads:

```
double CS_unitlu ();
{
     grads = degrees / CS_unitlu (cs_UTYP_ANG,"GRAD");
}
```

CS_unitlu knows about the first and second abbreviations provided for in the `cs_Unittab_` structure. Therefore, the following are equivalent to the above:

```
double CS_unitlu ();
{
     meters = feet * CS_unitlu (cs_UTYP_LEN,"FT");
}

double CS_unitlu ();
{
     grads = degrees / CS_unitlu (cs_UTYP_ANG,"GR");
}
```

ERRORS

CS_unitlu will return zero and set `cs_Error` to **cs_INV_UNIT** if the unit name pointed to by **unit_nm** is not defined in `cs_Unittab` for the specified type, or the specified type is not valid.

High Performance Interface

Functions which are considered part of the High Performance Interface are described in this section. Several of these functions return addresses (i.e. pointers to) malloc'ed memory, and therefore these functions are not suitable for all languages. Function prototype definitions are given in the C syntax only.

CS_audflt Angular Unit DeFauLT

```
char *CS_audlt (Const char *new_dflt);
```

Use *CS_audflt* to control the status of the "defaultable" angular unit reference feature of CS-MAP. **new_dflt** must be either a pointer to a valid angular unit name, a pointer to the null string, or the NULL pointer. In the case where **new_dflt** is a pointer to a valid angular unit name, *CS_audflt* causes the default angular unit feature to be activated, using the angular unit name provided as the, possibly new, default value. When **new_dflt** is a pointer to the null string, *CS_audflt* disables the default angular unit feature. When **new_dflt** is the NULL pointer, the status of the angular unit default feature remains

unchanged.

In all cases, *CS_audflt* returns the previous status (or in the case of **new_dflt** == NULL, the current status) in the form of a pointer to a static character array that contains the name of the previous default angular unit. Should the returned pointer point to a null string, the indicated previous status is disabled.

ERRORS

CS_audflt will return the **NULL** pointer if the key name provided is not that of a valid angular unit. In this event, the current status of the default angular unit feature remains unchanged.

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_cscnv Coordinate System CoNVergence

```
double CS_cscnv (Const struct cs_Csprm_ *csprm,Const double ll [2]);
```

Given the definition of the coordinate system, **csprm**, such as returned by *CS_csloc*, *CS_cscnv* will return the convergence angle in degrees east of north at the location given by **ll**.

The location, as given by **ll** is in terms of latitude and longitude. The longitude is the first element of the **ll** array, latitude is the second, and both must be given in degrees. Positive values are used to specify north latitude and east longitude, negative values are used to specify south latitude and west longitude.

CS_csdef Coordinate System DEFinition locator

```
struct cs_Csdef_ *CS_csdef (Const char *key_nm);
```

CS_csdef will return a pointer to a *malloc*'ed `cs_Csdef_` structure that contains the definition of the coordinate system indicated by **key_nm**. **Key_nm** must point to an array that contains the null terminated key name of the desired coordinate system. The memory allocated for the coordinate system definition may be released by calling *CS_free* when no longer needed.

ERRORS

CS_csdef will return a **NULL** pointer and set `cs_Error` if any of the following conditions are detected:

cs_CSDICT	The Coordinate System Dictionary file could not be found or otherwise opened. (See *CS_altdr*.
cs_IOERR	A physical I/O error occurred during access to the Coordinate System Dictionary file.
cs_CS_BAD_MAGIC	The file accessed under the assumption that it was a Coordinate System Dictionary wasn't a Coordinate System Dictionary after all; it had an invalid magic number on the front end.
cs_CS_NOT_FND	A coordinate system definition with the name given by **key_nm** was not found in the Coordinate System Dictionary.
cs_NO_MEM	Insufficient dynamic memory was available to allocate space for the `cs_Csdef_` structure.

CS_csdel Coordinate System definition DELete

```
int CS_csdel (struct cs_Csdef_ *csdef);
```

CS_csdel will delete from the Coordinate System Dictionary the definition of the coordinate system pointed to by **csdef**.

The delete is accomplished by creating a new Coordinate System Dictionary file and copying all but the referenced coordinate system definitions from the existing dictionary to the new one. This implies that sufficient disk space must exist to perform the copy. A zero is returned if the delete was successfully completed, a -1 if a problem occurred. An attempt to delete a non-existent coordinate system definition is NOT considered an error.

If the value of the global variable `cs_Protect` is greater than or equal to zero, *CS_csdel* will not delete a coordinate system which is marked as being a distribution coordinate system (i.e. `cs_Csdef_.protect == 1`). If the value of `cs_Protect` is greater than zero, it is interpreted as the number of days after which a user defined coordinate system is protected. For example, is `cs_Protect` is 60, a user-defined coordinate system becomes protected 60 days after it is last modified.

ERRORS

CS_csdel will return a -1 and set `cs_Error` appropriately if any of the following conditions are encountered:

cs_CSDICT	The Coordinate System Dictionary could not be found or otherwise opened. (See *CS_altdr*)
cs_IOERR	A physical I/O error occurred in copying the Coordinate System Dictionary to the new file.
cs_CS_BAD_MAGIC	The file assumed to be the Coordinate System Dictionary by virtue of its name was not a Coordinate System Dictionary; it had an invalid magic number.
cs_TMP_CRT	The attempt to create a new file, to which the modified Coordinate System Dictionary was to be copied, failed.
cs_DISK_FULL	Insufficient disk space was available to accommodate the copying of the Coordinate System Dictionary to the new file.
cs_UNLINK	The request to remove the old copy of the Coordinate System Dictionary failed.
cs_RENAME	The request to rename the new Coordinate System Dictionary file from its temporary name to COORDSYS failed.
cs_CS_PROT	The coordinate system to be deleted is a distribution coordinate system and may not be deleted.
cs_CS_UPROT	The coordinate system is a user defined coordinate system that has not been modified for 60 days and is therefore protected.

CS_csEnumByGroup Coordinate System ENUMerator By Group

```
int CS_csEnumByGroup (int index,Const char *grp_name,struct cs_Csgrplst_
*cs_descr);
```

CS_csEnumByGroup is used to enumerate a specific group of coordinate systems in the Coordinate System Dictionary. *CS_csEnumByGroup* returns a completed `cs_Csgrplst_` structure at the location given by the **cs_descr** argument containing information that describes the **index**'th entry in the coordinate system group named by the **grp_name** argument. **Index** is a zero based index; the index of the first coordinate system in any group is zero. The `next` element of the returned `cs_Csgrplst_` structure is always set to the **NULL** pointer.

CS_csEnumByGroup returns a positive 1 to indicate success. If **index** is too large, a zero is returned.

ERRORS

CS_csEnumByGroup will return a -1 and set `cs_Error` appropriately if any of the following conditions are encountered:

cs_CSDICT	The Coordinate System Dictionary could not be found or otherwise opened. (See *CS_altdr*)
cs_IOERR	A physical I/O error occurred in accessing the Coordinate System Dictionary.
cs_CS_BAD_MAGIC	The file assumed to be the Coordinate System Dictionary by virtue of its name was not a Coordinate System Dictionary; it had an invalid magic number.
cs_CSGRP_INVKEY	The **grp_name** argument was not that of a valid group name.
cs_INV_INDX	The index argument was negative.

CS_csGrpEnum Coordinate System GRouP ENUMerator

```
int CS_csGrpEnum (int index,char *grp_name,int name_sz,char *grp_dscr,int
dscr_sz);
```

CS_csGrpEnum is used to enumerate all groups in the Coordinate System Group table. *CS_csGrpEnum* returns in the memory buffer pointer to by the **grp_name** argument the key name of the **index**'th entry in the Coordinate System Group Table. *CS_csGrpEnum* will never write more than **name_sz** bytes to the indicated location. Similarly, *CS_csGrpEnum* returns the Coordinate System Group description in the buffer pointed to be **grp_dscr** and whose size in indicated by **dscr_sz**. The **grp_name** and/or the **grp_dscr** arguments may be the **NULL** pointer to suppress return of the indicated item.

Index is a zero based index; the index of the first entry in the Coordinate System Group Table is zero. *CS_dtEnum* returns a positive 1 to indicate success. If *index* is too large, a zero is returned. Inactive groups (a feature planned for a future release) are ignored, and are not considered to exist as far as **index** is concerned.

ERRORS

CS_csGrpEnum will return a -1 and set `cs_Error` appropriately if any of the following conditions are encountered:

cs_INV_INDX	The index argument was negative.

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_cssch Coordinate System SCale H, along a meridian

```
double CS_cssch (Const struct cs_Csprm_ *csprm,Const double ll [2]);
```

Given the definition of the coordinate system, **csprm**, such as returned by *CS_csloc*, *CS_cssch* will compute the grid scale factor along a meridian at the location given by **ll** and return this value. See *CS_cssck* for the grid scale factor along a parallel. Note that in conformal projections, the grid scale along a parallel equals the grid scale along a meridian at any point.

The location, as given by **ll**, is in terms of latitude and longitude. The longitude is the first element of the **ll** array, latitude is the second, and both must be given in degrees. Positive values are used to specify north latitude and east longitude, negative values are used to specify south latitude and west longitude.

CS_cssck Coordinate System SCale K, along a parallel

```
double CS_cssck (Const struct cs_Csprm_ *csprm,Const double ll [2]);
```

Given the definition of the coordinate system, **csprm**, such as returned by *CS_csloc*, *CS_cssck* will compute the grid scale factor along a parallel at the location given by **ll** and return this value. See *CS_cssch* for the grid scale factor along a meridian. Note that in conformal projections, the grid scale along a parallel equals the grid scale along a meridian at any point.

The location, as given by **ll** is in terms of latitude and longitude. The longitude is the first element of the **ll** array, latitude is the second, and both must be given in degrees. Positive values are used to specify north latitude and east longitude, negative values are used to specify south latitude and west longitude.

CS_csscl Coordinate System SCaLe

```
double CS_csscl (Const struct cs_Csprm_ *csprm,Const double ll [2]);
```

Given the definition of the coordinate system, **csprm**, such as returned by *CS_csloc*, *CS_csscl* will compute the grid scale factor at the location given by **ll** and return this value.

The location, as given by **ll**, is in terms of latitude and longitude. The longitude is the first element of the **ll** array, latitude is the second, and both must be given in degrees. Positive values are used to specify north latitude and east longitude, negative values are used to specify south latitude and west longitude.

Non-conformal projections have two different grid scale factors: the scale along a meridian and the scale along a parallel. In the case of azimuthal projections, the two scale factors are along a radial line from the origin and normal to such radial lines, respectively. In these cases, *CS_csscl* will return the more interesting of the two. For example, in the American Polyconic, the grid scale factor along all parallels is always 1.0; therefore *CS_csscl* return the grid scale factor along a meridian for this projection.

CS_csupd Coordinate System dictionary UPDate

```
int CS_csupd (struct cs_Csdef_ *csdef,int crypt);
```

CS_csupd will cause coordinate system definition pointed to by **csdef** to be added to the Coordinate System Dictionary. If a coordinate system with the same key name already exists, it is replaced by the definition provided. If no such definition exists, the new definition is added to the dictionary. If **crypt** is non-zero, the entry will be encrypted before being written to the dictionary.

In the event that the indicated coordinate system already exists, *CS_csupd* will return a 1 to indicate a successful update. In the event that the provided coordinate system had to be added to the Coordinate System Dictionary, a zero is returned. A -1 is returned if the update failed for any reason.

Please note that the addition of a new coordinate system definition requires the sorting of the Coordinate System Definition file. This may take a few seconds to complete, depending upon the size of the Coordinate System Dictionary.

If the value of the global variable `cs_Protect` is greater than or equal to zero, *CS_csupd* will not change a coordinate system which is marked as being a distribution coordinate system (i.e. `cs_Csdef_.protect == 1`). If the value of `cs_Protect` is greater than zero, it is interpreted as the number of days after which a user defined coordinate system is protected. For example, if *cs_Protect* is 60, a user-defined coordinate system becomes protected 60 days after it is last modified.

Additionally, if the value of the global character variable `cs_Unique` is not the null character, *CS_csupd* will not add a coordinate system definition if its key name does not contain the character indicated. For example, if `cs_Unique` is set to the colon character, *CS_csupd* will not add a coordinate system whose key name does not contain a colon character.

ERRORS

CS_csupd will return a -1 and set `cs_Error` appropriately if any of the following conditions are encountered during the update:

cs_CSDICT	The Coordinate System Dictionary file could not be opened. (See *CS_altdr*).
cs_IOERR	A physical I/O error occurred during the update process.
cs_CS_BAD_MAGIC	The file that, by virtue of its name and location, was supposed to be a Coordinate System Dictionary wasn't a Coordinate System Dictionary; its magic number was invalid.
cs_DISK_FULL	There was insufficient disk space available to add the coordinate system definition to the dictionary.
cs_CS_PROT	The coordinate system to be updated is a distribution coordinate system and may not be updated.
cs_CS_UPROT	The coordinate system is a user defined coordinate system that has not been modified for 60 days and is therefore protected.
cs_UNIQUE	The coordinate system provided does not already exist and would need to be added; but the key name does not contain the unique character.

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*.

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_dtdef DaTum DEFinition locator

```
struct cs_Dtdef_ *CS_dtdef (Const char *key_nm);
```

CS_dtdef will return a pointer to a *malloc*'ed `cs_Dtdef_` structure which contains the definition of the datum indicated by **key_nm**. **Key_nm** must point to an array that contains the null terminated key name of the desired datum definition. The memory allocated for the datum definition should be released by using *CS_free* when no longer needed.

ERRORS

CS_dtdef will return a **NULL** pointer and set `cs_Error` if any of the following conditions are detected:

cs_DTDICT	The Datum Dictionary file could not be found or otherwise opened. (See *CS_altdr*)
cs_IOERR	A physical I/O error occurred during access to the Datum Dictionary file.
cs_DT_BAD_MAGIC	The file accessed under the assumption that it was a Datum Dictionary wasn't a Datum Dictionary after all; it had an invalid magic number on the front end.
cs_DT_NOT_FND	A datum definition with the name given by **key_nm** was not found in the Datum Dictionary.
cs_NO_MEM	Insufficient dynamic memory was available to allocate space for the `cs_Dtdef_` structure.

CS_dtdel DaTum definition DELete

```
int CS_dtdel (struct cs_Dtdef_ *dtdef);
```

CS_dtdel will delete from the Datum Dictionary the definition of the Datum pointed to by **dtdef**.

The delete is accomplished by creating a new Datum Dictionary file and copying all but the referenced datum definition from the existing dictionary to the new one. This implies that sufficient disk space must be available to perform this copy. A zero is returned if the delete was successfully completed, a -1 if a problem occurred. An attempt to delete a non-existent datum definition is NOT considered a

problem.

If the value of the global variable `cs_Protect` is greater than or equal to zero, *CS_dtdel* will not delete a datum definition which is marked as being a distribution datum definition (i.e. `cs_Dtdef_.protect == 1`). If the value of `cs_Protect` is greater than zero, it is interpreted as the number of days after which a user defined datum is protected. For example, is `cs_Protect` is 60, a user-defined datum becomes protected 60 days after it is last modified.

ERRORS

CS_dtdel will return a -1 and set `cs_Error` appropriately if any of the following conditions are encountered:

cs_DTDICT	The Datum Dictionary could not be found or otherwise opened. (See *CS_altdr*)
cs_IOERR	A physical I/O error occurred in copying the Datum Dictionary to the new file.
cs_DT_BAD_MAGIC	The file assumed to be the Datum Dictionary by virtue of its name and location was not a Datum Dictionary; it had an invalid magic number.
cs_TMP_CRT	The attempt to create a new file, to that the modified Datum Dictionary was to be copied, failed.
cs_DISK_FULL	Insufficient disk space was available to accommodate the copying of the Datum Dictionary to the new file.
cs_UNLINK	The request to remove the old copy of the Datum Dictionary failed.
cs_RENAME	The request to rename the new Datum Dictionary file from its temporary name to DATUMS failed.
cs_DT_PROT	The datum definition to be updated is a distribution datum definition and may not be deleted.
cs_DT_UPROT	The datum is a user-defined datum which has not been modified for 60 days and is therefore protected.

CS_dtdflt DaTum DeFauLT

```
char *CS_dtdflt (Const char *new_dflt);
```

Use *CS_dtdflt* to control the status of the "defaultable" datum reference feature of CS-MAP. **New_dflt** must be either a valid datum key name, a pointer to the null string, or the **NULL** pointer. In the case where **new_dflt** is a pointer to a valid datum definition key name, *CS_dtdflt* causes the default datum

feature to be active, using the datum key name provided as the default value. When **new_dflt** is a pointer to the null string, *CS_dtdflt* disables the default datum feature. When **new_dflt** is the **NULL** pointer, the status of the default feature remains unchanged.

In all cases, *CS_dtdflt* returns the previous status (or in the case of **new_dflt** == **NULL**, the current status) in the form of a pointer to a static character array which shall contain the name of the previous default datum. Should the returned pointer point to a null string, the indicated status is disabled.

ERRORS

CS_dtdflt will return the **NULL** pointer if the key name provided is not that of a valid datum. In this event, the status of the default datum feature remains unchanged.

CS_dtloc DaTum LOCate

```
struct cs_Datum_ *CS_dtloc (Const char *key_nm);
struct cs_Datum_ *Csdtloc1 (Const struct cs_Dtdef_ *dtPtr);
struct cs_Datum_ *Csdtloc2 (Const struct cs_Dtdef_ *dtPtr,Const struct
cs_Eldef_ *elPtr);
```

CS_dtloc will return a pointer to a *malloc*'ed `cs_Datum_` structure which contains the definition of the datum indicated by **key_nm** along with the ellipsoid information referenced by the datum definition. **Key_nm** must point to an array that contains the key name of the desired datum. The memory allocated for the datum definition should be released using *CS_free* when no longer needed.

CSdtloc1 and *CSdtloc2* are alternatives to *CS_dtloc* that enable alternative sources for datum and ellipsoid definitions. These have been provided for applications that may, for example, store the datum, and/or ellipsoid, definitions in an application database.

Note that while *CSdtloc1* will not need to access the Datum Dictionary, it will need to access the Ellipsoid Dictionary to resolve the ellipsoid reference in the datum definition provided. *CSdtloc2* is completely independent of both dictionaries.

ERRORS

CS_dtloc, *Csdtloc*, and *CSdtloc2* will return a **NULL** pointer and set **cs_Error** if any of the following conditions are detected:

cs_NO_MEM	Insufficient dynamic memory was available to allocate space for the `cs_Datum_` structure.

CS_dtloc uses *CS_dtdef* and *CS_eldef* to obtain definition records from the Datum and Ellipsoid Dictionaries. Therefore, all of the error conditions detected by these functions apply to this function as well.

CSdtloc1 uses *CS_eldef* to obtain definition records from the Ellipsoid Dictionary. Therefore, all of the error conditions detected by this function apply to this function as well.

CS_dtupd DaTum dictionary UPDate

```
int CS_dtupd (struct cs_Dtdef_ *dt_def,int crypt);
```

CS_dtupd will cause the datum definition pointed to by **dt_def** to be added to the Datum Dictionary. If a datum with the same key name exists, it is replaced by the definition provided. If no such definition exists, the new definition is added to the dictionary. If **crypt** is non-zero, the datum entry is encrypted before being written.

In the event that the indicated datum already existed, *CS_dtupd* will return a 1 to indicate a successful update. In the event that the provided datum definition had to be added to the Datum Dictionary, a zero is returned. A -1 is returned if the update failed for any reason.

Please note that the addition of a new datum definition requires the sorting of the Datum Dictionary file. This may take a few seconds to complete, depending upon the size of the Datum Dictionary.

If the value of the global variable `cs_Protect` is greater than or equal to zero, *CS_dtupd* will not change a datum definition which is marked as being a distribution datum definition (i.e. `cs_Dtdef_.protect == 1`). If the value of `cs_Protect` is greater than zero, it is interpreted as the number of days after which a user defined datum is protected. For example, is `cs_Protect` is 60, a user-defined datum becomes protected 60 days after it is last modified.

Additionally, if the value of the global character variable `cs_Unique` is not the null character, *CS_dtupd* will not add a datum definition if its key name does not contain the character indicated. For example, if `cs_Unique` is set to the colon character, *CS_dtupd* will not add a datum definition whose key name does not contain a colon character.

ERRORS

CS_dtupd will return a -1 and set `cs_Error` appropriately if any of the following conditions are encountered during the update:

cs_DTDICT	The Datum Dictionary file could not be opened. (See *CS_altdr*)
cs_IOERR	A physical I/O error occurred during the update process.
cs_DT_BAD_MAGIC	The file which, by virtue of its name and location, was assumed to be a Datum Dictionary wasn't a Datum Dictionary; its magic number was invalid.
Cs_DISK_FULL	There was insufficient disk space available to add the datum definition to the Datum Dictionary.
Cs_DT_PROT	The datum definition to be updated is a distribution datum definition and may not be updated.
cs_DT_UPROT	The datum is a user-defined datum that has not been modified for 60 days and is therefore protected.
cs_UNIQUE	The datum provided does not already exist and would need to be added; but the key name does not contain the unique character.

CS_eldef ELlipsoid DEFinition locator

```
struct cs_Eldef_ *CS_eldef (Const char *key_nm);
```

CS_eldef will return a pointer to a *malloc*'ed `cs_Eldef_` structure which contains the definition of the ellipsoid indicated by **key_nm**. **Key_nm** must point to an array that contains the null terminated key name of the desired ellipsoid definition. The memory allocated for the ellipsoid definition should be released using *CS_free* when no longer needed.

ERRORS

CS_eldef will return a **NULL** pointer and set `cs_Error` if any of the following conditions are detected:

cs_ELDICT	The Ellipsoid Dictionary file could not be found or otherwise opened. (See *CS_altdr*)
cs_IOERR	A physical I/O error occurred during access to the Ellipsoid Dictionary file.
cs_EL_BAD_MAGIC	The file accessed under the assumption that it was an Ellipsoid Dictionary wasn't an Ellipsoid Dictionary after all; it had an invalid magic number on the front end.
cs_EL_NOT_FND	A ellipsoid definition with the name given by **key_nm** was not found in the Ellipsoid Dictionary.
cs_NO_MEM	Insufficient dynamic memory was available to allocate space for the `cs_Eldef_` structure

CS_eldel ELlipsoid definition DELete

```
int CS_eldel (struct cs_Eldef_ *eldef);
```

CS_eldel will delete from the Ellipsoid Dictionary the definition of the Ellipsoid pointed to by **eldef**.

The delete is accomplished by creating a new Ellipsoid Dictionary file and copying all but the referenced ellipsoid definition from the existing dictionary to the new one. This implies that sufficient disk space must be available to perform this copy. A zero is returned if the delete was successfully completed, a -1 if a problem occurred. An attempt to delete a non-existent ellipsoid definition is NOT considered a problem.

If the value of the global variable `cs_Protect` is greater than or equal to zero, *CS_eldel* will not delete an ellipsoid definition which is marked as being a distribution ellipsoid definition (i.e. `cs_Eldef_.protect == 1`). If the value of `cs_Protect` is greater than zero, it is interpreted as the number of days after which a user defined ellipsoid is protected. For example, is `cs_Protect` is 60, a user-defined ellipsoid becomes protected 60 days after it is last modified.

ERRORS

CS_eldel will return a -1 and set `cs_Error` appropriately if any of the following conditions are encountered:

cs_ELDICT	The Ellipsoid Dictionary could not be found or otherwise opened. (See *CS_altdr*)
cs_IOERR	A physical I/O error occurred in copying the Ellipsoid Dictionary to the new file.
cs_EL_BAD_MAGIC	The file assumed to be the Ellipsoid Dictionary by virtue of its name and location was not an Ellipsoid Dictionary; it had an invalid magic number.
cs_TMP_CRT	The attempt to create a new file, to which the modified Ellipsoid Dictionary was to be copied, failed.
cs_DISK_FULL	Insufficient disk space was available to accommodate the copying of the Ellipsoid Dictionary to the new file.
cs_UNLINK	The request to remove the old copy of the Ellipsoid Dictionary failed.
cs_RENAME	The request to rename the new Ellipsoid Dictionary file from its temporary name to ELIPSOID failed.
cs_EL_PROT	The ellipsoid definition to be updated is a distribution ellipsoid definition and may not be deleted.
cs_EL_UPROT	The ellipsoid is a user-defined ellipsoid which has not been modified for 60 days and is therefore protected.

CS_eldflt ELlipsoid DeFauLT

```
char *CS_eldlt (Const char *new_dflt);
```

Use *CS_eldflt* to control the status of the "defaultable" ellipsoid reference feature of CS-MAP. **New_dflt** must be either a pointer to a valid ellipsoid key name, a pointer to the null string, or the **NULL** pointer. In the case where **new_dflt** is a pointer to a valid ellipsoid definition key name, *CS_eldflt* causes the default ellipsoid feature to be activated, using the ellipsoid key name provided as the, possibly new, default value. When **new_dflt** is a pointer to the null string, *CS_eldflt* disables the default ellipsoid feature. When **new_dflt** is the **NULL** pointer, the status of the default feature remains unchanged.

In all cases, *CS_eldflt* returns the previous status (or in the case of **new_dflt** == **NULL**, the current status) in the form of a pointer to a static character array that shall contain the name of the previous default ellipsoid. Should the returned pointer point to a null string, the indicated status is disabled.

ERRORS

CS_eldflt will return the **NULL** pointer if the key name provided is not that of a valid ellipsoid. In this event, the status of the default ellipsoid feature remains unchanged.

CS_elEnum ELlipsoid ENUMerator

```
int CS_elEnum (int index,char *key_name,int size);
```

CS_elEnum is used to enumerate all ellipsoids in the Ellipsoid Dictionary. *CS_elEnum* returns in the memory buffer pointer to by the **key_name** argument the key name of the **index**'th entry in the Ellipsoid Dictionary. *CS_elEnum* will never write more than **size** bytes to the indicated location. **Index** is a zero based index; the index of the first entry in the Ellipsoid Dictionary is zero.

CS_elEnum returns a positive 1 to indicate success. If **index** is too large, a zero is returned.

ERRORS

CS_elEnum will return a -1 and set `cs_Error` appropriately if any of the following conditions are encountered:

cs_ELDICT	The Ellipsoid Dictionary could not be found or otherwise opened. (See *CS_altdr*)
cs_IOERR	A physical I/O error occurred in accessing the Ellipsoid Dictionary.
cs_DT_BAD_MAGIC	The file assumed to be the Ellipsoid Dictionary by virtue of its name was not an Ellipsoid Dictionary; it had an invalid magic number.
cs_INV_INDX	The index argument was negative.

CS_elupd ELlipsoid dictionary UPDate

```
int CS_elupd (struct cs_Eldef_ *el_def,int crypt);
```

CS_elupd will cause the ellipsoid definition pointed to by **el_def** to be added to the Ellipsoid Dictionary. If an ellipsoid with the same key name exists, it is replaced by the definition provided. If no such definition exists, the new definition is added to the dictionary. If **crypt** is non-zero, the ellipsoid entry is encrypted before being written.

In the event that the indicated ellipsoid already existed, *CS_elupd* will return a 1 to indicate a successful update. In the event that the provided ellipsoid had to be added to the Ellipsoid Dictionary, a zero is returned. A -1 is returned if the update failed for any reason.

Please note that the addition of a new ellipsoid definition requires the sorting of the Ellipsoid Dictionary file. This may take a few seconds to complete, depending upon the size of the Ellipsoid Dictionary.

If the value of the global variable `cs_Protect` is greater than or equal to zero, *CS_elupd* will not change an ellipsoid definition which is marked as being a distribution ellipsoid definition (i.e. `cs_Eldef_.protect == 1`). If the value of `cs_Protect` is greater than zero, it is interpreted as the number of days after which a user defined ellipsoid is protected. For example, is `cs_Protect` is 60, a user-defined ellipsoid becomes protected 60 days after it is last modified.

Additionally, if the value of the global character variable `cs_Unique` is not the null character, *CS_elupd* will not add an ellipsoid definition if its key name does not contain the character indicated. For example, if `cs_Unique` is set to the colon character, *CS_elupd* will not add an ellipsoid definition whose key name does not contain a colon character.

ERRORS

CS_elupd will return a -1 and set `cs_Error` appropriately if any of the following conditions are encountered during the update:

cs_ELDICT	The Ellipsoid Dictionary file could not be opened. (See *CS_altdr*)
cs_IOERR	A physical I/O error occurred during the update process.
cs_EL_BAD_MAGIC	The file which, by virtue of its name and location, was assumed to be an Ellipsoid Dictionary wasn't an Ellipsoid Dictionary; its magic number was invalid.
cs_DISK_FULL	There was insufficient disk space available to add the ellipsoid definition to the Ellipsoid Dictionary.
cs_EL_PROT	The ellipsoid definition to be updated is a distribution ellipsoid definition and may not be updated.
cs_EL_UPROT	The ellipsoid is a user-defined ellipsoid that has not been modified for 60 days and is therefore protected.
cs_UNIQUE	The ellipsoid provided does not already exist and would need to be added; but the key name does not contain the unique character.

CS_errmsg ERRor MeSsaGe

```
void CS_errmsg (char msg_bufr,int bufr_size);
```

CS_errmsg returns to the calling function a null terminated string that 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_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_llchk Lat/Long limits CHecK

```
int CS_llchk (Const struct cs_Csprm_ *csprm,int cnt,Const double pnts
[][3]);
```

CS_llchk determines if the points, great circles, and regions defined by the point list provided by the **cnt** and **pnts** arguments are within the mathematical domain and useful range of the coordinate system provided by the **csprm** argument. All points in the point list are expected to be geographic coordinates. Use *CS_xychk* to check a list of cartesian coordinates.

CS_llchk returns **cs_CNVRT_OK** if all coordinate subject to the determination are both within the mathematical domain of the coordinate system and the useful range of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more coordinates is outside of the mathematical domain of the coordinate system. **cs_CNVRT_USFL** is returned if all coordinates subject to the determination are within the mathematical domain of the coordinate system, but one or more coordinates are outside of the useful range of the coordinate system.

The useful range of a coordinate system may be defined by the user as part of the coordinate system definition. In the absence of such a definition, the setup function for each projection computes a useful range based on the parameters for the projection. In some cases, this computed useful range will be too liberal; in others it may be too conservative. In any case, checking coordinates to be converted against the useful range is a good way to alert users of a possible problem, such as using the wrong coordinate system for a set of coordinates.

CS_ludflt Linear Unit DeFauLT

```
char *CS_ludlt (Const char *new_dflt);
```

Use *CS_ludflt* to control the status of the "defaultable" linear unit reference feature of CS-MAP. **New_dflt** must be either a pointer to a valid linear unit name, a pointer to the null string, or the **NULL** pointer. In the case where **new_dflt** is a pointer to a valid linear unit name, *CS_ludflt* causes the

default linear unit feature to be activated, using the linear unit name provided as the, possibly new, default value. When **new_dflt** is a pointer to the null string, *CS_ludflt* disables the default linear unit feature. When **new_dflt** is the **NULL** pointer, the status of the linear unit default feature remains unchanged.

In all cases, *CS_ludflt* returns the previous status (or in the case of **new_dflt** == NULL,, the current status) in the form of a pointer to a static character array which shall contain the name of the previous default linear unit. Should the returned pointer point to a null string, the indicated status is disabled.

ERRORS

CS_ludflt will return the **NULL** pointer if the key name provided is not that of a valid linear unit. In this event, the status of the default linear unit feature remains unchanged.

CS_xychk X and Y limits CHecK

```
int CS_xychk (Const struct cs_Csprm_ *csprm,int cnt,Const double pnts
[][3]);
```

CS_xychk determines if the points, line segments, and regions defined by the point list provided by the **cnt** and **pnts** arguments are within the mathematical domain and useful range of the coordinate system provided by the **csprm** argument. All points in the point list are expected to be cartesian coordinates. Use *CS_llchk* to check a list of geographic coordinates.

CS_xychk returns **cs_CNVRT_OK** if all coordinate subject to the determination are both within the mathematical domain of the coordinate system and the useful range of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more coordinates is outside of the mathematical domain of the coordinate system. **cs_CNVRT_USFL** is returned if all coordinates subject to the determination are within the mathematical domain of the coordinate system, but one or more coordinates are outside of the useful range of the coordinate system.

The useful range of a coordinate system may be defined by the user as part of the coordinate system definition. In the absence of such a definition, the setup function for each projection computes a useful range based on the parameters for the projection. In some cases, this computed useful range will be too liberal; in others it may be too conservative. In any case, checking coordinates to be converted against the useful range is a good way to alert users of a possible problem, such as using the wrong coordinate system for a set of coordinates.

CS_usrUnitPtr - Units Look Up Hook Function

```
double CS_usrUnitPtr (short type,Const char *unitName);
```

This name, `CS_usrUnitPtr`, does not refer to a function. Rather, it refers to a global variable which is defined as a pointer to a function which is defined as the above given prototype declares. Applications can use a function as declared above, and the related global pointer variable, to implement unit definitions in a dynamic manner.

If the global variable `CS_usrUnitPtr` (defined in *CSdata.c*) is not null, the indicated function is called whenever the CS-MAP library is asked to access a specific unit definition. This function, then, can be used to dynamically supply a unit conversion value which does not exist in the compiled unit table. Applications can use this to implement their own unit definition table or dynamically generate such a definition based on the unit name provided.

CS-MAP passes the **unitName** argument to the hook function prior to any validation, thus dynamic definition names need not adhere to the CS-MAP key name conventions. In the event that the hook function determines that it wishes to supply the definition, the desired conversion value must be returned. CS-MAP passed the unit type requested to the hook function using the **type** argument.

The hook function returns an integer value:

- a positive non-zero value to indicate that a conversion value is being supplied by the hook function, and the value returned is indeed the conversion value.
- zero is returned to indicate that normal CS-MAP unit table access is to be performed.
- a negative value is returned to indicate that an error is to be reported. It is expect that the nature of the error would have already reported through the use of *CS_erpt*.

CS_unitAdd - ADD UNIT to Table

```
int CS_unitAdd (struct cs_Unittab_ *unitPtr);
```

Use this function to add a new unit to the unit table at run time. Essentially, the unit definition pointed to by the **unitPtr** argument is copied to a disabled entry in the compiled unit table. This function does not check any of the entries in the provided unit definition, so use this function with great care.

Errors

CS_unitAdd returns a zero value for success. A negative return value indicates a failure. In this case, one of the following error conditions will have been reported through the use of CS_erpt:

cs_UADD_TYP	The type of unit specified in the provided definition was invalid. Must be either cs_UTYP_LIN or cs_UTYP_ANG.
cs_UADD_DUP	A unit definition with the (singular) name of given in the provided definition already exists in the unit table.
cs_UADD_FULL	All of the disabled slots in the unit table have been filled; thus the unit table is currently full.

CS_unitDel -- DELete UNIT from table

```
int CS_unitDel (short type,Const char *name);
```

Use this function to disable an entry in the unit table. The specific unit is identified by the **type** and **name** arguments. Type must be given as either **cs_UTYP_LIN** or **cs_UTYP_ANG**. Note that compiled (i.e. not necessarily added) unit entries can also be disabled. This will remove them from subsequent unit enumerations performed by *CS_unEnum*.

Errors

CS_unitDel will return a zero for success. A negative 1 is return to indicate failure. In the event of failure, one of the following error conditions will have been reported through *CS_erpt*:

cs_UDEL_NONE The named unit, of the provided type, did not exist in the unit table.

Low Level Interface Functions

Functions which are considered part of the Low Level Interface are described in this section. Several of these functions require geographic arguments. Remember that these are required to be:

1. in longitude, latitude, and height order, and
2. given in degrees, and

3. referenced to Greenwich meridian, and
4. where west lonigtude and south latitude are represented by negative values.

Function prototype definitions are given in the C syntax only.

CHAPTER 4

Cartographic Projection Funtions

Albers Equal Area Projection (CSalber)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Alber Equal Area Conic Projection.

CSalberF Forward conversion

```
int CSalberF (Const struct cs_Alber_ *alber,double xy[2],Const double ll
[2]);
```

Given a properly initialized `cs_Alber_` structure via the **alber** argument, *CSalberF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSalberF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSalberI Inverse conversion

```
int CSalberI (Const struct cs_Alber_ *alber,double ll [2],Const double xy
[2]);
```

Given a properly initialized `cs_Alber_` structure via the **alber** argument, *CSalberI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSalberI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are the first elements in these arrays, the Y coordinate and the latitude are the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSalberK grid scale (K) along parallel

```
double CSalberK (Const struct cs_Alber_ *alber,Const double ll [2]);
```

CSalberK returns the grid scale factor, along a parallel, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array. (The use of the **ll** array is the same as described above.)

CSalberH grid scale (H) along meridian

```
double CSalberH (Const struct cs_Alber_ *alber,Const double ll [2]);
```

CSalberH returns the grid scale factor, along a meridian, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array.

CSalberC Convergence angle

```
double CSalberC (Const struct cs_Alber_ *alber,Const double ll [2]);
```

CSalberC returns the convergence angle in degrees east of north of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array.

CSalberQ definition Quality check

```
int CSalberQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CSalberQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Alber Equal Area Projection. *CS_cschk* examines those definition components that are common to all coordinates system (datum or ellipsoid reference, map scale, and units) and, therefore, *CSalberQ* only examines those components specific to the Alber Equal Area Projection. *CSalberQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSalberQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSalberL Latitude/longitude check

```
int CSalberL (Const struct cs_Alber_ *alber,int cnt,Const double pnts
[][3]);
```

CSalberL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **alber** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSalbersL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSalberL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates is outside of the mathematical domain of the coordinate system.

CSalberX Xy coordinate check

```
int CSalberX (Const struct cs-Alber_ *alber,int cnt,Const double pnts
[][3]);
```

CSalberX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **alber** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSalbersX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSalberL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSalberS Setup

```
void CSalberS (struct cs_Csprm_ *csprm);
```

The *CSalberS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the standard parallels, the origin latitude and longitude, and other projection parameters are known, there are many calculations that need only be performed once. *CSalberS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the single argument provided to *CSalberS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. The specific elements of the `cs_Csdef_` structure

that must be initialized for the Albers projection are:

prj_prm1	Latitude, in degrees, of the northern standard parallel. Unlike other conics, it is important to distinguish between the northern and southern standard parallels for the Albers.
prj_prm2	Latitude, in degrees, of the southern standard parallel. This is, rarely, the same as prj_prm1, to obtain a conic with a single point of tangency.
org_lng	The longitude, in degrees, of the origin of the projection.
org_lat	The latitude, in degrees, of the origin of the projection.
Scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, selected to cause all X coordinates within the coordinate system to be positive values of reasonable size.
y_off	The false northing to be applied to all Y coordinates.
Quad	an integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The values of equatorial radius and eccentricity are extracted from the *datum* element of the *cs_Csprm_* structure. These are normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required elements are:

e_rad	The equatorial radius of the earth in meters.
eccent	This value represents the eccentricity of the ellipsoid.
to84_via	An integer code that specifies the technique that is to be used to convert geographic coordinates based on this datum to WGS84.

cs_Alber_ Structure

The results of the one-time calculations are recorded in the `alber` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSalberF*, *CSalberI*,

CSalberK, *CSalberH*, and *CSalberC* functions require as their first argument.

American Polyconic Projection (CSplycn)

This set of functions represent the Coordinate System Mapping Package's knowledge of the American Polyconic Projection.

CSplycnF Forward conversion

```
int CSplycnF (Const struct cs_Plycn_ *plycn,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Plycn_` structure via the **plycn** argument, *CSplycnF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSplycnF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSplycnI Inverse conversion

```
int CSplycnI (Const struct cs_Plycn_ *plycn,double ll [2],Const double xy
[2]);
```

Given a properly initialized `cs_Plycn_` structure via the **plycn** argument, *CSplycnI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSplycnI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **ll** and **xy** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSplycnK grid scale (K) along parallel

```
double CSplycnK (Const struct cs_Plycn_ *plycn,Const double ll [2]);
```

CSplycnK returns the value 1.0 which represents the grid scale along a parallel of any coordinate system based on this projection at any location.

CSplycnH grid scale (H) along meridian

```
double CSplycnH (Const struct cs_Plycn_ *plycn,Const double ll [2]);
```

CSplycnH returns the grid scale factor, along a meridian, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array.

CSplycnC Convergence angle

```
double CSplycnC (Const struct cs_Plycn_ *plycn,Const double ll [2]);
```

CSplycnC returns the convergence angle in degrees east of north of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array. At the current time, definitive formulas for the convergence angle of this projection elude us. The convergence angle is computed using the *CS_llazdd* function.

CSplycnQ definition Quality check

```
int CSplycnQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
     int *err_list [],int list_sz);
```

CSplycnQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the American Polyconic Projection. *CS_cschk* examines those definition components that are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CSplycnQ* only examines those components specific to the American Polyconic Projection. *CSplycnQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSplycnQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSplycnL Latitude/longitude check

```
int CSplycnL (Const struct cs_Plycn_ *plycn,int cnt,Const double pnts
[][3]);
```

CSplycnL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **plycn** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSplycnsL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSplycnL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSplycnX Xy coordinate check

```
int CSplycnX (Const struct cs_Plycn_ *plycn,int cnt,Const double pnts
[][3]);
```

CSplycnX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **plycn** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSplycnsX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSplycnL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSplycnS Setup

```
void CSplycnS (struct cs_Csprm_ *csprm);
```

The *CSplycnS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the central meridian, the origin latitude, and other projection parameters are known, there are many calculations that need only be performed once. *CSplycnS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the single argument provided to *CSplycnS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function;

but can be provided by the application at run time. The specific elements of the `cs_Csdef_` structure that must be initialized for the American Polyconic projection are:

prj_prm1	Longitude, in degrees, of the central meridian.
org_lat	The latitude, in degrees, of the origin of the projection.
scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size.
y_off	The false northing to be applied to all Y coordinates.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The values of equatorial radius and eccentricity are extracted from the *datum* element of the *cs_Csprm_* structure. These are normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required elements are:

e_rad	The equatorial radius of the earth in meters.
eccent	This value represents the eccentricity of the ellipsoid.
to84_via	An integer code that specifies the technique that is to be used to convert geographic coordinates based on this datum to WGS84.

cs_Plycn_ Structure

The results of the one-time calculations are recorded in the `plycn` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSplycnF*, *CSplycnI*, *CSplycnK*, *CSplycnH*, and *CSplycnC* functions require as their first argument.

Azimuthal Equal Area Projection (CSazmea)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Lambert Azimuthal Equal Area Projection.

CsazmeaF Forward conversion

```
int CSazmeaF (Const struct cs_Azmea_ *azmea,double xy [2],Const double ll
```

```
[2]);
```

Given a properly initialized cs_Azmea_ structure via the **azmea** argument, *CSazmeaF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSazmeaF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSazmeal Inverse conversion

```
int CSazmeaI (Const struct cs_Azmea_ *azmea,double ll [2],Const double xy
[2]);
```

Given a properly initialized cs_Azmea_ structure via the **azmea** argument, *CSazmeaI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSazmeaI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CsazmeaK grid scale (K) normal

```
double CSazmeaK (Const struct cs_Azmea_ *azmea,Const double ll [2]);
```

CSazmeaK returns the grid scale factor normal to the radial at the geodetic location specified by the **ll** argument. In the case of the ellipsoidal form of this projection, analytical formulas for this value have not been located and the result is arrived at using the *CS_llazdd* function.

CSazmeaH grid scale (H) radial

```
double CSazmeaH (Const struct cs_Azmea_ *azmea,Const double ll [2]);
```

CSazmeaH returns the grid scale factor along a radial line from the coordinate system origin to the point provided. Since this projection is authalic (i.e. equal area), the value returned is the reciprocal of that returned by *CSazmeaK.*

CSazmeaC Convergence angle

```
double CSazmeaC (Const struct cs_Azmea_ *azmea,Const double ll [2]);
```

CSazmeaC returns the convergence angle in degrees east of north of the geodetic location specified by the **ll** argument. Analytical formulas for this value have not been located and the result is arrived at using the *CS_llazdd* function.

CSazmeaQ definition Quality check

```
int CSazmeaQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CSazmeaQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Azimuthal Equal Area Projection. *CS_cschk* examines those definition components that are common to all coordinates system (datum or ellipsoid reference, map scale, and units) and, therefore, *CSazmeaQ* only examines those components specific to the Azimuthal Equal Area Projection. *CSazmeaQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSazmeaQ* may be called with the **NULL** pointer and/or a

zero for the **err_list** and **list_sz** arguments respectively.

CSazmeaL Latitude/longitude check

```
int CSazmeaL (Const struct cs_Azmea_ *azmea,int cnt,Const double pnts
[][3]);
```

CSazmeaL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **azmea** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSazmeasL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSazmeaL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSazmeaX Xy coordinate check

```
int CSazmeaX (Const struct cs_Azmea_ *azmea,int cnt,Const double pnts
[][3]);
```

CSazmeaX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **azmea** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSazmeasX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSazmeaL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSazmeaS Setup

```
void CSazmeaS (struct cs_Csprm_ *csprm);
```

The *CSazmeaS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the origin latitude and longitude, and other projection parameters are known, there are many calculations that need only be performed once. *CSazmeaS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the argument provided to *CSazmeaS* serves as the source for input and the repository for the results as described below

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time.

The following parameters must be set:

prj_prm1	The azimuth, in degrees east of north, of the positive Y-axis of the coordinate system.
org_lng	The longitude, in degrees, of the origin of the projection.
org_lat	The latitude, in degrees, of the origin of the projection.
scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size. This is the X coordinate of the coordinate system origin.
y_off	The false northing to be applied to all Y coordinates. This is the Y coordinate of the coordinate system origin.
quad	an integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The values of equatorial radius and eccentricity are extracted from the *datum* element of the *cs_Csprm_* structure. These are normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required elements are:

e_rad	The equatorial radius of the earth in meters.
eccent	This value represents the eccentricity of the ellipsoid.
to84_via	An integer code that specifies the technique that is to be used to convert geographic coordinates based on this datum to WGS84.

cs_Azmea_ Structure

The results of the one-time calculations are recorded in the `azmea` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSazmeaF*, *CSazmeaI*, *CSazmeaK*, *CSazmeaH*, and *CSazmeaC* functions require as their first argument.

Azmuthal Equidistant Projection (Csazmed)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Azimuthal Equidistant Projection.

CSazmedF Forward conversion

```
int CSazmedF (Const struct cs_Azmed_ *azmed,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Azmed_` structure via the **azmed** argument, *CSazmedF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSazmedF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSazmedI Inverse conversion

```
int CSazmedI (Const struct cs_Azmed_ *azmed,double ll [2],Const double xy
[2]);
```

Given a properly initialized `cs_Azmed_` structure via the **azmed** argument, *CSazmedI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSazmedI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSazmedK grid scale (K) normal

```
double CSazmedK (Const struct cs_Azmed_ *azmed,Const double ll [2]);
```

CSazmedK returns the grid scale factor normal to the radial at the geodetic location specified by the **ll** argument. In the case of the ellipsoidal form of this projection, analytical formulas for this value have not been located and the result is arrived at using the *CS_llazdd* function.

CSazmedH grid scale (H) radial

```
double CSazmedH (Const struct cs_Azmed_ *azmed,Const double ll [2]);
```

CSazmedH returns the value 1.0; the scale at any point in the direction of a line emanating from the origin and passing through the any point of any coordinate system based on this projection. (This is what makes this projection an Equidistant Projection.)

CSazmedC Convergence angle

```
double CSazmedC (Const struct cs_Azmed_ *azmed,Const double ll [2]);
```

CSazmedC returns the convergence angle in degrees east of north of the geodetic location specified by the **ll** argument. Analytical formulas for this value have not been located and the result is arrived at using the *CS_llazdd* function.

CSazmedQ definition Quality check

```
int CSazmedQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CSazmedQ determines if the coordinate system definition provided by the **csdef** argument is consistent

with the requirements of the Azimuthal Equidistant Projection. *CS_cschk* examines those definition components that are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CSazmedQ* only examines those components specific to the Azimuthal Equidistant Projection. *CSazmedQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSazmedQ* may be called with the NULL pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSazmedL Latitude/longitude check

```
int CSazmedL (Const struct cs_Azmed_ *azmed,int cnt,Const double pnts
[][3]);
```

CSazmedL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **azmed** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSazmedsL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSazmedL* returns cs_CNVRT_OK if all subject coordinates are within the mathematical domain the coordinate system. cs_CNVRT_DOMN is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSazmedX Xy coordinate check

```
int CSazmedX (Const struct cs_Azmed_ *azmed,int cnt,Const double pnts
[][3]);
```

CSazmedX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **azmed** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSazmedsX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSazmedL* returns cs_CNVRT_OK if all subject coordinates are within the mathematical domain of the coordinate system. cs_CNVRT_DOMN is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSazmedS Setup

```
void CSazmedS (struct cs_Csprm_ *csprm);
```

The *CSazmedS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the origin latitude and longitude, and other projection parameters are known, there are many calculations that need only be performed once. *CSazmedS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the first argument provided to *CSazmedS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the *csdef* element of the *cs_Csprm_* structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. There are two variations to this projection:

- Lambert Azimuthal Equidistant (cs_PRJCOD_AZMED)
- Lambert Azimuthal Equidistant, Elevated Ellipsoid (cs_PRJCOD_AZEDE)

The following parameters are common to both variations:

org_lng	The longitude, in degrees, of the origin of the projection.
org_lat	The latitude, in degrees, of the origin of the projection.
scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size. This is the X coordinate of the coordinate system origin.
y_off	The false northing to be applied to all Y coordinates. This is the Y coordinate of the coordinate system origin.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Lambert Azimuthal Equidistant Projection

This is the traditional variation of the Lambert Azimuthal Equidistant projection. Note that it differs slightly from many other implementations in that it accepts a parameter value for the azimuth of the Y axis relative to true north. This provides support for local/company coordinate system. Even better local/company coordinate system support is provided by the Lambert Azimuthal Equidistant, Elevated Ellipsoid variation.

The parameter must be specified in degrees east of north. An azimuth west of north would be specified with a negative value.

The following parameters must be set:

prj_prm1	The azimuth, in degrees east of north, of the positive Y-axis of the coordinate system.

Lambert Azimuthal Equidistant, Elevated Ellipsoid (cs_PRJCOD_AZEDE)

This variation of the Lambert Azimuthal Equidistant projection accepts an average elevation parameter which is added to the equatiorial radii of the ellipsoid. This better enbales CS-MAP to emulate a local/company coordinate system.

The parameter must be specified in system units. That is, if the coordinate system unit is, say, FEET; the average elevation must also be specified in feet.

The following parameters must be set:

prj_prm1	The azimuth, in degrees east of north, of the positive Y-axis of the coordinate system.
prj_prm2	The average elevation in the region of the system, expressed in coordinate system units.

Datum Definition

The values of equatorial radius and eccentricity are extracted from the *datum* element of the *cs_Csprm_* structure. These are normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required elements are:

e_rad	The equatorial radius of the earth in meters.
eccent	This value represents the eccentricity of the ellipsoid.
to84_via	An integer code that specifies the technique that is to be used to convert geographic coordinates based on this datum to WGS84.

cs_Azmed_ Structure

The results of the one-time calculations are recorded in the `azmed` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSazmedF*, *CSazmedI*, *CSazmedK*, *CSazmedH*, and *CSazmedC* functions require as their first argument.

Bonne Projection Projection (CSbonne)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Bonne Projection. Setting the standard parallel equal to the equator produces a Sinusoidal Projection. Setting the standard parallel to either pole produces the Werner Projection.

CSbonneF Forward conversion

```
int CSbonneF (Const struct cs_Bonne_ *bonne,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Bonne_` structure via the **bonne** argument, *CSbonneF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSbonneF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSbonnel Inverse conversion

```
int CSbonneI (Const struct cs_Bonne_ *bonne,double ll [2],Const double xy
```

```
[2]);
```

Given a properly initialized cs_Bonne_ structure via the **bonne** argument, *CSbonneI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSbonneI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSbonneK parallel scale (K)

```
double CSbonneK (Const struct cs_Bonne_ *bonne,Const double ll [2]);
```

CSbonneK returns the value 1.0 which is the grid scale factor, along any parallel, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array.

CSbonneH meridian scale (H)

```
double CSbonneH (Const struct cs_Bonne_ *bonne,Const double ll [2]);
```

CSbonneK returns the grid scale, along a meridian, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array. Analytical formulas for this value have not been located and the result is arrived at using the *CS_llazdd* function.

CSbonneC Convergence angle

```
double CSbonneC (Const struct cs_Bonne_ *bonne,Const double ll [2]);
```

CSbonneC returns the convergence angle of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array. Analytical formulas for this value have not been located and the result is arrived at using the *CS_llazdd* function.

CSbonneQ definition Quality check

```
int CSbonneQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
      int *err_list [],int list_sz);
```

CSbonneQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Boone Projection. *CS_cschk* examines those definition components that are common to all coordinates system (datum or ellipsoid reference, map scale, and units) and, therefore, *CSbonneQ* only examines those components specific to the Bonne Projection. *CSbonneQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSbonneQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSbonneL Latitude/longitude check

```
int CSbonneL (Const struct cs_Bonne_ *bonne,int cnt,Const double pnts
[][3]);
```

CSbonneL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **bonne** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSbonnesL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus

defined. *CSbonneL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates is outside of the mathematical domain of the coordinate system.

CSbonneX Xy coordinate check

```
int CSbonneX (Const struct cs_Bonne_ *bonne,int cnt,Const double pnts
[][3]);
```

CSbonneX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **bonne** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSbonnesX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSbonneL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSbonneS Setup

```
void CSbonneS (struct cs_Csprm_ *csprm);
```

The *CSbonneS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the standard parallel, the origin longitude, and other projection parameters are known, there are many calculations that need only be performed once. *CSbonneS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the single argument provided to *CSbonneS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. The specific elements of the `cs_Csdef_` structure that must be initialized for the Bonne Projection are:

org_lng	The longitude, in degrees, of the central meridian of the projection.
org_lat	The latitude, in degrees, of the standard parallel of the projection.
scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size.
y_off	The false northing to be applied to all Y coordinates.
quad	an integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The values of equatorial radius and eccentricity are extracted from the *datum* element of the *cs_Csprm_* structure. These are normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required elements are:

e_rad	The equatorial radius of the earth in meters.
eccent	This value represents the eccentricity of the ellipsoid.
to84_via	An integer code that specifies the technique that is to be used to convert geographic coordinates based on this datum to WGS84.

cs_Bonne_ Structure

The results of the one-time calculations are recorded in the `bonne` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSbonneF*, *CSbonneI*, *CSbonneK*, *CSbonneH*, and *CSbonneC* functions require as their first argument.

Bipolar Oblique Conformal Conic Projection (CSbpcnc)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Bipolar Oblique Conformal Conic Projection. This projection was developed, by O. M. Miller (of Miller Cylindrical fame), specifically for mapping both the North American and South American continents on the same conformal map. This projection is supported for the sphere only. The equatorial radius of

the referenced ellipsoid is used as the radius of the sphere.

CSbpcncF Forward conversion

```
int CSbpcncF (Const struct cs_Bpcnc_ *bpcnc,double xy [2],Const double ll
[2]);
```

Given a properly initialized cs_Bpcnc_ structure via the **bpcnc** argument, *CSbpcncF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSbpcncF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSbpcncI Inverse conversion

```
int CSbpcncI (Const struct cs_Bpcnc_ *bpcnc,double ll [2],Const double xy
[2]);
```

Given a properly initialized cs_Bpcnc_ structure via the **bpcnc** argument, *CSbpcncI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSbpcncI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSbpcncK parallel scale (K)

```
double CSbpcncK (Const struct cs_Bpcnc_ *bpcnc,Const double ll [2]);
```

CSbpcncK returns the grid scale factor, as measured along a parallel, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array. Since this is a conformal projection, there is no H function as the scale along a meridian equals the scale along a parallel.

CSbpcncC Convergence angle

```
double CSbpcncC (Const struct cs_Bpcnc_ *bpcnc,Const double ll [2]);
```

CSbpcncC returns the convergence angle of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array. At the current time, formulas that analytically define the convergence angle for this projection elude us. Thus, the convergence angle is determined empirically through the use of the *CS_llazdd* function.

CSbpcncQ definition Quality check

```
int CSbpcncQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CSbpcncQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Bi-Polar Conformal Conic Projection. *CS_cschk* examines those definition components that are common to all coordinates system (datum or ellipsoid reference, map scale, and units) and, therefore, *CSbpcncQ* only examines those components specific to the Bi-Polar Conformal Conic Projection. *CSbpcncQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSbpcncQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSbpcncL Latitude/longitude check

```
int CSbpcncL (Const struct cs_Bpcnc_ *bpcnc,int cnt,Const double pnts
[][3]);
```

CSbpcncL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **bpcnc** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). **CSbpcncsL**'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSbpcncL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSbpcncX Xy coordinate check

```
int CSbpcncX (Const struct cs_Bpcnc_ *bpcnc,int cnt,Const double pnts
[][3]);
```

CSbpcncX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **bpcnc** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSbpcncsX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSbpcncL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSbpcncS Setup

```
void CSbpcncS (struct cs_Csprm_ *csprm);
```

The *CSbpcncS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the poles, the standard parallels, and other projection parameters are known, there are many calculations that need only be performed once. *CSbpcncS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the single argument provided to *CSbpcncS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. As far as we know, this projection is usually used only for the specific coordinate system for which it was invented. However, to remain consistent with the rest of CS-MAP, the following parameters can be specified. These parameters specify the location of the two poles upon which the projection is based. The first is always specified by latitude and longitude. The latitude of the second pole must always be specified. However, the longitude of the second pole can be specified either directly (*prj_prm3*) or as an angular distance from the first pole (*prj_prm5*). If *prj_prm5* is greater than zero, the second method is used. In the listing of parameters given below, the values used for the specific map for which this projection was developed are given. The specific elements of the `cs_Csdef_` structure that must be initialized for the Bipolar Oblique Conformal Projection are:

prj_prm1	Longitude, in degrees, of the first pole (usually the southwest). [-110.0]
prj_prm2	Latitude, in degrees, of the first pole. [-20.0]
prj_prm3	Longitude, in degrees, of the second pole (usually the northeast). [-19.99333333333]
prj_prm4	Latitude, in degrees, of the second pole. [+45.0]
prj_prm5	If greater than zero, this value is considered to be the angular distance, in degrees, from the first pole to the second pole and the longitude of the second pole is computed as such. If this value is less than or equal to zero, the value provided in prj_prm3 is considered the longitude of the second pole. [+104.0]
prj_prm6	Angular distance, in degrees, from either pole to the first of two standard parallels. [+31.0]
prj_prm7	Angular distance, in degrees, from either pole to the second of two standard parallels. [+73.0]
scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied. [1.0]
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size. [0.0]
y_off	The false northing to be applied to all Y coordinates. [0.0]
quad	an integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The value of equatorial radius is extracted from the *datum* element of the `cs_Csprm_` structure and used as the radius of the sphere. This is normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required element is:

e_rad	The radius of the earth, as a sphere, in meters.

cs_Bpcnc_ Structure

The results of the one-time calculations are recorded in the `bpcnc` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSbpcncF*, *CSbpcncI*, *CSbpcncK*, *CSbpcncH*, *CSbpcncC*, and *CSbpcncB* functions require as their first argument.

Cassini Projection (CScsini)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Cassini Projection

CScsiniF Forward

```
int CScsiniF (Const struct cs_Csini_ *csini,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Csini_` structure via the **csini** argument, *CScsiniI* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CScsiniF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CScsiniI Inverse

```
int CScsiniI (Const struct cs_Csini_ *csini,double ll [2],Const double xy
[2]);
```

Given a properly initialized `cs_Csini_` structure via the **csini** argument, *CScsiniI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CScsiniI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are the first elements in these arrays, the Y coordinate and the latitude are the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CScsiniK parallel scale (K)

```
double CScsiniK (Const struct cs_Csini_ *csini,Const double ll [2]);
```

CScsiniK returns the grid scale factor along a line normal to the central meridian of the coordinate system at the geographic location defined by the latitude and longitude provided in the **ll** array. It is a specific feature of this projection that this scale factor is unity.

CScsiniH meridian scale (H)

```
double CScsiniH (Const struct cs_Csini_ *csini,Const double ll [2]);
```

CScsiniH returns the grid scale factor along a line parallel to the central meridian of the coordinate system at the geographic location defined by the latitude and longitude provided in the **ll** array.

CScsiniC Convergence angle

```
double CScsiniC (Const struct cs_Csini_ *csini,Const double ll [2]);
```

CScsiniC returns the convergence angle, in degrees east of north, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array. As analytical formulas for this quantity have not yet been located, the result is arrived at empirically using *CS_asllDd.*

CScsiniQ definition Quality check

```
int CScsiniQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CScsiniQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Cassini Projection. *CS_cschk* examines those definition components that are common to all coordinates system (datum or ellipsoid reference, map scale, and units) and, therefore, *CScsiniQ* only examines those components specific to the Cassini Projection. *CScsiniQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CScsiniQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CScsiniL Latitude/longitude check

```
int CScsiniL (Const struct cs_Csini_ *csini,int cnt,Const double pnts
[][3]);
```

CScsiniL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **csini** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CScsinisL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CScsiniL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CScsiniX Xy coordinate check

```
int CScsiniX (Const struct cs_Csini_ *csini,int cnt,Const double pnts
[][3]);
```

CScsiniX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **csini** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CScsinisX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CScsiniL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CScsiniS Setup

```
void CScsiniS (struct cs_Csprm_ *csprm);
```

The *CScsiniS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the central meridian, the origin latitude, and other projection parameters are known, there are many calculations that need only be performed once. *CScsiniS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the single argument provided to *CScsiniS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. The specific elements of the `cs_Csdef_` structure that must be initialized for the Cassini projection are:

prj_prm1	Longitude, in degrees, of the central meridian.
org_lat	The latitude, in degrees, of the origin of the projection.
Scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units, and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size.
y_off	The false northing to be applied to all Y coordinates.
Quad	an integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The values of equatorial radius and eccentricity are extracted from the *datum* element of the *cs_Csprm_* structure. These are normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required elements are:

e_rad	The equatorial radius of the earth in meters.
eccent	This value represents the eccentricity of the ellipsoid.
to84_via	An integer code that specifies the technique that is to be used to convert geographic coordinates based on this datum to WGS84.

cs_Csini_ Structure

The results of the one-time calculations are recorded in the `csini` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CScsiniF*, *CScsiniI*, *CScsiniK, CScsiniH*, and *CScsiniC* functions require as their first argument.

Danish System 34 (CSsys34)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Danish System 34 Projection. This projection is supported in ellipsoid form only; and most ordinary parameters are hard coded and selected via the 'region' (prj_prm1) parameter.

CSsys34C Convergence angle

```
double CSsys34C (Const struct cs_Sys34_ *sys34,Const double ll [2]);
```

CSsys34C returns the convergence angle in degrees east of north of the geodetic location specified by the **ll** argument. Analytical formulas for this value have not been located and the result is arrived at through the use of the *CS_azlld* function.

CSsys34F Forward conversion

```
int CSsys34F (Const struct cs_Sys34_ *sys34,double xy [2],Const double ll
[2]);
```

Given a properly initialized *cs_Sys34_* structure via the **sys34** argument, *CSsys34F* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSsys34F* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSsys34I Inverse conversion

```
int CSsys34I (Const struct cs_Sys34_ *sys34,double ll [2],Const double xy
[2]);
```

Given a properly initialized *cs_Sys34_* structure via the **sys34** argument, *CSsys34I* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSsys34I* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSsys34K parallel scale (K)

```
double CSsys34K (Const struct cs_Sys34_ *sys34,Const double ll [2]);
```

CSsys34K returns the grid scale factor along a parallel of any coordinate system based on this projection at any location. Analytical formulas for this value have not been located and the result is arrived at empirically the use of the *CS_llazdd* function.

CSsys34H meridian scale (H)

```
double CSsys34H (Const struct cs_Sys34_ *sys34,Const double ll [2]);
```

CSsys34H returns the grid scale factor along a meridian at the geodetic location specified by the **ll** argument. Analytical formulas for this value have not been located and the result is arrived at empirically the use of the CS_llazdd function.

CSsys34L Latitude/longitude check

```
int CSsys34L (Const struct cs_Sys34_ *sys34,int cnt,Const double pnts
[][3]);
```

CSsys34L determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **sys34** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSsys34sL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSsys34L* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. *cs_CNVRT_DOMN* is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSsys34Q definition Quality check

```
int CSsys34Q (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CSsys34Q determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Danish System 34 Projection. *CS_cschk* examines those definition components that are common to all coordinates system (datum or ellipsoid reference, map scale, and units) and, therefore, *CSsys34Q* only examines those components specific to the Danish System 34 Projection. *CSsys34Q* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSsys34Q* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSsys34X Xy coordinate check

```
int CSsys34X (Const struct cs_Sys34_ *sys34,int cnt,Const double pnts
[][3]);
```

CSsys34X determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **sys34** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSsys34sX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSsys34L* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSsys34S Setup (general)

```
void CSsys34S (struct cs_Csprm_ *csprm);
```

The *CSsys34S* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the zone and other projection parameters are known, there are many calculations that need only be performed once. *CSsys34S* performs these calculations and saves the results in the *cs_Csprm_* structure provided by its argument, **csprm**. Thus, the argument provided to *CSsys34S* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time.

The following parameters must be set:

prj_prm1	Indicates which of the three regions is to apply: 1.0 ==> jylland; 2.0 ==> sjælland, 3.0 ==> bornholm.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The values of equatorial radius and eccentricity are extracted from the *datum* element of the *cs_Csprm_* structure. These are normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required elements are:

e_rad	The equatorial radius of the earth in meters.
eccent	This value represents the eccentricity of the ellipsoid.
to84_via	An integer code that specifies the technique that is to be used to convert geographic coordinates based on this datum to WGS84.

cs_Sys34_ Structure

The results of the one-time calculations are recorded in the *sys34* element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSsys344F*, *CSsys344I*, *CSsys344K*, *CSsys344H,* and *CSsys34C* functions require as their first argument.

Equidistant Conic Projection (CSedcnc)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Equidistant Conic Projection, also known as the Simple Conic Projection.

CSedcncF Forward conversion

```
int CSedcncF (Const struct cs_Edcnc_ *edcnc,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Edcnc_` structure via the **edcnc** argument, *CSedcncF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSedcncF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSedcncI Inverse conversion

```
int CSedcncI (Const struct cs_Edcnc_ *edcnc,double ll [2],Const double xy
[2]);
```

Given a properly initialized `cs_Edcnc_` structure via the **edcnc** argument, *CSedcncI* will convert the X

and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSedcncI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSedcncK parallel scale (K)

```
double CSedcncK (Const struct cs_Edcnc_ *edcnc,Const double ll [2]);
```

CSedcncK returns the grid scale factor, along a parallel, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array.

CSedcncH meridian scale (H)

```
double CSedcncH (Const struct cs_Edcnc_ *edcnc,Const double ll [2]);
```

CSedcncH returns the value of 1.0 that represents the grid scale, along a meridian, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array. That is, all distances meaured along a meridian of this projection are true to scale, the essence of this projection.

CSedcncC Convergence angle

```
double CSedcncC (Const struct cs_Edcnc_ *edcnc,Const double ll [2]);
```

CSedcncC returns the convergence angle of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array.

CSedcncQ definition Quality check

```
int CSedcncQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CSedcncQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Equidistant Conic Projection. *CS_cschk* examines those definition components that are common to all coordinates system (datum or ellipsoid reference, map scale, and units) and, therefore, *CSedcncQ* only examines those components specific to the Equidistant Conic Projection. *CSedcncQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSedcncQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSedcncL Latitude/longitude check

```
int CSedcncL (Const struct cs_Edcnc_ *edcnc,int cnt,Const double pnts
[][3]);
```

CSedcncL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **edcnc** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSedcncsL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSedcncL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject

geographic coordinates are outside of the mathematical domain of the coordinate system.

CSedcncX Xy coordinate check

```
int CSedcncX (Const struct cs_Edcnc_ *edcnc,int cnt,Const double pnts
[][3]);
```

CSedcncX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **edcnc** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSedcncsX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSedcncL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSedcncS Setup

```
void CSedcncS (struct cs_Csprm_ *csprm);
```

The *CSedcncS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the standard parallels, the origin latitude and longitude, and other projection parameters are known, there are many calculations that need only be performed once. *CSedcncS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the single argument provided to *CSedcncS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. The specific elements of the `cs_Csdef_` structure that must be initialized for the Equidistant Conic projection are:

prj_prm1	Latitude, in degrees, of the first standard parallel, usually the northernmost (it makes no difference).
prj_prm2	Latitude, in degrees, of the second standard parallel, usually the southernmost. This is, rarely, the same as prj_prm1, to obtain a conic with a single point of tangency (i.e. a single standard parallel).
org_lng	The longitude, in degrees, of the origin of the projection.
org_lat	The latitude, in degrees, of the origin of the projection.
scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size.
y_off	The false northing to be applied to all Y coordinates.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The values of equatorial radius and eccentricity are extracted from the *datum* element of the *cs_Csprm_* structure. These are normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required elements are:

e_rad	The equatorial radius of the earth in meters.
eccent	This value represents the eccentricity of the ellipsoid.
to84_via	An integer code that specifies the technique that is to be used to convert geographic coordinates based on this datum to WGS84.

cs_Edcnc_ Structure

The results of the one-time calculations are recorded in the `edcnc` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSedcncF*, *CSedcncI*, *CSedcncK*, *CSedcncH*, and *CSedcncC* functions require as their first argument.

Equidistant Cylindrical Projection (CSedcyl)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Equidistant Cylindrical Projection. This projection is supported in spherical form only. The equatorial radius of the supplied ellipsoid is used as the radius of the sphere. This projection is also known as the Equirectangular or Rectangular projection. When the reference latitude of this projection is set to zero (i.e. the equator) the result is equivalent to what is known as the Plate Carrée or Simple Cylindrical projection. When the reference latitude is set to 45? (north or south), a Gall Isographic projection results.

CSedcylF Forward conversion

```
int CSedcylF (Const struct cs_Edcyl_ *edcyl,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Edcyl_` structure via the **edcyl** argument, *CSedcylF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSedcylF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSedcylI Inverse conversion

```
int CSedcylI (Const struct cs_Edcyl_ *edcyl,double ll [2],Const double xy
[2]);
```

Given a properly initialized `cs_Edcyl_` structure via the **edcyl** argument, *CSedcylI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSedcylI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system.

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSedcylK parallel scale (K)

```
double CSedcylK (Const struct cs_Edcyl_ *edcyl,Const double ll [2]);
```

CSedcylK returns the grid scale factor along a parallel at the geodetic location specified by the **ll** argument.

CSedcylH meridional scale (H)

```
double CSedcylH (Const struct cs_Edcyl_ *edcyl,Const double ll [2]);
```

CSedcylH returns the grid scale factor along a meridian at the geodetic location specified by the **ll** argument.

CSedcylC Convergence angle

```
double CSedcylC (Const struct cs_Edcyl_ *edcyl,Const double ll [2]);
```

CSedcylC returns the convergence angle in degrees east of north of the geodetic location specified by the **ll** argument. Analytical formulas for this value have not been located and the result is arrived at using the *CS_azsphr* function.

CSedcylQ definition Quality check

```
int CSedcylQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
```

```
int *err_list [],int list_sz);
```

CSedcylQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Equidistant Cylindrical Projection. *CS_cschk* examines those definition components that are common to all coordinates system (datum or ellipsoid reference, map scale, and units) and, therefore, *CSedcylQ* only examines those components specific to the Equidistant Cylindrical Projection. *CSedcylQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSedcylQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSedcylL Latitude/longitude check

```
int CSedcylL (Const struct cs_Edcyl_ *edcyl,int cnt,Const double pnts
[][3]);
```

CSedcylL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **edcyl** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSedcylsL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSedcylL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSedcylX Xy coordinate check

```
int CSedcylX (Const struct cs_Edcyl_ *edcyl,int cnt,Const double pnts
[][3]);
```

CSedcylX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **edcyl** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSedcylsX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSedcylL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSedcylS Setup

```
void CSedcylS (struct cs_Csprm_ *csprm);
```

The *CSedcylS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the reference parallel, origin longitude, and other projection parameters are known, there are many calculations which need only be performed once. *CSedcylS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the argument provided to *CSedcylS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. The following parameters are used:

prj_prm1	The latitude, in degrees, of the reference parallel.
org_lng	The longitude, in degrees, of the origin of the projection.
org_lat	The latitude, in degrees, of the origin of the projection.
scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size. This is the X coordinate of the coordinate system origin.
y_off	The false northing to be applied to all Y coordinates. This is the Y coordinate of the coordinate system origin.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The value of equatorial radius is extracted from the *datum* element of the `cs_Csprm_` structure and used as the radius of the sphere. This is normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required element is:

e_rad	The radius of the earth, as a sphere, in meters.

cs_Edcyl_ Structure

The results of the one-time calculations are recorded in the `edcyl` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSedcylF*, *CSedcylI*, *CSedcylK*, *CsedcylH*, and *CSedcylC* functions require as their first argument.

Eckert IV Projection (CSekrt4)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Eckert IV Projection. This projection is supported in spherical form only. The equatorial radius of the supplied ellipsoid is used as the radius of the sphere.

CSekrt4F Forward conversion

```
int CSekrt4F (Const struct cs_Ekrt4_ *ekrt4,double xy [2],Const double ll
[2]);
```

Given a properly initialized *cs_Ekrt4_* structure via the **ekrt4** argument, *CSekrt4F* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSekrt4F* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSekrt4I Inverse conversion

```
int CSekrt4I (Const struct cs_Ekrt4_ *ekrt4,double ll [2],Const double xy
[2]);
```

Given a properly initialized *cs_Ekrt4_* structure via the **ekrt4** argument, *CSekrt4I* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSekrt4I* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSekrt4K parallel scale (K)

```
double CSekrt4K (Const struct cs_Ekrt4_ *ekrt4,Const double ll [2]);
```

CSekrt4K returns the grid scale factor along a parallel of any coordinate system based on this projection at any location. Analytical formulas for this value have not been located and the result is arrived at empirically the use of spherical trigonometry.

CSekrt4H meridian scale (H)

```
double CSekrt4H (Const struct cs_Ekrt4_ *ekrt4,Const double ll [2]);
```

CSekrt4H returns the grid scale factor along a meridian at the geodetic location specified by the **ll** argument. Analytical formulas for this value have not been located and the result is arrived at empirically the use of spherical trigonometry.

CSekrt4C Convergence angle

```
double CSekrt4C (Const struct cs_Ekrt4_ *ekrt4,Const double ll [2]);
```

CSekrt4C returns the convergence angle in degrees east of north of the geodetic location specified by the **ll** argument. Analytical formulas for this value have not been located and the result is arrived at through the use of the *CS_azsphr* function.

CSekrt4Q definition Quality check

```
int CSekrt4Q (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CSekrt4Q determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Eckert IV Projection. *CS_cschk* examines those definition components that are common to all coordinates system (datum or ellipsoid reference, map scale, and units) and, therefore, *CSekrt4Q* only examines those components specific to the Eckert IV Projection. *CSekrt4Q* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSekrt4Q* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSekrt4L Latitude/longitude check

```
int CSekrt4L (Const struct cs_Ekrt4_ *ekrt4,int cnt,Const double pnts
[][3]);
```

CSekrt4L determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **ekrt4** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSekrt4sL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSekrt4L* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. *cs_CNVRT_DOMN* is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSekrt4X Xy coordinate check

```
int CSekrt4X (Const struct cs_Ekrt4_ *ekrt4,int cnt,Const double pnts
[][3]);
```

CSekrt4X determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **ekrt4** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSekrt4sX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSekrt4L* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSekrt4S Setup (general)

```
void CSekrt4S (struct cs_Csprm_ *csprm);
```

The *CSekrt4S* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the central meridian and other projection parameters are known, there are many calculations that need only be performed once. *CSekrt4S* performs these calculations and saves the results in the *cs_Csprm_* structure provided by its argument, **csprm**. Thus, the argument provided to *CSekrt4S* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. The following parameters are used:

org_lng	The longitude, in degrees, of the origin of the projection (central meridian).
scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size. This is the X coordinate of the coordinate system origin.
y_off	The false northing to be applied to all Y coordinates. This is the Y coordinate of the coordinate system origin.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The value of equatorial radius is extracted from the *datum* element of the `cs_Csprm_` structure and used as the radius of the sphere. This is normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required element is:

e_rad	The radius of the earth, as a sphere, in meters.

cs_Ekrt4 Structure

The results of the one-time calculations are recorded in the *ekrt4* element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSekrt4F*, *CSekrt4I*, *CSekrt4K*, *CSekrt4H,* and *CSekrt4C* functions require as their first argument.

Eckert VI Projection (CSekrt6)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Eckert VI Projection. This projection is supported in spherical form only. The equatorial radius of the supplied ellipsoid is used as the radius of the sphere.

CSekrt6F Forward conversion

```
int CSekrt6F (Const struct cs_Ekrt6_ *ekrt6,double xy [2],Const double ll
[2]);
```

Given a properly initialized *cs_Ekrt6_* structure via the *ekrt6* argument, *CSekrt6F* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSekrt6F* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate

system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSekrt6I Inverse conversion

```
int CSekrt6I (Const struct cs_Ekrt6_ *ekrt6,double ll [2],Const double xy
[2]);
```

Given a properly initialized *cs_Ekrt6_* structure via the *ekrt6* argument, *CSekrt6I* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSekrt6I* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSekrt6K parallel scale (K)

```
double CSekrt6K (Const struct cs_Ekrt6_ *ekrt6,Const double ll [2]);
```

CSekrt6K returns the grid scale factor along a parallel of any coordinate system based on this projection at any location. Analytical formulas for this value have not been located and the result is arrived at empirically the use of spherical trigonometry.

CSekrt6H meridian scale (H)

```
double CSekrt6H (Const struct cs_Ekrt6_ *ekrt6,Const double ll [2]);
```

CSekrt6H returns the grid scale factor along a meridian at the geodetic location specified by the **ll** argument. Analytical formulas for this value have not been located and the result is arrived at empirically the use of spherical trigonometry.

CSekrt6C Convergence angle

```
double CSekrt6C (Const struct cs_Ekrt6_ *ekrt6,Const double ll [2]);
```

CSekrt6C returns the convergence angle in degrees east of north of the geodetic location specified by the **ll** argument. Analytical formulas for this value have not been located and the result is arrived at through the use of the *CS_azsphr* function.

CSekrt6Q definition Quality check

```
int Csekrt6Q (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CSekrt6Q determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Eckert VI Projection. *CS_cschk* examines those definition components that are common to all coordinates system (datum or ellipsoid reference, map scale, and units) and, therefore, *CSekrt6Q* only examines those components specific to the Eckert VI Projection. *CSekrt6Q* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSekrt6Q* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSekrt6L Latitude/longitude check

```
int Csekrt6L (Const struct cs_Ekrt6_ *ekrt4,int cnt,Const double pnts
[][3]);
```

CSekrt6L determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **ekrt6** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSekrt6sL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSekrt6L* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSekrt6X Xy coordinate check

```
int Csekrt6X (Const struct cs_Ekrt6_ *ekrt4,int cnt,Const double pnts
[][3]);
```

CSekrt6X determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **ekrt6** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSekrt6sX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSekrt6L* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSekrt6S Setup (general)

```
void CSekrt6S (struct cs_Csprm_ *csprm);
```

The *CSekrt6S* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the central meridian and other projection parameters are known, there are many calculations which need only be performed once. *CSekrt6S* performs these calculations and saves the results in the *cs_Csprm_* structure provided by its argument, **csprm**. Thus, the argument provided to *CSekrt6S* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. The following parameters are used:

org_lng	The longitude, in degrees, of the origin of the projection (central meridian).
scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size. This is the X coordinate of the coordinate system origin.
y_off	The false northing to be applied to all Y coordinates. This is the Y coordinate of the coordinate system origin.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The value of equatorial radius is extracted from the *datum* element of the `cs_Csprm_` structure and used as the radius of the sphere. This is normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required element is:

e_rad	The radius of the earth, as a sphere, in meters.

cs_Ekrt6_ Structure

The results of the one-time calculations are recorded in the `ekrt6` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSekrt6F*, *CSekrt6I*, *CSekrt6K*, *CSekrt6H,* and *CSekrt6C* functions require as their first argument.

Gnomonic Projection (CSgnomc)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Gnomonic Projection. This projection is supported in spherical form only. The equatorial radius of the supplied ellipsoid is used as the radius of the sphere.

The Gnomonic projection cannot process locations that are 90 degrees or more away from the projection origin. Coordinates that exceed this limit are adjusted to fall on the great circle that defines this limit.

CsgnomcF Forward conversion

```
int CSgnomcF (Const struct cs_Gnomc_ *gnomc,double xy [2],Const double ll
```

```
[2]);
```

Given a properly initialized cs_Gnomc_ structure via the **gnomc** argument, *CSgnomcF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSgnomcF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CsgnomcI Inverse conversion

```
int CSgnomcI (Const struct cs_Gnomc_ *gnomc,double ll [2],Const double xy
[2]);
```

Given a properly initialized cs_Gnomc_ structure via the **gnomc** argument, *CSgnomcI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSgnomcI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CsgnomcK grid scale (K) normal to radial

```
double CSgnomcK (Const struct cs_Gnomc_ *gnomc,Const double ll [2]);
```

CSgnomcK returns the grid scale factor normal to the radial at the geodetic location specified by the **ll** argument.

CsgnomcH grid scale (H) along radial

```
double CSgnomcH (Const struct cs_Gnomc_ *gnomc,Const double ll [2]);
```

CSgnomcH returns the grid scale factor along a radial from the coordinate system origin to (and at) the geodetic location specified by the **ll** argument.

CsgnomcC Convergence angle

```
double CSgnomcC (Const struct cs_Gnomc_ *gnomc,Const double ll [2]);
```

CSgnomcC returns the convergence angle in degrees east of north of the geodetic location specified by the **ll** argument. Analytical formulas for this value have not been located and the result is arrived at through the use of the *CS_azsphr* function.

CsgnomcL Latitude/longitude check

```
int CSgnomcL (Const struct cs_Gnomc_ *gnomc,int cnt,Const double pnts
[][3]);
```

CSgnomcL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **gnomc** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSgnomcsL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSgnomcL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSgnomcQ definition Quality check

```
int CSgnomcQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
```

```
int *err_list [],int list_sz);
```

CSgnomcQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Gnomonic Projection. *CS_cschk* examines those definition components that are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CSgnomcQ* only examines those components specific to the Gnomonic Projection. *CSgnomcQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSgnomcQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CsgnomcX Xy coordinate check

```
int CSgnomcX (Const struct cs_Gnomc_ *gnomc,int cnt,Const double pnts
[][3]);
```

CSgnomcX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **gnomc** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSgnomcsX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSgnomcL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSgnomcS Setup

```
void CSgnomcS (struct cs_Csprm_ *csprm);
```

The *CSgnomcS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the origin latitude and longitude and other projection parameters are known, there are many calculations that need only be performed once. *CSgnomcS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the argument provided to *CSgnomcS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. The following parameters are used:

org_lng	The longitude, in degrees, of the origin of the projection.
org_lat	The latitude, in degrees, of the origin of the projection.
scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size. This is the X coordinate of the coordinate system origin.
y_off	The false northing to be applied to all Y coordinates. This is the Y coordinate of the coordinate system origin.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The value of equatorial radius is extracted from the *datum* element of the `cs_Csprm_` structure and used as the radius of the sphere. This is normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required element is:

e_rad	The radius of the earth, as a sphere, in meters.

cs_Gnomc_ Structure

The results of the one-time calculations are recorded in the `gnomc` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSgnomcF*, *CSgnomcI*, *CSgnomcK*, *CSgnomcH*, and *CSgnomcC* functions require as their first argument.

Goode Homolosine Projection (CShmlsn)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Goode Homolosine Projection. This projection is supported in spherical form only. The equatorial radius of the supplied ellipsoid is used as the radius of the sphere.

CShmlsnF Forward conversion

```
int CShmlsnF (Const struct cs_Hmlsn_ *hmlsn,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Hmlsn_` structure via the **hmlsn** argument, *CShmlsnF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy**

array. *CShmlsnF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CShmlsnI Inverse conversion

```
int CShmlsnI (Const struct cs_Hmlsn_ *hmlsn,double ll [2],Const double xy
[2]);
```

Given a properly initialized cs_Hmlsn_ structure via the **hmlsn** argument, *CShmlsnI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CShmlsnI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CShmlsnK parallel scale (K)

```
double CShmlsnK (Const struct cs_Hmlsn_ *hmlsn,Const double ll [2]);
```

CShmlsnK returns the grid scale factor along a parallel of any coordinate system based on this projection at any location. Analytical formulas for this value have not been located and the result is arrived at empirically the use of spherical trigonometry.

CShmlsnH meridian scale (H)

```
double CShmlsnH (Const struct cs_Hmlsn_ *hmlsn,Const double ll [2]);
```

CShmlsnH returns the grid scale factor along a meridian at the geodetic location specified by the **ll** argument. Analytical formulas for this value have not been located and the result is arrived at empirically the use of spherical trigonometry.

CShmlsnC Convergence angle

```
double CShmlsnC (Const struct cs_Hmlsn_ *hmlsn,Const double ll [2]);
```

CShmlsnC returns the convergence angle in degrees east of north of the geodetic location specified by the **ll** argument. Analytical formulas for this value have not been located and the result is arrived at through the use of the *CS_azsphr* function.

CShmlsnQ definition Quality check

```
int CShmlsnQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CShmlsnQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Goode Homolosine Projection. *CS_cschk* examines those definition components that are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CShmlsnQ* only examines those components specific to the Goode Homolosine Projection. *CShmlsnQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CShmlsnQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CShmlsnL Latitude/longitude check

```
int CShmlsnL (Const struct cs_Hmlsn_ *hmlsn,int cnt,Const double pnts
```

```
[][3]);
```

CShmlsnL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **hmlsn** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CShmlsnsL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CShmlsnL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates is outside of the mathematical domain of the coordinate system.

CShmlsnX Xy coordinate check

```
int CShmlsnX (Const struct cs_Hmlsn_ *hmlsn,int cnt,Const double pnts
[][3]);
```

CShmlsnX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **hmlsn** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CShmlsnsX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CShmlsnL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CShmlsnS Setup (general)

```
void CShmlsnS (struct cs_Csprm_ *csprm);
```

The *CShmlsnS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the origin longitude and other projection parameters are known, there are many calculations which need only be performed once. *CShmlsnS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the argument provided to *CShmlsnS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. The following parameters are used:

org_lng	The longitude, in degrees, of the origin of the projection (central meridian).
prj_prm1-24	The interrupted form of the Goode Homolosine Projection is fully supported. `cs_Csdef_` elements prj_prm1 thru prj_prm24 can be used to specify the extents of the different zones. See *CS_zones* for information on how to encode zones.
scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size. This is the X coordinate of the coordinate system origin.
y_off	The false northing to be applied to all Y coordinates. This is the Y coordinate of the coordinate system origin.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The value of equatorial radius is extracted from the *datum* element of the `cs_Csprm_` structure and used as the radius of the sphere. This is normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required element is:

e_rad	The radius of the earth, as a sphere, in meters.

cs_Hmlsn_ Structure

The results of the one-time calculations are recorded in the `hmlsn` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CShmlsnF*, *CShmlsnI*, *CShmlsnK*, *CShmlsnH,* and *CShmlsnC* functions require as their first argument.

Hotine Oblique Mercator Projection (CSoblqm)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Hotine Oblique Mercator Projection. Since this projection is conformal, the K and H scale factors are the same and there is no H function. Six variations of this projection are supported.

CSoblqmF Forward conversion

```
int CSoblqmF (Const struct cs_Oblqm_ *oblqm,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Oblqm_` structure via the **oblqm** argument, *CSoblqmF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSoblqmF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSoblqmI Inverse conversion

```
int CSoblqmI (Const struct cs_Oblqm_ *oblqm,double ll [2],Const double xy
[2]);
```

Given a properly initialized `cs_Oblqm_` structure via the **oblqm** argument, *CSoblqmI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSoblqmI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are the first elements in these arrays, the Y coordinate and the latitude are the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSoblqmK scale (K)

```
double CSoblqmK (Const struct cs_Oblqm_ *oblqm,Const double ll [2]);
```

CSoblqmK returns the grid scale factor, along a parallel, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array. This is calculated using *CS_llazdd* as we have been unable to locate definitive formulas for the grid scale factor for this projection.

CSoblqmC Convergence angle

```
double CSoblqmC (Const struct cs_Oblqm_ *oblqm,Const double ll [2]);
```

CSoblqmC returns the convergence angle is degrees east of north of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array. This is calculated using *CS_llazdd* as we have been unable to locate definitive formulas for the convergence angle for this projection.

CSoblqmQ definition Quality check

```
int CSoblqmQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CSoblqmQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Oblique Mercator (Hotine) Projection. *CS_cschk* examines those definition components that are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CSoblqmQ* only examines those components specific to the Oblique Mercator (Hotine) Projection. *CSoblqmQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSoblqmQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSoblqmL Latitude/longitude check

```
int CSoblqmL (Const struct cs_Oblqm_ *oblqm,int cnt,Const double pnts
[][3]);
```

CSoblqmL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **oblqm** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSoblqmsL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. **CSoblqmL** returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSoblqmX Xy coordinate check

```
int CSoblqmX (Const struct cs_Oblqm_ *oblqm,int cnt,Const double pnts
[][3]);
```

CSoblqmX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **oblqm** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSoblqmsX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSoblqmL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSoblqmS Setup

```
void CSoblqmS (struct cs_Csprm_ *csprm);
```

The *CSoblqmS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the central great circle, the origin latitude, and other projection parameters are known, there are many calculations that need only be performed once. *CSoblqmS* performs these calculations and saves the results in the *cs_Csprm_* structure provided by the **csprm** argument. Thus, this one argument provides *CSoblqmS* its input data and the repository for the results as described below.

CSoblqmS examines the `prj_code` element of the `cs_Csprm_` structure to determine which of the six variations of this projection is to be setup. In most cases, the variations require different usage of the parameters in the `cs_Csdef_` structure as defined below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csldef* function; but can be provided by the application at run time. The specific elements of the `cs_Csdef_` structure that must be initialized for the Oblique Mercator projection are dependent upon the variation being implemented.

The following parameters apply to all six variations of the projection:

scl_red	The scale reduction that is to be applied.
Scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size.
y_off	The false northing to be applied to all Y coordinates.
Quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Single Point, Unrectified Hotine Oblique Mercator (cs_PRJCOD_HOM1UV)

This variation produces unrectified cartesian coordinates whose origin is the intersection of the central geodesic and the equator of the "aposphere". Rarely, if ever used; retained primarily for historical purposes.

prj_prm1	Longitude, in degrees, of the central point of the projection.
prj_prm2	Latitude, in degrees, of the central point of the projection.
prj_prm3	The azimuth of the central great circle, in degrees east of north.

Two Point, Unrectified Hotine Oblique Mercator (cs_PRJCOD_HOM2UV)

This variation produces unrectified cartesian coordinates whose origin is the intersection of the central geodesic and the equator of the "aposphere." Rarely, if ever used; retained primarily for historical purposes.

prj_prm1	Longitude, in degrees, of the first point on the central geodesic.
prj_prm2	Latitude, in degrees, of the first point on the central geodesic.
prj_prm3	Longitude, in degrees, of the second point on the central geodesic.
prj_prm4	Latitude, in degrees, of the second point on the central geodesic.
org_lat	The latitude, in degrees, of the center of the coordinate system being defined. That is, the point on the central geodesic that has this latitude is the natural origin of the projection.

Alaska Variation, Hotine Oblique Mercator (cs_PRJCOD_HOM1XY)

This variation produces rectified cartesian coordinates whose origin is the intersection of the central geodesic and the equator of the "aposphere". The rectification technique is specific to Zone 1 of the Alaska State Plane Coordinate System. It is possible that this variation should also be used for the Great Lakes Survey, but this has not been verified as of this writing.

prj_prm1	Longitude, in degrees, of the central point of the projection.
prj_prm2	Latitude, in degrees, of the central point of the projection.
prj_prm3	The azimuth of the central great circle, in degrees east of north.

Two Point, Rectified Hotine Oblique Mercator (cs_PRJCOD_HOM2XY)

This variation produces rectified cartesian coordinates whose origin is the intersection of the central geodesic and the equator of the "aposphere". Rarely, if ever used; retained primarily for historical purposes. To remain consistent with prior releases of CS-MAP, this variation uses the same rectification technique as the Alaska variation described immediately above.

prj_prm1	Longitude, in degrees, of the first point on the central geodesic.
prj_prm2	Latitude, in degrees, of the first point on the central geodesic.
prj_prm3	Longitude, in degrees, of the second point on the central geodesic.
prj_prm4	Latitude, in degrees, of the second point on the central geodesic.
org_lat	The latitude, in degrees, of the center of the coordinate system being defined. That is, the point on the central geodesic that has this latitude is the natural origin of the projection.

Rectified Skew Orthomorphic (cs_PRJCOD_RSKEW)

This variation produces rectified cartesian coordinates whose origin is the intersection of the central geodesic and the equator of the "aposphere". The rectification technique is that commonly used in places other than Alaska.

prj_prm1	Longitude, in degrees, of the central point of the projection.
prj_prm2	Latitude, in degrees, of the central point of the projection.
prj_prm3	The azimuth of the central great circle, in degrees east of north.

Rectified Skew Orthomorphic Centered (cs_PRJCOD_RSKEWC)

This variation produces rectified cartesian coordinates, the origin of which is at the single defining point. The rectification technique is that commonly used in places other than Alaska.

prj_prm1	Longitude, in degrees, of the central point of the projection.
prj_prm2	Latitude, in degrees, of the central point of the projection.
prj_prm3	The azimuth of the central great circle, in degrees east of north.

Datum Definition

The values of equatorial radius and eccentricity are extracted from the *datum* element of the *cs_Csprm_* structure. These are normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required elements are:

e_rad	The equatorial radius of the earth in meters.
eccent	This value represents the eccentricity of the ellipsoid.
to84_via	An integer code that specifies the technique that is to be used to convert geographic coordinates based on this datum to WGS84.

cs_Oblqm_ Structure

The results of the one-time calculations are recorded in the `oblqm` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSoblqmF*, *CSoblqmI*, *CSoblqmK*, and *CSoblqmC* functions require as their first argument.

Krovak Oblique Conformal Conic

This set of functions represent the Coordinate System Mapping Package's knowledge of the Krovak Oblique Conformal Conic Projection. This projection is used in what used to be Czechoslovokia, and is now the Czech Republic and the Slovak Republic. Since this projection is conformal, the K and H scale factors are the same and there is no H function. Two variations of this projection are supported.

The first variation is the traditional projection as used since the 1920's. The second includes the affine transformation devised for use with the 1995 adjustment.

Please note that traditional Krovak X coordinates increase to the west. WHen such coordinates are used in the traditional CAD environment, the resuling images are mirrored and are (for most folks anyway) useless. Therefore, this implementation is such that what would normally be positive X coordinates are actually negative coordinates, and the magnitude of the values will be correct. In this way, the absolute value of the coordinate will be what is expected to see, but the coordinates will actually increase to the east, thus making AutoCAD, MicroStation, and other CAD type systems happy campers.

CSkrovkF Forward Conversion

```
int CSkrovkF (Const struct cs_Krovk_ *krovk,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Krovk_` structure via the **krovk** argument, *CSkrovkF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSkrovkF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

The **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are the first elements in these arrays, the Y coordinate and the latitude are the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSkrovkI Inverse conversion

```
int CSkrovkI (Const struct cs_Krovk_ *krovk,double ll [2],Const double xy
[2]);
```

Given a properly initialized `cs_Krovk_` structure via the **krovk** argument, *CSkrovkI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSkrovkI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system.

The **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are the first elements in these arrays, the Y coordinate and the latitude are the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSkrovkK scale (K)

```
double CSkrovkK (Const struct cs_Krovk_ *krovk,Const double ll [2]);
```

CSkrovkK returns the grid scale factor of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array. The value `cs_SclInf` (defined to be 9,999.00 is *CSdata.c*) is returned if the **ll** provided is the oblique pole.

CSkrovkC Convergence angle

```
double CSkrovkC (Const struct cs_Krovk_ *krovk,Const double ll [2]);
```

CSkrovkC returns the convergence angle in degrees east of north of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array.

CSkrovkQ definition Quality check

```
int CSkrovkQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CSkrovkQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Krovak Oblique Conformal Conic Projection. *CS_cschk* examines those definition components that are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CSkrovakQ* only examines those components specific to the Krovak Oblique Conformal Conic Projection. *CSkrovkQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSkrovkQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

Bugs

In the original implementation of this projection, all parameters were hard coded and no checking was necessary. For release 10, this projection was rewritten to accept user defined parameters, but this Quailty check function was never updated. Therefore, at the current time, there is no parameter checking performed for this projection.

CSkrovkL Latitude/longitude check

```
int CSoblqmL (Const struct cs_Oblqm_ *oblqm,int cnt,Const double pnts
[][3]);
```

CSoblqmL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the

coordinate system provided by the **oblqm** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSoblqmsL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. **CSoblqmL** returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSkrovkX Xy coordinate check

```
int CSoblqmX (Const struct cs_Oblqm_ *oblqm,int cnt,Const double pnts
[][3]);
```

At the current time, *CSoblqmX* returns **cs_CNVRT_OK** without performing any checks. The arguments are currently ignored. Again, this is due to the unusual legacy of this projection, and the fact that normal coordinates used in the Czech Repulic increase to the west rather than the east.

CSkrovkS Setup

```
void CSkrovkS (struct cs_Csprm_ *csprm);
```

The *CSkrovkS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the origin longitude, origin latitude, oblique pole location, and other projection parameters are known, there are many calculations that need only be performed once. *CSkrovkS* performs these calculations and saves the results in the *cs_Csprm_* structure provided by the **csprm** argument. Thus, this one argument provides *CSkrovkS* its input data and the repository for the results as described below.

CSkrovkS examines the `prj_code` element of the `cs_Csprm_` structure to determine which of the two variations of this projection is to be setup. The projection code, therefore, simply determines if the 1995 adjustment transformation is applied to the resulting cartesian coordinates. The parameters for the affine tranformation are (currently) hard coded.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csldef* function; but can be provided by the application at run time. The specific elements of the `cs_Csdef_` structure that must be initialized for the Krovak Oblique Conformal Conic Projection are dependent upon the variation being implemented.

The following parameters apply to both variations of the projection:

org_lng	Longitude, in degrees, of the origin of the projection. As is commonly used in the Czech Repoublic, this is usually the prime meridian of Ferro.
org_lat	Latitude, in degrees, of the origin of the projection.
prj_prm1	Longitude, in degrees, of the location of the pole of the oblique cone.
prj_prm2	Latitude, in degrees, of the location of the pole of the oblique cone.
prj_prm3	Latitude, in degrees on the oblique gaussian surface, of the single standard parallel of the conic projection surface.
scl_red	The scale reduction that is to be applied.
Scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size.
y_off	The false northing to be applied to all Y coordinates.
Quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

*Krovak Oblique Conformal Conic, Czechoslovokia (cs_PRJCOD_*

This variation produces the traditional (i.e. unadjusted) Krovak coordinates used in Czechoslovakia since the 1920's. There are no special parameter requirements. Specifying this variation simply turns off the application of the 1995 adjustment.

Krovak Oblique Conformal Conic/95 Adjustment

This variation causes an affine transformation to be applied to the traditional coordinates, thus producing coordinates appropriate for the 1995 adjustment. There are no special parameter requirements (at this time). Specifiying this variation simply turns on the affine transformation, the coefficients of which are hard coded.

Datum Definition

The values of equatorial radius and eccentricity are extracted from the *datum* element of the *cs_Csprm_* structure. These are normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required elements are:

e_rad	The equatorial radius of the earth in meters.
eccent	This value represents the eccentricity of the ellipsoid.
to84_via	An integer code that specifies the technique that is to be used to convert geographic coordinates based on this datum to WGS84.

cs_Krovk_ Structure

The results of the one-time calculations are recorded in the `oblqm` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSkrovkF*, *CSkrovkI*, *CSkrovkK*, and *CSkrovkC* functions require as their first argument.

Lambert Conformal Conic Projection (CSlmbrt)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Lambert Conformal Conic Projection. Since this projection is a conformal projection, the K and H scale factors are the same and there is no H function. Five variations of this projection are supported.

CSlmbrtF Forward conversion

```
int CSlmbrtF (Const struct cs_Lmbrt_ *lmbrt,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Lmbrt_` structure via the **lmbrt** argument, *CSlmbrtF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSlmbrtF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSlmbrtI Inverse conversion

```
int CSlmbrtI (Const struct cs_Lmbrt_ *lmbrt,double ll [2],Const double xy
[2]);
```

Given a properly initialized `cs_Lmbrt_` structure via the **lmbrt** argument, *CSlmbrtI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSlmbrtI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSlmbrtK scale (K)

```
double CSlmbrtK (Const struct cs_Lmbrt_ *lmbrt,Const double ll [2]);
```

CSlmbrtK returns the grid scale factor, along a parallel, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array.

CSlmbrtC Convergence angle

```
double CSlmbrtC (Const struct cs_Lmbrt_ *lmbrt,Const double ll [2]);
```

CSlmbrtC returns the convergence angle of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array.

CSlmbrtQ definition Quality check

```
int CSlmbrtQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CSlmbrtQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Lambert Conformal Conic Projection. *CS_cschk* examines those definition components that are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CSlmbrtQ* only examines those components specific to the Lambert Conformal Conic Projection. *CSlmbrtQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSlmbrtQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSlmbrtL Latitude/longitude check

```
int CSlmbrtL (Const struct cs_Lmbrt_ *lmbrt,int cnt,Const double pnts
[][3]);
```

CSlmbrtL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **lmbrt** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSlmbrtsL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSlmbrtL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSlmbrtX Xy coordinate check

```
int CSlmbrtX (Const struct cs_Lmbrt_ *lmbrt,int cnt,Const double pnts
[][3]);
```

CSlmbrtX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **lmbrt** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSlmbrtsX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSlmbrtL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSlmbrtS Setup

```
void CSlmbrtS (struct cs_Csprm_ *csprm);
```

The *CSlmbrtS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the standard parallels, the origin latitude and longitude, and other projection parameters are known, there are many calculations that need only be performed once. *CSlmbrtS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the single argument provided to *CSlmbrtS* serves as the

source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time.

Five variations of this projection are supported. *CSlmbrtS* determines which variation is to be setup by examining the `prj_code` element of the `cs_Csprm_` structure. The actual use of parameters in the `cs_Csdef_` structure (an element of the `cs_Csprm_` structure) is dependent on the variation being setup.

The following elements of the `cs_Csdef_` structure apply to all five variations:

Scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size.
y_off	The false northing to be applied to all Y coordinates.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Parameter use specific to the five variations is:

`Two Standard Parallels (cs_PRJCOD_LMBRT):`

This is the traditional version of this projection. The degree of scale reduction to reduce and distribute scale distorion is specified by two standard parallels:

prj_prm1	Latitude, in degrees, of the first standard parallel, usually the northernmost. For this projection, there is no distinction between the northern and southern standard parallels (i.e. they can be switched with no affect).
prj_prm2	Latitude, in degrees, of the second standard parallel, usually the southernmost. This is, rarely, the same as prj_prm1, to obtain a conic with a single point of tangency.
org_lng	The longitude, in degrees, of the origin of the projection.
org_lat	The latitude, in degrees, of the origin of the projection.

Single Standard Parallel (cs_PRJCOD_LM1SP):

This variation is commonly used outside of North America. It is, mathematically, virtually identical to what CS-MAP has long referred to as the Lambert Tangential. The degree of scale reduction to reduce and distribute scale distortion is specified by the scale reduction factor:

org_lng	The longitude, in degrees, of the origin of the projection.
org_lat	The latitude, in degrees, of the origin of the projection.
scl_red	The scale of the projection at the origin defined by org_lng and org_lat.

Belgian Variation (cs_PRJCOD_LMBLGN):

This is a minor variation to the traditional Two Standard Parallel version of the projection. This variation will produce the results required by some Belgian coordinate systems:

prj_prm1	Latitude, in degrees, of the first standard parallel, usually the northernmost. For this projection, there is no distinction between the northern and southern standard parallels (i.e. they can be switched with no affect).
prj_prm2	Latitude, in degrees, of the second standard parallel, usually the southernmost. This is, rarely, the same as prj_prm1, to obtain a conic with a single point of tangency.
org_lng	The longitude, in degrees, of the origin of the projection.
org_lat	The latitude, in degrees, of the origin of the projection.

Wisconsin Variation (cs_PRJCOD_WCCSL):

This is a minor variation to the traditional Two Standard Parallel version of the projection. This variation supports the Wisconsin County Coordinate System group of coordinate systems. This variation uses a parallel ellipsoid technique to adjust horizontal coordinates for average elevation of the region being mapped:

prj_prm1	Latitude, in degrees, of the first standard parallel, usually the northernmost. For this projection, there is no distinction between the northern and southern standard parallels (i.e. they can be switched with no affect).
prj_prm2	Latitude, in degrees, of the second standard parallel, usually the southernmost. This is, rarely, the same as prj_prm1, to obtain a conic with a single point of tangency.
prj_prm3	Average geoid separation, in meters, of the region being mapped.
prj_prm4	Average elevation above the geoid (i.e. orthometric height), in system units, of the region being mapped.
org_lng	The longitude, in degrees, of the origin of the projection.
org_lat	The latitude, in degrees, of the origin of the projection.

Minnesota Variation (cs_PRJCOD_MNDOTL):

This is a minor variation to the traditional Two Standard Parallel version of the projection. This variation supports the county coordinate systems developed by the Minnesota Department of Transportation. This variation uses a parallel ellipsoid technique (different from that used in Wisconsin, of course) to adjust horizontal coordinates for average elevation of the region being mapped:

prj_prm1	Latitude, in degrees, of the first standard parallel, usually the northernmost. For this projection, there is no distinction between the northern and southern standard parallels (i.e. they can be switched with no affect).
prj_prm2	Latitude, in degrees, of the second standard parallel, usually the southernmost. This is, rarely, the same as prj_prm1, to obtain a conic with a single point of tangency.
prj_prm3	Average height above the ellipsoid, in system units, of the region being mapped.
org_lng	The longitude, in degrees, of the origin of the projection.
org_lat	The latitude, in degrees, of the origin of the projection.

Datum Definition

The values of equatorial radius and eccentricity are extracted from the *datum* element of the *cs_Csprm_* structure. These are normally obtained from the Ellipsoid Dictionary by the *CS_dtloc*

function, but may be supplied by the application at run time. Specifically, the required elements are:

e_rad	The equatorial radius of the earth in meters.
eccent	This value represents the eccentricity of the ellipsoid.
to84_via	An integer code that specifies the technique that is to be used to convert geographic coordinates based on this datum to WGS84.

cs_Lmbrt_ Structure

The results of the one-time calculations are recorded in the `lmbrt` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSlmbrtF*, *CSlmbrtI*, *CSlmbrtK*, and *CSlmbrtC* functions require as their first argument.

Lambert Tangential Projection (CSlmtan)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Lambert Tangential Projection as used by the National Geographic Institute of France. Please note that the current implementation of this projection does not support the spherical form of the projection. With the addition of the single standard parallel variation of the Lambert Conformal Conic, these functions are now redundant; and will be removed in a future release.

CSlmtanF Forward conversion

```
int CSlmtanF (Const struct cs_Lmtan_ *lmtan,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Lmtan_` structure via the **lmtan** argument, *CSlmtanF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSlmtanF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSlmtanI Inverse conversion

```
int CSlmtanI (Const struct cs_Lmtan_ *lmtan,double ll [2],Const double xy
[2]);
```

Given a properly initialized `cs_Lmtan_` structure via the **lmtan** argument, *CSlmtanI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSlmtanI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSlmtanK parallel scale (K)

```
double CSlmtanK (Const struct cs_Lmtan_ *lmtan,Const double ll [2]);
```

CSlmtanK returns the grid scale factor, along a parallel, of the coordinate system at the geodetic

location given by the **ll** argument. Formulas for this calculation have not been located, therefore the result is obtained using the *CS_llazdd* function.

CSlmtanH meridian scale (H)

```
double CSlmtanH (Const struct cs_Lmtan_ *lmtan,Const double ll [2]);
```

CSlmtanH returns the grid scale factor, along a meridian, of the coordinate system at the geodetic location given by the **ll** argument. Formulas for this calculation have not been located, therefore the result is obtained from the use of the *CS_llazdd* function.

CSlmtanC Convergence angle

```
double CSlmtanC (Const struct cs_Lmtan_ *lmtan,Const double ll [2]);
```

CSlmtanC returns the convergence angle in degrees east of north of the coordinate system at the geodetic location given by the **ll** argument. Formulas for this calculation have not been located, therefore the result is obtained using the *CS_llazdd* function.

CSlmtanL Latitude/longitude check

```
int CSlmtanL (Const struct cs_Lmtan_ *lmtan,int cnt,Const double pnts
[][3]);
```

CSlmtanL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **lmtan** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSlmtansL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSlmtanL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSlmtanQ definition Quality check

```
int CSlmtanQ (Const struct cs_Csdef_ *csdef,iunsigned short prj_code,
              int *err_list [],int list_sz);
```

CSlmtanQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Lambert Tangential Projection. *CS_cschk* examines those definition components that are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CSlmtanQ* only examines those components specific to the Lambert Tangential Projection. *CSlmtanQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSlmtanQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSlmtanX Xy coordinate check

```
int CSlmtanX (Const struct cs_Lmtan_ *lmtan,int cnt,Const double pnts
[][3]);
```

CSlmtanX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **lmtan** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSlmtansX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSlmtanL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate

system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSlmtanS Setup

```
void CSlmtanS (struct cs_Csprm_ *csprm);
```

The *CSlmtanS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the origin latitude and longitude, the scale reduction factor, and other projection parameters are known, there are many calculations that need only be performed once. *CSlmtanS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the single argument provided to *CSlmtanS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. The specific elements of the `cs_Csdef_` structure that must be initialized for the Lambert Tangential projection are:

org_lng	The longitude, in degrees, of the origin of the projection relative to Greenwich.
org_lat	The latitude, in degrees, of the origin of the projection relative to the equator.
scl_red	The scale reduction factor that is to be applied to the projection.
scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size. This value is the X coordinate of the coordinate system origin.
y_off	The false northing to be applied to all Y coordinates. This value is the Y coordinate of the coordinate system origin.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The values of equatorial radius and eccentricity are extracted from the *datum* element of the *cs_Csprm_* structure. These are normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required elements are:

e_rad	The equatorial radius of the earth in meters.
eccent	This value represents the eccentricity of the ellipsoid.
to84_via	An integer code that specifies the technique that is to be used to convert geographic coordinates based on this datum to WGS84.

cs_Lmtan_ Structure

The results of the one-time calculations are recorded in the `lmtan` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSlmtanF*, *CSlmtanI*, *CSlmtanK*, *CSlmtanH*, and *CSlmtanC* functions require as their first argument.

Mercator Projection (CSmrcat)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Mercator Projection. Since this projection is conformal, the K and H grid scales are the same and there is no H function.

CSmrcatF Forward conversion

```
int CSmrcatF (Const struct cs_Mrcat_ *mrcat,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Mrcat_` structure via the **mrcat** argument, *CSmrcatF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSmrcatF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSmrcatI Inverse conversion

```
int CSmrcatI (Const struct cs_Mrcat_ *mrcat,double ll [2],Const double xy
[2]);
```

Given a properly initialized *cs_Mrcat_* structure via the **mrcat** argument, *CSmrcatI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSmrcatI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system.

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSmrcatK scale (K) along a parallel

```
double CSmrcatK (Const struct cs_Mrcat_ *mrcat,Const double ll [2]);
```

CSmrcatK returns the grid scale factor, along a parallel, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array.

CSmrcatC Convergence angle

```
double CSmrcatC (Const struct cs_Mrcat_ *mrcat,Const double ll [2]);
```

CSmrcatC returns the value 0.0 which represents the convergence in degrees east of north of nay coordinate system based on this projection at any latitude and longitude.

CSmrcatQ definition Quality check

```
int CSmrcatQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CSmrcatQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Mercator Projection. *CS_cschk* examines those definition components that are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CSmrcatQ* only examines those components specific to the Mercator Projection. *CSmrcatQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSmrcatQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSmrcatL Latitude/longitude check

```
int CSmrcatL (Const struct cs_Mrcat_ *mrcat,int cnt,Const double pnts
[][3]);
```

CSmrcatL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **mrcat** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSmrcatsL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSmrcatL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSmrcatX Xy coordinate check

```
int CSmrcatX (Const struct cs_Mrcat_ *mrcat,int cnt,Const double pnts
[][3]);
```

CSmrcatX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **mrcat** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSmrcatsX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSmrcatL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSmrcatS Setup

```
void CSmrcatS (struct cs_Csprm_ *csprm);
```

The *CSmrcatS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the standard parallel, the origin longitude, and other projection parameters are known, there are many calculations that need only be performed once. *CSmrcatS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the single argument provided to *CSmrcatS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. The specific elements of the *cs_Csdef_* structure which must be initialized for the Mercator Projection are:

prj_prm1	Longitude, in degrees, of the central meridian of the coordinate system (or map).
prj_prm2	Latitude, in degrees, of the standard parallel, usually zero indicating the equator. Using a non-zero value has an affect similar to that of the scale reduction factor of other cylindrical projections.
scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size.
y_off	The false northing to be applied to all Y coordinates.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The values of equatorial radius and eccentricity are extracted from the *datum* element of the *cs_Csprm_* structure. These are normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required elements are:

e_rad	The equatorial radius of the earth in meters.
eccent	This value represents the eccentricity of the ellipsoid.
to84_via	An integer code that specifies the technique that is to be used to convert geographic coordinates based on this datum to WGS84.

cs_Mrcat_ Structure

The results of the one-time calculations are recorded in the `mrcat` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSmrcatF*, *CSmrcatI*,

CSmrcatK, and *CSmrcatC* functions require as their first argument.

Miller Projection (CSmillr)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Miller Projection. This projection is only used in the spherical form. Thus, all functions assume a sphere with a radius equal to the equatorial radius of the ellipsoid provided.

CSmillrF Forward conversion

```
int CSmillrF (Const struct cs_Millr_ *millr,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Millr_` structure via the **millr** argument, *CSmillrF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSmillrF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSmillrI Inverse conversion

```
int CSmillrI (Const struct cs_Millr_ *millr,double ll [2],Const double xy
[2]);
```

Given a properly initialized `cs_Millr_` structure via the **millr** argument, *CSmillrI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSmillrI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system.

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSmillrK scale (K) along a parallel

```
double CSmillrK (Const struct cs_Millr_ *millr,Const double ll [2]);
```

CSmillrK returns the grid scale factor, along a parallel, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array.

CSmillrH scale (H) along a meridian

```
double CSmillrH (Const struct cs_Millr_ *millr,Const double ll [2]);
```

CSmillrH returns the grid scale factor, along a meridian, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array.

CSmillrC Convergence angle

```
double CSmillrC (Const struct cs_Millr_ *millr,Const double ll [2]);
```

CSmillrC returns the value 0.0 which represents the convergence angle in degrees east of north of any coordinate system based on this projection at any latitude and longitude.

CSmillrL Latitude/longitude check

```
int CSmillrL (Const struct cs_Millr_ *millr,int cnt,Const double pnts
[][3]);
```

CSmillrL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **millr** argument. The **pnts** and **cnt** arguments can define a single

coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSmillrsL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSmillrL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSmillrQ definition Quality check

```
int CSmillrQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CSmillrQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Miller Cylindrical Projection. *CS_cschk* examines those definition components that are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CSmillrQ* only examines those components specific to the Miller Cylindrical Projection. *CSmillrQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSmillrQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSmillrX Xy coordinate check

```
int CSmillrX (Const struct cs_Millr_ *millr,int cnt,Const double pnts
[][3]);
```

CSmillrX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **millr** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSmillrsX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSmillrL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSmillrS Setup

```
void CSmillrS (struct cs_Csprm_ *csprm);
```

The *CSmillrS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the origin longitude and other projection parameters are known, there are many calculations that need only be performed once. *CSmillrS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the single argument provided to *CSmillrS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. The specific elements of the `cs_Csdef_` structure that must be initialized for the Miller Projection are:

prj_prm1	Longitude, in degrees, of the central meridian of the coordinate system (or map).
scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size.
y_off	The false northing to be applied to all Y coordinates.
quad	an integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The value of equatorial radius is extracted from the *datum* element of the `cs_Csprm_` structure and used as the radius of the sphere. This is normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required element is:

e_rad	The radius of the earth, as a sphere, in meters.

cs_Millr_ Structure

The results of the one-time calculations are recorded in the `millr` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSmillrF*, *CSmillrI*, *CSmillrK*, *CSmillrH*, and *CSmillrC* functions require as their first argument.

Modified Polyconic Projection (CSmodpc)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Modified Polyconic Projection. That is, the projection developed by Lallemand of France and adopted by the International Map Committee (IMC) in London as the basis for the 1:1,000,000 scale International Map of the World (IMW) series in 1909.

CSmodpcF Forward conversion

```
int CSmodpcF (Const struct cs_Modpc_ *modpc,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Modpc_` structure via the **modpc** argument, *CSmodpcF* will convert the latitude and longitude provided in the *a* array to X and Y coordinates, returning the result in the **xy** array. *CSmodpcF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSmodpcI Inverse conversion

```
int CSmodpcI (Const struct cs_Modpc_ *modpc,double ll [2],Const double xy
[2]);
```

Given a properly initialized cs_Modpc_ structure via the **modpc** argument, *CSmodpcI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSmodpcI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSmodpcK grid scale (K), along parallel

```
double CSmodpcK (Const struct cs_Modpc_ *modpc,Const double ll [2]);
```

CSmodpcK returns the grid scale factor, as measured along a parallel, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array. (The use of the **ll** array is the same as described above.) At the current time, formulas that analytically produce the grid scale factor for this projection elude us. Thus, the grid scale factor is determined using the *CS_llazdd* function.

CSmodpcH grid scale (H), along meridian

```
double CSmodpcH (Const struct cs_Modpc_ *modpc,Const double ll [2]);
```

CSmodpcH returns the grid scale factor, as measured along a meridian, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array. At the current time, formulas that analytically produce the grid scale factor for this projection elude us. Thus, the grid scale factor is determined using the *CS_llazdd* function.

CSmodpcC Convergence angle

```
double CSmodpcC (Const struct cs_Modpc_ *modpc,Const double ll [2]);
```

CSmodpcC returns the convergence angle of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array. At the current time, formulas that analytically produce the convergence angle for this projection elude us. Thus, the grid scale factor is determined using the *CS_llazdd* function.

CSmodpcB Basic calculations

```
double CSmodpcB (Const struct cs_Modpc_ *modpc,Const double ll [2],double
xy [2],double *their_yc);
```

Inverse calculations for this projection are performed using an iterative algorithm calling the forward function. *CSmodpcB* converts geographic coordinates to cartesian coordinates in a form that can be used by both *CSmodpcF* and *CSmodpcI*; thus eliminating duplicate code in these modules. The **their_yc** argument provides for the return to the calling function of an additional intermediary result that is required for the inverse calculation.

CSmodpcQ definition Quality check

```
int CSmodpcQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CSmodpcQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Modified Polyconic Projection. *CS_cschk* examines those definition components that are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CSmodpcQ* only examines those components specific to the Modified Polyconic Projection. *CSmodpcQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSmodpcQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSmodpcL Latitude/longitude check

```
int CSmodpcL (Const struct cs_Modpc_ *modpc,int cnt,Const double pnts
[][3]);
```

CSmodpcL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **modpc** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSmodpcsL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSmodpcL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSmodpcX Xy coordinate check

```
int CSmodpcX (Const struct cs_Modpc_ *modpc,int cnt,Const double pnts
[][3]);
```

CSmodpcX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **modpc** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSmodpcsX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSmodpcL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSmodpcS Setup

```
void CSmodpcS (struct cs_Csprm_ *csprm);
```

The *CSmodpcS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the standard meridians, the standard parallels, and other projection parameters are known, there are many calculations that need only be performed once. *CSmodpcS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the single argument provided to *CSmodpcS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. The specific elements of the `cs_Csdef_` structure that must be initialized for the Modified Polyconic projection are:

prj_prm1	Longitude, in degrees, of the central meridian.
prj_prm2	Longitude, in degrees, of the Eastern meridian. The Western meridian is assumed to be west of the Central Meridian by the same amount that the Eastern Meridian is east of the Central Meridian. The Eastern meridian must be east of the central meridian, and not more than 15 degrees of longitude from the central meridian.
prj_prm3	Latitude, in degrees, of the Northern Standard Parallel.
prj_prm4	Latitude, in degrees, of the Southern Standard Parallel.
scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size.
y_off	The false northing to be applied to all Y coordinates.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Neither standard parallel may be a pole, nor can the two standard parallels be the same as is supported in other projections. In addition, the Northern Standard Parallel must be the northernmost of the two standard parallels. Finally, the two standard parallels must be within 15 degrees of each other. Note that the projection was designed for maps whose extents are 6 degrees of longitude and 4 degrees of latitude.

Datum Definition

The values of equatorial radius and eccentricity are extracted from the *datum* element of the *cs_Csprm_* structure. These are normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required elements are:

e_rad	The equatorial radius of the earth in meters.
eccent	This value represents the eccentricity of the ellipsoid.
to84_via	An integer code that specifies the technique that is to be used to convert geographic coordinates based on this datum to WGS84.

cs_Modpc_ Structure

The results of the one-time calculations are recorded in the `modpc` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSmodpcF*, *CSmodpcI*, *CSmodpcK*, *CSmodpcH*, *CSmodpcC*, and *CSmodpcB* functions require as their first argument.

BUGS

As is true with all other projections in CS-MAP, values submitted for conversion are not checked for validity before conversion for performance reasons. In every other case, this does not appear to be a problem. However, experience has show that values which are more than 50% outside the area covered by the projection parameters can produce errors which get reported through *matherr*. Checking of input values does need to be added to the functions of this projection.

Mollweide Projection (CSmolwd)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Mollweide Projection. This projection is supported in spherical form only. The equatorial radius of the supplied ellipsoid is used as the radius of the sphere.

CSmolwdF Forward conversion

```
int CSmolwdF (Const struct cs_Molwd_ *molwd,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Molwd_` structure via the **molwd** argument, *CSmolwdF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSmolwdF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSmolwdI Inverse conversion

```
int CSmolwdI (Const struct cs_Molwd_ *molwd,double ll [2],Const double xy
[2]);
```

Given a properly initialized `cs_Molwd_` structure via the **molwd** argument, *CSmolwdI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSmolwdI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSmolwdK parallel scale (K)

```
double CSmolwdK (Const struct cs_Molwd_ *molwd,Const double ll [2]);
```

CSmolwdK returns the grid scale factor along a parallel of any coordinate system based on this projection at any location. Analytical formulas for this value have not been located and the result is arrived at empirically the use of spherical trigonometry.

CSmolwdH meridian scale (H)

```
double CSmolwdH (Const struct cs_Molwd_ *molwd,Const double ll [2]);
```

CSmolwdH returns the grid scale factor along a meridian at the geodetic location specified by the **ll** argument. Analytical formulas for this value have not been located and the result is arrived at empirically the use of spherical trigonometry.

CSmolwdC Convergence angle

```
double CSmolwdC (Const struct cs_Molwd_ *molwd,Const double ll [2]);
```

CSmolwdC returns the convergence angle in degrees east of north of the geodetic location specified by the **ll** argument. Analytical formulas for this value have not been located and the result is arrived at through the use of the *CS_azsphr* function.

CSmolwdQ definition Quality check

```
int CSmolwdQ (Const struct cs_Csdef_ *csdef,unsgined short prj_code,
              int *err_list [],int list_sz);
```

CSmolwdQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Mollweide Projection. *CS_cschk* examines those definition components that are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CSmolwdQ* only examines those components specific to the Mollweide Projection. *CSmolwdQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of *err_list* as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSmolwdQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSmolwdL Latitude/longitude check

```
int CSmolwdL (Const struct cs_Molwd_ *molwd,int cnt,Const double pnts
[][3]);
```

CSmolwdL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **molwd** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSmolwdsL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSmolwdL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSmolwdX Xy coordinate check

```
int CSmolwdX (Const struct cs_Molwd_ *molwd,int cnt,Const double pnts
[][3]);
```

CSmolwdX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **molwd** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSmolwdsX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSmolwdL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSmolwdS Setup (general)

```
void CSmolwdS (struct cs_Csprm_ *csprm);
```

The *CSmolwdS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the origin longitude and other projection parameters are known, there are many calculations that need only be performed once. *CSmolwdS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the argument provided to *CSmolwdS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. The following parameters are used:

org_lng	The longitude, in degrees, of the origin of the projection (central meridian).
prj_prm1-24	The interrupted form of the Mollweide Projection is fully supported. cs_Csdef_ elements prj_prm1 thru prj_prm24 can be used to specify the extents of the different zones. See CS_zones for information on how to encode zones.
scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size. This is the X coordinate of the coordinate system origin.
y_off	The false northing to be applied to all Y coordinates. This is the Y coordinate of the coordinate system origin.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The value of equatorial radius is extracted from the *datum* element of the `cs_Csprm_` structure and used as the radius of the sphere. This is normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required element is:

e_rad	The radius of the earth, as a sphere, in meters.

cs_Molwd_ Structure

The results of the one-time calculations are recorded in the `molwd` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSmolwdF*, *CSmolwdI*, *CSmolwdK*, *CSmolwdH,* and *CSmolwdC* functions require as their first argument.

Modified Stereographic Projection (CSmstro)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Modified Stereographic Projection. Since the Modified Stereographic Projection is a conformal projection, the K (grid scale along a parallel) and H (grid scale along a meridian) are the same. Therefore, there is no H function for this projection.

CSmstroF Forward

```
int CSmstroF (Const struct cs_Mstro_ *mstro,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Mstro_` structure via the **mstro** argument, *CSmstroF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSmstroF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSmstroI Inverse

```
int CSmstroI (Const struct cs_Mstro_ *mstro,double ll [2],Const double xy
[2]);
```

Given a properly initialized `cs_Mstro_` structure via the **mstro** argument, *CSmstroI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSmstroI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are the first elements in these arrays, the Y coordinate and the latitude are the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSmstroK scale (K)

```
double CSmstroK (Const struct cs_Mstro_ *mstro,Const double ll [2]);
```

CSmstroK returns the grid scale factor, along a parallel, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array.

CSmstroC Convergence angle

```
double CSmstroC (Const struct cs_Mstro_ *mstro,Const double ll [2]);
```

CSmstroC returns the convergence angle, in degrees east of north, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array. Analytical formulas to compute the convergence angle for this projection to not exist; *CS_llazdd* is used to empirically calculate the convergence angle.

CSmstroQ definition Quality check

```
int CSmstroQ (Const struct cs_Csdef_ *csdef,unsgined short prj_code,
              int *err_list [],int list_sz);
```

CSmstroQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Modified Stereographic Projection. *CS_cschk* examines those definition components that are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CSmstroQ* only examines those components specific to the Modified Stereographic Projection. *CSmstroQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSmstroQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSmstroL Latitude/longitude check

```
int CSmstroL (Const struct cs_Mstro_ *mstro,int cnt,Const double pnts
[][3]);
```

CSmstroL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **mstro** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSmstrosL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSmstroL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSmstroX Xy coordinate check

```
int CSmstroX (struct cs_Mstro_ *mstro,int cnt,Const double pnts [][3]);
```

CSmstroX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **mstro** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSmstrosX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSmstroL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSmstroS Setup

```
void CSmstroS (struct cs_Csprm_ *csprm);
```

The *CSmstroS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the central meridian, the origin latitude, and other projection parameters are known, there are many calculations that need only be performed once. *CSmstroS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the single argument provided to *CSmstroS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. The specific elements of the `cs_Csdef_` structure that must be initialized for the Modified Stereographic projection are:

prj_prm1 thru prj_prm24	Use these elements to specify the power series coefficients. Odd and even number pairs, e.g. prj_prm1 and prj_prm2, are used to specify the real and imaginary components of the coefficients respectively. All other prj_prm's that represent unused coefficients must be set to zero. CSmstroS will determine the order of the series by looking for zero coefficients. Thus, orders as high as twelve are supported. Orders higher than twelve are not supported.
org_lng	The longitude, in degrees, of the origin of the projection.
org_lat	The latitude, in degrees, of the origin of the projection.
scl_red	This element must be set to 1.0.
Scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size.
y_off	The false northing to be applied to all Y coordinates.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The values of equatorial radius and eccentricity are extracted from the *datum* element of the *cs_Csprm_* structure. These are normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required elements are:

e_rad	The equatorial radius of the earth in meters.
eccent	This value represents the eccentricity of the ellipsoid.
to84_via	An integer code that specifies the technique that is to be used to convert geographic coordinates based on this datum to WGS84.

cs_Mstro_ Structure

The results of the one-time calculations are recorded in the `mstro` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSmstroF*, *CSmstroI*, *CSmstroK*, and *CSmstroC* functions require as their first argument.

New Zealand National Grid System (CSnzlnd)

This set of functions represent the Coordinate System Mapping Package's knowledge of the New Zealand National Grid System. Since this projection is a conformal projection, the K (grid scale along a parallel) and H (grid scale along a meridian) are the same. Therefore, there is no H function for this projection.

CSnzlndF Forward

```
int CSnzlndF (Const struct cs_Nzlnd_ *nzlnd,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Nzlnd_` structure via the **nzlnd** argument, *CSnzlndF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSnzlndF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSnzlndI Inverse

```
int CSnzlndI (Const struct cs_Nzlnd_ *nzlnd,double ll [2],Const double xy
[2]);
```

Given a properly initialized `cs_Nzlnd_` structure via the **nzlnd** argument, *CSnzlndI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSnzlndI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system.

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are the first elements in these arrays, the Y coordinate and the latitude are the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSnzlndK grid scale (K)

```
double CSnzlndK (Const struct cs_Nzlnd_ *nzlnd,Const double ll [2]);
```

CSnzlndK returns the grid scale factor, along a parallel, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array. While analytical formulas for the grid scale factor for this projection have been located, they have not yet been coded. The grid scale factor returned by *CSnzlndK* is determined empirically using *CS_llazdd*.

CSnzlndC Convergence angle

```
double CSnzlndC (Const struct cs_Nzlnd_ *nzlnd,Const double ll [2]);
```

CSnzlndC returns the convergence angle, in degrees east of north, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array. While analytical formulas for the convergence for this projection have been located, they have not yet been coded. The convergence angle returned by *CSnzlndC* is determined empirically using *CS_llazdd*.

CSnzlndQ definition Quality check

```
int CSnzlndQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CSnzlndQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the New Zealand National Grid System Projection. *CS_cschk* examines those definition components that are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CSnzlndQ* only examines those components specific to the New Zealand National Grid System Projection. *CSnzlndQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSnzlndQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSnzlndL Latitude/longitude check

```
int CSnzlndL (Const struct cs_Nzlnd_ *nzlnd,int cnt,Const double pnts
[][3]);
```

CSnzlndL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **nzlnd** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSnzlndsL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSnzlndL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSnzlndX Xy coordinate check

```
int CSnzlndX (Const struct cs_Nzlnd_ *nzlnd,int cnt,Const double pnts
[][3]);
```

CSnzlndX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **nzlnd** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSnzlndsX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSnzlndL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSnzlndS Setup

```
void CSnzlndS (struct cs_Csprm_ *csprm);
```

The *CSnzlndS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the central meridian, the origin latitude, and other projection parameters are known, there are many calculations that need only be performed once. *CSnzlndS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the single argument provided to *CSnzlndS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. Note that the coefficients of the complex power series upon which this projection is based are hard coded into the *CSnzlndS* module and cannot be changed by the user. The specific elements of the `cs_Csdef_` structure that must be initialized for the New Zealand National Grid System are:

org_lng	The longitude, in degrees, of the origin of the projection.
org_lat	The latitude, in degrees, of the origin of the projection.
scl_red	This value must be set to 1.0.
scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units, and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size.
y_off	The false northing to be applied to all Y coordinates.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The values of equatorial radius and eccentricity are extracted from the *datum* element of the *cs_Csprm_* structure. These are normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required elements are:

e_rad	The equatorial radius of the earth in meters.
eccent	This value represents the eccentricity of the ellipsoid.
to84_via	An integer code that specifies the technique that is to be used to convert geographic coordinates based on this datum to WGS84.

cs_Nzlnd_ Structure

The results of the one-time calculations are recorded in the `nzlnd` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSnzlndF*, *CSnzlndI*, *CSnzlndK*, and *CSnzlndC* functions require as their first argument.

Non-Earth Coordinate System (CSnerth)

This set of functions represent the Coordinate System Mapping Package's knowledge of Non-Earth Coordinate Systems. A non-earth coordinate system is a MapInfo invention (we believe). What this consists of is a coordinate system which CS-MAP does very little with other than apply a translation and, optionally, a units conversion. With the introduction of this coordinate system, the products of

several of our clients can now perform these rudimentary operations on a large variety of different data formats with a minimal amount of extra code. Perhaps more importantly, with a minimal amount of additional user interface. This is probably why MapInfo invented these things.

Thus, this "projection" is not really a projection. It is yet another pseudo projection. In order for this projection to function within the CS-MAP framework, there is a simple but necessary rule:

You can only convert a non-earth coordinate system to another non-earth coordinate system.

This rule enables the "non-earth" business to co-exist with all the other stuff in CS-MAP without major conflict. CS-MAP cannot enforce this explicitly. Your application must enforce this.

To enforce this rule implicity, the intermediary coordinates which are generated by the Forward function and which are accepted by the Inverse function are incredibly small numbers. So small, that doing anything with these numbers other than passing them to another "non-earth" function will produce obviously ridiculous results (i.e. all zeros).

CSnerthF - Forward Conversion

```
int CSnerthF (Const struct cs_Nerth_ *nerth,double xy [2],Const double
inter [2]);
```

Given a properly initialized `cs_Berth_` structure via the **nerth** argument, *CSnerthF* will convert the intermediary coordinates provided in the **inter** array to X and Y coordinates, returning the result in the **xy** array. *CSnerthF* always returns **cs_CNVRT_NRML**.

The **xy** and **inter** arrays may be the same array. The X coordinate is carried in the first element in the **xy** array, the Y coordinate in the second element. The intermediary coordinates are designed to be very small numbers on the order of 1.0E-50 to make sure they are never confused with latitude and longitude.

CSnerthI

```
int CSnerthI (Const struct cs_Millr_ *nerth,double inter [2],Const double
xy [2]);
```

Given a properly initialized `cs_Nerth_` structure via the **nerth** argument, *CSnerthI* will convert the X and Y coordinates given in the **xy** array to intermediary coordinates and return the result in the **inter** array. *CSnerthI* always returns **cs_CNVRT_NRML**.

The **xy** and **inter** arrays may be the same array. The X coordinate is carried in the first element of the **xy** array, the Y coordinate. Intermediary coordinate are designed to be exceedingly small number so that they will never be confused with longitude and latitude.

CSnerthK scale (K) along a parallel

```
double CSnerthK (Const struct cs_Nerth_ *nerth,Const double ll [2]);
```

CSnerthK ignores the **ll** argument and returns 1.0.

CSnerthC Convergence angle

```
double CSnerthC (Const struct cs_Nerth_ *nerth,Const double ll [2]);
```

CSnerthC ignores the **ll** argument and returns zero.

CSnerthQ definition Quality check

```
int CSnerthQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
```

```
int *err_list [],int list_sz);
```

CSnerthQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Non-Earth Coordinates Pseudo Projection. *CS_cschk* examines those definition components that are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CSnerthQ* only examines those components specific to the Non-Earth Coordinates Pseudo Projection. *CSnerthQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSnerthQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

Currently, Non-Earth coordinate systems are required to be cartographically referenced as all coordinate systems must be referenced to something and cartographiclly referencing a Non-Earth system seems like a helpful way to make sure this pseudo projection is not used erroneously. *CSnerthQ* only checks to see that the coordinate system is referenced to an ellipsoid.

CSnerthL

```
int CSnerthL (Const struct cs_Nerth_ *nerth,int cnt,Const double pnts
[][3]);
```

By design, "nerth" coordinates are very small numbers, on the order of 1.0E-50. C*SnerthL* determines if the intermediary coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are less than annother small (but not so small) number. If so, the coordinates are considered to be within the domain of the "nerth" system.

The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSnerthL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSnerthL* returns **cs_CNVRT_OK** if all subject coordinates are within the "domain". **cs_CNVRT_DOMN** is returned if one or more of the subject intermediary coordinates are outside of the "domain" of the coordinate system.

CSnerthX Xy coordinate check

```
int CSnerthX (Const struct cs_Nerth_ *nerth,int cnt,Const double pnts
[][3]);
```

By design, "nerth" coordinates are very small numbers, on the order of 1.0E-50. It is the nature of the small numbers which enables CS-MAP to insure that "nerth" conversions are not mistakenly used by other components of the system. C*SnerthX, therefore,* determines if the coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are sufficiently large so as not to be considered "nerth" intermediary coordinates and less than a large number (1.0E+07 at this writing) which represents a reasonable upper bound on "nerth" coordinates. If the coordinates meet these two criteria, the coordinates are considered to be within the domain of the "nerth" system.

The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSnerthX*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSnerthX* returns **cs_CNVRT_OK** if all subject coordinates are within the "domain". **cs_CNVRT_DOMN** is returned if one or more of the subject cartesian coordinates are outside of the "domain" of the coordinate system.

CSnerthS Setup

```
void CSnerthS (struct cs_Csprm_ *csprm);
```

The *CSnerthS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. For the Non-Earth pseudo projection, there is very little to do. However, the factor which is used to make "nerth" intermediary coordinates very small is hard coded in this function. *CSnerthS* saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the argument provided to *CSmolwdS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. The following parameters are used:

scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size. This is the X coordinate of the coordinate system origin.
y_off	The false northing to be applied to all Y coordinates. This is the Y coordinate of the coordinate system origin.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

While a datum definition reference is required to remain consistent with the remainder of the system (i.e. it is included in the cs_Csprm_ structure), this pseudo projection ignores the contents of it.

cs_Nerth_ Structure

The results of the one-time calculations are recorded in the `nerth` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSnerthF*, *CSnerthI*, *CSnerthK*, and *CSnerthC* functions require as their first argument.

Normal Aspect Authalic Cylindrical Projection (CSnacyl)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Normal Aspect of the Equal Area (Authalic) Cylindrical projection.

CSnacylF Forward conversion

```
int CSnacylF (Const struct cs_Nacyl_ *nacyl,double xy [2],Const double ll
[2]);
```

Given a properly initialized cs_Nacyl_ structure via the **nacyl** argument, *CSnacylF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSnacylF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSnacylI Inverse conversion

```
int CSnacylI (Const struct cs_Nacyl_ *nacyl,double ll [2],Const double xy
[2]);
```

Given a properly initialized cs_Nacyl_ structure via the **nacyl** argument, *CSnacylI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSnacylI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system.

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSnacylK parallel scale (K)

```
double CSnacylK (Const struct cs_Nacyl_ *nacyl,Const double ll [2]);
```

CSnacylK returns the grid scale factor, along a parallel, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array.

CSnacylH meridian scale (H)

```
double CSnacylH (Const struct cs_Nacyl_ *nacyl,Const double ll [2]);
```

CSnacylH returns the grid scale factor, along a meridian, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array.

CSnacylC Convergence angle

```
double CSnacylC (Const struct cs_Nacyl_ *nacyl,Const double ll [2]);
```

CSnacylC returns the value 0.0 which is the convergence angle in degrees east of north of any coordinate system based on this projection at the latitude and longitude provided by the **ll** argument.

CSnacylQ definition Quality check

```
int CSnacylQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CSnacylQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Normal Aspect of the Equal Area Cylindrical Projection. *CS_cschk* examines those definition components which are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CSnacylQ* only examines those components specific to the Normal Aspect of the Equal Area Cylindrical Projection. *CSnacylQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSnacylQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSnacylL Latitude/longitude check

```
int CSnacylL (Const struct cs_Nacyl_ *nacyl,int cnt,Const double pnts
[][3]);
```

CSnacylL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **nacyl** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSnacylsL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSnacylL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSnacylX Xy coordinate check

```
int CSnacylX (Const struct cs_Nacyl_ *nacyl,int cnt,Const double pnts
[][3]);
```

CSnacylX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **nacyl** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSnacylsX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSnacylL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSnacylS Setup

```
void CSnacylS (struct cs_Csprm_ *csprm);
```

The *CSnacylS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the origin longitude, and other projection parameters are known, there are many calculations which need only be performed once. *CSnacylS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the single argument provided to *CSnacylS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. The specific elements of the `cs_Csdef_` structure that must be initialized for the Normal Aspect of the Equidistant Cylindrical Projection are:

org_lng	Longitude, in degrees, of the central meridian of the coordinate system (or map).
prj_prm1	Latitude, in degrees, of the standard parallel, usually zero indicating the equator. Using a non-zero value has an affect similar to that of the scale reduction factor of other cylindrical projections.
scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size.
y_off	The false northing to be applied to all Y coordinates.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The values of equatorial radius and eccentricity are extracted from the *datum* element of the *cs_Csprm_* structure. These are normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required elements are:

e_rad	The equatorial radius of the earth in meters.
eccent	This value represents the eccentricity of the ellipsoid.
to84_via	An integer code that specifies the technique that is to be used to convert geographic coordinates based on this datum to WGS84.

cs_Nacyl_ Structure

The results of the one-time calculations are recorded in the `nacyl` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSnacylF*, *CSnacylI*, *CSnacylK*, and *CSnacylC* functions require as their first argument.

Oblique Cylindrical Projection (CSswiss)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Oblique Cylindrical Projection. Both spherical and ellipsoidal forms are supported. Since this projection is a

conformal projection, the K (grid scale along a parallel) and H (grid scale along a meridian) are the same. Therefore, there is no H function for this projection.

A specialized version of this projection was originally developed. It needed to be generalized in order to support coordinate systems in Hungary. However, the original name, `CSswiss?`, is retained in the code.

CSswissF Forward conversion

```
int CSswissF (Const struct cs_Swiss_ *swiss,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Swiss_` structure via the **swiss** argument, *CSswissF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSswissF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSswissI Inverse conversion

```
int CSswissI (Const struct cs_Swiss_ *swiss,ll,Const double xy [2]);
```

Given a properly initialized `cs_Swiss_` structure via the **swiss** argument, *CSswissI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSswissI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSswissK parallel scale (K)

```
double CSswissK (Const struct cs_Swiss_ *swiss,Const double ll [2]);
```

CSswissK returns the grid scale factor along a parallel at the geodetic location specified by the **ll** argument. Since the Swiss Oblique Cylindrical projection is conformal, the returned value is also valid as the grid scale factor along a meridian.

CSswissC Convergence angle

```
double CSswissC (Const struct cs_Swiss_ *swiss,Const double ll [2]);
```

CSswissC returns the convergence angle in degrees east of north of the geodetic location specified by the **ll** argument. Analytical formulas for this value have not been located and the result is arrived at through the use of the *CS_llazdd* function.

CSswissQ definition Quality check

```
int CSswissQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CSswissQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Swiss Oblique Cylindrical Projection. *CS_cschk* examines those definition components that are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CSswissQ* only examines those components specific to the Swiss Oblique Cylindrical Projection. *CSswissQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The

number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSswissQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSswissL Latitude/longitude check

```
int CSswissL (Const struct cs_Swiss_ *swiss,int cnt,Const double pnts
[][3]);
```

CSswissL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **swiss** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSswisssL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSswissL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSswissX Xy coordinate check

```
int CSswissX (Const struct cs_Swiss_ *swiss,int cnt,Const double pnts
[][3]);
```

CSswissX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **swiss** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSswisssX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSswissL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSswissS Setup

```
void CSswissS (struct cs_Csprm_ *csprm);
```

The *CSswissS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the origin latitude and longitude and other projection parameters are known, there are many calculations which need only be performed once. *CSswissS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the single argument provided to *CSswissS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time.

Two variations of this projection are supported. The first variation described is the more general variation, the second variation being a simple special case of the first. This may be considered strange, but the simple special case was implemented first, and the more general case developed at a later date.

The following elements of the `cs_Csdef_` structure that must be initialized for boith variations the Oblique Cylindrical Projection are as follows:

org_lng	The longitude of the origin of the projection, in degrees, where positive indicates east longitude. This is the longitude of the central point of the projection. This longitude is also considered the X origin of the projection.
org_lat	The latitude of the origin of the projection, in degrees, where positive indicates north latitude. This is the latitude of the central point of the projection. This latitude is also considered the Y origin of the projection.
scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size. This is the X coordinate of the projection origin specified by the org_lng parameter.
y_off	The false northing to be applied to all Y coordinates, usually selected to cause all Y coordinates within the coordinate system to be positive values of reasonable size. This is the Y coordinate of the projection origin specified by the org_lat parameter.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Oblique Cylindrical Projection, Generalized (cs_PRJCOD_OBQCYL)

This variation represents the generalized version of the Oblique Cylindrical projection.

The following elements of the `cs_Csdef_` structure are specific to this variation of the projection:

scl_red	The factor by which the cylindrical projection surface is shrunk into the gaussian sphere before the actual projection process begins.
prj_prm1	The latitude at which the radius of the gaussian sphere is calculated.

Oblique Cylindrical Projection, Switzerland (cs_PRJCOD_SWISS)

This variation represents the specialized version of the Oblique Cylindrical projection that was originally implemented for use in Switzerland. There are no specific parameters required. The generalized scale reduction parameter is hard coded to 1.0 for this variation, and the latitude at which the gaussian sphere is calculated is hard coded to be equal to the origin latitude.

Datum Definition

The values of equatorial radius and eccentricity are extracted from the *datum* element of the *cs_Csprm_* structure. These are normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required elements are:

e_rad	The equatorial radius of the earth in meters.
eccent	This value represents the eccentricity of the ellipsoid.
to84_via	An integer code that specifies the technique that is to be used to convert geographic coordinates based on this datum to WGS84.

cs_Swiss_ Structure

The results of the one-time calculations are recorded in the `swiss` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSswissF*, *CSswissI*, *CSswissK*, and *CSswissC* functions require as their first argument.

Oblique Stereographic Projection (CSostro)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Oblique Stereographic Projection. Several other forms of the St6ereographic projection are also supported. The algorithms implemented in this projection are those commonly in use. They are different from those presented by Snyder in Map Projections - A Working Manual.

CSostroF Forward conversion

```
int CSostroF (Const struct cs_Ostro_ *ostro,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Ostro_` structure via the **ostro** argument, *CSostroF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSostroF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

The **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSostroI Inverse conversion

```
int CSostroI (Const struct cs_Ostro_ *ostro,double ll [2],Const double xy
[2]);
```

Given a properly initialized `cs_Ostro_` structure via the **ostro** argument, *CSostroI* will convert the X

and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSostroI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

The **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSostroK grid scale (K) normal to radial

```
double CSostroK (Const struct cs_Ostro_ *ostro,Const double ll [2]);
```

CSostroK returns the grid scale factor normal to the radial at the geodetic location specified by the **ll** argument. As the Oblique Stereographic projection is conformal, the H scale factor is the same as the K scale factor. Therefore, there is no *CSostroH* function.

CSostroC Convergence angle

```
double CSostroC (Const struct cs_Ostro_ *ostro,Const double ll [2]);
```

CSostroC returns the convergence angle in degrees east of north of the geodetic location specified by the **ll** argument. Analytical formulas for this value have not been located and the result is arrived at through the use of the *CS_llazdd* function.

CSostroQ definition Quality check

```
int CSostroQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CSostroQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Stereographic Projection. *CS_cschk* examines those definition components that are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CSostroQ* only examines those components specific to the Oblique Stereographic Projection. *CSostroQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSostroQ* may be called with the NULL pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSostroL Latitude/longitude check

```
int CSostroL (Const struct cs_Ostro_ *ostro,int cnt,Const double pnts
[][3]);
```

CSostroL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **ostro** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSostrosL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSostroL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSostroX Xy coordinate check

```
int CSostroX (Const struct cs_Ostro_ *ostro,int cnt,Const double pnts
[][3]);
```

CSostroX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **ostro** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSostrosX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSostroL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSostroS Setup

```
void CSostroS (struct cs_Csprm_ *csprm);
```

The *CSostroS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the origin latitude and longitude, scale reduction, and other projection parameters are known, there are many calculations which need only be performed once. *CSostroS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the first argument provided to *CSostroS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time.

The following parameters must be set:

org_lng	The longitude, in degrees, of the origin of the projection.
org_lat	The latitude, in degrees, of the origin of the projection.
Scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units, the scale reduction factor, and the mapping scale that is to be applied.
scl_red	The scale reduction factor, independent of all other scaling, is obtained from this element and is necessary for correct computation of the grid scale factor in some cases.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size. This is the X coordinate of the coordinate system origin.
y_off	The false northing to be applied to all Y coordinates. This is the Y coordinate of the coordinate system origin.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The values of equatorial radius and eccentricity are extracted from the *datum* element of the *cs_Csprm_* structure. These are normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required elements are:

e_rad	The equatorial radius of the earth in meters.
eccent	This value represents the eccentricity of the ellipsoid.
to84_via	An integer code that specifies the technique that is to be used to convert geographic coordinates based on this datum to WGS84.

cs_Ostro_ Structure

The results of the one-time calculations are recorded in the `ostro` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSostroF*, *CSostroI*, *CSostroK*, and *CSostroC* functions require as their first argument.

Orthographic Projection (CSortho)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Orthographic Projection. This projection is supported in spherical form only. The equatorial radius of the supplied ellipsoid is used as the radius of the sphere.

CSorthoF Forward conversion

```
int CSorthoF (Const struct cs_Ortho_ *ortho,double xy [2],Const double ll
[2]);
```

Given a properly initialized *cs_Ortho_* structure via the **ortho** argument, *CSorthoF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSorthoF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSorthoI Inverse conversion

```
int CSorthoI (Const struct cs_Ortho_ *ortho,double ll [2],Const double xy
[2]);
```

Given a properly initialized *cs_Ortho_* structure via the **ortho** argument, *CSorthoI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSorthoI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSorthoK grid scale (K) normal to radial

```
double CSorthoK (Const struct cs_Ortho_ *ortho,Const double ll [2]);
```

CSorthoK returns the value 1.0 which is the grid scale factor normal to the radial at the geodetic location specified by the **ll** argument.

CSorthoH grid scale (H) along radial

```
double CSorthoH (Const struct cs_Ortho_ *ortho,Const double ll [2]);
```

CSorthoH returns the grid scale factor along a radial from the coordinate system origin to (and at) the geodetic location specified by the **ll** argument.

CSorthoC Convergence angle

```
double CSorthoC (Const struct cs_Ortho_ *ortho,Const double ll [2]);
```

CSorthoC returns the convergence angle in degrees east of north of the geodetic location specified by the **ll** argument. Analytical formulas for this value have not been located and the result is arrived at using the *CS_azsphr* function.

CSorthoQ definition Quality check

```
int CSorthoQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CSorthoQ determines if the coordinate system definition provided by the **csdef** argument is consistent

with the requirements of the Orthographic Projection. *CS_cschk* examines those definition components which are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CSorthoQ* only examines those components specific to the Orthographic Projection. *CSorthoQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSorthoQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSorthoL Latitude/longitude check

```
int CSorthoL (Const struct cs_Ortho_ *ortho,int cnt,Const double pnts
[][3]);
```

CSorthoL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **ortho** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSorthosL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSorthoL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSorthoX Xy coordinate check

```
int CSorthoX (Const struct cs_Ortho_ *ortho,int cnt,Const double pnts
[][3]);
```

CSorthoX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **ortho** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSorthosX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSorthoL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSorthoS Setup (general)

```
void CSorthoS (struct cs_Csprm_ *csprm);
```

The *CSorthoS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the origin latitude and longitude, scale reduction, and other projection parameters are known, there are many calculations which need only be performed once. *CSorthoS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the argument provided to *CSorthoS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. The following parameters are used:

org_lng	The longitude, in degrees, of the origin of the projection.
org_lat	The latitude, in degrees, of the origin of the projection.
Scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size. This is the X coordinate of the coordinate system origin.
y_off	The false northing to be applied to all Y coordinates. This is the Y coordinate of the coordinate system origin.
Quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The value of equatorial radius is extracted from the *datum* element of the `cs_Csprm_` structure and used as the radius of the sphere. This is normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required element is:

e_rad	The radius of the earth, as a sphere, in meters.

cs_Ortho_ Structure

The results of the one-time calculations are recorded in the `ortho` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSorthoF*, *CSorthoI*, *CSorthoK*, *CSorthoH,* and *CSorthoC* functions require as their first argument.

Polar Stereographic Projection (CSpstro)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Polar Stereographic Projection. There exist several other forms of the Stereographic projection.

CSpstroF Forward conversion

```
int CSpstroF (Const struct cs_Pstro_ *pstro,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Pstro_` structure via the **pstro** argument, *CSpstroF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSpstroF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate

system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSpstrol Inverse conversion

```
int CSpstroI (Const struct cs_Pstro_ *pstro,double ll [2],Const double xy
[2]);
```

Given a properly initialized `cs_Pstro_` structure via the **pstro** argument, **CSpstrol** will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSpstrol* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSpstroK grid scale (K) normal to radial

```
double CSpstroK (Const struct cs_Pstro_ *pstro,Const double ll [2]);
```

CSpstroK returns the grid scale factor normal to the radial at the geodetic location specified by the **ll** argument. As the Polar Stereographic projection is conformal, the H scale factor is the same as the K scale factor. Therefore, there is no *CSpstroH* function.

CSpstroC Convergence angle

```
double CSpstroC (Const struct cs_Pstro_ *pstro,Const double ll [2]);
```

CSpstroC returns the convergence angle in degrees east of north of the geodetic location specified by the **ll** argument. Analytical formulas for this value have not been located and the result is arrived at through the use of the *CS_llazdd* function.

CSpstroQ definition Quality check

```
int CSpstroQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CSpstroQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Polar Stereographic Projection. *CS_cschk* examines those definition components which are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CSpstroQ* only examines those components specific to the Polar Stereographic Projection. *CSpstroQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSpstroQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSpstroL Latitude/longitude check

```
int CSpstroL (Const struct cs_Pstro_ *pstro,int cnt,Const double pnts
[][3]);
```

CSpstroL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **pstro** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSpstrosL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSpstroL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical

domain the coordinate system. *cs_CNVRT_DOMN* is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSpstroX Xy coordinate check

```
int CSpstroX (Const struct cs_Pstro_ *pstro,int cnt,Const double pnts
[][3]);
```

CSpstroX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **pstro** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSpstrosX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSpstroL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSpstroS Setup

```
void CSpstroS (struct cs_Csprm_ *csprm);
```

The *CSpstroS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the origin latitude and longitude, scale reduction, and other projection parameters are known, there are many calculations which need only be performed once. *CSpstroS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the first argument provided to *CSpstroS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time.

The following parameters must be set:

org_lng	The longitude, in degrees, of the origin of the projection.
org_lat	The latitude, in degrees, of the origin of the projection. Must be either 90 degrees north (i.e. positive), or 90 degrees south (negative).
Scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units, the scale reduction factor, and the mapping scale that is to be applied.
scl_red	The scale reduction factor, independent of all other scaling, is obtained from this element and is necessary for correct computation of the grid scale factor in some cases.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size. This is the X coordinate of the coordinate system origin.
y_off	The false northing to be applied to all Y coordinates. This is the Y coordinate of the coordinate system origin.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The values of equatorial radius and eccentricity are extracted from the *datum* element of the *cs_Csprm_* structure. These are normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required elements are:

e_rad	The equatorial radius of the earth in meters.
eccent	This value represents the eccentricity of the ellipsoid.
to84_via	An integer code that specifies the technique that is to be used to convert geographic coordinates based on this datum to WGS84.

cs_Pstro_ Structure

The results of the one-time calculations are recorded in the `pstro` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSpstroF*, *CSpstroI*,

CSpstroK, and *CSpstroC* functions require as their first argument.

Robinson Projection (CSrobin)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Robinson Projection. This projection is supported in spherical form only. The equatorial radius of the supplied ellipsoid is used as the radius of the sphere.

CSrobinF Forward conversion

```
int CSrobinF (Const struct cs_Robin_ *robin,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Robin_` structure via the **robin** argument, *CSrobinF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSrobinF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSrobinI Inverse conversion

```
int CSrobinI (Const struct cs_Robin_ *robin,double ll [2],Const double xy
[2]);
```

Given a properly initialized `cs_Robin_` structure via the **robin** argument, *CSrobinI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSrobinI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSrobinK grid scale (K) normal to radial

```
double CSrobinK (Const struct cs_Robin_ *robin,Const double ll [2]);
```

CSrobinK returns the grid scale factor along a parallel of any coordinate system based on this projection at any location. Analytical formulas for this value have not been located and the result is arrived at empirically the use of spherical trigonometry.

CSrobinH grid scale (H) along radial

```
double CSrobinH (Const struct cs_Robin_ *robin,Const double ll [2]);
```

CSrobinH returns the grid scale factor along a meridian at the geodetic location specified by the **ll** argument. Analytical formulas for this value have not been located and the result is arrived at empirically the use of spherical trigonometry.

CSrobinC Convergence angle

```
double CSrobinC (Const struct cs_Robin_ *robin,Const double ll [2]);
```

CSrobinC returns the convergence angle in degrees east of north of the geodetic location specified by the **ll** argument. Analytical formulas for this value have not been located and the result is arrived at through the use of the *CS_azsphr* function.

CSrobinQ definition Quality check

```
int CSrobinQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
```

```
int *err_list [],int list_sz);
```

CSrobinQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Robinson Projection. *CS_cschk* examines those definition components that are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CSrobinQ* only examines those components specific to the Robinson Projection. *CSrobinQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSrobinQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSrobinL Latitude/longitude check

```
int CSrobinL (Const struct cs_Robin_ *robin,int cnt,Const double pnts
[][3]);
```

CSrobinL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **robin** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSrobinsL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSrobinL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates is outside of the mathematical domain of the coordinate system.

CSrobinX Xy coordinate check

```
int CSrobinX (Const struct cs_Robin_ *robin,int cnt,Const double pnts
[][3]);
```

CSrobinX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **robin** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSrobinsX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSrobinL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. *cs_CNVRT_DOMN* is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSrobinS Setup (general)

```
void CSrobinS (struct cs_Csprm_ *csprm);
```

The *CSrobinS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the origin longitude and other projection parameters are known, there are many calculations that need only be performed once. *CSrobinS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the argument provided to *CSrobinS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. The following parameters are used:

org_lng	The longitude, in degrees, of the origin of the projection.
scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size. This is the X coordinate of the coordinate system origin.
y_off	The false northing to be applied to all Y coordinates. This is the Y coordinate of the coordinate system origin.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The value of equatorial radius is extracted from the *datum* element of the `cs_Csprm_` structure and used as the radius of the sphere. This is normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required element is:

e_rad	The radius of the earth, as a sphere, in meters.

cs_Robin_ Structure

The results of the one-time calculations are recorded in the `robin` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSrobinF*, *CSrobinI*, *CSrobinK*, *CSrobinH,* and *CSrobinC* functions require as their first argument.

Sinusoidal Projection (CSsinus)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Sinusoidal Projection. Both spherical and ellipsoidal forms are supported.

CSsinusF Forward conversion

```
int CSsinusF (Const struct cs_Sinus_ *sinus,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Sinus_` structure via the **sinus** argument, *CSsinusF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSsinusF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSsinusI Inverse conversion

```
int CSsinusI (Const struct cs_Sinus_ *sinus,ll,Const double xy [2]);
```

Given a properly initialized cs_Sinus_ structure via the **sinus** argument, *CSsinusI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSsinusI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSsinusK parallel scale (K)

```
double CSsinusK (Const struct cs_Sinus_ *sinus,Const double ll [2]);
```

CSsinusK returns the grid scale factor along a parallel at the geodetic location specified by the **ll** argument. For the Sinusoidal Projection, this value is always 1.0.

CSsinusH meridian scale (H)

```
double CSsinusH (Const struct cs_Sinus_ *sinus,Const double ll [2]);
```

CSsinusH returns the grid scale factor along a meridian at the geodetic location specified by the **ll** argument. For the sphere, specific formulas are used to compute this value. For the ellipsoid, specific formulas could not be located and the result is arrived at using the *CS_llazdd* function.

CSsinusC Convergence angle

```
double CSsinusC (Const struct cs_Sinus_ *sinus,Const double ll [2]);
```

CSsinusC returns the convergence angle in degrees east of north of the geodetic location specified by the **ll** argument. Analytical formulas for this value have not been located and the result is arrived at through the use of the *CS_llazdd* function.

CSsinusQ definition Quality check

```
int CSsinusQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CSsinusQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Sinusoidal Projection. *CS_cschk* examines those definition components that are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CSsinusQ* only examines those components specific to the Sinusoidal Projection. *CSsinusQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSsinusQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSsinusL Latitude/longitude check

```
int CSsinusL (Const struct cs_Sinus_ *sinus,int cnt,Const double pnts
[][3]);
```

CSsinusL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the

coordinate system provided by the **sinus** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSsinussL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSsinusL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSsinusX Xy coordinate check

```
int CSsinusX (Const struct cs_Sinus_ *sinus,int cnt,Const double pnts
[][3]);
```

CSsinusX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **sinus** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSsinussX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSsinusL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSsinusS Setup

```
void CSsinusS (struct cs_Csprm_ *csprm);
```

The *CSsinusS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the central meridian and other projection parameters are known, there are many calculations which need only be performed once. *CSsinusS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the single argument provided to *CSsinusS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time.

The specific elements of the `cs_Csdef_` structure that must be initialized for the Sinusoidal Projection are as follows:

org_lng	The longitude of the central meridian of the projection, in degrees, where positive indicates east longitude. It is this line of longitude that is straight and true to scale. This longitude is also considered the X origin of the projection. The Y origin of the projection is always the equator.
prj_prm1-24	These 24 doubles can be used in groups of three to specify the zones of an interrupted Sinusoidal projection. See CS_zones(4CS) for more information on how to encode a specific zone or zones. Leave all values set to zero for a standard single zone projection based on the origin longitude specified in the org_lng element.
scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units, the scale reduction factor, and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size. This is the X coordinate of the coordinate system origin.
y_off	The false northing to be applied to all Y coordinates. This is the Y coordinate of the coordinate system origin.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The values of equatorial radius and eccentricity are extracted from the *datum* element of the *cs_Csprm_* structure. These are normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required elements are:

e_rad	The equatorial radius of the earth in meters.
eccent	This value represents the eccentricity of the ellipsoid.
to84_via	An integer code that specifies the technique that is to be used to convert geographic coordinates based on this datum to WGS84.

cs_Sinus_ Structure

The results of the one-time calculations are recorded in the `sinus` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSsinusF*, *CSsinusI*, *CSsinusK*, *CSsinusH*, and *CSsinusC* functions require as their first argument. Note that the `cs_Sinus_` structure includes an array of eight `cs_Zone_` structures in order to support an interrupted sinusoidal projection with up to eight zones.

Oblique Stereographic ala Snyder (CSsstro)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Oblique Stereographic Projection. Several other forms of the Stereographic projection are also supported. The algorithms implemented in this specific projection are those presented by John Parr Snyder in Map Projections - A Working Manual. These fomulations are not widely used, and certainly not those used in the maritime provinces of Canada, the Netherlands, amd Romania.

CSsstroF Forward conversion

```
int CSsstroF (Const struct cs_Sstro_ *sstro,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Sstro_` structure via the **sstro** argument, *CSsstroF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSsstroF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSsstroI Inverse conversion

```
int CSsstroI (Const struct cs_Sstro_ *sstro,double ll [2],Const double xy
[2]);
```

Given a properly initialized `cs_Sstro_` structure via the **sstro** argument, *CSsstroI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSsstroI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSsstroK grid scale (K) normal to radial

```
double CSsstroK (Const struct cs_Sstro_ *sstro,Const double ll [2]);
```

CSsstroK returns the grid scale factor normal to the radial at the geodetic location specified by the **ll** argument. As Snyder's development of the Oblique Stereographic projection is conformal, the H scale factor is the same as the K scale factor. Therefore, there is no *CSsstroH* function.

CSsstroC Convergence angle

```
double CSsstroC (Const struct cs_Sstro_ *sstro,Const double ll [2]);
```

CSsstroC returns the convergence angle in degrees east of north of the geodetic location specified by the **ll** argument. Analytical formulas for this value have not been located and the result is arrived at using the *CS_llazdd* function.

CSsstroQ definition Quality check

```
int CSsstroQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CSssteroQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Stereographic Projection. *CS_cschk* examines those definition components that are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CSsstroQ* only examines those components specific to Snyder's development of the Oblique Stereographic Projection. *CSsstroQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSsstroQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSsstroL Latitude/longitude check

```
int CSsstroL (Const struct cs_Sstro_ *sstro,int cnt,Const double pnts
[][3]);
```

CSssteroL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **sstro** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSssterosL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSssteroL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSsstroX Xy coordinate check

```
int CSsstroX (Const struct cs_Sstro_ *sstro,int cnt,Const double pnts
[][3]);
```

CSssteroX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **sstro** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSssterosX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSssteroL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSsstroS Setup

```
void CSsstroS (struct cs_Csprm_ *csprm);
```

The *CSssteroS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the origin latitude and longitude, scale reduction, and other projection parameters are known, there are many calculations which need only be performed once. *CSssteroS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the first argument provided to *CSssteroS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function;

but can be provided by the application at run time.

The following parameters must be set:

prj_prm1	The azimuth, in degrees east of north, of the positive Y-axis of the coordinate system.
org_lng	The longitude, in degrees, of the origin of the projection.
org_lat	The latitude, in degrees, of the origin of the projection.
scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units, the scale reduction factor, and the mapping scale that is to be applied.
scl_red	The scale reduction factor, independent of all other scaling, is obtained from this element and is necessary for correct computation of the grid scale factor in some cases.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size. This is the X coordinate of the coordinate system origin.
y_off	The false northing to be applied to all Y coordinates. This is the Y coordinate of the coordinate system origin.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The values of equatorial radius and eccentricity are extracted from the *datum* element of the *cs_Csprm_* structure. These are normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required elements are:

e_rad	The equatorial radius of the earth in meters.
eccent	This value represents the eccentricity of the ellipsoid.
to84_via	An integer code that specifies the technique that is to be used to convert geographic coordinates based on this datum to WGS84.

cs_Sstro_ Structure

The results of the one-time calculations are recorded in the `sstro` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSssteroF*, *CSssteroI*, *CSssteroK*, and *CSssteroC* functions require as their first argument.

Transverse Authalic Cylindrical Projection (CStacyl)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Transverse Aspect of the Equal Area (Authalic) Cylindrical projection.

CStacylF Forward conversion

```
int CStacylF (Const struct cs_Tacyl_ *tacyl,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Tacyl_` structure via the **tacyl** argument, *CStacylF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CStacylF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CStacylI Inverse conversion

```
int CStacylI (Const struct cs_Tacyl_ *tacyl,double ll [2],Const double xy
[2]);
```

Given a properly initialized `cs_Tacyl_` structure via the **tacyl** argument, *CStacylI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CStacylI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CStacylK parallel scale (K)

```
double CStacylK (Const struct cs_Tacyl_ *tacyl,Const double ll [2]);
```

CStacylK returns the grid scale factor, along a parallel, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array. As analytical formulas for this value had not been located as yet, this value is arrived at empirically using *CS_llazdd*.

CStacylH meridian scale (K)

```
double CStacylH (Const struct cs_Tacyl_ *tacyl,Const double ll [2]);
```

CStacylH returns the grid scale factor, along a meridian, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array. As analytical formulas for this value had not been located as yet, this value is arrived at empirically using *CS_llazdd*.

CStacylC Convergence angle

```
double CStacylC (Const struct cs_Tacyl_ *tacyl,Const double ll [2]);
```

CStacylC returns the convergence angle in degrees east of north of any coordinate system based on this projection at the latitude and longitude provided by the **ll** argument. As analytical formulas for this

value had not yet been located, this value is arrived at empirically using *CS_llazdd*.

CStacylQ definition Quality check

```
int CStacylQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CStacylQ determines if the coordinate system definition provided by the `csdef` argument is consistent with the requirements of the Transverse Aspect of the Equal Area Cylindrical Projection. *CS_cschk* examines those definition components that are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CStacylQ* only examines those components specific to the Transverse Aspect of the Equal Area Cylindrical Projection. *CStacylQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CStacylQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CStacylL Latitude/longitude check

```
int CStacylL (Const struct cs_Tacyl_ *tacyl,int cnt,Const double pnts
[][3]);
```

CStacylL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **tacyl** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CStacylsL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CStacylL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CStacylX Xy coordinate check

```
int CStacylX (Const struct cs_Tacyl_ *tacyl,int cnt,Const double pnts
[][3]);
```

CStacylX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **tacyl** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CStacylsX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CStacylL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CStacylS Setup

```
void CStacylS (struct cs_Csprm_ *csprm);
```

The *CStacylS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the central meridian and other projection parameters are known, there are many calculations that need only be performed once. *CStacylS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the single argument provided to *CStacylS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_`

structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. The specific elements of the `cs_Csdef_` structure that must be initialized for the Transverse Aspect of the Equal Area Cylindrical Projection are:

org_lng	Longitude, in degrees, of the central meridian of the coordinate system (or map).
org_lat	Latitude, in degrees, of the origin of the projection.
scl_red	The scale reduction to be applied. This is also referred to as the scale of the central meridian.
scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size.
y_off	The false northing to be applied to all Y coordinates.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The values of equatorial radius and eccentricity are extracted from the *datum* element of the *cs_Csprm_* structure. These are normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required elements are:

e_rad	The equatorial radius of the earth in meters.
eccent	This value represents the eccentricity of the ellipsoid.
to84_via	An integer code that specifies the technique that is to be used to convert geographic coordinates based on this datum to WGS84.

cs_Tacyl_ Structure

The results of the one-time calculations are recorded in the `tacyl` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CStacylF*, *CStacylI*, *CStacylK*, *CStacylH*, and *CStacylC* functions require as their first argument.

Transverse Mercator, ala Snyder, Projection (CStrmrs)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Transverse Mercator Projection as formulated by John P. Snyder and published in Map Projections - A Working Manual. Since the Transverse Mercator projection is a conformal projection, the K (grid scale along a parallel) and H (grid scale along a meridian) are the same. Therefore, there is no H function for this projection. The standard Transverse Mercator provided elsewhere is a superior formulation, but certain clients are comforted by obtaining the exact results they are used to. Thus, this projection is provided largely for historical purposes.

CStrmrsF Forward

```
int CStrmrsF (Const struct cs_Trmrs_ *trmrs,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Trmrs_` structure via the **trmrs** argument, **CStrmrsF** will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CStrmrsF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CStrmrsI Inverse

```
int CStrmrsI (Const struct cs_Trmrs_ *trmrs,double ll [2],Const double xy
[2]);
```

Given a properly initialized `cs_Trmrs_` structure via the **trmrs** argument, *CStrmrsI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CStrmrsI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are the first elements in these arrays, the Y coordinate and the latitude are the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CStrmrsK parallel scale (K)

```
double CStrmrsK (Const struct cs_Trmrs_ *trmrs,Const double ll [2]);
```

CStrmrsK returns the grid scale factor of the coordinate system at the specific geographic location defined by the latitude and longitude provided in the **ll** array.

CStrmrsC Convergence angle

```
double CStrmrsC (Const struct cs_Trmrs_ *trmrs,Const double ll [2]);
```

CStrmrsC returns the convergence angle, in degrees east of north, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array.

CStrmrsQ definition Quality check

```
int CStrmrsQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CStrmrsQ determines if the coordinate system definition provided by the `csdef` argument is consistent with the requirements of the Transverse Mercator Projection. *CS_cschk* examines those definition components that are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CStrmrsQ* only examines those components specific to the Transverse Mercator

Projection. *CStrmrsQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to **CSerpt** for a description of the various error codes and their meaning. *CStrmrsQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CStrmrsL Latitude/longitude check

```
int CStrmrsL (Const struct cs_Trmrs_ *trmrs,int cnt,Const double pnts
[][3]);
```

CStrmrsL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **trmrs** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CStrmrssL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CStrmrsL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CStrmrsX Xy coordinate check

```
int CStrmrsX (Const struct cs_Trmrs_ *trmrs,int cnt,Const double pnts
[][3]);
```

CStrmrsX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **trmrs** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CStrmrssX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CStrmrsL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CStrmrsS Snyder, Setup

```
void CStrmrsS (struct cs_Csprm_ *csprm);
```

The *CStrmrsS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the central meridian, the origin latitude, and other projection parameters are known, there are many calculations that need only be performed once. *CStrmrsS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the single argument provided to *CStrmrsS* serves as the source for input and the repository for the results as described below. The `prj_code` element of this structure is ignored.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time.

The following parameters must be set:

prj_prm1	Longitude, in degrees, of the central meridian.
org_lat	The latitude, in degrees, of the origin of the projection.
scl_red	The scale reduction to be applied. This is also referred to as the scale of the central meridian.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size.
y_off	The false northing to be applied to all Y coordinates.
scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The values of equatorial radius and eccentricity are extracted from the *datum* element of the *cs_Csprm_* structure. These are normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required elements are:

e_rad	The equatorial radius of the earth in meters.
eccent	This value represents the eccentricity of the ellipsoid.
to84_via	An integer code that specifies the technique that is to be used to convert geographic coordinates based on this datum to WGS84.

cs_Trmrs_ Structure

The results of the one-time calculations are recorded in the `trmrs` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CStrmrsF*, *CStrmrsI*, *CStrmrsK*, and *CStrmrsC* functions require as their first argument.

Transverse Mercator Projection (CStrmer)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Transverse Mercator Projection. The algorithms used have also been referred to as the Gauss-Kruger projection. Since the Transverse Mercator projection is a conformal projection, the K (grid scale along

a parallel) and H (grid scale along a meridian) are the same. Therefore, there is no H function for this projection. Five variations of this projection are supported. The `prj_code` element of the `cs_Csprm_` structure defines which of the eight variations is to be developed.

CStrmerF Forward

```
int CStrmerF (Const struct cs_Trmer_ *trmer,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Trmer_` structure via the **trmer** argument, *CStrmerI* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CStrmerF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CStrmerI Inverse

```
int CStrmerI (Const struct cs_Trmer_ *trmer,double ll [2],Const double xy
[2]);
```

Given a properly initialized `cs_Trmer_` structure via the **trmer** argument, *CStrmerI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CStrmerI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are the first elements in these arrays, the Y coordinate and the latitude are the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CStrmerK parallel scale (K)

```
double CStrmerK (Const struct cs_Trmer_ *trmer,Const double ll [2]);
```

CStrmerK returns the grid scale factor of the coordinate system at the specific geographic location defined by the latitude and longitude provided in the **ll** array.

CStrmerC Convergence angle

```
double CStrmerC (Const struct cs_Trmer_ *trmer,Const double ll [2]);
```

CStrmerC returns the convergence angle, in degrees east of north, of the coordinate system at the specific geodetic location defined by the latitude and longitude provided in the **ll** array.

CStrmerQ definition Quality check

```
int CStrmerQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CStrmerQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Transverse Mercator Projection. *CS_cschk* examines those definition components that are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CStrmerQ* only examines those components specific to the Transverse Mercator Projection. *CStrmerQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CStrmerQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CStrmerL Latitude/longitude check

```
int CStrmerL (Const struct cs_Trmer_ *trmer,int cnt,Const double pnts
[][3]);
```

CStrmerL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **trmer** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CStrmersL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CStrmerL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CStrmerX Xy coordinate check

```
int CStrmerX (Const struct cs_Trmer_ *trmer,int cnt,Const double pnts
[][3]);
```

CStrmerX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **trmer** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CStrmersX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CStrmerL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CStrmerS Setup

```
void CStrmerS (struct cs_Csprm_ *csprm);
```

The *CStrmerS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the central meridian, the origin latitude, and other projection parameters are known, there are many calculations that need only be performed once. *CStrmerS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the single argument provided to *CStrmerS* serves as the source for input and the repository for the results as described below. The `prj_code` element of this structure defines which of the eight variations is to be setup.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. The use of elements in the `cs_Csdef_` structure is dependent on the specific variation.

The following elements of the `cs_Csdef_` structure are used for all eight variations:

scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Transverse Mercator, aka Gauss Kruger (cs_PRJCOD_TRMER)

This variation is the standard development of the Transverse Mercator projection as used throughout the world.

prj_prm1	Longitude, in degrees, of the central meridian.
org_lat	The latitude, in degrees, of the origin of the projection.
scl_red	The scale reduction to be applied. This is also referred to as the scale of the central meridian.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size.
y_off	The false northing to be applied to all Y coordinates.

Universal Transverse Mercator (cs_PRJCOD_UTM)

This variation implements the same algorithms as the standard Transverse Mercator described above. However, the definition parameters are expressly tailored for the definition of zones of the Universal Transverse Mercator system of coordinate systems.

prj_prm1	UTM zone number. Note this must be an integer value between 1 and 60, even though it is carried in a variable of the double type.
prj_prm2	The hemisphere of the zone. Use a positive one for northern hemisphere, a negative one for the southern hemisphere.

South Oriented Transverse Mercator (cs_PRJCOD_SOTRM)

This variation of the standard development of the Transverse Mercator projection produces results required by, for example, coordinate systems used in South Africa.

prj_prm1	Longitude, in degrees, of the central meridian.
org_lat	The latitude, in degrees, of the origin of the projection.
scl_red	The scale reduction to be applied. This is also referred to as the scale of the central meridian.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size.
y_off	The false northing to be applied to all Y coordinates.

Wisconsin Variation (cs_PRJCOD_WCCST)

This is a minor variation of the traditional version of the projection. This variation supports the Wisconsin County Coordinate System group of coordinate systems. This variation uses a parallel ellipsoid technique to adjust horizontal coordinates for average elevation of the region being mapped.

prj_prm1	Longitude, in degrees, of the central meridian.
prj_prm2	Average geoid separation, in meters, of the region being mapped.
prj_prm3	Average elevation above the geoid (i.e. orthometric height), in system units, of the region being mapped.
org_lat	The latitude, in degrees, of the origin of the projection.
scl_red	The scale reduction to be applied. This is also referred to as the scale of the central meridian.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size.
y_off	The false northing to be applied to all Y coordinates.

Minnesota Variation (cs_PRJCOD_MNDOT):

This is a minor variation of the traditional version of the projection. This variation supports the county coordinate systems developed by the Minnesota Department of Transportation. This variation uses a parallel ellipsoid technique (different from that used in Wisconsin, or course) to adjust horizontal coordinates for average elevation of the region being mapped.

It should be noted that the original MNDOT implementation of this coordinate system applied the scale reduction factor in a non-standard way. Thus, using the standard Tranverse Mercator algorithms, even with the elevated ellipsoid, failed to produce the same coordinate values as the MNDOT

implementation. To overcome this issue, many vendors jiggle the false orign of the coordinate systems for each county, and theref ore produce a close approximation. The CS-MAP implementation uses the exact same algorithm as the MNDOT implementation, and thus can use the published flase origin values and produce the precise values as the MNDOT version.

As a result, the definitions in the CS-MAP dictionary will vary from those in the dictionaries of other vendors. However, the results produced by CS-MAP, using the official published definitions, match the MNDOT implementation at the millimeter level.

prj_prm1	Longitude, in degrees, of the central meridian.
prj_prm2	Average elevation above the ellipsoid, in system units, of the region being mapped.
org_lat	The latitude, in degrees, of the origin of the projection.
scl_red	The scale reduction to be applied. This is also referred to as the scale of the central meridian.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size.
y_off	The false northing to be applied to all Y coordinates.

Transverse Mercator (Gauss/Kruger) with Affine Post Process (cs_PRJCOD_TRMERAF)

This is variation of the traditional Transverse Mercatoor has been implemented to support the Swedish Land Survey system. It essentially adds an Affine Transformation post-process to the projection. The affine post-process is applied as the last item in the calculation, even after the application of the false origin. The form of the affine transformation is as follows:

```
newX = A0 + oldX * A1 + oldY * A2
newY = B0 + oldX * B1 + oldY * B2
```

The following parameters are used by this variation (in addition tot he standard ones for this projection):

prj_prm1	Longitude, in degrees, of the central meridian.
prj_prm2	Coefficient A0
prj_prm3	Coefficient B0
prj_prm4	Coefficient A1
prj_prm5	Coefficient A2
prj_prm6	Coefficient B1
prj_prm7	Coefficient B2
org_lat	The latitude, in degrees, of the origin of the projection.
scl_red	The scale reduction to be applied. This is also referred to as the scale of the central meridian.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size.
y_off	The false northing to be applied to all Y coordinates.

Ordnance Survey National Grid Transformation of 1997 (cs_PRJCOD_OSTN97)

There are two elements which make this variation differ4ent from the standard Transverse Mercator (aka Gauss Kruger) projection. First, the basic projection parameters are hard coded to be those of the standard Ordnance Survey National Grid coordinate system. Thus, this variation really doesn't have any parameters at all.

Second, after the basic projection calculation is performed, the transformation known as Ordnance Survey National Transformation of 1997 (OSTN97) is applied. This transformation requires access to a data file which is expected to be named "OSTN97.txt" and is expected to reside in the primary data directory.

The end result of this variation is the ability to convert ETRF89 based geographic coordinates to National Grid coordinates, where the actual National Grid coordinates are the same (for a specific geographic point) as the 1936 National Grid coordinates.

Yes, this is strange as the datum shift calculation is actually performed on the cartesian coordinates which result from the application of the projection. That is why this datum shift calculation had to be performed here in the cartographic code.

As described above, this variation does not require any parameters. (In fact, what is considered a standard parameter for the Transverse Mercator projection, the Scale Reduction, is also ignored by this variation. The OSGB value of 0.9996012717 is hard coded in.)

Ordnance Survey National Grid Transformation of 1997 (cs_PRJCOD_OSTN02)

There are two elements which make this variation different from the standard Transverse Mercator (aka Gauss Kruger) projection. First, the basic projection parameters are hard coded to be those of the standard Ordnance Survey National Grid coordinate system. Thus, this variation really doesn't have any parameters at all.

Second, after the basic projection calculation is performed, the transformation known as Ordnance Survey National Transformation of 2002 (OSTN02) is applied. This transformation requires access to a data file which is expected to be named "OSTN02.txt" and is expected to reside in the primary data directory.

The end result of this variation is the ability to convert ETRF89 based geographic coordinates to National Grid coordinates, where the actual National Grid coordinates are the same (for a specific geographic point) as the 1936 National Grid coordinates.

Yes, this is strange as the datum shift calculation is actually performed on the cartesian coordinates which result from the application of the projection. That is why this datum shift calculation had to be performed here in the cartographic code.

As described above, this variation does not require any parameters. (In fact, what is considered a standard parameter for the Transverse Mercator projection, the Scale Reduction, is also ignored by this variation. The OSGB value of 0.9996012717 is hard coded in.)

Note that the OSTN97 and OSTN02 variations are very similar, but do not produce precisely the same results. Generally, it is assumed that the OSTN02 values are to be preffered. CS-MAP can be used to convert data sets which were derived from the OSTN97 transformation to OSTN02.

Datum Definition

The values of equatorial radius and eccentricity are extracted from the *datum* element of the *cs_Csprm_* structure. These are normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required elements are:

e_rad	The equatorial radius of the earth in meters.
eccent	This value represents the eccentricity of the ellipsoid.
to84_via	An integer code that specifies the technique that is to be used to convert geographic coordinates based on this datum to WGS84.

cs_Trmer_ Structure

The results of the one-time calculations are recorded in the `trmer` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CStrmerF*, *CStrmerI*,

CStrmerK, and *CStrmerC* functions require as their first argument.

Unity Pseudo Projection (CSunity)

These functions implement the pseudo projection referred to as the Unity projection. This projection enables coordinate systems to be defined in which the coordinates are actually latitude and longitude. The projection does nothing other than converting latitude and longitude values to/from the internal representation of degrees based on Greenwich to the form indicated by the coordinate system definition.

Technically all values are within the domain of this function. However, for the purposes of limit checking, the domain of this function is considered to consist of latitudes between -90 and +90 inclusive, and longitudes within the user defined range, inclusive of both the upper and lower limit. The Q function requires that the user definable range must be less than 540 degrees in extent.

CSunityF Forward conversion

```
int CSunityF (Const struct cs_Unity_ *unity,double xy [2],Const double ll
[2]);
```

This function simply converts the contents of the **ll** array from the internal representation of degrees based on Greenwich to the units and origin indicated by the **unity** argument and places the results in the **xy** array. *CSunityF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

The **xy** and **ll** arguments may point to the same array.

CSunityI Inverse conversion

```
int CSunityI (Const struct cs_Unity_ *unity,double ll [2],Const double xy
[2]);
```

This function expects the contents of **xy** to contain a latitude and longitude in the units and with the origin specified by the **unity** argument. These values are converted to the standard internal form, degrees based on Greenwich, and returned in **ll**. *CSunityI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system.

The **ll** and **xy** arguments may point to the same array.

CSunityK scale (K)

```
double CSunityK (Const struct cs_Unity_ *unity,Const double ll [2]);
```

This function simply returns a 1.0. Its arguments are ignored. Grid scale factor has no meaning with regard to latitudes and longitudes. However, since applications are usually unaware of the coordinate systems involved, this function provides an appropriate value.

CSunityC Covergence angle

```
double CSunityC (Const struct cs_Unity_ *unity,Const double ll [2]);
```

This function simply returns a 0.0. Its arguments are ignored.

CSunityQ definition Quality check

```
int CSunityQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CSunityQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Unity Pseudo Projection. *CS_cschk* examines those definition components

that are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CSunityQ* only examines those components specific to the Unity Pseudo Projection. *CSunityQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSunityQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and **list_sz** arguments respectively.

CSunityL Latitude/longitude check

```
int CSunityL (Const struct cs_Unity_ *unity,int cnt,Const double pnts
[][3]);
```

CSunityL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the constraints placed on internal geographic coordinates, namely (in terms of degrees) longitudes greater than -270 but less than or equal to +270; and latitudes in the range of -90 and +90, inclusive.

CSunityX Xy coordinate check

```
int CSunityX (Const struct cs_Unity_ *unity,int cnt,Const double pnts
[][3]);
```

In this special case, the coordinates provided in the list are indeed geographic coordinates, but possibly in units other than degrees and possibly referenced to a prime meridian other than Greenwich. *CSunityX* essentially converts the geographic coordinates provided to internal form, and then verifies that all of the resulting coordinates meet internal geographic coordinate requirements. If the converted list does indeed meet internal requirements (after conversion), **cs_CNVRT_OK** is returned; otherwise, **cs_CNVRT_DOMN** is returned.

CSunityS Setup

```
void CSunityS (struct cs_Csprm_ *csprm);
```

This function sets the projection scale to 1.0, captures the units and origin of the latitude and longitude system, sets up the user defined range, and sets pointers to *CSunityF*, *CSunityI*, *CSunityK,* and *CSunityC* in the appropriate locations of the **csprm** argument.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. The following parameters are used:

org_lng	The longitude, in degrees relative to the Greenwich Prime Meridian, of the prime meridian for this coordinate system.
Scale	The scale of the coordinate system. This one factor must convert degrees to coordinate system units by multiplication.
prj_prm1	The minimum value of the range of longitude desired for this coordinate system. Must be specified in user units, and relative to the user's origin. The range between this value and prj_prm2, upon conversion to degrees, must be greater than or equal to 360 and less than 540. Disable this feature by setting both prj_prm1 and prj_prm2 equal to zero.
prj_prm2	The maximum value of the range of longitude desired for this coordinate system. Must be specified in user units, and relative to the user's origin. The range between this value and prj_prm1, upon conversion to degrees, must be greater than or equal to 360 and less than 540. Disable this feature by setting both prj_prm2 and prj_prm1 equal to zero.

Van Der Grinten Projection (CSvdgrn)

This set of functions represent the Coordinate System Mapping Package's knowledge of the Van Der Grinten Projection. This projection is supported in spherical form only. The equatorial radius of the supplied ellipsoid is used as the radius of the sphere.

CSvdgrnF Forward conversion

```
int CSvdgrnF (Const struct cs_Vdgrn_ *vdgrn,double xy [2],Const double ll
[2]);
```

Given a properly initialized `cs_Vdgrn_` structure via the **vdgrn** argument, *CSvdgrnF* will convert the latitude and longitude provided in the **ll** array to X and Y coordinates, returning the result in the **xy** array. *CSvdgrnF* normally returns **cs_CNVRT_NRML**. If **ll** is not within the domain of the coordinate system, **xy** is set to a "rational" result and **cs_CNVRT_RNG** is returned.

CSvdgrnI Inverse conversion

```
int CSvdgrnI (Const struct cs_Vdgrn_ *vdgrn,double ll [2],Const double xy
[2]);
```

Given a properly initialized `cs_Vdgrn_` structure via the **vdgrn** argument, *CSvdgrnI* will convert the X and Y coordinates given in the **xy** array to latitude and longitude and return the result in the **ll** array. *CSvdgrnI* normally returns **cs_CNVRT_NRML**. It will return **cs_CNVRT_RNG** if the **xy** value is not within the domain of the coordinate system, or **cs_CNVRT_INDF** if the result is indefinite (e.g. longitude is not defined at the poles).

In both cases above, the **xy** and **ll** arrays may be the same array. The X coordinate and the longitude are carried in the first element in these arrays, the Y coordinate and the latitude in the second element. The latitude and longitude values are in degrees where negative values are used to represent west longitude and south latitude.

CSvdgrnK parallel scale (K)

```
double CSvdgrnK (Const struct cs_Vdgrn_ *vdgrn,Const double ll [2]);
```

CSvdgrnK returns the grid scale factor along a parallel at the geodetic location specified by the **ll** argument. As analytical formulas for this value have not been located, this value is arrived at empirically using spherical trigonometry.

CSvdgrnH meridian scale (H)

```
double CSvdgrnH (Const struct cs_Vdgrn_ *vdgrn,Const double ll [2]);
```

CSvdgrnH returns the grid scale factor along a meridian at the geodetic location specified by the **ll** argument. As analytical formulas for this value have not been located, this value is arrived at empirically using spherical trigonometry.

CSvdgrnC Convergence angle

```
double CSvdgrnC (Const struct cs_Vdgrn_ *vdgrn,Const double ll [2]);
```

CSvdgrnC returns the convergence angle in degrees east of north of the geodetic location specified by the **ll** argument. Analytical formulas for this value have not been located and the result is arrived at through the use of the *CS_azsphr* function.

CSvdgrnQ definition Quality check

```
int CSvdgrnQ (Const struct cs_Csdef_ *csdef,unsigned short prj_code,
              int *err_list [],int list_sz);
```

CSvdgrnQ determines if the coordinate system definition provided by the **csdef** argument is consistent with the requirements of the Van Der Grinten Projection. *CS_cschk* examines those definition components that are common to all coordinates systems (datum or ellipsoid reference, map scale, and units) and, therefore, *CSvdgrnQ* only examines those components specific to the Van Der Grinten Projection. *CSvdgrnQ* returns in **err_list** an integer code value for each error condition detected, being careful not to exceed the size of **err_list** as indicated by the **list_sz** argument. The number of errors detected, regardless of the size of **err_list**, is always returned. Refer to *CSerpt* for a description of the various error codes and their meaning. *CSvdgrnQ* may be called with the **NULL** pointer and/or a zero for the **err_list** and *list_sz* arguments respectively.

CSvdgrnL Latitude/longitude check

```
int CSvdgrnL (Const struct cs_Vdgrn_ *vdgrn,int cnt,Const double pnts
[][3]);
```

CSvdgrnL determines if the geographic coordinates, great circles, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system provided by the **vdgrn** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a great circle (**cnt** == 2), or a closed region (**cnt** > 3). *CSvdgrnsL*'s return value will apply to all coordinates, coordinates on the great circles, and all coordinates within the regions thus defined. *CSvdgrnL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject geographic coordinates are outside of the mathematical domain of the coordinate system.

CSvdgrnX Xy coordinate check

```
int CSvdgrnX (Const struct cs_Vdgrn_ *vdgrn,int cnt,Const double pnts
[][3]);
```

CSvdgrnX determines if the cartesian coordinates, lines, and/or regions defined by the coordinate list provided by the **pnts** and **cnt** arguments are within the mathematical domain of the coordinate system

provided by the **vdgrn** argument. The **pnts** and **cnt** arguments can define a single coordinate (**cnt** == 1), a line (**cnt** == 2), or a closed region (**cnt** > 3). *CSvdgrnsX*'s return value will apply to all coordinates, coordinates on the lines, and all coordinates within the regions thus defined. *CSvdgrnL* returns **cs_CNVRT_OK** if all subject coordinates are within the mathematical domain of the coordinate system. **cs_CNVRT_DOMN** is returned if one or more of the subject coordinates is outside of the mathematical domain of the coordinate system.

CSvdgrnS Setup

```
void CSvdgrnS (struct cs_Csprm_ *csprm);
```

The *CSvdgrnS* function performs all calculations that need only be performed once, given the definition of a specific coordinate system. That is, once the central meridian and other projection parameters are known, there are many calculations which need only be performed once. *CSvdgrnS* performs these calculations and saves the results in the `cs_Csprm_` structure provided by its argument, **csprm**. Thus, the argument provided to *CSvdgrnS* serves as the source for input and the repository for the results as described below.

Coordinate System Definition

The definition of the coordinate system is extracted from the `csdef` element of the `cs_Csprm_` structure. Usually, this is obtained from the Coordinate System Dictionary by the *CS_csdef* function; but can be provided by the application at run time. The following parameters are used:

org_lng	The longitude, in degrees, of the origin of the projection.
scale	The scale of the coordinate system. This one factor must include the conversion from meters to coordinate system units and the mapping scale that is to be applied.
x_off	The false easting to be applied to all X coordinates, usually selected to cause all X coordinates within the coordinate system to be positive values of reasonable size. This is the X coordinate of the coordinate system origin.
y_off	The false northing to be applied to all Y coordinates. This is the Y coordinate of the coordinate system origin.
quad	An integer that indicates the cartesian quadrant of the coordinate system, 1 thru 4. A negative value indicates that the axes are to be swapped after the coordinates have been placed in the indicated quadrant.

Datum Definition

The value of equatorial radius is extracted from the *datum* element of the `cs_Csprm_` structure and used as the radius of the sphere. This is normally obtained from the Ellipsoid Dictionary by the *CS_dtloc* function, but may be supplied by the application at run time. Specifically, the required element is:

e_rad	The radius of the earth, as a sphere, in meters.

cs_Vdgrn_ Structure

The results of the one-time calculations are recorded in the `vdgrn` element of the `prj_prms` union of the `cs_Csprm_` structure. It is a pointer to this initialized structure that the *CSvdgrnF*, *CSvdgrnI*, *CSvdgrnK*, *CSvdgrnH*, and *CSvdgrnC* functions require as their first argument.

Geodetic Conversion (Datum) Functions

Functions used in the conversion of geographic coordinates from one datum to another are described in this section.

Note that for all transformations, functions for both 2D and 3D conversions are provided. Generally, you should only use the 3D version if you are dealing with a three diemnsional database. That is, a database which will carry the resulting Z value so when it is time to invert the conversion, you will be able to supply the Z value to the inverse algorithm.

In the case of a 2D database, you will not be able to supply the Z coordinate when it is time to calculate the inverse, and thus the reulting horizontal components (X and Y) will not be the same as the oroginal coordinates.

Three Parameter Transformation

The Three Parameter Transformation implements a datum shift by translating geo-centric coordinates in three dimensions. The three parameters are the components of the translation vector. They are expressed in meters, and represent the shift from the local reference system (datum) to the WGS84 reference system (datum).

CS_3pInit 3 Parameter INITialize

```
struct cs_Parm3_ *CS_3pInit (Const struct cs_Datum_* srcDatum,
                             Const struct cs_Datum_* trgDatum)
```

CS_3pInit is essentially a constructor for the `cs_Parm3_` structure in C syntax. *CS_3pInit* will return a pointer to a *malloc*'ed `cs_Parm3_` structure which has been initialized for the use of the Three Parameter transformation technique to convert geodetic coordinates from the datum indicated by the **srcDatum** argument to the datum indicated by the **trgDatum**. (**trgDatum** is usually WGS84, but this is not required.)

The equivalent destructor is the *CS_free* function.

CS_3pInit returns **NULL** in the event of failure. Failure is unlikely and can only be caused by a *malloc* failure or completely absurd numbers in either of the `cs_Datum_` structures provided by the arguments.

CS_3p3dFowrd 3 Parameter 3D FOrWaRD conversion

```
int CS_3p3dFowrd (double trgLl [3],Const double srcLl [3],
                  Const struct cs_Parm3_ *parm3)
```

Given a Three Parameter transformation in the form of an initialized `cs_Parm3_` structure, *CS_3p3dFowrd* calculates the datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument.

The conversion is a full three dimensional calculation.

CS_3p3dFowrd returns zero to indicate success and –1 to indicate failure. Failure can only be caused by a failure of the inverse geocentric calculation to converge.

CS_3p2dFowrd 3 Parameter 2D FOrWaRD conversion

```
int CS_3p2dFowrd (double trgLl [3],Const double srcLl [3],
                  Const struct cs_Parm3_ *parm3)
```

Given a Three Parameter transformation in the form of an initialized `cs_Parm3_` structure, *CS_3p2dFowrd* calculates the datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. The conversion is a two dimensional calculation, the third element of **srcLl** is simply copied to the **trgLl** array.

CS_3p2dFowrd returns zero to indicate success and –1 to indicate failure. Failure can only be caused by a failure of the inverse geocentric calculation to converge.

CS_3p3dInvrs 3 Parameter 3D INVeRSe transformation

```
int CS_3p3dInvrs (double trgLl [3],Const double srcLl [3],
                  Const struct cs_Parm3_ *parm3)
```

Given a Three Parameter transformation in the form of an initialized `cs_Parm3_` structure, *CS_3p3dInvrs* calculates the inverse datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. However, unlike *CS_3p3dFowrd*, this function assumes that the coordinates provided by the **srcLl** argument are based on the target datum provided when the provided `cs_Parm3_` structure was initialized. The results are based, of course, on the source datum provided when the provided `cs_Parm3_` structure was initialized. The conversion is a three dimensional calculation.

CS_3p3dInvrs returns zero to indicate success and –1 to indicate failure. Failure can only be caused by a failure of the inverse geocentric calculation to converge.

CS_3p2dInvrs 3 Parameter 2D INVeRSe transformation

```
int CS_3p2dInvrs (double trgLl [3],Const double srcLl [3],
                  Const struct cs_Parm3_ *parm3)
```

Given a Three Parameter transformation in the form of an initialized `cs_Parm3_` structure, *CS_3p2dInvrs* calculates the inverse datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. However, unlike *CS_3p2dFowrd*, this function assumes that the coordinates provided by the **srcLl** argument are based on the target datum provided when the provided `cs_Parm3_` structure was initialized. The results are based, of course, on the source datum provided when the provided `cs_Parm3_` structure was initialized. The conversion is a two dimensional calculation, the third element of the **srcLl** argument is copied to the third element of the **trgLl** argument.

CS_3p2dInvrs returns zero to indicate success and –1 to indicate failure. Failure can only be caused by a failure of the inverse geocentric calculation to converge.

Four Parameter Transformation

The original developers of CS-MAP invented this transformation technique, so you may want to remove it from your distribution. Since are Three, Six, and Seven Parameter Transformations in general use, it seemed obvious that there should also exist a Four Parameter Transformation.

The Four Parameter Transformation is arrived at by eliminating the three rotation angles from the Seven Parameter Transformation. The same result could be achieved by setting the rotation angles of the Seven Parameter Transformation to zero.

CS_4pInit 4 Parameter INITialize

```
struct cs_Parm4_ *CS_4pInit (Const struct cs_Datum_* srcDatum,
                             Const struct cs_Datum_* trgDatum)
```

CS_4pInit is essentially a constructor for the `cs_Parm4_` structure in C syntax. *CS_4pInit* will return a pointer to a *malloc*'ed `cs_Parm4_` structure which has been initialized for the use of the Four Parameter transformation technique to convert geodetic coordinates from the datum indicated by the **srcDatum** argument to the datum indicated by the **trgDatum**. (**trgDatum** is usually WGS84, but this is not required.)

The equivalent destructor is the *CS_free* function.

CS_4pInit returns **NULL** in the event of failure. Failure is unlikely and can only be caused by a *malloc* failure or completely absurd numbers in either of the `cs_Datum_` structures provided by the arguments.

CS_4p3dFowrd 4 Parameter 3D FOrWaRD conversion

```
int CS_4p3dFowrd (double trgLl [3],Const double srcLl [3],
                  Const struct cs_Parm4_ *parm4)
```

Given a Four Parameter transformation in the form of an initialized `cs_Parm4_` structure, *CS_4p3dFowrd* calculates the datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. The conversion is a full three dimensional calculation.

CS_4p3dFowrd returns zero to indicate success and –1 to indicate failure. Failure can only be caused by a failure of the inverse geocentric calculation to converge.

CS_4p2dFowrd 4 Parameter 2D FOrWaRD conversion

```
int CS_4p2dFowrd (double trgLl [3],Const double srcLl [3],
      Const struct cs_Parm4_ *parm4)
```

Given a Four Parameter transformation in the form of an initialized `cs_Parm4_` structure, *CS_4p2dFowrd* calculates the datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. The conversion is a two dimensional calculation, the third element of **srcLl** is simply copied to the **trgLl** array.

CS_4p2dFowrd returns zero to indicate success and –1 to indicate failure. Failure can only be caused by a failure of the inverse geocentric calculation to converge.

CS_4p3dInvrs 4 Parameter 3D INVeRSe transformation

```
int CS_4p3dInvrs (double trgLl [3],Const double srcLl [3],
      Const struct cs_Parm4_ *parm4)
```

Given a Four Parameter transformation in the form of an initialized `cs_Parm4_` structure, *CS_4p3dInvrs* calculates the inverse datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. However, unlike *CS_4p3dFowrd*, this function assumes that the coordinates provided by the **srcLl** argument are based on the target datum provided when the provided `cs_Parm4_` structure was initialized. The results are based, of course, on the source datum provided when the provided `cs_Parm4_` structure was initialized. The conversion is a three dimensional calculation.

CS_4p3dInvrs returns zero to indicate success and –1 to indicate failure. Failure can only be caused by a failure of the inverse geocentric calculation to converge.

CS_4p2dInvrs 4 Parameter 2D INVeRSe transformation

```
int CS_4p2dInvrs (double trgLl [3],Const double srcLl [3],
      Const struct cs_Parm4_ *parm4)
```

Given a Four Parameter transformation in the form of an initialized `cs_Parm4_` structure, *CS_4p2dInvrs* calculates the inverse datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. However, unlike *CS_4p2dFowrd*, this function assumes that the coordinates provided by the **srcLl** argument are based on the target datum provided when the provided `cs_Parm4_` structure was initialized. The results are based, of course, on the source datum provided when the provided `cs_Parm4_` structure was initialized. The conversion is a two dimensional calculation, the third element of the **srcLl** argument is copied to the third element of the **trgLl** argument.

CS_4p2dInvrs returns zero to indicate success and –1 to indicate failure. Failure can only be caused by a failure of the inverse geocentric calculation to converge.

Six Parameter Transformation

The Six Parameter Transformation is used occassionally. It is, essentially, the Seven Parameter Transformation with the Scale parameter set to zero. Remember, that the scale parameter is actually the number of parts per million the true scale factor differs from unity. Thus, a zero scale parameter actually means a scale factor of unity (1.0).

CS_6pInit 6 Parameter INITialize

```
struct cs_Parm6_ *CS_6pInit (Const struct cs_Datum_* srcDatum,
     Const struct cs_Datum_* trgDatum)
```

CS_6pInit is essentially a constructor for the `cs_Parm6_` structure in C syntax. *CS_6pInit* will return a pointer to a *malloc*'ed `cs_Parm6_` structure which has been initialized for the use of the Six Parameter transformation technique to convert geodetic coordinates from the datum indicated by the **srcDatum** argument to the datum indicated by the **trgDatum**. (**trgDatum** is usually WGS84, but this is not required.)

The equivalent destructor is the *CS_free* function.

CS_6pInit returns **NULL** in the event of failure. Failure is unlikely and can only be caused by a *malloc* failure or completely absurd numbers in either of the `cs_Datum_` structures provided by the arguments.

CS_6p3dFowrd 6 Parameter 3D FOrWaRD conversion

```
int CS_6p3dFowrd (double trgLl [3],Const double srcLl [3],Const struct
cs_Parm6_ *parm6)
```

Given a Six Parameter transformation in the form of an initialized `cs_Parm6_` structure, *CS_6p3dFowrd* calculates the datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. The conversion is a full three dimensional calculation.

CS_6p3dFowrd returns zero to indicate success and –1 to indicate failure. Failure can only be caused by a failure of the inverse geocentric calculation to converge.

CS_6p2dFowrd 6 Parameter 2D FOrWaRD conversion

```
int CS_6p2dFowrd (double trgLl [3],Const double srcLl [3],
      Const struct cs_Parm6_ *parm6)
```

Given a Six Parameter transformation in the form of an initialized `cs_Parm6_` structure, *CS_6p2dFowrd* calculates the datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. The conversion is a two dimensional calculation, the third element of the **srcLl** is simply copied to the **trgLl** array.

CS_6p2dFowrd returns zero to indicate success and –1 to indicate failure. Failure can only be caused by a failure of the inverse geocentric calculation to converge.

CS_6p3dInvrs 6 Parameter 3D INVeRSe transformation

```
int CS_6p3dInvrs (double trgLl [3],Const double srcLl [3],
      Const struct cs_Parm6_ *parm6)
```

Given a Six Parameter transformation in the form of an initialized `cs_Parm6_` structure, *CS_6p3dInvrs* calculates the inverse datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. However, unlike *CS_6p3dFowrd*, this function assumes that the coordinates provided by the **srcLl** argument are based on the target datum provided when the provided `cs_Parm6_` structure was initialized. The results are based, of course, on the source datum provided when the provided `cs_Parm6_` structure was initialized. The conversion is a three dimensional calculation.

CS_6p3dInvrs returns zero to indicate success and –1 to indicate failure. Failure can only be caused by a failure of the inverse geocentric calculation to converge.

CS_6p2dInvrs 6 Parameter 2D INVeRSe transformation

```
int CS_6p2dInvrs (double trgLl [3],Const double srcLl [3],Const struct
cs_Parm6_ *parm6)
```

Given a Six Parameter transformation in the form of an initialized cs_Parm6_ structure, CS_6p2dInvrs calculates the inverse datum transformation. That is, the coordinates provided by the srcLl argument are transformed and the results are returned in the array indicated by the trgLl argument. However, unlike CS_6p2dFowrd, this function assumes that the coordinates provided by the srcLl argument are based on the target datum provided when the provided cs_Parm6_ structure was initialized. The results are based, of course, on the source datum provided when the provided cs_Parm6_ structure was initialized. The conversion is a two dimensional calculation, the third element of the srcLl argument is copied to the third element of the trgLl argument.

CS_6p2dInvrs returns zero to indicate success and –1 to indicate failure. Failure can only be caused by a failure of the inverse geocentric calculation to converge.

Seven Parameter Transformation

This transformation is the rigorous implementation of the widely used Seven Parameter Transformation. The seven parameters include:

- the three components of the geocentric translation vector expressed in meters, and
- three rotation angles. one for each geocentric axis, expressed in seconds of arc, and
- a scale factor express in the deviiation from unit in parts per million.

Note that a negative scale factor parameter indicates a resulting scale factor less than unity.

Please note that this transformation is called the Bursa/Wolf Transformation by many. However, in the contect of CS-MAP, the term Bursa/Wolf Transformation has a decidely different meaning.

CS_7pInit 7 Parameter INITialize

```
struct cs_Parm7_ *CS_7pInit (Const struct cs_Datum_* srcDatum,
     Const struct cs_Datum_* trgDatum)
```

CS_7pInit is essentially a constructor for the `cs_Parm7_` structure in C syntax. *CS_7pInit* will return a pointer to a *malloc*'ed `cs_Parm7_` structure which has been initialized for the use of the Seven Parameter transformation technique to convert geodetic coordinates from the datum indicated by the **srcDatum** argument to the datum indicated by the **trgDatum**. (**trgDatum** is usually WGS84, but this is not required.)

The equivalent destructor is the *CS_free* function.

CS_7pInit returns **NULL** in the event of failure. Failure is unlikely and can only be caused by a *malloc* failure or completely absurd numbers in either of the `cs_Datum_` structures provided by the arguments.

CS_7p3dFowrd 7 Parameter 3D FOrWaRD conversion

```
int CS_7p3dFowrd (double trgLl [3],Const double srcLl [3],
      Const struct cs_Parm7_ *parm7)
```

Given a Seven Parameter transformation in the form of an initialized `cs_Parm7_` structure, *CS_7p3dFowrd* calculates the datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. The conversion is a full three dimensional calculation.

CS_7p3dFowrd returns zero to indicate success and –1 to indicate failure. Failure can only be caused by a failure of the inverse geocentric calculation to converge.

CS_7p2dFowrd 7 Parameter 2D FOrWaRD conversion

```
int CS_7p2dFowrd (double trgLl [3],Const double srcLl [3],
      Const struct cs_Parm7_ *parm7)
```

Given a Seven Parameter transformation in the form of an initialized `cs_Parm7_` structure, *CS_7p2dFowrd* calculates the datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. The conversion is a two dimensional calculation, the third element of **srcLl** is simply copied to the **trgLl** array.

CS_7p2dFowrd returns zero to indicate success and –1 to indicate failure. Failure can only be caused by a failure of the inverse geocentric calculation to converge.

CS_7p3dInvrs 7 Parameter 3D INVeRSe transformation

```
int CS_7p3dInvrs (double trgLl [3],Const double srcLl [3],
      Const struct cs_Parm7_ *parm7)
```

Given a Seven Parameter transformation in the form of an initialized `cs_Parm7_` structure, *CS_7p3dInvrs* calculates the inverse datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. However, unlike *CS_7p3dFowrd*, this function assumes that the coordinates provided by the **srcLl** argument are based on the target datum provided when the provided `cs_Parm7_` structure was initialized. The results are based, of course, on the source datum provided when the provided `cs_Parm7_` structure was initialized. The conversion is a three dimensional calculation.

CS_7p3dInvrs returns zero to indicate success and –1 to indicate failure. Failure can only be caused by a failure of the inverse geocentric calculation to converge.

CS_7p2dInvrs 7 Parameter 2D INVeRSe transformation

```
int CS_7p2dInvrs (double trgLl [3],Const double srcLl [3],
     Const struct cs_Parm7_ *parm7)
```

Given a Seven Parameter transformation in the form of an initialized `cs_Parm7_` structure, *CS_7p2dInvrs* calculates the inverse datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. However, unlike *CS_7p2dFowrd*, this function assumes that the coordinates provided by the **srcLl** argument are based on the target datum provided when the provided `cs_Parm7_` structure was initialized. The results are based, of course, on the source datum provided when the provided `cs_Parm7_` structure was initialized. The conversion is a two dimensional calculation, the third element of the **srcLl** argument is copied to the third element of the **trgLl** argument.

CS_7p2dInvrs returns zero to indicate success and –1 to indicate failure. Failure can only be caused by a failure of the inverse geocentric calculation to converge.

Bursa/Wolf Transformation

In the context of CS-MAP, the term Bursa/Wolf Transformation refers to an approximation of the Seven Parameter Transformation. This approximation was widely used prior to 1990 and is included in CS-MAP so that users can duplicate results prosuced by this approximation.

The approximation is based on three assumptions:

- the sine of a small angle is equal to the angle itself when expressed in radians; and
- the product of the sine of two small angles is zero; and
- the cosine of a small angle is unity (i.e. 1.0).

Taking these assumptions to be true, the calculation of the normal Seven Parameter Transformation is greatly simplified. Thus, its use was highly popular before PC's became to be used widely in GIS/Mapping/Geodetic field work. This is especially true as the rotation angle usually used in a Seven Parameter Transformation are indeed very small (on the order of a few seconds of arc).

The end result as far as CS-MAP is concerned, therefore, is that the term Seven Parameter Transformation refers to the rigourous implementation of the transformation technique, while the Bursa/Wolf refers to the frequently used approximation described above. New work should use the Seven Parameter Transformation. the Bursa/Wolf should only be used when working with data previously processed using that transformation.

CS_bwInit Bursa Wolf INITialize

```
struct cs_Bursa_ *CS_bwInit (Const struct cs_Datum_* srcDatum,
     Const struct cs_Datum_* trgDatum)
```

CS_bwInit is essentially a constructor for the `cs_Bursa_` structure in C syntax. *CS_bwInit* will return a pointer to a *malloc*'ed `cs_Bursa_` structure which has been initialized for the use of the Bursa/Wolf transformation technique to convert geodetic coordinates from the datum indicated by the **srcDatum** argument to the datum indicated by the **trgDatum**. (**trgDatum** is usually WGS84, but this is not required.)

The equivalent destructor is the *CS_free* function.

CS_bwInit returns **NULL** in the event of failure. Failure is unlikely and can only be caused by a *malloc* failure or completely absurd numbers in either of the `cs_Datum_` structures provided by the arguments.

CS_bw3dFowrd Bursa Wolf 3D FOrWaRD conversion

```
int CS_bw3dFowrd (double trgLl [3],Const double srcLl [3],
      Const struct cs_Bursa_ *bursa)
```

Given a Bursa/Wolf transformation in the form of an initialized `cs_Bursa_` structure, *CS_bw3dFowrd* calculates the datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. The conversion is a full three dimensional calculation.

CS_bw3dFowrd returns zero to indicate success and –1 to indicate failure. Failure can only be caused by a failure of the inverse geocentric calculation to converge.

CS_bw2dFowrd Bursa Wolf 2D FOrWaRD conversion

```
int CS_bw2dFowrd (double trgLl [3],Const double srcLl [3],
      Const struct cs_Bursa_ *bursa)
```

Given a Bursa/Wolf transformation in the form of an initialized `cs_Bursa_` structure, *CS_bw2dFowrd* calculates the datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. The conversion is a two dimensional calculation, the third element of **srcLl** is simply copied to the **trgLl** array.

CS_bw2dFowrd returns zero to indicate success and –1 to indicate failure. Failure can only be caused by a failure of the inverse geocentric calculation to converge.

CS_bw3dInvrs Bursa Wolf 3D INVeRSe transformation

```
int CS_bw3dInvrs (double trgLl [3],Const double srcLl [3],
      Const struct cs_Bursa_ *bursa)
```

Given a Bursa/Wolf transformation in the form of an initialized `cs_Bursa_` structure, *CS_bw3dInvrs* calculates the inverse datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. However, unlike *CS_bw3dFowrd*, this function assumes that the coordinates provided by the **srcLl** argument are based on the target datum provided when the provided `cs_Bursa_` structure was initialized. The results are, of course, based on the source datum provided when the provided `cs_Bursa_` structure was initialized. The conversion is a three dimensional calculation.

CS_bw3dInvrs returns zero to indicate success and –1 to indicate failure. Failure can only be caused by a failure of the inverse geocentric calculation to converge.

CS_bw2dInvrs Bursa Wolf 2D INVeRSe transformation

```
int CS_bw2dInvrs (double trgLl [3],Const double srcLl [3],
     Const struct cs_Bursa_ *bursa)
```

Given a Bursa/Wolf transformation in the form of an initialized `cs_Bursa_` structure, *CS_bw2dInvrs* calculates the inverse datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. However, unlike *CS_bw2dFowrd*, this function assumes that the coordinates provided by the **srcLl** argument are based on the target datum provided when the provided `cs_Bursa_` structure was initialized. The results are based, of course, on the source datum provided when the provided `cs_Bursa_` structure was initialized. The conversion is a two dimensional calculation, the third element of the **srcLl** argument is copied to the third element of the **trgLl** argument.

CS_bw2dInvrs returns zero to indicate success and –1 to indicate failure. Failure can only be caused by a failure of the inverse geocentric calculation to converge.

DMA Multiple Regression

The DMA Multiple Regression Transformation implements the technique published by the DMA (Defense Mapping Agency, USA) in is defining document DMA Technical Report 8350.2-B (1 December 1987). In this document, the DMA published polynomial based transformations for converting from many local reference systems to WGS84. These transformations were developed using Multiple Regression techniques.

In the context of CS-MAP, there exists a *.mrt* file for each such transformation. The *.mrt* file essentially carries the coefficients of the transformation. The code described in the following sub-sections builds a transformation based on the information contained in the *.mrt* file. All such files are expected to reside in the main data directory.

Note that *.mrt* files are produced by the Dictionary Compiler from the data in an ASCII file named *MREG.ASC*. The *MREG.ASC* file contains a transcription of the data presented in the DMA report and is, of course, in appropriate form for version control.

CS_dmaMrInit DMA Multiple Regression INITialize

```
struct cs_dmaMReg_ *CS_dmaMrInit (Const struct cs_Datum_* srcDatum,
     Const struct cs_Datum_* trgDatum)
```

CS_dmaMrInit is essentially a constructor for the `cs_dmaMReg_` structure in C syntax. *CS_dmaMrInit* will return a pointer to a *malloc*'ed `cs_dmaMReg_` structure which has been initialized for the use of the DMA Multiple Regression transformation technique to convert geodetic coordinates from the datum indicated by the **srcDatum** argument to the datum indicated by the **trgDatum**. (**trgDatum** is usually WGS84, but this is not required.)

The equivalent destructor is the *CS_free* function.

CS_dmaMrInit returns **NULL** in the event of failure. Failure is usually caused by the non-existence or corruption of an appropriately named multiple regression definition file. Multiple regression definition files have the same name as the datum key name with the .mrt extension applied.

CS_dmaMr3dFowrd DMA Multiple Regression 3D FOrWaRD conversion

```
int CS_dmaMr3dFowrd (double trgLl [3],Const double srcLl [3],
      Const struct cs_dmaMReg_ *mreg)
```

Given a DMA Multiple Regression transformation in the form of an initialized `cs_dmaMReg_` structure, *CS_dmaMr3dFowrd* calculates the datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. The conversion is a full three dimensional calculation.

CS_dmaMr3dFowrd returns zero to indicate success and +1 to indicate failure. Failure can occur if the coordinates to be converted are outside of the domain of the multiple regression transformation. Note that the multiple regression formulas use normalized coordinates for the actual calculation. CS-MAP assumes that a normalized coordinate greater than 1.4 or less than –1.4 are outside the domain of the multiple regression definition.

CS_dmaMr2dFowrd DMA Multiple Regression 2D FOrWaRD conversion

```
int CS_dmaMr2dFowrd (double trgLl [3],Const double srcLl [3],
      Const struct cs_dmaMReg_ *mreg)
```

Given a DMA Multiple Regression transformation in the form of an initialized `cs_dmaMReg_` structure,

CS_dmaMr2dFowrd calculates the datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. The conversion is a two dimensional calculation, the third element of **srcLl** is simply copied to the **trgLl** array.

CS_dmaMr2dFowrd returns zero to indicate success and +1 to indicate failure. Failure can occur if the coordinates to be converted are outside of the domain of the multiple regression transformation. Note that the multiple regression formulas use normalized coordinates for the actual calculation. CS-MAP assumes that a normalized coordinate greater than 1.4 or less than –1.4 are outside the domain of the multiple regression definition.

CS_dmaMr3dInvrs DMA Multiple Regression 3D INVeRSe transformation

```
int CS_dmaMr3dInvrs (double trgLl [3],Const double srcLl [3],
      Const struct cs_dmaMReg_ *mreg)
```

Given a DMA Multiple Regression transformation in the form of an initialized `cs_dmaMReg_` structure, *CS_dmaMr3dInvrs* calculates the inverse datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. However, unlike *CS_dmaMr3dFowrd*, this function assumes that the coordinates provided by the **srcLl** argument are based on the target datum provided when the provided `cs_dmaMReg_` structure was initialized. The results are based, of course, on the source datum provided when the provided `cs_dmaMReg_` structure was initialized. The conversion is a three dimensional calculation.

CS_dmaMr3dInvrs returns zero to indicate success and +1 to indicate failure. Failure is caused by a failure of the iterative forward calculation to converge, or the supplied coordinate being outside the domain of the multiple regression definition.

CS_dmaMr2dInvrs DMA Multiple Regression 2D INVeRSe transformation

```
int CS_dmaMr2dInvrs (double trgLl [3],Const double srcLl [3],
      Const struct cs_dmaMReg_ *mreg)
```

Given a DMA Multiple Regression transformation in the form of an initialized `cs_dmaMReg_` structure, *CS_dmaMr2dInvrs* calculates the inverse datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. However, unlike *CS_dmaMr2dFowrd*, this function assumes that the coordinates provided by the **srcLl** argument are based on the target datum provided when the provided `cs_dmaMReg_` structure was initialized. The results are based, of course, on the source datum provided when the provided `cs_dmaMReg_` structure was initialized. The conversion is a two dimensional calculation, the third element of the **srcLl** argument is copied to the third element of the **trgLl** argument.

CS_dmaMr2dInvrs returns zero to indicate success and +1 to indicate failure. Failure is caused by a failure of the iterative forward calculation to converge, or the supplied coordinate being outside the domain of the multiple regression definition.

DMA Molodensky Transformation

The DMA Molodensky (I've seen this spelled in a number of ways, this spelling seems to be popular in the US) Transformation is simply another mathematical technique for calculating the Three Parameter Transformation. This technique is a bit faster, and the inverse does not require an iterative technique as the Three Parameter Transformation does.

This transformation is widely used as it was prominently published in DMA TR 8350.2-B which is the primary document used by software developers when implementing datum shift software.

While the DMA Molodensky and Three Parameter Transformations accomplish the same thing, since the mathematical techniques used are quite different, the results do not match precisely.

CS_moInit MOlodensky INITialize

```
struct cs_Molo_ *CS_moInit (Const struct cs_Datum_* srcDatum,
     Const struct cs_Datum_* trgDatum)
```

CS_moInit is essentially a constructor for the `cs_Molo_` structure in C syntax. *CS_moInit* will return a pointer to a *malloc*'ed `cs_Molo_` structure which has been initialized for the use of the Molodensky transformation technique to convert geodetic coordinates from the datum indicated by the **srcDatum** argument to the datum indicated by the **trgDatum**. (**trgDatum** is usually WGS84, but this is not required.)

The equivalent destructor is the *CS_free* function.

CS_moInit returns **NULL** in the event of failure. Failure is unlikely and can only be caused by a *malloc* failure.

CS_mo3dFowrd MOlodensky 3D FOrWaRD conversion

```
int CS_mo3dFowrd (double trgLl [3],Const double srcLl [3],
      Const struct cs_Molo_ *molo)
```

Given a Molodensky transformation in the form of an initialized `cs_Molo_` structure, *CS_mo3dFowrd* calculates the datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. The conversion is a full three dimensional calculation.

CS_mo3dFowrd returns zero to indicate success. Currently, this function is always successful.

CS_mo2dFowrd MOlodensky 2D FOrWaRD conversion

```
int CS_mo2dFowrd (double trgLl [3],Const double srcLl [3],
      Const struct cs_Molo_ *molo)
```

Given a Molodensky transformation in the form of an initialized `cs_Molo_` structure, *CS_mo2dFowrd* calculates the datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. The conversion is a two dimensional calculation, the third element of **srcLl** is simply copied to the **trgLl** array.

CS_mo2dFowrd returns zero to indicate success. Currently, this function is always successful.

CS_mo3dInvrs MOlodensky 3D INVeRSe transformation

```
int CS_mo3dInvrs (double trgLl [3],Const double srcLl [3],
      Const struct cs_Molo_ *molo)
```

Given a Molodensky transformation in the form of an initialized `cs_Molo_` structure, *CS_mo3dInvrs* calculates the inverse datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. However, unlike *CS_mo3dFowrd*, this function assumes that the coordinates provided by the **srcLl** argument are based on the target datum provided when the provided `cs_Molo_` structure was initialized. The results are based, of course, on the source datum provided when the provided `cs_Molo_` structure was initialized. The conversion is a three dimensional calculation.

CS_mo3dInvrs returns zero to indicate success. Currently, this function is always successful.

CS_mo2dInvrs MOlodensky 2D INVeRSe transformation

```
int CS_mo2dInvrs (double trgLl [3],Const double srcLl [3],
     Const struct cs_Molo_ *molo)
```

Given a Molodensky transformation in the form of an initialized `cs_Molo_` structure, *CS_mo2dInvrs* calculates the inverse datum transformation. That is, the coordinates provided by the **srcLl** argument are transformed and the results are returned in the array indicated by the **trgLl** argument. However, unlike *CS_mo2dFowrd*, this function assumes that the coordinates provided by the **srcLl** argument are based on the target datum provided when the provided `cs_Molo_` structure was initialized. The results are based, of course, on the source datum provided when the provided `cs_Molo_` structure was initialized. The conversion is a two dimensional calculation, the third element of the **srcLl** argument is copied to the third element of the **trgLl** argument.

CS_mo2dInvrs returns zero to indicate success and –1 to indicate failure. Failure can only be caused by a failure of the iterative inverse calculation to converge.

Geocentric Coordinate Calculation

Geocentric coordinates are the coordinates of a point, often on the surface of the earth, based on a three dimensional cartesian coordinate system whose origin is the center of the earth. The cartesian coordinate system is a right handed system where:

- the X / Y plane of the cartesian system is coplanar with the plane of the equator of the earth; and
- the X axis intersects the ellipsoid at the point of zero longitude (Greenwich) and zero latiotude (equator); and
- the Y axis is orthogonal to the X axis and is in the plane of the equator (i.e. intersects the ellipsoid in the Indian Ocean); and
- the Z axis protrudes from the ellipsoid at the north pole.

The functions described here are used extensively in datum shift calculations, and are useful for a wide variety of other purposes. Note that the equatorial radius and eccentricity squared are used to specify the ellipsoid. Also note that the cartesian coordinates are in (must be in) the same units that are used to specify the equatorial radius of the ellipsoid.

CS_llhToXyz LongLatHgt TO XYZ geocentric

```
void CS_llhToXyz (double xyz [3],Const double llh [3],double e_rad,double
e_sq)
```

CS_llhToXyz converts the geodetic coordinates provided by the **llh** argument to geocentric form and returns the result in the array provided by the **xyz** argument. The **e_rad** and **e_sq** arguments must

indicate the equatorial radius and square of the eccentricity, respectively, for the ellipsoid upon which the calculation is to be based.

The third element of the **llh** array is considered to be height above the ellipsoid. The equatorial radius, **e_rad**, must be specified in the same linear units as used to specify the ellipsoidal height. The results returned in the **xyz** array will be based on the same linear unit. Usually, the linear unit involved is meters.

CS_xyzToLlh XYZ geocentric TO LongLatHgt

```
int CS_xyzToLlh (double llh [3],Const double xyz [3],double e_rad,double
e_sq)
```

CS_xyzToLlh converts the geocentric coordinates provided by the **xyz** argument to geodetic form and returns the result in the array provided by the **llh** argument. The **e_rad** and **e_sq** arguments must indicate the equatorial radius and square of the eccentricity, respectively, for the ellipsoid upon which the calculation is to be based.

The equatorial radius, **e_rad**, must be specified using the same unit as the geocentric coordinates provided by the **xyz** argument. The third element of the returned geodetic coordinate is height above the ellipsoid and is in the same units used to specify the equatorial radius of the ellipsoid. The linear unit involved is usually meters.

CS_xyzToLlh returns a zero to indicate success. A –1 is returned to indicate failure. This calculation requires an iterative solution to obtain the necessary accuracy. Failure is caused when this iterative solution does not converge.

Australian Geodetic Datum of 1966

This transformation converts geographic coordinates based on the Australian Geodetic Datum of 1966 to geographic coordinates based on the Geocentric Datum of Australia (1994). The conversion is based on grid shift files which are in the in the Canadian National Transformation, Version 2, format. The files which are to be used must be listed in the Geodetic Data Catalog named Agd66ToGda94.gdc.

Note that AGD66 applies to certain Australian states, while AGD84 applies to others. CS-MAP uses the Geodetic Data Catalog feature to select the appropriate file from the list provided. Therefore, to the end user AGD66 looks like a single entity.

CSagd66Cls Australian Geodetic Datum of 1966 CLoSe

```
void CSagd66Cls (void)
```

Use *CSagd66Cls* to release all resources allocated by a previous call to *CSagd66Init*. Obviously, subsequent calls to the *CSagd66ToGda94* or *CSgda94ToAgd66* will fail until such time that *CSagd66Init* is called again.

CSagd66Init Australian Geodetic Datum of 1966 INITialize

```
int CSagd66Init (void)
```

Use this function to initialize the AGD66GDA94 conversion system. The initialization will occur using the Geodetic Data Catalog file named by the contents of the `cs_Agd66Name` global variable. *CSagd66Init* returns zero on success and –1 on failure. Failure is usually caused by a problem encountered in processing the Geodetic Data Catalog, or the data files listed in the catalog file.

CSagd66ToGda94Log

```
Const char *CSagd66ToGda94Log (Const double ll_66 [2]);
```

Given a AGD66 based geographic point by the **ll_66** argument, *CSagd66ToGda94Log* returns a pointer to a static string which contains the name of the grid shift file which would be used to convert the point to GDA94. A null pointer is returned for any type of error. Errors are usually caused by a failure to initialize the AGD66 to GDA94 conversion system or if the provided point is not within the coverage of the AGD66 to GDA94 transformation system.

CS_agd66Name Australian Geodetic Datum of 1966 NAME

```
void CS_agd66Name (Const char *newName)
```

Use this function to change the file name portion of the Geodetic Data Catalog associated with the Australian Geodetic Datum of 1966 to the name provided by the **newName** argument.

CSagd66ToGda94 AGD66 TO GDA94 conversion

```
int CSagd66ToGda94 (double ll_94 [3],Const double ll_66 [3])
```

CSagd66ToGda94 will convert the geographic coordinate provided by the **ll_66** argument to the equivalent GDA94 argument and return the results in the array indicated by the **ll_94** argument. *CSagd66Init* must have been successfully called prior to using this function. This is (currently) a two dimensional function, the third element of each array is currently ignored.

CSagd66ToGda94 will return zero on success and a –1 for a hard failure. A hard failure is usually caused by a failure to successfully call *CSagd66Init* prior to calling *CSagd66ToGda94*. A positive 1 is returned in the case of a soft failure. A soft failure is caused by a AGD66 coordinate that is outside of the region covered by the data files listed in the Geodetic Data Catalog file used to initialize the transformation. A positive 2 is returned in the case of a coverage failure where a fallback datum definition is used to supply approximate results.

CSgda94ToAgd66 GDA94 TO AGD66 conversion

```
int CSgda94ToAgd66 (double ll_66 [3],Const double ll_94 [3])
```

CSgda94ToAgd66 will convert the geographic coordinate provided by the **ll_94** argument to the equivalent AGD66 geographic coordinates and return the results in the array indicated by the **ll_66** argument. *CSagd66Init* must have been successfully called prior to using this function. This is (currently) a two dimensional function, the third element of each array is currently ignored.

CSgda94ToAgd66 will return zero on success and a –1 for a hard failure. A hard failure is usually caused by a failure to successfully call *CSagd66Init* prior to calling *CSgda94ToAgd66*. A positive 1 is returned in the case of a soft failure. A soft failure us caused by a GDA94 coordinate that is outside of the region covered by the AGD66 GDA94 data files listed in the Geodetic Data Catalog file used to initialize the AGD66 system. A positive 2 value is returned if the point to be converted was outside the coverage of the data files available and the fallback datum was used to calculate approximate results.

Australian Geodetic Datum of 1984

This transformation converts geographic coordinates based on the Australian Geodetic Datum of 1984 to geographic coordinates based on the Geocentric Datum of Australia (1994). The conversion is based on grid shift files which are in the in the Canadian National Transformation, Version 2, format. The files which are to be used must be listed in the Geodetic Data Catalog named *Agd84ToGda94.gdc*.

Note that AGD84 applies to certain Australian states, while AGD66 applies to others. CS-MAP uses the Geodetic Data Catalog feature to select the appropriate file from the list provided. Therefore, to the end user AGD84 looks like a single entity.

CSagd84Cls Australian Geodetic Datum of 1984 CLoSe

```
void CSagd84Cls (void)
```

Use *CSagd84Cls* to release all resources allocated by a previous call to *CSagd84Init*. Obviously, subsequent calls to the *CSagd84ToGda94* or *CSgda94ToAgd84* will fail until such time that *CSagd84Init* is called again.

CSagd84Init Australian Geodetic Datum of 1984 INITialize

```
int CSagd84Init (void)
```

Use this function to initialize the AGD84GDA94 conversion system. The initialization will occur using the Geodetic Data Catalog file named by the contents of the `cs_Agd84Name` global variable. *CSagd84Init* returns zero on success and –1 on failure. Failure is usually caused by a problem encountered in processing the Geodetic Data Catalog, or the data files listed in the catalog file.

CSagd84ToGda94Log AGD66 TO GDA94 LOG

```
Const char *CSagd84ToGda94Log (Const double ll_84 [2]);
```

Given a AGD84 based geographic point by the **ll_84** argument, *CSagd84ToGda94Log* returns a pointer to a static string which contains the name of the grid shift file which would be used to convert the point to GDA94. A null pointer is returned for any type of error. Errors are usually caused by a failure to initialize the AGD84 to GDA94 conversion system or if the provided point is not within the coverage of the AGD84 to GDA94 transformation system.

CS_agd84Name Australian Geodetic Datum of 1984 NAME

```
void CS_agd84Name (Const char *newName)
```

Use this function to change the file name portion of the Geodetic Data Catalog associated with the Australian Geodetic Datum of 1984 to the name provided by the **newName** argument.

CSagd84ToGda94 AGD84 TO GDA94 conversion

```
int CSagd84ToGda94 (double ll_94 [3],Const double ll_84 [3])
```

CSagd84ToGda94 will convert the geographic coordinate provided by the **ll_84** argument to the equivalent GDA94 argument and return the results in the array indicated by the **ll_94** argument. *CSagd84Init* must have been successfully called prior to using this function. This is (currently) a two dimensional function, the third element of each array is currently ignored.

CSagd84ToGda94 will return zero on success and a –1 for a hard failure. A hard failure is usually caused by a failure to successfully call *CSagd84Init* prior to calling *CSagd84ToGda94*. A positive 1 is returned in the case of a soft failure. A soft failure us caused by an AGD84 coordinate that is outside of the region covered by the AGD84 data files listed in the Geodetic Data Catalog file used to initialize

the AGD84 system. A positive 2 is returned if the point provided by the **ll_84** argument is outside the coverage of the AGD84 to GDA94 transformation system and the fallback datum was used to produce approximate results.

CSgda94ToAgd84 GDA94 TO AGD84 conversion

```
int CSgda94ToAgd84 (double ll_84 [3],Const double ll_94 [3])
```

CSgda94ToAgd84 will convert the geographic coordinate provided by the **ll_94** argument to the equivalent AGD84 argument and return the results in the array indicated by the **ll_84** argument. *CSagd84Init* must have been successfully called prior to using this function. This is (currently) a two dimensional function, the third element of each array is currently ignored.

CSgda94ToAgd84 will return zero on success and a –1 for a hard failure. A hard failure is usually caused by a failure to successfully call *CSagd84Init* prior to calling *CSgda94ToAgd84*. A positive 1 is returned in the case of a soft failure. A soft failure us caused by a GDA94 coordinate that is outside of the region covered by the AGD84 GDA94 data files listed in the Geodetic Data Catalog file used to initialize the AGD84 system. A positive 2 is returned if the point provided by the **ll_94** argument is outside the coverage of the AGD84 to GDA94 transformation system and the fallback datum was used to produce an approximate result.

Average Terrestrial System of 1977 (ATS77)

This transformation converts geographic coordinates based on the Average Terrestrial System of 1977 to geographic coordinates based on the Canadian Spatial Reference System (Nad83/1994). This transformation is typically used in the maritime provinces of Canada. The conversion is based on grid shift files which are in the in the Canadian National Transformation, Version 2, format. The files which are to be used must be listed in the Geodetic Data Catalog named *Ats77ToCSRS.gdc*.

Note that ATS77 applies only to the maritime provinces of New Brunswick, Nova Scotia, and Prince Edward Island. CS-MAP uses the Geodetic Data Catalog feature to select the appropriate file from the list provided. Therefore, to the end user ATS77 looks like a single entity. Take great care in ordering the files in the Geodetic Data Catalog, as there is serious overlap, and the results derived from the different files can differ significantly. In fact, it is recommended that if a client is based in one of these provinces, that they delete (or comment out) the other two files from the Geodetic Data Catalog to assure that the desired results are obtained.

CSats77Cls Average Terrestial System of 1977 CLoSe

```
void CSats77Cls (void)
```

Use *CSats77Cls* to release all resources allocated by a previous call to *CSats77Init*. Obviously, subsequent calls to the *CSats77ToCsrs* or *CScsrsToAts77* will fail until such time that *CSats77Init* is called again.

CSats77Init Average Terrestial System of 1977 INITialize

```
int CSats77Init (void)
```

Use this function to initialize the ATS77CSRS conversion system. The initialization will occur using the Geodetic Data Catalog file named by the contents of the `cs_Ats77Name` global variable. *CSats77Init* returns zero on success and –1 on failure. Failure is usually caused by a problem encountered in processing the Geodetic Data Catalog, or the data files listed in the catalog file.

CS_ats77Name Average Terrestrial System of 1977 NAME

```
void CS_ats77Name (Const char *newName)
```

Use this function to change the file name portion of the Geodetic Data Catalog associated with the Average Terrestrial System (eastern Canada) of 1977 to the name provided by the **newName** argument.

CSats77ToCsrsLog ATS77 TO CSRS LOG function

```
Const char * EXP_LVL7 CSats77ToCsrsLog (Const double ll_77 [2])
```

Use this function to obtain a pointer to a string which contains the name of the file (from the Geodetic Data Catalog) which would be used to convert the point provided by the **ll_77** argument. A null pointer is returned if the ATS77 to CSRS conversion system has not been initialized, or the point provided is not within the coverage of the ATS77 to CSRS conversion system.

CSats77ToCsrs ATS77 TO CSRS conversion

```
int CSats77ToCsrs (double ll_csrs [3],Const double ll_77 [3])
```

CSats77ToCsrs will convert the geographic coordinate provided by the **ll_ 77** argument to the equivalent CSRS (Canadian Spatial Reference System) values and return the results in the array indicated by the **ll_csrs** argument. *CSats77Init* must have been successfully called prior to using this function. This is (currently) a two dimensional function, the third element of each array is currently ignored.

CSats77ToCsrs will return zero on success and a –1 for a hard failure. A hard failure is usually caused by a failure to successfully call *CSats77Init* prior to calling *CSats77ToCsrs*. A positive 1 is returned in the case of a soft failure. A soft failure us caused by an ATS77 coordinate that is outside of the region covered by the data files listed in the Geodetic Data Catalog file used to initialize the ATS77 system.

CScsrsToAts77 CSRS TO ATS77 conversion

```
int CscsrsToAts77 (double ll_77 [3],Const double ll_csrs [3])
```

CScsrsToAts77 will convert the geographic coordinate provided by the **ll_csrs** argument to the equivalent ATS77 values and return the results in the array indicated by the **ll_77** argument. *CSats77Init* must have been successfully called prior to using this function. This is (currently) a two dimensional function, the third element of each array is currently ignored.

CScsrsToAts77 will return zero on success and a –1 for a hard failure. A hard failure is usually caused by a failure to successfully call *CSats77Init* prior to calling *CScsrsToAts77*. A positive 1 is returned in the case of a soft failure. A soft failure us caused by an ATS77 coordinate that is outside of the region covered by the ATS77 CSRS data files listed in the Geodetic Data Catalog file used to initialize the ATS77 system.

Canadian Spatial Reference System

Use this transformation to convert coordinates based on the North American Datum of 1983 (NAD83) to coordinates based on the Canadian Spatial Reference System. This transformation is based on datum shift data files in the Canadian National Transformation, Version 2, format which are listed in the in the Geodetic Data Catalog named *Nad83ToCsrs.gdc*.

CScsrsCls Canadian Spatial Reference System CLoSe

```
void CSnadCls (void)
```

Use *CsnadCls* to release all resources allocated by a previous call to *CScsrsInit*. Obviously, subsequent calls to the *CSnad83ToCsrs* or *CScsrsToNad83* will fail until such time that *CScsrsInit* is called again.

CScsrsInit Canadian Spatial Reference System INITialize

```
int CScsrsInit (void)
```

Use this function to initialize the NAD83CSRS conversion system. The initialization will occur using the Geodetic Data Catalog file named by the contents of the `cs_CsrsName` global variable. *CScsrsInit* returns zero on success and –1 on failure. Failure is usually caused by a problem encountered in processing the Geodetic Data Catalog, or the data files listed in the catalog file.

CS_csrsName Canadian Spatial Reference System NAME

```
void CS_csrsName (Const char *newName)
```

Use this function to change the file name portion of the Geodetic Data Catalog associated with the Canadian Spatial Reference System to the name provided by the **newName** argument.

CSnad83ToCsrsLog NAD83 TO CSRS LOG function

```
Const char* CSnad83ToCsrsLog (Const double ll_83 [2])
```

Use this function to obtain a pointer to a string which contains the name of the file (from the Geodetic Data Catalog) which would be used to convert the point provided by the **ll_83** argument. A null pointer is returned if the NAD83 to CSRS conversion system has not been initialized, or the point provided is not within the coverage of the NAD83 to CSRS conversion system.

CSnad83ToCsrs NAD83 TO Canadian Spatial Reference System conversion

```
int CSnad83ToCsrs (double ll_csrs [3],Const double ll_83 [3])
```

CSnad83ToCsrs will convert the geographic coordinate provided by the **ll_83** argument to the equivalent CSRS values and return the results in the array indicated by the **ll_csrs** argument. *CScsrsInit* must have been successfully called prior to using this function. This is (currently) a two dimensional function, the third element of each array is currently ignored.

CSnad83ToCsrs will return zero on success and a –1 for a hard failure. A hard failure is usually caused by a failure to successfully call *CScsrsInit* prior to calling *CSnad83ToCsrs*. A positive 1 is returned in the case of a soft failure. A soft failure us caused by a NAD83 coordinate that is outside of the region covered by the CSRS data files listed in the Geodetic Data Catalog file used to initialize the CSRS system.

CScsrsToNad83 Canadian Spatial Reference System TO NAD83 conversion

```
int CScsrsToNad83 (double ll_83 [3],Const double ll_csrs [3])
```

CScsrsToNad83 will convert the geographic coordinate provided by the **ll_csrs** argument to the equivalent NAD83 values and return the results in the array indicated by the **ll_83** argument. *CScsrsInit* must have been successfully called prior to using this function. This is (currently) a two dimensional function, the third element of each array is currently ignored.

CScsrsToHarn will return zero on success and a –1 for a hard failure. A hard failure is usually caused by a failure to successfully call *CScsrsInit* prior to calling *CScsrsToNad83*. A positive 1 is returned in the case of a soft failure. A soft failure us caused by a CSRS coordinate that is outside of the region covered by the HARN data files listed in the Geodetic Data Catalog file used to initialize the HARN system.

High Precision Gps Network

Use this transformation to convert coordinates based on the North American Datum of 1983 (NAD83) to coordinates based on the High Accuracy Reference Network (HARN). In the past, this has been referred to as High Precision GPS Network (HPGN) and NAD83/91. This transformation is based on datum shift data files in the NADCON LAS/LOS format which are listed in the in the Geodetic Data Catalog named *Nad83ToHarn.gdc*.

These functions are defined in the module named *CS_hpgn.c*.

CSharnCls HARN CLoSe
```
void CSharnCls (void)
```

Use *CSharnCls* to release all resources allocated by a previous call to *CSharnInit*. Obviously, subsequent calls to the *CSnad83ToHarn* or *CSharnToNad83* will fail until such time that *CSharnInit* is called again.

CSharnInit HARN INITialize
```
int CSharnInit (void)
```

Use this function to initialize the NAD83HARN conversion system. The initialization will occur using the Geodetic Data Catalog file named by the contents of the `cs_HarnName` global variable. *CSharnInit* returns zero on success and –1 on failure. Failure is usually caused by a problem encountered in processing the Geodetic Data Catalog, or the data files listed in the catalog file.

CSnad83ToHarnLog NAD83 TO HARN LOG
```
Const char* CSnad83ToHarnLog (Const double ll_83 [3])
```

Use this function to obtain a pointer to a string which contains the name of the file (from the Geodetic Data Catalog) which would be used to convert the point provided by the **ll_83** argument. A null pointer is returned if the NAD83 to HARN conversion system has not been initialized, or the point provided is not within the coverage of the NAD83 to HARN conversion system.

CSharnToNad83 HARN TO NAD83 conversion
```
int CSharnToNad83 (double ll_83 [3],Const double ll_harn [3])
```

CSharnToNad83 will convert the geographic coordinate provided by the **ll_harn** argument to the equivalent NAD83 values and return the results in the array indicated by the **ll_83** argument. *CSharnInit* must have been successfully called prior to using this function. This is (currently) a two dimensional function, the third element of each array is currently ignored.

CSharnToNad83 will return zero on success and a –1 for a hard failure. A hard failure is usually caused by a failure to successfully call *CSharnInit* prior to calling *CSharnToNad83*. A positive 1 is returned in the case of a soft failure. A soft failure us caused by a HARN coordinate that is outside of the region covered by the HARN data files listed in the Geodetic Data Catalog file used to initialize the HARN system. A positive 2 is returned in the case of a coverage failure where a fallback datum definition is used to supply approximate results.

CSnad83ToHarn NAD83 TO HARN conversion
```
int CSnad83ToHarn (double ll_harn [3],Const double ll_83 [3])
```

CSnad83ToHarn will convert the geographic coordinate provided by the **ll_83** argument to the equivalent HARN argument and return the results in the array indicated by the **ll_harn** argument.

CSharnInit must have been successfully called prior to using this function. This is (currently) a two dimensional function, the third element of each array is currently ignored.

CSnad83ToHarn will return zero on success and a –1 for a hard failure. A hard failure is usually caused by a failure to successfully call *CSharnInit* prior to calling *CSnad83ToHarn*. A positive 1 is returned in the case of a soft failure. A soft failure us caused by a NAD83 coordinate that is outside of the region covered by the HARN data files listed in the Geodetic Data Catalog file used to initialize the HARN system. A positive 2 is returned in the case of a coverage failure where a fallback datum definition is used to supply approximate results.

North American Datum of 1983

This transformation converts geographic coordinates based on the North American Datum of 1927 to geographic coordinates based on the North American Datum of 1983. Two different file formats are supported:

1 NADCON LAS/LOS format files as generated and distributed by the US National Geodetic Survey (NGS); and

2 Canadian National Transformation, Version 2, format files as distributed by Geomatics Canada.

The files which are to be used must be listed in the Geodetic Data Catalog named *Nad27ToNad83.gdc*.

Clearly, the files published by the US NGS apply to US geography (i.e. the 48 conterminous states, Alaska, Hawaii, etc.), and the file(s) distributed by Geomatics Canada apply to Canadian geography. Through the use of the Geodetic Data Catalog feature, CS-MAP presents NAD83 to the user as a single entity.

Note that there is substantial overlap between the data files supplied by these two different sources. The files will not produce identical results. Therefore it is important that the features of the Geodetic Data Catalog be used to obtain the results desired.

CSnadCls NADcon CLoSe

```
void CSnadCls (void)
```

Use *CsnadCls* to release all resources allocated by a previous call to *CSnadInit*. Obviously, subsequent calls to the *CSnad27ToNad83* or *CSnad83ToNad27* will fail until such time that *CSnadInit* is called again.

CSnadInit NADcon INITialize

```
int CSnadInit (void)
```

Use this function to initialize the NAD27NAD83 conversion system. The initialization will occur using the Geodetic Data Catalog file named by the contents of the `cs_NadName` global variable. *CSnadInit* returns zero on success and –1 on failure. Failure is usually caused by a problem encountered in processing the Geodetic Data Catalog, or the data files listed in the catalog file.

CSnad27ToNad83Log NAD27 TO NAD83 LOG

```
Const char* CSnad27ToNad83Log (Const double ll_27 [2])
```

Given a NAD27 based geographic point by the **ll_27** argument, *CSnad27ToNad83Log* returns a pointer to a static string which contains the name of the grid shift file which would be used to convert the point to NAD83. A null pointer is returned for any type of error. Errors are usually caused by a failure to initialize the NAD27 to NAD83 conversion system or if the provided point is not within the coverage of the NAD27 to NAD83 transformation system.

CSnad27ToNad83 NAD27 TO NAD83 conversion

```
int CSnad27ToNad83 (double ll_83 [3],Const double ll_27 [3])
```

CSnad27ToNad83 will convert the geographic coordinate provided by the **ll_27** argument to the equivalent NAD83 argument and return the results in the array indicated by the **ll_83** argument. *CSnadInit* must have been successfully called prior to using this function. This is (currently) a two dimensional function, the third element of each array is currently ignored.

CSnad27ToNad83 will return zero on success and a –1 for a hard failure. A hard failure is usually caused by a failure to successfully call *CSnadInit* prior to calling *CSnad27ToNad83*. A positive 1 is returned in the case of a soft failure. A soft failure us caused by a NAD27 coordinate that is outside of the region covered by the NADCON data files listed in the Geodetic Data Catalog file used to initialize the NADCON system. A positive 2 is returned if the point provided by the **ll_27** argument is outside the coverage of the NAD27 to NAD83 transformation system and the fallback datum was used to produce approximate results.

CSnad83ToNad27 NAD83 TO NAD27 conversion

```
int CSnad83ToNad27 (double ll_27 [3],Const double ll_83 [3])
```

CSnad83ToNad27 will convert the geographic coordinate provided by the **ll_83** argument to the equivalent NAD27 argument and return the results in the array indicated by the **ll_27** argument. *CSnadInit* must have been successfully called prior to using this function. This is (currently) a two dimensional function, the third element of each array is currently ignored.

CSnad83ToNad27 will return zero on success and a –1 for a hard failure. A hard failure is usually caused by a failure to successfully call *CSnadInit* prior to calling *CSnad83ToNad27*. A positive 1 is returned in the case of a soft failure. A soft failure us caused by a NAD83 coordinate that is outside of the region covered by the NADCON data files listed in the Geodetic Data Catalog file used to initialize the NADCON system. A positive 2 is returned if the point provided by the **ll_83** argument is outside the coverage of the NAD27 to NAD83 transformation system and the fallback datum was used to produce approximate results.

New Zealand Geodetic Datum of 1949

This transformation converts geographic coordinates based on the New Zealand Geodetic Datum of 1949 to geographic coordinates based on the New Zealand Geodetic Datum of 2000. The conversion is based on grid shift files which are in the in the Canadian National Transformation, Version 2, format. The files which are to be used must be listed in the Geodetic Data Catalog named *Nzgd49ToNzgd2k.gdc*.

This set oif functions relies on the Geodetic Data Catalog concept, even though there is a single file, and it is unlikely that there would ever be more than one file.

CSnzgd49Cls New Zealand Geodetic Datum of 1949 CLoSe

```
void CSnzgd49Cls (void)
```

Use *CSnzgd49Cls* to release all resources allocated by a previous call to *CSnzgd49Init*. Obviously,

subsequent calls to the *CSnzgd49ToNzgd2K* or *CSnzgd2KToNzgd49* will fail until such time that *CSnzgd49Init* is called again.

CSnzgd49Init New Zealand Geodetic Datum of 1949 INITialize

```
int CSnzgd49Init (void)
```

Use this function to initialize the NZGD49NZGD2000 conversion system. The initialization will occur using the Geodetic Data Catalog file named by the contents of the `cs_Nzgd49Name` global variable. *CSnzgd49Init* returns zero on success and –1 on failure. Failure is usually caused by a problem encountered in processing the Geodetic Data Catalog, or the data files listed in the catalog file.

CS_nzgd49Name New Zealand Geodetic Datum of 1949 NAME

```
void CS_nzgd49Name (Const char *newName)
```

Use this function to change the file name portion of the Geodetic Data Catalog associated with the New Zealand Geodetic Datum of 1949 to the name provided by the **newName** argument.

CSnzgd2KToNzgd49 NZGD2K TO NZGD49 conversion

```
int CSnzgd2KToNzgd49 (double ll_49 [3],Const double ll_2k [3])
```

CSnzgd2KToNzgd49 will convert the geographic coordinate provided by the **ll_2k** argument to the equivalent NZGD49 argument and return the results in the array indicated by the **ll_49** argument. *CSnzgd49Init* must have been successfully called prior to using this function. This is (currently) a two dimensional function, the third element of each array is currently ignored.

CSnzgd2KToNzgd49 will return zero on success and a –1 for a hard failure. A hard failure is usually caused by a failure to successfully call *CSnzgd49Init* prior to calling *CSnzgd2KToNzgd49*. A positive 1 is returned in the case of a soft failure. A soft failure us caused by a NZGD2000 coordinate that is outside of the region covered by the NZGD49 NZGD2000 data files listed in the Geodetic Data Catalog file used to initialize the NZGD49 system. A positive 2 is returned in the case of a coverage failure where a fallback datum definition is used to supply approximate results.

CSnzgd49ToNzgd2K NZGD49 TO NZGD2K conversion

```
int CSnzgd49ToNzgd2K (double ll_2k [3],Const double ll_49 [3])
```

CSnzgd49ToNzgd2K will convert the geographic coordinate provided by the **ll_49** argument to the equivalent NZGD2000 values and return the results in the array indicated by the **ll_2k** argument. *CSnzgd49Init* must have been successfully called prior to using this function. This is (currently) a two dimensional function, the third element of each array is currently ignored.

CSnzgd49ToNzgd2K will return zero on success and a –1 for a hard failure. A hard failure is usually caused by a failure to successfully call *CSnzgd49Init* prior to calling *CSnzgd49ToNzgd2K*. A positive 1 is returned in the case of a soft failure. A soft failure us caused by a NZGD49 coordinate that is outside of the region covered by the NZGD49 data files listed in the Geodetic Data Catalog file used to initialize the NZGD49 system. A positive 2 is returned in the case of a coverage failure where a fallback datum definition is used to supply approximate results.

World Geodetic System of 1972

This transformation converts geographic coordinates based on World Geodetic System of 1972 (WGS72) to geographic coordinates based on the World Geodetic System of 1984 (WGS84). This transformation is analytical. That is, some simple formulas are available and no data files (or Geodetic Data Catalog files) are necessary.

CSwgs72284 WGS72 TO wgs84

```
int CSwgs72284 (Const double ll_72 [2],double ll_84 [2]);
```

CSwgs72284 will convert the WGS-72 based latitude and longitude provided in the **ll_72** array to their equivalent WGS-84 values and return the result in the **ll_84** array. **ll_72** and **ll_84** may be the same array. In these arrays, the first element is the longitude and the second element is the latitude. Longitude and latitudes must be given in degrees where negative values indicate west longitude and south latitude. *CSwgs72284* returns **FALSE** to indicate a successful conversion.

ERRORS

CSwgs72284 returns **FALSE** to indicate a successful conversion. Future versions of *CSwgs72284* may return **TRUE** to indicate a failure of some sort.

CSwgs8472 WGS84 to wgs72

```
int CSwgs8472 (Const double ll_84 [2],double ll_72 [2]);
```

CSwgs8472 iteratively calls *CSwgs72284* in order to convert the WGS-84 geographic coordinates given by the **ll_84** argument to an equivalent pair of WGS-72 coordinates. The results are returned in the **ll_72** array. **ll_84** and **ll_72** may be the same array. In both arrays, the first element is the longitude and the second element is the latitude. Values must be in degrees where negative values are used to indicate west longitude and south latitude. *CSwgs8472* returns **FALSE** to indicate a successful conversion.

ERRORS

CSwgs8472 returns **TRUE** and sets `cs_Error` appropriately if any of the following conditions are encountered during the calculations:

cs_WGS_CNVRG	The iterative calculation of the WGS-72 values failed to converge after six iterations.

North American Datum of 1983

For most practical applications, the North American Datum of 1983 (NAD83) and the World Geodetic System of 1984 (WGS84) are the same. These are two very precise measurements of the same thing by two different organizations using similar, but not identical, techniques. Thus, there is significant debate as whether there are any significant differences between the two.

CS-MAP considers NAD83 and WGS84 to be the same. this is true for the reason given above, and also because we have not located (as yet) a transformation which can be used to convert between these two. (The absence of such a published conversion leads us to believe that everyone else also considers these two datums to be the same.)

However, in order to make it easy for an application programmer to institute a transformation for this conversion, two stub functions are incorporated into CS-MAP which are called whenever a transformation would be appropriate.

CSnad8384 NAD38 to wgs84

```
int CSnad8384 (Const double ll_83 [2],double ll_84 [2]);
```

Currently, *CSnad8384* simply copies the contents of the **ll_83** array to the **ll_84** array and returns **FALSE** to indicate that it did so successfully.

There are differences between NAD83 and WGS 84. However, both are very accurate measurements of the same thing. Therefore, the differences are slight and are within the tolerance of error associated with WGS-84. Currently, there are no generally accepted techniques for converting one to the other. This function is a hook to provide such conversion should a generally recognized technique become available in the future.

ERRORS

CSnad8384 will **FALSE** to indicate success. Future versions of *CSnad8384* may return a **TRUE** value to indicate an error of some sort.

CSwgs8483 WGS84 to nad 83

```
int CSwgs8483 (Const double ll_84 [2],double ll_83 [2]);
```

Currently, *CSwgs8383* simply copies the contents of the **ll_84** array to the **ll_83** array and returns **FALSE** to indicate that it did so successfully.

There are differences between NAD83 and WGS 84. However, both are very accurate measurements of the same thing. Therefore, the differences are slight and are within the tolerance of error associated with WGS-84. Currently, there are no generally accepted techniques for converting one to the other. This function is a hook to provide such conversion should a generally recognized technique become available in the future.

ERRORS

CSwgs8483 will **FALSE** to indicate success. Future versions of *CSwgs8483* may return a **TRUE** value to indicate an error of some sort.

Geocentric Datum of Australia, 1994

CSgda94ToWgs84 Gda94 to WGS84

```
int CSgda94ToWgs84 (Const double ll_wgs84 [3],double ll_gda94 [3]);
```

Currently, *CSgda94ToWgs84* simply copies the contents of the **ll_gda94** array to the **ll_wgs84** array and returns **FALSE** to indicate that it did so successfully.

There are differences between GDA94 and WGS84. However, the differences are slight, and considering them to be the same produces acceptable results for most practical applications. This function is provided as a hook for those who wish to implement a transformation for this conversion.

ERRORS

CSgda94ToWgs84 will **FALSE** to indicate success. Future versions of *CSgda94ToWgs84* may return a **TRUE** value to indicate an error of some sort.

CSwgs84ToGda94 WGS84 To GDA94

```
int CSwgs84ToGda94 (Const double ll_gda94 [3],double ll_wgs84 [3]);
```

Currently, *CSwgs84ToGda94* simply copies the contents of the **ll_wgs84** array to the **ll_gda94** array and returns **FALSE** to indicate that it did so successfully.

There are differences between GDA94 and WGS84. However, the differences are slight, and considering them to be the same produces results acceptable for most practical applications. This function is provided as a hook for those who wish to implement a transformation for this conversion.

ERRORS

CSwgs84ToGda94 will **FALSE** to indicate success. Future versions of *CSwgs84ToGda94* may return a **TRUE** value to indicate an error of some sort.

Geodetic Datum of New Zealand, 2000

CSnzgd

```
int CSnzgd2KToWgs84 (Const double ll_wgs84 [3],Const double ll_nzgd2k [3]);
```

Currently, *CSnzgd2kToWgs84* simply copies the contents of the **ll_nzgd2k** array to the **ll_wgs84** array and returns **FALSE** to indicate that it did so successfully.

There are differences between NZGD 2000 and WGS84. However, the differences are slight, and considering them to be the same produces acceptable results for most practical applications. This function is provided as a hook for those who wish to implement a transformation for this conversion.

Note, the 'K' in this function name is an uppercase 'K'.

ERRORS

CSnzgd2KToWgs84 will **FALSE** to indicate success. Future versions of *CSnzgd2KToWgs84* may return a **TRUE** value to indicate an error of some sort.

CSwgs84ToNzgd2k WGS84 to NZGD2000

```
int CSwgs84Tonzgd2K (Const double ll_nzgd2k [3],Const double ll_wgs84 [3]);
```

Currently, *CSwgs84Tonzgd2K* simply copies the contents of the **ll_wgs84** array to the **ll_nzgd2k** array and returns **FALSE** to indicate that it did so successfully.

There are differences between NZGD 2000 and WGS84. However, the differences are slight, and considering them to be the same produces results acceptable for most practical applications. This function is provided as a hook for those who wish to implement a transformation for this conversion.

Note, the 'K' in this function name is an uppercase 'K'.

ERRORS

CSwgs84Tonzgd2K will **FALSE** to indicate success. Future versions of *CSwgs84Tonzgd2K* may return a **TRUE** value to indicate an error of some sort.

European Terrestrial Reference System of 1989

CSetrf89ToWgs84 ETRF89 To WGS84

```
int CSetrf89ToWgs84 (Const double ll_wgs84 [3],double ll_etrf89 [3]);
```

Currently, *CSetrf89ToWgs84* simply copies the contents of the **ll_etrf89** array to the **ll_wgs84** array and returns **FALSE** to indicate that it did so successfully.

There are differences between ETRF89 and WGS84. However, the differences are slight, and considering them to be the same produces acceptable results for most practical applications. This function is provided as a hook for those who wish to implement a transformation for this conversion.

ERRORS

CSetrf89ToWgs84 will **FALSE** to indicate success. Future versions of *CSetrf89ToWgs84* may return a **TRUE** value to indicate an error of some sort.

CSwgs84ToEtrf89 WGS84 To ETRF89

```
int CSwgs84ToEtrf89 (Const double ll_etrf89 [3],Const double ll_wgs84 [3]);
```

Currently, *CSwgs84ToEtrf89* simply copies the contents of the **ll_wgs84** array to the **ll_etrf89** array and returns **FALSE** to indicate that it did so successfully.

There are differences between ETRF89 and WGS84. However, the differences are slight, and considering them to be the same produces results acceptable for most practical applications. This function is provided as a hook for those who wish to implement a transformation for this conversion.

ERRORS

CSwgs84ToEtrf89 will **FALSE** to indicate success. Future versions of *CSwgs84ToEtrf89* may return a **TRUE** value to indicate an error of some sort.

European Datum of 1950

This transformation converts geographic coordinates based on the European Datum of 1950 to geographic coordinates based on the European Terrestrial Reference System of 1989. Currently, a single file format, the Canadian National Transformation Version 2, is supported. The files which are to be used must be listed in the Geodetic Data Catalog named *Ed50ToEtrf89.gdc*. As of this writing, only one data file is known to exist. That file covers Spain, and is not yet declared official.

Note that is very likely that there will be substantial overlap between the data files supplied by the various governmental agencies. Since these will be produced by different governments, it is even more likely that the results produce in the regions of overlap will be significantly different. Therefore it is important that the features of the Geodetic Data Catalog be used to obtain the results desired.

CSed50Cls Eurpean Datum of 1950 CLoSe

```
void CSed50Cls (void)
```

Use *CSed50Cls* to release all resources allocated by a previous call to *CSed50Init*. Obviously, subsequent calls to the *CSed50ToEtrf89* or *CSetrf89ToEd50* will fail until such time that *CSed50Init* is called again.

CSed50Init European Datum of 1950 INITialize

```
int CSed50Init (void)
```

Use this function to initialize the ED50 to ETRF89 conversion system. The initialization will occur using the Geodetic Data Catalog file named by the contents of the `cs_Ed50Name` global variable. *CSed50Init* returns zero on success and –1 on failure. Failure is usually caused by a problem encountered in processing the Geodetic Data Catalog, or the data files listed in the catalog file.

The default Geodetic Dataum Catalog file name is *Ed50ToEtrf89.gdc*.

CS_ed50Name European Datum of 1950 NAME

```
void CS_ed50Name (Const char *newName)
```

Use this function to change the file name portion of the Geodetic Data Catalog associated with the European Datum of 1950 to the name provided by the **newName** argument.

CSed50ToEtrf89 ED50 To ETRF89 conversion

```
int CSed50ToEtrf89 (double ll_etrf89 [3],Const double ll_ed50 [3])
```

CSed50ToEtrf89 will convert the geographic coordinate provided by the **ll_ed50** argument to the appropriate ETRF89 equivalent and return the results in the array indicated by the **ll_etrf89** argument. *CSed50Init* must have been successfully called prior to using this function. This is (currently) a two dimensional function, the third element of each array is currently ignored.

CSed50ToEtrf89 will return zero on success and a –1 for a hard failure. A hard failure is usually caused by a failure to successfully call *CSed50Init* prior to calling *CSed50ToEtrf89*. A positive 1 is returned in the case of a soft failure. A soft failure is caused by a ED50 coordinate that is outside of the region covered by the data files listed in the Geodetic Data Catalog file used to initialize the transformation. A positive 2 is returned in the case of a coverage failure where a fallback datum definition is used to supply approximate results.

CSetrf89ToEd50 ETRF89 To ED50 conversion

```
int CSetrf89ToEd50 (double ll_ed50 [3],Const double ll_etrf89 [3])
```

CSetrf89ToEd50 will convert the geographic coordinate provided by the **ll_etrf89** argument to the equivalent ED50 geographic coordinates and return the results in the array indicated by the **ll_ed50** argument. *CSed50Init* must have been successfully called prior to using this function. This is (currently) a two dimensional function, the third element of each array is currently ignored.

CSetrf89ToEd50 will return zero on success and a –1 for a hard failure. A hard failure is usually caused by a failure to successfully call *CSed50Init* prior to calling *CSetrf89ToEd50*. A positive 1 is returned in the case of a soft failure. A soft failure us caused by a ETRF89 coordinate that is outside of the region covered by the ED50 to ETRF89 data files listed in the Geodetic Data Catalog file used to initialize the ED50 system. A positive 2 value is returned if the point to be converted was outside the coverage of the data files available and the fallback datum was used to calculate approximate results.

Tokyo Datum

This transformation converts geographic coordinates based on the Tokyo Datum to geographic coordinates based on the Japanese Geodetic Datum of 2000. Currently, a single file format is supported, a text format devised by an agency of the Japanese government. It appears that there is a single large file which covers the entire nation of Japan. Due to the file format, it is easy to produce smaller regional files. Thus, it may be that you will have several files to work with, and a Geodetic Data Catalog has been implemented for this purpose. The files which are to be used must be listed in the Geodetic Data Catalog named *TokyoToJgd2k.gdc*.

Note that usually the naming convention of this group usually uses the name of the source datum (Tokyo in this case). Due to a rather strange sequence of events, the functions associated with this transformation got named for the target datum, Japanese Geodetic Datum of 2000 (JGD2K). While an inconvenience, it certainly does effect the quality of the numeric results.

Note that the data files are actually normal text files. Since there does not seem to be any guarantees as to the fixed length nature of the records in these files, the CS-MAP implementation converts the text file to a binary form on its first use. This enables random access to the data in the file without hogging up virtual memory space. Thus, the first time this transformation is used, you are liekly to experience a slight pause as the text to binarty conversion takes place. CS-MAP will always check the dates and times on the two files, and regenerate the binary version of the file when it appears appropriate.

CSjgd2kCls Japanese Geodetic Datum of 2000 CLoSe

```
void CSjgd2kCls (void)
```

Use *CSjgd2kCls* to release all resources allocated by a previous call to *CSjgd2kInit*. Obviously, subsequent calls to the *CSjgd2kToTokyo* or *CStokyoToJgd2k* will fail until such time that *CSjgd2kInit* is called again.

CSjgd2kInit

```
int CSjgd2kInit (void)
```

Use this function to initialize the Tokyo to JGD2K conversion system. The initialization will occur using the Geodetic Data Catalog file named by the contents of the `cs_Jgd2kName` global variable. *CSjgd2kInit* returns zero on success and –1 on failure. Failure is usually caused by a problem encountered in processing the Geodetic Data Catalog, or the data files listed in the catalog file.

CS_j

```
void CS_jgd2kName (Const char *newName)
```

Use this function to change the file name portion of the Geodetic Data Catalog associated with the Japanese Geodetic Name of 2000 to the name provided by the **newName** argument.

CStokyoToJgd2k TOKYO

```
int CStokyoToJgd2k (double ll_jgd2k [3],Const double ll_tokyo [3])
```

CStokyoToJgd2k will convert the geographic coordinate provided by the **ll_tokyo** argument to the equivalent JGD 2000 geographic coordinate and return the results in the array indicated by the **ll_jgd2k** argument. *CSjgd2kInit* must have been successfully called prior to using this function. This is (currently) a two dimensional function, the third element of each array is currently ignored.

CStokyoToJgd2k will return zero on success and a –1 for a hard failure. A hard failure is usually caused by a failure to successfully call *CSjgd2kInit* prior to calling *CStokyoToJgd2k*. A positive 1 is returned in the case of a soft failure. A soft failure is caused by a tokyo coordinate that is outside of the region covered by the data files listed in the Geodetic Data Catalog file used to initialize the transformation. A positive 2 is returned in the case of a coverage failure where a fallback datum definition is used to supply approximate results.

CSjgd2kToTokyo JGD2000 to TOKYO conversion

```
int CSjgd2kToTokyo (double ll_tokyo [3],Const double ll_jgd2k [3])
```

CSjgd2kToTokyo will convert the geographic coordinate provided by the **ll_jgd2k** argument to the equivalent tokyo geographic coordinates and return the results in the array indicated by the **ll_tokyo** argument. *CSjgd2kInit* must have been successfully called prior to using this function. This is (currently) a two dimensional function, the third element of each array is currently ignored.

CSjgd2kToTokyo will return zero on success and a –1 for a hard failure. A hard failure is usually caused by a failure to successfully call *CSjgd2kInit* prior to calling *CSjgd2kToTokyo*. A positive 1 is returned in the case of a soft failure. A soft failure us caused by a JGD 2000 coordinate that is outside of the region covered by the data files listed in the Geodetic Data Catalog file used to initialize the Tokyo system. A positive 2 value is returned if the point to be converted was outside the coverage of the data files available and the fallback datum was used to calculate approximate results.

Nouvelle Triangulation de la France (NTF)

This transformation transforms Réseau Géodésique Français of 1993 (RGF93) based coordinates to Nouvelle Triangulation de la France (NTF) based coordinates. This transformation is unique in four ways.

First, most transformations of this sort convert from the older datum to the newer datum. In this case, the conversion is actually the other way. That is, the algorithms and data files are designed to convert from the newer datum to the older datum. Since we are converting from RGF93 to NTF, the function naming uses `rgf93` as the base name.

Second, most transformations of this type use data files to carry the actual latitude and longitude shifts between the two datums (often in seconds of arc). In this case, the shifts are maintained in the data file in the form of geocentric cartesian coordinate translations in meters. That is, upon examining the data file, and performing some bi-linear interpolations, the result is the equivalent of a Three Parameter Transformation which is customized for each point in the domain of the transformation (i.e. coverage of the data file).

Third, the supplied data file is actually a normal text file. Therefore, for performance and reliability reasons, the CS-MAP implementation will read the entire data file into memory and organize the data in such a form as to maximize performance and minimize disk thrashing.

Fourth, since there is only one known data file, and the client for whom it was originally implemented was in a hurry, there is no Geodetic Data Catalog for this transformation. This is also due to the fact that many of the Geodetic Data Catalog features do not apply due to the rather unique nature of this transformation technique. So, there exists a single data file, it must be named *gr3df97a.txt*, and it must reside in the primary data directory (i.e. the directory set by *CS_altdr*). This is likely to change in the future.

Finally, note that the French produced a program which will perform these calculations which can be used to compare with CS-MAP results. In doing the inverse, the French program simply reversed the direction of the translation in the Three Parameter Transformation. This produces fully acceptable results in most cases, but does not produce the exact inverse of the forward calculation. The code in CS-MAP includes two functions for calculating the inverse. A version which will produce the exact inverse is commented out. The version which matches the results which are produced by the French program is active by default in the distribution. You pays your money, and you takes your choice. This transformation is implemented in a module named *CS_rgf93ToNtf.c*.

Bugs

In addition to not using a Geodetic Data Catalog, this set of functions uses hardcoded ellipsoid references to Clarke 1880 (actually, "CLRK-IGN") and GRS 1980. This hard coded references should, perhaps, be replaced with references in a datum definition somewhere.

CSrgf93Init RGF93 INITialize

```
int CSrgf93Init (void)
```

Use this function to initialize the Réseau Géodésique Français of 1993 (RGF93) to Nouvelle Triangulation de la France (NTF) conversion system. While likely to change in future releases, in Release 11 there is no Geodetic Data Catalog associated with this tranformation. There is a single data file, it must be named and it must reside in the primary data directory.

Since this is a text file, and thus the integrity of fix length records is questionable, this function will process the text file and produce a binary copy of it. This copying process is skipped if the date/time of the binary file is newer that the date/time of the text file. This, there will be a slight pause the first time this function is called.

CSrgf93Init returns zero on success and –1 on failure. Failure is usually caused by either not finding the data file in the primary data directory or a corrupted data file. (As the data file is a text file, corruption of the file is quite easily accomplished, even by accident.)

While perhaps not necessary after the first call (and the generation of the binary copy of the file, the text file must remain in the primary directory for the system to work. (Although, I must admit that, you can probably replace the text file with an empty file as long as the name and time stamp are appropriate.)

CSrgf93Cls Réseau Géodésique Français of 1993 CLoSe

```
void CSrgf93Cls (void)
```

Use *CSrgf93Cls* to release all resources allocated by a previous call to *CSrgf93Init*. Obviously, subsequent calls to the *CSrgf93ToNtf* or *CSntfToRgf93* will fail until such time that *CSrgf93Init* is called again.

CSrgf93ToNtf - RGF93 To NTF conversion

```
int CSrgf93ToNtf (double ll_ntf [3],Const double ll_rgf93 [3])
```

CSrgf93ToNtf will convert the geographic coordinate provided by the **ll_rgf93** argument to the appropriate Nouvelle Triangulation de la France (NTF) equivalent and return the results in the array indicated by the **ll_ntf** argument. *CSrgf93Init* must have been successfully called prior to using this function. This is (currently) a two dimensional function, the third element of each array is currently ignored.

CSrgf93ToNtf will return zero on success and a –1 for a hard failure. A hard failure is usually caused by a failure to successfully call *CSrgf93Init* prior to calling *CSrgf93ToNtf*. A positive 1 is returned in the case of a soft failure. A soft failure is caused by an Réseau Géodésique Français of 1993 (RGF93) coordinate that is outside of the region covered by the data file used in the transformation. (In future, at such time as a Geodetic Data Catalog is implemented, a positive 2 return value will be possible and will indicate that the fallback datum definition was used to obtain approximate results due to a coverage failure.)

CSntfToRgf93 NTF to RGF93 conversion

```
int CSntfToRgf93 (double ll_rgf93 [3],Const double ll_ntf [3])
```

CSntfToRgf93 will convert the geographic coordinate provided by the **ll_ntf** argument to the equivalent Réseau Géodésique Français of 1993 (RGF93) geographic coordinates and return the results in the array indicated by the **ll_rgf93** argument. *CSrgf93Init* must have been successfully called prior to using this function. This is (currently) a two dimensional function, the third element of each array is currently ignored.

CSntfToRgf93 will return zero on success and a –1 for a hard failure. A hard failure is usually caused by a failure to successfully call *CSrgf93Init* prior to calling *CSntfToRgf93*. A positive 1 is returned in the case of a soft failure. A soft failure us caused by a Nouvelle Triangulation de la France (NTF) coordinate that is outside of the region covered by the RGF93 to NTF data file used to in the transformation. (In future, at such time as a Geodetic Data Catalog is implemented, a positive 2 return value will be possible and will indicate that the fallback datum definition was used to obtain approximate results due to a coverage failure.)

Microsoft MFC User Dialog Functions

The functions described in this section provide the means by which programmers in the Windows MFC environment can display dialog boxes to accomplish common tasks with regard to using the CS-MAP library. Obviously, these functions are not of much value in an environment other than Windows 32.

While the functions themselves are declared with normal 'C' linkage, they must access MFC using "mangled" C++ linkage. For convenience, the C++ components of these functions are enclosed within a conditional compile based on the **__CPP__** preprocessor constant. Therefore, should you desire to have these functions compiled into your library, you will need to arrange for the **__CPP__** preprocessor constant to be defined.

CS_csDataDir Coordinate System DATA DIRectory dialog

```
int CS_csDataDir (void);
```

Use of this function is valid only in the Microsoft MFC environment.

This function presents the user with a data directory dialog and enables the user to specify the directory in which the dictionaries reside, and the file names assigned to the different dictionaries. Users who routinely develop new and different coordinate systems enjoy being able to maintain a "development" set of dictionaries separate from the "distribution" set.

Upon successful dismissal, this dialog will transfer the results of the dialog to the CS-MAP system through the use of the *CS_altdr*, *CS_csfnm*, *CS_dtfnm*, and *CS_elfnm* functions. The return value indicates the status of the check boxes included in the dialog. A value of zero indicates that neither box was checked. The "one" bit is set if the "Save in .INI file" box was checked. The "two" bit is set if the "Save in registry" box was checked.

BUGS

The function does not provide a means for disabling the check boxes for environments where they do not apply.

CS_csDualBrowser Coordinate System DUAL BROWSER

```
int CS_csBrowser (char *csKeyName);
```

Use of this function is valid only in the Microsoft MFC environment.

This function presents the user with an MFC based GUI dialog which facilitates the selection of source and target coordinate systems from the 1,000 or more contained in the Coordinate System Dictionary. The dialog is self-contained and requires little, if any, help from the application programmer.

The **csKeyName** argument to *CS_csBrowser* must be a pointer to a character array of not less than 24 characters (use **cs_KEYNM_DEF** from cs_map.h). The **csKeyName** argument is used to provide the source coordinate system key name that will be the initial content of the browser upon display. If the initial value of the argument is not that of a valid coordinate system key name, *CS_csBrowser* displays "LL" as a default.

CS_csBrowser will return **IDOK** (that's a Windows/MFC manifest constant) if the user dismissed the dialog box with a valid selection. In this case, the selected key name is returned in the character array pointed to by the argument. If the user dismisses the dialog box by any other means, **IDCANCEL** (another Windows/MFC manifest constant) is returned, and the character array pointed to by the argument remains unchanged.

The dialog contains a Help button that is grayed out if the module cannot locate the Help file. Normally, *CS_csBrowser* expects to find the Help file in the same directory as the coordinate system dictionary. You can use *CS_setHelpPath* to specify a different location and/or file. The help context integers used by this function are defined in *cs_hlp.h*.

CS_csBrowser Coordinate System BROWSER

```
int CS_csDualBrowser (char *srcKeyName,char *trgKeyName)
```

Use of this function is valid only in the Microsoft MFC environment.

This function presents the user with an MFC based GUI dialog which facilitates the selection of two coordinate systems from the 1,000 or more contained in the Coordinate System Dictionary. The dialog enables the use to select two coordinate systems, the first being labeled the "Source Coordinate Systems" and the second being labeled the "Target Coordinate System." The dialog is self-contained and requires little, if any, help from the application programmer.

The arguments to *CS_csDualBrowser* must be pointers to a character array of not less than 24 characters (use **cs_KEYNM_DEF** from *cs_map.h*). *CS_csDualBrowser* will use the value provided by the **srcKeyName** argument as the key name of the coordinate system definition that is to be displayed as the Source Coordinate System upon initial activation. Similarly, the value provided by the **trgKeyName** argument will be that which is displayed as the Target Coordinate System upon initial activation. If the initial value of **srcKeyName** or **trgKeyName** is not that of a valid coordinate system key name, *CS_csDualBrowser* displays "US48" and "LL27", respectively, as default values.

CS_csDualBrowser will return **IDOK** (that's a Windows/MFC manifest constant) if the user dismissed the dialog box with a valid selection. In this case, the key names as selected by the user are returned in the character arrays pointed to by the **srcKeyName** and **trgKeyName** arguments. If the user dismisses the dialog box by any other means, **IDCANCEL** (another Windows/MFC manifest constant) is returned, and the contents of the character array pointer to by arguments remain unchanged.

The dialog contains a Help button that is grayed out if the module cannot locate the Help file. Normally, *CS_csDualBrowser* expects to find the Help file in the same directory as the coordinate

system dictionary. You can use *CS_setHelpPath* to specify a different location and/or file. The help context integers used by this function are defined in *cs_hlp.h*.

CS_csEditor Coordinate System EDITOR dialog

```
void CS_csEditor (char *csKeyName);
```

Use of this function is valid only in the Microsoft MFC environment.

This function presents the user with an MFC based GUI dialog which enables the examination, modification, creation, and deletion of coordinate system definitions in the Coordinate System Dictionary. The dialog is self-contained and requires little (or no) interaction on the part of the application programmer.

The single argument to *CS_csEditor* must be a pointer to a character array of not less than 24 characters (use **cs_KEYNM_DEF** from cs_map.h). *CS_csEditor* will use the value of this character array as the key name of the coordinate system definition that is to be displayed upon initial activation. Upon dismissal, *CS_csEditor* will return in this array the key name of the coordinate system definition displayed upon exit. If the initial value of **csKeyName** is not that of a valid coordinate system key name, *CS_csEditor* displays "US48" as a default.

The dialog contains a Help button that is grayed out if the module cannot locate the Help file. Normally, *CS_csEditor* expects to find the Help file in the same directory as the coordinate system dictionary. You can use *CS_setHelpPath* to specify a different location and/or file. The help context integers used by this function are defined in *cs_hlp.h*.

CS_dtEditor DaTum EDITOR dialog

```
void CS_dtEditor (char *dtKeyName);
```

Use of this function is valid only in the Microsoft MFC environment.

This function presents the user with an MFC based GUI dialog which enables the examination, modification, creation, and deletion of datum definitions in the Datum Dictionary. The dialog is self-contained and requires little (or no) interaction on the part of the application programmer.

The single argument to *CS_dtEditor* must be a pointer to a character array of not less than 24 characters (use **cs_KEYNM_DEF** from cs_map.h). *CS_dtEditor* will use the value of this character array as the key name of the datum definition that is to be displayed upon initial activation. Upon dismissal, *CS_dtEditor* will return in this array the key name of the datum definition displayed upon exit. If the initial value of **dtKeyName** is not that of a valid datum key name, *CS_dtEditor* displays "WGS84" as a default.

The dialog contains a Help button that is grayed out if the module cannot locate the Help file. Normally, *CS_dtEditor* expects to find the Help file in the same directory as the coordinate system dictionary. You can use *CS_setHelpPath* to specify a different location and/or file. The help context integers used by this function are defined in cs_hlp.h.

CS_gdcEdit Geodetic Data Catalog EDITor

```
void CS_gdcEditor (char *gdcName);
```

Use this function to present to the user a dialog box which enables the examination and modification of any Geodetic Data Catalog currently known to the system. In this case, known to the system means any Geodetic Data Catalog file which resides in the general data directory established by a call to *CS_altdr*.

The **gdcName** argument to *CS_gdcEditor* must be a pointer to a character array of not less than 64 characters (use **cs_FNM_MAXLEN** from cs_map.h). The **gdcName** argument is used to provide the name of the Geodetic Data Catalog (sans extension) that will be the initial content of the editor upon display. If the initial value of the argument is not that of a valid Geodetic Data Catalog, *CS_gdcEditor* displays "Nad27ToNad83" as a default.

Upon dismisal from the dialog box, *CS_gdcEditor* will cause the character array pointed to by the single argument to contain the name of the last Geodetic Data Catalog display in the editor. It is advisable to call *CS_recvr* immediately before calkling this function, or after a return from this function, to cause the release of all geodetic objects which may rely on the Geodetic Data Catalogs edited by the user. This will cause a reconstruction of these objects using the possibly edited values in the Geodetic Data Catalog files.

The dialog contains a Help button that is grayed out if the module cannot locate the Help file. Normally, *CS_csBrowser* expects to find the Help file in the same directory as the coordinate system dictionary. You can use *CS_setHelpPath* to specify a different location and/or file. The help context integers used by this function are defined in *cs_hlp.h*.

CS_elEditor ELlipsoid EDITOR dialog

```
void CS_elEditor (char *elKeyName);
```

Use of this function is valid only in the Microsoft MFC environment.

This function presents the user with an MFC based GUI dialog which enables the examination, modification, creation, and deletion of ellipsoid definitions in the Ellipsoid Dictionary. The dialog is self-contained and requires little (or no) interaction on the part of the application programmer.

The single argument to *CS_elEditor* must be a pointer to a character array of not less than 24 characters (use **cs_KEYNM_DEF** from cs_map.h). *CS_elEditor* will use the value of this character array as the key name of the ellipsoid definition that is to be displayed upon initial activation. Upon dismissal, *CS_elEditor* will return in this array the key name of the datum definition displayed upon exit. If the initial value of **elKeyName** is not that of a valid datum key name, *CS_elEditor* displays "WGS84" as a default.

The dialog contains a Help button that is grayed out if the module cannot locate the Help file. Normally, *CS_elEditor* expects to find the Help file in the same directory as the coordinate system dictionary. You can use *CS_setHelpPath* to specify a different location and/or file. The help context integers used by this function are defined in cs_hlp.h.

CS_csTest Coordinate System TEST dialog

```
void CS_csTest (char *srcSystem,char *trgSystem,double srcXYZ [3]);
```

Use of this function is valid only in the Microsoft MFC environment.

This function presents the user with an MFC based GUI dialog which enables the conversion of single keyboard coordinates from any coordinate system to another. The original purpose of this dialog was to facilitate testing, but has been found to very useful for many other purposes.

The arguments to *CS_csTest* determine the initial presentation of the dialog. Use the **srcSystem** argument to specify the initial source coordinate system key name and the **trgSystem** argument to specify the initial target coordinate system key name. Note that upon dismissal of the dialog, the last user specified values are returned in these arrays. Therefore, these arguments must point to character arrays that are not less than 24 characters in length. Use the **cs_KEYNM_DEF** manifest constant from cs_map.h to define these arrays. *CS_csTest* uses the **srcXYZ** argument to initialize the source coordinate values that are to be converted. Again, upon dismissal of the dialog, the last values entered by the user for the source system coordinate values are returned in this array; therefore it must be dimensioned at 3 (or more).

The dialog contains a Help button that is grayed out if the module cannot locate the Help file. Normally, *CS_csTest* expects to find the Help file in the same directory as the coordinate system dictionary. You can use *CS_setHelpPath* to specify a different location and/or file. The help context integers used by this function are defined in *cs_hlp.h*.

CS_mgTest Military Grid TEST dialog

```
void CS_mgTest (char *elKeyName);
void CS_mgTestA (char *elKeyName,int* prec,long* latFrmt,long* lngFrmt);
```

Both of these functions cause presentation of the MGRS Test/Calculation dialog box to the user.

In the case of *CS_mgTest*, the ellipsoid selection is initially set to that provided by the **elKeyName** argument and all other values in the dialog are defaulted to hard coded values (i.e. zeros).

In the case of *CS_mgTestA*, initial values for precision, format of the display of latitude values, and the format of the display of longitude values are proviuded by the **prec**, **latFrmt**, and **lngFrmt** arguments respectively.

At such time that the user dismisses the dialog box, *CS_mgTest* and *CS_mgTestA* will cause the key name of the last selected ellipsoid to be copied to the character array pointed to by the **elKeyName** arguemnt. *CS_mgTestA* will also cause the **prec**, **latFrmt**, and **lngFrmt** values to be updated to the last values used in the dialog.

Users specify a format for the longitude and/or latitude display by entering a longitude and/or latitude in the appropriate fields in the format which they desire to see the results displayed. This may be an extra step for the user, but by remembering these settings as is possible with the A version of this function, it does not need to be repeated (if the application remembers these values, of course).

The dialog contains a Help button that is grayed out if the module cannot locate the Help file. Normally, *CS_csDualBrowser* expects to find the Help file in the same directory as the coordinate system dictionary. You can use *CS_setHelpPath* to specify a different location and/or file. The help context integers used by this function are defined in *cs_hlp.h*.

CS_dtSelector DaTum SELECTOR

```
int CS_dtSelector (char *dtKeyName);
```

Call this function to present the operator with an MFC based dialog box which can be used to select a specific datum from the Datum Dictionary. The **dtKeyName** argument must point to a character array of not less than 24 characters (use **cs_KEYNM_DEF** from cs_map.h). The key name contained in the character array pointed to by the single argument will be presented to the user as the initial selection. If this value is not that of a datum definition in the dictionary, "WGS84" is displayed as a default.

CS_dtSelector will return **IDOK** (that's a Windows/MFC manifest constant) if the user dismissed the dialog box with a valid selection. In this case, the selected key name is returned in the character array pointed to by the argument. If the user dismisses the dialog box by any other means, **IDCANCEL** (another Windows/MFC manifest constant) is returned, and the character array pointed to by the argument remains unchanged.

The dialog contains a Help button that is grayed out if the module cannot locate the Help file. Normally, *CS_dtSelector* expects to find the Help file in the same directory as the coordinate system dictionary. You can use *CS_setHelpPath* to specify a different location and/or file. The help context integers used by this function are defined in *cs_hlp.h*.

CS_elSelector ELipsoid SELECTOR

```
int CS_elSelector (char *elKeyName);
```

Call this function to present the operator with an MFC based dialog box which can be used to select a specific ellipsoid from the Ellipsoid Dictionary. The **elKeyName** argument must point to a character array of not less than 24 characters (use **cs_KEYNM_DEF** from cs_map.h). The key name contained in the character array pointed to by the single argument will be presented to the user as the initial selection. If this value is not that of a ellipsoid definition in the dictionary, "WGS84" is displayed as a default.

CS_elSelector will return **IDOK** (that's a Windows/MFC manifest constant) if the user dismissed the dialog box with a valid selection. In this case, the selected key name is returned in the character array pointed to by the argument. If the user dismisses the dialog box by any other means, **IDCANCEL** (another Windows/MFC manifest constant) is returned, and the character array pointed to by the argument remains unchanged.

The dialog contains a Help button that is grayed out if the module cannot locate the Help file. Normally, *CS_elSelector* expects to find the Help file in the same directory as the coordinate system dictionary. You can use *CS_setHelpPath* to specify a different location and/or file. The help context integers used by this function are defined in *cs_hlp.h*.

CSwinhlp WINdows HeLP

```
int CSwinhlp (void *hWnd,unsigned long context);
```

Use *CSwinhlp* to activate the Windows help program displaying the CS-MAP help file, cs-map.hlp. The **hWnd** argument is the handle to the window that is requesting the help. The **context** argument is used to select the specific article of help in the help file, see cs_hlp.h.

ERRORS

CSwinhlp returns non-zero to indicate success, i.e. at least the WinHelp.exe executable and the cs-map.hlp file were found and successfully started. *CSwinhlp* returns zero if a help screen could not be displayed.

General Support Functions

This section contains descriptions of functions of a general support nature. These are used quite liberally inside of CS-MAP. Your application may find some of these useful.

CS_adj1pi ADJust angle to 2 PI

```
double CS_adj1pi (double az);
```

The **az** argument is assumed to be an angle in radians. *CS_adj1pi* returns **az** after normalizing it to be greater than or equal to -/2 and less than or equal to /2.

CS_adj180 ADJust angle to 180 degrees

```
double CS_adj180 (double az);
```

The **az** argument is assumed to be an angle in degrees. *CS_adj180* returns **az** after normalizing it to be greater than -180 and less than or equal to 180.

CS_adj270 ADJust angle to 270 degrees

```
double CS_adj270 (double az);
```

The **az** argument is assumed to be an angle in degrees. *CS_adj270* returns **az** after normalizing it to be greater than -270 and less than or equal to +270.

CS_adj2pi ADJust angle to 2 PI

```
double CS_adj2pi (double az);
```

The **az** argument is assumed to be an angle in radians. *CS_adj2pi* returns **az** after normalizing it to be greater than - and less than or equal to .

CS_adj2piI ADJust angle to 2 PI Inclusive

```
double CS_adj2piI (double az);
```

The **az** argument is assumed to be an angle in radians. *CS_adj2piI* returns **az** after normalizing it to be greater than or equal to - and less than or equal to .

CS_adj90 ADJust angle to 90 degrees

```
double CS_adj90 (double az);
```

The **az** argument is assumed to be an angle in degrees. *CS_adj90* returns **az** after normalizing it to be greater than -90 and less than or equal to 90.

CS_adjll ADJust Lat/Long

```
void CS_adjll (double ll [2]);
```

CS_adjll adjusts the longitude in the array provided by the **ll** argument (i.e. the first element of the array) to be within the range of greater than -180 degrees and less than or equal to 180 degrees. The latitude (i.e. the second element of the array) is adjusted to be within the range of greater than -90 degrees and less than or equal to 90 degrees. Of course, the values provided must be in degrees.

Latitude values that indicate a traversal of the pole cause the longitude to be properly adjusted. For example, a coordinate pair of [90,135] is adjusted to [-90,45].

CS_bins BINary Search

```
int CS_bins (int fd,long start,long eof,int rs,Const void *rec,
             int (*comp)(Const void *,Const void*));
```

CS_bins will perform a binary search on the sorted file of fixed length records indicated by the file descriptor **fd**. **Rs** must specify the size, in bytes, of the record in the file, and **rec** must point to a record whose contents indicate the specific record that is to be searched for. The records are compared by calling the function **comp** with pointers to the records to be compared. **Comp** must return a value which is less than zero, zero, or greater than zero to indicate that the first record precedes, matches, or follows the second record respectively. (*Strcmp* is an acceptable compare function.)

The extent of the file that is searched is limited to the records which start at the file position indicated by *start* and end before the file position indicated by **eof**. This enables files with magic numbers and the like to be "bins'ed". If the value of **start** is less than zero, the current position of the file is used as the start position. If the value of **eof** is less than or equal to zero, the current end of file is used as the end of the search extent.

CS_bins returns a **TRUE** (i.e. +1) value if a record that matches the record provided (**rec**) was located; and returns a **FALSE** (i.e. zero) value if no such record could be located.

Upon a successful return from *CS_bins* (i.e. a matching record was located), the file will be positioned such that a read will produce the matched record. If more than one record existed with a matching key, the read will always produce the first of such records. Upon an unsuccessful return, (i.e. no matching record was found) the file will be positioned at the point where the record would be if the desired record had existed.

ERRORS

CS_bins will return a -1 and set `cs_Error` appropriately if any of the following conditions are detected:

cs_NO_MEM	Insufficient memory exists to *malloc* a buffer of **rs** bytes for I/O purposes.
cs_IOERR	A physical I/O error was encountered while performing the search.
cs_INV_FILE	A read of **rs** bytes failed to produce same even though the start and **eof** indicated that it should have.

CS_bswap Byte SWAPer

```
int CS_bswap (void *rec,Const char *frmt);
```

CS_bswap swaps bytes in a data record from little endian to big endian byte order, but only if the current system is a big endian byte order system. The **frmt** argument specifies the size and data types

contained in the data record that is pointed to by the **rec** argument as described below. The swapping occurs in place in the data area pointed to by **rec**.

The **frmt** argument points to an ordinary null terminated character string. In the simple case, each character in the string defines a data type:

c	a one byte character (no swapping)
s	a 16 bit integer
l	a 32 bit long integer
f	a 32 bit IEEE real (i.e. a float)
d	a 64 bit IEEE real (i.e. a double)

Case is significant, all characters must be lower case

Any such specification can be preceded by a decimal number to indicate an array of the item type that terminates the number. For example, "32c4l6ds" describes a record which consists of a 32 character array, an array of 4 longs, an array of 6 doubles, followed by a single short. Thus, with regard to swapping, the form of a structure can be easily and efficiently defined for this function.

CS_bswap has been written under the assumption that all data files containing binary data will be written using the little endian byte order. Thus, if *CS_bswap* detects that the current system is a little endian system, it does nothing successfully. If it detects that the current system is a big endian system, it swaps the bytes of all shorts, longs, floats, and doubles in the record. Obviously, it does nothing to character arrays.

CS_bswap returns a zero if no swapping was performed due to the fact that it determined that the current system is a little endian machine. It returns positive one if swapping was performed indicating that the current system is a big endian machine. It returns a -1 to indicate that the byte ordering of the current system is not supported.

CS_bswap uses a single 4 byte `long` to determine the byte ordering of the current system. Further, it only supports little endian and big endian byte orderings. The rather strange byte orderings that exist on some machines, such as some older DEC machines, are not supported.

The swap algorithm from little to big endian is exactly the same as big to little. I.e. the same function does either one. Therefore, there are no arguments to tell *CS_bswap* what the end result is supposed to be. It is up to the calling module to know what it's got, and what *CS_bswap* will return. Release 6.0 of CS-MAP, and later releases, assume that all binary data read from disk is in little endian order. It further assumes that all data written to disk will be in little endian order.

To disable dynamic byte swapping for whatever reason, simply write a stub for this function that does nothing, returns zero, and force link this stub before the linker searches the CS-MAP library.

BUGS

This function will not successfully swap bytes for some older machines, such as the PDP-11 or the VAX.

CS_cschk Coordinate System CHecK

```
int CS_cschk (Const struct cs_Csdef_ *cs_def,unsigned short chk_flg,int
err_list [],
      int list_sz);
```

CS_cschk verifies the validity of the coordinate system definition provided. For each abnormal condition detected, *CS_cschk* will add a CS-MAP error code to the integer array provided by the **err_list** argument. The **list_sz** argument must indicate the dimension of the **err_list** array. If **list_sz** is zero and/or **err_list** is the **NULL** pointer, *CS_cschk* will not attempt to return any error codes. As implied by the calling sequence, *CS_cschk* attempts to find all problems with a definition in a single call.

The **chk_flg** argument is a bit map indicating which of several options is to be active. This complication is necessary to enable this function to be used in the Dictionary Compiler whose environment is rather strange. The "orable" options are:

cs_CSCHK_DATUM	enables the checking of the datum reference, if any, in the coordinate system definition.
cs_CSCHK_ELLIPS	enables the checking of the ellipsoid reference, if any, in the coordinate system definition
cs_CHCHK_REPORT	instructs *CS_cschk* to report each error to *CS_erpt* before returning.

In all cases, *CS_cschk* will return the total number of abnormal conditions located, even if this value is greater than **list_sz**. In any case, this flag word, along with support for non-existent **err_list**, provides great flexibility for the calling module.

CS_cschk checks all elements of the coordinate system definition that are common to all coordinate systems. It then verifies the validity of the projection key name by looking it up in the projection table. Once the appropriate entry in the projection table is located, the specific projection check function is called to finish the evaluation.

CS_cslcl Coordinate System, LoCaL

```
struct cs_Csprm_ *CS_cslcl (Const double min_ll [2],
      Const double max_ll [2],
      Const char *units,
      Const struct cs_Datum_ *datum,
      double map_scl);
```

Given a region on the ellipsoid as specified by the **min_ll** and **max_ll** arguments, *CS_cslcl* creates a coordinate system based on the Transverse Mercator projection optimized for the defined region. A pointer to a *malloc*'ed coordinate system definition structure, initialized for the optimized coordinate system, is returned. This pointer is suitable for use with the *CS_cs2ll*, *CS_ll2cs*, *CS_csscl*, and *CS_cscnv* functions.

The **min_ll** array defines the southwestern corner of the region; the first element specifies the

longitude, the second the latitude. Both values are given in degrees where negative values are used to indicate south latitude and west longitude. Similarly, the **max_ll** array defines the northeastern corner of the region.

The **units** argument must point to a null terminated string that defines the units of the coordinate system, usually "METER" or "FOOT" (see CsdataU.c).

The **datum** argument must point to a datum definition structure that carries the definition of the datum to be used. Such a pointer can be obtained from *CS_dtloc*.

The **map_scl** argument can be used to scale the resulting coordinates to a specific map scale, or, perhaps, to plotter or display units. Specify a value of 1.0 to get unscaled values.

The resulting coordinate system definition should be released using *CS_free* when no longer needed.

The resulting coordinate system is based on the Transverse Mercator projection and has the following properties:

1. Central meridian bisects the region.
2. Latitude origin is the minimum latitude given in **min_ll**.
3. Scale reduction is optimized for the specified region.
4. Datum is as specified.
5. Units are as specified.
6. False northing set to zero. That is, the minimum latitude maps to a Y coordinate of zero.
7. False easting set to minimum longitude; i.e. the minimum longitude maps to an X coordinate value of zero.

Mapping scale is applied to all coordinates after conversion.

BUGS

Since the Transverse Mercator projection is used, this function is not recommended for high accuracy mapping where the east/west extent of the region exceeds six degrees of longitude.

CS_erpt Error RePorT

```
extern int cs_Error,cs_Errno;
void CS_erpt (int err_num);
```

CS_erpt is called by all functions in the Coordinate System Mapping Package whenever an error condition is detected. The value of **err_num** indicates the specific error condition detected and must be one of the manifest constants defined in cs_map.h.

At the current time, *CS_erpt* does nothing other than set the value of global variable `cs_Error` to the supplied value of *err_num* and set the global variable of `cs_Errno` to the current value of the system's global variable `errno`.

It is expected that users will want to write their own *CS_erpt* function which will properly inform the operator of the nature of the problem encountered.

Each function in the Coordinate System Mapping Package is programmed to clean up after itself after return from *CS_erpt*. That is, upon return from *CS_erpt*, all memory *malloc*'ed by the function detecting the error is *free*'ed and any temporary file created by the function detecting the error is removed.

CS_fillin coordinate system definition FILL IN

```
void CS_fillin (struct cs_Csdef_ *csPtr);
```

CS_fillin is designed to be called by a GUI function that displays coordinate system definitions. This function provides values for those projection parameters which are fixed, defaulted, or calculated for a specific projection. *CS_fillin* examines the provided definition, determines the projection, and then "fills in" parameters with the appropriate values.

For example, the relatively new UTM System Projection accepts zone number and hemisphere as the primary parameters. From these parameters, ordinary projection parameters such as origin longitude, false easting, false northing, etc. are computed from these values. *CS_fillin* can be used to have these values computed and set in the `cs_Csdef_` structure provided by the **csPtr** argument.

CS_ii??? Imaginary Arithmetic Functions

The functions described in this section represent CS-MAP's ability to deal with complex numbers, as is required for certain projections such as the Modified Sterographic and the New Zealand National Grid System. A complex number consists of an instance of the `cs_Cmplx_` structure that has real and imaginary elements.

CS_iiadd - ImagInary ADD

```
CS_iiadd (Const struct cs_Cmplx_ *aa,Const struct cs_Cmplx_ *bb,struct
cs_Cmplx_ *cc);
```

CS_iiadd will add the complex numbers referenced by the **aa** and **bb** arguments and return the result in the structure referenced by the **cc** argument. The **cc** argument may point to either the **aa** or **bb** array. The **aa** and **bb** arguments may point to the same structure.

CS_iisub - ImagInary SUBtract

```
CS_iisub (Const struct cs_Cmplx_ *aa,Const struct cs_Cmplx_ *bb,struct
cs_Cmplx_ *cc);
```

CS_iisub will subtract the complex number referenced by the **bb** argument from that referenced by the **aa** argument and return the result in the structure referenced by the **cc** argument. The **cc** argument may point to either the **aa** or **bb** structure. The **aa** and **bb** arguments may point to the same structure.

CS_iikmul - ImagInary Konstant MULtiply

```
CS_iikmul (Const struct cs_Cmplx_ *aa,double kk,struct cs_Cmplx_ *cc);
```

CS_iikmul will multiply the complex number referenced by the **aa** argument by the value of the **kk** argument, a real value, and return the result in the structure referenced by the **cc** argument. The **cc** argument may point to the same structure as the **aa** argument.

CS_iimul - ImagInary MULtiply

```
CS_iimul (Const struct cs_Cmplx_ *aa,Const struct cs_Cmplx_ *bb,struct
cs_Cmplx_ *cc);
```

CS_iimul will multiply the complex numbers referenced by the **aa** and **bb** arguments and return the result in the structure referenced by the **cc** argument. The **cc** argument may point to either the **aa** or **bb** structure. The **aa** and **bb** arguments may point to the same structure.

CS_iidiv - ImagInary DIVide

```
CS_iidiv (Const struct cs_Cmplx_ *aa,Const struct cs_Cmplx_ *bb,struct
cs_Cmplx_ *cc);
```

CS_iidiv will divide the complex number referenced by the **aa** by the complex number referenced by the **bb** argument and return the result in the structure referenced by the **cc** argument. The **cc** argument may point to either the **aa** or **bb** structure. The **aa** and **bb** arguments may point to the same structure.

CS_iisrs - ImagInary SeRieS

```
CS_iisrs (Const struct cs_Cmplx_ *aa,Const struct cs_Cmplx_ AB[],int nn,
     struct cs_Cmplx_ *cc);
```

CS_iisrs uses Horner's method to calculate the power series expansion of a complex number, the **aa** argument, to the **nn** power, using the array referenced by the **AB** argument as the coefficients of the series expansion. NOTE: the first element, i.e. the zero element, of the array referenced by the **AB** argument must contain a complex zero. The array referenced by the **AB** argument must contain **nn**+1 elements. The result is returned in the structure referenced by the **cc** argument. The **cc** argument may point to the same structure as the **aa** argument.

CS_iisrs1 - ImagInary SeRieS, 1st derivative

```
CS_iisrs1(Const struct cs_Cmplx_ *aa,Const struct cs_Cmplx_ AB[],int nn,
     struct cs_Cmplx_ *cc);
```

CS_iisrs1 operates in the same manner as *CS_iisrs* described elsewhere. However, the return value is the value of the first derivative of the series.

CS_iisrs0 - ImagInary SeRieS, alternate for new zealand

```
CS_iisrs0(Const struct cs_Cmplx_ *aa,Const struct cs_Cmplx_ AB[],int nn,
     struct cs_Cmplx_ *cc);
```

CS_iisrs0 operates in the same manner as *CS_iisrs* described elsewhere. However, the Horner method is not used (as yet) and the return value is that required by the New Zealand National Grid System. We suspect that Snyder's method of inverting the Modified Sterographic series is superior to that published for the New Zealand. If this suspicion survives further scrutiny, this function will not be present in future releases of CS_MAP.

CS_iiabs - ImagInary ABSolute value

```
double CS_iiabs (Const struct cs_Cmplx_ *aa);
```

CS_iiabs returns the absolute value of the complex number referenced by the **aa** argument.

CS_iicnj - ImagInary CoNJugate

```
CS_iicnj (Const struct cs_Cmplx_ *aa,struct cs_Cmplx_ *cc);
```

CS_iicnj computes the complex conjugate of the complex number referenced by the **aa** argument and returns the result in the structure referenced by the **cc** argument. The arguments may reference the same structure.

CS_init INITialize

```
int CS_init (int keepers);
```

CS_init initializes an instance of CS-MAP for independent operation. This function is provided for the sole purpose of initializing a new thread of execution in a multi-threaded environment. Use of *CS_init* is not normally required in multi-tasking environments. *CS_init* needs to be used only in environments where separate execution threads share the same data space.

When compiled for use in a multi-threaded environment, all global variables which CS-MAP uses dynamically are declared with the `__declspec (thread)` or `__thread` as is appropriate for the compiler in use. Thus, each thread will have it's own copy of these variables. *CS_init* will initialize these variables to values suitable for use by the new thread, independent of any other currently executing thread.

The **keepers** argument to *CS_init* is an integer bit map that can be constructed from the manifest constants described below. A value of zero provides for complete initialization of the thread, and is normally used. The global variables `cs_Dir` and `cs_DirP` are always inherited from the parent task (assuming that they are valid). All other values are initialized to their standard CS-MAP default values unless the **keepers** argument specifically instructs *CS_init* to preserve the value inherited from the parent thread.

Any, or all, of the following constants may be 'or'ed together to construct a value of the **keepers** argument. The typical value for **keepers** is zero.

cs_THRD_CSNAME	Preserve the parent thread's setting for the file name of the Coordinate System Dictionary (`cs_Csname`).
cs_THRD_DTNAME	Preserve the parent thread's setting for the file name of the Datum Dictionary (`cs_Dtname`).
cs_THRD_ELNAME	Preserve the parent thread's setting for the file name of the Ellipsoid Dictionary (`cs_Elname`).
cs_THRD_DTDFLT	Preserve the parent thread's setting for the default datum (`cs_DtDflt`).
cs_THRD_ELDFLT	Preserve the parent thread's setting for the default ellipsoid (`cs_ElDflt`).
cs_THRD_LUDFLT	Preserve the parent thread's setting for the default linear unit (`cs_LuDflt`).
cs_THRD_AUDFLT	Preserve the parent thread's setting for the default angular unit (`cs_AuDflt`).

BUGS

While *CS_init* assures that invalid dictionary directories and file names are never inherited from the parent thread, it does not check the validity of inherited defaults.

CS_ips In Place Sort

```
int CS_ips (int fd,int rs,long eof,int (*comp)(Const void *,Const void *));
```

CS_ips sorts a file of fixed length records into order under the control of the comparison function provided by **comp**. The sort is done in place, requiring no additional disk space. The file descriptor of the open file that is to be sorted is provided by **fd**. Of course, read and write access is required. The length of the records in the file, as a number of bytes, is specified by **rs**. Only that portion of the file between its current position when *CS_ips* is invoked and the position specified by **eof** is actually sorted. A value of zero or less for **eof** is taken to mean the current end of the file. *CS_ips* will return an indication of the number of record swaps required to sort the file. A zero return value indicates that the file was in order. A return value greater than zero indicates that the file had to be sorted. A -1 is returned if the sort failed for some reason. No other significance should be attributed to the return value.

The comparison function is called with two arguments, pointers to the two records that are to be compared. The comparison function must return a value that is negative if the first record should precede the second, zero if the records are equivalent or greater than zero if the first record should

follow the second. In the case where the sort key is a character array which is the first item in the records being sorted, *strcmp* is an acceptable comparison function.

THIS IS NOT A STABLE SORT. That is, the relative order of records with equal keys after sorting is not guaranteed to be, and will almost always not be, the same as it was prior to the sort.

The position of the file when this function is called establishes the position of the first fixed length record to be sorted. This enables files with magic numbers, headers, or other such stuff to be sorted. Failing to properly position the file before calling *CS_ips* produces some interesting results.

CS_ips uses a *malloc*'ed sort buffer of 16K bytes. The size of this sort buffer can be controlled by the application by setting the desired size, in bytes, in the global variable `cs_Sortbs`. `cs_Sortbs` is declared as an integer, therefore the size of the sort buffer cannot be larger than MAXINT.

CS_ips is not the speediest sort in the world. It is designed especially for sorting small files (256K or less) and for doing so without incurring disk space liability problems (i.e. in place, no extra disk space required). Performance on large files (i.e. greater than 512K) has been found to be abominable. Of course, if you can spare a larger buffer, sort performance will improve somewhat.

ERRORS

CS_ips will return a -1 and set the value of global variable `cs_Error` appropriately if any of the following conditions is encountered:

cs_NO_MEM	Insufficient memory was available to support the *malloc* of the sort buffer.
cs_IOERR	A physical I/O error occurred during the sort.

CS_isHlpAvailable IS HeLP file AVAILABLE

```
int CS_isHlpAvailable (void);
```

CS_isHlpAvailable will return +1 (i.e. TRUE) if CS-MAP is aware of the location of a file named cs-map.hlp. Otherwise, *CS_isHlpAvailable* returns zero (i.e. FALSE).

CS_lget Left justified GET

```
char *CS_lget (char *str,Const char *fld,int size,char fill);
```

CS_lget will produce, in the character array pointed to by **str**, a null terminated string that represents the data in a left justified data field pointed to by **fld**. The size of the field is given by **size**. The field is assumed to be filled (on the right) by the character given by **fill**. That is, trailing **fill** characters will be omitted in the null terminated result.

CS_lget returns a pointer to the terminating null character in **str**.

CS_lput Left justified field PUT

```
void CS_lput (char *fld,Const char *str,int size,char fill);
```

CS_lput creates a left justified field of **size** bytes at the location given by **fld**. The field is populated with data from the null terminated string given by **str** and padded on the right with **fill** characters as necessary. If the length of **str** exceeds **size**, the value in the field is the first **size** characters of **str**.

CS_nampp NAMe PreProcessor

```
int CS_nampp (char *key_nm);
```

By convention, coordinate system key names, datum definition key names, and ellipsoid definition key names:

not more than 23 characters in length;

may start with a special character only if that special character is the *cs_Unique* character (see CSdata (5CS));

may not include any special characters other than:

the hyphen character,

the underscore character,

the dollar sign character,

the period character (decimal point),

the semi-colon character,

the colon character,

the space character,

the `cs_Unique` character;

must contain at least one alphabetic character;

must not contain two consecutive space characters;

may start with numeric characters providing the first non-numeric character is alphabetic;

otherwise must start with either an alphabetic character or the `cs_Unique` character.

CS_nampp is used to force adherence to these requirements. While key names are case insensitive, case is preserved in dictionaries for display purposes. *CS_nampp* will strip leading and trailing white space from the provided name. *CS_nampp* will also ignore the characters involved in the default system.

Given a pointer to a character array containing a null terminated key name, *CS_nampp* will strip all leading and trailing white space and verify that the resulting name meets the conventions outlined above. If all of these conditions are met, *CS_nampp* will return a zero. Otherwise, a -1 is returned.

Note that *CS_nampp* will modify the contents of the array pointed to by the **key_nm** argument to meet the standards required of key names.

ERRORS

CS_nampp will return a -1 and set `cs_Error` appropriately if any of the following conditions are detected:

cs_INV_NAME	The supplied name contained non-printable characters, was longer than eight characters, or did not start with an alphabetic character.
cs_DBL_SPACE	The supplied name contained two consecutive spaces; a situation which can be difficult to observe in many GUI's.

CS_prchk Protection CHecK

```
int CS_prchk (short protect);
```

The single argument to the *CS_prchk* function, **protect**, is expected to be a protection value obtained from a dictionary definition. *CS_prchk* examines the value and determines if, in the current environment, the value indicates that the dictionary definition it was obtained from would be considered to be protected by the appropriate dictionary update function. A non-zero return value indicates that the item is protected.

CS_prjEnum PRoJection ENUMerator

```
int CS_prjEnum (int index,long *prj_flags,char *prj_keynm,int keynm_sz,
      char *prj_descr,int descr_sz);
```

CS_prjEnum is used to enumerate all projections in the projection table. *CS_prjEnum* returns in the memory buffer pointer to by the **prj_keynm** argument the key name of the **index**'th entry in the projection table. *CS_prjEnum* will never write more than **keynm_sz** bytes to the indicated location. Similarly, *CS_prjEnum* will write no more than **descr_sz** bytes of the description of the **index**'th entry in the projection table to the buffer referenced by the **prj_descr** argument. *CS_prjEnum* will copy the projection flags word to the location indicated by the **prj_flags** argument. Any, or all, of the three pointer arguments may be the **NULL** pointer, in which case *CS_prjEnum* does not attempt to return that specific piece of information.

If index is valid, *CS_prjEnum* returns the numeric projection code value assigned to the projection. If **index** is too large, a zero is returned. **Index** is a zero based index; the index of the first entry in the projection table is zero.

PROJECTION FLAGS

The following constants define specific bits in the 32 bit projection flag word. If the indicated bit is set, the condition/feature applies to the specific projection. Three bits are reserved for use by application programmers with source licenses. All other bits are reserved for use by OSGeo contributors. These constants are defined in cs_map.h.

cs_PRJFLG_SPHERE	Spherical form of this projection is supported.
cs_PRJFLG_ELLIPS	Ellipsoidal form of this projection is supported.
cs_PRJFLG_SCALK	An analytical k scale function is available.
cs_PRJFLG_SCALH	An analytical h scale function is available.
cs_PRJFLG_CNVRG	An analytical convergence angle function is available.
cs_PRJFLG_CNFRM	The projection is generally considered to be conformal.
cs_PRJFLG_EAREA	The projection is generally considered to be equal area.
cs_PRJFLG_EDIST	The projection is generally considered to be equidistant.
cs_PRJFLG_AZMTH	The projection is generally considered to be azimuthal.
cs_PRJFLG_GEOGR	The projection produces geographic coordinates, i.e. non-cartesian coordinates.
cs_PRJFLG_OBLQ	The projection is based on the oblique aspect of the base projection surface.
cs_PRJFLG_TRNSV	The projection is based on the transverse aspect of the base projection surface.
cs_PRJFLG_PSEUDO	The projection is generally considered to be based on a "pseudo" surface. This qualifies the actual surface in use. For example, the Eckert projections are considered to be pseudo cylindrical.
cs_PRJFLG_INTR	The projection supports interruptions.
cs_PRJFLG_CYLND	The base projection surface is the cylinder.
cs_PRJFLG_CONIC	The base projection surface is the cone.
cs_PRJFLG_FLAT	The base projection surface is the flat plane (e.g. azimuthal).
cs_PRJFLG_OTHER	The base projection surface is other than the three indicated immediately above.
cs_PRJFLG_SCLRED	The projection supports the concept of scale reduction.
cs_PRJFLG_ORGFLS	The projection does not use either false origin parameter.

cs_PRJFLG_ORGLAT	The projection does not use an origin latitude parameter. Origin latitude is deduced from other parameters and need not be specified directly.
cs_PRJFLG_ORGLNG	The projection does not use an origin longitude parameter. Origin longitude is deduced from other parameters and need not be specified directly.
cs_PRJFLG_USER1	Reserved for application programmers.
cs_PRJFLG_USER2	Reserved for application programmers.
cs_PRJFLG_USER3	Reserved for application programmers.

ERRORS

CS_prjEnum will return a -1 and set `cs_Error` appropriately if any of the following conditions are encountered:

cs_INV_INDX	The index argument was negative.

CS_prjprm PRoJection PaRaMeter usage

```
int CS_prjprm (struct cs_Prjprm_ *prj_prm,short prj_code,int prm_nbr);
```

CS_prjprm will return a positive one (+1) if the projection indicated by the **prj_code** argument uses the parameter indicated by the **prm_nbr** argument. **Prm_nbr** refers to one of the 24 parameters in the `cs_Csdef_` structure that is used to define coordinate systems. **Prj_prm** is zero based, so values of this argument should not exceed 23. **Prj_code** values are the numeric values now assigned to all projections in the cs_map.h header file (e.g. **cs_PRJCOD_ALBER**). If the **prj_prm** argument is not NULL, CS_prjprm will populate the structure provided with information as to how the indicated projection uses the indicated parameter.

If the projection referenced by the **prj_cod** argument does not use the parameter indicated, *CS_prjprm* returns a zero. If either the **prj_code** or the **prm_nbr** arguments are invalid, *CS_prjprm* returns a negative one (-1).

cs_Prjprm_ *Structure*

CS_prjprm uses an initialized array of `cs_Prjprm_` structures in order to do its work. This structure is declared in cs_map.h and the array is defined and initialized in CSdataPJ.c. Application programmers may need/wish to modify the contents of this initialized structure. The elements of this structure are:

min_val	minimum allowable value for the parameter.
max_val	maximum allowable value for the parameter.
deflt;	a suitable value to initialize this parameter to for a new definition.
format	a *CS_ftoa* format specification suitable for displaying this parameter. The log_typ element described below may be more useful for this purpose.
help_id	not used by CS-MAP (as yet). Reserved for use as a context sensitive help-id for this parameter.
labl_id	not used by CS-MAP (as yet). Reserved for use as the string resource ID of a suitable label for this parameter.
label	an 8 bit ASCII label for this parameter in English, standard C code page.
phys_typ	an integer code value indicating the physical type of this parameter. Since all parameters are doubles (currently), this value is always set to **cs_PRMPTYP_DBL** (currently).
log_typ	an integer code value indicating the logical type of this parameters. For example, **cs_PRMLTYP_LNG** indicates a longitude parameter.
prj_code	cs_Prjprm_ structures returned to users will have the **prj_code** argument copied into this element for identification purposes.
parm_nbr	cs_Prjprm_ structures returned to users will have the **prm_nbr** argument copied into this element for identification purposes.
sprf_type	a non-zero value indicates that the string contained in label or referenced by labl_id contains *sprintf* format specifications. The specific value of this element indicates the nature of the additional arguments that are to be passed to *sprintf*. Currently, only the value 1 is supported and it provides *sprintf* with an integer argument equal to (2 * **prm_nbr** + 1), suitable for labeling complex coefficient arguments for the Modified Stereographic.

CS_quadF QUADrant Forward

```
void CS_quadF (double xy [2],double xx,double yy,double x_off,double
```

```
y_off,short quad);
```

CS_quadF applies quadrant processing to the coordinates given by the **xx** and **yy** arguments, returning the results in the **xy** array provided. The **x_off** and the **y_off** arguments are the false easting and the false northing respectively. The quadrant processing requested is indicated by the **quad** argument.

The **quad** argument is a bit map of the following instructions:

cs_QUAD_INVX	Invert the X axis
cs_QUAD_INVY	Invert the Y axis
cs_QUAD_SWAP	Sawp the axes.

Note that the **quad** argument to this function is distinctly different from the **quad** element of the `cs_Csdef_` structure. Each projection setup function maps the **quad** element of the `cs_Csdef_` structure to the value described immediately above.

CS_quadF will always invert the appropriate axes first, then add the false origin values, and finally, if requested, swap the axes. A **quad** value of zero is valid; in which case *CS_quadF* becomes an expensive way to add the false origin to the coordinates.

CS_quadI QUADrant Inverse

```
void CS_quadI (double *xx,double *yy,Const double xy [2],double
x_off,double y_off,
      short quad);
```

CS_quadI performs the inverse of *CS_quadF*. In this case, quadrant processing is removed from the coordinate provided by the **xy** argument and the results are returned in the located pointed to by the **xx** and **yy** arguments.

CS_renam RENAMe a file

```
void CS_renam (Const char *old,Const char *new);
```

CS_renam will change the name of the existing file named **old** to that indicated by **new**. On MS-DOS systems, it simply calls the *rename* function. On UNIX systems, it uses the *link* and *unlink* system calls to obtain the same result. This function simply provides a degree of operating system independence.

ERRORS

CS_renam will return a -1 and set the value of global variable `cs_Error` appropriately if any of the following conditions is encountered:

cs_RENAME	The rename request failed for the reason indicated by the value of the `cs_Errno` global variable. Under UNIX, this means that either the link or the unlink system call failed.

CS_setHelpPath SET HELP PATH

```
int CS_setHelpPath (const char *helpPath);
```

Use the *CS_setHelpPath* function to set the directory that you desire to have CS-MAP search when seeking the MFC dialog help file. The **helpPath** argument must point to a null terminated string that carries the full path to the desired directory.

CS_setHelpPath returns +1 (i.e. **TRUE**) if a properly named file exists in the indicated directory. Zero (i.e. **FALSE**) is returned if such a file does not exist.

CS_stcpy STring CoPY

```
char *CS_stcpy (char *dest,Const char *source);
```

CS_stcpy copies the null terminated character string pointed to by **source** to the character array pointed to by **dest**. A pointer to the null terminating character in **dest** is returned.

CS_stcpy returns a pointer to the null character that terminates the newly copied string (as opposed to its first argument). Otherwise, it is identical to the *strcpy* function.

BUGS

Obviously, *CS_stcpy* knows nothing of the size of the destination character array and cannot prevent the source string from exceeding its size.

CS_stncp STring, N characters at most, CoPy

```
char *CS_stncp (char *dest,Const Const char *source,int size);
```

CS_stncp copies at most **size** minus one characters from the null terminated character string pointed to by **source** to the character array pointed to by **dest**. The result is guaranteed to be null terminated. The result, including the terminating null character, will not occupy more than **size** bytes. A pointer to the null character that terminates the result in **dest** is returned.

CS_stncp differs from *strncpy* in two ways. Most importantly, it guarantees a null terminated result, copying one less character than *strncpy* when the source string will not fit in the destination array. Second, it returns a pointer to the terminating null character in the destination array as opposed to a pointer to the destination array itself.

ERRORS

Since *CS_stncp* guarantees the result to be **NULL** terminated, it cannot perform successfully if given a zero (or negative) **size** argument. In this case, the **NULL** pointer is returned.

CS_stricmp STRing, case Insensitive, CoMPare

```
int CS_stricmp (Const char *str1,Const char *str2);
```

CS_stricmp compares the string pointed to by the **str1** argument to the string pointer to by **str2**. The comparison ignores case. The returned result is negative if **str1** should collate before **str2**, zero if the two strings are identical (case excepted), and positive if **str1** should collate after **str2**.

While similar functions are available in most 'C' run time libraries and are a part of the ANSI standard, the developers of CS-MAP have found variations in how various libraries deal with the special characters in between the upper and lower case characters. Thus, to have CS-MAP operate consistently on all platforms, we have our own implementation of this function.

CS_strincmp STRing, case Insensitive, N chars max, CoMPare

```
int CS_strincmp (Const char *str1,Const char *str2,int nCount);
```

CS_strincmp compares the string pointed to by the **str1** argument to the string pointer to by **str2**. No more than **nCount** characters are compared. The comparison ignores case. The returned result is negative if **str1** should collate before **str2**, zero if the two strings are identical (case excepted), and positive if **str1** should collate after **str2**.

While similar functions are available in most 'C' run time libraries, the developers of CS-MAP have found variations in how various libraries deal with the special characters in between the upper and lower case characters. Thus, to have CS-MAP operate consistently on all platforms, we have our own implementation of this function.

CS_stristr find STRing, case Insensitive, in a STRing

```
Const char *CS_stristr (Const char *str1,Const char *str2);
```

CS_stristr searches the string pointed to by the **str1** argument for an occurrence of the string pointed to by the **str2** argument. *CS_stristr* ignores case while looking for a match. If an occurrence of **str2** is found in **str1**, *CS_stristr* returns a pointer to the first such occurrence in **str1**. Otherwise, *CS_stristr* returns the **NULL** pointer.

While similar functions are available in most 'C' run time libraries, the developers of CS-MAP have found variations in how various libraries deal with the special characters in between the upper and lower case characters. Thus, to have CS-MAP operate consistently on all platforms, we have our own implementation of this function.

CS_swpal SWaP ALl binary data files

```
int CS_swpal (void (*prog)(Const char *file_name));
```

CS_swpal will cause all CS-MAP binary data files in the current data directory (i.e. `cs_Dir`) to be byte swapped. If the **prog** argument is not NULL, it is called with the name of the file to be swapped just prior to initiating the swap (see *CS_swpfl*). If the swap of the entire directory was successful, *CS_swpal*

returns a zero. Otherwise, the CS-MAP error code of the condition that caused termination of the swap operation is returned.

CS_swpal swaps all data files into a complete set of temporary files. Only when all such conversions are successfully completed will it then replace the original files with the swapped temporaries. In the event of an error, the temporaries are simply removed and the originals remain unchanged.

CS_swpal uses the *CS_bswap* function to perform byte swaps. It does not expect to be called if *CS_bswap* will not swap bytes. The result of calling *CS_swpal* when *CS_bswap* is not swapping is undefined. The result is usually a big long no-op, but this is not guaranteed.

CS_swpal was written specifically for testing purposes. It could, however, be used as part of a manufacturing process to swap all data files for distribution. *CS_swpal* will swap bytes in NADCON (i.e. .LAS and .LOS files) as well as Canadian National Transformation (version 1) data files.

BUGS

CS_swpal will not properly swap bytes in encrypted dictionary files, nor the Australian version of the NTv2 files.

CS_swpfl SWaP a single FiLe

```
char *CS_swpfl (Const char *file_name);
```

CS_swfll will swap the bytes in the file named by the **file_name** argument. The file is required to reside in the CS-MAP data directory indicated by `cs_Dir` and the result is written to a temporary file (see *CS_tmpfn*) in the same directory. A pointer to a static memory array containing the name of this temporary file, and only the name, is returned. In the event of an error, the **NULL** pointer is returned.

CS_swpfl was written primarily for testing purposes. It uses *CS_bswap* to effect the swapping of the data elements in the file. It should not be called when *CS_bswap* is not swapping anything. (Why would you want to anyway?). The result of calling *CS_swpfl* when *CS_bswap* is not swapping anything is usually a big no-op, but this is not guaranteed.

BUGS

CS_swpfl will not properly swap bytes in encrypted dictionary files or Australian version NTv2 files.

CS_tpars Table PARSe

```
Const char *CS_tpars (char **pntr,Const char *table,int tab_size);
```

CS_tpars is designed to parse tokens from the text source, ***pntr**, of specific values; the values being given in the **table** array. **Tab_size** specifies the size of the elements in the **table** array.

Successful parses return a pointer to the matching table element and ***pntr** is updated to point at the character immediately following the matched token. Unsuccessful parses return the **NULL** pointer value and the value of ***pntr** is unaltered.

The first byte of each element of **table** must contain the size of the token that is described in the **table** element. The actual key value must begin in the second byte of the **table** element. The remainder of

the **table** entry can be used to satisfy application requirements. A **table** element with a zero for the key size must be present to terminate the table. Such parsing tables are useful in converting specific token values to code values while parsing and validating at the same time. For example:

```
struct prs_tab_
{
        unsigned char key_len;    /* Assumes byte alignment */
        char key [8];
        short code_value;
} prs_tab [] =
{
        {5,"NORTH",0},
        {5,"SOUTH",2},
        {4,"EAST",1},
        {4,"WEST",3}
};
char text [128];
char *cp;
struct prs_tab_ *tab_ptr;
.
.
cp = text;
tab_ptr = (struct prs_tab_ *)CS_tpars (&cp,(struct prs_tab_ *)prs_tab,sizeof (struct
prs_tab_));
if (tab_ptr == NULL)
{
        printf ("Invalid direction.\n");
}
else if (tab_ptr->code_value == 1)
{
        /* Here if EAST was specified. */
}
.
.
```

It is important to note that it is usually best to have **table** ordered by the size of the key, the largest values first. This will cause the longest possible match to be returned which is usually what is desired.

BUGS

CS_tpars will always return the first match encountered. If the table is not ordered properly (by key size, largest first) and keys with duplicate initial characters exist (e.g. "WEST" and "WESTERN"), the result may not be what is desired.

CS_trim character array TRIM

```
int CS_trim (char array);
```

CS_trim trims leading and trailing white space from the null terminated character string in the array

pointed to by the **array** argument. The length of the resulting null terminated string is returned. White space, for this function, is considered to be blank, tab, and new-line characters.

CS_zones extract ZONES from definition

```
int CS_zones (Const struct cs_Csdef_ *csdef,struct cs_Zone_ zones [8]);
```

CS_zones will extract zone definitions from the 24 projection parameters in the coordinate system definition provided by the **csdef** argument, and place these extracted definitions in the `cs_Zone_` structure array pointed to by the **zones** argument. The **zones** argument must point to an array of not less than 8 `cs_Zone_` structures. *CS_zones* returns the number of valid zones extracted.

A single zone is defined by three doubles in the `prj_prm` section of the `cs_Csdef_` structure. Each group of three doubles is used to represent a zone, up to eight zones. `Prj_prm1` thru `prj_prm3` are used to defined the first zone; `prj_prm4` thru `prj_prm6` are used to defined the second zone, and so on. *CS_zone* needs to extract four elements of information for each zone: the longitude of the western extent of the zone, the central meridian of the zone, the longitude of the eastern extent of the zone, and whether the zone is for the northern or southern hemisphere. However, in order to support eight zones, we only have three doubles in which to encode this information. Therefore, the coding scheme described below is used for each group of three doubles and applies to each zone.

A zone is defined by a group of three doubles. *CS_zones* will ignore any group of three doubles in which the first and the third are both zero. In all other cases, the first of the three doubles is taken to be the longitude of the western extent of the zone. The second of the three doubles is taken to be the central meridian of the zone. The third of the three doubles is taken to be the longitude of the eastern extent of the zone. To indicate the hemisphere (north or south) to which the zone definition applies, the magnitude of the first double is increased by a certain amount (the sign remains unaffected).

Magnitude increases are interpreted as follows:

0.0	Zone includes both hemispheres.
1000.0	Zone applies to the northern hemisphere only.
2000.0	Zone applies to the southern hemisphere only.
3000.0	Zone applies to both hemispheres (redundant, but maintained for consistency).

For example, for a zone where the western extent is 100 degrees west longitude (-100.0), and which is to apply only to the southern hemisphere, the value of the first of the three doubles would be -2100.0.

CS_znlocF ZoNe LOCator Forward

```
Const struct cs_Zone_ *CS_znlocF (Const struct cs_Zones_ zones [8],int
zone_cnt,
     double lng,double lat);
```

Given a pointer to an array of **zone_cnt** `cs_Zone_` structures, and a coordinate pair, *CS_znlocF* and

CS_znlocI will return a pointer to the specific cs_Zone_ element in the array to which the given location belongs. The **NULL** pointer is returned if the provided location does not reside in any zone. In the case of *CS_znlocF*, the location is provided by the **lng** and **lat** arguments where **lng** gives the longitude in degrees east of Greenwich (i.e. negative values for west longitude) and **lat** provides the latitude in degrees north of the equator.

These functions are called by the forward and inverse conversion functions of projections that support interrupted zones. By CS-MAP convention, zones other than the easternmost zone include the zone's westernmost boundary but not its easternmost boundary. The easternmost zone includes both its western and eastern boundaries. In all cases, the equator is considered part of the northern hemisphere.

CS_znlocI ZoNe LOCator Inverse

```
Const struct cs_Zone_ *CS_znlocI (Const struct cs_Zones_ zones [8],int
zone_cnt,
      double xx,double yy);
```

Given a pointer to an array of **zone_cnt** cs_Zone_ structures, and a coordinate pair, *CS_znlocI* and *CS_znlocF* will return a pointer to the specific cs_Zone_ element in the array to which the given location belongs. The **NULL** pointer is returned if the provided location does not reside in any zone. In the case of *CS_znlocI*, the location is provided by the **xx** and **yy** arguments that are the cartesian X and Y coordinates of the coordinate system.

These functions are called by the forward and inverse conversion functions of projections that support interrupted zones. By CS-MAP convention, zones other than the easternmost zone include the zone's westernmost boundary but not its easternmost boundary. The easternmost zone includes both its western and eastern boundaries. In all cases, the equator is considered part of the northern hemisphere.

CSbcclu Basic Cached Coordinate system Look Up

```
struct cs_Csprm_ *CSbcclu (Const char *cs_name);
```

Csbcclu searches the coordinate system cache for a coordinate system with a name that matches the **cs_name** argument. If such an entry is found, the associated cs_Csprm_ structure pointer is returned. If a coordinate system with the name provided is not found, *CSbcclu* uses *CS_csloc* to obtain such a pointer and adds the name and associated pointer to the cache, deleting the least recently access entry if necessary.

In any case, *Csbcclu* returns a pointer to an initialized cs_Csprm_ structure that defines the named coordinate system.

This function was originally written in support of an interface designed for application programmers using the Basic language; hence the name.

ERRORS

Csbcclu will return the **NULL** pointer and set cs_Error appropriately if any of the following conditions are encountered while obtaining the definition of the named coordinate system:

cs_UNKWN_PROJ	The projection specified in the coordinate system definition is unknown to the system.

Csbcclu uses *CS_csloc* which uses the following functions which detect a majority of the exceptional conditions which 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.

CSbdclu Basic Datum Conversion Look Up

```
struct cs_Dtcprm_ *CSbdclu (Const struct cs_Csprm_ *src_cs,
     Const struct cs_Csprm_ *dst_cs,
     int dat_err,
     int blk_err);
```

Csbdclu searches the datum conversion cache for a datum conversion parameter block generated by the same coordinate systems as those provided by the **src_cs** and the **dst_cs** arguments. If such an entry is found, the associated `cs_Dtcprm_` structure pointer is returned. If a datum conversion based on the two coordinate systems provided is not found, *CSbdclu* uses *CS_dtcsu* to obtain such a pointer and adds the names and associated pointer to the cache, deleting the least recently access entry if necessary.

In any case, *Csbdclu* returns a pointer to an initialized `cs_Dtcprm_` structure that defines the required datum conversion.

The **dat_err** argument is the value that is to be passed to *CS_dtcsu* in the event that function needs to be called in order to obtain a datum conversion parameter structure. 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 value of the **blk_err** argument is stored with the datum conversion parameter block for access by the impending call to *CS_dtcvt.* 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 which 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, report the error condition once, using the datum conversion parameter block as the memory device for recording the location of the block which caused the error.
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.

This function was originally written in support of an interface designed for application programmers using the Basic language; hence the name.

ERRORS

Csbdclu will return the NULL pointer and set `cs_Error` appropriately if a call to *CS_dtcsu* is necessary and that call fails. See *CS_dtcsu* for a description of the possible error conditions.

CSbt???? BeTa (authalic latitude) calculation

Functions described in this section implement the calculations associated with authalic (equal area) latitude.

CSbtFcal BeTa Forward CALculation

```
double CSbtFcal (Const struct cs_BtcofF_ *bt_ptr,double lat);
```

Given the geographic latitude, **lat**, in radians, *CSbtIcal* returns the corresponding authalic latitude (in radians, negative for south latitude), which Snyder refers to as beta. The **bt_ptr** argument points to a structure of power series coefficients, as developed by *CSbtFsu*, which can be determined solely from the ellipsoid in use. Therefore, *CSbtFsu* performs all calculations that can be performed once, while *CSbtFcal* performs all calculations that must be performed for each coordinate to be converted. See Snyder, John P., Map Projections - A Working Manual, U.S. Geological Survey Professional Paper 1395, pages 16-19.

CSbtFsu BeTa Forward SetUp

```
void CSbtFsu (struct cs_BtcofF_ *bt_ptr,double e_sq);
```

CSbtFsu develops the coefficients of the power series required to compute the authalic latitude (which Snyder refers to as beta) from the geographic latitude. The resulting coefficients are stored in the structure pointed to **bt_ptr**. The **e_sq** argument is the square of the eccentricity of the ellipsoid.

Since the square of the eccentricity is the only input, these calculations can be performed once, once the ellipsoid definition being used is known. The calculations required for each individual latitude are performed by *CSbtFcal*. See Snyder, John P., Map Projections - A Working Manual, U.S. Geological Survey Professional Paper 1395, pages 15, 19, and 45.

CSbtIcal BeTa Inverse CALculation

```
double CSbtIcal (Const struct cs_BtcofI_ *bt_ptr,double beta);
```

Given the authalic latitude, **beta**, in radians, *CSbtIcal* returns the corresponding geographic latitude, in radians. The **bt_ptr** argument points to a structure of power series coefficients, as developed by *CSbtIsu*, which can be determined solely from the ellipsoid in use. Therefore, *CSbtIsu* performs all calculations that can be performed once, while *CSbtIcal* performs all calculations that must be performed for each coordinate to be converted. See Snyder, John P., Map Projections - A Working Manual, U.S. Geological Survey Professional Paper 1395, pages 15, 19, and 45.

CSbtIsu BeTa Inverse SetUp

```
void CSbtIsu (struct cs_BtcofI_ *bt_ptr,double e_sq);
```

CSbtIsu develops the coefficients of the power series required to compute the geographic latitude from the authalic latitude (which Snyder refers to as beta). The resulting coefficients are stored in the structure pointed to by **bt_ptr**. The **e_sq** argument is the square of the eccentricity of the ellipsoid.

Since the square of the eccentricity is the only input, these calculations can be performed once, once the ellipsoid definition being used is known. The calculations required for each individual latitude are performed by *CSbtIcal*. See Snyder, John P., Map Projections - A Working Manual, U.S. Geological Survey Professional Paper 1395, pages 15, 19, and 45.

CSccsphrD angular distance (CC) on SPHeRe in Degrees

```
double CSccsphrD (Const double ll0 [2],Const double ll1 [2]);
```

CSccsphrD returns the angular distance, in degrees, between the two geographic locations given by **ll0** and **ll1**. For this function, **ll0** and **ll1** are in degrees.

A spherical earth is assumed. The technique used should produce accurate results from very close to zero to very close to .

CSccsphrR angular distance (CC) on SPHeRe in Radians

```
double CSccsphrR (Const double ll0 [2],Const double ll1 [2]);
```

CSccsphrR returns the angular distance, in radians, between the two geographic locations given by **ll0** and **ll1**. For this function, **ll0** and **ll1** are in radians.

A spherical earth is assumed. The technique used should produce accurate results from very close to zero to very close to .

CScsKeyNames Coordinate System Key Names

```
char *CScsKeyNames (void);
```

CScsKeyNames returns a pointer to a list that contains the key names of all coordinate systems in the Coordinate System Dictionary. The list consists of null terminated strings and the entire list is terminated by two consecutive null characters. The coordinate system key names are in the same order as they appear in the Coordinate System Dictionary.

The resulting pointer is cached by *CScsKeyNames* in a global variable named `cs_CsKeyNames`. Thus, once generated, *CScsKeyNames* returns the same pointer. Thus, it is important applications do not free the returned pointer unless they also set the `cs_CsKeyNames` global variable to the **NULL** pointer. *CS_recvr* frees the list and is the normal means for freeing this list. The cached list is used by the *CS_csEnum* and *CS_csIsValid* functions.

ERRORS

CScsKeyNames will return the **NULL** pointer and set `cs_Error` appropriately if any of the following conditions are encountered:

cs_CSDICT	The Coordinate System Dictionary could not be found or otherwise opened. (See *CS_altdr*)
cs_IOERR	A physical I/O error occurred in accessing the Coordinate System Dictionary.
cs_CS_BAD_MAGIC	The file assumed to be a Coordinate System Dictionary by virtue of its name was not a Coordinate System Dictionary; it had an invalid magic number.
cs_NO_MEM	Insufficient memory was available for the creation of the key name list.

CSchi???? CHI (conformal latitude) calculation

Functions described in this section implement the calculations associated with isometric (conformal) latitude.

CSchiFcal CHI Forward CALculation

```
double CSchiFcal (Const struct cs_ChicofF_ *chi_ptr,double lat);
```

Given the geographic latitude, **lat**, in radians, *CSchiIcal* returns the corresponding conformal latitude (in radians, negative for south latitude), which Snyder refers to as **chi**. The **chi_ptr** argument points to a structure of power series coefficients, as developed by *CSchiFsu*, which can be determined solely from the ellipsoid in use. Therefore, *CSchiFsu* performs all calculations that can be performed once, while *CSchiFcal* performs all calculations that must be performed for each coordinate to be converted. See Snyder, John P., Map Projections - A Working Manual, U.S. Geological Survey Professional Paper 1395, pages 15, 19, and 45.

CSchiFsu CHI Forward SetUp

```
void CSchiFsu (struct cs_ChicofF_ *chi_ptr,double e_sq);
```

CSchiFsu develops the coefficients of the power series required to compute the conformal latitude (which Snyder refers to as chi) from the geographic latitude. The resulting coefficients are stored in the structure pointed to **chi_ptr**. The **e_sq** argument is the square of the eccentricity of the ellipsoid.

Since the square of the eccentricity is the only input, these calculations can be performed once, once the ellipsoid definition being used is known. The calculations required for each individual latitude are performed by *CSchiFcal*. See Snyder, John P., Map Projections - A Working Manual, U.S. Geological Survey Professional Paper 1395, pages 15, 19, and 45.

CSchiIcal CHI Inverse CALculation

```
double CSchiIcal (Const struct cs_ChicofI_ *chi_ptr,double chi);
```

Given the conformal latitude, **chi**, *CSchiIcal* returns the corresponding geographic latitude. The **chi_ptr** argument points to a structure of power series coefficients, as developed by *CSchiIsu*, which can be determined solely from the ellipsoid in use. Therefore, *CSchiIsu* performs all calculations that can be performed once, while *CSchiIcal* performs all calculations that must be performed for each coordinate to be converted. See Snyder, John P., Map Projections - A Working Manual, U.S. Geological Survey Professional Paper 1395, pages 15, 19, and 45.

CSchiIsu CHI Inverse SetUp

```
void CSchiIsu (struct cs_ChicofI_ *chi_ptr,double e_sq);
```

CSchiIsu develops the coefficients of the power series required to compute the geodetic latitude from the conformal latitude (which Snyder refers to as chi). The resulting coefficients are stored in the structure pointed to **chi_ptr**. The **e_sq** argument is the square of the eccentricity of the ellipsoid.

Since the square of the eccentricity is the only input, these calculations can be performed once, once the ellipsoid definition being used is known. The calculations required for each individual latitude are performed by *CSchiIcal*. See Snyder, John P., Map Projections - A Working Manual, U.S. Geological Survey Professional Paper 1395, pages 15, 19, and 45.

Csdfltpro DeFaULT PROcessor

```
int CSdfltpro (int type,char *name,int size);
```

Csdfltpro is the internal function used to perform default processing. The **type** argument must be set to one to the manifest constants that define one of the four possible "defaultable" elements of a coordinate system or datum definition. **Name** points to the name that may be a "defaultable" reference, and **size** indicates the size of the character array pointed to by the **name** argument.

If **name** is enclosed with the default character sequences defined in cs_map.h and the specific default feature indicated by **type** is active, *Csdfltpro* replaces the value in name with the default, surrounding it with the replacement character sequences also defined in cs_map.h.

If a replacement is made, *Csdlftpro* returns **TRUE**, otherwise it returns **FALSE**.

As distributes, an element is considered "defaultable" if it is enclosed with square brackets. Once replaced with a default value, it is enclosed in parenthesis as an indication that a default substitution was made.

The manifest constants that are valid values for the **type** argument are:

Cs_DFLTSW_DT	Datum name replacement.
Cs_DFLTSW_EL	Ellipsoid name replacement.
Cs_DFLTSW_LU	Linear unit replacement.
Cs_DFLTSW_AU	Angular unit replacement.

CSdtKeyNames DaTum Key Names

```
char *CSdtKeyNames (void);
```

CSdtKeyNames returns a pointer to a list that contains the key names of all datums in the Datum Dictionary. The list consists of null terminated strings and the entire list is terminated by two consecutive null characters. The datum key names are in the same order as they appear in the Datum Dictionary.

The resulting pointer is cached by *CSdtKeyNames* in a global variable named `cs_DtKeyNames`. Thus, once generated, *CSdtKeyNames* returns the same pointer. Thus, it is important applications do not free the returned pointer unless they also set the `cs_DtKeyNames` global variable to the NULL pointer. *CS_recvr* frees the list and is the normal means for freeing this list. The cached list is used by the *CS_dtEnum* and *CS_dtIsValid* functions.

ERRORS

CSdtKeyNames will return the NULL pointer and set `cs_Error` appropriately if any of the following conditions are encountered:

Cs_DTDICT	The Datum Dictionary could not be found or otherwise opened. (See *CS_altdr*)
Cs_IOERR	A physical I/O error occurred in accessing the Datum Dictionary.
Cs_DT_BAD_MAGIC	The file assumed to be a Datum Dictionary by virtue of its name was not a Datum Dictionary; it had an invalid magic number.
Cs_NO_MEM	Insufficient memory was available for the creation of the key name list.

CSelKeyNames ELlipsoid Key Names

```
char *CSelKeyNames (void);
```

CSelKeyNames returns a pointer to a list that contains the key names of all ellipsoids in the Ellipsoid Dictionary. The list consists of null terminated strings and the entire list is terminated by two consecutive null characters. The ellipsoid key names are in the same order as they appear in the Ellipsoid Dictionary.

The resulting pointer is cached by *CSelKeyNames* in a global variable named `cs_ElKeyNames`. Thus, once generated, *CSelKeyNames* returns the same pointer. Thus, it is important applications do not free the returned pointer unless they also set the `cs_ElKeyNames` global variable to the **NULL** pointer. *CS_recvr* frees the list and is the normal means for freeing this list. The cached list is used by the *CS_elEnum* and *CS_elIsValid* functions.

ERRORS

CSelKeyNames will return the **NULL** pointer and set `cs_Error` appropriately if any of the following conditions are encountered:

Cs_ELDICT	The Ellipsoid Dictionary could not be found or otherwise opened. (See *CS_altdr*)
Cs_IOERR	A physical I/O error occurred in accessing the Ellipsoid Dictionary.
Cs_EL_BAD_MAGIC	The file assumed to be a Ellipsoid Dictionary by virtue of its name was not a Ellipsoid Dictionary; it had an invalid magic number.
Cs_NO_MEM	Insufficient memory was available for the creation of the key name list.

CSllnrml Latitude/Longitude NoRMaL

```
void CSllnrml (Const double oll [2],Const double ll [2],double ll1
[2],double ll2 [2]);
```

Given the endpoints of an arc on the ellipsoid in terms of latiude and longitude, *CSllnrml* returns the latitude and longitude of an arc on the ellipsoid that is normal to the original arc and has a length of one second of arc.

The original arc is specific by the **oll** and **ll** arguments. The returned arc is normal to the supplied arc at the location specified by the **ll** argument. The endpoints of the returned arc are returned in the arrays pointed to by the **ll1** and the **ll2** arguments.

All latitudes and longitudes are in degrees where negative values are used for west longitude and south latitude. In all cases, the longitude is the first element in each array and the latitude is the second element.

This function has been developed expressly for the empirical calculation of K grid scale factors for azimuthal projections. The returned latitude/longitude positions are converted to grid coordinates and the grid distance is compared to the actual geodetic distance on the ellipsoid as computed by *CS_llazdd*. Therefore, customary usage is to provide the origin of the coordinate system as the **oll** argument, and the point at which the grid scale factor is to be computed as the **ll** argument.

CSllnrml uses spherical trigonometry, i.e. assumes the earth is a sphere.

ERRORS

CSllnrml makes no checks for possible errors. Given reasonable values for the latitudes and longitudes, and values for **oll** and **ll** that are not antipodal (i.e. opposite ends of a line passing through the center of the earth) no errors should occur.

CSmm???? Meridional distance functions

Functions described in this section implement the calculations associated with calculating the meridional distance from the equator to a geographic latitude.

CSmmFcal M Forward CALculation

```
double CSmmFcal (Const struct cs_MmcofF_ *mm_ptr,double lat,double sin_lat,
      double cos_lat);
```

Given the geographic latitude, **lat**, in radians, and its sine (**sin_lat**) and its cosine (**cos_lat**), *CSmmFcal* returns the meridional distance from the equator to the geographic latitude on the ellipsoid. (Snyder calls this *M*). The return value is in the same units of the scaled equatorial radius argument that was supplied to the *CSmmFsu* function when populating the structure pointed to by the **mm_ptr** argument (see below).

The **mm_ptr** argument points to a structure of power series coefficients as developed by the *CSmmFsu* function, which can be calculated solely from the ellipsoid in use. Therefore, the coefficients pointed to by **mm_ptr** can be calculated once, once the ellipsoid definition is known.

BUGS

The calling sequence requires three forms of the geodetic latitude as these values are usually available to calling functions. Thus, while redundant, this calling sequence assists in maintaining high performance levels.

CSmmFsu M Forward SetUp

```
void CSmmFsu (struct cs_MmcofF_ *mm_ptr,double ka,double e_sq);
```

CSmmFsu develops the coefficients of the power series required to compute the meridional distance from the equator to a specific geographic latitude. The resulting coefficients are stored in the structure pointed to **mm_ptr**. The **ka** argument is the equatorial radius of the ellipsoid in the units of which the calculated meridional distance is to be returned. The **e_sq** argument is the square of the eccentricity

of the ellipsoid.

Since the basic parameters of the ellipsoid are the only inputs, these calculations can be performed once, once the ellipsoid definition being used is known. The calculations required for separate latitudes are performed by *CSmmFcal*.

CSmmIcal M Inverse CALculation

```
double CSmmIcal (Const struct cs_MmcofI_ *mm_ptr,double mm);
```

Given the meridional distance **mm**, *CSmmIcal* returns the corresponding geographic latitude in radians where south latitude is negative. The **mm_ptr** argument points to a structure of power series coefficients, as developed by the *CSmmIsu*, which can be determined solely from the ellipsoid in use. Therefore, *CSmmIsu* performs all calculations that can be performed once, while *CSmmIcal* performs all calculations that must be performed for each coordinate to be converted. The units of the supplied value of **mm** must be the same as that provided to the *CSmmIsu* function, via the **ka** argument, which produced the contents of the structure pointed to by **mm_ptr**.

CSmmIsu M Inverse SetUp

```
void CSmmIsu (struct cs_MmcofI_ *mm_ptr,double ka,double e_sq);
```

CSmmIsu develops the coefficients of the power series required to compute the geographic latitude from the meridional distance (which Snyder calls M). The resulting coefficients are stored in the structure pointed to **mm_ptr**. The **ka** argument is the equatorial radius of the ellipsoid in the same units as the meridional distances *CSmmIcal* will be expected to process. The **e_sq** argument is the eccentricity of the ellipsoid squared.

Since the basic parameters of the ellipsoid are the only inputs, these calculations can be performed once, once the ellipsoid definition being used is known. The calculations required for separate latitudes are performed by *CSmmIcal*.

Dictionary Access Functions

Functions that can be used to read, write, and otherwise manipulate the dictionary files are described in this section. Dictionary files are ordinarily distributed in encrypted form. This enables a rather simple, but somewhat effective means of protecting the valuable information which often resides in the dictionaries.

The dictionary compilers can produce unencrypted dictionaries for testing purposes.

CS_cscmp Coordinate System CoMPare

```
int CS_cscmp (Const struct cs_Csdef_ *pp,Const struct cs_Csdef_ *qq);
```

CS_cscmp compares the two coordinate system definition structures provided to it by **pp** and **qq** and returns an integer that represents the collating sequence relationship between the two coordinate system definitions. The collating sequence is based on the key name of the coordinate system definition. This function is used rather than *strcmp* as this function can compare encrypted entries as

well as unencrypted entries.

This function is used in conjunction with *CS_ips* and *CS_bins* to access Coordinate System definitions in the Coordinate System Dictionary.

CS_csDictCls Coordinate System DICTionary file CLoSe

void CS_csDictCls (csFILE* stream);

Applications must use this function to close a stream opened with the *CS_csopn* function. Using this function assures that any memory of the open stream used for deferred close is erased.

CS_csfnm Coordinate System dictionary File NaMe

```
int CS_csfnm (Const char *new_name);
```

CS_csfnm changes the file name that CS-MAP uses when opening the Coordinate System Dictionary to that specified by **new_name**. This does not change the directory. Use *CS_altdr* to change the directory.

CS_csgrp Coordinate System dictionary GRouP

```
int CS_csgrp (Const char *grp_key,struct cs_Csgrplst_ **grp_lstp);
void CS_csgrpf (struct cs_Csgrplst_ *grp_lst);
```

Given the name of a coordinate system dictionary group via the **grp_key** argument, *CS_csgrp* returns a count of the number of coordinate systems in the specified group. At the location provided by the **grp_lstp** argument, *CS_csgrp* also returns a pointer to a linked list of *malloc*'ed `cs_Csgrplst_` structures, one per coordinate system in the group. This feature is provided to ease the selection of a single coordinate system from the 1,000 or more that are provided with the CS_MAP distribution.

Given a pointer to a linked list of `cs_Csgrplst_` structures, *CS_csgrpf* will free all memory resources allocated by the linked list.

CS_csgrp returns a zero if no coordinate system definitions we located for the named group, or -1 if the group key name provided is invalid. In both cases, the group list pointer provided by the *a* argument is set to **NULL**.

The list of currently supported groups, and suitable textual descriptions thereof, are provided in a table initialized in the CSdata.c module.

CS_csopn Coordinate System dictionary OPeN

```
csFILE *CS_csopn (Const char *mode);
```

CS_csopn will open the coordinate system dictionary for access as indicated by the **mode** argument (**_STRM_BINRD** or **_STRM_BINWR** as defined in *cs_map.h* for example). The file stream of the open file is returned. Upon successful return, the open file is positioned immediately after the magic number that will have already been verified as being correct.

ERRORS

CS_csopn will return **NULL** and set *cs_Error* appropriately if any of the following conditions are encountered during the update:

cs_CSDICT	The Coordinate System Dictionary file could not be opened. (See *CS_altdr*)
cs_CS_BAD_MAGIC	The file that, by virtue of its name and location, was supposed to be a Coordinate System Dictionary wasn't a Coordinate System Dictionary; its magic number was invalid.

CS_csrd Coordinate System dictionary ReaD

```
int CS_csrd (csFILE *strm,struct cs_Csdef_ *cs_def,int *crypt);
```

CS_csrd reads one record from the open Coordinate System Dictionary file indicated by the file stream **strm** returning the results in the memory buffer pointed to by **cs_def**. The returned entry is always in unencrypted form. **Crypt** is set to **TRUE** if the entry was encrypted in the file; otherwise **crypt** is set to FALSE.

CS_csrd calls *CS_bswap* after reading and decrypting to effect a byte swap, if necessary, to the byte order of the native machine.

CS_csrd will return a value of +1 if a record was successfully read, zero if the end of file was encountered.

ERRORS

CS_csrd will return a -1 and set *cs_Error* appropriately if any of the following conditions are encountered during the update:

cs_IOERR	A physical I/O error was detected during the read operation or *CS_csrd* could not read an entire `cs_Csdef_` structure before encountering the end of file.

CS_csrup Coordinate System Release UPdate

```
int CS_csrup (Const char *distrb,Const char *bkupnm);
```

CS_csrup is designed for use in application installation programs and is used to update a user's Coordinate System Dictionary file. The update is accomplished by updating the user's Coordinate System Dictionary to the current release level, and merging the distribution file with the upgraded user's file. Merging is performed based on coordinate system key name where the distribution version is used wherever duplicate names are encountered.

The **distrb** argument should be the name of the distribution file. If no directory information is present (i.e. no directory separators), the file is expected to reside in the directory indicated by the `cs_Dir` global variable. If directory information is present, the string provided is considered to be a complete path to the distribution file. If **distrb** is the **NULL** pointer or points to the null string, *CS_csrup* simply upgrades the user's existing Coordinate System Dictionary to the current release.

In all cases, *CS_csrup* expects to locate the user's current Coordinate System Dictionary using the standard technique of combining the contents of the `cs_Dir` and `cs_Csname` global variables. If no such file exists, *CS_csrup* creates it and copies the contents of the distribution file to the newly created file.

If the **bkupnm** argument is not the **NULL** pointer and does not point to the null string, *CS_csrup* considers it to be a file name and attempts to rename the user's existing Coordinate System Dictionary to this name before replacing the Coordinate System Dictionary with the newly updated and merged results.

CS_csrup fully supports automatic byte swapping.

CS_csrup writes the new Coordinate System Dictionary to a temporary file, and deletes the existing Coordinate System Dictionary only after all processing has completed successfully. *CS_csrup* returns zero upon success.

ERRORS

CS_csrup will return a -1 and set `cs_Error` appropriately if any of the following conditions are encountered during the update:

cs_IOERR	A physical I/O error was detected during the read operation or *CS_csrup* could not read an entire `cs_Csdef_` structure before encountering the end of file.
cs_FL_OPEN	The open of the distribution file failed.
cs_INV_FILE	Either file was not a valid Coordinate System Dictionary as it did not contain records of the proper size.
cs_CSDEF_MAGIC	One of the files involved did not have the expected magic number in the first 4 bytes of the file.
cs_CS_NOT_FND	*CS_csrup* could not find either the distribution file, or the user's previous Coordinate System Dictionary file, and therefore could not do anything.
cs_NWCS_WRIT	A write error occurred while writing to the new Coordinate System Dictionary. Usually indicates the disk is full.
cs_NOMEM	Heap memory was insufficient to accommodate the allocation of a `cs_Csdef_` structure.
cs_ISER	*CS_csrup* encountered a condition that could only be caused by a coding error in the module itself.

CS_cswr Coordinate System dictionary WRite

```
int CS_cswr (csFILE *strm,Const struct cs_Csdef_ *cs_def,int crypt);
```

CS_cswr writes the coordinate system definition pointed to by the **cs_def** argument at the current position of the open Coordinate System Dictionary file indicated by the file stream **strm**. The definition provided by the **cs_def** argument is always expected to be unencrypted. If the **crypt** argument is non-zero, the definition is encrypted before being written.

CS_cswr calls *CS_bswap* before encrypting and writing to effect a byte swap, if necessary, to little endian byte order.

CS_cswr will return a value of **FALSE** if the record was successfully written, **TRUE** if an error condition was detected.

ERRORS

CS_cswr will return **TRUE** and set `cs_Error` appropriately if any of the following conditions are encountered during the update:

cs_IOERR	A physical I/O error was detected during the write operation.
cs_DISK_FULL	A disk full indication was received as a result of the write attempt.

CS_usrCsDef

```
int CS_usrCsDefPtr (struct cs_Csdef_ *csDef,Const char *keyName);
```

This name, `CS_usrCsDefPtr`, does not refer to a function. Rather, it refers to a global variable which is defined as a pointer to a function which is defined as the above given prototype declares. Applications can use a function as declared above, and the related global pointer variable, to implement specialized coordinate system definitions in a dynamic manner.

If the global variable `CS_usrCsDefPtr` (defined in *CSdata.c*) is not null, the indicated function is called whenever the CS-MAP library is asked to access a specific coordinate system definition. This function, then, can be used to dynamically supply a coordinate system definition which does not exist in the dictionary. Applications can use this to implement their own definition source (i.e. an external database) or dynamically generate such a definition based on the name provided.

CS-MAP passes the **keyName** argument to the hook function prior to any validation, thus dynamic definition names need not adhere to the CS-MAP key name conventions. In the event that the hook function determines that it wishes to supply the definition, the desired definition must be placed in (copied to) the specific structure pointed to by the **csDef** argument.

The hook function returns an integer value:

- -1 is returned to indicate that normal dictionary access function is to return an error condition (i.e. the null pointer). In this case, the hook function must have already reported the specific nature of the error condition using *CS_erpt*.
- +1 is returned to indicate that normal CS-MAP dictionary access is to be performed.
- 0 is returned to indicate that the hook function has supplied a definition that is to be used. In this case, CS-MAP will allocate new memory from the heap, copy the hook function supplied definition to the allocated memory, and return a pointer to the allocated memory to the calling function.

CS_dtDictCls DaTum DICTionary file CLoSe

void CS_dtDictCls (csFILE* stream);

Applications must use this function to close a stream opened with the *CS_dtopn* function. Using this function assures that any memory of the open stream used for deferred close is erased.

CS_elDictCls ELlipsoid DICTionary file CLoSe

void CS_elDictCls (csFILE* stream);

Applications must use this function to close a stream opened with the *CS_elopn* function. Using this function assures that any memory of the open stream used for deferred close is erased.

CS_dtcmp DaTum definition CoMPare

```
int CS_dtcmp (Const struct cs_Dtdef_ *pp,Const struct cs_Dtdef_*qq);
```

CS_dtcmp compares the two datum definition structures provided to it by **pp** and **qq** and returns an integer which represents the collating sequence relationship between the two datum definitions. The collating sequence is based on the key name of the datum definition. This function is used rather than *strcmp* as it can compare encrypted entries as well as unencrypted entries.

This function is used in conjunction with *CS_ips* and *CS_bins* to access datum definitions in the Datum Dictionary.

CS_dtfnm DaTum dictionary File NaMe

```
int CS_dtfnm (Const char *new_name);
```

CS_dtfnm changes the name used by CS-MAP when opening the Datums Dictionary to that specified by **new_name**. The directory that is searched remains the same. Use *CS_altdr* to change the directory.

CS_dtopn DaTum dictionary OPeN

```
csFILE *CS_dtopn (Const char *mode);
```

CS_dtopn will open the Datum Dictionary for access as indicated by the **mode** argument (**_STRM_BINRD** or **_STRM_BINWR** as defined in *cs_map.h* for example). The file stream of the open file is returned. Upon successful return, the open file is positioned immediately after the magic number that will have already been verified as being correct.

ERRORS

CS_dtopn will return **NULL** and set `cs_Error` appropriately if any of the following conditions are encountered during the update:

cs_DTDICT	The Datum Dictionary file could not be opened. (See *CS_altdr*).
cs_DT_BAD_MAGIC	The file that, by virtue of its name and location, was supposed to be a Datum Dictionary wasn't a Datum Dictionary; its magic number was invalid.

CS_dtrd DaTum dictionary ReaD

```
int CS_dtrd (csFILE *strm,struct cs_Dtdef_ *dt_def,int *crypt);
```

CS_dtrd reads one record from the open Datum Dictionary file indicated by the file stream **strm** returning the results in the memory buffer pointed to by **dt_def**. The returned entry is always in unencrypted form. **Crypt** is set to **TRUE** if the entry was encrypted in the file; otherwise, **crypt** is set to **FALSE**.

CS_dtrd calls *CS_bswap* to effect a byte swap, if necessary, to the byte ordering of the native machine.

CS_dtrd will return a value of +1 if a record was successfully read, zero if the end of file was encountered.

ERRORS

CS_dtrd will return a -1 and set `cs_Error` appropriately if any of the following conditions are encountered during the update:

cs_IOERR	A physical I/O error was detected during the read operation or *CS_dtrd* could not read an entire `cs_Dtdef_` structure before encountering the end of file.

CS_dtrup DaTum dictionary Release UPdate

```
int CS_dtrup (Const char *distrb,Const char *bkupnm);
```

CS_dtrup is designed for use in application installation programs and is used to update a user's Datum Dictionary file. The update is accomplished by updating the user's Datum Dictionary to the current release level, and merging the distribution file with the upgraded user's file. Merging is performed based on datum key name where the distribution version is used wherever duplicate names are encountered.

The **distrb** argument should be the name of the distribution file. If no directory information is present (i.e. no directory separators), the file is expected to reside in the directory indicated by the `cs_Dir` global variable. If directory information is present, the string provided is considered to be a complete path to the distribution file. If **distrb** is the **NULL** pointer or points to the null string, *CS_dtrup* simply upgrades the user's existing Datum Dictionary to the current release.

In all cases, *CS_dtrup* expects to locate the user's current Datum Dictionary using the standard technique of combining the contents of the `cs_Dir` and `cs_Dtname` global variables. If no such file exists, *CS_dtrup* creates it and copies the contents of the distribution file to the newly created file.

If the **bkupnm** argument is not the **NULL** pointer and does not point to the null string, *CS_dtrup* considers it to be a file name and attempts to rename the user's existing Datum Dictionary to this name before replacing the Datum Dictionary with the newly updated and merged results.

CS_dtrup fully supports automatic byte swapping.

CS_dtrup writes the new Datum Dictionary to a temporary file, and deletes the existing Datum Dictionary only after all processing has completed successfully. *CS_dtrup* returns zero upon success.

ERRORS

CS_dtrup will return a -1 and set `cs_Error` appropriately if any of the following conditions are encountered during the update:

cs_IOERR	A physical I/O error was detected during the read operation or *CS_dtrup* could not read an entire `cs_Dtdef_` structure before encountering the end of file.
cs_FL_OPEN	The open of the distribution file failed.
cs_INV_FILE	Either file was not a valid Datum Dictionary as it did not contain records of the proper size.
cs_DTDEF_MAGIC	One of the files involved did not have the expected magic number in the first 4 bytes of the file.
cs_DT_NOT_FND	*CS_dtrup* could not find either the distribution file, or the user's previous Datum Dictionary file, and therefore could not do anything.
cs_NWDT_WRIT	A write error occurred while writing to the new Datum Dictionary. Usually indicates the disk is full.
cs_NOMEM	Heap memory was insufficient to accommodate the allocation of a `cs_Dtdef_` structure.
cs_ISER	*CS_dtrup* encountered a condition that could only be caused by a coding error in the module itself.

CS_dtwr DaTum dictionary WRite

```
int CS_dtwr (csFILE *strm,Const struct cs_Dtdef_ *dt_def,int crypt);
```

CS_dtwr writes the datum definition pointed to by the **dt_def** argument to the current position of the Datum Dictionary file indicated by the file stream **strm**. The datum definition provided is always expected to be in unencrypted form. If **crypt** is non-zero, the definition is encrypted before being written to the Datum Dictionary.

CS_dtwr calls *CS_bswap* prior to encrypting and writing to effect a byte swap, if necessary, to little endian byte order ala Intel/DOS.

CS_dtwr will return a value of zero if the definition was successfully written, -1 if an error condition was detected.

ERRORS

CS_dtwr will return a -1 and set `cs_Error` appropriately if any of the following conditions are encountered during the update:

cs_IOERR	A physical I/O error was detected during the write operation.

CS_usrDtDefPtr - Datum Definition Hook Function

```
int CS_usrDtDefPtr (struct cs_Dtdef_ *dtDef,Const char *keyName);
```

This name, `CS_usrDtDefPtr`, does not refer to a function. Rather, it refers to a global variable which is defined as a pointer to a function which is defined as the above given prototype declares. Applications can use a function as declared above, and the related global pointer variable, to implement specialized datum definitions in a dynamic manner.

If the global variable `CS_usrDtDefPtr` (defined in *CSdata.c*) is not null, the indicated function is called whenever the CS-MAP library is asked to access a specific datum definition. This function, then, can be used to dynamically supply a datum definition which does not exist in the dictionary. Applications can use this to implement their own definition source (i.e. an external database) or dynamically generate such a definition based on the name provided.

CS-MAP passes the **keyName** argument to the hook function prior to any validation, thus dynamic definition names need not adhere to the CS-MAP key name conventions. In the event that the hook function determines that it wishes to supply the definition, the desired definition must be placed in (copied to) the specific structure pointed to by the **dtDef** argument.

The hook function returns an integer value:

- -1 is returned to indicate that normal dictionary access function is to return an error condition (i.e. the null pointer). In this case, the hook function must have already reported the specific nature of the error condition using *CS_erpt*.
- +1 is returned to indicate that normal CS-MAP dictionary access is to be performed.
- 0 is returned to indicate that the hook function has supplied a definition that is to be used. In this case, CS-MAP will allocate new memory from the heap, copy the hook function supplied definition to the allocated memory, and return a pointer to the allocated memory to the calling function.

CS_elcmp ELlipsoid definition CoMPare

```
int CS_elcmp (Const struct cs_Eldef_ *pp,Const struct cs_Eldef_*qq);
```

CS_elcmp compares the two ellipsoid definition structures provided to it by **pp** and **qq** and returns an integer that represents the collating sequence relationship between the two ellipsoid definitions. The collating sequence is based on the key name of the ellipsoid definition. This function is used rather than *strcmp* as it can compare encrypted entries as well as unencrypted entries.

This function is used in conjunction with *CS_ips* and *CS_bins* to access ellipsoid definitions in the Ellipsoid Dictionary.

CS_elfnm ELlipsoid dictionary File NaMe

```
int CS_elfnm (Const char *new_name);
```

CS_elfnm changes the name used by CS-MAP when opening the Ellipsoid Dictionary to that specified by **new_name**. The directory that is searched remains the same. Use *CS_altdr* to change the directory.

CS_elopn ELlipsoid dictionary OPeN

```
csFILE *CS_elopn (Const char *mode);
```

CS_elopn will open the Ellipsoid Dictionary for access as indicated by the **mode** argument (**_STRM_BINRD** or **_STRM_BINWR** as defined in *cs_map.h* for example). The file stream of the open file is returned. Upon successful return, the open file is positioned immediately after the magic number that will have already been verified as being correct.

ERRORS

CS_elopn will return **NULL** and set `cs_Error` appropriately if any of the following conditions are encountered while opening the file:

cs_ELDICT	The Ellipsoid Dictionary file could not be opened. (See *CS_altdr*).
cs_EL_BAD_MAGIC	The file that, by virtue of its name and location, was supposed to be an Ellipsoid Dictionary wasn't an Ellipsoid Dictionary; its magic number was invalid.

CS_elrd ELlipsoid dictionary ReaD

```
int CS_elrd (csFILE *strm,struct cs_Eldef_ *el_def,int *crypt);
```

CS_elrd reads one record from the open Ellipsoid Dictionary file indicated by the file stream **strm** returning the results in the memory buffer pointed to by **el_def**. The returned entry is always in

unencrypted form. **Crypt** is set to **TRUE** if the entry was encrypted in the file; otherwise, **crypt** is set to **FALSE**.

CS_elrd calls *CS_bswap* after reading and decrypting to effect, if necessary, a byte swap to the byte ordering of the native machine.

CS_elrd will return a value of +1 if a record was successfully read, zero if the end of file was encountered.

ERRORS

CS_elrd will return a -1 and set `cs_Error` appropriately if any of the following conditions are encountered during the update:

cs_IOERR	A physical I/O error was detected during the read operation or *CS_elrd* could not read an entire `cs_Eldef_` structure before encountering the end of file.

CS_elrup ELipsoid dictionary Release UPdate

```
int CS_elrup (Const char *distrb,Const char *bkupnm);
```

CS_elrup is designed for use in application installation programs and is used to update a user's Ellipsoid Dictionary file. The update is accomplished by updating the user's Ellipsoid Dictionary to the current release level, and merging the distribution file with the upgraded user's file. Merging is performed based on ellipsoid key name where the distribution version is used wherever duplicate names are encountered.

The **distrb** argument should be the name of the distribution file. If no directory information is present (i.e. no directory separators), the file is expected to reside in the directory indicated by the `cs_Dir` global variable. If directory information is present, the string provided is considered to be a complete path to the distribution file. If **distrb** is the **NULL** pointer or points to the null string, *CS_elrup* simply upgrades the user's existing Ellipsoid Dictionary to the current release.

In all cases, *CS_elrup* expects to locate the user's current Ellipsoid Dictionary using the standard technique of combining the contents of the `cs_Dir` and `cs_Elname` global variables. If no such file exists, *CS_elrup* creates it and copies the contents of the distribution file to the newly created file.

If the **bkupnm** argument is not the **NULL** pointer and does not point to the null string, *CS_elrup* considers it to be a file name and attempts to rename the user's existing Ellipsoid Dictionary to this name before replacing the Ellipsoid Dictionary with the newly updated and merged results.

CS_elrup fully supports automatic byte swapping.

CS_elrup writes the new Ellipsoid Dictionary to a temporary file, and deletes the existing Ellipsoid Dictionary only after all processing has completed successfully. *CS_elrup* returns zero upon success.

ERRORS

CS_elrup will return a -1 and set `cs_Error` appropriately if any of the following conditions are

encountered during the update:

cs_IOERR	A physical I/O error was detected during the read operation or *CS_elrup* could not read an entire `cs_Eldef_` structure before encountering the end of file.
cs_FL_OPEN	The open of the distribution file failed.
cs_INV_FILE	Either file was not a valid Ellipsoid Dictionary as it did not contain records of the proper size.
cs_ELDEF_MAGIC	One of the files involved did not have the expected magic number in the first 4 bytes of the file.
cs_EL_NOT_FND	*CS_elrup* could not find either the distribution file, or the user's previous Ellipsoid Dictionary file, and therefore could not do anything.
cs_NWEL_WRIT	A write error occurred while writing to the new Ellipsoid Dictionary. Usually indicates the disk is full.
cs_NOMEM	Heap memory was insufficient to accommodate the allocation of a `cs_Eldef_` structure.
cs_ISER	*CS_elrup* encountered a condition that could only be caused by a coding error in the module itself.

CS_elwr ELlipsoid dictionary WRite

```
int CS_elwr (csFILE *strm,Const struct cs_Eldef_ *el_def,int crypt);
```

CS_elwr writes the ellipsoid definition pointed to by the **el_def** argument to the current position of the Ellipsoid Dictionary file indicated by the file stream **strm**. The ellipsoid definition provided is always expected to be in unencrypted form. If **crypt** is non-zero, the definition is encrypted before being written to the Ellipsoid Dictionary.

CS_elwr calls *CS_bswap* before writing the data. This effects a byte swap, if necessary, to write the ellipsoid definition in little endian (i.e. Intel/DOS) byte order.

CS_elwr will return a value of **FALSE** if the definition was successfully written, **TRUE** if an error condition was detected.

ERRORS

CS_elwr will return a -1 and set `cs_Error` appropriately if any of the following conditions are encountered during the update:

cs_IOERR	A physical I/O error was detected during the write operation.
cs_DISK_FULL	A disk full condition was detected during the write operation.

CS_usrElDefPtr - Ellipsoid Definition Hook Function

```
int CS_usrElDefPtr (struct cs_Eldef_ *elDef,Const char *keyName);
```

This name, `CS_usrElDefPtr`, does not refer to a function. Rather, it refers to a global variable which is defined as a pointer to a function which is defined as the above given prototype declares. Applications can use a function as declared above, and the related global pointer variable, to implement specialized ellipsoid definitions in a dynamic manner.

If the global variable `CS_usrElDefPtr` (defined in *CSdata.c*) is not null, the indicated function is called whenever the CS-MAP library is asked to access a specific ellipsoid definition. This function, then, can be used to dynamically supply a ellipsoid definition which does not exist in the dictionary. Applications can use this to implement their own definition source (i.e. an external database) or dynamically generate such a definition based on the name provided.

CS-MAP passes the **keyName** argument to the hook function prior to any validation, thus dynamic definition names need not adhere to the CS-MAP key name conventions. In the event that the hook function determines that it wishes to supply the definition, the desired definition must be placed in (copied to) the specific structure pointed to by the **elDef** argument.

The hook function returns an integer value:

- -1 is returned to indicate that normal dictionary access function is to return an error condition (i.e. the null pointer). In this case, the hook function must have already reported the specific nature of the error condition using *CS_erpt*.
- +1 is returned to indicate that normal CS-MAP dictionary access is to be performed.
- 0 is returned to indicate that the hook function has supplied a definition that is to be used. In this case, CS-MAP will allocate new memory from the heap, copy the hook function supplied definition to the allocated memory, and return a pointer to the allocated memory to the calling function.

Well Known Text Implementation

While the syntax used by Well Known Text (WKT) represents a reliable standard, the manner in which is anything but standard. There are six flavors of WKT that the developers of CS-MAP have identified; undoubtedly there are more. CS-MAP currently includes reliable data for the six flavors it knows about; however significant data sets for only three flavors have been found; thus serious testing has been limited to those three flavors:

1. ESRI
2. Oracle

3. GeoTools

Thus, WKT remains a work in progress and probably will so remain for years to come.

Please note that the major issue with WKT is that most all WKT definitions do not include reliable data concerning the datum in use. Yes, the datum is named, but that is all we have to go on. Thus, successful translation of a WKT definition relies heavily on being able to map the datum name (and indirectly the ellipsoid name) to a known definition. Known definition in this case means a definition currently existing in the CS-MAP Datum DIctionary.

Successful conversion of WKT definitions relies heavily on:

1. identifying the flavor of the WKT definition being processed.
2. mapping the datum name used by a specific flavor to a CS-MAP datum definition.
3. mapping the projection name used in the WKT definition to a CS-MAP projection.
4. mapping the various parameter names used in the WKT definition to CS-MAP projection parameters.

This there is a lot of room for failure and errors of all sorts. As of this release (11.15) most all WKT definitions for which a corresponding EPSG definition exists, and both the projection and datum transformation technique are supported by CS-MAP, are handled properly. Fortunately, this includes most all of the widely used coordinate systems, datums, and ellipsoids.

The subsections which follow describe the functions implemented within CS-MAP to process WKT definitions.

Objects/Functions Implemented in C++

The functions and object described in this section are implemented in C++. That is, they implement a C++ object in the form of a class, or they use C++ features such as std::istream or std::string. Obviously, these can only be compiled by a compiler capable of compiling C++ in addition to standard 'C'. Fortunately, that includes most all compilers today.

The C++ code in all of these modules is encapsulated within a conditional compile controlled by the __CPP__ preprocessor constant. Developers will need to arrange to have this constant defined while compiling the code modules containing the implementation of these objects and functions.

TrcWktElement Object

CS-MAP implements the WKT structure as a C++ object named TrcWktElement. Such an element is used to represent the various WKT entities, and is designed such that it can contain sub-elements of itself. A TrcWktElement object representing a Datum, for example, will contain an instance of a TrcWktElement which represents a Spheroid. In this way, the parsing and maintenance of a wide variety of WKT strings can be defined by a single object.

Note that TrcWktElement is a C++ object (i.e. a class). The implementation of this object is encapsulated with a conditional compile controlled by the **__CPP__** preprocessor constant. Thus, to implement the WKT feature one must be using a compiler capable of compiling C++ code, and arrange to have the **__CPP__** preprocessor constant defined before compiling.

The name of this object deviates substantially from the CS-MAP standard naming conventions. This is the case as it was originally coded for use in a new product which uses an enhanced naming convention.

TrcWktElement - Parameters

The following parameters are used by the member functions of the *TrcWktElement* object:

Name	Type	Description
type	ErcWktEleType	The WKT type of the element.
value	char*	The value of the element. Not the name, but the value, usually numeric.
name	char*	The name associated with several elements.
flavor	ErcWktFlavor	The flavor of the WKT string to be parsed.
csMapParamCode	short	A CS-MAP parameter code, ala cs_PRMCOD_????
parent	TrcWktElement*	Pointer to the parent WKT element of the current WKT element.
bufr	char*	Character array in which WKT values are returned as a null terminated character array.
fieldNbr	size_t	The zero based index of a value to be extracted from the WKT string. For those elements which require a name, the name is not considered a value.
childElement	TrcWktElement*	Pointer to a new WKT element object that is to be added as a child to the current WKT element.
toBeRemoved	TrcWktElement*	Pointer to a specific child element which is to be removed from its parent element.

eleStart	size_t	Index into the WKT string being parsed of the first character of the extracted WKT element (in string form).
eleTerm	size_t	Index into the WKT string being parsed of the last character of the extracted WKT element (in string form).
bufrSize	size_t	The size of a character array in which results are to be returned.
wellKnownText	char*	A Well Known Text definition in a null terminated character array form. Generally, the character string may have other text preceding and following the actual WKT definition.

TrcWktElement -- Construction / Destruction / Assignment

Constructors

```
TrcWktElement (void);
```

Constructs an empty object; an object with wktTypNOne, null name, and no values.

```
TrcWktElement (ErcWktEleType type,const char *value);
```

Constructs and object of the indicated type with a single value.

```
TrcWktElement (ErcWktEleType type,const char *name,const char *value);
```

Constructs an object of the given type, with the provided name, and a single value.

```
TrcWktElement (ErcWktEleType type,const std::string& value);
```

Constructs an object of the indicated type with a single value.

```
TrcWktElement (ErcWktEleType type,const std::string& name,const std::string&
value);
```

Constructs an object of the indicated type, with the provided name and a single value.

```
TrcWktElement (const char *wellKnownText);
```

Constructs an object by parsing the provided WKT string. The result has a type indicated by the preamble to the WKT string provided, and the value is set to the contents of the bracketed string which follows the preamble. If the type is one of those which require a name, the name is extracted, and removed, from the value.

```
TrcWktElement (const TrcWktElement& source);
```

Copy constructor; nothing exceptional of note here.

Destructor

```
~TrcWktElement (void);
```

No surprises here.

Assignment

```
TrcWktElement& operator= (const TrcWktElement& source);
```

No surprises here.

TrcWktElement Member Functions

```
ErcWktEleType GetElementType (void) const;
```

Returns the WKT type of the element.

```
ErcWktEleType GetParentType (void) const;
```

Returns the WKT type of the parent element.

```
std::string GetElementName (void) const;
```

Returns the name of the element. For example, the spheroid name of a SPHEROID type element as a string.

```
const char* GetElementNameC (void) const;
```

Returns the name of the element as a character pointer.

```
std::string GetCompleteValue (void) const;
```

Returns the complete value of the element as a string. The is, the entire contents of the unparsed bracketed value of the element.

```
bool HasInitialName (void) const;
```

Returns true if the type of this element is one that requires an initial name.

```
void SetParent (const TrcWktElement* parent);
```

Sets the parent of the element to that provided. Used by the parent object when parsing out the child elements. Probably should be protected, but it isn't.

```
void SetParentType (ErcWktEleType type);
```

Sets the type of the parent of the element to that provided. Used by the parent object when parsing out the child elements. Probably should be protected, but it isn't.

```
const TrcWktElement *ChildLocate (ErcWktEleType type) const;
```

Returns a pointer to the first child element of the type provided. Returns null pointer if no child of the type requested was found.

```
const TrcWktElement *ChildLocate (ErcWktEleType type,size_t& index) const;
```

Returns a pointer to the first child element of the type requested. Search starts at the element indicated by index. Index is set to the where a subsequent search should start. Returns null pointer if no such type was found.

const TrcWktElement *ChildSearch (ErcWktEleType type) const;

Searches for a child element of the requested type. Immediate child elements are searched first. If no element is located, each child is then searched for a child element of the requested type. The search is recursive, to the lowest level. Null pointer is returned if no child is found.

const TrcWktElement *ParameterLocate (ErcWktFlavor flavor,short csMapParamCode) const;

Searches all child elements of the PARAMETER type for a parameter with the indicated parameter type. Flavor indicates the set of parameter names to be used in the search.

```
void GetFieldChar (char *bufr,size_t bufrSize,size_t fieldNbr) const;
```

Returns the value, as a character pointer. of the value indicated by fieldNbr. Returns null pointer if such a value does not exist.

```
std::string GetFieldStr (size_t fieldNbr) const;
```

Returns the value, as a std::string. of the value indicated by fieldNbr. Returns null pointer if such a value does not exist.

```
double GetFieldDouble (size_t fieldNbr) const;
```

Returns the indicated value as a double. Returns zero if fieldNbr is invalid. Conversion to double accomplished via atof.

```
long GetFieldLong (size_t fieldNbr) const;
```

Returns the indicated value as a long. Returns zero if fieldNbr is invalid. Conversion to double accomplished via atol.

```
ErcWktAxisId GetAxisId (void) const;
```

Returns an axis id value as indicated by the name of the element. Returns rcWktAxisIdNone if value is invalid or type of element is not AXIS.

```
ErcWktAxisValue GetAxisValue (void) const;
```

Returns an axis value as indicated by the value of the element. Returns rcWktAxisNone if value is invalid or type of element is not AXIS.

```
ErcWktFlavor DetermineFlavor (void) const;
```

Attempts to determine the flavor of a fully parsed element. Returns wktFlvrUnknown if not successful.

```
void AddChild (const TrcWktElement& childElement);
```

Adds a child WKT element to the current element.

```
void RemoveChild (const TrcWktElement* toBeRemoved);
```

Removes a child element from the current element.

```
void ReconstructValue (void);
```

Reconstructs the value of the current element from the name, values, and child elements of the current element. Used when converting CS-MAP definitions to WKT.

```
std::string ProduceWkt (void) const;
```

Returns a WKT representation of the current element.

```
void ParseChildren (void);
```

Parses the current value of the element, producing and adding child elements as necessary. Process is recursive.

WKT Object Support

The functions described in this section implement CS-MAP's ability to convert definitions from/to WKT representation. While all of these functions are declared with standard 'C' linkage, many interact with the C++ TrcWktElement object. Therefore, a C++ compiler is required to compile these functions and the code implementing most of these function is encapsulated in a conditional compile segment controlled by the **__CPP__** preprocessor constant.

CS_isWkt IS Well Known Text

```
int CS_isWkt (const char *wellKnownText);
```

CS_isWkt will return a non zero value if it considers it very likely that the string provided by the **wellKnownText** argument is a WKT definition. This determination is mad by counting matching square bracket characters. If the left and right bracket counts are equal and non-zero, the string is assumed to be a WKT string. A zero is returned if the string fails this simple test.

CS_wktToCs Well Known Text To Coordinate System

```
int CS_wktToCs (struct cs_Csdef_ *csDef,struct cs_Dtdef_ *dtDef,
                    struct cs_Eldef_ *elDef,
                    ErcWktFlavor flavor,
                    const char *wellKnownText)
```

CS_wktToCs converts the Well Known Text (WKT) provided by the **wellKnownText** argument to the form used internally by CS-MAP. Use the **flavor** argument to indicate which flavor (there are several) of WKT is being processed. The results are returned in the structures pointed to by the **csDef**, **dtDef**, and **elDef** arguments. The **dtDef** and **elDef** arguments may be null.

CS_wktToCs will always copy the full name of the PROJCS element to the `desc_nm` member of the `cs_Csdef_` structure (and the name member of the `cs_Dtdef_` and `cs_Eldef_` structures). It will attempt to establish a keyname of 23 characters or less for each of these structures. If an AUTHORITY element is present in the PROJCS (and DATUM and SPHEROID sub-elements) being processed, the EPSG code will be used to manufacture a keyname of the form "Epsg:nnnn" where the actual code value replaces the n's. If, for whatever reason, the manufacturing of a keyname of 23 characters or less fails, *CS_wktToCs* will return a positive, non-zero, value.

It is the intent of the design that this function, and it close relatives, be dependent solely on the WKT information provided by the **wellKnownText** argument. Additional functions (see *CS_wktToCsEx*) are provided which attempt to enhance the information produced by this function given access to mapping tables and the CS-MAP dictionaries.

A zero return status, which is rare, indicates complete success in parsing an conversion of the three required elements. A non-zero positive status is returned in the case of partial success. The value returned is a bit map of the following conditions:

Preprocessor Constant	Value	Description
cs_EL2WKT_NMTRUNC	1	Ellipsoid key name truncated
cs_DT2WKT_NMTRUNC	2	Datum key name truncated
cs_CS2WKT_NMTRUNC	4	Coordinate system key name truncated
cs_DT2WKT_DTDEF	8	Datum definition extracted from CS-MAP dictionary by name.
cs_DT2WKT_NODEF	16	No datum definition is present in dtDef.

CS_wktToCs will return a -1 and set `cs_Error` appropriately if any of the following conditions are encountered during the conversion:

`cs_WKT_WRNGTYP` The type of WKT element provided to the function was was not that of PROJCS. (Use CS_wktToDt to process GEOGTRAN objects.)

`cs_WKT_NOUNIT` A linear unit specification for the PROJCS definition was not found.

`cs_WKT_INVUNIT` The linear unit extracted for the PROJCS was not one recognized by CS_MAP.

`cs_WKT_NOGEOCS` *CS_wktToCs* could not locate a GEOGCS element within the PROJCS element. A GEOGCS is required in order to determine datum/ellipsoid.

`cs_WKT_NOGUNIT` The angular unit of the internal GEOGCS could not be located.

`cs_WKT_INVGUNIT` The angular unit extracted from the internal GEOGCS definition was not one recognized by CS-MAP.

`cs_WKT_NOPROJ` *CS_wktToCs* could not locate a PROJECTION specification in the PROJCS definition.

`cs_WKT_INVPROJ`	The PROJECTION specification located in the PROJCS definition specified a projection that is not supported by CS-MAP (or otherwise unrecognized).
`cs_WKT_NODATUM`	A DATUM specification within the internal GEOGCS element is required, and *CS_wktToCs* could not locate same.
`cs_WKT_NOELLIP`	A SPHEROID specification within the DATUM specification is required, and *CS_wktToCs* could not locate same.

Whenever a negative value is returned, You may use *CS_errmsg* to obtain a textual description of the problem.

CS_wktToCsEx Well Known Text To Coordinate System EXtended

```
int CS_wktToCsEx (struct cs_Csdef_ *csDef,struct cs_Dtdef_ *dtDef,
                                          struct cs_Eldef_ *elDef,
                                          ErcWktFlavor flavor,
                                          const char *wellKnownText)
```

CS_wktToCsEx converts the Well Known Text (WKT) provided by the **wellKnownText** argument to the form used internally by CS-MAP. Use the **flavor** argument to indicate which flavor (there are several) of WKT is being processed. The results are returned in the structures pointed to by the **csDef**, **dtDef**, and **elDef** arguments. If the **flavor** argument is set to **wktFlvrNone**, *CS_wktToCsEx* will attempt to determine the flavor automatically. An negative status value is returned if this flavor determination fails. The **dtDef** and **elDef** arguments may be null.

CS_wktToCsEx attempts to improve the results of *CS_wktToCs* by using definition name mapping tables to resolve issues encountered during the WKT parsing process. Using the flavor specification, it attempts to map the names of the coordinate system, the datum, and the ellipsoid contained in the WKT to known CS-MAP definitions. In so doing, many of the issues regarding processing of WKT (such as missing datum information) are often resolved.

CS_wktToCsEx returns zero on success, indicating that key nams for all three elements have been successfullly mapped to CS_MAP names and valid definitions exist in the CS-MAP dictionary for all three names. In the event of partial success, non-zero positive status is returned which is a bit map of the following conditions:

Preprocessor Constant	Value	Description
cs_EL2WKT_NMTRUNC	1	Ellipsoid key name truncated
cs_DT2WKT_NMTRUNC	2	Datum key name truncated
cs_CS2WKT_NMTRUNC	4	Coordinate system key name truncated
cs_DT2WKT_DTDEF	8	Datum definition extracted from CS-MAP dictionary by name.
cs_DT2WKT_NODEF	16	No datum definition is present in dtDef.

In the evnt of failure, CS_wktToCs will return a -1 and set `cs_Error` appropriately if any of the following conditions are encountered during the conversion:

`cs_WKT_BADFORM`	`The provided WKT was not parsable. That is it did not adhere to WKT syntax as understood by CS-MAP.`
`cs_WKT_FLAVOR`	`The` **flavor** `argument value` **wktFlvrNone** `and` *CS_wktToCsEx* `could not determine the flavor of the WKT entry provided.`
`cs_WKT_DTMAP`	`No datum transformation information was provided in the WKT entry provided and` *CS_wktToCsEx* `was unable to determine an appropriate CS-MAP datum definition to use.`
`cs_WKT_WRNGTYP`	The type of WKT element provided to the function was was not that of PROJCS or GEOGCS. (Use *CS_wktToDt* to process GEOGTRAN objects.)
`cs_WKT_NOUNIT`	A linear unit specification for the PROJCS definition was not found.
`cs_WKT_INVUNIT`	The linear unit extracted for the PROJCS was not one recognized by CS_MAP.
`cs_WKT_NOGEOCS`	*CS_wktToCs* could not locate a GEOGCS element within the PROJCS element. A GEOGCS is required in order to determine datum/ellipsoid.
`cs_WKT_NOGUNIT`	The angular unit of the internal GEOGCS could not be located.
`cs_WKT_INVGUNIT`	The angular unit extracted from the internal GEOGCS definition was not one recognized by CS-MAP.
`cs_WKT_NOPROJ`	*CS_wktToCs* could not locate a PROJECTION specification in the PROJCS definition.
`cs_WKT_INVPROJ`	The PROJECTION specification located in the PROJCS definition specified a projection that is not supported by CS-MAP (or otherwise unrecognized).
`cs_WKT_NODATUM`	A DATUM specification within the internal GEOGCS element is required, and *CS_wktToCs* could not locate same.
`cs_WKT_NOELLIP`	A SPHEROID specification within the DATUM specification is required, and *CS_wktToCs* could not locate same.

Whenever a negative value is returned, You may use *CS_errmsg* to obtain a textual description of the problem.

CS_wktToDt Well Known Text To DaTum

```
int CS_wktToDt (struct cs_Dtdef_ *dtDef,struct cs_Eldef_ *elDef,
                                        ErcWktFlavor flavor,
```

```
                                        const char *wellKnownText);
```

CS_wktToDt will parse the Well Known Text provided by the **wellKnownText** argument and populate the pre-existing structures provided by the **dtDef** and **elDef** arguments; neither of which may be the null pointer. If the **flavor** argument is set to **wktFlvrNone**, *CS_wktToDt* will attempt to determine the flavor for you. Otherwise, it will use the supplied flavor to interpret transformation method and parameter names (which are, of course, non-standard).

CS_wktToDt converts a WKT GEOTRANS object to CS-MAP format in the form of a *cs_Dtdef_* and a *cs_Eldef_* structure. If you have the newer format where the Datum element in the PROJCS element has the TOWGS84 element embedded in it, simply use *CS_wktToCs* and convert the whole mess in one shot. Use this function only when dealing with the older GEOTRAN WKT type string.

CS_wktToDt returns a positive value upon successful conversion. A non-zero positive result is a bit map of the following conditions:

Preprocessor Constant	Value	Description
cs_EL2WKT_NMTRUNC	1	Ellipsoid key name truncated

CS_wktToDt will return a negative value and set cs_Error accordingly should any of the following conditions be encountered:

cs_WKT_FLAVOR	The flavor of the provided Well Known Text string could not be determined.
cs_WKT_GEOGCNT	A valid GEOGTRAN WKT definition requires exactly two GEOGCS specifications; the source and target systems. CS_wktTodT encountered less than, or more than, two such defintiions.
cs_WKT_WRNGTRG	CS-MAP requires that the target datum specification be WGS84. IN the WKT string provided, a target other than WGS84 was encountered.
cs_WKT_NOSRCDT	The source GEOGCS definition did not contain a datum specification, or it was not parseable.
cs_WKT_NOMETH	A transformation METHOD specificationcould not be located in the provided WKT string.
cs_WKT_MTHERR	While a METHOD specification did exist, the actual method specified was not one of the method names understood, or supported, by CS-MAP.

cs_WKT_WRNGTYP	The type of WKT element provided to the function was was not that of PROJCS. (Use CS_wktToDt to process GEOGTRAN objects.)
cs_WKT_NOELLIP	A SPHEROID specification within the DATUM specification is required, and CS_wktToCs could not locate same.

Whenever a negative return value is returned, you can use *CS_errmsg* to obtain a textual description of the cause.

CScs2Wkt

```
int CScs2Wkt (char *csWktBufr,size_t bufrSize,struct cs_Csdef_ *cs_def)
```

Use *CScs2Wkt* to produce a complete PROJCS or GEOGCS object in the Well Known Text (WKT) format. *CScs2Wkt* will access the Datum and Ellipsoid Dictionaries as necessary to retrieve the information necessary to complete this request. The result is returned in the buffer pointed to by the **csWktBufr** argument, but no more than **bufrSize** byte will be written there. The **cs_def** argument provides the definition which is to be converted.

If the projection upon which the provided definition is based is the Unity projection, a GEOGCS object is produced. Otherwise a PROJCS object is produced. CScs2Wkt returns a negative value in the event of an error. Attempting to convert a coordinate system definition based on a projection which is not supported by the WKT format is the most common cause of an error.

CS_wktToDict Well Known Text To DICTionary

```
int EXP_LVL1 CS_wktToDict (const char *csKeyName,const char *dtKeyName,
                                                 const char *elKeyName,
                                                 const char *wellKnownText,
                                                 int flavor);
```

CS_wktToDict will parse the Well Known Text string provided by the **wellKnownText** argument and, if successful, add the resulting components to the CS-MAP dictionaries. Use the **flavor** argument to indicate the flavor of the Well Known Text that is to be parsed; *CS_wktToDict* will not determine the flavor for you.

Upon addition, the new entries in the dictionary will be assigned key names as provided by the **csKeyName**, **dtKeyName**, and **elKeyName** arguments respectively. Should any of the key name arguments be the null pointer, or point to the null string, that specific dictionary update will not be attempted. Note that these dictionary modifications are updates, and will replace existing definitions with the same name.

A dictionary update will fail, and cause subsequent dictionary updates to be skipped, if *CS_wktToDict* attempts to replace a protected definition.

CS_wktToDict will return a -1 upon failure for any reason. Use *CS_errmsg* to obtain a textual description of the cause of the failure. Note that *CS_wktToDict* uses *CS_wktToCs* to parse the provided Well Known Text string, and *CS_elupd*, *CS_dtupd*, and *CS_csupd* (in that order) to update the dictionaries. See these functions for a complete list of possible failure conditions.

Name/Number mapping Functions

The several functions in this section are used by the WKT processing facility to map definition names and ID codes between CS-MAP, EPSG, and the various flavors of WKT. These are straight 'C' functions, are not linked to the C++ implementation of the WKT object. Therefore, they can of general value to all applications. New mapping functions are added quite regularly, so checking the *cs_map.h* header file for new prototypes on each release is advised.

CS_epsg2msi EPSG code to MSI key name

```
int CS_epsg2msi (long epsgNbr,char* msiKeyName,int size);
```

CS_epsg2msi returns in the character array provided by the ***msiKeyName*** argument the CS-MAP coordinate system key name which corresponds to the EPSG code value provided by the ***epsgNbr*** argument. *CS_epsg2msi* returns a zero for success, a -1 if the ***epsgNbr*** argument was not a valid EPSG coordinate system code value as far as CS-MAP is concerned.

This function is very similar to *CSepsg2msiCS*, but is much easier for the Visual Basic programmer to use.

CS_esriName2mai ESRI Name TO CS-MAP name

```
Const char* CS_esriName2Msi (Const char* esriName,unsigned short* flags);
```

Returns a pointer to the CS-MAP coordinate system key name which corresponds to the ESRI name provided by the **esriName** argument. Returns null pointer if the ESRI definition name is unknown to CS-MAP or an equivalent CS-MAP name does not exist. The **flags** argument is required, set to zero, and is otherwise unused at the current time.

CS_msi2epsg MSI key name TO EPSG code value

```
long CS_msi2epsg (Const char *msiKeyName);
```

CS_msi2epsg return the EPSG code value associated with a CS_MAP coordinate system key name. The ***msiKeyName*** argument specifies the CS-MAP key name for which the EPSG code value is to be returned.

This is function is very similar to *CSmsi2epsgCS*, but it is easier for the Visual Basic programmer to use.

CS_msiName2Esri CS-MAP NAME TO ESRI name

```
Const char* EXP_LVL1 CS_msiName2Esri (Const char* msiName)
```

Returns a pointer to the ESRI coordinate system key name which corresponds to the CS-MAP name provided by the **msiName** argument. Returns null pointer if the ESRI definition name is unknown to CS-MAP.

CSepsg2msiCS EPSG code to MSI key name, Coordinate Systems

```
Const char* CSepsg2msiCS (long epsgNbr,short* flags);
```

CSepsg2msiCS returns a pointer to a constant string which is the CS-MAP key name which corresponds to the EPSG code number provided by the **epsgNbr** argument. A null pointer is returned if a corresponding Mentor key name does not exists for the given EPSG code value.

In those cases where *CSepsg2msiCS* returns a non-null pointer, it will also return bit map in the variable pointed to by the **flags** argument. The least significant bits in this bit map value have the following significance:

Bit Number (0 = LSB)	Meaning (when set)
0	CS-MAP distributions prior to 11.10 included a definition of this coordinate system.
1	EPSG has deprecated this definition. It should no longer be used for output.

CSepsg2msiDT EPSG code to MSI key name, DaTums

```
Const char* CSepsg2msiDT (long epsgNbr,short* flags);
```

CSepsg2msiDT returns a pointer to a constant string which is the CS-MAP key name which corresponds to the EPSG code number provided by the **epsgNbr** argument. A null pointer is returned if a corresponding CS-MAP key name does not exists for the given EPSG code value.

In those cases where *CSepsg2msiDT* returns a non-null pointer, it will also return bit map in the variable pointed to by the **flags** argument. The least significant bits in this bit map value have the following significance:

Bit Number (0 = LSB)	Meaning (when set)
0	CS-MAP distributions prior to 11.10 included a definition of this coordinate system.
1	EPSG has deprecated this definition. It should not be used for output.

CSepsg2msiDT EPSG code to MSI key name, ELlipsoids

```
Const char* CSepsg2msiEL (long epsgNbr,short* flags);
```

CSepsg2msiEL returns a pointer to a constant string which is the CS-MAP key name which corresponds to the EPSG code number provided by the **epsgNbr** argument. A null pointer is returned if a corresponding Mentor key name does not exists for the given EPSG code value.

In those cases where *CSepsg2msiEL* returns a non-null pointer, it will also return bit map in the variable pointed to by the **flags** argument. The least significant bits in this bit map value have the following significance:

Bit Number (0 = LSB)	Meaning (when set)
0	CS-MAP distributions prior to 11.10 included a definition of this ellipsoid.
1	EPSG has deprecated this definition. It should not be used for output.

CSepsgByIdxCS EPSG codes BY InDeX, Coordinate Systems

```
long CSepsgByIdxCS (int index);
```

Use *CSepsgByIdxCS* to iterate through CS-MAP's knowledge of EPSG coordinate system code values. *CSepsgByIdxCS* returns the EPSG code in the ***index*** location (first = 0) of CS-MAP's internal EPSG code table. If the value of ***index*** is too large, a zero value is returned.

Thus, one may iterate through the entire EPSG code table by simply starting with an index value of zero, and incrementing it by one until *CSepsgByIdxCS* returns a zero value. The returned EPSG code values can be used with *CSepsg2msiCS* to obtain the corresponding coordinate system key name.

CSepsgByIdxDT EPSG codes BY InDeX, DaTums

```
long CSepsgByIdxDT (int index);
```

Use *CSepsgByIdxDT* to iterate through CS-MAP's knowledge of EPSG datum code values. *CSepsgByIdxDT* returns the EPSG code in the ***index*** location (first = 0) of CS-MAP's internal EPSG code table. If the value of ***index*** is too large, a zero value is returned.

Thus, one may iterate through the entire EPSG code table by simply starting with an index value of zero, and incrementing it by one until *CSepsgByIdxDT* returns a zero value. The returned EPSG code values can be used with *CSepsg2msiDT* to obtain the corresponding coordinate system key name.

CSepsgByIdxCS EPSG codes BY InDeX, ELlipsoids

```
long CSepsgByIdxEL (int index);
```

Use *CSepsgByIdxEL* to iterate through CS-MAP's knowledge of EPSG ellipsoid code values. *CSepsgByIdxEL* returns the EPSG code in the ***index*** location (first = 0) of CS-MAP's internal EPSG code table. If the value of ***index*** is too large, a zero value is returned.

Thus, one may iterate through the entire EPSG code table by simply starting with an index value of zero, and incrementing it by one until *CSepsgByIdxEL* returns a zero value. The returned EPSG code values can be used with *CSepsg2msiEL* to obtain the corresponding coordinate system key name.

CSmsi2epsgCS MSI to EPSG, Coordinate Systems

```
long CSmsi2epsgCS (Const char *msiKeyName,short* flags);
```

CSmsi2epsgCS returns the actual EPSG code value associated with the CS-MAP key name provided by the **msiKeyName** argument. A zero is returned if the provided name is invalid, or an EPSG equivalent does not exist. *CSmsi2epsgCS* returns, in the variable pointed to by the **flags** argument, to a bit map value which indicates the status of the definition referred to. The value has no meaning if the return value of the function is zero.

Bit Number (0 = LSB)	Meaning (when set)
0	CS-MAP distributions prior to 11.10 included a definition of this coordinate system.
1	EPSG has deprecated this definition. It should no longer be used for output.

CSmsi2epsgDT MSI to EPSG, DaTums

```
long CSmsi2epsgCS (Const char *msiKeyName,short* flags);
```

CSmsi2epsgDT returns the actual EPSG code value associated with the CS-MAP key name provided by the **msiKeyName** argument. A zero is returned if the provided name is invalid, or an EPSG equivalent does not exist. *CSmsi2epsgDT* returns, in the variable pointed to by the **flags** argument, to a bit map value which indicates the status of the definition referred to. The value has no meaning if the return value of the function is zero.

Bit Number (0 = LSB)	Meaning (when set)
0	CS-MAP distributions prior to 11.10 included a definition of this datum.
1	EPSG has deprecated this definition, it should not be used for output.

CSmsi2epsgEL MSI to EPSG, ELlipsoids

```
long CSmsi2epsgEL (Const char *msiKeyName,short* flags);
```

CSmsi2epsgEL returns the actual EPSG code value associated with the CS-MAP key name provided by the **msiKeyName** argument. A zero is returned if the provided name is invalid, or an EPSG equivalent does not exist. *CSmsi2epsgEL* returns, in the variable pointed to by the **flags** argument, to a bit map value which indicates the status of the definition referred to. The value has no meaning if the return value of the function is zero.

Bit Number (0 = LSB)	Meaning (when set)
0	CS-MAP distributions prior to 11.10 included a definition of this ellipsoid.
1	EPSG has deprecated this definition, it should not be used for output.

Legacy Functions

Functions which are obsolete and/or replaced by newer implementations are described in this section.

CS842grf wgs 84 TO local Geodetic ReFerence system

```
int CS842grf (Const double ll_84,double ll_lcl,struct cs_Grfprm_ *grf);
```

Given a properly initialized local geodetic reference system parameter block via the **grf** argument,

CS842grf will convert the geographic coordinate, i.e. latitude and longitude, provided in **ll_84** to the equivalent coordinates in the local geodetic reference system and return the results in **ll_lcl**. **ll_84** and **ll_lcl** may point to the same array. In both arrays, the first element is the longitude and second element is the latitude. Latitudes and longitudes must in degrees relative to Greenwich where negative values indicate south latitude and west longitude.

CS842grf returns **FALSE** if the conversion was correctly performed, else **TRUE** is returned.

ERRORS

CS842grf will return **TRUE** and set `cs_Error` appropriately if any of the following conditions are encountered during the conversion:

Cs_WGG_CNVRG	The iterative calculation of the local geodetic coordinate failed to converge after 6 iterations.

CS_bwcalc Bursa/Wolfe CALCulation

```
int CS_bwcalc (Const double lcl_ll [3],double ll_84 [3],
     Const struct cs_GrfBurs_ *bursa);
```

CS_bwcalc uses the information provided in the *cs_GrfBurs_* structure provided by the *bursa* argument to convert the latitude and longitude in the *lcl_ll* array to WGS84 based latitude and longitude. The results are returned in the array provided by the *ll_84* argument.

All elements of the *ll_lcl* and *ll_84* arrays must be in degrees. Longitude is carried in the first element of each array; lataitude in the second element; and the third element is currently unused. In all cases, negative values are used for west longitude and south latitude.

CS_bwcalc returns zero to indicate successful calculation, and non-zero otherwise. Currently, *CS_bwcalc* is always successful.

CS_getcs GET Coordinate System definition

```
int CS_getcs (Const char *key_nm,struct cs_Csdef_ *bufr);
```

CS_getcs will return in the memory location indicated by the **bufr** argument the definition of the coordinate system whose key name is given by the **key_nm** argument. *CS_getcs* normally returns zero.

ERRORS

CS_getcs will return a -1 and set `cs_Error` if any of the following conditions are detected:

cs_CSDICT	The Coordinate System Dictionary file could not be found or otherwise opened. (See *CS_altdr*)
cs_IOERR	A physical I/O error occurred during access to the Coordinate System Dictionary file.
cs_CS_BAD_MAGIC	The file accessed under the assumption that it was a Coordinate System Dictionary wasn't a Coordinate System Dictionary after all; it had an invalid magic number on the front end.
cs_CS_NOT_FND	A coordinate system definition with the name given by **key_nm** was not found in the Coordinate System Dictionary.
cs_NO_MEM	Insufficient dynamic memory was available to allocate space for a `cs_Csdef_` structure.

CS_getdt GET DaTum definition

```
int CS_getdt (Const char *key_nm,struct cs_Dtdef_ *bufr);
```

CS_getdt will return in the memory location indicated by the **bufr** argument the definition of the datum whose name is given by the **key_nm** argument. *CS_getdt* normally returns zero.

ERRORS

CS_getdt will return a -1 and set `cs_Error` if any of the following conditions are detected:

cs_DTDICT	The Datum Dictionary file could not be found or otherwise opened. (See *CS_altdr*)
cs_IOERR	A physical I/O error occurred during access to the Datum Dictionary file.
cs_DT_BAD_MAGIC	The file accessed under the assumption that it was a Datum Dictionary wasn't a Datum Dictionary after all; it had an invalid magic number on the front end.
cs_DT_NOT_FND	A datum definition with the name given by **key_nm** was not found in the Datum Dictionary.
cs_NO_MEM	Insufficient dynamic memory was available to allocate space for a `cs_Dtdef_` structure.

CS_getel GET ELlipsoid definition

```
int CS_getel (Const char *key_nm,struct cs_Eldef_ *bufr);
```

CS_getel will return in the memory location indicated by the **bufr** argument the definition of the ellipsoid whose name is given by the **key_nm** argument. *CS_getel* normally returns zero.

ERRORS

CS_getel will return a -1 and set `cs_Error` if any of the following conditions are detected:

cs_ELDICT	The Ellipsoid Dictionary file could not be found or otherwise opened. (See *CS_altdr*)
cs_IOERR	A physical I/O error occurred during access to the Ellipsoid Dictionary file.
cs_EL_BAD_MAGIC	The file accessed under the assumption that it was an Ellipsoid Dictionary wasn't a Ellipsoid Dictionary after all; it had an invalid magic number on the front end.
cs_EL_NOT_FND	An ellipsoid definition with the name given by **key_nm** was not found in the Ellipsoid Dictionary.
cs_NO_MEM	Insufficient dynamic memory was available to allocate space for a `cs_Eldef_` structure.

CSgrf284 local Geodetic ReFerence system TO wgs 84

```
int CSgrf284 (Const double ll_lcl [2],double ll_84 [2],struct cs_Grfprm_
*grf);
```

Given a properly initialized Geodetic Reference System parameter block via the **grf** argument, *CSgrf284* will convert the local geodetic reference system coordinates in the **lcl_ll** array to their equivalent WGS 84 values and return the result in the **ll_84** array. Both pointers may point to the same array. In both arrays, the first element is the longitude and the second element is the latitude. Latitudes and longitudes are given in degrees where negative values indicate south latitude and west longitude.

CSgrf284 will use *CS_mrcalc* to calculate the new values if the provided `cs_Grfprm_` structure has a valid multiple regression formula initialized within it. If a multiple regression formula is not available, or if the multiple regression calculation fails for any reason, *CSgrf284* uses *CS_mocalc* or *CS_bwcalc* as is appropriate to arrive at the desired result.

ERRORS

CSgrf284 will return zero to indicate that the conversion proceeded to a normal result; **TRUE** to indicate

that an error occurred. In the event of an error the global variable `cs_Error` will be set to **cs_MREG_RANGE** in any case where a valid multiple regression formula did indeed exist, but the location given by **ll_lcl** was outside of the domain of the multiple regression formula.

CSgrfinit local Geodetic ReFerence system INITialize

```
int CSgrfinit (Const struct cs_Csprm_ *csprm,grf,struct cs_Grfprm_ *grf);
```

Using the datum information contained in the `datum` element of the `cs_Csprm_` structure provided by the **csprm** argument, *CSgrfinit* will initialize the `cs_Grfprm_` structure provided by the **grf** argument for datum conversions. The initialized `cs_Grfprm_` structure contains all the information necessary to convert geographic coordinates (i.e. latitude/longitude) between the datum described (i.e. the local geodetic reference system) and WGS 84.

Depending on the information contained in the `datum` element, either the Molodensky form or the Bursa/Wolfe form of the conversion is always initialized as a backup to the possible initialization of a multiple regression formula. If a valid and appropriately named multiple regression formula definition file is located in the directory indicated by the `cs_Dir` global variable (see CSdata.c), the multiple regression conversion is also initialized.

ERRORS

CSgrfinit always returns **FALSE** to indicate that no error condition has been detected. It is possible that future versions of *CSgrfinit* will return **TRUE** to indicate the existence of an error condition.

CS_mocalc MOlodensky CALCulator

```
void CS_mocalc (Const double ll_lcl [2],double ll_84 [2],
      Const struct cs_GrfMolo_ *molo);
```

Given a properly initialized `cs_GrfMolo_` structure via the **molo** argument, *CS_mocalc* converts the local geodetic reference system latitude and longitude pair provided by **ll_lcl** to WGS84 base latitudes and longitudes and returns the result in the array pointed to by the **ll_84** argument. *CSgrfinit* is usually used to obtain an initialized `cs_GrfMolo_` structure.

The formulas used are the unabridged Molodensky formulas where the elevation is assumed to be zero.

ERRORS

At the current time, *CS_mocalc* always returns 0 to indicate that the calculation was performed successfully.

CS_mrcalc Multiple Regression CALCulator

```
int CS_mrcalc (Const double ll_lcl [2],double ll_84 [2],
      Const struct cs_GrfMreg_ *mr_ptr);
```

Given a properly initialized `cs_GrfMreg_` structure via the **mr_ptr** argument, *CS_mrcalc* converts the local geodetic reference system latitude and longitude pair provided by **ll_lcl** to WGS84 base latitudes and longitudes and returns the result in the array pointed to by the **ll_84** argument. *CSgrfinit* is usually used to obtain an initialized `cs_GrfMreg_` structure.

ERRORS

CS_mrcalc will copy the contents of **ll_lcl** to **ll_84** and return 1 if the location provided by **ll_lcl** is outside of the domain of the Multiple Regression formula provided. Otherwise, a zero is returned.

CS_p7calc 7 Parameter CALCulation

```
int CS_bwcalc (Const double lcl_ll [3],double ll_84 [3],
      Const struct cs_Grf7Prm_ *parm7);
```

CS_p7calc uses the information provided in the `cs_GrfBurs_` structure provided by the **parm7** argument to convert the latitude and longitude in the **lcl_ll** array to WGS84 based latitude and longitude. The results are returned in the array provided by the **ll_84** argument.

All elements of the **ll_lcl** and **ll_84** arrays must be in degrees. Longitude is carried in the first element of each array; latitude in the second element; and the third element is currently unused. In all cases, negative values are used for west longitude and south latitude.

CS_p7calc returns zero to indicate successful calculation, and non-zero otherwise. Currently, *CS_p7calc* is always successful.

CS_putcs PUT Coordinate System to dictionary

```
int CS_putcs (Const struct cs_Csdef_ *csDef,int crypt);
```

CS_putcs writes the coordinate system definition pointed to by the **csDef** argument to the coordinate system dictionary. If the **crypt** argument is non-zero, the definition is encrypted before writing. *CS_putcs* returns a zero on success, a negative value in the event of an error.

ERRORS

CS_putcs uses *CS_csupd* for a majority of its functionality; the return value is primarily what *CS_csupd* returns.

CS_putdt PUT DaTum to dictionary

```
int CS_putdt (Const struct cs_Dtdef_ *dtDef,int crypt);
```

CS_putdt writes the datum definition pointed to by the **dtDef** argument to the datum dictionary. If the **crypt** argument is non-zero, the definition is encrypted before writing. *CS_putdt* returns a zero on success, a negative value in the event of an error.

ERRORS

CS_putdt uses *CS_dtupd* for a majority of its functionality; the return value is primarily what *CS_dtupd* returns.

CS_putel PUT ELlipsoid to dictionary

```
int CS_putel (Const struct cs_Eldef_ *elDef,int crypt);
```

CS_putel writes the ellipsoid definition pointed to by the **elDef** argument to the ellipsoid dictionary. If the **crypt** argument is non-zero, the definition is encrypted before writing. *CS_putel* returns a zero on success, a negative value in the event of an error.

ERRORS

CS_putel uses *CS_elupd* for a majority of its functionality; the return value is primarily what *CS_elupd* returns.

CS_un2d Units, Name TO Double

```
double CS_un2d (Const char *uname);
```

This function is now obsolete, being replaced by *CS_unitlu*. It is being maintained to provide compatibility with previous releases only.

CS_un2d will return a double that represents the multiplier required to convert a value in the unit system indicated by **uname** to units of meters. **Uname** must be a null terminated string defining one of the supported units as defined in CSdataU.c. *CS_un2d* returns zero in the event the provided unit name is not known.

For example, to convert a value in feet to meters, one could code:

```
double CS_un2d ();
{
     meters = feet * CS_un2d ("FOOT");
}
```

Or to convert meters to feet:

```
double CS_un2d ();
{
     feet = meters / CS_un2d ("FOOT");
}
```

CS_un2d knows about the first and second abbreviations provided for in the `cs_Unittab_` structure. Therefore, the following are equivalent to the above:

```
double CS_un2d ();
{
     meters = feet * CS_un2d ("FT");
}
double CS_un2d ();
{
     feet = meters / CS_un2d ("FT");
}
```

ERRORS

CS_un2d will return zero and set `cs_Error` to **cs_INV_UNIT** if the unit name pointed to by **uname** is not defined in `cs_Unittab`.

CS842grf wgs 84 TO local Geodetic ReFerence system

```
int CS842grf (Const double ll_84,double ll_lcl,struct cs_Grfprm_ *grf);
```

Given a properly initialized local geodetic reference system parameter block via the **grf** argument, *CS842grf* will convert the geographic coordinate, i.e. latitude and longitude, provided in **ll_84** to the equivalent coordinates in the local geodetic reference system and return the results in **ll_lcl**. **ll_84** and **ll_lcl** may point to the same array. In both arrays, the first element is the longitude and second element is the latitude. Latitudes and longitudes must in degrees relative to Greenwich where negative values indicate south latitude and west longitude.

CS842grf returns **FALSE** if the conversion was correctly performed, else **TRUE** is returned.

ERRORS

CS842grf will return **TRUE** and set `cs_Error` appropriately if any of the following conditions are encountered during the conversion:

cs_WGG_CNVRG	The iterative calculation of the local geodetic coordinate failed to converge after 6 iterations.

CSgrf284 local Geodetic ReFerence system TO wgs 84

```
int CSgrf284 (Const double ll_lcl [2],double ll_84 [2],struct cs_Grfprm_
*grf);
```

Given a properly initialized Geodetic Reference System parameter block via the **grf** argument, *CSgrf284* will convert the local geodetic reference system coordinates in the **lcl_ll** array to their equivalent WGS 84 values and return the result in the **ll_84** array. Both pointers may point to the same array. In both arrays, the first element is the longitude and the second element is the latitude. Latitudes and longitudes are given in degrees where negative values indicate south latitude and west longitude.

CSgrf284 will use *CS_mrcalc* to calculate the new values if the provided `cs_Grfprm_` structure has a valid multiple regression formula initialized within it. If a multiple regression formula is not available, or if the multiple regression calculation fails for any reason, *CSgrf284* uses *CS_mocalc* or *CS_bwcalc* as is appropriate to arrive at the desired result.

ERRORS

CSgrf284 will return zero to indicate that the conversion proceeded to a normal result; **TRUE** to indicate that an error occurred. In the event of an error the global variable `cs_Error` will be set to **cs_MREG_RANGE** in any case where a valid multiple regression formula did indeed exist, but the location given by **ll_lcl** was outside of the domain of the multiple regression formula.

CSgrfinit local Geodetic ReFerence system INITialize

```
int CSgrfinit (Const struct cs_Csprm_ *csprm,grf,struct cs_Grfprm_ *grf);
```

Using the datum information contained in the `datum` element of the `cs_Csprm_` structure provided by the **csprm** argument, *CSgrfinit* will initialize the `cs_Grfprm_` structure provided by the **grf** argument for datum conversions. The initialized `cs_Grfprm_` structure contains all the information necessary to

convert geographic coordinates (i.e. latitude/longitude) between the datum described (i.e. the local geodetic reference system) and WGS 84.

Depending on the information contained in the `datum` element, either the Molodensky form or the Bursa/Wolfe form of the conversion is always initialized as a backup to the possible initialization of a multiple regression formula. If a valid and appropriately named multiple regression formula definition file is located in the directory indicated by the `cs_Dir` global variable (see CSdata.c), the multiple regression conversion is also initialized.

ERRORS

CSgrfinit always returns **FALSE** to indicate that no error condition has been detected. It is possible that future versions of *CSgrfinit* will return **TRUE** to indicate the existence of an error condition.

CSgeoidCls GEOID, CLoSe

```
void CSgeoidCls (void);
```

CSgeoidCls decrements the global variable `csGeoidOpenCnt` and, if the result is zero or less, closes all open GEOID database files and *free*'s the memory allocated to the GEOID file control block cache and the GEOID grid cell cache.

CSgeoidInit increments the global variable `csGeoidOpenCnt` each time it is called. *CSgeoidCls* decrements this count each time it is called. Resources are released only when `csGeoidOpenCnt` reaches zero. Thus, resources are not released prematurely in processes where more than one geoid height process is active at any given time.

Currently, *CSgeoidCls* does not *free* the memory allocated to the GEOID database directory. The GEOID database directory does not require much memory (32 bytes per database) and is rather expensive, computationally, to initialize. Therefore, this system resource is left alone by this function. *CSgeoidInit* will not attempt to reinitialize the directory if the value of `csGeoidDirP` is not **NULL**. Non-source licensees who find this objectionable can free the GEOID database directory directly (i.e. *CS_free (*`csGeoidDirP);`). Be sure to set the value of `csGeoidDirP` to **NULL** and the value of `csGeoidDirM` and `csGeoidDirU` to zero.

CSgeoiddbo GEOID, DataBase Open

```
extern int cs_Error;
Const struct csGeoidFcb_ *CSgeoiddbo (Const double ll_84 [2]);
```

The database for GEOID height calculations consists of several different files, each covering a specific geographic area. *CSgeoiddbo* returns a pointer to a `csGeoidFcb_` structure that provides access to the appropriate GEIOD database file for the geographic region containing the coordinate indicated by the **ll_84** argument. **LI_84** must contain the longitude in the first element, and the latitude in the second element. Both must be in degrees where negative values are used to indicate west and south. This geographic coordinate is expected to be a WGS 84 geographic coordinate.

CSgeoiddbo searches the GEOID file control block cache to see if the required database file is already open. If so, a pointer to it is returned after this block is made the most recently accessed (i.e. moved to the top of the linked list). Otherwise, the appropriate GEOID file is opened and a pointer to the resulting file control block is returned.

If a GEOID database file that covers the geographic region containing **ll_84** cannot be found on the

system, `cs_Error` is set to zero and the **NULL** pointer is returned. If the database open failed for another reason, the **NULL** pointer will be returned but `cs_Error` will be set to indicate the nature of the failure.

CSgeoiddbo is called by *CSgeoidptr* when *CSgeoidptr* needs to access a database to fetch a new grid cell. It is not typically called by an application program directly. Should an application program need to access this function directly, *CSgeoidInit* must be called prior to the first call to this function.

ERRORS

For failures due to causes other than the availability of appropriate data, *CSgeoiddbo* will return the **NULL** pointer and set `cs_Error` as follows:

Cs_GEOID_NO_SETUP	*CSgeoidInit* was not called prior to calling this function, or the affect of calling *CSgeoidInit* was canceled by a call to *CSgeoidCls*.
Cs_INV_FILE	A required GEOID database file is corrupted beyond use.
Cs_GEOID_FILE	A required GEOID database file that does exist could not be opened.

CSgeoiddir GEOID, database DIRectory

```
Const struct csGeoidDir_ *CSgeoiddir (Const double ll_84 [2]);
```

CSgeoiddir returns a pointer to the `csGeoidDir_` structure in the GEOID database file directory that contains the grid cell required to convert the coordinate given by the **ll_84** argument. The first element of **ll_84** must be the longitude of the coordinate to be converted; the second element must contain the latitude. Both elements must be in degrees. Negative values are used to indicate west longitude and south latitude. The geographic coordinate supplied by the **ll_84** argument is expected to be a WGS 84 geographic coordinate.

In the event a database covering the specific location provided by the **ll_84** argument cannot be located, *CSgeoiddir* will set `cs_Error` to zero and return the **NULL** pointer.

CSgeoiddbo, upon determining that a GEOID database file system needs to be opened, calls *CSgeoiddir* to determine the base name of the database required to convert a specific coordinate. The GEOID database directory pointed to by `csGeoidDirP` contains a list of the base names of all GEOID database files present on the system, and the geographic region covered by each.

Application programs do not normally call this function directly. Should an application need to do so, *CSgeoidInit* must be called prior to the first call to this function.

ERRORS

CSgeoiddir will return the **NULL** pointer, and set `cs_Error` to the indicated value if any of the following conditions are encountered:

Cs_GEOID_NO_SETUP	*CSgeoidInit* was not called prior to calling this function, or the affect of calling *CSgeoidInit* was canceled by a call to *CSgeoidCls*.

CSgeoidHgt GEOID HeiGhT

```
int CSgeoidHgt (Const double ll_84 [2],double *height);
```

Given a WGS 84 based geographic coordinate via the **ll_84** argument, *CSgeoidHgt* will return in the double pointed to by the **height** argument the geoid height, in meters, at the indicated point. Some prefer the term geoid separation to geoid height. Your application needs to call the *CSgeoidInit* function once before calling *CSgeoidHgt*.

CSgeoidHgt will return a zero if it was successful, a -1 if a system error of some sort occurred, or a +1 to indicate that data for the specific location requested was not available. In any case, when *CSgeoidHgt* returns a non-zero value, the **height** variable will be set to zero.

CSgeoidHgt relies on data files being present in the data directory. Currently, these data files must be in the format used by the GEOID96 program published by the National Geodetic Service. In order to extend the range of this function beyond that of US geography, additional file formats will be supported in the future.

ERRORS

CSgeoidHgt will return a -1 and set the value of global variable `cs_Error` appropriately if any of the following conditions are encountered during the conversion:

Cs_GEOID_NO_SETUP	*CSgeoidInit* was not called prior to calling this function, or the affect of calling *CSgeoidInit* was canceled by a call to *CSgeoidCls*.
Cs_INV_FILE	A required GEOID96 database file is corrupted beyond use.
Cs_GEOID_FILE	A required GEOID96 database file that does exist could not be opened.
Cs_NO_MEM	An operating system request for additional memory failed.

CSgeoidinit GEOID, INITialize

```
int CSgeoidInit (void);
```

CSgeoidInit causes the application program to be initialized for the computation of geoid heights.

CSgeoidInit, must be called prior to calling any other function in the group of functions related to geoid height computation.

CSgeoidinit will return a zero value to indicate that the geoid height computation system has been properly initialized. A positive non-zero status is returned if the initialization failed due to a lack of data files. A negative non-zero status is returned if the initialization failed due to a system error such as a physical I/O error or insufficient memory.

CSgeoidInit allocates and initializes the GEOID database directory (`csGeoidDirP`), the GEOID file control block cache (`csGeoidFcb`) and the GEOID grid cell cache (`csGeoidGrdP`). The effects of *CSgeoidInit* upon system resources can be undone by calling *CSgeoidCls*.

CSgeoidInit may be called repeatedly without adverse affects. However, be aware that *CSgeoidInit* increments the global variable `csGeoidOpenCnt` once each time it is called. *CSgeoidCls* decrements this count and releases system resource only when the count reaches zero.

ERRORS

CSgeoidInit will return a negative non-zero value and set `cs_Error` to the indicated value should any of the following conditions be encountered:

Cs_NO_MEM	Insufficient memory is available to allocate the GEOID database directory, the GEOID file control block cache, or the GEOID grid cell cache.

CSgeoidptr GEOID, return grid cell PoinTeR

```
extern int cs_Error
Const struct csGeoidGrd_ *CSgeoidptr (Const double ll_84 [2]);
```

CSgeoidptr will return a pointer to a `csGeoidGrd_` grid cell structure appropriate for use in computing the geoid height at the location specified by the **ll_27** argument. The **NULL** pointer is returned if a grid cell for the coordinate could not be returned. If the cause of the failure was simple non-availability of the data, `cs_Error` is set to zero. Otherwise, `cs_Error` will contain the appropriate error code indicating the nature of the failure.

CSgeoidptr returns a pointer to an entry in the grid cell cache pointed to by `csGeoidGrd` after recording the returned entry the most recently accessed by making it the first in the linked list. Obviously, if the required grid cell does not already exist in the cache, it must be fetched from disk. In so doing, *CSgeoidptr* always uses the last entry in the linked list that will always be the least recently accessed grid cell. Therefore, repeated requests for the grid cell covering coordinates in the same local geographic region will be satisfied with a minimum of disk I/O.

To a certain extent, increasing the number of entries in the grid cell cache will improve performance on conversion projects that are large both geographically and in the number of points to be converted. However, since the cache is searched linearly, a point of diminishing returns can be reached. The number of grid cell cache entries is specified by the value contained in the `csGeoidGrdCnt` global variable. This variable must be set to the desired value prior to the first call to *CSgeoidInit*.

CSgeoidptr is normally called by *CSgeoidHgt* to obtain the appropriate grid cell for each geoid height computation. Applications do not normally call this function directly. Should an application need to

access this function directly, *CSgeoidInit* must be called prior to the first call to this function.

ERRORS

CSgeoidptr will return the **NULL** pointer, and set `cs_Error` to the indicated value if any of the following conditions are encountered:

Cs_GEOID_NO_SETUP	*CSgeoidInit* was not called prior to calling this function, or the affect of calling *CSgeoidInit* was canceled by a call to *CSgeoidCls*.
Cs_INV_FILE	A required GEOID database file is corrupted beyond use.
Cs_GEOID_FILE	A required GEOID database file that does exist could not be opened.

CShpg283 High Precision Gps network, 91 TO 83 conversion

```
int CShpg283 (Const double ll_91[2],double ll_83 [2],int blk_err);
```

Given latitude and longitude values (based on the High Precision GPS network) in the **ll_91** array, *CShpg283* will return in the **ll_83** array the North American Datum of 1983 equivalent values. **Ll_91** and **ll_83** may point to the same array. Latitude and longitude values must be given in degrees where negative values are used to indicate south and west respectively. Longitude values are carried in the first element in each array; latitude is carried in the second.

CShpg283 will return a zero if the requested conversion was successfully performed. Otherwise, the **ll_91** array is copied to the **ll_83** array unaltered and a non-zero value returned.

The algorithm used and the data accessed are identical to that used by the National Geodetic Survey's NADCON program. Therefore, the numerical results are identical to those of NADCON. NADCON is the only known method of converting to/from High Precision GPS Network coordinates.

Typically, this function is not called directly by an application. This function is typically accessed via a call to *CS_dtcvt* which in turns calls this function due to the presence of a reference to this conversion in the `cs_Dtcprm_` structure which is provided to *CS_dtcvt*. In this manner, application code does not directly reference any datum conversion software. New datum conversions can be added without modification of application code.

If, for some reason, an application needs to access this function directly, the application must invoke *CShpginit* prior to the first call to this function.

The **blk_err** argument indicates the disposition of datum conversion errors caused by a lack of data covering the geographic region containing the coordinate to be converted. *CS_dtcvt* causes the value specified at the time of datum conversion set up to be passed to *CShpg283*. The valid values for this argument are:

cs_DTCFLG_BLK_I	Errors caused by data availability are silently ignored and a zero status value is returned.
cs_DTCFLG_BLK_W	Errors caused by data availability are reported directly to *CS_erpt* as a warning and a positive non-zero status value returned.
cs_DTCFLG_BLK_1	Errors caused by data availability are reported to *CShpgbnf* and a positive non-zero status value returned. *CShpgbnf* will suppress repeated error messages concerning the same one degree by one degree block and suppress all such error messages after 10 have been reported to *CS_erpt*. (Note, *CShpgbnf* is also obsolete.)
cs_DTCFLG_BLK_F	Errors caused by data availability are reported directly to *CS_erpt* as a fatal error condition and a negative non-zero status value returned.

The null conversion is always performed before any other processing is attempted. The null conversion consists of simply copying **ll_91** to **ll_83**.

ERRORS

CShpg283 will return a -1 and set the value of global variable `cs_Error` appropriately if any of the following conditions are encountered during the conversion:

cs_DTC_NO_SETUP	*CShpginit* was not called prior to calling this function, or the affect of calling *CShpginit* was canceled by a call to *CShpgcls*.
cs_HPGN_ICNT	Nine iterations of the algorithm (ala NADCON) failed to produce a result within acceptable tolerance.
cs_INV_FILE	A required NADCON database file is corrupted beyond use.
cs_DTC_FILE	A required NADCON database file that does exist could not be opened.
cs_HPGN_CONS	Two properly named files that are supposed to represent a single HPGN database contain inconsistent information.

BUGS

The specifications for this function required a duplication of NADCON's functionality. Therefore, the precision produced by this function is purposely limited to that provided by NADCON; i.e. precision is limited to 9 decimal places.

CShpg291 High Precision Gps network, (from 83) TO 91 conversion

```
int CShpg291 (Const double ll_83 [2],double ll_91 [2],intblk_err);
```

Given latitude and longitude values (based on the North American Datum of 1983) in the **ll_83** array, *CShpg291* will return in the **ll_91** array the High Precision GPS Network equivalent values. **Ll_83** and **ll_91** may point to the same array. Latitude and longitude values must be given in degrees where negative values are used to indicate south and west respectively. Longitude values are carried in the first element in each array; latitude is carried in the second.

CShpg291 will return a zero if the requested conversion was successfully performed. Otherwise, the **ll_83** array is copied to the **ll_91** array unaltered and a non-zero value returned.

The algorithm used and the data accessed are identical to the National Geodetic Survey's NADCON program. Therefore, the numerical results are identical to NADCON (round off errors excepted).

Typically, this function is not called directly by an application. This function is typically accessed via a call to *CS_dtcvt* which in turns calls this function due to the presence of a reference to this conversion in the `cs_Dtcprm_` structure which is provided to *CS_dtcvt*. In this manner, application code does not directly reference any datum conversion software. New datum conversions can be added without modification of application code.

If, for some reason, an application needs to access this function directly, the application must invoke *CShpginit* prior to the first call to this function.

The **blk_err** argument indicates the disposition of datum conversion errors caused by a lack of data covering the geographic region containing the coordinate to be converted. *CS_dtcvt* causes the value specified at the time of datum conversion set up to be passed to *CShpg291*. The valid values for this argument are:

cs_DTCFLG_BLK_I	Errors caused by data availability are silently ignored and a zero status value is to be returned.
cs_DTCFLG_BLK_W	Errors caused by data availability are reported directly to *CS_erpt* as a warning and a positive non-zero status value returned.
cs_DTCFLG_BLK_1	Errors caused by data availability are reported to *CShpgbnf* and a positive non-zero status value returned. *CShpgbnf* will suppress repeated error messages concerning the same one degree by one degree block and suppress all such error messages after 10 have been reported to *CS_erpt*.
cs_DTCFLG_BLK_F	Errors caused by data availability are reported directly to *CS_erpt* as a fatal error condition and a negative non-zero status value returned.

ERRORS

CShpg291 will return a -1 and set the value of global variable `cs_Error` appropriately if any of the following conditions are encountered:

cs_DTC_NO_SETUP	*CShpginit* was not called prior to calling this function, or the affect of calling *CShpginit* was canceled by a call to *CShpgcls*.
cs_INV_FILE	A required NADCON database file is corrupted beyond use.
cs_DTC_FILE	A required NADCON database file that does exist could not be opened.
cs_HPGN_CONS	Two properly named files that are supposed to represent a single NADCON database contain inconsistent information.

CShgndbo High Precision Gps network, DataBase Open

```
extern int cs_Error;
Const struct csNadfcb_ *CShgndbo (Const double ll_83 [2]);
```

The database for NADCON datum conversions consists of several different file systems, each covering a specific geographic area and consisting of two separate files. *CShgndbo* returns a pointer to a `csNadfcb_` structure which provides access to the appropriate HPGN database file system for the geographic region containing the coordinate indicated by the **ll_83** argument. Note that this location is expected to be based on the North American Datum of 1983. **Ll_83** must contain longitude in the first element, and latitude in the second element. Both must be in degrees where negative values are used to indicate west and south.

CShgndbo searches the HPGN file control block cache to see if the required database file system is already open. If so, a pointer to it is returned after this block is made the most recently accessed (i.e. moved to the top of the linked list). Otherwise, the appropriate HPGN file system is opened and a pointer to the resulting file control block is returned.

If a HPGN database file system that covers the geographic region containing **ll_83** cannot be found on the system, `cs_Error` is set to zero and the **NULL** pointer is returned. If the database open failed for another reason, the **NULL** pointer will be returned but `cs_Error` will be set to indicate the nature of the failure.

CShgndbo is called by *CShgnptr* when *CShgnptr* needs to access a database to fetch a new grid cell. It is not typically called by an application program directly. Should an application program need to access this function directly, *CShgninit* must be called prior to the first call to this function.

ERRORS

For failures due to causes other than the availability of appropriate data, *CShgndbo* will return the **NULL** pointer and set `cs_Error` as follows:

cs_DTC_NO_SETUP	*CShgninit* was not called prior to calling this function, or the affect of calling *CShgninit* was canceled by a call to *CShgncls*.
cs_INV_FILE	A required NADCON database file is corrupted beyond use.
cs_DTC_FILE	A required NADCON database file that does exist could not be opened.
cs_NADCON_CONS	Two properly named files that are supposed to represent a single NADCON database contain inconsistent information.

CShpgdir High Precision Gps network database DlRectory

```
Const struct csNaddir_ *CShpgdir (Const double ll_83 [2]);
```

CShpgdir returns a pointer to the `csNaddir_` structure in the HPGN database file directory that contains the grid cell required to convert the coordinate given by the **ll_83** argument. The content of the **ll_83** array is expected to be a longitude latitude pair based on the North American Datum of 1983. The first element of **ll_83** must be the longitude of the coordinate to be converted; the second element must contain the latitude. Both elements must be in degrees. Negative values are used to indicate west longitude and south latitude.

In the event a database covering the specific location provided by the **ll_83** argument cannot be located, *CShpgdir* will set `cs_Error` to zero and return the **NULL** pointer.

CShpgdbo, upon determining that a HPGN database file system needs to be opened, calls *CShpgdir* to determine the base name of the database required to convert a specific coordinate. The HPGN database directory pointed to by `csHpgdirP` contains a list of the base names of all HPGN database file systems present on the system, and the geographic region covered by each.

Application programs do not normally call this function directly. Should an application need to do so, *CShpginit* must be called prior to the first call to this function.

ERRORS

CShpgdir will return the **NULL** pointer, and set `cs_Error` to the indicated value if any of the following conditions are encountered:

cs_DTC_NO_SETUP	*CShpginit* was not called prior to calling this function, or the affect of calling *CShpginit* was canceled by a call to *CShpgcls*.

CShpginit High Precision Gps network, INITialize

```
int CShpginit (void);
```

CShpginit causes the application program to be initialized for the conversion of coordinates based on the North American Datum of 1983 (NAD83) to coordinates based on the High Precision GPS Network (HPGN), also known as High Accuracy Reference Network (HARN), NAD83/91, and NAD83/92.

CShpginit is not usually called directly by application code. *CShpginit* is usually called by *CS_dtcsu* when a datum conversion involving HPGN to NAD83 or NAD83 to HPGN is requested. Applications which access *CShpg283*, *CShpg291*, *CShpgptr*, *CShpgdbo*, or *CShpgdir* directly will need to call this function prior to the first call to any of these functions.

CShpginit will return a zero value to indicate that the NAD83 to HPGN (or vice versa) conversion has been properly initialized. A positive non-zero status is returned if the initialization failed due to a lack of data files. A negative non-zero status is returned if the initialization failed due to a system error such as a physical I/O error or insufficient memory.

CShpginit allocates and initializes the NADCON database directory (`csHpgdirP`), the HPGN file control block cache (`csHpgfcbP`), and the HPGN grid cell cache (`csHpggrdP`). The effects of *CShpginit* upon system resources can be undone by calling *CShpgcls*.

CShpginit may be called repeatedly without adverse affects. However, be aware that *CShpginit* increments the global variable `csHpgcnt` once each time it is called. *CShpgcls* decrements this count and releases system resource only when the count reaches zero.

ERRORS

CShpginit will return a negative non-zero value and set `cs_Error` to the indicated value should any of the following conditions be encountered:

cs_NO_MEM	Insufficient memory is available to allocate the HPGN database directory, the HPGN file control block cache, or the HPGN grid cell cache.

CShpgptr High Precision Gps network, return grid cell PoinTeR

```
extern int cs_Error
Const struct csNadgrd_ *CShpgptr (Const double ll_83 [2]);
```

CShpgptr will return a pointer to a `csNadgrd_` grid cell structure appropriate for use in converting the coordinate given by the **ll_83** argument. The **NULL** pointer is returned if a grid cell for the coordinate could not be returned. If the cause of the failure was simple non-availability of the data, `cs_Error` is set to zero. Otherwise, `cs_Error` will contain the appropriate error code indicating the nature of the failure.

CShpgptr returns a pointer to an entry in the grid cell cache pointed to by `csHpggrdP` after recording the returned entry as the most recently accessed by making it the first in the linked list. Obviously, if

the required grid cell does not already exist in the cache, it must be fetched from disk. In so doing, *CShpgptr* always uses the last entry in the linked list that will always be the least recently accessed grid cell. Therefore, repeated requests for the grid cell covering coordinates in the same local geographic region will be satisfied with a minimum of disk I/O.

To a certain extent, increasing the number of entries in the grid cell cache will improve performance on conversion projects that are large both geographically and in the number of points to be converted. However, since the cache is searched linearly, a point of diminishing returns can be reached. The number of grid cell cache entries is specified by the value contained in the `csHpgccnt` (see Csdcdata.c) global variable. This variable must be set to the desired value prior to the first call to *CS_dtcsu*.

CShpgptr is normally called by *CShpg283* to obtain the appropriate grid cell for each coordinate to be converted. Applications do not normally call this function directly. Should an application need to access this function directly, *CShpginit* must be called prior to the first call to this function.

ERRORS

CShpgptr will return the NULL pointer and set `cs_Error` to the indicated value should any of the following conditions be encountered:

cs_DTC_NO_SETUP	*CShpginit* was not called prior to calling this function, or the affect of calling *CShpginit* was canceled by a call to *CShpgcls*.
cs_INV_FILE	A required HPGN database file is corrupted beyond use.
cs_DTC_FILE	A required HPGN database file that does exist could not be opened.
cs_HPGN_CONS	Two properly named files that are supposed to represent a single HPGN database contain inconsistent information.

CSnad227 North American Datum, 83 TO 27 conversion

```
int CSnad227 (Const double ll_83 [2],double ll_27 [2],int blk_err);
```

Given latitude and longitude values (based on the North American Datum of 1983) in the **ll_83** array, *CSnad227* will return in the **ll_27** array the 1927 datum equivalent values. **Ll_83** and **ll_27** may point to the same array. Latitude and longitude values must be given in degrees where negative values are used to indicate south and west respectively. Longitude values are carried in the first element in each array; latitude is carried in the second.

CSnad227 will return a zero if the requested conversion was successfully performed. Otherwise, the **ll_83** array is copied to the **ll_27** array unaltered and a non-zero value returned.

The algorithm used and the data accessed are identical to that used by the National Geodetic Survey's NADCON program. Therefore, the numerical results are identical to those of NADCON. NADCON is rapidly being accepted as the standard for NAD27 to NAD83 conversion (and vice versa).

Typically, this function is not called directly by an application. This function is typically accessed via a call to *CS_dtcvt* which in turns calls this function due to the presence of a reference to this conversion in the `cs_Dtcprm_` structure which is provided to *CS_dtcvt*. In this manner, application code does not directly reference any datum conversion software. New datum conversions can be added without modification of application code.

If, for some reason, an application needs to access this function directly, the application must invoke *CSnadinit* prior to the first call to this function.

The **blk_err** argument indicates the disposition of datum conversion errors caused by a lack of data covering the geographic region containing the coordinate to be converted. *CS_dtcvt* causes the value specified at the time of datum conversion set up to be passed to *CSnad227*. The valid values for this argument are:

cs_DTCFLG_BLK_I	Errors caused by data availability are silently ignored and a zero status value is returned.
cs_DTCFLG_BLK_W	Errors caused by data availability are reported directly to *CS_erpt* as a warning and a positive non-zero status value returned.
cs_DTCFLG_BLK_1	Errors caused by data availability are reported to *CSdtcbnf* and a positive non-zero status value returned. *CSdtcbnf* will suppress repeated error messages concerning the same one degree by one degree block and suppress all such error messages after 10 have been reported to *CS_erpt*.
cs_DTCFLG_BLK_F	Errors caused by data availability are reported directly to *CS_erpt* as a fatal error condition and a negative non-zero status value returned.

The null conversion is always performed before any other processing is attempted. The null conversion consists of simply copying **ll_83** to **ll_27**.

ERRORS

CSnad227 will return a -1 and set the value of global variable `cs_Error` appropriately if any of the following conditions are encountered during the conversion:

cs_DTC_NO_SETUP	*CSnadinit* was not called prior to calling this function, or the affect of calling *CSnadinit* was canceled by a call to *CSnadcls*.
cs_NADCON_ICNT	Nine iterations of the algorithm (ala NADCON) failed to produce a result within acceptable tolerance.
cs_INV_FILE	A required NADCON database file is corrupted beyond use.
cs_DTC_FILE	A required NADCON database file that does exist could not be opened.
cs_NADCON_CONS	Two properly named files that are supposed to represent a single NADCON database contain inconsistent information.

BUGS

The specifications for this function required a duplication of NADCON's functionality. Therefore, the precision produced by this function is purposely limited to that provided by NADCON; i.e. precision is limited to 9 decimal places.

CSnad283 North American Datum, (from 27) TO 83 conversion

```
int CSnad283 (Const double ll_27 [2],double ll_83 [2],int blk_err);
```

Given latitude and longitude values (based on the North American Datum of 1927) in the **ll_27** array, *CSnad283* will return in the **ll_83** array the 1983 datum equivalent values. **Ll_27** and **ll_83** may point to the same array. Latitude and longitude values must be given in degrees where negative values are used to indicate south and west respectively. Longitude values are carried in the first element in each array; latitudes are carried in the second.

CSnad283 will return a zero if the requested conversion was successfully performed. Otherwise, the **ll_27** array is copied to the **ll_83** array unaltered and a non-zero value returned.

The algorithm used and the data accessed are identical to the National Geodetic Survey's NADCON program. Therefore, the numerical results are identical to NADCON (round off errors excepted). NADCON is rapidly being accepted as the standard for NAD27 to NAD83 conversions.

Typically, this function is not called directly by an application. This function is typically accessed via a call to *CS_dtcvt* which in turns calls this function due to the presence of a reference to this conversion in the `cs_Dtcprm_` structure which is provided to *CS_dtcvt*. In this manner, application code does not directly reference any datum conversion software. New datum conversions can be added without modification of application code.

If, for some reason, an application needs to access this function directly, the application must invoke *CSnadinit* prior to the first call to this function.

The **blk_err** argument indicates the disposition of datum conversion errors caused by a lack of data covering the geographic region containing the coordinate to be converted. *CS_dtcvt* causes the value specified at the time of datum conversion set up to be passed to *CSnad283*. The valid values for this argument are:

cs_DTCFLG_BLK_I	Errors caused by data availability are silently ignored and a zero status value is to be returned.
cs_DTCFLG_BLK_W	Errors caused by data availability are reported directly to *CS_erpt* as a warning and a positive non-zero status value returned.
cs_DTCFLG_BLK_1	Errors caused by data availability are reported to *CSdtcbnf* and a positive non-zero status value returned. *CSdtcbnf* will suppress repeated error messages concerning the same one degree by one degree block and suppress all such error messages after 10 have been reported to *CS_erpt*.
cs_DTCFLG_BLK_F	Errors caused by data availability are reported directly to *CS_erpt* as a fatal error condition and a negative non-zero status value returned.

ERRORS

CSnad283 will return a -1 and set the value of global variable `cs_Error` appropriately if any of the following conditions are encountered:

cs_DTC_NO_SETUP	*CSnadinit* was not called prior to calling this function, or the affect of calling *CSnadinit* was canceled by a call to *CSnadcls*.
cs_INV_FILE	A required NADCON database file is corrupted beyond use.
cs_DTC_FILE	A required NADCON database file that does exist could not be opened.
cs_NADCON_CONS	Two properly named files that are supposed to represent a single NADCON database contain inconsistent information.

CSnad83284 NAD-83 TO wgs 84

```
int CSnad83284 (Const double ll_83 [2],double ll_84 [2]);
```

Currently, *CSnad83284* simply copies the latitude and longitude in the **ll_83** array to the *ll_84* array and returns **FALSE** to indicate that it did so successfully.

There are differences between NAD83 and WGS 84. However, both are very accurate measurements of the same thing. Therefore, the differences are slight and are within the tolerance of error associated with WGS-84. Currently, there are no generally accepted techniques for converting one to the other. This function is a hook to provide such a conversion should a generally recognized technique become available in the future.

ERRORS

CSnad83284 returns **FALSE** to indicate success. Future versions may return **TRUE** to indicate an error condition of some sort.

CSnadcls North American Datum, CLoSe

```
void CSnadcls (void);
```

CSnadcls decrements the global variable `csNadcnt` and, if the result is zero or less, closes all open NADCON database files and *free*'s the memory allocated to the NADCON file control block cache and the NADCON grid cell cache.

CSnadinit increments the global variable `csNadcnt` each time it is called. *CSnadcls* decrements this count each time it is called. Resources are released only when `csNadcnt` reaches zero. Thus, resources are not released prematurely in processes where more than one datum conversion process is active at any given time.

Currently, *CSnadcls* does not *free* the memory allocated to the NADCON database directory. The NADCON database directory does not require much memory (32 bytes per database) and is rather expensive, computationally, to initialize. Therefore, this system resource is left alone by this function. *CSnadinit* will not attempt to reinitialize the directory if the value of `csNaddirP` is not **NULL**. Non-source licensees who find this objectionable can free the NADCON database directory directly (i.e. *free (*`csNaddirP);`). Be sure to set the value of `csNaddirP` to **NULL** and the value of `csNaddirM` and `csNaddirU` to zero.

CSnaddbo North American Datum, DataBase Open

```
extern int cs_Error;
Const struct csNadfcb_ *CSnaddbo (Const double ll_27 [2]);
```

The database for NADCON datum conversions consists of several different file systems, each covering a specific geographic area and consisting of two separate files. *CSnaddbo* returns a pointer to a `csNadfcb_` structure that provides access to the appropriate NADCON database file system for the geographic region containing the coordinate indicated by the **ll_27** argument. **ll_27** must contain longitude in the first element, and latitude in the second element. Both must be in degrees where negative values are used to indicate west and south.

CSnaddbo searches the NADCON file control block cache to see if the required database file system is already open. If so, a pointer to it is returned after this block is made the most recently accessed (i.e. moved to the top of the linked list). Otherwise, the appropriate NADCON file system is opened and a pointer to the resulting file control block is returned.

If a NADCON database file system that covers the geographic region containing **ll_27** cannot be found on the system, `cs_Error` is set to zero and the **NULL** pointer is returned. If the database open failed for another reason, the **NULL** pointer will be returned but `cs_Error` will be set to indicate the nature of the failure.

CSnaddbo is called by *CSnadptr* when *CSnadptr* needs to access a database to fetch a new grid cell. It is not typically called by an application program directly. Should an application program need to access this function directly, *CSnadinit* must be called prior to the first call to this function.

ERRORS

For failures due to causes other than the availability of appropriate data, *CSnaddbo* will return the **NULL** pointer and set `cs_Error` as follows:

cs_DTC_NO_SETUP	*CSnadinit* was not called prior to calling this function, or the affect of calling *CSnadinit* was canceled by a call to *CSnadcls*.
cs_INV_FILE	A required NADCON database file is corrupted beyond use.
cs_DTC_FILE	A required NADCON database file that does exist could not be opened.
cs_NADCON_CONS	Two properly named files that are supposed to represent a single NADCON database contain inconsistent information.

CSnaddir NADcon database DIRectory

```
Const struct csNaddir_ *CSnaddir (Const double ll_27 [2]);
```

CSnaddir returns a pointer to the `csNaddir_` structure in the NADCON database file directory that contains the grid cell required to convert the coordinate given by the **ll_27** argument. The first element of **ll_27** must be the longitude of the coordinate to be converted; the second element must contain the latitude. Both elements must be in degrees. Negative values are used to indicate west longitude and south latitude.

In the event a database covering the specific location provided by the **ll_27** argument cannot be located, *CSnaddir* will set `cs_Error` to zero and return the **NULL** pointer.

CSnaddbo, upon determining that a NADCON database file system needs to be opened, calls *CSnaddir* to determine the base name of the database required to convert a specific coordinate. The NADCON database directory pointed to by `csNaddirP` contains a list of the base names of all NADCON database file systems present on the system, and the geographic region covered by each.

Application programs do not normally call this function directly. Should an application need to do so, *CSnadinit* must be called prior to the first call to this function.

ERRORS

CSnaddir will return the **NULL** pointer, and set `cs_Error` to the indicated value if any of the following conditions are encountered:

cs_DTC_NO_SETUP	*CSnadinit* was not called prior to calling this function, or the affect of calling *CSnadinit* was canceled by a call to *CSnadcls*.

CSnadinit North American Datum, INITialize

```
int CSnadinit (void);
```

CSnadinit causes the application program to be initialized for the conversion of coordinates based on the North American Datum of 1927 (NAD27) to coordinates based on the North American Datum of 1983 (NAD83).

CSnadinit is not usually called directly by application code. *CSnadinit* is usually called by *CS_dtcsu* when a datum conversion involving NAD27 to NAD83 or NAD83 to NAD27 is requested. Applications which access *CSnad227*, *CSnad283*, *CSnadptr*, *CSnaddbo*, or *CSnaddir* directly will need to call this function prior to the first call to any of these functions.

CSnadinit will return a zero value to indicate that the NAD27 to NAD83 (or vice versa) conversion has been properly initialized. A positive non-zero status is returned if the initialization failed due to a lack of data files. A negative non-zero status is returned if the initialization failed due to a system error such as a physical I/O error or insufficient memory.

CSnadinit allocates and initializes the NADCON database directory (`csNaddirP`), the NADCON file control block cache (`csNadfcb`) and the NADCON grid cell cache (`csNadgrd`). The effects of *CSnadinit* upon system resources can be undone by calling *CSnadcls*.

CSnadinit may be called repeatedly without adverse affects. However, be aware that *CSnadinit* increments the global variable `csNadcnt` once each time it is called. *CSnadcls* decrements this count and releases system resource only when the count reaches zero.

ERRORS

CSnadinit will return a negative non-zero value and set `cs_Error` to the indicated value should any of the following conditions be encountered:

cs_NO_MEM	Insufficient memory is available to allocate the NADCON database directory, the NADCON file control block cache, or the NADCON grid cell cache.

CSnadptr North American Datum, return grid cell PoinTeR

```
extern int cs_Error
Const struct csNadgrd_ *CSnadptr (Const double ll_27 [2]);
```

CSnadptr will return a pointer to a `csNadgrd_` grid cell structure appropriate for use in converting the coordinate given by the **ll_27** argument. The **NULL** pointer is returned if a grid cell for the coordinate

could not be returned. If the cause of the failure was simple non-availability of the data, `cs_Error` is set to zero. Otherwise, `cs_Error` will contain the appropriate error code indicating the nature of the failure.

CSnadptr returns a pointer to an entry in the grid cell cache pointed to by `csNadgrd` after recording the returned entry the most recently accessed by making it the first in the linked list. Obviously, if the required grid cell does not already exist in the cache, it must be fetched from disk. In so doing, *CSnadptr* always uses the last entry in the linked list that will always be the least recently accessed grid cell. Therefore, repeated requests for the grid cell covering coordinates in the same local geographic region will be satisfied with a minimum of disk I/O.

To a certain extent, increasing the number of entries in the grid cell cache will improve performance on conversion projects that are large both geographically and in the number of points to be converted. However, since the cache is searched linearly, a point of diminishing returns can be reached. The number of grid cell cache entries is specified by the value contained in the `csNadccnt` (see CSdcdata.c) global variable. This variable must be set to the desired value prior to the first call to *CS_dtcsu*.

CSnadptr is normally called by *CSnad283* to obtain the appropriate grid cell for each coordinate to be converted. Applications do not normally call this function directly. Should an application need to access this function directly, *CSnadinit* must be called prior to the first call to this function.

ERRORS

CSnadptr will return the **NULL** pointer, and set `cs_Error` to the indicated value if any of the following conditions are encountered:

cs_DTC_NO_SETUP	*CSnadinit* was not called prior to calling this function, or the affect of calling *CSnadinit* was canceled by a call to *CSnadcls*.
cs_INV_FILE	A required NADCON database file is corrupted beyond use.
cs_DTC_FILE	A required NADCON database file that does exist could not be opened.
cs_NADCON_CONS	Two properly named files that are supposed to represent a single NADCON database contain inconsistent information.

CHAPTER 5

Chapter 5 -- Data Modules

In this chapter the several data modules are defined. Most all static data constants are defined in the modules which do not contain any executable code.

CSdata -- general DATA module

```
extern char cs_Dir [];
extern char *cs_DirP;
extern char cs_DirsepC;
extern char cs_OptchrC;
extern char cs_ExtsepC;
extern int cs_Sortbs;
extern int cs_Error, cs_Errno;
extern int csErrlat, csErrlng;
extern char cs_Csname[];
extern char cs_Dtname[];
extern char cs_Elname[];
extern char cs_Envvar [];
extern char csErrnam [];
extern double cs_AnglTest;
extern double cs_AnglTest1;
extern double cs_SclInf;
extern double cs_MinLat, MaxLat;
extern double cs_MinLng, MaxLng;
extern double cs_MinLatFz, MaxLatFz;
extern double cs_MinLngFz, MaxLngFz;
extern double cs_NPTest, cs_SPTest;
extern double cs_EETest, cs_WETest;
extern struct cs_Grptbl_ cs_CsGrptbl [];
extern char cs_Mentor [];
extern short cs_Protect;
extern char cs_Unique;
```

CSdata is the module in which these global variables are defined and initialized to an appropriate value. This module contains no executable code.

cs_Dir

This character array must always contain the path to the directory in which all Coordinate System Mapping Package data files reside. This name must end with directory separator character.

cs_DirP

This character pointer must point to the character that immediately follows the directory separator character that follows the last directory name in `cs_Dir`. That is, `cs_Dir` and `cs_DirP` must be initialized at all times such that the following code will produce access to any specific Coordinate System Mapping Package data file:

```
extern char cs_Dir [];
extern char *cs_DirP;
{
     (void) strcpy (cs_DirP,"file_name");
     fd = open (cs_Dir,O_MODE);
}
```

Setting `cs_Dir` and `cs_DirP` up properly is very simple using *CS_stcpy*. For example:

```
cs_DirP = CS_stcpy (cs_Dir,"C:\\MAPPING\\");
```

cs_DirsepC

This character variable contains the directory separator character in use on the executing system.

cs_OptchrC

This character variable contains the command line option character in use on the executing system.

cs_ExtsepC

This character variable contains the extension separator character in use on the executing system.

cs_Error cs_Errno

Prior to returning a failed status code (i.e. either -1, **TRUE**, or the **NULL** pointer as the case may be) each function sets `cs_Error` to a code value indicating the specific nature of the problem. The value of the system's `errno` is saved in the `cs_Errno` variable. Neither of these variables should be examined unless a function has returned a failed status indication. The specific code values used to identify specific causes of failure are defined in the *cs_map.h* file.

csErrlng csErrlat

These integer variables are used to communicate the integer portion of the latitude and longitude of datum conversion data availability errors to *CS_erpt*. In this manner, the specific geographic area for which coverage doesn't exist can be reported to the user.

cs_Csname

This array carries the Coordinate System Dictionary file name that is currently in use.

cs_Dtname

This array carries the Datum Dictionary file name that is currently in use.

cs_Elname

This array carries the Ellipsoid Dictionary file name that is currently in use.

cs_Envvar

This array carries the environmental variable name currently which *CS_altdr* will use when extracting a directory path from the environment.

csErrnam

The name that precipitated certain error conditions is communicated to the error function through this globally defined character array. For example, when a file open fails, the name of the file is copied to this array before *CS_erpt* is called.

Mathematical Constants

Several mathematical constants, used throughout CS_MAP are declared globally here to reduce data and code space requirements. Some of these deserve special note:

cs_NPTest, cs_SPTest

These doubles are used to test for proximity to the north and south poles, respectively. Both values are in radians and are 0.001 arc seconds short of the respective pole.

cs_EETest, cs_WETest

These doubles are used to test for proximity to the singularity points of transverse projections. Read *EETest* as eastern extent test value, and *WETest* as the western extent test value. These values are in radians and are 0.001 seconds of arc short of ±/2.

cs_AnglTest, cs_AnglTest1

`cs_AnglTest` is set to 0.001 seconds of arc in radians. In many cases, especially geographic coordinates, absolute values less than this are considered zero. `cs_AnglTest1` is 1.0 minus `cs_AnglTest`.

cs_SclInf

This double is set to 9.999E+04, and is the value returned by scale functions when the true result would otherwise require a division by zero.

cs_MinLat, cs_MaxLat

These doubles are used, primarily, to test coordinate system definition parameters. They contain -90 and +90 degrees, respectively.

cs_MinLng, cs_MaxLng

These doubles are used, primarily, to test coordinate system definition parameters. They contain -180 and +180 degrees, respectively.

cs_MinLatFz, cs_MaxLatFz

These doubles are used, primarily, to test coordinate system definition parameters. In degrees, they represent values which are 0.01 seconds of arc short of ±90.

cs_MinLngFz, cs_MaxLngFz

These doubles are used, primarily, to test coordinate system definition parameters. In degrees, they represent values which are 0.01 seconds of arc short of ±180.

cs_SclRedMin, cs_SclRedMax

These doubles carry the minimum and maximum values of scale reduction which the coordinate system definition check functions (i.e. the projection Q functions) will accept. Note, that these values are set to accept values equal to, and slightly greater than, 1.0. Such values are occasionally used.

cs_DelMax, cs_RotMax, cs_SclMax

These doubles carry the maximum values allowed for Molodensky, Seven Parameter, and Bursa/Wolf datum conversions. `cs_Delmax`, `cs_RotMax`, and `cs_SclMax` are the maximum values allowed for translation, rotation, and scaling respectively.

cs_Protect

This variable controls how the dictionary protection system operates. Set this variable to -1 to disable the entire protection scheme. Set this variable to zero to enable the protection of distribution definitions, but to disable the protection of user defined definitions. A value greater than zero indicates the number of days after which an unchanged user definition becomes protected.

Mentor distributions have this variable set to 60. This implies that a user defined definition which remains unaltered for 60 days automatically becomes protected and cannot be deleted or further modified.

Note, for the purpose of this feature, a distribution system is defined as one produced by the Dictionary compiler. (Actually, any definition whose *protect* member is set to +1.)

cs_Unique

CS-MAP normally requires that the key names applied to user defined dictionary entries contain the colon character in the key name somewhere. Since Mentor never generates a definition with a key name containing a colon, user definitions are unambiguously defined using this technique. Thus, the update procedure will never cause a user definition to be replaced with a distribution system. This feature is controlled by the value of this global variable.

`cs_Unique` carries the character which is actually used for this purpose. If you like this feature, but prefer a different character, simply change the character to which this variable is set. If you don't like this feature, setting the value of this variable to '\0', i.e. the null character, essentially disables the entire feature.

Coordinate System Group Table

The coordinate system group table is initialized in this module. It consists of an array of `cs_Grptbl_` structures, terminated by an entry that has the null string as a group name. The group table performs two functions. First, presence of an entry in this table makes a specific group name valid. Second, it associates a descriptive string with the group name.

CSdataPJ -- DATA, ProJection table

```
extern struct cs_Prjtab_ cs_Prjtab [];
extern struct cs_PrjprmMap_ cs_PrjprmMap [];
```

```
extern struct cs_Prjprm_ cs_Prjprm [];
```

CSdataPJ carries the definition and initialization of the projection table used within the Coordinate System Mapping Package. This module contains no executable code. The projection table assigns a code name to each supported projection and associates this name with a setup function and a projection code number.

The table also includes a full textual name for each projection, a numeric code which will uniquely identify each projection, and a bit mapped flag word which will provide information specific to each projection. The meaning of each bit in the flag word is defined by manifest constants in *cs_map.h*. It is this flag word that is copied to the `cs_Csprm_` structure when a coordinate system definition is initialized by *CS_csloc*. The manifest constants and their meaning are:

cs_PRJFLG_SPHERE	a spherical form of this projection is supported.
cs_PRJFLG_ELLIPS	an ellipsoidal form of this projection is supported.
cs_PRJFLG_SCALK	analytical K scale is available
cs_PRJFLG_SCALH	analytical H scale is available.
cs_PRJFLG_CNFRM	projection is generally considered to be conformal.
cs_PRJFLG_EAREA	projection is generally considered to be equal area.
cs_PRJFLG_EDIST	projection is generally considered to be equidistant.
cs_PRJFLG_AZMTH	projection is generally considered to be azimuthal.
cs_PRJFLG_GEOGR	projection returns geographic coordinates (i.e. Unity).
cs_PRJFLG_OBLQ	oblique projection.
cs_PRJFLG_TRNSV	transverse projection.
cs_PRJFLG_PSEUDO	generally considered a pseudo projection, i.e. pseudo cylindrical.
cs_PRJFLG_INTR	projection is interruptable..
cs_PRJFLG_CYLND	generally considered cylindrical.
cs_PRJFLG_CONIC	generally considered to be a conic.
cs_PRJFLG_FLAT	generally considered to be an azimuthal (i.e. flat plane).
cs_PRJFLG_OTHER	something other than cylindrical, conic, or azimuthal.
cs_PRJFLG_SCLRED	the projection requires the specification of scale reduction.
cs_PRJFLG_ORGFLS	the projection does not use either false origin feature.

cs_PRJFLG_ORGLAT	the projection does not use the origin latitude parameter.
cs_PRJFLG_ORGLNG	the projection does not use the origin longitude parameter.

To add new projections to the system, one adds an entry to this table providing the name that is to be used to reference the projection and the setup module to be called to prepare for the use of the projection. Of course, one must code the setup function and its companions.

The `cs_PrjprmMap` array defines the use of all 31 (currently) parameters for each of the 38 supported projections. Each of the definitions consists of an index into the `cs_Prjprm` array described next.

The `cs_Prjprm_` array carries a definition for each of the (currently 31) different parameter types used in coordinate system definitions. For example, a "Northern Standard Parallel" parameter may be used in four or more projections, but is defined once in this array.

At times, such as our own dictionary compiler *CS_COMP*, it is desirable to have access to portions of the projection table, without incurring the penalty of having the entire repertoire of CS-MAP projection functions added to your executable module. You may now compile *CSdataPJ* with the **__NO_SETUP__** manifest constant defined and obtain a projection table with the **NULL** pointer for all setup functions. Similarly, compiling with **__NO_QCHK__** defined will provide **NULL** pointers for the projection check functions. Combining the two will provide a projection table suitable for, as an example, checking projection names without referencing any other code.

CSdataU -- DATA module, Units table

```
extern struct cs_Unittab_ cs_Unittab [];
```

The units table, i.e. that which enables *CS_unitlu* to perform its function in life, is defined in this module. This module contains no executable code. An entry appears in the table for each supported unit, and each unit is classified as being of the length or angular type. The table has provisions for a full name and an abbreviation for each supported unit. The factor for each entry must be the multiplier required to convert values in the units being defined to meters or degrees.

The table is terminated by an entry with a zero `type` value. The string " z" (i.e. space lowercase z) is used to indicate that an abbreviation entry is not to be used. The table is searched linearly, so the order of entries is, currently, unimportant.

Currently supported linear and angular units of length are listed in the tables given below. Check the source module for the latest unit definitions. Of course, you can add additional units by simply making a new table entry and recompiling.

Note, as of release 11, the table has been modified such that it includes the plural name of the unit (i.e. FEET in addition to FOOT) and also classifies each linear unit as belonging to the metric or english class of units.

Name	Abbreviation	Description
Meter	MT	Meter
Foot	FT	US Survey Foot
Inch	IN	US Survey Inch
Ifoot	IFT	International Foot
ClarkeFoot		Clarke's Foot
Iinch	IIN	International Inch
Centimeter	CM	Centimeter
Kilometer	KM	Kilometer
Yard	YD	English Yard
SearsYard		Sears Yard
Mile	MI	US Survey Statue Mile
Iyard	IYD	International Yard
Imile	IMI	International Mile
Knot	KT	Nautical Mile
NautM	NM	Nautical Mile
Decimeter	DM	Decimeter
Millimeter	KM	Millimeter
Decameter		Decameter
Hectometer		Hectometer
GermanMeter		German Legal Meter
CaGrid		Canadian Grid Unit
GunterChain		Chain, Gunter
ClarkeChain		Chain, Clarke

BenoitChain		Chain, Benoit
SearsChain		Chain,Sears
GunterLink		Link, Gunter
ClarkeLink		Link, Clarke
BenoitLink		Link, Benoit
SearsLink		Link, Sears
Rod		Rod
Perch		Perch
Pole		Pole
Furlong		Furlong
Rood		South African Rood
CapeFoot		Cape Foot

Supported Linear Units

Brealey		Brealey Unit

Name	Abbreviation	Description
DEGREE	DG	Degrees
GRAD	GR.	Grad
GRADE	GR.	Grad
MAPINFO	MI	Degrees * 1,000,000 for MapInfo
MIL		6400 mils = 1 degree
MINUTE	MN	1/60th of a degree
RADIAN	RD	57.295.... degrees
SECOND	SEC	1/3600th of a degree
DECISECOND		1/10th of a second of arc
CENTISECOND		1/100th of a second of arc
MILLISECOND		1/1,000 of a second of arc

Supported Angular Units