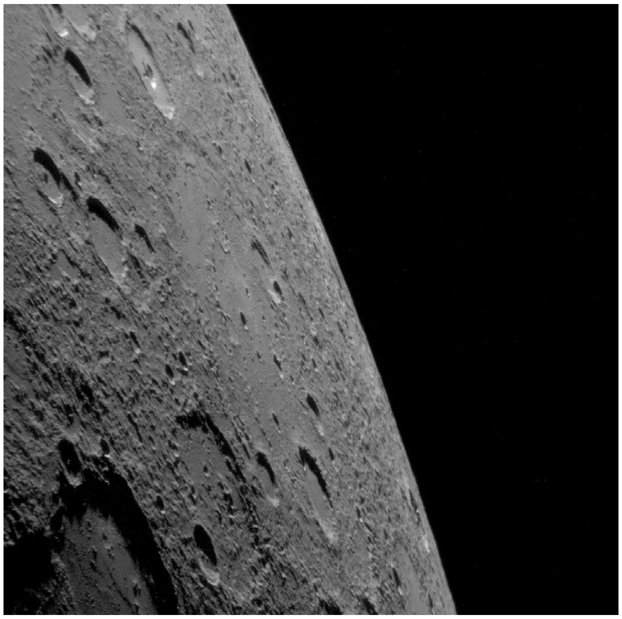
General Astrodynamics Library Reference Manual



Version 0.6

General Astrodynamics Library

Reference Manual
Version 0.6
April 22, 2011

Email: vp9mu@amsat.org

Cover photograph courtesy of NASA: Mercury from the Messenger spacecraft.

Contents

Preface

This manual documents the use of the General Astrodynamics Library, a numerical library for C, C++, and Objective C (Cocoa) programmers. The GAL Project is an attempt to gather a comprehensive set of astrodynamics routines in a single library and in a consistent form.

The project started life as an extension to the GNU Scientific Library, however once the authors discovered the IAU's SOFA Library it was decided to drop GSL and adopt SOFA. Much of the core functionality of SOFA is directly applicable to Astrodynamics applications. The main reason for dropping GSL compatibility was the GSL approach to matrix and vector storage – an overly complicated scheme. The GAL implementations of the SOFA routines do differ from the official IAU versions. This is mainly due to the different mechanism for reporting error and warning codes.

Starting with version 0.6, GAL contains some routines derived from NAIF SPICE Toolkit. The NAIF SPICE Toolkit is an excellent piece of work, and has been valuable in providing many test cases for GAL. A lot of SPICE overlaps with the functionality of GAL, and so only the value add routines have been included in GAL. SPICE is a complete eco-system, more like an application than a library. This is not a complaint merely an observation. GAL's implementations of the SPICE routines are low level, and without the extensive SPICE safety net, and associated overheads.

The test framework is central to GAL's design, nearly all routines have a corresponding test routine. The test framework allows routines to be upgraded as new techniques are published, whilst ensuring that nothing gets broken in the process. Users of the library may also use the test routines as examples of how to use the main routines.

The General Astrodynamics Library is *free software*. The term “free software” is sometimes misunderstood – it has nothing to do with price. It is about freedom. It refers to your freedom to run, copy, distribute, study, change and improve the software. With the General Astrodynamics Library you have all those freedoms.

Paul Willmott
Somerset, Bermuda
March 8, 2010

Chapter 1 – Introduction

The General Astrodynamics Library (GAL) is a collection of routines for numerical computing. The routines have been written in C, and present a modern Applications Programming Interface (API) for C, C++, and Objective C (Cocoa) programmers, allowing wrappers to be written for very high level languages. The source code is distributed under the GNU General Public License.

What is new in version 0.6?

The GAL Team are sorry to announce that the interfaces for many routines have been changed from prior versions of GAL. Many attempts were made to provide for backwards compatibility between 0.6 and 0.5, but these attempts just created more pain. So it was decided to bite the bullet and fix those areas of GAL that have become strained since the release of version 0.1.

A major change has been the introduction of a status recording mechanism. This mechanism replaces the simple integer return codes for a large number of routines. The mechanism provides a more robust status (error and warning) reporting mechanism for the user. This is particularly important now that routines are getting more complex, and the routine calls deeper.

This change does mean that the GAL and corresponding SOFA routines now have different parameters in many cases. Given this fact the opportunity has been taken to make other SOFA derived routines more consistent with related but not SOFA derived routines.

The implementation of gravity models have been changed from using global external variables to use provider routines.

The header file gal_const.h has been removed, and it's functions split across several new header files.

Version 0.6 sees the introduction of the following new features:

- JPL and IMCCE ephemerides
- NAIF SPICE SPK file support
- Osculating to Mean Keplerian element conversion
- US Strategic Command Satellite Catalog
- Gravity model import/export
- Earth orientation parameters import

Routines available in GAL

The library covers a wide range of topics in astrodynamics computing. Routines are available for the following areas:

- Vector and Matrix Manipulation
- Dates and Times
- Ellipsoids
- Earth Orientation
- Reference Frames
- Ephemerides
- SGP4 Propagation
- ODE Integrators
- Force Models
- Gravity Models
- Classical Keplerian Propagators

The use of these routines is described in this manual. Each chapter provides detailed definitions of the functions, including references to the articles upon which the algorithms are based.

The header file gal.h will include all the header files for the complete library.

Standards for Fundamental Astronomy Library (SOFA)

GAL is built upon an independent translation of the IAU's SOFA FORTRAN Library. The majority of the routines included in release 0.1 of GAL are translations of SOFA routines. These routines have not been verified by the IAU and are not supported by the IAU or the SOFA Review Board. Any errors introduced by the translation process are the responsibility of the GAL Team solely. That said, the GAL Team would like to thank Patrick Wallace, past Chair of the SOFA Review Board, for making the SOFA test suite available to the GAL Team, and for answering many questions and providing insight into the thinking behind the FORTRAN SOFA implementations. Since the release of version 0.1 of GAL an authorized IAU SOFA C library has been written and released. The results of GAL and the IAU C version have been compared, and shown that identical results are computed. The entries in this document for the SOFA derived routines are based upon the comments in the original SOFA Fortran code. SOFA derived routines added from December 31, 2009 onwards are directly derived from the official C SOFA versions and are not independent translations of the FORTRAN versions. They have however been modified to confirm to GAL formatting standards, status recording mechanism, and constant consistency with other routines. The individual header files detail any other differences from the code upon which the routine was derived. The user should also comply with the SOFA licence for these routines, this licence is included in the file SOFA_LICENSE.TXT in the distribution.

GAL is Free Software

The subroutines in the General Astrodynamics Library are “free software”; this means that everyone is free to use them, and to redistribute them in other free programs. The library is not in the public domain; it is copyrighted and there are conditions on its distribution. These conditions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of the software that they might get from you.

Specifically, we want to make sure that you have the right to share copies of programs that you are given which use the General Astrodynamics Library, that you receive their source code or else can get it if you want it, that you can change these programs or use pieces of them in new free programs, and that you know you can do these things.

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of any code which uses the General Astrodynamics Library, you must give the recipients all the rights that you have received. You must make sure that they, too, receive or can get the source code, both to the library and the code which uses it. And you must tell them their rights. This means that the library should not be redistributed in proprietary programs.

Also, for our own protection, we must make certain that everyone finds out that there is no warranty for the General Astrodynamics Library. If these programs are modified by someone else and passed on, we want their recipients to know that what they have is not what we distributed, so that any problems introduced by others will not reflect on our reputation.

The precise conditions for the distribution of software related to the General Astrodynamics Library are found in the GNU General Public License.

Further information about this license is available from the GNU Project webpage Frequently Asked Questions about the GNU GPL,

http://www.gnu.org/copyleft/gpl-faq.html

Obtaining GAL

The source code for the library can be obtained in different ways, by copying it from a friend, or downloading it from the internet.

http://www.homepage.mac.com/pclwillmott/GAL/index.html

No Warranty

The software described in this manual has no warranty, it is provided “as is”. It is your responsibility to validate the behavior of the routines and their accuracy using the source code provided, or to purchase support and warranties from commercial redistributors. Consult the GNU General Public license for further details (see GNU General Public License).

Not all routines in the GAL library have corresponding test routines. This is usually due to the unavailability of independent third party test cases. Users should take particular care when using routines that do not have test routines. The GAL Team would greatly appreciate any worked examples that would fill some of these gaps.

Reporting Bugs

A list of known bugs can be found in the BUGS file included in the GAL distribution. Details of compilation problems can be found in the INSTALL file. If you find a bug which is not listed in these files, please report it to vp9mu@amsat.org. All bug reports should include:

- The version number of GAL
- The hardware and operating system
- The compiler used, including version number and compilation options
- A description of the bug behavior
- A short program which exercises the bug, showing actual and expected results

It is useful if you can check whether the same problem occurs when the library is compiled without optimization. Thank you. Any errors or omissions in this manual can also be reported to the same address.

Compatibility with C++

The library header files automatically define functions to have extern “C” linkage when included in C++ programs. This allows the functions to be called directly from C++.

Deprecated Functions

From time to time, it may be necessary for the definitions of some functions to be altered or removed from the library. In these circumstances the functions will first be declared deprecated and then removed from subsequent versions of the library. Functions that are deprecated can be disabled in the current release by setting the preprocessor definition GAL_DISABLE_DEPRECATED. This allows existing code to be tested for forwards compatibility.

ANSI C Compliance

The library is written in ANSI C and is intended to conform to the ANSI C standard (C89). It should be portable to any system with a working ANSI C compiler. The library does not rely on any non-ANSI extensions in the interface it exports to the user. Programs you write using GAL can be ANSI compliant. To avoid namespace conflicts all exported function names and variables have the prefix gal_, while exported macros have the prefix GAL_.

Free Software Needs Free Documentation

The following article was written by Richard Stallman, founder of the GNU Project. The biggest deficiency in the free software community today is not in the software - it is the lack of good free documentation that we can include with the free software. Many of our most important programs do not come with free reference manuals and free introductory texts. Documentation is an essential part of any software package; when an important free software package does not come with a free manual and a free tutorial, that is a major gap. We have many such gaps today. Consider Perl, for instance. The tutorial manuals that people normally use are non-free. How did this come about? Because the authors of those manuals published them with restrictive terms - no copying, no modification, source files not available - which exclude them from the free software world. That wasn't the first time this sort of thing happened, and it was far from the last. Many times we have heard a GNU user eagerly describe a manual that he is writing, his intended contribution to the community, only to learn that he had ruined everything by signing a publication contract to make it non-free. Free documentation, like free software, is a matter of freedom, not price. The problem with the non-free manual is not that publishers charge a price for printed copies - that in itself is fine. (The Free Software Foundation sells printed copies of manuals, too.) The problem is the restrictions on the use of the manual. Free manuals are available in source code form, and give you permission to copy and modify. Non-free manuals do not allow this. The criteria of freedom for a free manual are roughly the same as for free software. Redistribution (including the normal kinds of commercial redistribution) must be permitted, so that the manual can accompany every copy of the program, both on-line and on paper. Permission for modification of the technical content is crucial too. When people modify the software, adding or changing features, if they are conscientious they will change the manual too—so they can provide accurate and clear documentation for the modified program. A manual that leaves you no choice but to write a new manual to document a changed version of the program is not really available to our community. Some kinds of limits on the way modification is handled are acceptable. For example, requirements to preserve the original author's copyright notice, the distribution terms, or the list of authors, are ok. It is also no problem to require modified versions to include notice that they were modified. Even entire sections that may not be deleted or changed are acceptable, as long as they deal with nontechnical topics (like this one). These kinds of restrictions are acceptable because they don't obstruct the community's normal use of the manual. However, it must be possible to modify all the technical content of the manual, and then distribute the result in all the usual media, through all the usual channels. Otherwise, the restrictions obstruct the use of the manual, it is not free, and we need another manual to replace it. Please spread the word about this issue. Our community continues to lose manuals to proprietary publishing. If we spread the word that free software needs free reference manuals and free tutorials, perhaps the next person who wants to contribute by writing documentation will realize, before it is too late, that only free manuals contribute to the free software community. If you are writing documentation, please insist on publishing it under the GNU Free Documentation License or another free documentation license. Remember that this decision requires your approval - you don’t have to let the publisher decide. Some commercial publishers will use a free license if you insist, but they will not propose the option; it is up to you to raise the issue and say firmly that this is what you want. If the publisher you are dealing with refuses, please try other publishers. If you’re not sure whether a proposed license is free, write to licensing@gnu.org. You can encourage commercial publishers to sell more free, copylefted manuals and tutorials by buying them, and particularly by buying copies from the publishers that paid for their writing or for major improvements. Meanwhile, try to avoid buying non-free documentation at all. Check the distribution terms of a manual before you buy it, and insist that whoever seeks your business must respect your freedom. Check the history of the book, and try reward the publishers that have paid or pay the authors to work on it. The Free Software Foundation maintains a list of free documentation published by other publishers:

http://www.fsf.org/doc/other-free-books.html

SOFA Julian Date Format

GAL uses Julian Dates stored in standard SOFA two-piece format. The Julian Date is apportioned in any convenient way between two arguments. For example, the Julian Date 2450123.7 could be expressed in any of these ways, among others:

2450123.7	0.0	Julian Date method
2451545.0	-1421.3	J2000 method
2400000.5	50123.2	Modified Julian Date method
2450123.5	0.2	date & time method

The GAL routines are optimized assuming that the first date argument is of a greater magnitude than the second argument. The routines will work with either ordering, but greatest precision is obtained by using the recommended ordering.

Position & Velocity Vectors

GAL stores position and velocity vectors in a single 2 by 3 array. This allows both vectors to be passed to functions as a single entity. The combined position and velocity vectors’ array is called a pv-vector.

pv[0][0]	x position
pv[0][1]	y position
pv[0][2]	z position
pv[1][0]	x velocity
pv[1][1]	y velocity
pv[1][2]	z velocity

A pv-vector may be split into individual p-vectors (1 by 3 array).

Status codes

As of version 0.6 GAL provides a unified and consistent mechanism for recording and reporting errors and warnings to the routines that call the GAL routines. Prior to version 0.6 errors and warnings were reported via integer function return values. As routines became more complex and deeper (i.e. layered calls to more and more GAL routines), this mechanism broke down.

The mechanism introduced in version 0.6 is based upon linked lists encapsulated within the GAL status structure. More details about this mechanism can be found in Chapter 2 – Status Routines.

Using GAL with Apple's Xcode

By default GAL will be installed in the path `/usr/local` . Unfortunately this path is invisible to the Finder on OS-X systems. However the library is still usable in Xcode projects provided that a few steps are taken.

Inside the Xcode project that you wish to link with GAL, select "Edit Project Settings" from the "Project" menu. Select the "Build" tab. Xcode stores separate configurations for build and release so you have to do the following in both the build and release configurations. Scroll down to the "Linking" section. Under "Other Linker Flags" enter `-lgal`, this tells Xcode to link the project to the GAL library. Under "Header Search Paths" enter `/usr/local/include` . Under "Library Search Paths" enter `/usr/local/lib` . If you installed GAL anywhere apart from the default then change the previous paths as necessary.

When GAL is installed it is compiled for the host machine processor. Xcode by default compiles universal binaries, i.e. the same program can run on both Intel and PPC machines. GAL cannot be linked to a program compiled as a universal binary. Again in the Build tab near the top remove from "Valid Architectures" any references that do not refer to the host machine processor. For modern Intel machines you should just select "x86_64".

Save the changes and that is it, GAL should link correctly now.

Chapter 2 – Status Routines

The routines detailed in this chapter are defined in the “gal_status.h” header file. The header file “gal_status_t.h” defines the status structure types. These structures should only be manipulated by the GAL status routines. Each status structure instance contains two lists one for errors and one for warnings. Errors indicate that the called routine did not complete successfully or that invalid parameters were passed to a routine. It is the responsibility of the user to monitor these and take appropriate actions. Warnings are an indication that the routines completed successfully but detected something that the user may wish to investigate further. The status recording mechanism allows a routine to report many errors and warnings back to the user, it also allows a routine to pass back any errors and warnings set by a routine that is called by the routine. The status structure also records the filename and the line number of when the error or warning was registered. The routine gal_stsprn is useful for debugging.

For completeness the definitions of the structures are listed below:

```
/*
 * ------------------------------------------------
 * Type definitions for Status recoding structures
 * ------------------------------------------------
 */

#define GAL_STS_MAXFILENAME (256)

struct snode {
  int status ;
  int linenum ;
  char filename[GAL_STS_MAXFILENAME] ;
  struct snode *next ;
} ;

typedef struct snode gal_snode_t ;

typedef struct {
  int errc ;
  gal_snode_t *errors ;
  int warnc ;
  gal_snode_t *warnings ;
} gal_status_t ;
```

The header file gal_status.h defines the following constants to represent the GAL error and warning codes. The user should use these constants rather than the numeric values as the numeric values may be changed in subsequent releases of GAL.

```
/*
 * --------------------------------
 * Constants for Error Status Codes
 * --------------------------------
 */

enum {

  GAL_SUCCESS                    =  0,
  GAL_FAILURE                    =  1,
  GAL_NOT_FOUND                  =  2,
  GAL_DISTANCE_OVERRIDEN         =  3,
  GAL_EXCESSIVE_VELOCITY         =  4,
  GAL_NO_CONVERGENCE             =  5,
  GAL_BAD_YEAR                   =  6,
  GAL_BAD_MONTH                  =  7,
  GAL_BAD_DAY                    =  8,
  GAL_BAD_FRACTION               =  9,
  GAL_ALLOC_FAILED               = 10,
  GAL_INVALID_ID                 = 11,
  GAL_OUTSIDE_DATE_RANGE         = 12,
  GAL_STEPSIZE_UNDERFLOW         = 13,
  GAL_STEPSIZE_TOO_SMALL         = 14,
  GAL_OUT_OF_RANGE               = 15,
  GAL_NULL_VECTOR                = 16,
  GAL_SYSTEM_ERROR               = 17,
  GAL_INVALID_ECCENTRICITY       = 18,
  GAL_INVALID_SMA                = 19,
  GAL_INVALID_SEMI_LATUS_RECTUM = 20,
  GAL_EPOCH_SUB_ORBITAL          = 21,
  GAL_SATELLITE_DECAYED          = 22,
  GAL_NULL_POINTER               = 23,
  GAL_INVALID_INDEX              = 24,
  GAL_INVALID_ERROR_CODE         = 25,
  GAL_INVALID_WARNING_CODE       = 26,
  GAL_EXCESSIVE_FACTORIAL        = 27,
  GAL_INVALID_FLATTENING         = 28,
  GAL_ARITHMETIC_EXCEPTION       = 29,
  GAL_INVALID_MEAN_MOTION        = 30,
  GAL_MAX_STEPS_EXCEEDED         = 31,
  GAL_NOT_GRAVITY_MODEL          = 32,
  GAL_INVALID_DOUBLE             = 33,
  GAL_INVALID_INTEGER            = 34,
  GAL_FILE_NOT_OPEN              = 35,
  GAL_INVALID_DEGREE             = 36,
  GAL_INVALID_ORDER              = 37,
  GAL_INVALID_FORMAT             = 38,
  GAL_INVALID_STRING             = 39,
  GAL_UNEXPECTED_EOF             = 40,
  GAL_DATES_NOT_CONTIGUOUS       = 41,
  GAL_FILE_NOT_FOUND             = 42,
  GAL_INVALID_DATA_TYPE_FORMAT   = 43,
  GAL_INVALID_DATE_RANGE         = 44,
  GAL_COEFFICIENTS_NOT_FOUND     = 45,
  GAL_STSADD_BREAKPOINT          = 46,
  GAL_OPEN_FILE_FAILED           = 47,
  GAL_INVALID_COORDINATE         = 48,
  GAL_CANNOT_INVERT_MATRIX       = 49,
  GAL_UNKNOWN_SPK_TYPE           = 50,
  GAL_INVALID_GM                 = 51,
  GAL_ZERO_POSITION              = 52,
  GAL_ZERO_VELOCITY              = 53,
  GAL_NON_CONIC_MOTION           = 54,
  GAL_INVALID_SIZE               = 55,
  GAL_INVALID_STEP_SIZE          = 56,
  GAL_DIVIDE_BY_ZERO             = 57,
  GAL_VECTORS_NOT_ORTHOGONAL     = 58,

/*
 * ----------------------------------
 * Constants for Warning Status Codes
 * ----------------------------------
 */

  GAL_DUBIOUS_YEAR        = 10000,
  GAL_NO_OCCULTATION      = 10001,
  GAL_PARTIAL_OCCULTATION = 10002,
  GAL_TOTAL_OCCULTATION   = 10003,
  GAL_NEVER_SETS          = 10004,
  GAL_NEVER_RISES         = 10005,
  GAL_NOT_0_TO_9          = 10006,
  GAL_MAX_DEGREE_SET      = 10007,
  GAL_MAX_ORDER_SET       = 10008,
  GAL_START_DATE_SET      = 10009,
  GAL_END_DATE_SET        = 10010,
  GAL_WARN_BAD_DAY        = 10011

} ;

#define GAL_MAXSYMLEN 64
```

gal_errc [0.6]

Return the number of errors set in the error list of an initialized status structure instance.

```
int
gal_errc
(
  gal_status_t *status
) ;
```

This routine returns the count of the number of errors recorded in the status structure instance.

Usage:

```
…
j = foo ( a, b, status ) ;
if ( gal_errc ( status ) ) {
/* handle error here */
}
```

gal_errget [0.6]

This routine returns the error code recorded at the specified index in the error list of an initialized status structure instance.

```
int
gal_errget
(
  int n,
  gal_status_t *status
) ;
```

This routine returns the error code recorded at the specified index. The index of the first entry is one (1).

gal_errisset [0.6]

This routine checks if a specific error code is set.

```
int
gal_errisset
(
  int err,
  gal_status_t *status
) ;
```

If the specified error code is set in the error list of the status structure instance then the routine returns 1, if the code is not set then 0 is returned.

Usage:

```
…
j = foo ( a, b, status ) ;
if ( gal_errisset ( GAL_INVALID_DOUBLE, status ) ) {
/* handle invalid double error here */
}
```

gal_errset [0.6]

Set an error code in the error list of an initialized status structure instance.

```
void
gal_errset
(
  char *filename,
  int linenum,
  int err,
  gal_status_t *status
) ;
```

This routine sets an error code in the error list of an initialized status structure instance. GAL error codes must be in the range 0-9999, user defined error codes must be in the range 20000-29999.

The routine only records the filename and line number first occurrence of an error code. The exception to this rule is for the GAL_STSADD_BREAKPOINT error code for which the most recent filename and line number are stored.

Usage:

```
/* Alloc failed, set error code */
gal_errset ( __FILE__, __LINE__, GAL_ALLOC_FAILED, status ) ;
```

gal_stsadd [0.6]

Add the contents of one status structure instance to another instance.

```
void
gal_stsadd
(
  gal_status_t *status1,
  gal_status_t *status2
) ;
```

This routine adds all the error and warning codes recorded in the STATUS1 instance to the error and warning lists in the structure instance STATUS2. Duplicates are discarded. It is recommended that the error code GAL_STSADD_BREAKPOINT is added to the returning status structure after the call to gal_stsadd. This enables the user to trace the source of the error.

Usage:

```
int
foo_top
(
  int a,
  int b,
  gal_status_t *status
)

{

  gal_status_t *stat ;

  int i ;

/* Initialize status structure */

  gal_stsinit ( status ) ;

/* Create status structure for calls */

  if ( (stat = gal_stsalloc () ) == NULL ) {
    gal_errset ( GAL_ALLOC_FAILED, status ) ;
    return 0 ;
  }

/* Call a routine */

  i = foo ( a, b, stat ) ;

/* Add all errors reported by foo to status */

  if ( gal_errc ( stat ) ) {
    gal_stsadd ( stat, status ) ;
    gal_errset ( __FILE__, __LINE,
    GAL_STSADD_BREAKPOINT, status ) ;
  }
/* Free private status structure */

  gal_stsfree ( stat ) ;

  return i ;

}
```

gal_stsalloc [0.6]

Creates an empty status recording structure.

```
gal_status_t *
gal_stsalloc
(
) ;
```

This routine creates a new GAL status structure instance. The instance is initialized as empty. Each routine that is passed a pointer to the structure as a parameter should initialize the instance with a call to gal_stsinit.

The status instance must be de-allocated by using the gal_stsfree routine.

gal_stsfree [0.6]

Free status structure created by gal_stsalloc.

```
void
gal_stsfree
(
  gal_status_t *status
) ;
```

This routine de-allocates the specified status structure instance. This routine must be called instead of free as the status instance allocates the error and warning lists dynamically which must also be de-allocated.

gal_stsgetsym [0.6]

This routine gets a symbolic string for the specified error or warning code.

```
int
gal_stsgetsym
(
  int n,
  char *sym
) ;
```

This routine gets a symbolic string for the specified error or warning code N. The symbol is the GAL constant name for the error or warning code. The routine returns GAL_SUCCESS if it found the specified error or warning code, and GAL_NOT_FOUND otherwise. Also see the routine gal_stsprn.

gal_stsinit [0.6]

This routine initializes a previously created status structure instance.

```
void
gal_stsinit
(
  gal_status_t *status
) ;
```

This routine initializes a status structure instance previously created with gal_stsalloc. If the instance contains any error or warning codes then they are removed.

gal_stsprn [0.6]

This routine writes the contents of the error and warning lists of a status structure instance to the selected file.

```
int
gal_stsprn
(
  FILE *file,
  gal_status_t *status
) ;
```

This routine writes the contents of the error and warning lists to the selected file. The file must be open for writing on entry to the routine, and closed after the call to gal_stsprn completes. The errors are written first in symbolic form (see gal_stsgetsym) , followed by the warnings again in symbolic form. If no errors or warnings are recorded in the status structure instance then nothing is written. This routine is intended as a debugging tool, and when used as such stderr would be the logical choice for the file.

gal_warnc [0.6]

Return number of warnings set in the warning list of an initialized status structure instance.

```
int
gal_warnc
(
  gal_status_t *status
) ;
```

This routine returns the count of the number of warnings recorded in the status structure instance.

gal_warnget [0.6]

This routine returns the warning code recorded at the specified index in the warning list of an initialized status structure instance.

```
int
gal_warnget
(
  int n,
  gal_status_t *status
) ;
```

This routine returns warning code recorded at the specified index. The index of the first entry is one (1).

gal_warnisset [0.6]

This routine checks if a specific warning code is set.

```
int
gal_warnisset
(
  int warn,
  gal_status_t *status
) ;
```

If the specified warning code is set in the warning list of the status structure instance then the routine returns 1, if the code is not set then 0 is returned.

gal_warnset [0.6]

Set an warning code in the warning list of an initialized status structure instance.

```
void
gal_warnset
(
  int warn,
  gal_status_t *status
) ;
```

This routine sets a warning code in the warning list of an initialized status structure instance. GAL warning codes must be in the range 10000-19999, user defined warning codes must be in the range 30000-39999.

Chapter 3 – Math Routines

The routines detailed in this chapter are defined in the gal_math.h header file.

The header file gal_math_macros.h (included by gal_math.h) defines the following constants and commonly used macros:

```
/*
 * -------------------
 * Numerical Constants
 * -------------------
 */

#define GAL_PI     (3.141592653589793238462643)    /* Pi                                       */
#define GAL_2PI    (6.283185307179586476925287)    /* 2 * Pi                                   */
#define GAL_R2H    (3.819718634205488058453210)    /* Radians to hours                         */
#define GAL_R2D    (57.29577951308232087679815)    /* Radians to degrees                       */
#define GAL_R2S    (13750.98708313975701043156)    /* Radians to seconds                       */
#define GAL_R2AS   (206264.8062470963551564734)    /* Radians to arc seconds                   */
#define GAL_H2R    (0.2617993877991494365385536)   /* Hours to radians                         */
#define GAL_D2R    (1.745329251994329576923691e-2) /* Degrees to radians                       */
#define GAL_S2R    (7.272205216643039903848712e-5) /* Seconds to radians                       */
#define GAL_AS2R   (4.848136811095359935899141e-6) /* Arc seconds to radians                   */
#define GAL_TURNAS (1296000.0)                     /* Arc seconds in a full circle             */
#define GAL_U2R    ( GAL_AS2R / 1e7 )              /* Units of 0.1 microarcsecond to radians   */
#define GAL_MAS2R  ( GAL_AS2R / 1e3 )              /* Milliarcseconds to radians
*/

/*
 * Macro to simulate the FORTRAN SIGN function
 */

#define GAL_SIGN( a, b) fabs ( a ) * ( ( ( b ) >= 0.0 ) ? 1.0 : -1.0 )

/*
 * Macro for Maximum value
 */

#define GAL_MAX( a, b ) ( a ) >= ( b ) ? ( a ) : ( b )


/*
 * Macro for Minimum value
 */

#define GAL_MIN( a, b ) ( a ) <= ( b ) ? ( a ) : ( b )


/*
 * Macro for Even check
 */

#define GAL_EVEN( a ) ( ( a ) % 2 == 0 )
```

```
/*
 * Macro for Odd check
 */

#define GAL_ODD( a ) ( ( a ) % 2 != 0 )


/*
 * Macro for DNNT
 */

#define GAL_DNNT( x ) ( ( int ) ( ( x ) >= 0.0 ? floor ( ( x ) + 0.5 ) : -floor ( 0.5 - ( x ) ) )
)

/*
 * Macro for BRCKTD
 */

#define GAL_BRCKTD( number, end1, end2 ) \
( ( end1 ) < ( end2 ) ) ? \
   GAL_MAX ( ( end1 ), GAL_MIN ( ( end2 ), ( number ) ) ) \
: \
   GAL_MAX ( ( end2 ), GAL_MIN ( ( end1 ), ( number ) ) )


/*
 * Constant for undefined results
 */

#define GAL_UNDEFINED DBL_MAX
```

gal_a2af [0.1]

Decompose angle in radians into degrees, arc-minutes, arc-seconds, and fraction.

```
void
gal_a2af
(
  int ndp,
  double angle,
  char *sign,
  int idmsf[4]
) ;
```

On entry NDP specifies the required resolution, and is interpreted as follows:

```
      NDP        resolution

       :      ...0000 00 00
      -7         1000 00 00
      -6          100 00 00
      -5           10 00 00
      -4            1 00 00
      -3            0 10 00
      -2            0 01 00
      -1            0 00 10
       0            0 00 01
       1            0 00 00.1
       2            0 00 00.01
       3            0 00 00.001
       :            0 00 00.000...
```

The largest positive useful value for NDP is determined by the size of angle, the format of double precision floating-point numbers on the target platform, and the risk of overflowing IDMSF[3]. On a typical platform, for angles up to 2π, the available floating-point precision might correspond to NDP equals 12. However, the practical limit is typically NDP equals 9, set by the capacity of a 32-bit IDMSF[3]. ANGLE is the angle in radians. On return SIGN contains '+' or '-', and IDMSF contains degrees, arc-minutes, arc-seconds, and fraction. The absolute value of ANGLE may exceed 2π. In cases where it does not, it is up to the caller to test for and handle the case where ANGLE is very nearly 2π and rounds up to 360 degrees, by testing for IDMSF[0] equals 360 and setting IDMSF[0-3] to zero.

gal_a2tf [0.1]

Decompose angle in radians into hours, minutes, seconds, and fraction.

```
void
gal_a2tf
(
  int ndp,
  double angle,
  char *sign,
  int ihmsf[4]
) ;
```

On entry NDP specifies required resolution, and is interpreted as follows:

```
      NDP        resolution

       :      ...0000 00 00
      -7         1000 00 00
      -6          100 00 00
      -5           10 00 00
      -4            1 00 00
      -3            0 10 00
      -2            0 01 00
      -1            0 00 10
       0            0 00 01
       1            0 00 00.1
       2            0 00 00.01
       3            0 00 00.001
       :            0 00 00.000...
```

ANGLE is the angle in radians. On return SIGN contains '+' or '-', and IHMSF contains hours, minutes, seconds, and fraction. The largest useful value for NDP is determined by the size of angle, the format of double floating-point numbers on the target platform, and the risk of overflowing IHMSF[3]. On a typical platform, for angle up to 2π, the available floating-point precision might correspond to NDP equals 12. However, the practical limit is typically NDP equals 9, set by the capacity of a 32-bit IHMSF[3]. The absolute value of ANGLE may exceed 2π. In cases where it does not, it is up to the caller to test for and handle the case where ANGLE is very nearly 2π and rounds up to 24 hours, by testing for IHMSF[0] equals 24 and setting IHMSF[0-3] to zero.

gal_anp [0.1]

Normalize angle into the range 0 <= a < 2π.

```
double
gal_anp
(
  double a
) ;
```

On entry A is the angle in radians. The routine returns the normalized angle.

gal_anpm [0.1]

Normalize angle into the range -π <= A < +π.

```
double
gal_anpm
(
  double a
) ;
```

A is the angle in radians.

gal_c2s [0.1]

p-vector to spherical coordinates.

```
void
gal_c2s
(
  double p[3],
  double *theta,
  double *phi
) ;
```

On return THETA and PHI contain the longitude and latitude angles in radians respectively. P can have any magnitude; only its direction is used. If P is null, zero THETA and PHI are returned. At either pole, zero THETA is returned.

gal_chbint [0.6]

Given the coefficients for the Chebyshev expansion of a polynomial, this returns the value and derivative of the polynomial evaluated at the input X.

```
void
gal_chbint
(
  double *cp,
  int degp,
  double *x2s,
  double x,
  double *p,
  double *dpdx
) ;
```

On entry:

*CP Pointer DEGP+1 Chebyshev polynomial coefficients.
DEGP Degree of polynomial.
*X2S Transformation parameters of polynomial.
X Value for which the polynomial is to be evaluated

Returned:

*P Value of the polynomial at X.
*DPDX Derivative of the polynomial at X.

References:

"The Chebyshev Polynomials" by Theodore J. Rivlin

"CRC Handbook of Tables for Mathematics"

gal_chbval [0.6]

Given the coefficients for the Chebyshev expansion of a polynomial, this returns the value of the polynomial evaluated at the input X.

```
void
gal_chbval
(
  double *cp,
  int degp,
  double *x2s,
  double x,
  double *p
) ;
```

On entry:

*CP Pointer DEGP+1 Chebyshev polynomial coefficients.
DEGP Degree of polynomial.
*X2S Transformation parameters of polynomial.
X Value for which the polynomial is to be evaluated

Returned:

*P Value of the polynomial at X.

References:

"The Chebyshev Polynomials" by Theodore J. Rivlin

"CRC Handbook of Tables for Mathematics"

gal_cp [0.1]

Copy a P vector.

```
void
gal_cp
(
  double p[3],
  double c[3]
) ;
```

On return C contains a duplicate of P.

gal_cpv [0.1]

Copy a PV vector

```
void
gal_cpv
(
  double pv[2][3],
  double c[2][3]
) ;
```

On return C contains a duplicate of PV.

gal_cr [0.1]

Copy an R matrix.

```
void
gal_cr
(
  double r[3][3],
  double c[3][3]
) ;
```

On return C contains a duplicate of R.

gal_d2tf [0.1]

Decompose days to hours, minutes, seconds, and fraction.

```
void
gal_d2tf
(
  int ndp,
  double days,
  char *sign,
  int ihmsf[4]
) ;
```

On entry NDP contains the resolution, and DAYS contain the interval in days. On return SIGN contains '+' or '-', and IHMSF contain the hours, minutes, seconds, and fraction. NDP is interpreted as follows:

NDP	resolution
:	...0000 00 00
-7	1000 00 00
-6	100 00 00
-5	10 00 00
-4	1 00 00
-3	0 10 00
-2	0 01 00
-1	0 00 10
0	0 00 01
1	0 00 00.1
2	0 00 00.01
3	0 00 00.001
:	0 00 00.000...

The largest positive useful value for NDP is determined by the size of DAYS, the format of double floating-point numbers on the target platform, and the risk of overflowing IHMSF[3]. On a typical platform, for DAYS up to 1.0, the available floating-point precision might correspond to NDP equals 12. However, the practical limit is typically NDP equals 9, set by the capacity of a 32-bit IHMSF[3]. The absolute value of days may exceed 1.0. In cases where it does not, it is up to the caller to test for and handle the case where days is very nearly 1.0 and rounds up to 24 hours, by testing for IHMSF[0] equals 24 and setting IHMSF[0-3] to zero.

gal_facexp_alloc [0.3]

This routine computes the factorial exponent lookup table required by the gal_factorial routine.

```
gal_facexp_t *
gal_facexp_alloc
(
  int max_factorial,
  gal_status_t *status
) ;
```

Returns a pointer to the factorial exponent lookup table if successful, returns NULL otherwise. MAX_FACTORIAL determines the maximum factorial for which exponents are determined. If an error occurs then the applicable error code is set.

gal_facexp_free [0.3]

Free factorial exponent lookup table.

```
void
gal_facexp_free
(
  gal_facexp_t *facexp
) ;
```

This routine frees a factorial exponent lookup table previously allocated by the gal_facexp_alloc routine. On entry the pointer FACEXP contains a pointer to a table previously allocated by gal_facexp_alloc.

gal_factorial [0.3]

Computes the factorial N!, or the value of N! / M!, or N! x M!.

```
void
gal_factorial
(
  gal_facexp_t *facexp,
  int n,
  int m,
  int s,
  long double *f,
  gal_status_t *status
) ;
```

If the requested factorial is beyond the range of the lookup table then the error code

GAL_OUT_OF_RANGE is set. If the requested factorial is greater than 1754!, then the error code GAL_EXCESSIVE_FACTORIAL is set. The pointer FACEXP points to a lookup table allocated by gal_facexp_alloc. Parameters N and M must be greater than or equal to zero. On return when S equals 0, F contains N!, when S equals -1, f contains N! divided by M!, and when S equals +1, F contains the product of N! and M!. On compilers that define long double to be the same precision as double the maximum factorial or result that can be returned is 170!, otherwise it is 1754!.

References:

Calculation of Factorials, M. L. Charnow and Jesse L. Maury, Jr., NASA TM X-55733 GSFC X-542-66-460, September 1966

gal_hrmesp [0.6]

Evaluate, at a specified point, an Hermite interpolating polynomial for a specified set of coordinate pairs whose abscissas are equally spaced.

```
void
gal_hrmesp
(
  int n,
  double first,
  double step,
  double *yvals,
  double x,
  double *work,
  double *f,
  double *df,
  gal_status_t *status
) ;
```

Given:

N	Number of points defining the polynomial.
FIRST	First abscissa value.
STEP	Step size.
*YVALS	Ordinate and derivative values.
X	Point at which to interpolate the polynomial.
*WORK	Work space array.

Returned:

*F	Interpolated function value at X.
*DF	Interpolated function's derivative at X.

*STATUS	Pointer to status structure

gal_hrmint [0.6]

Evaluate, at a specified point, an Hermite interpolating polynomial for a specified set of coordinate pairs whose abscissas are unequally spaced.

```
void
gal_hrmint
(
  int n,
  double *xvals,
  double *yvals,
  double x,
  double *work,
  double *f,
  double *df,
  gal_status_t *status
) ;
```

Given:

N	Number of points defining the polynomial.
*XVALS	Abscissa values.
*YVALS	Ordinate and derivative values.
X	Point at which to interpolate the polynomial.
*WORK	Work space array.

Returned:

*F	Interpolated function value at X.
*DF	Interpolated function's derivative at X.
*STATUS	Pointer to status structure

gal_ir [0.1]

Initialize an r-matrix to the identity matrix.

```
void
gal_ir
(
  double r[3][3]
) ;
```

On return R contains an identity matrix.

gal_lgresp [0.6]

Evaluate a Lagrange interpolating polynomial for a specified set of coordinate pairs whose first components are equally spaced, at a specified abcissisa value.

```
double
gal_lgresp
(
  int n,
  double first,
  double step,
  double *yvals,
  double *work,
  double x,
  gal_status_t *status
) ;
```

Given:

N	Number of points defining the polynomial.
FIRST	First abscissa value.
STEP	Step size.
*YVALS	Ordinate and derivative values.
X	Point at which to interpolate the polynomial.
*WORK	Work space array.

Returned:

*STATUS	Pointer to status structure
GAL_LGRESP	Result

gal_lgrint [0.6]

Evaluate a Lagrange interpolating polynomial for a specified set of coordinate pairs, at a specified abcissisa value.

```
double
gal_lgrint
(
  int n,
  double *xvals,
  double *yvals,
  double *work,
  double x,
  gal_status_t *status
) ;
```

Given:

N	Number of points defining the polynomial.
*XVALS	Abscissa values.
*YVALS	Ordinate values.
X	Point at which to interpolate the polynomial.
*WORK	Work space array.

Returned:

*STATUS	Pointer to status structure
GAL_LGRINT	Result

gal_lstled [0.6]

Given a number X and an array of non-decreasing numbers, find the index of the largest array element less than or equal to X.

```
int
gal_lstled
(
  double x,
  int n,
  double *array
) ;
```

Given:

X	value to search against
N	number of elements in array
*ARRAY	array to search

Returned:

GAL_LSTLED	Index of element found, -1 if not found

An array of double precision numbers is given. The array ARRAY[I] (0 <= I < N) forms a non-decreasing sequence of numbers. Given a real number X, there will be a last one of these numbers that is less than or equal to X. This routine finds the index LSTLED such that ARRAY[LSTLED] is that number. If X is not greater than ARRAY[0], LSTLED will be set to -1.

gal_lstltd [0.6]

Given a number X and an array of non-decreasing numbers, find the index of the largest array element less than X.

```
int
gal_lstltd
(
  double x,
  int n,
  double *array
) ;
```

Given:

X	value to search against
N	number of elements in array
*ARRAY	array to search

Returned:

GAL_LSTLTD	Index of element found, -1 if not found

An array of double precision numbers is given. The array ARRAY[I] (0 <= I < N) forms a non-decreasing sequence of numbers. Given a real number X, there will be a last one of these numbers that is less than X. This routine finds the index LSTLTD such that ARRAY[LSTLTD] is that number. If X is not greater than ARRAY[0], LSTLTD will be set to -1.

gal_minv6 [0.6]

6 x 6 matrix inversion.

```
void
gal_minv6
(
  double m[6][6],
  double r[6][6],
  gal_status_t *status
) ;
```

On return R contains the inversion of matrix M.

gal_p2pv [0.1]

Extend a p-vector to a pv-vector by appending a zero velocity.

```
void
gal_p2pv
(
  double p[3],
  double pv[2][3]
) ;
```

On return PV[0][0-2] contains P[0-2], and PV[1][0-2] contains zero.

gal_p2s [0.1]

p-vector to spherical polar coordinates.

```
void
gal_p2s
(
  double p[3],
  double *theta,
  double *phi,
  double *r
) ;
```

On return THETA and PHI contain the longitude and latitude angles in radians respectively, and R contains the radial distance. If P is null, zero THETA, PHI and R are returned. At either pole, zero THETA is returned.

gal_pap [0.1]

Position-angle from two p-vectors.

```
double
gal_pap
(
  double a[3],
  double b[3]
) ;
```

Given A the direction of the reference point, and B the direction of the point whose position angle is required, the function returns the position angle of B with respect to A in radians. The result is the position angle, in radians, of direction B with respect to direction A. It is in the range -π to +π. The sense is such that if B is a small distance "north" of A the position angle is approximately zero, and if B is a small distance "east" of A the position angle is approximately + π/2. A and B need not be unit vectors. Zero is returned if the two directions are the same or if either vector is null. If A is at a pole, the result is ill-defined.

gal_pas [0.1]

Position-angle from spherical coordinates.

```
double
gal_pas
(
  double al,
  double ap,
  double bl,
  double bp
) ;
```

Given AL the longitude of point A (e.g. right ascension), AP the latitude of point A (e.g. declination), BL the longitude of point B, and BP the latitude of point B. All angles in radians. The result is the bearing (position angle), in radians, of point B with respect to point A. It is in the range -π to +π. The sense is such that if B is a small distance "east" of point A, the bearing is approximately + π/2. Zero is returned if the two points are coincident.

gal_pdp [0.1]

p-vector dot product.

```
double
gal_pdp
(
  double a[3],
  double b[3]
) ;
```

Returns the dot product of vectors A and B.

gal_pm [0.1]

Modulus of p-vector.

```
double
gal_pm
(
  double p[3]
) ;
```

Returns the modulus of the p-vector P.

gal_pmp [0.1]

p-vector subtraction.

```
void
gal_pmp
(
  double a[3],
  double b[3],
  double amb[3]
) ;
```

On return p-vector AMB contains p-vector A minus p-vector B.

gal_pn [0.1]

Convert a p-vector into modulus and unit vector.

```
void
gal_pn
(
  double p[3],
  double *r,
  double u[3]
) ;
```

On return the p-vector U contains the unit vector of p-vector P, and R contains the modulus of p-vector P. If P is null, the result is null. Otherwise the result is a unit vector.

gal_ppp [0.1]

p-vector addition.

```
void
gal_ppp
(
  double a[3],
  double b[3],
  double apb[3]
) ;
```

On return p-vector APB contains the sum of p-vectors A and B.

gal_ppsp [0.1]

p-vector plus scaled p-vector.

```
void
gal_ppsp
(
  double a[3],
  double s,
  double b[3],
  double apsb[3]
) ;
```

On return p-vector APSB contains the sum of p-vector A and the product of scalar S and p-vector B.

gal_pv2p [0.1]

Discard velocity component of a pv-vector.

```
void
gal_pv2p
(
  double pv[2][3],
  double p[3]
) ;
```

On return the p-vector P contains a copy of the position vector portion of pv-vector PV.

gal_pv2s [0.1]

Convert position/velocity from Cartesian to spherical coordinates.

```
void
gal_pv2s
(
  double pv[2][3],
  double *theta,
  double *phi,
  double *r,
  double *td,
  double *pd,
  double *rd
) ;
```

On return THETA contains the longitude angle, PHI contains the latitude angle, R contains the radial distance, TD contains the rate of change of THETA, PD contains the rate of change of PHI, and RD contains the rate of change of R. All angles are in radians. If the position part of PV is null, THETA, PHI, TD and PD are indeterminate. This is handled by extrapolating the position through unit time by using the velocity part of PV. This moves the origin without changing the direction of the velocity component. If the position and velocity components of PV are both null, zeroes are returned for all six results. If the position is a pole, THETA, TD and PD are indeterminate. In such cases zeroes are returned for THETA, TD and PD.

gal_pvdpv [0.1]

Dot product of two pv-vectors.

```
void
gal_pvdpv
(
  double a[2][3],
  double b[2][3],
  double adb[2]
) ;
```

On return pv-vector ADB contains the dot product of pv-vectors A and B. If the position and velocity components of the two pv-vectors are (AP, AV) and (BP, BV), the result, A . B, is the pair of numbers (AP . BP, AP . BV + AV . BP). The two numbers are the dot-product of the two p-vectors and its derivative.

gal_pvm [0.1]

Modulus of pv-vector.

```
void
gal_pvm
(
  double pv[2][3],
  double *r,
  double *s
) ;
```

On return R and S contain the modulus of the position and velocity components of the pv-vector PV respectively.

gal_pvmpv [0.1]

Subtract one pv-vector from another.

```
void
gal_pvmpv
(
  double a[2][3],
  double b[2][3],
  double amb[2][3]
) ;
```

On return the pv-vector AMB contains pv-vector A minus pv-vector B.

gal_pvppv [0.1]

Add one pv-vector to another.

```
void
gal_pvppv
(
  double a[2][3],
  double b[2][3],
  double apb[2][3]
) ;
```

On return the pv-vector APB contains the sum of pv-vectors A and B.

gal_pvu [0.1]

Update a pv-vector.

```
void
gal_pvu
(
  double dt,
  double pv[2][3],
  double upv[2][3]
) ;
```

"Update" means refer the position component of the vector to a new epoch DT time units from the existing epoch. The time units of DT must match those of the velocity. The velocity component is unchanged.

gal_pvup [0.1]

Update a pv-vector, discarding the velocity component.

```
void
gal_pvup
```

```
(
  double dt,
  double pv[2][3],
  double p[3]
) ;
```

"Update" means refer the position component of the vector to a new epoch DT time units from the existing epoch. The time units of DT must match those of the velocity.

gal_pvxpv [0.1]

Cross product of two pv-vectors.

```
void
gal_pvxpv
(
  double a[2][3],
  double b[2][3],
  double axb[2][3]
) ;
```

On return the pv-vector AXB contains the cross product of pv-vectors A and B. If the position and velocity components of the two pv-vectors are (AP, AV) and (BP, BV), the result, A x B, is the pair of vectors (AP x BP, AP x BV + AV x BP). The two vectors are the cross-product of the two p-vectors and its derivative.

gal_pxp [0.1]

p-vector cross product.

```
void
gal_pxp
(
  double a[3],
  double b[3],
  double axb[3]
) ;
```

On return the p-vector AXB contains the cross product of p-vectors A and B.

gal_rint [0.6]

Round double to nearest integer value.

```
double
gal_rint
```

```
(
  double d
) ;
```

This routine returns the double D rounded to the nearest integer value .

gal_rkf [0.3]

This routine integrates an ordinary deferential equation using the Runge-Kutte-Fehlberg method.

```
void gal_rkf
(
  double ystart[],
  int nvar,
  double x1,
  double x2,
  double eps,
  double h1,
  double hmin,
  void ( *derivs ) ( double, double [], double [], int *,
gal_status_t * ),
  void ( *rkfs ) ( double [], double [], int, double, double,
double [], double [], void ( * ) ( double, double [], double [],
int *, gal_status_t * ), int *, gal_status_t * ) ,
  int *derivsp,
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

YSTART	Starting y values
NVAR	Number of equations to integrate
X1	Starting X value
X2	Ending X value
EPS	Accuracy
H1	First guess step-size
HMIN	Minimum step-size
DERIVS	User defined function for calculating the right hand side derivatives
RKFS	Required Runge-Kutte-Fehlberg stepper routine
DERIVSP	Pointer to parameters structure for DERIVS routine

On return YSTART contains the ending Y values. If an error occurs then the applicable error code is set in STATUS.

References:

NASA Technical Report TR R-352, Some Experimental Results Concerning The Error Propagation in Runge-Kutte type integration formulas by Erwin Fehlberg October 1970

gal_rkfcks45 [0.3]

This routine takes a Runge-Kutte-Fehlberg-Cash-Karp 4(5) step

```
void
gal_rkfcks45
(
  double y[],
  double dydx[],
  int n,
  double x,
  double h,
  double yout[],
  double yerr[],
  void ( *derivs ) ( double, double [], double [], int *,
gal_status_t * ),
  int *derivsp,
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

Y	Dependent variable vector
DYDX	Derivative of dependent variable vector
N	Number of equations to integrate
X	Independent variable value
H	Step size
DERIVS	User defined function for calculating the right hand side derivatives
DERIVSP	Pointer to parameters structure for DERIVS routine

On return the variables are set as follows:

YOUT	Ending Y values
YERR	Errors

If an error occurs then the applicable error code is set in STATUS.

The parameters (but not the code) (Cash-Karp version) are from “Numerical Recipes” for RKF45. These values are taken from the c code and not from the table on page 717 which has different values. The Cash-Karp values seem to make the routine a bit faster compared to the Fehlberg values.

References:

Numerical Recipes in C The Art of Scientific Computing Second Edition by William H. Press, Saul A. Teukolsky, William T. Vettering & Brian P. Flannery Pages 710 - 722

gal_rkfqs [0.3]

This routine takes one "quality-controlled" Runge-Kutte-Fehlberg step

```
void
gal_rkfqs
(
  double y[],
  double dydx[],
  int n,
  double *x,
  double htry,
  double eps,
  double yscal[],
  double *hdid,
  double *hnext,
  void ( *derivs ) ( double, double [], double [], int *,
gal_status_t * ),
  void ( *rkfs ) ( double [], double [], int, double, double,
double [], double [], void ( * ) ( double, double [], double [],
int *, gal_status_t * ), int *, gal_status_t * ),
  int *derivsp,
  gal_status_t *status
) ;
```

On entry the variables are set as follows:

Y	Dependent variable vector
N	Number of equations to integrate
X	Independent variable value
HTRY	Step size to attempt
EPS	Accuracy
DERIVS	User defined function for calculating the right hand side derivatives
RKFS	Required Runge-Kutte-Fehlberg stepper routine
DERIVSP	Pointer to parameters structure for derivs routine

On return the variables are set as follows:

DYDX	Derivative of dependent variable vector

YSCAL	Used for error scaling
HDID	Step size accomplished
HNEXT	Estimated next step size

If an error occurs then the applicable error code is set in STATUS.

References:

NASA Technical Report TR R-352, Some Experimental Results Concerning The Error Propagation in Runge-Kutte type integration formulas by Erwin Fehlberg, October 1970

gal_rkfs45 [0.3]

This routine takes a Runge-Kutte-Fehlberg 4(5) step

```
void
gal_rkfs45
(
  double y[],
  double dydx[],
  int n,
  double x,
  double h,
  double yout[],
  double yerr[],
  void ( *derivs ) ( double, double [], double [], int *,
gal_status_t * ),
  int *derivsp,
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

Y	Dependent variable vector
DYDX	Derivative of dependent variable vector
N	Number of equations to integrate
X	Independent variable value
H	Step size
DERIVS	User defined function for calculating the right hand side derivatives
DERIVSP	Pointer to parameters structure for DERIVS routine

On return the variables are set as follows:

YOUT	Ending Y values
YERR	Errors

If an error occurs then the applicable error code is set in STATUS.

References:

NASA Technical Report TR R-352, Some Experimental Results Concerning The Error Propagation in Runge-Kutte type integration formulas by Erwin Fehlberg, October 1970

gal_rkfs56 [0.3]

This routine takes a Runge-Kutte-Fehlberg 5(6) step

```
void
gal_rkfs56
(
  double y[],
  double dydx[],
  int n,
  double x,
  double h,
  double yout[],
  double yerr[],
  void ( *derivs ) ( double, double [], double [], int *,
gal_status_t * ),
  int *derivsp,
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

Y	Dependent variable vector
DYDX	Derivative of dependent variable vector
N	Number of equations to integrate
X	Independent variable value
H	Step size
DERIVS	User defined function for calculating the right hand side derivatives
DERIVSP	Pointer to parameters structure for DERIVS routine

On return the variables are set as follows:

YOUT	Ending Y values
YERR	Errors

If an error occurs then the applicable error code is set in STATUS.

References:

NASA Technical Report TR R-352, Some Experimental Results Concerning The Error Propagation in Runge-Kutte type integration formulas by Erwin Fehlberg, October 1970

gal_rkfs67 [0.3]

This routine takes a Runge-Kutte-Fehlberg 6(7) step

```
void
gal_rkfs67
(
  double y[],
  double dydx[],
  int n,
  double x,
  double h,
  double yout[],
  double yerr[],
  void ( *derivs ) ( double, double [], double [], int *,
gal_status_t * ),
  int *derivsp,
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

Y	Dependent variable vector
DYDX	Derivative of dependent variable vector
N	Number of equations to integrate
X	Independent variable value
H	Step size
DERIVS	User defined function for calculating the right hand side derivatives
DERIVSP	Pointer to parameters structure for DERIVS routine

On return the variables are set as follows:

YOUT	Ending Y values
YERR	Errors

If an error occurs then the applicable error code is set in STATUS.

References:

NASA Technical Report TR R-352, Some Experimental Results Concerning The Error Propagation in Runge-Kutte type integration formulas by Erwin Fehlberg, October 1970

gal_rkfs78 [0.3]

This routine takes a Runge-Kutte-Fehlberg 7(8) step

```
void
gal_rkfs78
(
  double y[],
  double dydx[],
  int n,
  double x,
  double h,
  double yout[],
  double yerr[],
  void ( *derivs ) ( double, double [], double [], int *,
gal_status_t * ),
  int *derivsp,
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

Y	Dependent variable vector
DYDX	Derivative of dependent variable vector
N	Number of equations to integrate
X	Independent variable value
H	Step size
DERIVS	User defined function for calculating the right hand side derivatives
DERIVSP	Pointer to parameters structure for DERIVS routine

On return the variables are set as follows:

YOUT	Ending Y values
YERR	Errors

If an error occurs then the applicable error code is set in STATUS.

References:

NASA Technical Report TR R-352, Some Experimental Results Concerning The Error Propagation in Runge-Kutte type integration formulas by Erwin Fehlberg, October 1970

gal_rm2v [0.1]

Express an r-matrix as an r-vector.

```
void
gal_rm2v
(
  double r[3][3],
  double w[3]
) ;
```

On return W contains the rotation vector. A rotation matrix describes a rotation through some angle about some arbitrary axis called the Euler axis. The "rotation vector" returned by this routine has the same direction as the Euler axis, and its magnitude is the angle in radians. The magnitude and direction can be separated by means of the routine gal_pn. If R is null, so is the result. If R is not a rotation matrix the result is undefined. R must be proper (i.e. have a positive determinant) and real orthogonal (inverse equals transpose). The reference frame rotates clockwise as seen looking along the rotation vector from the origin.

gal_rv2m [0.1]

Form the r-matrix corresponding to a given r-vector.

```
void
gal_rv2m
(
  double w[3],
  double r[3][3]
) ;
```

On return the r-matrix R contains the rotation matrix. A rotation matrix describes a rotation through some angle about some arbitrary axis called the Euler axis. The rotation vector supplied to this routine has the same direction as the Euler axis, and its magnitude is the angle in radians. If W is null, the unit matrix is returned. The reference frame rotates clockwise as seen looking along the rotation vector from the origin.

gal_rx [0.1]

Rotate an r-matrix about the X-axis.

```
void
gal_rx
(
  double phi,
  double r[3][3]
) ;
```

On return the r-matrix R has been rotated by the angle PHI about the X axis. The angle PHI is in radians. Sign convention: the matrix can be used to rotate the reference frame of a vector. Calling this routine with positive phi incorporates in the matrix an additional rotation, about the X-axis, anticlockwise as seen looking towards the origin from positive X.

gal_rxp [0.1]

Multiply a p-vector by an r-matrix.

```
void
gal_rxp
(
  double r[3][3],
  double p[3],
  double rp[3]
) ;
```

On return the p-vector RP contains the product of the r-matrix R and the p-vector P.

gal_rxpv [0.1]

Multiply a pv-vector by an r-matrix.

```
void
gal_rxpv
(
  double r[3][3],
  double pv[2][3],
  double rpv[2][3]
) ;
```

On return the pv-vector RPV contains the product of the r-matrix R and the pv-vector PV.

gal_rxr [0.1]

Multiply two r-matrices.

```
void
gal_rxr
(
  double a[3][3],
  double b[3][3],
  double atb[3][3]
) ;
```

On return the r-matrix ATB contains the product of the r-matrix A and the r-matrix B.

gal_ry [0.1]

Rotate an r-matrix about the y-axis.

```
void
gal_ry
(
  double theta,
  double r[3][3]
) ;
```

On return the r-matrix R has been rotated by the angle THETA about the y-axis. The angle THETA is in radians. Sign convention: the matrix can be used to rotate the reference frame of a vector. Calling this routine with positive theta incorporates in the matrix an additional rotation, about the y-axis, anticlockwise as seen looking towards the origin from positive y.

gal_rz [0.1]

Rotate an r-matrix about the z-axis.

```
void
gal_rz
(
  double psi,
  double r[3][3]
) ;
```

On return the r-matrix R has been rotated by the angle PSI about the z-axis. The angle PSI is in radians. Sign convention: the matrix can be used to rotate the reference frame of a vector. Calling this routine with positive psi incorporates in the matrix an additional rotation, about the z-axis, anticlockwise as seen looking towards the origin from positive z.

gal_s2c [0.1]

Convert spherical coordinates to Cartesian.

```
void
gal_s2c
(
  double theta,
  double phi,
  double c[3]
) ;
```

On return the p-vector C contains the direction cosines, given THETA the longitude angle, and PHI the latitude angle. All angles in radians.

gal_s2p [0.1]

Convert spherical polar coordinates to p-vector.

```
void
gal_s2p
(
  double theta,
  double phi,
  double r,
  double p[3]
) ;
```

On return the p-vector P contains the polar coordinates given THETA the longitude angle, PHI the latitude angle, and R the radial distance. The angles are both in radians.

gal_s2pv [0.1]

Convert position/velocity from spherical to Cartesian coordinates.

```
void
gal_s2pv
(
  double theta,
  double phi,
  double r,
  double td,
  double pd,
  double rd,
  double pv[2][3]
) ;
```

On return the pv-vector PV contains the position and velocity in Cartesian coordinates given THETA the longitude angle, PHI the latitude angle, R the radial distance, TD the rate of change of THETA, PD the rate of change of PHI, and RD the rate of change of R.

gal_s2xpv [0.1]

Multiply a pv-vector by two scalars.

```
void
gal_s2xpv
(
  double s1,
  double s2,
  double pv[2][3],
  double spv[2][3]
) ;
```

On return the position component of pv-vector SPV contains the product of the scalar S1 and the position component of pv-vector PV, and the velocity component of pv-vector SPV contains the product of scalar S2 and the velocity component of pv-vector PV.

gal_sepp [0.1]

Angular separation between two p-vectors.

```
double
gal_sepp
(
  double a[3],
  double b[3]
) ;
```

The routine returns the angular separation between the p-vectors A and B in radians (always positive). If either vector is null, a zero result is returned. The angular separation is most simply formulated in terms of scalar product. However, this gives poor accuracy for angles near zero and π. The algorithm uses both cross product and dot product, to deliver full accuracy whatever the size of the angle.

gal_seps [0.1]

Angular separation between two sets of spherical coordinates.

```
double
gal_seps
(
  double al,
  double ap,
  double bl,
  double bp
) ;
```

Returns the angular separation between first longitude and latitude (AL, AP) and the second longitude and latitude (BL, BP). All angles in radians.

gal_sxp [0.1]

Multiply a p-vector by a scalar.

```
void
gal_sxp
(
  double s,
  double p[3],
  double sp[3]
) ;
```

On return the p-vector SP contains the product of the scalar S and the p-vector P.

gal_sxpv [0.1]

Multiply a pv-vector by a scalar.

```
void
gal_sxpv
(
  double s,
  double pv[2][3],
  double spv[2][3]
) ;
```

On return the pv-vector SPV contains the product of the scalar S and the pv-vector PV.

gal_tr [0.1]

Transpose an r-matrix.

```
void
gal_tr
(
  double r[3][3],
  double rt[3][3]
) ;
```

On return the r-matrix RT contains the transpose of the r-matrix R.

gal_trxp [0.1]

Multiply a p-vector by the transpose of an r-matrix.

```
void
gal_trxp
(
  double r[3][3],
  double p[3],
  double trp[3]
) ;
```

On return the p-vector TRP contains the product of the transpose of the r-matrix R and the p-vector P.

gal_trxpv [0.1]

Multiply a pv-vector by the transpose of an r-matrix.

```
void
gal_trxpv
(
  double r[3][3],
  double pv[2][3],
  double trpv[2][3]
) ;
```

On return the pv-vector TRPV contains the product of the transpose of the r-matrix R and the pv-vector PV.

gal_vlcom [0.6]

Compute a vector linear combination of two p-vectors.

```
void
gal_vlcom
(
  double a,
  double v1[3],
  double b,
  double v2[3],
  double sum[3]
) ;
```

Given:

A	Coefficient of V1
V1	Vector in 3-space
B	Coefficient of V2
V2	Vector in 3-space

Returned:

SUM	Linear Vector Combination A*V1 + B*V2

gal_vlcom3 [0.6]

Compute a vector linear combination of three p-vectors.

```
void
gal_vlcom3
(
  double a,
  double v1[3],
  double b,
  double v2[3],
  double c,
  double v3[3],
  double sum[3]
) ;
```

Given:

A	Coefficient of V1
V1	Vector in 3-space
B	Coefficient of V2
V2	Vector in 3-space
C	Coefficient of V3
V3	Vector in 3-space

Returned:

SUM	Linear Vector Combination A*V1 + B*V2 + C*V3

gal_vproj [0.6]

Find the projection of one p-vector onto another p-vector.

```
void
```

```
gal_vproj
(
  double a[3],
  double b[3],
  double p[3]
) ;
```

Given:

A	The vector to be projected.
B	The vector onto which A is to be projected.

Returned:

P	The projection of A onto B.

gal_vrotv [0.6]

Rotate a vector about a specified axis vector by a specified angle and return the rotated vector

```
void
gal_vrotv
(
  double v[3],
  double axis[3],
  double theta,
  double r[3]
) ;
```

Given:

V	Vector to be rotated.
AXIS	Axis of the rotation.
THETA	Angle of rotation (radians).

Returned:

R	Result of rotating V about AXIS by THETA.

gal_xposeg [0.6]

Transpose a matrix of arbitrary size (in place, the matrix need not be square).

```
void
```

```
gal_xposeg
(
  double *matrix,
  int nrow,
  int ncol,
  double *xposem
) ;
```

Given:

*MATRIX	Matrix to be transposed
NROW	number of rows
NCOL	number of columns

Returned:

*XPOSEM	Transposed matrix (can overwrite MATRIX)

gal_zp [0.1]

Zero a p-vector.

```
void
gal_zp
(
  double p[3]
) ;
```

On return the all elements of the p-vector P are set to zero.

gal_zpv [0.1]

Zero a pv-vector.

```
void
gal_zpv
(
  double pv[2][3]
) ;
```

On return all the elements of the pv-vector PV are set to zero.

gal_zr [0.1]

Initialize an r-matrix to the null matrix.

```
void
gal_zr
(
  double r[3][3]
) ;
```

On return all elements of the r-matrix R are set to zero.

Chapter 4 - File Manipulation

The routines detailed in this chapter are defined in the gal_pstrings.h header file. The gal_pstrings sub-library was originally simply a set string manipulation routines. However, starting with version 0.6 of GAL the scope has been extended to file manipulation. Due to a lack on consensus on a better name for the expanded sub-library, the pstrings name has been retained.

The pstrings sub-library provides the low level support for NAIF DAF files. Double Precision Array File, DAF, is an architecture for files that stores arrays of double precision data. The NAIF SPICE SPK, CK, and binary PCK files use the DAF architecture. DAF files may be stored in big or little endian format, the GAL routines will convert the endian format automatically as necessary to match the machine's native endian format.

The header file gal_daf_t.h defines the following types used by the DAF support routines:

```
/*
 * DAF summary structure definition
 */

struct sum_t
{
  char          *aname      ; /* Array name                              */
  int           istart      ; /* Start address ( index )                 */
  int           iend        ; /* End address   ( index )                 */
  int           count       ; /* Number of elements in array             */
  int           *icon       ; /* Integer summary constants               */
  double        *dcon       ; /* Double summary constants                */
  struct sum_t *next        ; /* Pointer to next summary record          */
  struct sum_t *previous    ; /* Pointer to previous summary record      */
} ;

typedef struct sum_t gal_dafsum_t ;

/*
 * DAF workspace structure definition
 */

typedef struct {
  FILE          *fp         ; /* File pointer                            */
  char          idw[9]      ; /* Identification word                     */
  char          *reserved   ; /* Pointer to reserved blocks ( maybe text ) */
  int           rsize       ; /* Size of reserved blocks in bytes        */
  int          nd          ; /* Number of doubles in summary              */
  int          ni          ; /* Number of integers in summary             */
  int          nc          ; /* Number of summary block in bytes          */
  int          ss          ; /* Length of summary block in words ( dbl )  */
  int          maxs        ; /* Maximum number of summaries per block     */
  char         inam[61]    ; /* Internal name of array file               */
  int          fsb         ; /* First summary block                       */
  int          lsb         ; /* Last summary bock                         */
  int          nfb         ; /* Next free block                           */
  int          swapbytes   ; /* 1 if endian format different from machine */
  gal_dafsum_t *fsum       ; /* Pointer to first summary structure        */
  gal_dafsum_t *lsum       ; /* Pointer to last summary structure         */
} gal_daf_t ;
```

gal_center [0.1]

Center string in field.

```
char *
gal_center
(
  char *s1,
  char *s2,
  int  l
) ;
```

This routine copies the source string S2 to S1, then centers the trimmed string in a field of length L. Returns a pointer to the start of the target string. The target string S1 must be at least the same length as the source string S2.

gal_dafclose [0.6]

Close an open NAIF DAF file.

```
void
gal_dafclose
(
  gal_daf_t *daf
) ;
```

Given:

*DAF	Pointer to DAF structure

The routine gal_dafclose must be used to close the DAF file and deallocate memory.

gal_dafopen [0.6]

Open a NAIF DAF file and create linked list lookup mechanism.

```
gal_daf_t *
gal_dafopen
(
  char *filename,
  gal_status_t *status
) ;
```

Given:

*FILENAME	Pointer to filename

Returned:

*STATUS	Pointer to status structure
*GAL_DAFOPEN	Pointer to DAF instance

The routine gal_dafclose must be used to close the DAF file and deallocate memory. This routine handles files on both big and little endian format. Any changes in format are transparent to the user. The data structure returned is defined and described in the gal_daf_t.h header file. This also describes the structure of the double linked list that can be used to locate the required array within the file. The structure of the reserved blocks is application specific and may contain text or data. For this reason no endian format change is done for these blocks.

References:

Reference for the low-level file structure known as Double Precision Array File, which is used to implement the high-level SPICE SPK and CK kernels. Last revised on 2008 JAN 17 by B. V. Semenov.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/daf.html

gal_dafread [0.6]

Read a block from an array in a NAIF DAF file.

```
int
gal_dafread
(
  gal_daf_t *daf,
  gal_dafsum_t *sum,
  int address,
  int num,
  double *block,
  gal_status_t *status
) ;
```

Given:

*DAF	Pointer to the DAF structure
*SUM	Pointer to summary for required array
ADDRESS	Index of first element required in array
NUM	Maximum number of elements to return

Returned:

*BLOCK	Array to hold required elements
*STATUS	Pointer to status structure
GAL_DAFREAD	Number of elements returned

The index number follows the C language convention, the first element of an array has an index value of zero. This routine handles files on both big and little endian format. Any changes in format are transparent to the user.

References:

Reference for the low-level file structure known as Double Precision Array File, which is used to implement the high-level SPICE SPK and CK kernels. Last revised on 2008 JAN 17 by B. V. Semenov.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/daf.html

gal_dafseek [0.6]

Seek for a particular word in a NAIF DAF file.

```
void
gal_dafseek
(
  gal_daf_t *daf,
  gal_dafsum_t *sum,
  int address,
  gal_status_t *status
) ;
```

Given:

*DAF	Pointer to the DAF structure
*SUM	Pointer to summary for required array
ADDRESS	Index of first element required in array

Returned:

*STATUS	Pointer to status structure

The index number follows the C language convention, the first element of an array has an index value of zero.

References:

Reference for the low-level file structure known as Double Precision Array File, which is used to implement the high-level SPICE SPK and CK kernels. Last revised on 2008 JAN 17 by B. V. Semenov.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/daf.html

gal_delete [0.1]

Delete characters from string.

```
char *
gal_delete
(
  char *s,
  int  n,
  int  l
) ;
```

This routine deletes a sequence of characters of length L, starting a position N. A pointer to the start of the target string S is returned.

gal_endian [0.6]

Determine the endian format of the machine.

```
int
gal_endian
(
) ;
```

If the machine is big-endian then the routine returns the constant GAL_ENDIAN_BIG, otherwise it returns the constant GAL_ENDIAN_LITTLE

gal_fscan [0.6]

This routine unpacks a data record using a FORTRAN style format definition.

```
void
gal_fscan
(
  char *form,
  char *source,
  gal_status_t *status,
  ...
);
```

On entry FORMAT contains the format definition, SOURCE the data string. If the format contains errors then the error code GAL_INVALID_FORMAT is set. On return the decoded data is stored in the variables pointed to by the parameters that follow the STATUS parameter. The FORMAT string follows the FORTRAN conventions (but not precisely or comprehensively:

In means integer in field of size n

Dn.d means double in field of size n with implied d decimal places.

En.d means double in exponent form in field of size n with implied d decimal places for mantissa. The routine will also process D exponents in the same way.

Gn.d if the string contains an 'E' exponent then means the same as En.d, otherwise it means that same as Dn.d

An means a character string of length n

Xn means skip n characters

C means character

Nothing else is allowed in the format definition. Where the data contains a decimal point in a different place than the format, then the data takes precedence over the format. Spaces in numeric values are assumed to mean zero. The destination variables must only be double, integer, or character array. Any character arrays must be large enough to hold the string and the string terminator character.

gal_insert [0.1]

Insert sub-string into string.

```
char *
gal_insert
(
  char *s1,
  char *s2,
  int  n
) ;
```

This routine inserts the sub-string S2 into string S1 at the specified character position N. Returns a pointer to the start of the target string.

gal_instr [0.1]

Find sub-string in string.

```
int
gal_instr
(
  char *s1,
  char *s2
) ;
```

This routine finds the first occurrence of the sub-string S2 in the string S1. It returns the position of the first character of the sub-string in S1. If the sub-string cannot be found then -1 is returned.

gal_justl [0.1]

Left justify string.

```
char *
gal_justl
(
  char *s1,
  char *s2,
  int  l
) ;
```

This routine copies the source string S2 to S1, then trims white-space from the beginning and end of the string. If the resultant string length is less than L then spaces are added on the right hand side to bring the string to length L. If the resultant string length is greater than L then the left-most L characters of the resultant string are returned in S1.

gal_justr [0.1]

Right justify string.

```
char *
gal_justr
(
  char *s1,
  char *s2,
  int  l
) ;
```

This routine copies the source string S2 to S1, then trims white-space from the beginning and end of the string. If the resultant string length is less than L then spaces are added on the left hand side to bring the string to length L. If the resultant string length is greater than L then the right-most L characters of the resultant string are returned in S1. The target string S1 must be at least the same length as the source.

gal_lcase [0.6]

Convert string to lower case.

```
char *
gal_lcase
(
  char *s1,
  char *s2
) ;
```

This routine copies the source string S2 to S1, then converts all upper case characters to lower case. The target string S1 must be at least the same length as the source.

gal_leftstr [0.1]

Copy sub-string from left of string.

```
char *
gal_leftstr
(
  char *s1,
  char *s2,
  int  l
) ;
```

This routine copies the left-most L characters from S2 to S1. If the length of S2 is less than or equal to L then S2 is copied to S1 unchanged. The target string S1 must be at least the same length as the source string S2.

gal_midstr [0.1]

Copy sub-string from middle of string.

```
char *
gal_midstr
(
  char *s1,
  char *s2,
  int  n,
  int  l
) ;
```

This routine copies the L characters from S2 to S1 starting at character position N in S2. If there are less than L characters remaining in the string S2 from position N onwards then all the available characters are returned.

gal_padl [0.1]

Pad string with spaces on left.

```
char *
gal_padl
(
  char *s1,
  char *s2,
  int  l
) ;
```

This routine copies a maximum of L characters from the right side of S2 to S1. If the length of S2 is less than L then the left hand side is padded with spaces up to the required length. The target string S1 must be at least the same length as the source string S2.

gal_padr [0.1]

Pad string with spaces on right.

```
char *
gal_padr
(
  char *s1,
  char *s2,
  int  l
) ;
```

This routine copies a maximum of L characters from the left side of S2 to S1. If the length of S2 is less than L then the right hand side is padded with spaces up to the required length L. The target string S1 must be at least the same length as the source string S2.

gal_replace [0.1]

Find and replace sub-string in string.

```
char *
gal_replace
(
  char *s1,
  char *s2,
  char *s3,
  char *s4
) ;
```

This routine copies the source string S2 to the target string S1. Then replaces all occurrences of the sub-string S3 in S1 with sub-string S4.

gal_rightstr [0.1]

Copy right sub-string from string.

```
char *
gal_rightstr
(
  char *s1,
  char *s2,
  int  l
) ;
```

This routine copies the right-most L characters from S2 to S1. If the length of S2 is less than or equal to L then S2 is copied to S1 unchanged. The target string S1 must be at least the same length as the source string S2.

gal_sbod [0.6]

Swap byte order of double variable.

```
void
gal_sbod
(
  double *d
) ;
```

This routine swaps the byte ordering of the double variable D, i.e. change from big endian to little endian format and vice versa.

gal_sboda [0.6]

Swap byte order of an array of double variables.

```
void
gal_sboda
(
  double *d,
  int n
) ;
```

This routine swaps the byte ordering of a double variable array D, i.e. change from big endian to little endian format and vice versa. On entry N contains the number of elements in the array

gal_sboi [0.6]

Swap byte order of integer variable.

```
void
gal_sboi
(
  int *i
) ;
```

This routine swaps the byte ordering of the integer variable I, i.e. change from big endian to little endian format and vice versa.

gal_sboia [0.6]

Swap byte order of an array of integer variables.

```
void
gal_sboia
(
  int *i,
  int n
) ;
```

This routine swaps the byte ordering of a double variable array I, i.e. change from big endian to little endian format and vice versa. On entry N contains the number of elements in the array

gal_sgfcon [0.6]

Given the descriptor for a generic segment in a DAF file associated with DAF, fetch from the constants partition of the segment the double precision numbers from FIRST to LAST.

```
void
```

```
gal_sgfcon
(
  gal_daf_t *daf,
  gal_dafsum_t *sum,
  int first,
  int last,
  double *values,
  gal_status_t *status
) ;
```

Given:

*DAF	Pointer to open DAF work space
*SUM	Pointer to selected DAF segment summary
FIRST	Index of the first value to fetch from the constants section of the generic segment
LAST	Index of the last value to fetch from the constants section of the generic segment

Returned:

*VALUES	Array of constant values obtained from the constants section of the generic segment
*STATUS	Pointer to status structure

References:

The NAIF Web site providing SPICE data for various missions, additional SPICE utilities and documentation can be found here:

http://naif.jpl.nasa.gov

gal_sgfpkt [0.6]

Given the descriptor for a generic segment in a DAF file associated with HANDLE, fetch the data packets indexed from FIRST to LAST from the packet partition of the generic segment.

```
void
gal_sgfpkt
(
  gal_daf_t *daf,
  gal_dafsum_t *sum,
  int first,
  int last,
  double *values,
  int *ends,
  gal_status_t *status
) ;
```

Given:

*DAF	Pointer to open DAF work space
*SUM	Pointer to selected DAF segment summar
FIRST	The index of the first data packet to fetch
LAST	The index of the last data packet to fetch

Returned:

*VALUES	The data packets that have been fetched
*ENDS	An array of pointers to the ends of the packets
*STATUS	Pointer to status structure

References:

The NAIF Web site providing SPICE data for various missions, additional SPICE utilities and documentation can be found here:

http://naif.jpl.nasa.gov

gal_sgfrvi [0.6]

Given the handle of a DAF and the descriptor associated with a generic DAF segment in the file, find the reference value associated with the value X and it's index.

```
void
gal_sgfrvi
(
  gal_daf_t *daf,
  gal_dafsum_t *sum,
  double x,
  double *value,
  int *indx,
  gal_status_t *status
) ;
```

Given:

*DAF	Pointer to open DAF work space
*SUM	Pointer to selected DAF segment summary
X	Value for which the associated reference value and reference index

is requested.

Returned:

VALUE — Reference value associated with the input value X

INDX — Index of VALUE within the set of reference values for the generic segment. This value may be used to obtain a particular packet of data from the generic segment.

*STATUS — Pointer to status structure. If the value is not found then the error code GAL_NOT_FOUND is set.

References:

The NAIF Web site providing SPICE data for various missions, additional SPICE utilities and documentation can be found here:

http://naif.jpl.nasa.gov

gal_sgmeta [0.6]

Obtain the value of a specified generic segment meta data item.

```
void
gal_sgmeta
(
  gal_daf_t *daf,
  gal_dafsum_t *sum,
  int mnemon,
  int *value,
  gal_status_t *status
) ;
```

Given:

*DAF — Pointer to open DAF work space

*SUM — Pointer to selected DAF segment summary

MNEMON — Mnemonic used to represent the desired piece of meta data.

Returned:

VALUE — Value of the meta data item associated with the mnemonic MNEMON that is in the generic segment specified by DAF and

SUM.

*STATUS Pointer to status structure

References:

The NAIF Web site providing SPICE data for various missions, additional SPICE utilities and documentation can be found here:

http://naif.jpl.nasa.gov

gal_sisd [0.6]

Check that string could represent a double value.

```
char *
gal_sisd
(
  char *source,
  gal_status_t *status
) ;
```

If the string does not contain a valid representation of a double value then the error code GAL_INVALID_DOUBLE is set. On return all white space is trimmed from both ends of the source string. On return any 'D', 'd', or 'E' characters in the source string are replaced with 'e'. In the event of an internal error NULL is returned. This routine only checks for valid characters. It does not check for number components that are out of sequence, e.g. "6-7e89+e99" would pass the test.

gal_sisi [0.6]

Check that string could represent an integer value.

```
char *
gal_sisi
(
  char *source,
  gal_status_t *status
) ;
```

If the string does not contain a valid representation of a integer value then the error code GAL_INVALID_INTEGER is set. On return all white space is trimmed from both ends of the source string. In the event of an internal error NULL is returned.

gal_sseek [0.6]

This routine seeks for the specified string in a text file then skips the specified number of lines.

```
void
gal_sseek
(
  FILE *fp,
  char *token,
  int  skip,
  gal_status_t *status
) ;
```

Given:

*FP	Pointer to an open text file
*TOKEN	Token string to find
SKIP	Number of lines to skip

If the specified token string is not found in the file then the error code GAL_NOT_FOUND is set. If the file pointer does not point to an open file then the error code GAL_FILE_NOT_OPEN is set. If the end of file is reached before all the requested number of lines are skipped then the error code GAL_UNEXPECTED_EOF is set. The routine starts searching from the current file position. The maximum line length is 2048.

gal_strn [0.1]

Fill string with character.

```
char *
gal_strn
(
  char *s,
  char c,
  int  l
) ;
```

This routine fills the target string with L characters of value C.

gal_trim [0.1]

Trim white-space from left and right of string.

```
char *
gal_trim
(
  char *s1,
  char *s2
) ;
```

This routine copies S2 to S1, then deletes any leading or trailing white-space characters at the beginning or end of S1. The target string S1 must be at least the same length as the source string S2.

gal_triml [0.1]

Trim white-space from left side of string.

```
char *
gal_triml
(
  char *s1,
  char *s2
) ;
```

This routine copies S2 to S1, then deletes any leading white-space characters at the beginning of S1. The target string S1 must be at least the same length as the source string S2.

gal_trimr [0.1]

Trim white-space from right of string.

```
char *
gal_trimr
(
  char *s1,
  char *s2
) ;
```

This routine copies S2 to S1, then deletes any trailing white-space characters at the end of S1. The target string S1 must be at least the same length as the source string S2.

gal_ucase [0.1]

Force string to upper-case.

```
char *
gal_ucase
(
  char *s1,
  char *s2
)  ;
```

This routine copies S2 to S1, then forces all lower case characters in S1 to upper case. The target string S1 must be at least the same length as the source string S2.

Chapter 5 - Test Framework

The routines detailed in this chapter are defined in the gal_test.h header file. The test framework is central to the GAL development methodology. These routines are the primitives upon which the test framework is constructed. The following is an extract from the test program for the pstrings sub-library, this illustrates the structure of a test program, note the inclusion of the “gal_test_common.h” header file.

```
#include "gal_test_common.h"
#include "gal_test.h"
#include "gal_instr_test.h"
#include "gal_insert_test.h"
#include "gal_delete_test.h"
#include "gal_leftstr_test.h"
#include "gal_rightstr_test.h"
#include "gal_midstr_test.h"
#include "gal_trimr_test.h"
#include "gal_triml_test.h"
#include "gal_trim_test.h"
#include "gal_strn_test.h"
#include "gal_padr_test.h"
#include "gal_padl_test.h"
#include "gal_justr_test.h"
#include "gal_justl_test.h"
#include "gal_center_test.h"
#include "gal_replace_test.h"
#include "gal_ucase_test.h"
#include "gal_sisd_test.h"
#include "gal_sisi_test.h"
#include "gal_fscan_test.h"
#include "gal_dafopen_test.h"

int
main
(
)

{

/*
 * - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 */
```

```
  gal_test_start ( "P Strings", 0 ) ;

  gal_sisd_test      () ;
  gal_sisi_test      () ;
  gal_instr_test     () ;
  gal_insert_test    () ;
  gal_delete_test    () ;
  gal_leftstr_test   () ;
  gal_rightstr_test  () ;
  gal_midstr_test    () ;
  gal_trimr_test     () ;
  gal_triml_test     () ;
  gal_trim_test      () ;
  gal_strn_test      () ;
  gal_padr_test      () ;
  gal_padl_test      () ;
  gal_justr_test     () ;
  gal_justl_test     () ;
  gal_center_test    () ;
  gal_replace_test   () ;
  gal_ucase_test     () ;
  gal_fscan_test     () ;
  gal_dafopen_test   () ;

  return gal_test_stop () ;

/*
 * Finished.
 */

}
```

gal_test_start [0.1]

Start test run.

```
void
gal_test_start
(
  char *libname,
  int verbose
) ;
```

This starts a test run and resets the various statistics. On entry LIBNAME contains the name of the sub-library under test. If VERBOSE is set to 1 then both success and failure messages are output by the test routines, and when set to 0 then only failure messages are output. On return the external variables GAL_TPASS, GAL_TFAIL, and GAL_TFUNC are set to zero. The external variable GAL_TVERB is set to the value of the parameter VERBOSE. The library name is copied to the external variable gal_tlibn to be used later by gal_test_stop.

gal_test_stop [0.1]

Stop test run and print statistics.

```
int
gal_test_stop
(
) ;
```

This stops a test run and prints the statistics. If no tests failed during the run then 0 is returned, otherwise 1 is returned.

gal_vcv [0.1]

Validate character result.

```
void
gal_vcv
(
  char cval,
  char cvalok,
  char *func,
  char *test
) ;
```

This routine validates a character result. On entry CVAL contains the value computed by the routine under test, CVALOK contains the correct value, FUNC contains the name of the routine under test, and TEST contains the name of the individual test. The external variables GAL_TPASS and GAL_TFAIL are incremented depending upon the outcome of the test. If the external variable GAL_TVERB is set to 1 then both test success and test failure messages are sent to the standard output. If set to 0 then only test failure messages are sent to the standard output.

gal_vdv [0.1]

Validate a double precision result.

```
void
gal_vdv
(
  double dval,
  double dvalok,
  double dtol,
  char   *func,
  char   *test
) ;
```

This routine validates a double precision result. On entry DVAL contains the value computed by the routine under test, DVALOK contains the correct value, DTOL the tolerance, FUNC contains the name of the routine under test, and TEST contains the name of the individual test. The external variables GAL_TPASS and GAL_TFAIL are incremented depending upon the outcome of the test. If the external variable GAL_TVERB is set to 1 then both test success and test failure messages are sent to the standard output. If set to 0 then only test failure messages are sent to the standard output.

gal_viv [0.1]

Validate an integer result.

```
void
gal_viv
(
  int  ival,
  int  ivalok,
  char *func,
  char *test
) ;
```

This routine validates an integer result. On entry IVAL contains the value computed by the routine under test, IVALOK contains the correct value, FUNC contains the name of the routine under test, and TEST contains the name of the individual test. The external variables GAL_TPASS and GAL_TFAIL are incremented depending upon the outcome of the test. If the external variable GAL_TVERB is set to 1 then both test success and test failure messages are sent to the standard output. If set to 0 then only test failure messages are sent to the standard output.

gal_vldv [0.1]

Validate long double precision result.

```
void
gal_vldv
(
  long double dval,
  long double dvalok,
  double dtol,
  char   *func,
  char   *test
) ;
```

This routine validates a long double precision result. On entry DVAL contains the value computed by the routine under test, DVALOK contains the correct value, DTOL contains the tolerance, FUNC contains the name of the routine under test, and TEST contains the name of the individual test. The external variables GAL_TPASS and GAL_TFAIL are incremented depending upon the outcome of the test. If the external variable GAL_TVERB is set to 1 then both test success and test failure messages are sent to the standard output. If set to 0 then only test failure messages are sent to the standard output.

gal_vsv [0.1]

Validate string result.

```
void
gal_vsv
(
  char *sval,
  char *svalok,
  char *func,
  char *test
) ;
```

This routine validates a string result. On entry SVAL points to the value computed by the routine under test, SVALOK contains the correct value, FUNC contains the name of the routine under test, and TEST contains the name of the individual test. The external variables GAL_TPASS and GAL_TFAIL are incremented depending upon the outcome of the test. If the external variable GAL_TVERB is set to 1 then both test success and test failure messages are sent to the standard output. If set to 0 then only test failure messages are sent to the standard output.

Chapter 6 - Date & Time

The GAL date and time routines are largely based upon the IAU's SOFA date & time support. In GAL releases prior to 0.6 additional routines were included for those areas not handled by SOFA, with the December 2010 release of SOFA many of those gaps have been filled. Additionally the December 2010 release of SOFA added an additional nuance to the SOFA date format has been added specifically to handle the leap second in UTC dates. GAL now implements the new SOFA date format, and new GAL interpretations of the new SOFA routines. Where there is an existing GAL routine functionally equivalent to a new SOFA routine, the existing GAL routine has been retained but the code updated, if necessary, to match the SOFA implementation. In these cases the notes have been updated to detail the SOFA routine that it is equivalent to. The IAU has published a very good cookbook that both explains the different time scales and shows how to use the date and time routines. As GAL parallels SOFA closely here, the cookbook is recommended reading for all users of GAL. It can be downloaded from here for free:

http://www.iausofa.org/cookbooks.html

The routines detailed in this chapter are defined in the gal_datetime.h header file. The gal_date_macros.h header file defines the following constants:

```
/*
 * ---------------------
 * Date & Time Constants
 * ---------------------
 */

#define GAL_DJM    (365250.0)             /* Days per Julian millennium                      */

#define GAL_DJC    (36525.0)              /* Days per Julian century                         */

#define GAL_DJY    (365.25)               /* Days per Julian year                            */

#define GAL_D2S    (86400.0)              /* Days to Seconds, same as SOFA's DAYSEC          */

#define GAL_D2M    (1440.0)               /* Days to Minutes                                 */

#define GAL_D2H    (24.0)                 /* Days to Hours                                   */

#define GAL_J2000  (2451545.0)            /* Reference epoch (J2000.0), Julian date          */

#define GAL_MJD0   (2400000.5)            /* Julian Date of Modified Julian Date zero        */
                                          /* Same as SOFA's DJM0                             */

#define GAL_MJ2000 (51544.5)              /* Reference epoch (J2000.0), Modified Julian Date */
                                          /* Same as SOFA's DJM00                            */

#define GAL_DTY    (365.242198781)        /* Length of tropical year B1900 (days)            */

#define GAL_DJM77  (43144.0)              /* 1977 Jan 1.0 as MJD                             */
```

```
#define GAL_TTMTAI (32.184)             /* TT minus TAI (s)                             */

#define GAL_ELG    (6.969290134e-10)    /* L_G = 1 - d(TT)/d(TCG)                       */
                                        /* This is SOFA's December 2010 value           */

#define GAL_ELB    (1.550519768e-8)     /* L_B = 1 - d(TDB)/d(TCB)                      */
                                        /* This is SOFA's December 2010 value           */

#define TDB0       (-6.55e-5)           /* TDB (s) at TAI 1977/1/1.0                    */
                                        /* This is SOFA's December 2010 value           */
```

gal_cal2jd [0.1]

Gregorian Calendar to Julian Date.

```
void
gal_cal2jd
(
  int iy,
  int im,
  int id,
  double *djm0,
  double *djm,
  gal_status_t *status
) ;
```

On entry IY contains the year, IM the month, and ID the day in the Gregorian calendar. On return DJM0 contains the Modified Julian Date zero-point of 2400000.5, and DJM contains the Modified Julian Date for 0 hours. If an invalid year and/or month are specified then the error code GAL_BAD_YEAR and/or GAL_MONTH are set and the routine sets DJM0 and DJM to zero. If an invalid day is specified then the error code GAL_BAD_DAY is set, however the date is computed and returned in DJM0 and DJM. The algorithm used is valid from -4800 March 1, but this implementation rejects dates before -4799 January 1. The Julian Date is returned in the standard SOFA two-piece format, which is designed to preserve time resolution. The Julian Date is available as a single number by adding DJM0 and DJM. In early eras the conversion is from the "Proleptic Gregorian Calendar"; no account is taken of the date(s) of adoption of the Gregorian Calendar, nor is the CE/BCE numbering convention observed.

References:

Explanatory Supplement to the Astronomical Almanac, P. Kenneth Seidelmann (ed.), University Science Books (1992), Section 12.92 (p604).

gal_dat [0.1]

Calculate difference between International Atomic Time (TAI) and Coordinated Universal Time (UTC): TAI - UTC

```
void
gal_dat
(
  int iy,
  int im,
  int id,
  double fd,
  double *deltat,
  gal_status_t *status
) ;
```

This routine for a given Coordinated Universal Time (UTC) date, calculates delta(AT) = TAI-UTC. On entry IY, IM, ID, and FD contain the UTC year, month, day, and fractional part of day. On return DELTAT contains International Atomic Time (TAI) minus Coordinated Universal Time (UTC) in seconds. UTC began at 1960 January 1.0 (JD 2436934.5) and it is improper to call the routine with an earlier date. If this is attempted, zero is returned and the error code GAL_OUTSIDE_DATE_RANGE is set. Because leap seconds cannot, in principle, be predicted in advance, a reliable check for dates beyond the valid range is impossible. To guard against gross errors, a year five or more after the release year of this routine (see parameter IYV) is considered dubious. In this case a warning code GAL_DUBIOUS_YEAR is set but the result is computed in the normal way. If an invalid year, month, day or fraction of a day are specified then any of the following error codes are set: GAL_BAD_YEAR, GAL_BAD_MONTH, GAL_BAD_DAY, or GAL_BAD_FRACTION. If the specified date is for a day which ends with a leap second, the UTC-TAI value returned is for the period leading up to the leap second. If the date is for a day which begins as a leap second ends, the UTC-TAI returned is for the period following the leap second. The day number must be in the normal calendar range, for example 1 through 30 for April. The "almanac" convention of allowing such dates as January 0 and December 32 is not supported in this routine, in order to avoid confusion near leap seconds. The fraction of day is used only for dates before the introduction of leap seconds, the first of which occurred at the end of 1971. It is tested for validity (zero to less than 1 is the valid range) even if not used; if invalid, zero is used and status GAL_BAD_FRACTION is returned. For many applications, setting FD to zero is acceptable; the resulting error is always less than 3 ms (and occurs only pre-1972).

References:

For dates from 1961 January 1 onwards, the expressions from the file:
ftp://maia.usno.navy.mil/ser7/tai-utc.dat are used.

The 5ms time step at 1961 January 1 is taken from the Explanatory Supplement to the Astronomical Almanac, P. Kenneth Seidelmann (ed.), University Science Books (1992), Section 2.58.1 (p87).

gal_datenorm [0.6]

Normalize a two piece SOFA date for maximum precision.

```
void
gal_datenorm
```

```
(
  double base,
  double *date1,
  double *date2
) ;
```

On entry BASE contains the pivot Julian Date, e.g. J2000 or MJD0. DATE1 and DATE2 contain a standard SOFA two piece Julian date. On return DATE1 and DATE2 have been normalized such that DATE1 contains the BASE value and DATE2 the remainder. The normalization process ensures the maximum retention of precision throughout.

gal_days2cal [0.1]

Convert the day of the year, and year, to Gregorian year, month, day, and fraction of a day.

```
void
gal_days2cal
(
  int year,
  double days,
  int *iy,
  int *im,
  int *id,
  double *fd,
  gal_status_t *status
) ;
```

On entry YEAR contains the year number between 1900 and 2100, and DAYS contains the day count including fraction of day. On return IY, IM, ID, and FD contain the Gregorian year, month, day, and fractional part of day respectively. In early eras the conversion is from the "Proleptic Gregorian Calendar"; no account is taken of the date(s) of adoption of the Gregorian Calendar, nor is the CE/BCE numbering convention observed. January 1 is day number 1.

gal_days2jd [0.6]

Convert the day of the year, and year, to a Julian date.

```
void
gal_days2jd
(
  int year,
  double days,
  double *djm0,
  double *djm,
  gal_status_t *status
) ;
```

On entry YEAR contains the year number between 1900 and 2100, and DAYS contains the day count including fraction of day. On return DJM0 and DJM contain the Julian date in standard two-piece SOFA format. January 1 is day number 1.

gal_dtdb [0.1]

An approximation to TDB-TT, the difference between Barycentric Dynamical Time and Terrestrial Time, for an observer on the Earth.

```
double
gal_dtdb
(
  double date1,
  double date2,
  double ut,
  double elong,
  double u,
  double v
) ;
```

On entry DATE1 and DATE2 contain the Barycentric Dynamical Time (TDB) in standard SOFA two-piece format, UT contains Universal Time (UT1) in fraction of one day, ELONG contains longitude (east positive in radians), U contains the distance from Earth spin (kilometers), and V contains distance north of equatorial plane (kilometers). The function returns TDB-TT in seconds. Although the date is, formally, Barycentric Dynamical Time (TDB), the Terrestrial Time (TT) can be used with no practical effect on the accuracy of the prediction.

References:

Fairhead, L., & Bretagnon, P., Astronomy & Astrophysics, 229, 240-247 (1990).

IAU 2006 Resolution 3. McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

Moyer, T.D., Celestial Mechanics, 23, 33 (1981).

Murray, C.A., Vectorial Astrometry, Adam Hilger (1983).

Seidelmann, P.K. et al., Explanatory Supplement to the Astronomical Almanac, Chapter 2, University Science Books (1992).

Simon, J.L., Bretagnon, P., Chapront, J., Chapront-Touze, M., Francou, G. & Laskar, J., Astronomy & Astrophysics, 282, 663-683 (1994).

gal_epb [0.1]

Julian Date to Besselian Epoch.

```
double
gal_epb
(
  double dj1,
  double dj2
) ;
```

On entry DJ1 and DJ2 contain the Julian Date, the Besselian Epoch is returned. The Julian Date is supplied in standard SOFA two-piece format.

References:

Lieske, J.H., 1979. Astronomy & Astrophysics,73,282.

gal_epb2jd [0.1]

Besselian Epoch to Julian Date.

```
void
gal_epb2jd
(
  double epb,
  double *djm0,
  double *djm
) ;
```

On entry EPB contains the date in the Besselian Epoch (e.g. 1957.3), on return DJM0 contains the Modified Julian Date zero-point of 2400000.5, and DJM contains the date as a Modified Julian Date in standard SOFA two-piece format.

References:

Lieske, J.H., 1979. Astronomy & Astrophysics,73,282.

gal_epj [0.1]

Julian Date to Julian Epoch.

```
double
gal_epj
```

```
(
  double dj1,
  double dj2
) ;
```

This routine returns the Julian epoch for the given Julian Date. On entry DJ1 and DJ2 contain the Julian Date in standard SOFA two-piece format.

References:

Lieske, J.H., 1979. Astronomy & Astrophysics,73,282.

gal_epj2jd [0.1]

Julian Epoch to Julian Date.

```
void
gal_epj2jd
(
  double epj,
  double *djm0,
  double *djm
) ;
```

On entry EPJ contains the Julian Epoch (e.g. 1996.8). On return DJM0 contains the Modified Julian Date zero-point of 2400000.5, and DJM contains the Modified Julian Date.

References:

Lieske, J.H., 1979. Astronomy & Astrophysics,73,282.

gal_fdate [0.6]

Convert a date to a string in specified format.

```
char *
gal_fdate
(
  char *format,
  double date1,
  double date2,
  char *result,
  gal_status_t *status
) ;
```

On entry FORMAT contains the format string, DATE1 and DATE2 contain a Julian date in standard two-piece SOFA format. On return RESULT contains the date in string format. If an error occurs then the applicable error code is set and the function returns NULL. If successful then the function returns a pointer to RESULT.

The format string contains a list of tokens which will converted into the date elements in a similar way as printf. The tokens can be in any order, and any non-token characters will be returned unchanged in the result string. The valid tokens are:

%yyyy or %YYYY	year number e.g. 1972
%yy or %YY	last two digits of year e.g. 05
%mmmm	month full English name in lower case e.g. january
%MMMM	month full English name in upper case e.g. JANUARY
%Mmmm	month full English name with leading upper case e.g. January
%mmm	short form English month name in lower cas e.g. jan
%MMM	short form English month name in upper case e.g. JAN
%Mmm	short form English month name with leading upper case e.g. Jan
%mm or %MM	number of month with leading zeroes e.g. 01
%m or %M	number of month with no leading zeroes e.g. 1
%dd or %DD	day number with leading zeroes e.g. 01
%d or %D	day number with no leading zeroes e.g. 1
%hh	hour number in 12 hour format and leading zeroes e.g. 01 for 13hrs
%h	hour number in 12 hour format with no leading zeroes
%HH	hour number in 24 hour format with leading zeros e.g. 01 for 1hrs
%H	hour number in 24 hour format with no leading zeroes e.g. 1 for 1hrs
%NN or %nn	minute number with leading zeroes e.g. 02
%N or %n	minute number with no leading zeroes e.g. 2
%SS or %ss	second number with leading zeroes e.g. 02
%s or %S	second number with no leading zeroes e.g. 2
%ampm	am or pm depending on the hour
%AMPM	AM or PM depending on the hour

gal_jd2cal [0.1]

Julian Date to Gregorian year, month, day, and fraction of a day.

```
void
gal_jd2cal
(
  double dj1,
  double dj2,
  int *iy,
  int *im,
  int *id,
  double *fd,
  gal_status_t *status
) ;
```

On entry DJ1 and DJ2 contain the Julian Date in standard SOFA two-piece format. On return IY contains the year, IM the month, ID the day, and FD the fractional part of day. The earliest valid date is -68569.5 (-4900 March, 1), and the largest value accepted is 10^9. If the date is outside of this range then the error code GAL_OUTSIDE_DATE_RANGE is set. In early eras the conversion is from the "Proleptic Gregorian Calendar"; no account is taken of the date(s) of adoption of the Gregorian Calendar, nor is the CE/BCE numbering convention observed.

References:

Explanatory Supplement to the Astronomical Almanac, P. Kenneth Seidelmann (ed.), University Science Books (1992), Section 12.92 (p604).

gal_jd2days [0.6]

This routine converts a JD date to year and day of year.

```
void
gal_jd2days
(
  double jd1,
  double jd2,
  int *year,
  double *days,
  gal_status_t *status
) ;
```

On entry JD1 and JD2 contain the Julian Date in standard two-piece SOFA format, on return YEAR contains the year number, and DAYS contains the day number. If an internal error occurs then the applicable error code is set, and YEAR and DAYS are set to zero.

gal_jdcalf [0.1]

Julian Date to Gregorian Calendar, expressed in a form convenient for formatting messages: rounded to a specified precision, and with the fields stored in a single array.

```
void
gal_jdcalf
(
  int ndp,
  double dj1,
  double dj2,
  int iymdf[4],
  gal_status_t *status
 ) ;
```

On entry DJ1 and DJ2 contain the Julian Date to be converted in standard SOFA two-piece format, NDP contains the required number of decimal places of days in fraction. On return IYMDF contain the year, month, day, and fraction in Gregorian calendar. If an error occurs then the applicable error code is set. In early eras the conversion is from the "Proleptic Gregorian Calendar"; no account is taken of the date(s) of adoption of the Gregorian Calendar, nor is the CE/BCE numbering convention observed. Refer to the routine gal_jd2cal. NDP should be 4 or less if internal overflows are to be avoided on machines which use 16-bit integers.

References:

Explanatory Supplement to the Astronomical Almanac, P. Kenneth Seidelmann (ed.), University Science Books (1992), Section 12.92 (p604).

gal_mafms [0.5]

This routine calculates the parameters for the Mars Fictitious Mean Sun and related parameters.

```
void
gal_mafms
(
  double tt1,
  double tt2,
  double *m,
  double *fms,
  double *pbs,
  double *ls,
  double *eot
) ;
```

On entry TT1 and TT2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return M contains the mean anomaly of Mars, FMS the Fictitious Mean Sun Angle, PBS the sum of angular perturbations in longitude, LS the areocentric solar longitude, and EOT the equation of time. All angles are in radians. LS may be used to determine the seasons of Mars: 0, $\pi/2$, π, $3\pi/2$ equates to the start of spring, summer, fall, and winter respectively. The referenced URL contains corrections to the referenced article.

References:

A post-Pathfinder evaluation of areocentric solar coordinates with improved timing recipes for Mars seasonal/diurnal climate studies by Michael Allison, Megan McEwen, Planetary and Space Science 48 (2000) 215-235

Mars24 URL: http://www.giss.nasa.gov/tools/mars24/help/algorithm.html

gal_maltst [0.5]

This routine converts a Mars Coordinated Time (MTC) date to Mars Local True Solar Time. Mars Coordinated Time is the Mean Solar Time on the Mars prime meridian.

```
double
gal_maltst
(
  double mtc1,
  double mtc2,
  double lambda,
  double eot
) ;
```

On entry MTC1 and MTC2 contain the Mars Coordinated Time (MTC) date in standard SOFA two-piece format, LAMBDA the longitude, and EOT the equation of time. The routine returns the Local True Solar Time. All angles are in radians. As defined, consistent with the terrestrial convention for Mean Solar Time, JD 2451549.5 (2000 January 6 00:00:00) corresponds to a near coincidence of the terrestrial Greenwich mean solar midnight and the Martian mean solar (prime meridian) midnight. The addition of the integer number 44796 assures a positive result for any date since JD 2405522 (1873 December 29.5). Longitude is measured westwards from the prime meridian (0 to 2π). The referenced URL contains corrections to the referenced article.

References:

A post-Pathfinder evaluation of areocentric solar coordinates with improved timing recipes for Mars seasonal/diurnal climate studies by Michael Allison, Megan McEwen, Planetary and Space Science 48 (2000) 215-235

Mars24 URL: http://www.giss.nasa.gov/tools/mars24/help/algorithm.html

gal_masudat [0.5]

This routine calculates a number of Sun data values for the planet Mars.

```
void
gal_masudat
(
  double mtc1,
  double mtc2,
  double m,
  double ls,
  double eot,
  double lon,
  double lat,
  double *ssl,
  double *sdec,
  double *mhd,
  double *mhlon,
  double *mhlat,
  double *sel,
  double *saz
) ;
```

On entry MTC1 and MTC2 contain the Mars Coordinated Time (MTC) date in standard SOFA two-piece format, M the Mars mean anomaly, LS the Areocentric solar longitude, EOT the equation of time, LON and LAT the longitude and latitude of the local observer. On return SSL contains the sub-solar longitude, SDEC the solar declination (planetographic), MHD the Mars heliocentric distance (AU), MHLON and MHLAT Mars' heliocentric longitude and latitude, and SEL and SAZ the local solar elevation and azimuth. All angles are in radians. As defined, consistent with the terrestrial convention for Mean Solar Time, JD 2451549.5 (2000 January 6 00:00:00) corresponds to a near coincidence of the terrestrial Greenwich mean solar midnight and the Martian mean solar (prime meridian) midnight. The addition of the integer number 44796 assures a positive result for any date since JD 2405522 (1873 December 29.5). Longitude is measured westwards from the prime meridian (0-2π). The referenced URL contains corrections to the referenced article.

References:

A post-Pathfinder evaluation of areocentric solar coordinates with improved timing recipes for Mars seasonal/diurnal climate studies by Michael Allison, Megan McEwen, Planetary and Space Science 48 (2000) 215-235

Mars24 URL: http://www.giss.nasa.gov/tools/mars24/help/algorithm.html

gal_mtc2tt [0.5]

This routine converts a Mars Coordinated Time (MTC) date to a Terrestrial Time (TT) date. Mars Coordinated Time is the Mean Solar Time on the Mars prime meridian.

```
void
gal_mtc2tt
(
  double mtc1,
  double mtc2,
  double *tt1,
  double *tt2
) ;
```

On entry MTC1 and MTC2 contain the Mars Coordinated Time (MTC) date. On return TT1 and TT2 contain the Terrestrial Time (TT) Julian Date. Both dates are in standard SOFA two-piece format As defined, consistent with the terrestrial convention for Mean Solar Time, JD 2451549.5 (2000 January 6 00:00:00) corresponds to a near coincidence of the terrestrial Greenwich mean solar midnight and the Martian mean solar (prime meridian) midnight. The addition of the integer number 44796 assures a positive result for any date since JD 2405522 (1873 December 29.5). The referenced URL contains corrections to the referenced article.

References:

A post-Pathfinder evaluation of areocentric solar coordinates with improved timing recipes for Mars seasonal/diurnal climate studies by Michael Allison, Megan McEwen, Planetary and Space Science 48 (2000) 215-235

Mars24 URL: http://www.giss.nasa.gov/tools/mars24/help/algorithm.html

gal_tai2tt [0.2]

This routine converts an International Atomic Time (TAI) Julian Date to a Terrestrial Time (TT) Julian Date.

```
void
gal_tai2tt
(
  double tai1,
  double tai2,
  double *tt1,
  double *tt2
) ;
```

On entry TAI1 and TAI2 contain an International Atomic Time (TAI) Julian Date. On return TT1 and TT2 contain the Terrestrial Time (TT) Julian Date. Both dates are in standard SOFA two-piece format.

References:

Explanatory Supplement to the Astronomical Supplement, Seidelmann P. Kenneth 1992, Pages 47-48

gal_tt2mtc [0.5]

This routine converts a Terrestrial Time (TT) date to a Mars Coordinated Time (MTC) date. Mars Coordinated Time is the Mean Solar Time on the Mars prime meridian.

```
void
gal_tt2mtc
(
  double tt1,
  double tt2,
  double *mtc1,
  double *mtc2
) ;
```

On entry TT1 and TT2 contain the Terrestrial Time (TT) Julian Date. On return MTC1 and MTC2 contain the Mars Coordinated Time (MTC) date. Both dates are in standard SOFA two-piece format. MTC1 contains the sol number, and MTC2 the fractional part of the sol. A Mars "day" is called a "sol". As defined, consistent with the terrestrial convention for Mean Solar Time, JD 2451549.5 (2000 January 6 00:00:00) corresponds to a near coincidence of the terrestrial Greenwich mean solar midnight and the Martian mean solar (prime meridian) midnight. The addition of the integer number 44796 assures a positive result for any date since JD 2405522 (1873 December 29.5). The referenced URL contains corrections to the referenced article.

References:

A post-Pathfinder evaluation of areocentric solar coordinates with improved timing recipes for Mars seasonal/diurnal climate studies by Michael Allison, Megan McEwen, Planetary and Space Science 48 (2000) 215-235

Mars24 URL: http://www.giss.nasa.gov/tools/mars24/help/algorithm.html

gal_utc2tai [0.2]

This routine converts a Coordinated Universal Time (UTC) Julian Date to an International Atomic Time (TAI) Julian Date.

```
void
```

```
gal_utc2tai
 (
  double utc1,
  double utc2,
  double *tai1,
  double *tai2,
  gal_status_t *status
 ) ;
```

On entry UTC1 and UTC2 contain the Coordinated Universal Time (UTC) Julian Date. On return TAI1 and TAI2 contain the International Atomic Time (TAI) Julian Date. All dates are in standard SOFA two-piece format. TAI began at 1960 January 1.0 (JD 2436934.5) and it is improper to call the routine with an earlier date. If this is attempted, zero is returned together with a warning status. Because leap seconds cannot, in principle, be predicted in advance, a reliable check for dates beyond the valid range is impossible. To guard against gross errors, a year five or more after the release year of this routine is considered dubious. In this case the warning status GAL_DUBIOUS_DATE is set but the result is computed in the normal way. If an error occurs then the applicable error code is set.

gal_utc2tt [0.2]

This routine converts a Coordinated Universal Time (UTC) Julian Date to a Terrestrial Time (TT) Julian Date.

```
void
gal_utc2tt
(
  double utc1,
  double utc2,
  double *tt1,
  double *tt2,
  gal_status_t *status
) ;
```

On entry UTC1 and UTC2 contain the Coordinated Universal Time (UTC) Julian Date. On return TT1 and TT2 contain the Terrestrial Time (TT) Julian Date. Both dates are in standard SOFA two-piece format. TAI began at 1960 January 1.0 (JD 2436934.5) and it is improper to call the routine with an earlier epoch. If this is attempted, zero is returned together with a warning status. Because leap seconds cannot, in principle, be predicted in advance, a reliable check for dates beyond the valid range is impossible. To guard against gross errors, a year five or more after the release year of this routine is considered dubious. In this case the warning code GAL_DUBIOUS_DATE is set but the result is computed in the normal way. If an error occurs then the applicable error code is set.

gal_utc2ut1 [0.2]

This routine converts a Coordinated Universal Time (UTC) Julian Date to a Universal Time (UT1) Julian Date.

```
void
gal_utc2ut1
(
  double utc1,
  double utc2,
  double dut1,
  double *ut1a,
  double *ut1b
) ;
```

On entry UTC1 and UTC2 contain the Coordinated Universal Time (UTC) Julian Date in standard SOFA two-piece format, DUT1 contains the UT1-UTC offset in seconds. On return UT1A and UT1B contain the Universal Time (UT1) Julian Date in standard SOFA two-piece format.

Chapter 7 – Reference Frames

The routines detailed in this chapter are defined in the gal_frames.h header file. The header file gal_frame_macros.h (included by gal_frames.h) defines the following constants:

```
/*
 * ---------------------------------------------
 * Constants for the Ellipsoid Model Identifiers
 * ---------------------------------------------
 */

enum {
  GAL_EMEA_DEL1800    =  0,  /* Delambre 1800                  */
  GAL_EMEA_AIRY1830   =  1,  /* Airy 1830                      */
  GAL_EMEA_EVER1830   =  2,  /* Everest 1830                   */
  GAL_EMEA_EVER1830BA =  3,  /* Everest 1830 Boni Alt          */
  GAL_EMEA_BESL1841   =  4,  /* Bessel 1841                    */
  GAL_EMEA_CL1866     =  5,  /* Clarke 1866                    */
  GAL_EMEA_CL1880     =  6,  /* Clarke 1880                    */
  GAL_EMEA_CLA1880M   =  7,  /* Clarke 1880 Modified           */
  GAL_EMEA_HEL1906    =  8,  /* Helmert 1906                   */
  GAL_EMEA_INTL1909   =  9,  /* International 1909             */
  GAL_EMEA_KRSV       = 10,  /* Krassovsky                     */
  GAL_EMEA_MERC1960   = 11,  /* Mercury 1960                   */
  GAL_EMEA_WGS1960    = 12,  /* World Geodetic System 1960     */
  GAL_EMEA_IAU1964    = 13,  /* IAU 1964                       */
  GAL_EMEA_AUSNAT1965 = 14,  /* Australian National 1965       */
  GAL_EMEA_WGS1966    = 15,  /* World Geodetic System 1966     */
  GAL_EMEA_MERC1968M  = 16,  /* Modified Mercury 1968          */
  GAL_EMEA_SA1969     = 17,  /* South American 1969            */
  GAL_EMEA_GRS1967    = 18,  /* Geodetic Reference System 1967 */
  GAL_EMEA_WGS1972    = 19,  /* World Geodetic System 1972     */
  GAL_EMEA_IAG1975    = 20,  /* IAG 1975                       */
  GAL_EMEA_IAU1976    = 21,  /* IAU 1976                       */
  GAL_EMEA_GRS1980    = 22,  /* Geodetic Reference System 1980 */
  GAL_EMEA_MERIT1983  = 23,  /* MERIT 1983                     */
  GAL_EMEA_WGS1984    = 24,  /* World Geodetic System 1984     */
  GAL_EMEA_IERS1989   = 25,  /* IERS 1989                      */
  GAL_EMEA_IERS2000   = 26,  /* IERS 2000                      */
  GAL_EMME_IAU1991    = 27,  /* IAU/IAG/COSPAR 1991 Mercury    */
  GAL_EMVE_IAU1991    = 28,  /* IAU/IAG/COSPAR 1991 Venus      */
  GAL_EMEA_IAU1991    = 29,  /* IAU/IAG/COSPAR 1991 Earth      */
  GAL_EMMA_IAU1991    = 30,  /* IAU/IAG/COSPAR 1991 Mars       */
  GAL_EMJU_IAU1991    = 31,  /* IAU/IAG/COSPAR 1991 Jupiter    */
  GAL_EMSA_IAU1991    = 32,  /* IAU/IAG/COSPAR 1991 Saturn     */
  GAL_EMUR_IAU1991    = 33,  /* IAU/IAG/COSPAR 1991 Uranus     */
  GAL_EMNE_IAU1991    = 34,  /* IAU/IAG/COSPAR 1991 Neptune    */
  GAL_EMPL_IAU1991    = 35,  /* IAU/IAG/COSPAR 1991 Pluto      */
  GAL_EMSU_IAU1991    = 36   /* IAU/IAG/COSPAR 1991 Sun        */
} ;

/*
 * ------------------------------
 * Constants for Reference Frames
 * ------------------------------
 */

enum {
  GAL_RF_J2000      =  1, /* Earth mean equator, dynamical equinox of J2000  */
  GAL_RF_B1950      =  2, /* Earth mean equator, dynamical equinox of B1950  */
  GAL_RF_FK4        =  3, /* Fundamental Catalog (4)                         */
  GAL_RF_DE118      =  4, /* JPL Developmental Ephemeris (118)               */
  GAL_RF_DE96       =  5, /* JPL Developmental Ephemeris (96)                */
  GAL_RF_DE102      =  6, /* JPL Developmental Ephemeris (102)               */
  GAL_RF_DE108      =  7, /* JPL Developmental Ephemeris (108)               */
  GAL_RF_DE111      =  8, /* JPL Developmental Ephemeris (111)               */
  GAL_RF_DE114      =  9, /* JPL Developmental Ephemeris (114)               */
  GAL_RF_DE122      = 10, /* JPL Developmental Ephemeris (122)               */
  GAL_RF_DE125      = 11, /* JPL Developmental Ephemeris (125)               */
  GAL_RF_DE130      = 12, /* JPL Developmental Ephemeris (130)               */
  GAL_RF_GALACTIC   = 13, /* Galactic System II                              */
  GAL_RF_DE200      = 14, /* JPL Developmental Ephemeris (200)               */
  GAL_RF_DE202      = 15, /* JPL Developmental Ephemeris (202)               */
  GAL_RF_MARSIAU    = 16, /* Mars Mean Equator and IAU vector of J2000       */
  GAL_RF_ECLIPJ2000 = 17, /* Ecliptic coordinates based upon the J2000 frame */
  GAL_RF_ECLIPB1950 = 18, /* Ecliptic coordinates based upon the B1950 frame */
  GAL_RF_DE140      = 19, /* JPL Developmental Ephemeris (140)               */
  GAL_RF_DE142      = 20, /* JPL Developmental Ephemeris (142)               */
  GAL_RF_DE143      = 21  /* JPL Developmental Ephemeris (143)               */
} ;
```

gal_bf2c [0.5]

This routine transforms a pv-vector from the planet body fixed reference frame to the alignment of the International Celestial Reference frame (ICRF).

```
void
gal_bf2c
(
  double bfrf[2][3],
  double alpha,
  double delta,
  double w,
  double omega,
  double icrf[2][3]
) ;
```

On entry the variables are set as follows:

BFRF	pv-vector in planet mean equator frame (meters, meters per second)
ALPHA	Right ascension of planet spin axis (radians)
DELTA	Declination of planet spin axis (radians)
W	Prime meridian angle (radians)
OMEGA	Planet rotation rate (radians per second)

On return ICRF contains the pv-vector in the ICRF inertial frame.

References:

Mars Pathfinder Project Planetary Constants and Models, Jet Propulsion Laboratory, December 1995, Chapters 2, 4, and 5

gal_bf2me [0.5]

This routine transforms a pv-vector from the planet body fixed reference frame to the planet mean equator reference frame.

```
void
gal_bf2me
(
  double bfrf[2][3],
  double w,
  double omega,
  double merf[2][3]
) ;
```

On entry the parameters are set as follows:

BFRF	pv-vector in planet body fixed frame (meters, meters per second)
W	Prime meridian angle (radians)
OMEGA	Planet rotation rate (radians per second)

On return MERF contains the pv-vector in planet mean equator frame.

References:

Mars Pathfinder Project Planetary Constants and Models, Jet Propulsion Laboratory, December 1995, Chapters 2, 4, and 5

gal_bi00 [0.1]

Frame bias components of IAU 2000 precession-nutation models (part of MHB2000 with additions).

```
void
gal_bi00
(
  double *dpsibi,
  double *depsbi,
  double *dra
) ;
```

On return DPSIBI and DEPSBI contain the longitude and obliquity corrections and DRA the ICRS right ascension of the J2000 mean equinox. The frame bias corrections in longitude and obliquity (radians) are required in order to correct for the offset between the GCRS pole and the mean J2000 pole. They define, with respect to the GCRS frame, a J2000 mean pole that is consistent with the rest of the IAU 2000A precession-nutation model. In addition to the displacement of the pole, the complete description of the frame bias requires also an offset in right ascension. This is not part of the IAU 2000A model, and is from Chapront et al. (2002). It is returned in radians. This is a supplemented implementation of one aspect of the IAU 2000A nutation model, formally adopted by the IAU General Assembly in 2000, namely MHB2000 (Mathews et al. 2002).

References:

Chapront, J., Chapront-Touze, M. & Francou, G., Astronomy & Astrophysics, 387, 700, 2002.

Mathews, P.M., Herring, T.A., Buffet, B.A., "Modeling of nutation and precession New nutation series for non-rigid Earth and insights into the Earth's interior", Journal

Geophysical Research, 107, B4, 2002. The MHB2000 code itself was obtained on 9th September 2002 from ftp://maia.usno.navy.mil/conv2000/chapter5/IAU2000A.

gal_bp00 [0.1]

Frame bias and precession, IAU 2000.

```
void
gal_bp00
(
  double date1,
  double date2,
  double rb[3][3],
  double rp[3][3],
  double rbp[3][3]
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return RB contains the frame bias matrix, RP the precession matrix, and RBP the bias-precession matrix. The matrix RB transforms vectors from GCRS to mean J2000 by applying frame bias. The matrix RP transforms vectors from J2000 mean equator and equinox to mean equator and equinox of date by applying precession. The matrix RBP transforms vectors from GCRS to mean equator and equinox of date by applying frame bias then precession. It is the product RP x RB. The celestial ephemeris origin (CEO) was renamed "celestial intermediate origin" (CIO) by IAU 2006 Resolution 2.

References:

Capitaine, N., Chapront, J., Lambert, S. and Wallace, P., "Expressions for the Celestial Intermediate Pole and Celestial Ephemeris Origin consistent with the IAU 2000A precession-nutation model", Astronomy & Astrophysics, 400, 1145-1154 (2003)

gal_bp06 [0.1]

Frame bias and precession, IAU 2006.

```
void
gal_bp06
(
  double date1,
  double date2,
  double rb[3][3],
  double rp[3][3],
  double rbp[3][3]
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return RB contains the frame bias matrix, RP the precession matrix, and RBP the bias-precession matrix. The matrix RB transforms vectors from GCRS to mean J2000 by applying frame bias. The matrix RP transforms vectors from mean J2000 to mean of date by applying precession. The matrix RBP transforms vectors from GCRS to mean of date by applying frame bias then precession. It is the product RP x RB.

References:

Capitaine, N. & Wallace, P.T., 2006, Astronomy & Astrophysics 450, 855

Wallace, P.T. & Capitaine, N., 2006, Astronomy & Astrophysics 459, 981

gal_bpn2xy [0.1]

Extract from the bias-precession-nutation matrix the X,Y coordinates of the Celestial Intermediate Pole.

```
void
gal_bpn2xy
(
  double rbpn[3][3],
  double *x,
  double *y
) ;
```

On entry RBPN contains the celestial-to-true matrix. On return X and Y contain the Celestial Intermediate Pole. The matrix rbpn transforms vectors from GCRS to true equator (and CIO or equinox) of date, and therefore the Celestial Intermediate Pole unit vector is the bottom row of the matrix. X and Y are components of the Celestial Intermediate Pole unit vector in the Geocentric Celestial Reference System. The celestial ephemeris origin (CEO) was renamed "celestial intermediate origin" (CIO) by IAU 2006 Resolution 2.

References:

Capitaine, N., Chapront, J., Lambert, S. and Wallace, P., "Expressions for the Celestial Intermediate Pole and Celestial Ephemeris Origin consistent with the IAU 2000A precession-nutation model", Astronomy & Astrophysics, 400, 1145-1154 (2003)

gal_c2bf [0.5]

This routine transforms a pv-vector from the alignment of the ICRF reference frame to the planet body fixed reference frame.

```
void
gal_c2bf
(
  double icrf[2][3],
  double alpha,
  double delta,
  double w,
  double omega,
  double bfrf[2][3]
) ;
```

On entry the parameters are set as follows:

ICRF	pv-vector in ICRF inertial frame (meters, meters per second)
ALPHA	Right ascension of planet spin axis (radians)
DELTA	Declination of planet spin axis (radians)
W	Prime meridian angle (radians)
OMEGA	Planet rotation rate (radians per second)

On return BFRF contains the pv-vector in planet mean equator frame.

References:

Mars Pathfinder Project Planetary Constants and Models, Jet Propulsion Laboratory, December 1995, Chapters 2, 4, and 5

gal_c2i00a [0.1]

Form the celestial-to-intermediate matrix for a given date using the IAU 2000A precession-nutation model.

```
void
gal_c2i00a
(
  double date1,
  double date2,
  double rc2i[3][3]
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return RC2I contains the celestial-to-intermediate matrix. The matrix RC2I is the first stage in the transformation from celestial to terrestrial coordinates:

[ITRS] = RPOM * R_3(ERA) * RC2I * [GCRS]

= RC2T * [GCRS]

where [GCRS] is a vector in the Geocentric Celestial Reference System and [ITRS] is a vector in the International Terrestrial Reference System (see IERS Conventions 2003), ERA is the Earth Rotation Angle and RPOM is the polar motion matrix. A faster, but slightly less accurate result (about 1 mas), can be obtained by using instead the gal_c2i00b routine. The celestial ephemeris origin (CEO) was renamed "celestial intermediate origin" (CIO) by IAU 2006 Resolution 2.

References:

Capitaine, N., Chapront, J., Lambert, S. and Wallace, P., "Expressions for the Celestial Intermediate Pole and Celestial Ephemeris Origin consistent with the IAU 2000A precession-nutation model", Astronomy & Astrophysics, 400, 1145-1154 (2003)

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

gal_c2i00b [0.1]

Form the celestial-to-intermediate matrix for a given date using the IAU 2000B precession-nutation model.

```
void
gal_c2i00b
(
  double date1,
  double date2,
  double rc2i[3][3]
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return RC2I contains the celestial-to-intermediate matrix. The matrix RC2I is the first stage in the transformation from celestial to terrestrial coordinates:

[ITRS] = RPOM * R_3(ERA) * RC2I * [GCRS]

= RC2T * [GCRS]

where [GCRS] is a vector in the Geocentric Celestial Reference System and [ITRS] is a vector in the International Terrestrial Reference System (see IERS Conventions 2003), ERA is the Earth Rotation Angle and RPOM is the polar motion matrix. This routine is faster, but slightly less accurate (about 1 mas), than the gal_c2i00a routine. The celestial ephemeris origin (CEO) was renamed "celestial intermediate origin" (CIO) by IAU 2006 Resolution 2.

References:

Capitaine, N., Chapront, J., Lambert, S. and Wallace, P., "Expressions for the Celestial Intermediate Pole and Celestial Ephemeris Origin consistent with the IAU 2000A precession-nutation model", Astronomy & Astrophysics, 400, 1145-1154 (2003).

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

gal_c2i06a [0.1]

Form the celestial-to-intermediate matrix for a given date using the IAU 2006 precession and IAU 2000A nutation models.

```
void
gal_c2i06a
(
  double date1,
  double date2,
  double rc2i[3][3]
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return RC2I contains the celestial-to-intermediate matrix. The matrix RC2I is the first stage in the transformation from celestial to terrestrial coordinates:

[ITRS] = RPOM * R_3(ERA) * RC2I * [GCRS]

= RC2T * [GCRS]

where [GCRS] is a vector in the Geocentric Celestial Reference System and [ITRS] is a vector in the International Terrestrial Reference System (see IERS Conventions 2003), ERA is the Earth Rotation Angle and RPOM is the polar motion matrix.

References:

McCarthy, D. D., Petit, G. (eds.), 2004, IERS Conventions (2003), IERS Technical Note No. 32, BKG

Capitaine, N. & Wallace, P.T., 2006, Astronomy & Astrophysics 450, 855

Wallace, P.T. & Capitaine, N., 2006, Astronomy & Astrophysics 459, 981

gal_c2ibpn [0.1]

Form the celestial-to-intermediate matrix for a given date given the bias – precession - nutation matrix. IAU 2000.

```
void
gal_c2ibpn
(
  double date1,
  double date2,
  double rbpn[3][3],
  double rc2i[3][3]
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format, and RBPN contains the celestial-to-true matrix. On return RC2I contains the celestial-to-intermediate matrix. The matrix RBPN transforms vectors from GCRS to true equator (and CIO or equinox) of date. Only the CIP (bottom row) is used. The matrix RC2I is the first stage in the transformation from celestial to terrestrial coordinates:

[ITRS] = RPOM * R_3(ERA) * RC2I * [GCRS]

= RC2T * [GCRS]

where [GCRS] is a vector in the Geocentric Celestial Reference System and [ITRS] is a vector in the International Terrestrial Reference System (see IERS Conventions 2003), ERA is the Earth Rotation Angle and RPOM is the polar motion matrix. Although its name does not include "00", this routine is in fact specific to the IAU 2000 models. The celestial ephemeris origin (CEO) was renamed "celestial intermediate origin" (CIO) by IAU 2006 Resolution 2.

References:

Capitaine, N., Chapront, J., Lambert, S. and Wallace, P., "Expressions for the Celestial Intermediate Pole and Celestial Ephemeris Origin consistent with the IAU 2000A precession-nutation model", Astronomy & Astrophysics, 400, 1145-1154 (2003).

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

gal_c2ixy [0.1]

Form the celestial to intermediate-frame-of-date matrix for a given date when the CIP X,Y coordinates are known. IAU 2000.

```
void
gal_c2ixy
(
  double date1,
  double date2,
  double x,
  double y,
  double rc2i[3][3]
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format, X and Y contain the Celestial Intermediate Pole. On return RC2I contain the celestial-to-intermediate matrix. The Celestial Intermediate Pole coordinates are the X and Y components of the unit vector in the Geocentric Celestial Reference System. The matrix RC2I is the first stage in the transformation from celestial to terrestrial coordinates:

[ITRS] = RPOM * R_3(ERA) * RC2I * [GCRS]

= RC2T * [GCRS]

where [GCRS] is a vector in the Geocentric Celestial Reference System and [ITRS] is a vector in the International Terrestrial Reference System (see IERS Conventions 2003), ERA is the Earth Rotation Angle and RPOM is the polar motion matrix. Although its name does not include "00", this routine is in fact specific to the IAU 2000 models.

References:

McCarthy D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004).

gal_c2ixys [0.1]

Form the celestial to intermediate-frame-of-date matrix given the CIP x,y and the CIO locator s.

```
void
gal_c2ixys
(
  double x,
  double y,
  double s,
  double rc2i[3][3]
) ;
```

On entry X and Y contain the coordinates of the Celestial Intermediate Pole, and S contains the CIO locator. On return RC2I contains the celestial-to-intermediate matrix. The Celestial Intermediate Pole coordinates are the X and Y components of the unit vector in the Geocentric Celestial Reference System. The CIO locator (radians) positions the Celestial Intermediate Origin on the equator of the CIP. The matrix RC2I is the first stage in the transformation from celestial to terrestrial coordinates:

[ITRS] = RPOM * R_3(ERA) * RC2I * [GCRS]

= RC2T * [GCRS]

where [GCRS] is a vector in the Geocentric Celestial Reference System and [ITRS] is a vector in the International Terrestrial Reference System (see IERS Conventions 2003), ERA is the Earth Rotation Angle and RPOM is the polar motion matrix.

References:

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

gal_c2me [0.5]

This routine transforms a pv-vector from the alignment of the International Celestial Reference Frame (ICRF) to the planet mean equator reference frame.

```
void
gal_c2me
(
  double icrf[2][3],
  double alpha,
  double delta,
  double merf[2][3]
) ;
```

On entry the parameters are set as follows:

ICRF	pv-vector in ICRF inertial frame (meters, meters per second)
ALPHA	Right ascension of planet spin axis (radians)
DELTA	Declination of planet spin axis (radians)

On return MERF contains the pv-vector in planet mean equator frame.

References:

Mars Pathfinder Project Planetary Constants and Models, Jet Propulsion Laboratory, December 1995, Chapters 2, 4, and 5

gal_c2radec [0.2]

This routine converts a position and velocity vector in the Geocentric Celestial Reference Frame (GCRF) to right ascension and declination.

```
void
gal_c2radec
(
  double gcrf[2][3],
  double *ra,
  double *dec,
  double *range,
  double *radot,
  double *decdot,
  double *rangedot
) ;
```

On entry GCRF contains the GCRF position and velocity vector (meters, meters per second). On return the variables are set as follows:

RA	Right ascension (radians)
DEC	Declination (radians)
RANGE	Range (meters)
RADOT	Rate of change of right ascension (radians per second)
DECDOT	Rate of change of declination (radians per second)
RANGEDOT	Rate of change of range (range rate) (meters per second)

References:

Fundamentals of Astrodynamics and Applications, Vallado, David A. Second Edition 2004, Pages 248-250

gal_c2t00a [0.1]

Form the celestial to terrestrial matrix given the date, the Universal Time (UT1) and the polar motion, using the IAU 2000A nutation model.

```
void
```

```
gal_c2t00a
(
  double tta,
  double ttb,
  double uta,
  double utb,
  double xp,
  double yp,
  double rc2t[3][3]
) ;
```

On entry TTA and TTB contain the Terrestrial Time (TT) Julian Date, UTA and UTB the Universal Time (UT1) Julian Date, and XP and YP contain the coordinates of the pole (radians). All dates are in standard SOFA two-piece format. On return RC2T contains the celestial-to-terrestrial matrix. In the case of UTA and UTB, the date & time method is best matched to the Earth rotation angle algorithm used: maximum accuracy (or, at least, minimum noise) is delivered when the UTA argument is for 0hrs UT1 on the day in question and the UTB argument lies in the range 0 to 1, or vice versa. XP and YP are the "coordinates of the pole", in radians, which position the Celestial Intermediate Pole in the International Terrestrial Reference System (see IERS Conventions 2003), measured along the meridians to 0 and 90 deg west respectively. The matrix RC2T transforms from celestial to terrestrial coordinates:

[ITRS] = RPOM * R_3(ERA) * RC2I * [GCRS]

= RC2T * [GCRS]

where [GCRS] is a vector in the Geocentric Celestial Reference System and [ITRS] is a vector in the International Terrestrial Reference System (see IERS Conventions 2003), RC2I is the celestial-to-intermediate matrix, ERA is the Earth rotation angle and RPOM is the polar motion matrix. A faster, but slightly less accurate result (about 1 mas), can be obtained by using instead the gal_c2t00b routine.

References:

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

gal_c2t00b [0.1]

Form the celestial to terrestrial matrix given the date, the Universal Time (UT1) and the polar motion, using the IAU 2000B nutation model.

```
void
gal_c2t00b
```

```
(
  double tta,
  double ttb,
  double uta,
  double utb,
  double xp,
  double yp,
  double rc2t[3][3]
) ;
```

On entry TTA and TTB contain the Terrestrial Time (TT) Julian Date, UTA and UTB the Universal Time (UT1) Julian Date, and XP and YP contain the coordinates of the pole (radians). All dates are in standard SOFA two-piece format. On return RC2T contains the celestial-to-terrestrial matrix. In the case of UTA and UTB, the date & time method is best matched to the Earth rotation angle algorithm used: maximum accuracy (or, at least, minimum noise) is delivered when the UTA argument is for 0hrs UT1 on the day in question and the UTB argument lies in the range 0 to 1, or vice versa. XP and YP are the "coordinates of the pole", in radians, which position the Celestial Intermediate Pole in the International Terrestrial Reference System (see IERS Conventions 2003), measured along the meridians to 0 and 90 deg west respectively. The matrix RC2T transforms from celestial to terrestrial coordinates:

[ITRS] = RPOM * R_3(ERA) * RC2I * [GCRS]

= RC2T * [GCRS]

where [GCRS] is a vector in the Geocentric Celestial Reference System and [ITRS] is a vector in the International Terrestrial Reference System (see IERS Conventions 2003), RC2I is the celestial-to-intermediate matrix, ERA is the Earth rotation angle and RPOM is the polar motion matrix. This routine is faster, but slightly less accurate (about 1 mas), than the gal_c2t00a routine.

References:

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

gal_c2t06a [0.1]

Form the celestial to terrestrial matrix given the date, the Universal Time (UT1) and the polar motion, using the IAU 2006 precession and IAU 2000A nutation models.

```
void
gal_c2t06a
(
  double tta,
  double ttb,
  double uta,
  double utb,
  double xp,
  double yp,
  double rc2t[3][3]
) ;
```

On entry TTA and TTB contain the Terrestrial Time (TT) Julian Date, UTA and UTB the Universal Time (UT1) Julian Date, and XP and YP contain the coordinates of the pole (radians). All dates are in standard SOFA two-piece format. On return RC2T contains the celestial-to-terrestrial matrix. In the case of UTA and UTB, the date & time method is best matched to the Earth rotation angle algorithm used: maximum accuracy (or, at least, minimum noise) is delivered when the UTA argument is for 0hrs UT1 on the day in question and the UTB argument lies in the range 0 to 1, or vice versa. XP and YP are the "coordinates of the pole", in radians, which position the Celestial Intermediate Pole in the International Terrestrial Reference System (see IERS Conventions 2003), measured along the meridians to 0 and 90 deg west respectively. The matrix RC2T transforms from celestial to terrestrial coordinates:

[ITRS] = RPOM * R_3(ERA) * RC2I * [GCRS]

= RC2T * [GCRS]

where [GCRS] is a vector in the Geocentric Celestial Reference System and [ITRS] is a vector in the International Terrestrial Reference System (see IERS Conventions 2003), RC2I is the celestial-to-intermediate matrix, ERA is the Earth rotation angle and RPOM is the polar motion matrix.

References:

McCarthy, D. D., Petit, G. (eds.), 2004, IERS Conventions (2003), IERS Technical Note No. 32, BKG

gal_c2tceo [0.1]

Assemble the celestial to terrestrial matrix from CIO-based components (the celestial-to-intermediate matrix, the Earth Rotation Angle and the polar motion matrix).

```
#define gal_c2tceo( rc2i, era, rpom, rc2t ) gal_c2tcio( rc2i, (
era ), rpom, rc2t )
```

On entry RC2I contains the celestial-to-intermediate matrix, ERA the Earth rotation angle, and RPOM the polar-motion matrix. On return RC2T contains the celestial-to-terrestrial matrix. The name of this routine, gal_c2tceo, reflects the original name of the celestial intermediate origin (CIO), which before the adoption of IAU 2006 Resolution 2 was called the "celestial ephemeris origin" (CEO). When the name change from CEO to CIO occurred, a new routine called gal_c2tcio was introduced as the successor to the existing gal_c2tceo. This routine is merely a front end to the new one. The routine is included in the collection only to support existing applications. It should not be used in new applications. The routine is a candidate for deprecation.

gal_c2tcio [0.1]

Assemble the celestial to terrestrial matrix from CIO-based components (the celestial-to-intermediate matrix, the Earth Rotation Angle and the polar motion matrix).

```
void
gal_c2tcio
(
  double rc2i[3][3],
  double era,
  double rpom[3][3],
  double rc2t[3][3]
) ;
```

On entry RC2I contains the celestial-to-intermediate matrix, ERA the Earth rotation angle, and RPOM the polar-motion matrix. On return RC2T contains the celestial-to-terrestrial matrix. This routine constructs the rotation matrix that transforms vectors in the celestial system into vectors in the terrestrial system. It does so starting from pre-computed components, namely the matrix which rotates from celestial coordinates to the intermediate frame, the Earth rotation angle and the polar motion matrix. One use of this routine is when generating a series of celestial-to-terrestrial matrices where only the Earth Rotation Angle changes, avoiding the considerable overhead of re-computing the precession-nutation more often than necessary to achieve given accuracy objectives. The relationship between the arguments is as follows:

[ITRS] = RPOM * R_3(ERA) * RC2I * [GCRS]

= RC2T * [GCRS]

where [GCRS] is a vector in the Geocentric Celestial Reference System and [ITRS] is a vector in the International Terrestrial Reference System (see IERS Conventions 2003).

References:

McCarthy, D. D., Petit, G. (eds.), 2004, IERS Conventions (2003), IERS Technical Note No. 32, BKG

gal_c2teqx [0.1]

Assemble the celestial to terrestrial matrix from equinox-based components (the celestial-to-true matrix, the Greenwich Apparent Sidereal Time and the polar motion matrix).

```
void
gal_c2teqx
(
  double rbpn[3][3],
  double gst,
  double rpom[3][3],
  double rc2t[3][3]
) ;
```

On entry RBPN contains the celestial-to-true matrix, GST the Greenwich (Apparent) Sidereal Time, and RPOM the polar-motion matrix. On return RC2T contains the celestial-to-terrestrial matrix. This routine constructs the rotation matrix that transforms vectors in the celestial system into vectors in the terrestrial system. It does so starting from pre-computed components, namely the matrix which rotates from celestial coordinates to the true equator and equinox of date, the Greenwich Apparent Sidereal Time and the polar motion matrix. One use of the routine is when generating a series of celestial-to-terrestrial matrices where only the Sidereal Time changes, avoiding the considerable overhead of re-computing the precession-nutation more often than necessary to achieve given accuracy objectives. The relationship between the arguments is as follows:

[ITRS] = RPOM * R_3(GST) * RBPN * [GCRS]

= RC2T * [GCRS]

where [GCRS] is a vector in the Geocentric Celestial Reference System and [ITRS] is a vector in the International Terrestrial Reference System (see IERS Conventions 2003).

References:

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

gal_c2tpe [0.1]

Form the celestial to terrestrial matrix given the date, the Universal Time (UT1), the nutation and the polar motion. IAU 2000.

```
void
```

```
gal_c2tpe
(
  double tta,
  double ttb,
  double uta,
  double utb,
  double dpsi,
  double deps,
  double xp,
  double yp,
  double rc2t[3][3]
) ;
```

On entry TTA and TTB contain the Terrestrial Time (TT) Julian Date, UTA and UTB contain the Universal Time (UT1) Julian Date, DPSI and DEPS the nutation, and XP and YP the coordinates of the pole (radians). All dates are in standard SOFA two-piece format. On return RC2T contains the celestial-to-terrestrial matrix. In the case of UTA and UTB, the date & time method is best matched to the Earth rotation angle algorithm used: maximum accuracy (or, at least, minimum noise) is delivered when the UTA argument is for 0hrs UT1 on the day in question and the UTB argument lies in the range 0 to 1, or vice versa. The caller is responsible for providing the nutation components; they are in longitude and obliquity, in radians and are with respect to the equinox and ecliptic of date. For high-accuracy applications, free core nutation should be included as well as any other relevant corrections to the position of the CIP. XP and YP are the "coordinates of the pole", in radians, which position the Celestial Intermediate Pole in the International Terrestrial Reference System (see IERS Conventions 2003), measured along the meridians to 0 and 90 deg west respectively. The matrix RC2T transforms from celestial to terrestrial coordinates:

[ITRS] = RPOM * R_3(GST) * RBPN * [GCRS]

= RC2T * [GCRS]

where [GCRS] is a vector in the Geocentric Celestial Reference System and [ITRS] is a vector in the International Terrestrial Reference System (see IERS Conventions 2003), RBPN is the bias-precession-nutation matrix, GST is the Greenwich (Apparent) Sidereal Time and RPOM is the polar motion matrix. Although its name does not include "00", this routine is in fact specific to the IAU 2000 models.

References:

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

gal_c2tpv00a [0.2]

This routine converts a position & velocity vector in the Geocentric Celestial Reference Frame (GCRF) to the International Terrestrial Reference Frame (ITRF) (IAU 2000A Resolutions).

```
void
gal_c2tpv00a
(
  double gcrf[2][3],
  double utc1,
  double utc2,
  double dut1,
  double lod,
  double xp,
  double yp,
  double itrf[2][3],
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

GCRF	GCRF position & velocity vector (meters, meters per second)
UTC1	Date part 1 (UTC)
UTC2	Date part 2 (UTC)
DUT1	UT1 - UTC (seconds)
LOD	Excess length of day (seconds)
XP	X coordinate of the pole (radians)
YP	Y coordinate of the pole (radians)

On return ITRF contains the ITRF position & velocity vector (meters, meters per second). The date UTC1 and UTC2 is a Coordinated Universal Time Julian Date in standard SOFA two-piece format. XP and YP are the "coordinates of the pole", in seconds, which position the Celestial Intermediate Pole in the International Terrestrial Reference System (see IERS Conventions 2003, measured along the meridians to 0 and 90 deg west respectively. If any internal errors occur the applicable error code is set.

References:

SOFA Tools for Earth Attitude IAU Standards for Fundamental Astronomy Review Board 2007

http://www.iau-sofa.rl.ac.uk

Fundamentals of Astrodynamics and Applications, Vallado, David A. Second Edition

2004, Pages 217-219

gal_c2tpv00b [0.2]

This routine converts a position & velocity vector in the Geocentric Celestial Reference Frame (GCRF) to the International Terrestrial Reference Frame (ITRF) (IAU 2000B Resolutions).

```
void
gal_c2tpv00b
(
  double gcrf[2][3],
  double utc1,
  double utc2,
  double dut1,
  double lod,
  double xp,
  double yp,
  double itrf[2][3],
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

GCRF	GCRF position & velocity vector (meters, meters per second)
UTC1	Date part 1 (UTC)
UTC2	Date part 2 (UTC)
DUT1	UT1 - UTC (seconds)
LOD	Excess length of day (seconds)
XP	X coordinate of the pole (radians)
YP	Y coordinate of the pole (radians)

On return ITRF contains the ITRF position & velocity vector (meters, meters per second). The date UTC1 and UTC2 is a Coordinated Universal Time Julian Date in standard SOFA two-piece format. XP and YP are the "coordinates of the pole", in seconds, which position the Celestial Intermediate Pole in the International Terrestrial Reference System (see IERS Conventions 2003, measured along the meridians to 0 and 90 deg west respectively. If any internal errors occur then the applicable error code is set.

References:

SOFA Tools for Earth Attitude IAU Standards for Fundamental Astronomy Review Board 2007 http://www.iau-sofa.rl.ac.uk

Fundamentals of Astrodynamics and Applications, Vallado, David A. Second Edition 2004, Pages 217-219

gal_c2tpv06a [0.2]

This routine converts a position & velocity vector in the Geocentric Celestial Reference Frame (GCRF) to the International Terrestrial Reference Frame (ITRF) (IAU 2006A Resolutions).

```
void
gal_c2tpv06a
(
  double gcrf[2][3],
  double utc1,
  double utc2,
  double dut1,
  double lod,
  double xp,
  double yp,
  double itrf[2][3],
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

GCRF	GCRF position & velocity vector (meters, meters per second)
UTC1	Date part 1 (UTC)
UTC2	Date part 2 (UTC)
DUT1	UT1 - UTC (seconds)
LOD	Excess length of day (seconds)
XP	X coordinate of the pole (radians)
YP	Y coordinate of the pole (radians)

On return ITRD contains the ITRF position & velocity vector (meters, meters per second). The date UTC1 and UTC2 is a Coordinated Universal Time Julian Date in standard SOFA two-piece format. XP and YP are the "coordinates of the pole", in seconds, which position the Celestial Intermediate Pole in the International Terrestrial Reference System (see IERS Conventions 2003, measured along the meridians to 0 and 90 deg west respectively. If any internal errors occur then the applicable error codes is set.

References:

SOFA Tools for Earth Attitude IAU Standards for Fundamental Astronomy Review Board 2007

http://www.iau-sofa.rl.ac.uk

Fundamentals of Astrodynamics and Applications, Vallado, David A. Second Edition 2004, Pages 217-219

gal_c2txy [0.1]

Form the celestial to terrestrial matrix given the date, the Universal Time (UT1), the CIP coordinates and the polar motion. IAU 2000.

```
void
gal_c2txy
(
  double tta,
  double ttb,
  double uta,
  double utb,
  double x,
  double y,
  double xp,
  double yp,
  double rc2t[3][3]
) ;
```

On entry TTA and TTB contain the Terrestrial Time (TT) Julian Date, UTA and UTB the Universal Time (UT1) Julian Date, X and Y the Celestial Intermediate Pole, and XP and YP the coordinates of the pole (radians). All dates are in standard SOFA two-piece format. On return RC2T contains the celestial-to-terrestrial matrix. In the case of UTA and UTB, the date & time method is best matched to the Earth rotation angle algorithm used: maximum accuracy (or, at least, minimum noise) is delivered when the UTA argument is for 0hrs UT1 on the day in question and the UTB argument lies in the range 0 to 1, or vice versa. The Celestial Intermediate Pole coordinates are the X and Y components of the unit vector in the Geocentric Celestial Reference System. XP and YP are the "coordinates of the pole", in radians, which position the Celestial Intermediate Pole in the International Terrestrial Reference System (see IERS Conventions 2003), measured along the meridians to 0 and 90 deg west respectively. The matrix rc2t transforms from celestial to terrestrial coordinates:

[ITRS] = RPOM * R_3(ERA) * RC2I * [GCRS]

= RC2T * [GCRS]

where [GCRS] is a vector in the Geocentric Celestial Reference System and [ITRS] is a vector in the International Terrestrial Reference System (see IERS Conventions 2003), ERA is the Earth Rotation Angle and RPOM is the polar motion matrix. Although its name does not include "00", this routine is in fact specific to the IAU 2000 models.

References:

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

gal_chgirf [0.6]

Form rotation matrix to change from one inertial reference frame to another.

```
void
gal_chgirf
(
  int refa,
  int refb,
   double rotab[3][3],
  gal_status_t *status
) ;
```

On entry REFA contains the identifier code of the inertial reference frame to change from, and REFB contains the identifier code of the inertial reference frame to change to. On return ROTAB contains the rotation matrix such that B = ROTAB * A. If an error occurs then the applicable error codes are set. The header file gal_frame_macros.h defines the constants for the identifier codes.

References:

[1] Jay Lieske, ``Precession Matrix Based on IAU (1976)
System of Astronomical Constants," Astron. Astrophys.
73, 282-284 (1979).

[2] E.M. Standish, Jr., ``Orientation of the JPL Ephemerides,
DE 200/LE 200, to the Dynamical Equinox of J2000,"
Astron. Astrophys. 114, 297-302 (1982).

[3] E.M. Standish, Jr., ``Conversion of Ephemeris Coordinates
from the B1950 System to the J2000 System," JPL IOM
314.6-581, 24 June 1985.

[4] E.M. Standish, Jr., ``The Equinox Offsets of the JPL
Ephemeris," JPL IOM 314.6-929, 26 February 1988.

[5] Jay Lieske, ``Expressions for the Precession Quantities
Based upon the IAU (1976) System of Astronomical
Constants" Astron. Astrophys. 58, 1-16 (1977).

[6] Laura Bass and Robert Cesarone "Mars Observer Planetary
Constants and Models" JPL D-3444 November 1990.

[7] "Explanatory Supplement to the Astronomical Almanac"
edited by P. Kenneth Seidelmann. University Science
Books, 20 Edgehill Road, Mill Valley, CA 94941 (1992)

gal_ee00 [0.1]

The equation of the equinoxes, compatible with IAU 2000 resolutions, given the nutation in longitude and the mean obliquity.

```
double
gal_ee00
(
  double date1,
  double date2,
  double epsa,
  double dpsi
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format, EPSA contains the mean obliquity, and DPSI the nutation in longitude. The routine returns the equation of the equinoxes. The obliquity (radians), is mean of date. The result, which is in radians, operates in the following sense:

> Greenwich Apparent Sidereal Time = Greenwich Mean Sidereal Time + equation of the equinoxes

The result is compatible with the IAU 2000 resolutions. For further details, see IERS Conventions 2003 and Capitaine et al. (2002).

References:

Capitaine, N., Wallace, P.T. and McCarthy, D.D., "Expressions to implement the IAU 2000 definition of UT1", Astronomy & Astrophysics, 406, 1135-1149 (2003)

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

gal_ee00a [0.1]

Equation of the equinoxes, compatible with IAU 2000 resolutions.

```
double
gal_ee00a
(
  double date1,
  double date2
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. The routine returns the equation of the equinoxes. The result, which is in radians, operates in the following sense:

> Greenwich Apparent Sidereal Time = Greenwich Mean Sidereal Time + equation of the equinoxes

The result is compatible with the IAU 2000 resolutions. For further details, see IERS Conventions 2003 and Capitaine et al. (2002).

References:

Capitaine, N., Wallace, P.T. and McCarthy, D.D., "Expressions to implement the IAU 2000 definition of UT1", Astronomy & Astrophysics, 406, 1135-1149 (2003)

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

gal_ee00b [0.1]

Equation of the equinoxes, compatible with IAU 2000 resolutions but using the truncated nutation model IAU 2000B.

```
double
gal_ee00b
(
  double date1,
  double date2
) ;
```

On entry DATE1 and DATE2 contains the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. The routine returns the equation of the equinoxes. The result, which is in radians, operates in the following sense:

> Greenwich Apparent Sidereal Time = Greenwich Mean Sidereal Time + equation

of the equinoxes

The result is compatible with the IAU 2000 resolutions except that accuracy has been compromised for the sake of speed. For further details, see McCarthy & Luzum (2001), IERS Conventions 2003 and Capitaine et al. (2003).

References:

Capitaine, N., Wallace, P.T. and McCarthy, D.D., "Expressions to implement the IAU 2000 definition of UT1", Astronomy & Astrophysics, 406, 1135-1149 (2003)

McCarthy, D.D. & Luzum, B.J., "An abridged model of the precession-nutation of the celestial pole", Celestial Mechanics & Dynamical Astronomy, 85, 37-49 (2003)

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

gal_ee06a [0.1]

Equation of the equinoxes, compatible with IAU 2000 resolutions and IAU 2006/2000A precession-nutation.

```
double
gal_ee06a
(
  double date1,
  double date2
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. The routine returns the equation of the equinoxes. The result, which is in radians, operates in the following sense:

> Greenwich Apparent Sidereal Time = Greenwich Mean Sidereal Time + equation of the equinoxes

References:

McCarthy, D. D., Petit, G. (eds.), 2004, IERS Conventions (2003), IERS Technical Note No. 32, BKG

gal_eect00 [0.1]

Equation of the equinoxes complementary terms, consistent with IAU 2000 resolutions.

```
double
gal_eect00
(
  double date1,
  double date2
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. The routine returns the complementary terms. The "complementary terms" are part of the equation of the equinoxes (EE), classically the difference between apparent and mean Sidereal Time:

> Greenwich Apparent Sidereal Time = Greenwich Mean Sidereal Time + EE.
>
> with:
>
> EE = DPSI * cos(EPS)

where DPSI is the nutation in longitude and EPS is the obliquity of date. However, if the rotation of the Earth were constant in an inertial frame the classical formulation would lead to apparent irregularities in the UT1 timescale traceable to side-effects of precession-nutation. In order to eliminate these effects from UT1, "complementary terms" were introduced in 1994 (IAU, 1994) and took effect from 1997 (Capitaine and Gontier, 1993):

> Greenwich Apparent Sidereal Time = Greenwich Mean Sidereal Time + complementary terms + equation of the equinoxes

By convention, the complementary terms (CT) are included as part of the equation of the equinoxes rather than as part of the mean Sidereal Time. This slightly compromises the "geometrical" interpretation of mean sidereal time but is otherwise inconsequential. This routine computes CT in the above expression, compatible with IAU 2000 resolutions (Capitaine et al., 2002, and IERS Conventions 2003).

References:

Capitaine, N. & Gontier, A.-M., Astronomy & Astrophysics, 275, 645-650 (1993)

Capitaine, N., Wallace, P.T. and McCarthy, D.D., "Expressions to implement the IAU 2000 definition of UT1", Astronomy & Astrophysics, 406, 1135-1149 (2003)

IAU Resolution C7, Recommendation 3 (1994)

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

gal_eform [0.6]

Get Earth ellipsoid parameters.

```
void
gal_eform
(
  int n,
  double *a,
  double *f,
  gal_status_t *status
) ;
```

On entry N contains the identifier code of the requested ellipsoid model. On return the variables are set as follows:

A	Equatorial radius
F	Flattening factor

The ellipsoid parameters are returned in the form of equatorial radius in meters (A) and flattening (F). The latter is a number around 0.00335, i.e. around 1/298. For the case where an unsupported N value is supplied, zero A and F are returned, and the error status GAL_INVALID_ID is set.

gal_emdetails [0.2]

This routine returns the full details of the requested ellipsoid model.

```
void
gal_emdetails
(
  const int em,
  int *body,
  char *name,
  double *sma,
  double *inf,
  gal_status_t *status
) ;
```

On entry EM contains the identifier code of the requested ellipsoid model. On return the variables are set as follows:

BODY	Solar System Body Identifier
NAME	Ellipsoid Model name
SMA	Semi-major axis (meters)

INF	Inverse flattening factor

If value of the ellipsoid identifer EM is unknown to the routine then the error code GAL_INVALID_ID is set and SMA and INF are set to zero.

The header file gal_frame_macros.h defines the constants for the valid values of EM.

References:

Explanatory Supplement to the Astronomical Almanac Edited by P. Kenneth Seidelmann, 1992 Page 220

Map Projection Transformations by Qihe Yang, John P. Snyder and Waldo R. Tobler Page 14

McCarthy, D.D., IERS Conventions 2000, Chapter 4 (2002).

Report of the IAU/IAG/COSPAR Working Group on Cartographic Coordinates and Rotational Elements of the Planets and Satellites: 1991 M. E. Davis et al.

gal_emname [0.2]

This routine returns the name of the requested ellipsoid model.

```
char *
gal_emname
(
  const int em,
  char *name,
  gal_status_t *status
 ) ;
```

On entry EM contains the identifier code of the required ellipsoid model. On return NAME contains the model name. The header file gal_frame_macros.h defines constants for the supported model identifiers. The routine returns a pointer to the string name. If the value of EM is unsupported by the routine then the error code GAL_INVALID_ID is set, and the routine returns NULL.

References:

Explanatory Supplement to the Astronomical Almanac Edited by P. Kenneth Seidelmann, 1992 Page 220

Map Projection Transformations by Qihe Yang, John P. Snyder and Waldo R. Tobler Page 14

McCarthy, D.D., IERS Conventions 2000, Chapter 4 (2002).

Report of the IAU/IAG/COSPAR Working Group on Cartographic Coordinates and Rotational Elements of the Planets and Satellites: 1991 M. E. Davis et al.

gal_emparams [0.2]

This routine returns the parameters of the requested ellipsoid model.

```
int
gal_emparams
(
  const int em,
  double *sma,
  double *inf,
  gal_status_t *status
) ;
```

On entry EM contains the identifier code of the requested ellipsoid model. The header file gal_frame_macros.h defines constants for the supported model identifiers. On return the variables are set as follows:

SMA	Semi-major axis (meters)
INF	Inverse flattening factor

If value of the ellipsoid identifer EM is unknown to the routine then the error code GAL_INVALID_ID is set and SMA and INF are set to zero.

References:

Explanatory Supplement to the Astronomical Almanac Edited by P. Kenneth Seidelmann, 1992 Page 220

Map Projection Transformations by Qihe Yang, John P. Snyder and Waldo R. Tobler Page 14

McCarthy, D.D., IERS Conventions 2000, Chapter 4 (2002).

Report of the IAU/IAG/COSPAR Working Group on Cartographic Coordinates and Rotational Elements of the Planets and Satellites: 1991 M. E. Davis et al.

gal_eo06a [0.1]

Equation of the origins, IAU 2006 precession and IAU 2000A nutation.

```
double
gal_eo06a
(
  double date1,
  double date2
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. The routine returns the equation of the origins in radians. The equation of the origins is the distance between the true equinox and the celestial intermediate origin and, equivalently, the difference between Earth rotation angle and Greenwich apparent sidereal time (ERA-GST). It comprises the precession (since J2000.0) in right ascension plus the equation of the equinoxes (including the small correction terms).

References:

Capitaine, N. & Wallace, P.T., 2006, Astronomy & Astrophysics 450, 855

Wallace, P.T. & Capitaine, N., 2006, Astronomy & Astrophysics 459, 981

gal_eop [0.6]

Read Earth Orientation Parameters from CSSI data file.

```
void
gal_eop
(
  char *filename,
  double date1,
  double date2,
  int buffer,
  double *xp,
  double *yp,
  double *dut1,
  double *lod,
  double *dpsi,
  double *deps,
  double *dx,
  double *dy,
  int *dat,
  int *dtype,
  gal_status_t *status
) ;
```

Given:

*FILENAME	CSSI EOP data file name
DATE1	Date part 1
DATE2	Date part 2
BUFFER	Buffering flag
	0 = do not buffer results
	1 = buffer results

Returned:

*XP	x coordinate of the pole (radians)
*YP	y coordinate of the pole (radians)
*DUT1	UT1 - UTC (seconds)
*LOD	Excess length of day (seconds)
*DPSI	Nutation (radians)
*DEPS	Nutation (radians)
*DX	? (radians)
*DY	? (radians)
*DAT	TAI - UTC (seconds)
*TYPE	Parameter type:
	0 = Invalid Coefficients (failure)
	1 = Observed Coefficients
	2 = Predicted Coefficients
*STATUS	Pointer to status structure

The date DATE1+DATE2 is a Julian date in standard SOFA two-piece format. The CSSI EOP data files can be downloaded from here:

http://celestrak.com/SpaceData/

The CSSI file format and data details are documented here:

http://celestrak.com/SpaceData/EOP-format.asp

XP and YP are the coordinates (in radians) of the Celestial Intermediate Pole with respect to the International Terrestrial Reference System (see IERS Conventions 2003), measured along the meridians to 0 and 90 deg west respectively. The nutation components (luni-solar + planetary, IAU 2000A) in longitude and obliquity are in radians and with respect to the equinox and ecliptic of date. If the date is outside of the date range of the data file then the error code GAL_OUTSIDE_DATE_RANGE is set. If any errors occur then the applicable error codes are set and all return values are set to zero. If the BUFFER parameter is set to 1 then the routine buffers the return results. If the date requested is the same as the previous call, then the buffered results are returned and the file is not accessed. If the BUFFER parameter is set to 0 then the file is accessed on each call. Using the buffered results improves the execution speed, however the user should use caution if multiple data files are used, or if the file contents are changed during program execution.

References:

Vallado, David A. and T.S. Kelso, "Using EOP and Solar Weather Data for Satellite Operations," presented at the 15th AIAA/AAS Astrodynamics Specialist Conference, Lake Tahoe, CA, 2005 August 7–11. http://celestrak.com/publications/AAS/05-406/

gal_eors [0.1]

Equation of the origins, given the classical NPB matrix and the quantity s.

```
double
gal_eors
(
  double rnpb[3][3],
  double s
) ;
```

On entry RNPB contains the classical nutation x precession x bias matrix, and S the quantity s (the CIO locator). The routine returns the equation of the origins in radians. The equation of the origins is the distance between the true equinox and the celestial intermediate origin and, equivalently, the difference between Earth rotation angle and Greenwich apparent sidereal time (ERA-GST). It comprises the precession (since J2000.0) in right ascension plus the equation of the equinoxes (including the small correction terms). The algorithm is from Wallace & Capitaine (2006).

References:

Capitaine, N. & Wallace, P.T., 2006, Astronomy & Astrophysics 450, 855

Wallace, P. & Capitaine, N., 2006, Astronomy & Astrophysics (submitted)

gal_eqeq94 [0.1]

Equation of the equinoxes, IAU 1994 model.

```
double
gal_eqeq94
(
  double date1,
  double date2
) ;
```

On entry DATE1 and DATE2 contain the Barycentric Dynamical Time (TDB) Julian Date in standard SOFA two-piece format. The routine returns the equation of the equinoxes. The result, which is in radians, operates in the following sense:

> Greenwich Apparent Sidereal Time = Greenwich Mean Sidereal Time + equation of the equinoxes

References:

IAU Resolution C7, Recommendation 3 (1994)

Capitaine, N. & Gontier, A.-M., Astronomy & Astrophysics, 275, 645-650 (1993)

gal_era00 [0.1]

Earth rotation angle (IAU 2000 model).

```
double
gal_era00
(
  double dj1,
  double dj2
) ;
```

On entry DJ1 and DJ2 contain the Universal Time (UT1) Julian Date in standard SOFA two-piece format. The routine returns the Earth rotation angle (radians), in the range 0 to 2π. The date & time method is best matched to the algorithm used: maximum accuracy (or, at least, minimum noise) is delivered when the DJ1 argument is for 0hrs UT1 on the day in question and the DJ2 argument lies in the range 0 to 1, or vice versa. The algorithm is adapted from Expression 22 of Capitaine et al. 2000. The time argument has been expressed in days directly, and, to retain precision, integer contributions have been eliminated. The same formulation is given in IERS Conventions (2003), Chap. 5, Eq. 14.

References:

Capitaine N., Guinot B. and McCarthy D.D, 2000, Astronomy & Astrophysics, 355, 398-405.

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

gal_fad03 [0.1]

Fundamental argument, IERS Conventions (2003): mean elongation of the Moon from the Sun.

```
double
gal_fad03
(
  double t
) ;
```

On entry T contains the Barycentric Dynamical Time (TDB) date in Julian centuries since J2000. The routine returns D in radians. Though T is strictly Barycentric Dynamical Time (TDB), it is usually more convenient to use Terrestrial Time (TT), which makes no significant difference. The expression used is as adopted in IERS Conventions (2003) and is from Simon et al. (1994).

References:

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

Simon, J.-L., Bretagnon, P., Chapront, J., Chapront-Touze, M., Francou, G., Laskar, J. 1994, Astronomy & Astrophysics 282, 663-683

gal_fae03 [0.1]

Fundamental argument, IERS Conventions (2003): mean longitude of Earth.

```
double
gal_fae03
(
  double t
) ;
```

On entry T contains the Barycentric Dynamical Time (TDB) date in Julian centuries since J2000. The routine returns the mean longitude of Earth in radians. Though T is strictly Barycentric Dynamical Time (TDB), it is usually more convenient to use Terrestrial Time (TT), which makes no significant difference. The expression used is as adopted in IERS Conventions (2003) and comes from Souchay et al. (1999) after Simon et al. (1994).

References:

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

Simon, J.-L., Bretagnon, P., Chapront, J., Chapront-Touze, M., Francou, G., Laskar, J. 1994, Astronomy & Astrophysics 282, 663-683

Souchay, J., Loysel, B., Kinoshita, H., Folgueira, M. 1999, Astronomy & Astrophysics Supplement Series 135, 111

gal_faf03 [0.1]

Fundamental argument, IERS Conventions (2003): mean longitude of the Moon minus mean longitude of the ascending node.

```
double
gal_faf03
(
  double t
) ;
```

On entry T contains the Barycentric Dynamical Time (TDB) date in Julian centuries since J2000. The routine returns the mean longitude of the Moon in radians. Though T is strictly Barycentric Dynamical Time (TDB), it is usually more convenient to use Terrestrial Time (TT), which makes no significant difference. The expression used is as adopted in IERS Conventions (2003) and is from Simon et al. (1994).

References:

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

Simon, J.-L., Bretagnon, P., Chapront, J., Chapront-Touze, M., Francou, G., Laskar, J. 1994, Astronomy & Astrophysics 282, 663-683

gal_faju03 [0.1]

Fundamental argument, IERS Conventions (2003): mean longitude of Jupiter.

```
double
gal_faju03
(
  double t
) ;
```

On entry T contains the Barycentric Dynamical Time (TDB) date in Julian centuries since J2000. The routine returns the mean longitude of Jupiter in radians. Though T is strictly Barycentric Dynamical Time (TDB), it is usually more convenient to use Terrestrial Time (TT), which makes no significant difference. The expression used is as adopted in IERS Conventions (2003) and comes from Souchay et al. (1999) after Simon et al. (1994).

References:

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

Simon, J.-L., Bretagnon, P., Chapront, J., Chapront-Touze, M., Francou, G., Laskar, J. 1994, Astronomy & Astrophysics 282, 663-683

Souchay, J., Loysel, B., Kinoshita, H., Folgueira, M. 1999, Astronomy & Astrophysics Supplement Series 135, 111

gal_fal03 [0.1]

Fundamental argument, IERS Conventions (2003): mean anomaly of the Moon.

```
double
gal_fal03
(
  double t
) ;
```

On entry T contains the Barycentric Dynamical Time (TDB) date in Julian centuries since J2000. The routine returns L in radians. Though T is strictly Barycentric Dynamical Time (TDB), it is usually more convenient to use Terrestrial Time (TT), which makes no significant difference. The expression used is as adopted in IERS Conventions (2003) and is from Simon et al. (1994).

References:

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

Simon, J.-L., Bretagnon, P., Chapront, J., Chapront-Touze, M., Francou, G., Laskar, J. 1994, Astronomy & Astrophysics 282, 663-683

gal_falp03 [0.1]

Fundamental argument, IERS Conventions (2003): mean anomaly of the Sun.

```
double
gal_falp03
(
  double t
) ;
```

On entry T contains the Barycentric Dynamical Time (TDB) date in Julian centuries since J2000. The routine returns L' in radians. Though T is strictly Barycentric Dynamical Time (TDB), it is usually more convenient to use Terrestrial Time (TT), which makes no significant difference. The expression used is as adopted in IERS Conventions (2003) and is from Simon et al. (1994).

References:

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

Simon, J.-L., Bretagnon, P., Chapront, J., Chapront-Touze, M., Francou, G., Laskar, J. 1994, Astronomy & Astrophysics 282, 663-683

gal_fama03 [0.1]

Fundamental argument, IERS Conventions (2003): mean longitude of Mars.

```
double
gal_fama03
(
  double t
) ;
```

On entry T contains the Barycentric Dynamical Time (TDB) date in Julian centuries since J2000. The routine returns the mean longitude of Mars in radians. Though T is strictly Barycentric Dynamical Time (TDB), it is usually more convenient to use Terrestrial Time (TT), which makes no significant difference. The expression used is as adopted in IERS Conventions (2003) and comes from Souchay et al. (1999) after Simon et al. (1994).

References:

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

Simon, J.-L., Bretagnon, P., Chapront, J., Chapront-Touze, M., Francou, G., Laskar, J. 1994, Astronomy & Astrophysics 282, 663-683

Souchay, J., Loysel, B., Kinoshita, H., Folgueira, M. 1999, Astronomy & Astrophysics Supplement Series 135, 111

gal_fame03 [0.1]

Fundamental argument, IERS Conventions (2003): mean longitude of Mercury.

```
double
gal_fame03
(
  double t
) ;
```

On entry T contains the Barycentric Dynamical Time (TDB) date in Julian centuries since J2000. The routine returns the mean longitude of Mercury in radians. Though T is strictly Barycentric Dynamical Time (TDB), it is usually more convenient to use Terrestrial Time (TT), which makes no significant difference. The expression used is as adopted in IERS Conventions (2003) and comes from Souchay et al. (1999) after Simon et al. (1994).

References:

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

Simon, J.-L., Bretagnon, P., Chapront, J., Chapront-Touze, M., Francou, G., Laskar, J. 1994, Astronomy & Astrophysics 282, 663-683

Souchay, J., Loysel, B., Kinoshita, H., Folgueira, M. 1999, Astronomy & Astrophysics Supplement Series 135, 111

gal_fane03 [0.1]

Fundamental argument, IERS Conventions (2003): mean longitude of Neptune.

```
double
gal_fane03
(
  double t
) ;
```

On entry T contains the Barycentric Dynamical Time (TDB) date in Julian centuries since J2000. The routine returns the mean longitude of Neptune in radians. Though T is strictly Barycentric Dynamical Time (TDB), it is usually more convenient to use Terrestrial Time (TT), which makes no significant difference. The expression used is as adopted in IERS Conventions (2003) and is adapted from Simon et al. (1994).

References:

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

Simon, J.-L., Bretagnon, P., Chapront, J., Chapront-Touze, M., Francou, G., Laskar, J. 1994, Astronomy & Astrophysics 282, 663-683

gal_faom03 [0.1]

Fundamental argument, IERS Conventions (2003): mean longitude of the Moon's ascending node.

```
double
gal_faom03
(
  double t
) ;
```

On entry T contains the Barycentric Dynamical Time (TDB) date in Julian centuries since J2000. The routine returns Omega in radians. Though T is strictly Barycentric Dynamical Time (TDB), it is usually more convenient to use Terrestrial Time (TT), which makes no significant difference. The expression used is as adopted in IERS Conventions (2003) and is from Simon et al. (1994).

References:

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

Simon, J.-L., Bretagnon, P., Chapront, J., Chapront-Touze, M., Francou, G., Laskar, J. 1994, Astronomy & Astrophysics 282, 663-683

gal_fapa03 [0.1]

Fundamental argument, IERS Conventions (2003): general accumulated precession in longitude.

```
double
gal_fapa03
(
  double t
) ;
```

On entry T contains the Barycentric Dynamical Time (TDB) date in Julian centuries since J2000. The routine returns the general precession in longitude in radians. Though

T is strictly Barycentric Dynamical Time (TDB), it is usually more convenient to use Terrestrial Time (TT), which makes no significant difference. The expression used is as adopted in IERS Conventions (2003). It is taken from Kinoshita & Souchay (1990) and comes originally from Lieske et al. (1977).

References:

Kinoshita, H. and Souchay J. 1990, Celestial Mechanics and Dynamical Astronomy 48, 187

Lieske, J.H., Lederle, T., Fricke, W. & Morando, B. 1977, Astronomy & Astrophysics 58, 1-16

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

gal_fasa03 [0.1]

Fundamental argument, IERS Conventions (2003): mean longitude of Saturn.

```
double
gal_fasa03
(
  double t
) ;
```

On entry T contains the Barycentric Dynamical Time (TDB) date in Julian centuries since J2000. The routine returns the mean longitude of Saturn in radians. Though T is strictly Barycentric Dynamical Time (TDB), it is usually more convenient to use Terrestrial Time (TT), which makes no significant difference. The expression used is as adopted in IERS Conventions (2003) and comes from Souchay et al. (1999) after Simon et al. (1994).

References:

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

Simon, J.-L., Bretagnon, P., Chapront, J., Chapront-Touze, M., Francou, G., Laskar, J. 1994, Astronomy & Astrophysics 282, 663-683

Souchay, J., Loysel, B., Kinoshita, H., Folgueira, M. 1999, Astronomy & Astrophysics Supplement Series 135, 111

gal_faur03 [0.1]

Fundamental argument, IERS Conventions (2003): mean longitude of Uranus.

```
double
gal_faur03
(
  double t
) ;
```

On entry T contains the Barycentric Dynamical Time (TDB) date in Julian centuries since J2000. The routine returns the mean longitude of Uranus in radians. Though T is strictly Barycentric Dynamical Time (TDB), it is usually more convenient to use Terrestrial Time (TT), which makes no significant difference. The expression used is as adopted in IERS Conventions (2003) and is adapted from Simon et al. (1994).

References:

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

Simon, J.-L., Bretagnon, P., Chapront, J., Chapront-Touze, M., Francou, G., Laskar, J. 1994, Astronomy & Astrophysics 282, 663-683

gal_fave03 [0.1]

Fundamental argument, IERS Conventions (2003): mean longitude of Venus.

```
double
gal_fave03
(
  double t
) ;
```

On entry T contains the Barycentric Dynamical Time (TDB) date in Julian centuries since J2000. The routine returns the mean longitude of Venus in radians. Though T is strictly Barycentric Dynamical Time (TDB), it is usually more convenient to use Terrestrial Time (TT), which makes no significant difference. The expression used is as adopted in IERS Conventions (2003) and comes from Souchay et al. (1999) after Simon et al. (1994).

References:

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

Simon, J.-L., Bretagnon, P., Chapront, J., Chapront-Touze, M., Francou, G., Laskar, J. 1994, Astronomy & Astrophysics 282, 663-683

Souchay, J., Loysel, B., Kinoshita, H., Folgueira, M. 1999, Astronomy & Astrophysics Supplement Series 135, 111

gal_fw2m [0.1]

Form rotation matrix given the Fukushima-Williams angles.

```
void
gal_fw2m
(
  double gamb,
  double phib,
  double psi,
  double eps,
  double r[3][3]
) ;
```

On entry GAMB contains the F-W angle gamma_bar, PHIB the F-W angle phi_bar, PSI the F-W angle psi, and EPS the F-W angle epsilon. All angles are in radians. On return R contains the rotation matrix.

Naming the following points:

e	J2000 ecliptic pole
p	GCRS pole
E	ecliptic pole of date
P	CIP

the four Fukushima-Williams angles are as follows:

gamb	= gamma	= epE
phib	= phi	= pE
psi	= psi	= pEP
eps	= epsilon	= EP

The matrix representing the combined effects of frame bias, precession and nutation is:

NxPxB = R_1(-EPS).R_3(-PSI).R_1(PHIB).R_3(GAMB)

Three different matrices can be constructed, depending on the supplied angles:

To obtain the nutation x precession x frame bias matrix, generate the four precession angles, generate the nutation components and add them to the psi_bar and epsilon_A angles, and call this routine. To obtain the precession x frame bias matrix, generate the four precession angles and call this routine. To obtain the frame bias matrix, generate the four precession angles for date J2000.0 and call this routine. The nutation-only and precession-only matrices can if necessary be obtained by combining these three appropriately.

References:

Hilton, J. et al., 2006, Celestial Mechanics and Dynamical Astronomy 94, 351

gal_fw2xy [0.1]

CIP X,Y given Fukushima-Williams bias-precession-nutation angles.

```
void
gal_fw2xy
(
  double gamb,
  double phib,
  double psi,
  double eps,
  double *x,
  double *y
) ;
```

On entry GAMB contains the F-W angle gamma_bar, PHIB the F-W angle phi_bar, PSI the F-W angle psi, and EPS the F-W angle epsilon. All angles are in radians. On return X and Y contain the CIP X and Y in radians.

Naming the following points:

e	J2000 ecliptic pole,
p	GCRS pole,
E	ecliptic pole of date,
P	CIP,

the four Fukushima-Williams angles are as follows:

gamb	= gamma	= epE
phib	= phi	= pE
psi	= psi	= pEP
eps	= epsilon	= EP

The matrix representing the combined effects of frame bias, precession and nutation is:

NxPxB = R_1(-epsa).R_3(-psi).R_1(phib).R_3(gamb)

X and Y are elements [0][2] and [1][2] of the matrix.

References:

Hilton, J. et al., 2006, Celestial Mechanics and Dynamical Astronomy 94, 351

gal_gc2gd [0.6]

Transform geocentric coordinates to geodetic using the specified reference ellipsoid.

```
void
gal_gc2gd (
  int    n,
  double xyz[3],
  double *elong,
  double *phi,
  double *height,
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

N	Ellipsoid identifier
XYZ	Geocentric p-vector

On return the variables are set as follows:

ELONG	Longitude (radians, east positive)
PHI	Latitude (geodetic, radians)
HEIGHT	Height above ellipsoid (geodetic)

The identifier N is a number that specifies the choice of reference ellipsoid. The header file "gal_frame_macros.h" defines the constants for the identifiers. The geocentric vector (XYZ, given) and HEIGHT (height, returned) are in meters. An error code of GAL_INVALID_ID means that the identifier N is illegal. In all error cases, PHI and HEIGHT are both set to -1e9. The inverse transformation is performed in the function gal_gd2gc.

gal_gc2gde [0.6]

Transform geocentric coordinates to geodetic for a reference ellipsoid of specified form.

```
void
```

```
gal_gc2gde (
  double a,
  double f,
  double xyz[3],
  double *elong,
  double *phi,
  double *height,
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

A	Equatorial radius
F	Flattening
XYZ	Geocentric p-vector

On return the variables are set as follows:

ELONG	Longitude (radians, east positive)
PHI	Latitude (geodetic, radians)
HEIGHT	Height above ellipsoid (geodetic)

This function is based on the GCONV2H Fortran subroutine by Toshio Fukushima (see reference). The equatorial radius, A, can be in any units, but meters is the conventional choice. The flattening, F, is (for the Earth) a value around 0.00335, i.e. around 1/298. The equatorial radius, A, and the geocentric vector, XYZ, must be given in the same units, and determine the units of the returned height, HEIGHT. If an error occurs, ELONG, PHI and HEIGHT are unchanged. The inverse transformation is performed in the function gal_gd2gce. The transformation for a standard ellipsoid (such as WGS84) can more conveniently be performed by calling gal_gc2gd, which uses a numerical code to identify the required A and F values. The error codes GAL_INVALID_SMA and GAL_INVALID_FLATTENING are set when A and F respectively have invalid values.

Reference:

Fukushima, T., "Transformation from Cartesian to geodetic coordinates accelerated by Halley's method", J.Geodesy (2006) 79: 689-693

gal_gd2gc [0.6]

Transform geodetic coordinates to geocentric using the specified reference ellipsoid.

```
void
gal_gd2gc (
  int    n,
  double elong,
  double phi,
  double height,
  double xyz[3],
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

N	Ellipsoid identifier
ELONG	Longitude (radians, east positive)
PHI	Latitude (geodetic, radians)
HEIGHT	Height above ellipsoid (geodetic)

On return the variables are set as follows:

XYZ	Geocentric vector

The identifier N is a number that specifies the choice of reference ellipsoid. The header file "gal_frame_macros.h" defines the constants for the identifiers. The geocentric vector (XYZ, given) and height (HEIGHT, returned) are in meters. No validation is performed on the arguments ELONG, PHI and HEIGHT. An error status GAL_INVALID_ID means that the identifier N is illegal. Error GAL_ARITHMETIC_EXCEPTION protects against cases that would lead to arithmetic exceptions. In all error cases, XYZ is set to zeros. The inverse transformation is performed in the function gal_gc2gd.

gal_gd2gce [0.6]

Transform geodetic coordinates to geocentric for a reference ellipsoid of specified form.

```
void
gal_gd2gce (
  double a,
  double f,
  double elong,
  double phi,
  double height,
  double xyz[3],
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

A	Equatorial radius
F	Flattening
ELONG	Longitude (radians, east positive)
PHI	Latitude (geodetic, radians)
HEIGHT	Height above ellipsoid (geodetic)

On return the variables are set as follows:

XYZ	Geocentric vector

The equatorial radius, A, can be in any units, but meters is the conventional choice. The flattening, F, is (for the Earth) a value around 0.00335, i.e. around 1/298. The equatorial radius, A, and the height, HEIGHT, must be given in the same units, and determine the units of the returned geocentric vector, XYZ. No validation is performed on individual arguments. The error status GAL_ARITHMETIC_EXCEPTION protects against (unrealistic) cases that would lead to arithmetic exceptions. If an error occurs, XYZ is unchanged. The inverse transformation is performed in the function gal_gc2gde. The transformation for a standard ellipsoid (such as WGS84) can more conveniently be performed by calling gal_gd2gc, which uses a numerical code to identify the required A and F values.

References:

Green, R.M., Spherical Astronomy, Cambridge University Press, (1985) Section 4.5, p96.

Explanatory Supplement to the Astronomical Almanac, P. Kenneth Seidelmann (ed), University Science Books (1992), Section 4.22, p202.

gal_gmst00 [0.1]

Greenwich Mean Sidereal Time (model consistent with IAU 2000 resolutions).

```
double
gal_gmst00
(
  double uta,
  double utb,
  double tta,
  double ttb
) ;
```

On entry UTA and UTB contain the Universal Time (UT1) Julian Date, and TTA and TTB contain the Terrestrial Time (TT) Julian Date. Both dates in standard SOFA two-piece format. The routine returns the Greenwich Mean Sidereal Time in radians, in the range 0 to 2π. Both UT1 and TT are required, UT1 to predict the Earth rotation and TT to predict the effects of precession. If UT1 is used for both purposes, errors of order 100 microarcseconds result. This GMST is compatible with the IAU 2000 resolutions and must be used only in conjunction with other IAU 2000 compatible components such as precession-nutation and equation of the equinoxes. The algorithm is from Capitaine et al. (2003) and IERS Conventions 2003.

References:

Capitaine, N., Wallace, P.T. and McCarthy, D.D., "Expressions to implement the IAU 2000 definition of UT1", Astronomy & Astrophysics, 406, 1135-1149 (2003)

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

gal_gmst06 [0.1]

Greenwich mean sidereal time (consistent with IAU 2006 precession).

```
double
gal_gmst06
(
  double uta,
  double utb,
  double tta,
  double ttb
) ;
```

On entry UTA and UTB contain the Universal Time (UT1) Julian Date, and TTA and TTB contain the Terrestrial Time (TT) Julian Date. Both dates in standard SOFA two-piece format. The routine returns the Greenwich Mean Sidereal Time in radians, in the range 0 to 2π. Both UT1 and TT are required, UT1 to predict the Earth rotation and TT to predict the effects of precession. If UT1 is used for both purposes, errors of order 100 microarcseconds result. This GMST is compatible with the IAU 2006 precession and must not be used with other precession models.

References:

Capitaine, N., Wallace, P.T. & Chapront, J., 2005, Astronomy & Astrophysics 432, 355

gal_gmst82 [0.1]

Universal Time to Greenwich Mean Sidereal Time (IAU 1982 model).

```
double
gal_gmst82
(
  double dj1,
  double dj2
) ;
```

On entry DJ1 and DJ2 contain the Universal Time (UT1) Julian Date in standard SOFA two-piece format. The routine returns the Greenwich Mean Sidereal Time (GMST) in radians, in the range 0 to 2π. The algorithm is based on the IAU 1982 expression. This is always described as giving the GMST at 0 hours UT1. In fact, it gives the difference between the GMST and the UT, the steady 4-minutes-per-day drawing-ahead of ST with respect to UT. When whole days are ignored, the expression happens to equal the GMST at 0 hours UT1 each day. In this routine, the entire UT1 (the sum of the two arguments DJ1 and DJ2) is used directly as the argument for the standard formula, the constant term of which is adjusted by 12 hours to take account of the noon phasing of Julian Date. The UT1 is then added, but omitting whole days to conserve accuracy.

References:

Transactions of the International Astronomical Union, XVIII B, 67 (1983).

Aoki et al., Astronomy & Astrophysics 105, 359-361 (1982).

gal_gst00a [0.1]

Greenwich Apparent Sidereal Time (consistent with IAU 2000 resolutions).

```
double
gal_gst00a
(
  double uta,
  double utb,
  double tta,
  double ttb
) ;
```

On entry UTA and UTB contain the Universal Time (UT1) Julian Date, TTA and TTB the Terrestrial Time (TT) Julian Date. The routine return the Greenwich Apparent Sidereal Time (GAST) in radians, in the range 0 to 2π. Both UT1 and TT are required, UT1 to predict the Earth rotation and TT to predict the effects of precession-nutation. If UT1 is used for both purposes, errors of order 100 microarcseconds result. This GAST is compatible with the IAU 2000 resolutions and must be used only in conjunction with other IAU 2000 compatible components such as precession-nutation. The algorithm is from Capitaine et al. (2003) and IERS Conventions 2003.

References:

Capitaine, N., Wallace, P.T. and McCarthy, D.D., "Expressions to implement the IAU 2000 definition of UT1", Astronomy & Astrophysics, 406, 1135-1149 (2003)

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

gal_gst00b [0.1]

Greenwich Apparent Sidereal Time (consistent with IAU 2000 resolutions but using the truncated nutation model IAU 2000B).

```
double
gal_gst00b
(
  double uta,
  double utb
) ;
```

On entry UTA and UTB contain the Universal time (UT1) Julian Date in standard SOFA two-piece format. The routine returns the Greenwich Apparent Sidereal Time (GAST) in radians, in the range 0 to 2π. The result is compatible with the IAU 2000 resolutions, except that accuracy has been compromised for the sake of speed and convenience in two respects: (1) UT is used instead of TDB (or TT) to compute the precession component of Greenwich Mean Sidereal Time (GMST) and the equation of the equinoxes. This results in errors of order 0.1 mas at present. (2) The IAU 2000B abridged nutation model (McCarthy & Luzum, 2001) is used, introducing errors of up to 1 mas. This GAST is compatible with the IAU 2000 resolutions and must be used only in conjunction with other IAU 2000 compatible components such as precession-nutation. The algorithm is from Capitaine et al. (2003) and IERS Conventions 2003.

References:

Capitaine, N., Wallace, P.T. and McCarthy, D.D., "Expressions to implement the IAU 2000 definition of UT1", Astronomy & Astrophysics, 406, 1135-1149 (2003)

McCarthy, D.D. & Luzum, B.J., "An abridged model of the precession-nutation of the celestial pole", Celestial Mechanics & Dynamical Astronomy, 85, 37-49 (2003)

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

gal_gst06 [0.1]

Greenwich apparent sidereal time, IAU 2006, given the NPB matrix.

```
double
gal_gst06
(
  double uta,
  double utb,
  double tta,
  double ttb,
  double rnpb[3][3]
) ;
```

On entry UTA and UTB contain the Universal Time (UT1) Julian Date, TTA and TTB contain the Terrestrial Time (TT) Julian Date, RNPB contains the nutation x precession x bias matrix. The routine returns the Greenwich Apparent Sidereal Time (GAST) in radians, in the range 0 to 2π. Both UT1 and TT are required, UT1 to predict the Earth rotation and TT to predict the effects of precession-nutation. If UT1 is used for both purposes, errors of order 100 microarcseconds result. Although the routine uses the IAU 2006 series for s+XY/2, it is otherwise independent of the precession-nutation model and can in practice be used with any equinox-based NPB matrix.

References:

Wallace, P.T. & Capitaine, N., 2006, Astronomy & Astrophysics 459, 981

gal_gst06a [0.1]

Greenwich Apparent Sidereal Time (consistent with IAU 2000 and 2006 resolutions).

```
double
gal_gst06a
(
  double uta,
  double utb,
  double tta,
  double ttb
) ;
```

On entry UTA and UTB contain the Universal Time (UT1) Julian Date, TTA and TTB contain the Terrestrial Time (TT) Julian Date. The routine returns the Greenwich Apparent Sidereal Time (GAST) in radians, in the range 0 to 2π. All dates are in standard SOFA two-piece format. Both UT1 and TT are required, UT1 to predict the Earth rotation and TT to predict the effects of precession-nutation. If UT1 is used for both purposes, errors of order 100 microarcseconds result. This GAST is compatible with the IAU 2000/2006 resolutions and must be used only in conjunction with IAU 2006 precession and IAU 2000A nutation.

References:

Wallace, P.T. & Capitaine, N., 2006, Astronomy & Astrophysics 459, 981

gal_gst94 [0.1]

Greenwich Apparent Sidereal Time (consistent with IAU 1982/94 resolutions).

```
double
gal_gst94
(
  double uta,
  double utb
) ;
```

On entry UTA and UTB contain the Universal time (UT1) Julian Date in standard SOFA two-piece format. The routine returns the Greenwich Apparent Sidereal Time (GAST) in radians, in the range 0 to 2π. The result is compatible with the IAU 1982 and 1994 resolutions, except that accuracy has been compromised for the sake of convenience in that Universal Time (UT1) is used instead of Barycentric Dynamical Time (TDB) (or Terrestrial Time (TT)) to compute the equation of the equinoxes. This GAST must be used only in conjunction with contemporaneous IAU standards such as 1976 precession, 1980 obliquity and 1982 nutation. It is not compatible with the IAU 2000 resolutions.

References:

Explanatory Supplement to the Astronomical Almanac, P. Kenneth Seidelmann (ed.), University Science Books (1992)

IAU Resolution C7, Recommendation 3 (1994)

gal_i2tpv00 [0.2]

This routine converts a position & velocity vector in the Celestial Intermediate Reference System (CIRS) to the International Terrestrial Reference Frame (ITRF) (IAU 2000 Resolutions).

```
void
gal_i2tpv00
(
  double cirs[2][3],
  double tta,
  double ttb,
  double ut1a,
  double ut1b,
  double lod,
  double xp,
  double yp,
  double itrf[2][3]
) ;
```

On entry the parameters are set as follows:

CIRS	CIRS position & velocity vector (meters, meters per second)
TTA	Date part 1 (TT)
TTB	Date part 2 (TT)
UT1A	UT1 date part 1
UT1B	UT1 date part 2
LOD	Excess length of day (seconds)
XP	X coordinate of the pole (radians)
YP	Y coordinate of the pole (radians)

TTA and TTB contain a Terrestrial Time (TT) Julian Date, and UT1A and UT1B a Universal Time (UT1) Julian Date. Both dates are in standard SOFA two-piece format. On return ITRF contains the ITRF position & velocity vector (meters, meters per second). XP and YP are the "coordinates of the pole", in seconds, which position the Celestial Intermediate Pole in the International Terrestrial Reference System (see IERS Conventions 2003). In a geocentric right-handed triad u, v, w, where the w-axis points at the north geographic pole, the v-axis points towards the origin of longitudes and the u axis completes the system, XP = +u and YP = -v.

References:

SOFA Tools for Earth Attitude IAU Standards for Fundamental Astronomy Review Board 2007

http://www.iau-sofa.rl.ac.uk

Fundamentals of Astrodynamics and Applications, Vallado, David A. Second Edition 2004, Pages 217-219

gal_ijk2pqw [0.4]

This routine transforms a pv-vector from the IJK frame to the to PQW frame.

```
void
gal_ijk2pqw
(
  double ijk[2][3],
  double raan,
  double argp,
  double inc,
  double pqw[2][3]
) ;
```

On entry the parameters are set as follows:

IJK	pv-vector in IJK frame (meters, meters per second)
RAAN	Longitude of the ascending mode (radians)
ARGP	Argument of pericenter (radians)
INC	Inclination (radians)

On return PQW contains the pv-vector in the PQW frame.

References:

Satellite Orbits, Oliver Montenbruck, Eberhard Gill, Springer 2005, Chapter 2

gal_illum [0.5]

This routine calculates the degree of the Sun's occultation by a body of a satellite.

```
int
gal_illum
(
  double rsu,
  double rb,
  double psu[3],
  double pb[3],
  double ps[3],
  double *illum
) ;
```

On entry the parameters are set as follows:

RSU	Radius of the Sun's disc (meters)
RB	Radius of the occulting body (meters)
PSU	Sun's position vector (meters)
PB	Position vector of occulting body (meters)
PS	Position vector of satellite (meters)

On return the ILLUM is set to the degree of occultation (0 to 1).

0 = the satellite is in umbra
1 = the satellite is in sunlight
0 < illum < 1 the satellite is in penumbra

The routine returns one of the following status codes:

GAL_NO_OCCULTATION

GAL_PARTIAL_OCCULTATION
GAL_TOTAL_OCCULTATION

References:

Satellite Orbits, Oliver Montenbruck, Eberhard Gill, Springer 2005, Pages 81-83

gal_me2bf [0.5]

This routine transforms a pv-vector from the planet mean equator reference frame to the planet body fixed reference frame.

```
void
gal_me2bf
(
  double merf[2][3],
  double w,
  double omega,
  double bfrf[2][3]
) ;
```

On entry the parameters are set as follows:

MERF	pv-vector in planet mean equator frame (meters, meters per second)
W	Prime meridian angle (radians)
OMEGA	Planet rotation rate (radians per second)

On return BFRF contains the pv-vector in planet body fixed frame.

References:

Mars Pathfinder Project Planetary Constants and Models, Jet Propulsion Laboratory, December 1995, Chapters 2, 4, and 5

gal_me2c [0.5]

This routine transforms a pv-vector from planet mean equator reference frame reference frame to an inertial frame aligned with the International Celestial Reference frame (ICRF).

```
void
gal_me2c
(
  double merf[2][3],
  double alpha,
  double delta,
  double icrf[2][3]
) ;
```

On entry the parameters are set as follows:

MERF	pv-vector in planet mean equator frame (meters, meters per second)
ALPHA	Right ascension of planet spin axis (radians)
DELTA	Declination of planet spin axis (radians)

On return ICRF contains the pv-vector in inertial frame aligned to ICRF.

References:

Mars Pathfinder Project Planetary Constants and Models, Jet Propulsion Laboratory, December 1995, Chapters 2, 4, and 5

gal_morotel00 [0.5]

This routine returns the rotational elements for the Earth's Moon (IAU/IAG 2000).

```
void
gal_morotel00
(
  double tdb1,
  double tdb2,
  double *alpha,
  double *delta,
  double *w,
  double *omega
) ;
```

On entry TDB1 and TDB2 contain the Barycentric Dynamical Time (TDB) Julian Date in standard two-piece SOFA format. On return the variables are set as follows:

ALPHA	Right ascension of Moon spin axis (radians)
DELTA	Declination of Moon spin axis (radians)
W	Prime meridian angle (radians)
OMEGA	Rotation Rate (radians per second)

Planetary coordinate systems are defined relative to their mean axis of rotation and various definitions of longitude depending on the body. The longitude systems of most of those bodies with observable rigid surfaces have been defined by references to a surface feature such as a crater. Approximate expressions for these rotational elements with respect to the J2000 inertial coordinate system have been derived. The

International Celestial Reference Frame (ICRF) is the reference coordinate frame of epoch 2000 which is January 1.5 TDB. The variable quantities are expressed in units of days or Julian centuries of 36525 days. The north pole is that pole of rotation that lies on the north side of the invariable plane of the solar system. The direction of the north pole is specified by the value of its right ascension ra and declination dec, whereas the location of the prime meridian is specified by the angle that is measured along the planet's equator in an easterly direction with respect to the planet's north pole from the node Q (located at right ascension 90 degrees + RA) of the planet's equator on the standard equator to the point B where the prime meridian crosses the planet's equator. The right ascension of the point Q is 90 degrees + RA and the inclination of the planet's equator to the standard equator is 90 degrees - DEC. Because the prime meridian is assumed to rotate uniformly with the planet, W accordingly varies linearly with time. In addition, RA, DEC, and W may vary with time due to a precession of the axis of rotation of the planet. If W increases with time, the planet has direct (or prograde) rotation and if W decreases with time, the rotation is said to the retrograde.

References:

Satellite Orbits, Oliver Montenbruck & Eberhard Gill, Springer 2005, Pages 167, 191

Explanatory Supplement to the Astronomical Almanac, P. Kenneth Seidelmann Ed. Page 51

Report of the IAU/IAG Working Group on Cartographic Coordinates and Rotational Elements of the Planets and Satellites: 2000, P. K. Seidelmann et al.

gal_morotel91 [0.5]

This routine returns the rotational elements for the Earth's Moon (IAU/IAG/COSPAR 1991).

```
void
gal_morotel91
(
  double tdb1,
  double tdb2,
  double *alpha,
  double *delta,
  double *w,
  double *omega
) ;
```

On entry TDB1 and TDB2 contain a Barycentric Dynamical Time (TDB) Julian Date in standard SOFA two-piece format. On return the variables are set as follows:

ALPHA	Right ascension of Moon spin axis (radians)
DELTA	Declination of Moon spin axis (radians)
W	Prime meridian angle (radians)
OMEGA	Rotation Rate (radians per second)

Planetary coordinate systems are defined relative to their mean axis of rotation and various definitions of longitude depending on the body. The longitude systems of most of those bodies with observable rigid surfaces have been defined by references to a surface feature such as a crater. Approximate expressions for these rotational elements with respect to the J2000 inertial coordinate system have been derived. The International Celestial Reference Frame (ICRF) is the reference coordinate frame of epoch 2000 which is January 1.5 TDB. The variable quantities are expressed in units of days or Julian centuries of 36525 days. The north pole is that pole of rotation that lies on the north side of the invariable plane of the solar system. The direction of the north pole is specified by the value of its right ascension ra and declination dec, whereas the location of the prime meridian is specified by the angle that is measured along the planet's equator in an easterly direction with respect to the planet's north pole from the node Q (located at right ascension 90 degrees + RA) of the planet's equator on the standard equator to the point B where the prime meridian crosses the planet's equator. The right ascension of the point Q is 90 degrees + RA and the inclination of the planet's equator to the standard equator is 90 degrees - DEC. Because the prime meridian is assumed to rotate uniformly with the planet, W accordingly varies linearly with time. In addition, ra, dec, and W may vary with time due to a precession of the axis of rotation of the planet. If W increases with time, the planet has direct (or prograde) rotation and if W decreases with time, the rotation is said to the retrograde.

References:

Satellite Orbits, Oliver Montenbruck & Eberhard Gill, Springer 2005, Pages 167, 191

Explanatory Supplement to the Astronomical Almanac, P. Kenneth Seidelmann Ed. Page 51

Report of the IAU/IAG/COSPAR Working Group on Cartographic Coordinates and Rotational Elements of the Planets and Satellites: 1991, M. E. Davies et al.

gal_num00a [0.1]

Form the matrix of nutation for a given date, IAU 2000A model.

```
void
gal_num00a
(
  double date1,
  double date2,
  double rmatn[3][3]
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return RMATN contains the nutation matrix. The matrix operates in the sense V(true) = RMATN * V(mean), where the p-vector V(true) is with respect to the true equatorial triad of date and the p-vector V(mean) is with respect to the mean equatorial triad of date. A faster, but slightly less accurate result (about 1 mas), can be obtained by using instead the gal_num00b routine.

References:

Explanatory Supplement to the Astronomical Almanac, P. Kenneth Seidelmann (ed.), University Science Books (1992), Section 3.222-3 (p114).

gal_num00b [0.1]

Form the matrix of nutation for a given date, IAU 2000B model.

```
void
gal_num00b
(
  double date1,
  double date2,
  double rmatn[3][3]
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return RMATN contains the nutation matrix. The matrix operates in the sense V(true) = RMATN * V(mean), where the p-vector V(true) is with respect to the true equatorial triad of date and the p-vector V(mean) is with respect to the mean equatorial triad of date. This routine is faster, but slightly less accurate (about 1 mas), than the gal_num00a routine.

References:

Explanatory Supplement to the Astronomical Almanac, P. Kenneth Seidelmann (ed.), University Science Books (1992), Section 3.222-3 (p114).

gal_num06a [0.1]

Form the matrix of nutation for a given date, IAU 2006/2000A model.

```
void
gal_num06a
```

```
(
  double date1,
  double date2,
  double rmatn[3][3]
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return RMATN contains the nutation matrix. The matrix operates in the sense V(true) = RMATN * V(mean), where the p-vector V(true) is with respect to the true equatorial triad of date and the p-vector V(mean) is with respect to the mean equatorial triad of date.

References:

Capitaine, N., Wallace, P.T. & Chapront, J., 2005, Astronomy & Astrophysics 432, 355

Wallace, P.T. & Capitaine, N., 2006, Astronomy & Astrophysics 459, 981

gal_numat [0.1]

Form the matrix of nutation.

```
void
gal_numat
(
  double epsa,
  double dpsi,
  double deps,
  double rmatn[3][3]
) ;
```

On entry EPSA contains the mean obliquity of date, DPSI and DEPS contain the nutation. On return RMATN contains the nutation matrix. The supplied mean obliquity EPSA, must be consistent with the precession-nutation models from which DPSI and DEPS were obtained. The caller is responsible for providing the nutation components; they are in longitude and obliquity, in radians and are with respect to the equinox and ecliptic of date. The matrix operates in the sense V(true) = RMATN * V(mean), where the p-vector V(true) is with respect to the true equatorial triad of date and the p-vector V(mean) is with respect to the mean equatorial triad of date.

References:

Explanatory Supplement to the Astronomical Almanac, P. Kenneth Seidelmann (ed.), University Science Books (1992), Section 3.222-3 (p114).

gal_nut00a [0.1]

Nutation, IAU 2000A model (MHB2000 luni-solar and planetary nutation with free core nutation omitted).

```
void
gal_nut00a
(
  double date1,
  double date2,
  double *dpsi,
  double *deps
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return DPSI and DEPS contain the nutation (luni-solar and planetary). The nutation components in longitude and obliquity are in radians and with respect to the equinox and ecliptic of date. The obliquity at J2000 is assumed to be the Lieske et al. (1977) value of 84381.448 arcsec. Both the luni-solar and planetary nutations are included. The latter are due to direct planetary nutations and the perturbations of the lunar and terrestrial orbits. The routine computes the MHB2000 nutation series with the associated corrections for planetary nutations. It is an implementation of the nutation part of the IAU 2000A precession-nutation model, formally adopted by the IAU General Assembly in 2000, namely MHB2000 (Mathews et al. 2002), but with the free core nutation (FCN) omitted. The full MHB2000 model also contains contributions to the nutations in longitude and obliquity due to the free-excitation of the free-core-nutation during the period 1979-2000.These FCN terms, which are time-dependent and unpredictable, are NOT included in this routine and, if required, must be independently computed. With the FCN corrections included, this routine delivers a pole which is at current epochs accurate to a few hundred microarcseconds. The omission of FCN introduces further errors of about that size. This routine provides classical nutation. The MHB2000 algorithm, from which it is adapted, deals also with (i) the offsets between the GCRS and mean poles and (ii) the adjustments in longitude and obliquity due to the changed precession rates. These additional functions, namely frame bias and precession adjustments, are supported by the routines gal_bi00 and gal_pr00. The MHB2000 algorithm also provides "total" nutations, comprising the arithmetic sum of the frame bias, precession adjustments, luni-solar nutation and planetary nutation. These total nutations can be used in combination with an existing IAU 1976 precession implementation, such as gal_pmat76, to deliver GCRS-to-true predictions of sub-mas accuracy at current epochs. However, there are three shortcomings in the MHB2000 model that must be taken into account if more accurate or definitive results are required (see Wallace 2002):

(i) The MHB2000 total nutations are simply arithmetic sums, yet in reality the various components are successive Euler rotations. This slight lack of rigor leads to cross terms that exceed 1 mas after a century. The rigorous procedure is to form the GCRS-to-true rotation matrix by applying the bias, precession and nutation in that order.

(ii) Although the precession adjustments are stated to be with respect to Lieske et al. (1977), the MHB2000 model does not specify which set of Euler angles are to be used and how the adjustments are to be applied. The most literal and straightforward procedure is to adopt the 4-rotation epsilon_0, psi_A, omega_A, xi_A option, and to add dpsipr to psi_A and depspr to both omega_A and eps_A.

(iii) The MHB2000 model predates the determination by Chapront et al. (2002) of a 14.6 mas displacement between the J2000 mean equinox and the origin of the ICRS frame. It should, however, be noted that neglecting this displacement when calculating star coordinates does not lead to a 14.6 mas change in right ascension, only a small second-order distortion in the pattern of the precession-nutation effect.

For these reasons, the routines do not generate the "total nutations" directly, though they can of course easily be generated by calling gal_bi00, gal_pr00 and this routine and adding the results.

References:

Chapront, J., Chapront-Touze, M. & Francou, G. 2002, Astronomy & Astrophysics 387, 700

Lieske, J.H., Lederle, T., Fricke, W. & Morando, B. 1977, Astronomy & Astrophysics 58, 1-16

Mathews, P.M., Herring, T.A., Buffet, B.A. 2002, Journal Geophysical Research 107, B4. The MHB_2000 code itself was obtained on 9th September 2002 from:

ftp://maia.usno.navy.mil/conv2000/chapter5/IAU2000A

Simon, J.-L., Bretagnon, P., Chapront, J., Chapront-Touze, M., Francou, G., Laskar, J. 1994, Astronomy & Astrophysics 282, 663-683

Souchay, J., Loysel, B., Kinoshita, H., Folgueira, M. 1999, Astronomy & Astrophysics Supplement Series 135, 111

Wallace, P.T., "Software for Implementing the IAU 2000 Resolutions", in IERS Workshop 5.1 (2002)

gal_nut00b [0.1]

Nutation, IAU 2000B model.

```
void
gal_nut00b
(
  double date1,
  double date2,
  double *dpsi,
  double *deps
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return DPSI and DEPS contain the nutation (luni-solar and planetary). The nutation components in longitude and obliquity are in radians and with respect to the equinox and ecliptic of date. The obliquity at J2000 is assumed to be the Lieske et al. (1977) value of 84381.448 arcsec. (The errors that result from using this routine with the IAU 2006 value of 84381.406 arcseconds can be neglected.) The nutation model consists only of luni-solar terms, but includes also a fixed offset which compensates for certain long-period planetary terms. This routine is an implementation of the IAU 2000B abridged nutation model formally adopted by the IAU General Assembly in 2000. The routine computes the MHB_2000_SHORT luni-solar nutation series (Luzum 2001), but without the associated corrections for the precession rate adjustments and the offset between the GCRS and J2000 mean poles. The full IAU 2000A (MHB2000) nutation model contains nearly 1400 terms. The IAU 2000B model (McCarthy & Luzum 2003) contains only 77 terms, plus additional simplifications, yet still delivers results of 1 mas accuracy at present epochs. This combination of accuracy and size makes the IAU 2000B abridged nutation model suitable for most practical applications. The routine delivers a pole accurate to 1 mas from 1900 to 2100 (usually better than 1 mas, very occasionally just outside 1 mas). The full IAU 2000A model, which is implemented in the routine gal_nut00a (q.v.), delivers considerably greater accuracy at current epochs; however, to realize this improved accuracy, corrections for the essentially unpredictable free-core-nutation (FCN) must also be included. The routine provides classical nutation. The MHB_2000_SHORT algorithm, from which it is adapted, deals also with (i) the offsets between the GCRS and mean poles and (ii) the adjustments in longitude and obliquity due to the changed precession rates. These additional functions, namely frame bias and precession adjustments, are supported by the routines gal_bi00 and gal_pr00. The MHB_2000_SHORT algorithm also provides "total" nutations, comprising the arithmetic sum of the frame bias, precession adjustments, and nutation (luni-solar + planetary). These total nutations can be used in combination with an existing IAU 1976 precession implementation, such as gal_pmat76, to deliver GCRS-to-true predictions of mas accuracy at current epochs. However, for symmetry with the gal_nut00a routine (q.v. for the reasons), the routines do not generate the "total nutations" directly. Should they be required, they could of course easily be generated by calling gal_bi00, gal_pr00 and this routine and adding the results. The IAU 2000B model includes "planetary bias" terms that are fixed in size but compensate for long-period nutations. The amplitudes quoted in McCarthy & Luzum

(2003), namely Dpsi = -1.5835 mas and Depsilon = +1.6339 mas, are optimized for the "total nutations" method described above. The Luzum (2001) values used in this implementation, namely -0.135 mas and +0.388 mas, are optimized for the "rigorous" method, where frame bias, precession and nutation are applied separately and in that order. During the interval 1995-2050, the implementation delivers a maximum error of 1.001 mas (not including FCN).

References:

Lieske, J.H., Lederle, T., Fricke, W., Morando, B., "Expressions for the precession quantities based upon the IAU /1976/ system of astronomical constants", Astronomy & Astrophysics 58, 1-2, 1-16. (1977)

Luzum, B., private communication, 2001 (Fortran code MHB_2000_SHORT)

McCarthy, D.D. & Luzum, B.J., "An abridged model of the precession-nutation of the celestial pole", Celestial Mechanics & Dynamical Astronomy, 85, 37-49 (2003)

Simon, J.-L., Bretagnon, P., Chapront, J., Chapront-Touze, M., Francou, G., Laskar, J., Astronomy & Astrophysics 282, 663-683 (1994)

gal_nut06a [0.1]

IAU 2000A nutation with adjustments to match the IAU 2006 precession.

```
void
gal_nut06a
(
  double date1,
  double date2,
  double *dpsi,
  double *deps
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return DPSI and DEPS contain the nutation (luni-solar and planetary). The nutation components in longitude and obliquity are in radians and with respect to the mean equinox and ecliptic of date, IAU 2006 precession model (Hilton et al. 2006, Capitaine et al. 2005). The routine first computes the IAU 2000A nutation, then applies adjustments for (i) the consequences of the change in obliquity from the IAU 1980 ecliptic to the IAU 2006 ecliptic and (ii) the secular variation in the Earth's dynamical flattening. This routine provides classical nutation, complementing the IAU 2000 frame bias and IAU 2006 precession. It delivers a pole which is at current epochs accurate to a few tens of microarcseconds, apart from the free core nutation.

References:

Wallace, P.T. & Capitaine, N., 2006, Astronomy & Astrophysics 459, 981

gal_nut80 [0.1]

Nutation, IAU 1980 model.

```
void
gal_nut80
(
  double date1,
  double date2,
  double *dpsi,
  double *deps
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. on return DPSI contains the nutation in longitude (radians), and DEPS the nutation in obliquity (radians). The nutation components are with respect to the ecliptic of date.

References:

Explanatory Supplement to the Astronomical Almanac, P. Kenneth Seidelmann (ed.), University Science Books (1992), Section 3.222 (p111).

gal_nutm80 [0.1]

Form the matrix of nutation for a given date, IAU 1980 model.

```
void
gal_nutm80
(
  double date1,
  double date2,
  double rmatn[3][3]
) ;
```

On entry DATE1 and DATE2 contain the TDB Julian Date in standard SOFA two-piece format. On return RMATN contains the nutation matrix. The matrix operates in the sense V(true) = RMATN * V(mean), where the p-vector V(true) is with respect to the true equatorial triad of date and the p-vector V(mean) is with respect to the mean equatorial triad of date.

gal_obl06 [0.1]

Mean obliquity of the ecliptic, IAU 2006 precession model.

```
double
gal_obl06
(
  double date1,
  double date2
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. The routine returns the obliquity of the ecliptic in radians. The result is the angle between the ecliptic and mean equator of date (DATE1 and DATE2).

References:

Hilton, J. et al., 2006, Celestial Mechanics and Dynamical Astronomy 94, 351

gal_obl80 [0.1]

Mean obliquity of the ecliptic, IAU 1980 model.

```
double
gal_obl80
(
  double date1,
  double date2
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. the routine returns the obliquity of the ecliptic in radians. The result is the angle between the ecliptic and mean equator of date (DATE1 and DATE2).

References:

Explanatory Supplement to the Astronomical Almanac, P. Kenneth Seidelmann (ed.), University Science Books (1992), Expression 3.222-1 (p114).

gal_p06e [0.1]

Precession angles, IAU 2006, equinox based.

```
void
gal_p06e
```

```
(
  double date1,
  double date2,
  double *eps0,
  double *psia,
  double *oma,
  double *bpa,
  double *bqa,
  double *pia,
  double *bpia,
  double *epsa,
  double *chia,
  double *za,
  double *zetaa,
  double *thetaa,
  double *pa,
  double *gam,
  double *phi,
  double *psi
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. This routine returns the set of equinox based angles for the Capitaine et al. "P03" precession theory, adopted by the IAU in 2006. The angles are set out in Table 1 of Hilton et al. (2006):

EPS0	epsilon_0	obliquity at J2000
PSIA	psi_A	luni-solar precession
OMA	omega_A	inclination of equator wrt. J2000 ecliptic
BPA	P_A	ecliptic pole x, J2000 ecliptic triad
BQA	Q_A	ecliptic pole -y, J2000 ecliptic triad
PIA	pi_A	angle between moving and J2000 ecliptics
BPIA	Pi_A	longitude of ascending node of the ecliptic
EPSA	epsilon_A	obliquity of the ecliptic
CHIA	chi_A	planetary precession
ZA	z_A	equatorial precession: -3rd 323 Euler angle
ZETAA	zeta_A	equatorial precession: -1st 323 Euler angle
THETA	theta_A	equatorial precession: 2nd 323 Euler angle
PA	p_A	general precession
GAM	gamma_J2000	J2000 right ascension difference of ecliptic poles
PHI	phi_J2000	J2000 codeclination of ecliptic pole
PSI	psi_J2000	longitude difference of equator poles, J2000

The returned values are all radians. Hilton et al. (2006) Table 1 also contains angles that depend on models distinct from the P03 precession theory itself, namely the IAU 2000A frame bias and nutation. The quoted polynomials are used in other routines:

gal_xy06 contains the polynomial parts of the X and Y series.

gal_s06 contains the polynomial part of the s+XY/2 series.

gal_pfw06 implements the series for the Fukushima-Williams angles that are with respect to the GCRS pole (i.e. the variants that include frame bias).

The IAU resolution stipulated that the choice of parameterization was left to the user, and so an IAU compliant precession implementation can be constructed using various combinations of the angles returned by this routine.

The parameterization used is the Fukushima-Williams angles referred directly to the GCRS pole. These are the final four arguments returned by this routine, but are more efficiently calculated by calling the routine gal_pfw06. GAL also supports the direct computation of the CIP GCRS X,Y by series, available by calling gal_xy06. The agreement between the different parameterizations is at the 1 microarcsecond level in the present era. When constructing a precession formulation that refers to the GCRS pole rather than the dynamical pole, it may (depending on the choice of angles) be necessary to introduce the frame bias explicitly.

References:

Hilton, J. et al., 2006, Celestial Mechanics and Dynamical Astronomy 94, 351

gal_pb06 [0.1]

This routine forms three Euler angles which implement general precession from epoch J2000.0, using the IAU 2006 model. Frame bias (the offset between ICRS and mean J2000.0) is included.

```
void
gal_pb06
(
  double date1,
  double date2,
  double *bzeta,
  double *bz,
  double *btheta
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return the variables BZETA, BZ, and BTHETA are set as follows:

BZETA	1st rotation: radians clockwise around z
BZ	3rd rotation: radians clockwise around z
BTHETA	2nd rotation: radians counterclockwise around y

The traditional accumulated precession angles zeta_A, z_A, theta_A cannot be obtained in the usual way, namely through polynomial expressions, because of the frame bias. The latter means that two of the angles undergo rapid changes near this date. They are instead the results of decomposing the precession-bias matrix obtained by using the Fukushima-Williams method, which does not suffer from the problem. The decomposition returns values which can be used in the conventional formulation and which include frame bias. The three angles are returned in the conventional order, which is not the same as the order of the corresponding Euler rotations. The precession-bias matrix is R_3(-z) x R_2(+theta) x R_3(-zeta). Should zeta_A, z_A, theta_A angles be required that do not contain frame bias, they are available by calling the routine gal_p06e.

gal_pfw06 [0.1]

Precession angles, IAU 2006 (Fukushima-Williams 4-angle formulation).

```
void
gal_pfw06
(
  double date1,
  double date2,
  double *gamb,
  double *phib,
  double *psib,
  double *epsa
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return the routine sets the variables as follows:

GAMB	F-W angle gamma_bar (radians)
PHIB	F-W angle phi_bar (radians)
PSIB	F-W angle psi_bar (radians)
EPSA	F-W angle epsilon_A (radians)

Naming the following points:

e	J2000 ecliptic pole
p	GCRS pole
E	mean ecliptic pole of date
P	mean pole of date

the four Fukushima-Williams angles are as follows:

GAMB	= gamma_bar	= epE
PHIB	= phi_bar	= pE
PSIB	= psi_bar	= pEP
EPSA	= epsilon_A	= EP

The matrix representing the combined effects of frame bias and precession is:

P x B = R_1(-EPSA).R_3(-PSIB).R_1(PHIB).R_3(GAMB)

The matrix representing the combined effects of frame bias, precession and nutation is simply:

N x P x B = R_1(-EPSA-dE).R_3(-PSIB-dP).R_1(PHIB).R_3(GAMB)

where dP and dE are the nutation components with respect to the ecliptic of date.

References:

Hilton, J. et al., 2006, Celestial Mechanics and Dynamical Astronomy 94, 351

gal_plrotel00 [0.5]

This routine returns the rotational elements of a selected planet (IAU/IAG 2000).

```
int
gal_plrotel00
 (
   int body,
   double tdb1,
   double tdb2,
   double *alpha,
   double *delta,
   double *w,
   double *omega
 ) ; *** NEEDS FIXING ***
```

On entry the parameters are set as follows:

body	Planet identifier code
tdb1	Date part 1 (TDB)
tdb2	Date part 2 (TDB)

The header file gal_const.h defines the following constants for the planet identifier code:

GAL_SSB_SU	The Sun
GAL_SSB_ME	Mercury
GAL_SSB_VE	Venus
GAL_SSB_EA	Earth
GAL_SSB_MA	Mars
GAL_SSB_JU	Jupiter
GAL_SSB_SA	Saturn
GAL_SSB_UR	Uranus
GAL_SSB_NE	Neptune
GAL_SSB_PL	Pluto

tdb1 and tdb2 contain a Barycentric Dynamical Time (TDB) Julian Date in standard SOFA two-piece format. On return the variables are set as follows:

alpha	Right ascension of planet spin axis (radians)
delta	Declination of planet spin axis (radians)
w	Prime meridian angle (radians)
omega	Rotation Rate (radians per second)

The routine returns a status code of 0 if successful, and 1 if the planet identifier code is invalid.

Planetary coordinate systems are defined relative to their mean axis of rotation and various definitions of longitude depending on the body. The longitude systems of most of those bodies with observable rigid surfaces have been defined by references to a surface feature such as a crater. Approximate expressions for these rotational elements with respect to the J2000 inertial coordinate system have been derived. The International Celestial Reference Frame (ICRF) is the reference coordinate frame of epoch 2000 which is January 1.5 TDB. The variable quantities are expressed in units of days or Julian centuries of 36525 days. The north pole is that pole of rotation that lies on the north side of the invariable plane of the solar system. The direction of the north pole is specified by the value of its right ascension ra and declination dec, whereas the location of the prime meridian is specified by the angle that is measured along the planet's equator in an easterly direction with respect to the planet's north pole from the node Q (located at right ascension 90 degrees + ra) of the planet's equator on the standard equator to the point B where the prime meridian crosses the planet's equator. The right ascension of the point Q is 90 degrees + ra and the inclination of the planet's equator to the standard equator is 90 degrees - dec. Because the prime meridian is assumed to rotate uniformly with the planet, W accordingly varies linearly with time. In addition, ra, dec, and W may vary with time due to a precession of the axis of rotation of the planet. If W increases with time, the planet has direct (or prograde) rotation and if W decreases with time, the rotation is said to the retrograde.

References:

Satellite Orbits, Oliver Montenbruck & Eberhard Gill, Springer 2005, Pages 167, 191

Explanatory Supplement to the Astronomical Almanac, P. Kenneth Seidelmann Ed. Page 51

Report of the IAU/IAG Working Group on Cartographic Coordinates and Rotational Elements of the Planets and Satellites: 2000, P. K. Seidelmann et al.

gal_plrotel91 [0.5]

This routine returns the rotational elements of a selected planet (IAU/IAG 1991).

```
int
gal_plrotel91
 (
   int body,
   double tdb1,
   double tdb2,
   double *alpha,
   double *delta,
   double *w,
   double *omega
 ) ;  *** NEEDS FIXING ***
```

On entry the parameters are set as follows:

body	Planet identifier code
tdb1	Date part 1 (TDB)
tdb2	Date part 2 (TDB)

The header file gal_const.h defines the following constants for the planet identifier code:

GAL_SSB_SU	The Sun
GAL_SSB_ME	Mercury
GAL_SSB_VE	Venus
GAL_SSB_EA	Earth
GAL_SSB_MA	Mars
GAL_SSB_JU	Jupiter
GAL_SSB_SA	Saturn
GAL_SSB_UR	Uranus
GAL_SSB_NE	Neptune
GAL_SSB_PL	Pluto

tdb1 and tdb2 contain a Barycentric Dynamical Time (TDB) Julian Date in standard SOFA two-piece format. On return the variables are set as follows:

alpha	Right ascension of planet spin axis (radians)
delta	Declination of planet spin axis (radians)
w	Prime meridian angle (radians)
omega	Rotation Rate (radians per second)

The routine returns a status code of 0 if successful, and 1 if the planet identifier code is invalid.

Planetary coordinate systems are defined relative to their mean axis of rotation and various definitions of longitude depending on the body. The longitude systems of most of those bodies with observable rigid surfaces have been defined by references to a surface feature such as a crater. Approximate expressions for these rotational elements with respect to the J2000 inertial coordinate system have been derived. The International Celestial Reference Frame (ICRF) is the reference coordinate frame of epoch 2000 which is January 1.5 TDB. The variable quantities are expressed in units of days or Julian centuries of 36525 days. The north pole is that pole of rotation that lies on the north side of the invariable plane of the solar system. The direction of the north pole is specified by the value of its right ascension ra and declination dec, whereas the location of the prime meridian is specified by the angle that is measured along the planet's equator in an easterly direction with respect to the planet's north pole from the node Q (located at right ascension 90 degrees + ra) of the planet's equator on the standard equator to the point B where the prime meridian crosses the planet's equator. The right ascension of the point Q is 90 degrees + ra and the inclination of the planet's equator to the standard equator is 90 degrees - dec. Because the prime meridian is assumed to rotate uniformly with the planet, W accordingly varies linearly with time. In addition, ra, dec, and W may vary with time due to a precession of the axis of rotation of the planet. If W increases with time, the planet has direct (or prograde) rotation and if W decreases with time, the rotation is said to the retrograde.

References:

Satellite Orbits, Oliver Montenbruck & Eberhard Gill, Springer 2005, Pages 167, 191

Explanatory Supplement to the Astronomical Almanac, P. Kenneth Seidelmann Ed. Page 51

Report of the IAU/IAG/COSPAR Working Group on Cartographic Coordinates and Rotational Elements of the Planets and Satellites: 1991, M. E. Davies et al.

gal_pqw2ijk [0.4]

This routine transforms a pv-vector from the PQW frame to the to IJK frame.

```
void
gal_pqw2ijk
(
  double pqw[2][3],
  double raan,
  double argp,
  double inc,
  double ijk[2][3]
) ;
```

On entry the parameters are set as follows:

PQW	pv-vector in PQW frame (meters, meters per second)
RAAN	Longitude of the ascending mode (radians)
ARGP	Argument of pericenter (radians)
INC	Inclination (radians)

On return IJK contains the pv-vector in the IJK frame.

References:

Satellite Orbits, Oliver Montenbruck, Eberhard Gill, Springer 2005, Chapter 2

gal_pqw2ijkm [0.4]

This routine forms the PQW to IJK transformation matrix.

```
void
gal_pqw2ijkm
(
  double raan,
  double argp,
  double inc,
  double pqw2ijkm[3][3]
) ;
```

On entry the parameters are set as follows:

RAAN	Longitude of the ascending mode (radians)
ARGP	Argument of pericenter (radians)
INC	Inclination (radians)

On return PQW2IJKM contains the transformation matrix.

References:

Satellite Orbits, Oliver Montenbruck, Eberhard Gill, Springer 2005, Chapter 2

gal_sezm [0.5]

This routine forms the SEZ transformation matrix for specified latitude and longitude.

```
void
gal_sezm
(
  double latitude,
  double longitude,
  double sez[3][3]
) ;
```

On entry LATITUDE and LONGITUDE contain the required latitude and longitude in radians. On return SEZ contains the SEZ transformation matrix.

References:

Methods of Orbit Determination, P. R. Escobal 1965, Pages 405-406

gal_t2azel [0.2]

This routine converts a pv-vector in the International Terrestrial Reference Frame (ITRF) reference frame to azimuth, elevation, range & range-rate

```
void
gal_t2azel
(
  double itrf[2][3],
  double site[3],
  double lat,
  double lon,
  double *az,
  double *el,
  double *range,
  double *rdot
) ;
```

On entry the parameters are set as follows:

ITRF	ITRF position & velocity vector of target (meters, meters per second)
SITE	ITRF position vector of observer (meters)
LAT	Latitude of observer (radians)
LON	Longitude of observer (radians)

On return the variables are set as follows:

AZ	Azimuth (radians)
EL	Elevation (radians)
RANGE	Range (meters)
RDOT	Range Rate (meters, meters per second)

References:

Fundamentals of Astrodynamics and Applications, Vallado, David A. Second Edition 2004, Pages 252-257

gal_t2cpv00a [0.2]

This routine converts a position & velocity vector in the International Terrestrial Reference Frame (ITRF) to the Geocentric Celestial Reference Frame (GCRF) (IAU 2000A Resolutions).

```
void
gal_t2cpv00a
(
  double itrf[2][3],
  double utc1,
  double utc2,
  double dut1,
  double lod,
  double xp,
  double yp,
  double gcrf[2][3],
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

ITRF	ITRF position & velocity vector (meters, meters per second)
UTC1	Date part 1 (UTC)
UTC2	Date part 2 (UTC)
DUT1	UT1 - UTC (seconds)
LOD	Excess length of day (seconds)
XP	X coordinate of the pole (radians)

YP	Y coordinate of the pole (radians)

On return GCRF contains the GCRF position & velocity vector (meters, meters per second). UTC1 and UTC2 contain a Coordinated Universal Time (UTC) Julian Date in standard SOFA two-piece format. XP and YP are the "coordinates of the pole", in seconds, which position the Celestial Intermediate Pole in the International Terrestrial Reference System (see IERS Conventions 2003), measured along the meridians to 0 and 90 deg west respectively.

References:

SOFA Tools for Earth Attitude IAU Standards for Fundamental Astronomy Review Board 2007

http://www.iau-sofa.rl.ac.uk

Fundamentals of Astrodynamics and Applications, Vallado, David A. Second Edition 2004, Pages 217-219

gal_t2cpv00b [0.2]

This routine converts a position & velocity vector in the International Terrestrial Reference Frame (ITRF) to the Geocentric Celestial Reference Frame (GCRF) (IAU 2000B Resolutions).

```
void
gal_t2cpv00b
(
  double itrf[2][3],
  double utc1,
  double utc2,
  double dut1,
  double lod,
  double xp,
  double yp,
  double gcrf[2][3],
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

ITRF	ITRF position & velocity vector (meters, meters per second)
UTC1	Date part 1 (UTC)
UTC2	Date part 2 (UTC)
DUT1	UT1 - UTC (seconds)

LOD	Excess length of day (seconds)
XP	X coordinate of the pole (radians)
YP	Y coordinate of the pole (radians)

On return GCRF contains the GCRF position & velocity vector (meters, meters per second). UTC1 and UTC2 contain a Coordinated Universal Time (UTC) Julian Date in standard SOFA two-piece format. XP and YP are the "coordinates of the pole", in seconds, which position the Celestial Intermediate Pole in the International Terrestrial Reference System (see IERS Conventions 2003) , measured along the meridians to 0 and 90 deg west respectively.

References:

SOFA Tools for Earth Attitude IAU Standards for Fundamental Astronomy Review Board 2007

http://www.iau-sofa.rl.ac.uk

Fundamentals of Astrodynamics and Applications, Vallado, David A. Second Edition 2004, Pages 217-219

gal_t2cpv06a [0.2]

This routine converts a position & velocity vector in the International Terrestrial Reference Frame (ITRF) to the Geocentric Celestial Reference Frame (GCRF) (IAU 2006A Resolutions).

```
void
gal_t2cpv06a
(
  double itrf[2][3],
  double utc1,
  double utc2,
  double dut1,
  double lod,
  double xp,
  double yp,
  double gcrf[2][3],
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

ITRF	ITRF position & velocity vector (meters, meters per second)
UTC1	Date part 1 (UTC)

UTC2	Date part 2 (UTC)
DUT1	UT1 - UTC (seconds)
LOD	Excess length of day (seconds)
XP	X coordinate of the pole (radians)
YP	Y coordinate of the pole (radians)

On return gcrf contains the GCRF position & velocity vector (meters, meters per second). UTC1 and UTC2 contain a Coordinated Universal Time (UTC) Julian Date in standard SOFA two-piece format. XP and YP are the "coordinates of the pole", in seconds, which position the Celestial Intermediate Pole in the International Terrestrial Reference System (see IERS Conventions 2003) , measured along the meridians to 0 and 90 deg west respectively.

References:

SOFA Tools for Earth Attitude IAU Standards for Fundamental Astronomy Review Board 2007

http://www.iau-sofa.rl.ac.uk

Fundamentals of Astrodynamics and Applications, Vallado, David A. Second Edition 2004, Pages 217-219

gal_t2ipv00 [0.2]

This routine converts a position & velocity vector in the International Terrestrial Reference Frame (ITRF) to the CIRS reference frame (IAU 2000 Resolutions).

```
void
gal_t2ipv00
(
  double itrf[2][3],
  const double tta,
  const double ttb,
  const double ut1a,
  const double ut1b,
  const double lod,
  const double xp,
  const double yp,
  double cirs[2][3]
) ;
```

On entry the parameters are set as follows:

ITRF	ITRF position & velocity vector (meters, meters per second)

TTA	Date part 1 (TT)
TTB	Date part 2 (TT)
DUT1	UT1 - UTC (seconds)
LOD	Excess length of day (seconds)
XP	X coordinate of the pole (radians)
YP	Y coordinate of the pole (radians)

On return cirs contains the CIRS position & velocity vector (meters, meters per second). The TTA and TTB Terrestrial time (TT) Julian Date in is standard SOFA two-piece format. XP and YP are the "coordinates of the pole", in seconds, which position the Celestial Intermediate Pole in the International Terrestrial Reference System (see IERS Conventions 2003). In a geocentric right-handed triad u, v, w, where the w-axis points at the north geographic pole, the v-axis points towards the origin of longitudes and the u axis completes the system, XP = +u and YP = -v.

References:

SOFA Tools for Earth Attitude IAU Standards for Fundamental Astronomy Review Board 2007

http://www.iau-sofa.rl.ac.uk

Fundamentals of Astrodynamics and Applications, Vallado, David A. Second Edition 2004, Pages 217-219

gal_pmat00 [0.1]

Precession matrix (including frame bias) from GCRS to a specified date, IAU 2000 model.

```
void
gal_pmat00
(
  double date1,
  double date2,
  double rbp[3][3]
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return RBP contains the bias-precession matrix. The matrix operates in the sense V(date) = RBP * V(J2000), where the p-vector V(J2000) is with respect to the Geocentric Celestial Reference System (IAU, 2000) and the p-vector V(date) is with respect to the mean equatorial triad of the given date.

References:

IAU: Trans. International Astronomical Union, Vol. XXIVB; Proc. 24th General Assembly, Manchester, UK. Resolutions B1.3, B1.6. (2000)

gal_pmat06 [0.1]

Precession matrix (including frame bias) from GCRS to a specified date, IAU 2006 model.

```
void
gal_pmat06
(
  double date1,
  double date2,
  double rbp[3][3]
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return RBP contains the bias-precession matrix. The matrix operates in the sense V(date) = RBP * V(J2000), where the p-vector V(J2000) is with respect to the Geocentric Celestial Reference System (IAU, 2000) and the p-vector V(date) is with respect to the mean equatorial triad of the given date.

References:

Capitaine, N. & Wallace, P.T., 2006, Astronomy & Astrophysics 450, 855

Wallace, P.T. & Capitaine, N., 2006, Astronomy & Astrophysics 459, 981

gal_pmat76 [0.1]

Precession matrix from J2000 to a specified date, IAU 1976 model.

```
void
gal_pmat76
 (
    double date1,
    double date2,
    double rmatp[3][3]
 ) ;
```

On entry DATE1 and DATE2 contain the TDB Julian Date in standard SOFA two-piece format. On return RMATP contains the precession matrix, J2000 -> DATE1+DATE2. The matrix operates in the sense V(date) = RMATP * V(J2000), where the p-vector V(J2000) is with respect to the mean equatorial triad of epoch J2000 and the p-vector

V(date) is with respect to the mean equatorial triad of the given date. Though the matrix method itself is rigorous, the precession angles are expressed through canonical polynomials which are valid only for a limited time span. In addition, the IAU 1976 precession rate is known to be imperfect. The absolute accuracy of the present formulation is better than 0.1 arcseconds from 1960CE to 2040CE, better than 1 arcseconds from 1640CE to 2360CE, and remains below 3 arcseconds for the whole of the period 500BCE to 3000CE. The errors exceed 10 arcseconds outside the range 1200BCE to 3900CE, exceed 100 arcseconds outside 4200BCE to 5600CE and exceed 1000 arcseconds outside 6800BCE to 8200CE.

References:

Lieske, J.H., 1979. Astronomy & Astrophysics,73,282. equations (6) & (7), p283.

Kaplan, G.H., 1981. USNO circular no. 163, pA2.

gal_pn00 [0.1]

Precession-nutation, IAU 2000 model: a multi-purpose routine, supporting classical (equinox-based) use directly and CIO-based use indirectly.

```
void
gal_pn00
(
  double date1,
  double date2,
  double dpsi,
  double deps,
  double *epsa,
  double rb[3][3],
  double rp[3][3],
  double rbp[3][3],
  double rn[3][3],
  double rbpn[3][3]
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format, DPSI and DEPS contain the nutation. On return the variables are set as follows:

EPSA	mean obliquity
RB	frame bias matrix
RP	precession matrix
RBP	bias-precession matrix
RN	nutation matrix

RBPN GCRS-to-true matrix

The caller is responsible for providing the nutation components; they are in longitude and obliquity, in radians and are with respect to the equinox and ecliptic of date. For high-accuracy applications, free core nutation should be included as well as any other relevant corrections to the position of the CIP. The returned mean obliquity is consistent with the IAU 2000 precession-nutation models. The matrix RB transforms vectors from GCRS to J2000 mean equator and equinox by applying frame bias. The matrix RP transforms vectors from J2000 mean equator and equinox to mean equator and equinox of date by applying precession. The matrix RBP transforms vectors from GCRS to mean equator and equinox of date by applying frame bias then precession. It is the product RP x RB. The matrix RN transforms vectors from mean equator and equinox of date to true equator and equinox of date by applying the nutation (luni-solar and planetary). The matrix RBPN transforms vectors from GCRS to true equator and equinox of date. It is the product RN x RBP, applying frame bias, precession and nutation in that order.

References:

Capitaine, N., Chapront, J., Lambert, S. and Wallace, P., "Expressions for the Celestial Intermediate Pole and Celestial Ephemeris Origin consistent with the IAU 2000A precession-nutation model", Astronomy & Astrophysics, 400, 1145-1154 (2003)

gal_pn00a [0.1]

Precession-nutation, IAU 2000A model: a multi-purpose routine, supporting classical (equinox-based) use directly and CIO-based use indirectly.

```
void
gal_pn00a
(
  double date1,
  double date2,
  double *dpsi,
  double *deps,
  double *epsa,
  double rb[3][3],
  double rp[3][3],
  double rbp[3][3],
  double rn[3][3],
  double rbpn[3][3]
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return the variables are set as follows:

DPSI, DEPS	nutation
EPSA	mean obliquity
RB	frame bias matrix
RP	precession matrix
RBP	bias-precession matrix
RN	nutation matrix
RBPN	GCRS-to-true matrix

The nutation components (luni-solar and planetary, IAU 2000A) in longitude and obliquity are in radians and with respect to the equinox and ecliptic of date. Free core nutation is omitted; for the utmost accuracy, use the gal_pn00 routine, where the nutation components are caller-specified. For faster but slightly less accurate results, use the gal_pn00b routine. The mean obliquity is consistent with the IAU 2000 precession. The matrix RB transforms vectors from GCRS to J2000 mean equator and equinox by applying frame bias. The matrix RP transforms vectors from J2000 mean equator and equinox to mean equator and equinox of date by applying precession. The matrix RBP transforms vectors from GCRS to mean equator and equinox of date by applying frame bias then precession. It is the product RP x RB. The matrix RN transforms vectors from mean equator and equinox of date to true equator and equinox of date by applying the nutation (luni-solar and planetary). The matrix RBPN transforms vectors from GCRS to true equator and equinox of date. It is the product RN x RBP, applying frame bias, precession and nutation in that order. The X,Y,Z coordinates of the IAU 2000A Celestial Intermediate Pole are elements [0-2][2] of the matrix RBPN.

References:

Capitaine, N., Chapront, J., Lambert, S. and Wallace, P., "Expressions for the Celestial Intermediate Pole and Celestial Ephemeris Origin consistent with the IAU 2000A precession-nutation model", Astronomy & Astrophysics, 400, 1145-1154 (2003)

gal_pn00b [0.1]

Precession-nutation, IAU 2000B model: a multi-purpose routine, supporting classical (equinox-based) use directly and CIO-based use indirectly.

```
void
gal_pn00b
(
  double date1,
  double date2,
  double *dpsi,
  double *deps,
  double *epsa,
  double rb[3][3],
  double rp[3][3],
  double rbp[3][3],
  double rn[3][3],
  double rbpn[3][3]
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return the variables are set as follows:

DPSI, DEPS	nutation
EPSA	mean obliquity
RB	frame bias matrix
RP	bias-precession matrix
RBP	precession matrix
RN	nutation matrix
RBPN	GCRS-to-true matrix

The nutation components (luni-solar and planetary, IAU 2000B) in longitude and obliquity are in radians and with respect to the equinox and ecliptic of date. For more accurate results, but at the cost of increased computation, use the gal_pn00a routine. For the utmost accuracy, use the gal_pn00 routine, where the nutation components are caller-specified. The mean obliquity is consistent with the IAU 2000 precession. The matrix RB transforms vectors from GCRS to J2000 mean equator and equinox by applying frame bias. The matrix RP transforms vectors from J2000 mean equator and equinox to mean equator and equinox of date by applying precession. The matrix RBP transforms vectors from GCRS to mean equator and equinox of date by applying frame bias then precession. It is the product RP x RB. The matrix rn transforms vectors from mean equator and equinox of date to true equator and equinox of date by applying the nutation (luni-solar and planetary). The matrix RBPN transforms vectors from GCRS to true equator and equinox of date. It is the product RN x RBP, applying frame bias, precession and nutation in that order. The X,Y,Z coordinates of the IAU 2000B Celestial Intermediate Pole are elements [0-2][2] of the matrix RBPN.

References:

Capitaine, N., Chapront, J., Lambert, S. and Wallace, P., "Expressions for the Celestial Intermediate Pole and Celestial Ephemeris Origin consistent with the IAU 2000A precession-nutation model", Astronomy & Astrophysics, 400, 1145-1154 (2003)

gal_pn06 [0.1]

Precession-nutation, IAU 2006 model: a multi-purpose routine, supporting classical (equinox-based) use directly and CIO-based use indirectly.

```
void
gal_pn06
```

```
(
  double date1,
  double date2,
  double dpsi,
  double deps,
  double *epsa,
  double rb[3][3],
  double rp[3][3],
  double rbp[3][3],
  double rn[3][3],
  double rbpn[3][3]
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format, and dpsi and deps the nutation. On return the variables are set as follows:

EPSA	mean obliquity
RB	frame bias matrix
RP	precession matrix
RBP	bias-precession matrix
RN	nutation matrix
RBPN	GCRS-to-true matrix

The caller is responsible for providing the nutation components; they are in longitude and obliquity, in radians and are with respect to the equinox and ecliptic of date. For high-accuracy applications, free core nutation should be included as well as any other relevant corrections to the position of the CIP. The returned mean obliquity is consistent with the IAU 2006 precession. The matrix RB transforms vectors from GCRS to mean J2000 by applying frame bias. The matrix RP transforms vectors from mean J2000 to mean of date by applying precession. The matrix RBP transforms vectors from GCRS to mean of date by applying frame bias then precession. It is the product RP x RB. The matrix RN transforms vectors from mean of date to true of date by applying the nutation (luni-solar and planetary). The matrix RBPN transforms vectors from GCRS to true of date CIP/equinox). It is the product RN x RBP, applying frame bias, precession and nutation in that order. The X,Y,Z coordinates of the IAU 2006/2000A Celestial Intermediate Pole are elements [0-2][2] of the matrix RBPN.

References:

Capitaine, N. & Wallace, P.T., 2006, Astronomy & Astrophysics 450, 855

Wallace, P.T. & Capitaine, N., 2006, Astronomy & Astrophysics 459, 981

gal_pn06a [0.1]

Precession-nutation, IAU 2006/2000A models: a multi-purpose routine, supporting classical (equinox-based) use directly and CIO-based use indirectly.

```
void
gal_pn06a
(
  double date1,
  double date2,
  double *dpsi,
  double *deps,
  double *epsa,
  double rb[3][3],
  double rp[3][3],
  double rbp[3][3],
  double rn[3][3],
  double rbpn[3][3]
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return the variables are set as follows:

DPSI, DEPS	nutation
EPSA	mean obliquity
RB	frame bias matrix
RP	precession matrix
RBP	bias-precession matrix
RN	nutation matrix
RBPN	GCRS-to-true matrix

The nutation components (luni-solar and planetary, IAU 2000A) in longitude and obliquity are in radians and with respect to the equinox and ecliptic of date. Free core nutation is omitted; for the utmost accuracy, use the gal_pn06 routine, where the nutation components are caller-specified. The mean obliquity is consistent with the IAU 2006 precession. The matrix RB transforms vectors from GCRS to mean J2000 by applying frame bias. The matrix RP transforms vectors from mean J2000 to mean of date by applying precession. The matrix RBP transforms vectors from GCRS to mean of date by applying frame bias then precession. It is the product RP x RB. The matrix RN transforms vectors from mean of date to true of date by applying the nutation (luni-solar and planetary). The matrix RBPN transforms vectors from GCRS to true of date (CIP/equinox). It is the product RN x RBP, applying frame bias, precession and nutation in that order. The X,Y,Z coordinates of the IAU 2006/2000A Celestial Intermediate Pole are elements [0-2][2] of the matrix RBPN.

References:

Capitaine, N. & Wallace, P.T., 2006, Astronomy & Astrophysics 450, 855

gal_pnm00a [0.1]

Form the matrix of precession-nutation for a given date (including frame bias), equinox-based, IAU 2000A model.

```
void
gal_pnm00a
(
  double date1,
  double date2,
  double rbpn[3][3]
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return RBPN contains the classical NPB matrix. The matrix operates in the sense V(date) = RBPN x V(GCRS), where the p-vector V(date) is with respect to the true equatorial triad of date (DATE1 and DATE2) and the p-vector V(J2000) is with respect to the mean equatorial triad of the Geocentric Celestial Reference System (IAU, 2000). A faster, but slightly less accurate result (about 1 mas), can be obtained by using instead the gal_pnm00b routine.

References:

IAU: Trans. International Astronomical Union, Vol. XXIVB; Proc. 24th General Assembly, Manchester, UK. Resolutions B1.3, B1.6. (2000)

gal_pnm00b [0.1]

Form the matrix of precession-nutation for a given date (including frame bias), equinox-based, IAU 2000B model.

```
void
gal_pnm00b
(
  double date1,
  double date2,
  double rbpn[3][3]
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return RBPN the bias-precession-nutation matrix. The matrix operates in the sense V(date) = RBPN * V(GCRS), where the p-vector V(date) is with respect to the true equatorial triad of date (DATE1 and DATE2) and the p-vector

V(J2000) is with respect to the mean equatorial triad of the Geocentric Celestial Reference System (IAU, 2000). This routine is faster, but slightly less accurate (about 1 mas), than the gal_pnm00a routine.

References:

IAU: Trans. International Astronomical Union, Vol. XXIVB; Proc. 24th General Assembly, Manchester, UK. Resolutions B1.3, B1.6. (2000)

gal_pnm06a [0.1]

Form the matrix of precession-nutation for a given date (including frame bias), IAU 2006 precession and IAU 2000A nutation models.

```
void
gal_pnm06a
(
  double date1,
  double date2,
  double rnpb[3][3]
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return RNPB contains bias-precession-nutation matrix. The matrix operates in the sense V(date) = RNPB * V(GCRS), where the p-vector V(date) is with respect to the true equatorial triad of date (DATE1 and DATE2) and the p-vector V(J2000) is with respect to the mean equatorial triad of the Geocentric Celestial Reference System (IAU, 2000).

References:

Capitaine, N. & Wallace, P.T., 2006, Astronomy & Astrophysics 450, 855

gal_pnm80 [0.1]

Form the matrix of precession/nutation for a given date, IAU 1976 precession model, IAU 1980 nutation model.

```
void
gal_pnm80
(
  double date1,
  double date2,
  double rmatpn[3][3]
) ;
```

On entry DATE1 and DATE2 contain the Barycentric Dynamical Time (TDB) Julian Date in standard SOFA two-piece format. On return RMATPN contains the combined precession/nutation matrix. The matrix operates in the sense V(date) = RMATPN x V(J2000), where the p-vector V(date) is with respect to the true equatorial triad of date (DATE1 and DATE2) and the p-vector V(J2000) is with respect to the mean equatorial triad of epoch J2000.

References:

Explanatory Supplement to the Astronomical Almanac, P. Kenneth Seidelmann (ed.), University Science Books (1992), Section 3.3 (p145).

gal_pom00 [0.1]

Form the matrix of polar motion for a given date, IAU 2000.

```
void
gal_pom00
(
  double xp,
  double yp,
  double sp,
  double rpom[3][3]
) ;
```

On entry XP and YP contain the coordinates of the pole in radians and the TIO locator s' in radians. XP and YP are the "coordinates of the pole", in radians, which position the Celestial Intermediate Pole in the International Terrestrial Reference System (see IERS Conventions 2003), measured along the meridians to 0 and 90 deg west respectively. SP is the TIO locator s', in radians, which positions the Terrestrial Intermediate Origin on the equator. It is obtained from polar motion observations by numerical integration, and so is in essence unpredictable. However, it is dominated by a secular drift of about 47 microarcseconds per century, and so can be taken into account by using s' = -47 * T, where T is centuries since J2000.0. The routine gal_sp00 implements this approximation. The matrix operates in the sense V(TRS) = RPOM * V(CIP), meaning that it is the final rotation when computing the pointing direction to a celestial source.

References:

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

gal_pr00 [0.1]

Precession-rate part of the IAU 2000 precession-nutation models (part of MHB2000).

```
void
gal_pr00
(
  double date1,
  double date2,
  double *dpsipr,
  double *depspr
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return DPSIPR and DEPSPR contain the precession corrections. The precession adjustments are expressed as "nutation components", corrections in longitude and obliquity with respect to the J2000 equinox and ecliptic. Although the precession adjustments are stated to be with respect to Lieske et al. (1977), the MHB2000 model does not specify which set of Euler angles are to be used and how the adjustments are to be applied. The most literal and straightforward procedure is to adopt the 4-rotation epsilon_0, psi_A, omega_A, xi_A option, and to add DPSIPR to psi_A and DEPSPR to both omega_A and eps_A (Wallace 2002). This is an implementation of one aspect of the IAU 2000A nutation model, formally adopted by the IAU General Assembly in 2000, namely MHB2000 (Mathews et al. 2002).

References

Lieske, J.H., Lederle, T., Fricke, W. & Morando, B., "Expressions for the precession quantities based upon the IAU (1976) System of Astronomical Constants", Astronomy & Astrophysics, 58, 1-16 (1977)

Mathews, P.M., Herring, T.A., Buffet, B.A., "Modeling of nutation and precession New nutation series for non-rigid Earth and insights into the Earth's interior", Journal Geophysical Research, 107, B4, 2002. The MHB2000 code itself was obtained on 9th September 2002 from ftp://maia.usno.navy.mil/conv2000/chapter5/IAU2000A.

Wallace, P.T., "Software for Implementing the IAU 2000 Resolutions", in IERS Workshop 5.1 (2002)

gal_prec76 [0.1]

IAU 1976 precession model. This routine forms the three Euler angles which implement general precession between two epochs, using the IAU 1976 model (as for the FK5 catalog).

```
void
gal_prec76
```

```
(
  double ep01,
  double ep02,
  double ep11,
  double ep12,
  double *zeta,
  double *z,
  double *theta
) ;
```

On entry EP01 and EP02 contain the Barycentric Dynamical Time (TDB) starting epoch, and EP11 and EP12 contain the TDB ending epoch. Both dates are Julian Dates in standard SOFA two-piece format. On return the variables are set as follows:

ZETA	1st rotation: radians clockwise around z
Z	3rd rotation: radians clockwise around z
THETA	2nd rotation: radians counterclockwise around y

The accumulated precession angles ZETA, Z, THETA are expressed through canonical polynomials which are valid only for a limited time span. In addition, the IAU 1976 precession rate is known to be imperfect. The absolute accuracy of the present formulation is better than 0.1 arcseconds from 1960CE to 2040CE, better than 1 arcseconds from 1640CE to 2360CE, and remains below 3 arcseconds for the whole of the period 500BCE to 3000CE. The errors exceed 10 arcseconds outside the range 1200BCE to 3900CE, exceed 100 arcseconds outside 4200BCE to 5600CE and exceed 1000 arcseconds 1000 arcseconds outside 6800BCE to 8200CE. The three angles are returned in the conventional order, which is not the same as the order of the corresponding Euler rotations. The precession matrix is R_3(-Z) x R_2(+THETA) x R_3(-ZETA).

References:

Lieske, J.H., 1979. Astronomy & Astrophysics,73,282. equations (6) & (7), p283.

gal_s00 [0.1]

The CIO locator s, positioning the Celestial Intermediate Origin on the equator of the Celestial Intermediate Pole, given the CIP's X,Y coordinates. Compatible with IAU 2000A precession-nutation.

```
double
gal_s00
(
  double date1,
  double date2,
  double x,
  double y
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format, X and Y contain the CIP coordinates. The routine returns the CIO locator s in radians. The CIO locator s is the difference between the right ascensions of the same point in two systems: the two systems are the GCRS and the CIP,CIO, and the point is the ascending node of the CIP equator. The quantity s remains below 0.1 arcsecond throughout 1900CE-2100CE. The series used to compute s is in fact for s+xy/2, where x and y are the x and y components of the CIP unit vector; this series is more compact than a direct series for s would be. This routine requires X and Y to be supplied by the caller, who is responsible for providing values that are consistent with the supplied date. The model is consistent with the IAU 2000A precession-nutation.

References:

Capitaine, N., Chapront, J., Lambert, S. and Wallace, P., "Expressions for the Celestial Intermediate Pole and Celestial Ephemeris Origin consistent with the IAU 2000A precession-nutation model", Astronomy & Astrophysics, 400, 1145-1154 (2003)

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

gal_s00a [0.1]

The CIO locator s, positioning the Celestial Intermediate Origin on the equator of the Celestial Intermediate Pole, using the IAU 2000A precession-nutation model.

```
double
gal_s00a
(
  double date1,
  double date2
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. The routine returns the CIO locator s in radians. The CIO locator s is the difference between the right ascensions of the same point in two systems. The two systems are the GCRS and the CIP,CIO, and the point is the ascending node of the CIP equator. The CIO locator s remains a small fraction of 1 arcsecond throughout 1900CE-2100CE. The series used to compute s is in fact for s+XY/2, where X and Y are the x and y components of the CIP unit vector; this series is more compact than a direct series for s would be. This routine uses the full IAU 2000A nutation model when predicting the CIP position. Faster results, with no significant loss of accuracy, can be obtained via the routine gal_s00b, which uses instead the IAU 2000B truncated model.

References:

Capitaine, N., Chapront, J., Lambert, S. and Wallace, P., "Expressions for the Celestial Intermediate Pole and Celestial Ephemeris Origin consistent with the IAU 2000A precession-nutation model", Astronomy & Astrophysics, 400, 1145-1154 (2003) n.b. The celestial ephemeris origin (CEO) was renamed "celestial intermediate origin" (CIO) by IAU 2006 Resolution 2.

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

gal_s00b [0.1]

The CIO locator s, positioning the Celestial Intermediate Origin on the equator of the Celestial Intermediate Pole, using the IAU 2000B precession-nutation model.

```
double
gal_s00b
(
  double date1,
  double date2
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. The routine returns the CIO locator s in radians. The CIO locator s is the difference between the right ascensions of the same point in two systems. The two systems are the GCRS and the CIP,CIO, and the point is the ascending node of the CIP equator. The CIO locator s remains a small fraction of 1 arcsecond throughout 1900CE-2100CE. The series used to compute s is in fact for s+XY/2, where X and Y are the x and y components of the CIP unit vector; this series is more compact than a direct series for s would be. This routine uses the IAU 2000B truncated nutation model when predicting the CIP position. The routine gal_s00a uses instead the full IAU 2000A model, but with no significant increase in accuracy and at some cost in speed.

References:

Capitaine, N., Chapront, J., Lambert, S. and Wallace, P., "Expressions for the Celestial Intermediate Pole and Celestial Ephemeris Origin consistent with the IAU 2000A precession-nutation model", Astronomy & Astrophysics, 400, 1145-1154 (2003) n.b. The celestial ephemeris origin (CEO) was renamed "celestial intermediate origin" (CIO) by

IAU 2006 Resolution 2.

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

gal_s06 [0.1]

The CIO locator s, positioning the Celestial Intermediate Origin on the equator of the Celestial Intermediate Pole, given the CIP's X,Y coordinates. Compatible with IAU 2006/2000A precession-nutation.

```
double
gal_s06
(
  double date1,
  double date2,
  double x,
  double y
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format, X and Y contain CIP coordinates. The routine returns the CIO locator s in radians. The CIO locator s is the difference between the right ascensions of the same point in two systems: the two systems are the GCRS and the CIP,CIO, and the point is the ascending node of the CIP equator. The quantity s remains below 0.1 arcsecond throughout 1900CE - 2100CE. The series used to compute s is in fact for s+xy/2, where x and y are the x and y components of the CIP unit vector; this series is more compact than a direct series for s would be. This routine requires X,Y to be supplied by the caller, who is responsible for providing values that are consistent with the supplied date. The model is consistent with the "P03" precession (Capitaine et al. 2003), adopted by IAU 2006 Resolution 1, 2006, and the IAU 2000A nutation (with P03 adjustments).

References:

Capitaine, N., Wallace, P.T. & Chapront, J., 2003, Astronomy & Astrophysics 432, 355

McCarthy, D.D., Petit, G. (eds.) 2004, IERS Conventions (2003), IERS Technical Note No. 32, BKG

gal_s06a [0.1]

The CIO locator s, positioning the Celestial Intermediate Origin on the equator of the Celestial Intermediate Pole, using the IAU 2006 precession and IAU 2000A nutation models.

```
double
gal_s06a
(
  double date1,
  double date2
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. The routine returns the CIO locator s in radians. The CIO locator s is the difference between the right ascensions of the same point in two systems. The two systems are the GCRS and the CIP,CIO, and the point is the ascending node of the CIP equator. The CIO locator s remains a small fraction of 1 arcsecond throughout 1900CE-2100CE. The series used to compute s is in fact for s+XY/2, where X and Y are the x and y components of the CIP unit vector; this series is more compact than a direct series for s would be. This routine uses the full IAU 2000A nutation model when predicting the CIP position.

References:

Capitaine, N., Chapront, J., Lambert, S. and Wallace, P., "Expressions for the Celestial Intermediate Pole and Celestial Ephemeris Origin consistent with the IAU 2000A precession-nutation model", Astronomy & Astrophysics, 400, 1145-1154 (2003) n.b. The celestial ephemeris origin (CEO) was renamed "celestial intermediate origin" (CIO) by IAU 2006 Resolution 2.

Capitaine, N. & Wallace, P.T., 2006, Astronomy & Astrophysics 450, 855

McCarthy, D. D., Petit, G. (eds.), 2004, IERS Conventions (2003), IERS Technical Note No. 32, BKG

gal_sp00 [0.1]

The TIO locator s', positioning the Terrestrial Intermediate Origin on the equator of the Celestial Intermediate Pole.

```
double
gal_sp00
(
  double date1,
  double date2
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. The routine returns the TIO locator s' in radians. The TIO locator s' is obtained from polar motion observations by numerical integration, and so is in essence unpredictable. However, it is dominated by a secular drift of about 47 microarcseconds per century, which is the approximation evaluated by this routine.

References:

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

gal_xy06 [0.1]

X,Y coordinates of celestial intermediate pole from series based on IAU 2006 precession and IAU 2000A nutation.

```
void
gal_xy06
(
  double date1,
  double date2,
  double *x,
  double *y
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return X and Y contain the CIP X,Y coordinates. The X,Y coordinates are those of the unit vector towards the celestial intermediate pole. They represent the combined effects of frame bias, precession and nutation. The fundamental arguments used are as adopted in IERS Conventions (2003) and are from Simon et al. (1994) and Souchay et al. (1999). This is an alternative to the angles-based method, via the routine gal_fw2xy and as used in gal_xys06a for example. The two methods agree at the 1 microarcsecond level (at present), a negligible amount compared with the intrinsic accuracy of the models. However, it would be unwise to mix the two methods (angles-based and series-based) in a single application.

References:

Capitaine, N., Wallace, P.T. & Chapront, J., 2003, Astronomy & Astrophysics, 412, 567

Capitaine, N. & Wallace, P.T., 2006, Astronomy & Astrophysics 450, 855

McCarthy, D. D., Petit, G. (eds.), 2004, IERS Conventions (2003), IERS Technical Note No. 32, BKG

Simon, J.L., Bretagnon, P., Chapront, J., Chapront-Touze, M., Francou, G. & Laskar, J., Astronomy & Astrophysics, 1994, 282, 663

Souchay, J., Loysel, B., Kinoshita, H., Folgueira, M., 1999, Astronomy & Astrophysics Supplement Series 135, 111

Wallace, P.T. & Capitaine, N., 2006, Astronomy & Astrophysics 459, 981

gal_xys00a [0.1]

For a given TT date, compute the X,Y coordinates of the Celestial Intermediate Pole and the CIO locator s, using the IAU 2000A precession-nutation model.

```
void
gal_xys00a
(
  double date1,
  double date2,
  double *x,
  double *y,
  double *s
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return the variables are set as follows:

X, Y	Celestial Intermediate Pole
S	the CIO locator s

The Celestial Intermediate Pole coordinates are the X, Y components of the unit vector in the Geocentric Celestial Reference System. The CIO locator S (radians) positions the Celestial Intermediate Origin on the equator of the CIP. A faster, but slightly less accurate result (about 1 mas for X, Y), can be obtained by using instead the gal_xys00b routine.

References:

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

gal_xys00b [0.1]

For a given TT date, compute the X,Y coordinates of the Celestial Intermediate Pole and the CIO locator s, using the IAU 2000B precession-nutation model.

```
void
gal_xys00b
```

```
(
  double date1,
  double date2,
  double *x,
  double *y,
  double *s
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return the variables are set as follows:

X, Y	Celestial Intermediate Pole
S	the CIO locator s

The Celestial Intermediate Pole coordinates are the X, Y components of the unit vector in the Geocentric Celestial Reference System. The CIO locator S (radians) positions the Celestial Intermediate Origin on the equator of the CIP. This routine is faster, but slightly less accurate (about 1 mas in x,y), than the gal_xys00a routine.

References:

McCarthy, D. D., Petit, G. (eds.), IERS Conventions (2003), IERS Technical Note No. 32, BKG (2004)

gal_xys06a [0.1]

For a given TT date, compute the X,Y coordinates of the Celestial Intermediate Pole and the CIO locator s, using the IAU 2006 precession and IAU 2000A nutation models.

```
void
gal_xys06a
(
  double date1,
  double date2,
  double *x,
  double *y,
  double *s
) ;
```

On entry DATE1 and DATE2 contain the Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. On return the variables are set as follows:

X, Y	Celestial Intermediate Pole
S	the CIO locator s

The Celestial Intermediate Pole coordinates are the X, Y components of the unit vector in the Geocentric Celestial Reference System. The CIO locator S (radians) positions the Celestial Intermediate Origin on the equator of the CIP. Series-based solutions for generating X and Y are also available: see Capitaine & Wallace (2006) and gal_xy06.

References:

Capitaine, N. & Wallace, P.T., 2006, Astronomy & Astrophysics 450, 855

Wallace, P.T. & Capitaine, N., 2006, Astronomy & Astrophysics 459, 981

Chapter 8 - Star Routines

The routines detailed in this chapter are defined in the gal_star.h header file.

gal_fk52h [0.1]

Transform FK5 (J2000.0) star data into the Hipparcos system.

```
void
gal_fk52h
(
  double r5,
  double d5,
  double dr5,
  double dd5,
  double px5,
  double rv5,
  double *rh,
  double *dh,
  double *drh,
  double *ddh,
  double *pxh,
  double *rvh
) ;
```

On entry the parameters are set as follows (all FK5, equinox J2000.0, epoch J2000.0):

R5	right ascension (radians)
D5	declination (radians)
DR5	proper motion in right ascension (dRA/dt, radians per Julian year)
DD5	proper motion in declination (dDec/dt, radians per Julian year)
PX5	parallax (arcseconds)
RV5	radial velocity (positive = receding)

On return the variables are set as follows (all Hipparcos, epoch J2000.0):

RH	right ascension (radians)
DH	declination (radians)
DRH	proper motion in right ascension (dRA/dt, radians per Julian year)
DDH	proper motion in declination (dDec/dt, radians per Julian year)
PXH	parallax (arcseconds)
RVH	radial velocity (positive = receding)

This routine transforms FK5 star positions and proper motions into the system of the Hipparcos catalogue. The proper motions in right ascension are dRA/dt rather than cos(Dec)*dRA/dt, and are per year rather than per century. The FK5 to Hipparcos transformation is modeled as a pure rotation and spin; zonal errors in the FK5 catalogue are not taken into account. See also gal_h2fk5, gal_fk5hz, gal_hfk5z.

References:

F. Mignard & M. Froeschle, Astronomy & Astrophysics 354, 732-739 (2000).

gal_fk5hip [0.1]

FK5 to Hipparcos rotation and spin.

```
void
gal_fk5hip
(
  double r5h[3][3],
  double s5h[3]
) ;
```

On return R5H contains the r-matrix: FK5 rotation wrt Hipparcos, and S5H contains the r-vector: FK5 spin wrt Hipparcos. This routine models the FK5 to Hipparcos transformation as a pure rotation and spin; zonal errors in the FK5 catalogue are not taken into account. The r-matrix r5h operates in the sense: P_Hipparcos = R5H x P_FK5 where P_FK5 is a p-vector in the FK5 frame, and P_Hipparcos is the equivalent Hipparcos p-vector. The r-vector S5H represents the time derivative of the FK5 to Hipparcos rotation. The units are radians per year (Julian, TDB).

References:

F. Mignard & M. Froeschle, Astronomy & Astrophysics 354, 732-739 (2000).

gal_fk5hz [0.1]

Transform an FK5 (J2000) star position into the system of the Hipparcos catalogue, assuming zero Hipparcos proper motion.

```
void
gal_fk5hz
(
  double r5,
  double d5,
  double date1,
  double date2,
  double *rh,
   double *dh
) ;
```

On entry the parameters are set as follows:

R5	FK5 right ascension (radians), equinox J2000.0, at date
D5	FK5 declination (radians), equinox J2000.0, at date
DATE1	TDB date part 1 in standard SOFA two-piece format
DATE2	TDB date part 2 in standard SOFA two-piece format

On return the variables are set as follows:

RH	Hipparcos right ascension (radians)
DH	Hipparcos declination (radians)

This routine converts a star position from the FK5 system to the Hipparcos system, in such a way that the Hipparcos proper motion is zero. Because such a star has, in general, a non-zero proper motion in the FK5 system, the routine requires the date at which the position in the FK5 system was determined. The FK5 to Hipparcos transformation is modeled as a pure rotation and spin; zonal errors in the FK5 catalogue are not taken into account. It was the intention that Hipparcos should be a close approximation to an inertial frame, so that distant objects have zero proper motion; such objects have (in general) non-zero proper motion in FK5, and this routine returns those fictitious proper motions. The position returned by this routine is in the FK5 J2000 reference system but at date (DATE1 and DATE2). See also gal_fk52h, gal_h2fk5, gal_hfk5z.

References:

F. Mignard & M. Froeschle, Astronomy & Astrophysics 354, 732-739 (2000).

gal_h2fk5 [0.1]

Transform Hipparcos star data into the FK5 (J2000.0) system.

```
void
gal_h2fk5
(
  double rh,
  double dh,
  double drh,
  double ddh,
  double pxh,
  double rvh,
  double *r5,
  double *d5,
  double *dr5,
  double *dd5,
  double *px5,
  double *rv5
) ;
```

On entry the parameters are set as follows (all Hipparcos, epoch J2000.0):

RH	right ascension (radians)
DH	declination (radians)
DRH	proper motion in right ascension (dRA/dt, radians per Julian year)
DDH	proper motion in declination (dDec/dt, radians per Julian year)
PXH	parallax (arcseconds)
RVH	radial velocity (positive = receding)

On return the variables are set as follows (all FK5, equinox J2000.0, epoch J2000.0):

R5	right ascension (radians)
D5	declination (radians)
DR5	proper motion in right ascension (dRA/dt, radians per Julian year)
DD5	proper motion in declination (dDec/dt, radians per Julian year)
PX5	parallax (arcseconds)
RV5	radial velocity (positive = receding)

This routine transforms Hipparcos star positions and proper motions into FK5 J2000. The proper motions in right ascension are dRA/dt rather than cos(Dec)*dRA/dt, and are per year rather than per century. The FK5 to Hipparcos transformation is modeled as a pure rotation and spin; zonal errors in the FK5 catalogue are not taken into account. See also gal_fk52h, gal_fk5hz, gal_hfk5z.

References:

F. Mignard & M. Froeschle, Astronomy & Astrophysics 354, 732-739 (2000).

gal_hfk5z [0.1]

Transform a Hipparcos star position into FK5 J2000.0, assuming zero Hipparcos proper motion.

```
void
gal_hfk5z
(
  double rh,
  double dh,
  double date1,
  double date2,
  double *r5,
  double *d5,
  double *dr5,
  double *dd5
) ;
```

On entry the parameters are set as follows:

RH	Hipparcos right ascension (radians)
DH	Hipparcos declination (radians)
DATE1,DATE2	TDB date in standard SOFA two-piece format

On return the variables are set as follows (all FK5, equinox J2000, date date1+date2):

R5	right ascension (radians)
D5	declination (radians)
DR5	FK5 right ascension proper motion (radians per year)
DD5	declination proper motion (radians per year)

The proper motion in right ascension is dRA/dt rather than cos(Dec)*dRA/dt. The FK5 to Hipparcos transformation is modeled as a pure rotation and spin; zonal errors in the FK5 catalogue are not taken into account. It was the intention that Hipparcos should be a close approximation to an inertial frame, so that distant objects have zero proper motion; such objects have (in general) non-zero proper motion in FK5, and this routine returns those fictitious proper motions. The position returned by this routine is in the FK5 J2000.0 reference system but at date (DATE1 and DATE2). See also gal_fk52h, gal_h2fk5, gal_fk5zhz.

References:

F. Mignard & M. Froeschle, Astronomy & Astrophysics 354, 732-739 (2000).

gal_pvstar [0.1]

Convert star position & velocity vector to catalog coordinates.

```
void
gal_pvstar
(
  double pv[2][3],
  double *ra,
  double *dec,
  double *pmr,
  double *pmd,
  double *px,
  double *rv,
  gal_status_t *status
) ;
```

On entry PV contains the pv-vector (AU, AU per day). On return the variables are set as follows:

RA	right ascension (radians)
DEC	declination (radians)
PMR	right ascension proper motion (radians per year)
PMD	declination proper motion (radians per year)
PX	parallax (arcseconds)
RV	radial velocity (kilometers per second, positive = receding)

The specified pv-vector is the coordinate direction (and its rate of change) for the epoch at which the light leaving the star reached the solar-system Barycenter. The star data returned by this routine are "observables" for an imaginary observer at the solar-system Barycenter. Proper motion and radial velocity are, strictly, in terms of Barycentric Coordinate Time, TCB. For most practical applications, it is permissible to neglect the distinction between TCB and ordinary "proper" time on Earth (TT/TAI). The result will, as a rule, be limited by the intrinsic accuracy of the proper-motion and radial-velocity data; moreover, the supplied pv-vector is likely to be merely an intermediate result (for example generated by the routine gal_starpv), so that a change of time unit will cancel out overall.

In accordance with normal star-catalog conventions, the object's right ascension and declination are freed from the effects of secular aberration. The frame, which is aligned to the catalog equator and equinox, is Lorentzian and centered on the SSB. Summarizing, the specified pv-vector is for most stars almost identical to the result of applying the standard geometrical "space motion" transformation to the catalog data. The differences, which are the subject of the Stumpff paper cited below, are:

(i) In stars with significant radial velocity and proper motion, the constantly changing light-time distorts the apparent proper motion. Note that this is a classical, not a relativistic, effect.

(ii) The transformation complies with special relativity.

Care is needed with units. The star coordinates are in radians and the proper motions in radians per Julian year, but the parallax is in arcseconds; the radial velocity is in kilometers per second, but the pv-vector result is in AU and AU per day. The proper motions are the rate of change of the right ascension and declination at the catalog epoch and are in radians per Julian year. The right ascension proper motion is in terms of coordinate angle, not true angle, and will thus be numerically larger at high declinations. Straight-line motion at constant speed in the inertial frame is assumed. If the speed is greater than or equal to the speed of light, the routine sets the error code

GAL_EXCESSIVE_VELOCITY and aborts. The inverse transformation is performed by the routine gal_starpv.

If an internal error occurs then the appropriate error code is set.

References:

Stumpff, P., Astronomy & Astrophysics 144, 232-240 (1985).

gal_starpm [0.1]

Star proper motion: update star catalog data for space motion.

```
void
gal_starpm
(
  double ra1,
  double dec1,
  double pmr1,
  double pmd1,
  double px1,
  double rv1,
  double ep1a,
  double ep1b,
  double ep2a,
  double ep2b,
  double *ra2,
  double *dec2,
  double *pmr2,
  double *pmd2,
  double *px2,
  double *rv2,
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

RA1	right ascension (radians), before
DEC1	declination (radians), before
PMR1	right ascension proper motion (radians per year), before
PMD1	declination proper motion (radians per year), before
PX1	parallax (arcseconds), before
RV1	radial velocity (kilometers per second, positive = receding), before
EP1a	"before" epoch, part A
EP1b	"before" epoch, part B
EP2a	"after" epoch, part A

EP2b	"after" epoch, part B

On return the variables are set as follows:

RA2	right ascension (radians), after
DEC2	declination (radians), after
PMR2	right ascension proper motion (radians per year), after
PMD2	declination proper motion (radians per year), after
PX2	parallax (arcseconds), after
RV2	radial velocity (kilometers per second, positive = receding), after

The routine returns the following status codes:

-1	system error (should not occur)
0	no warnings or errors
1	distance overridden
2	excessive velocity
4	solution didn't converge
else	binary logical OR of the above warnings

The starting and ending Barycentric dynamical Time (TDB) epochs EP1A and EP1B and EP2A and EP2B are Julian Dates in standard SOFA two-piece format. In accordance with normal star-catalog conventions, the object's right ascension and declination are freed from the effects of secular aberration. The frame, which is aligned to the catalog equator and equinox, is Lorentzian and centered on the SSB. The proper motions are the rate of change of the right ascension and declination at the catalog epoch and are in radians per TDB Julian year. The parallax and radial velocity are in the same frame. Care is needed with units. The star coordinates are in radians and the proper motions in radians per Julian year, but the parallax is in arcseconds. The RA proper motion is in terms of coordinate angle, not true angle. If the catalog uses arcseconds for both RA and DEC proper motions, the RA proper motion will need to be divided by cos(DEC) before use. Straight-line motion at constant speed, in the inertial frame, is assumed. An extremely small (or zero or negative) parallax is interpreted to mean that the object is on the "celestial sphere", the radius of which is an arbitrary (large) value (see the gal_starpv routine for the value used). When the distance is overridden in this way, the error code GAL_DISTANCE_OVERRIDEN is set. If the space velocity is a significant fraction of c (see the constant VMAX in the routine gal_starpv), it is arbitrarily set to zero. When this action occurs the error code GAL_EXCESSIVE_VELOCITY is set. The relativistic adjustment carried out in the gal_starpv routine involves an iterative calculation. If the process fails to converge within a set number of iterations, the error code GAL_NO_CONVERGENCE is set.

gal_starpv [0.1]

Convert star catalog coordinates to position & velocity vector.

```
void
gal_starpv
(
  double ra,
  double dec,
  double pmr,
  double pmd,
  double px,
  double rv,
  double pv[2][3],
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

RA	right ascension (radians)
DEC	declination (radians)
PMR	right ascension proper motion (radians per year)
PMD	declination proper motion (radians per year)
PX	parallax (arcseconds)
RV	radial velocity (kilometers per second, positive means receding)

On return PV contains the pv-vector (AU, AU per day).

The routine returns one of the following status codes:

0	no warnings
1	distance overridden
2	excessive velocity
4	solution didn't converge
else	binary logical OR of the above

The star data accepted by this routine are "observables" for an imaginary observer at the solar-system Barycenter. Proper motion and radial velocity are, strictly, in terms of Barycentric Coordinate Time, TCB. For most practical applications, it is permissible to neglect the distinction between TCB and ordinary "proper" time on Earth (TT/TAI). The result will, as a rule, be limited by the intrinsic accuracy of the proper-motion and radial-velocity data; moreover, the pv-vector is likely to be merely an intermediate result, so that a change of time unit would cancel out overall. In accordance with normal star-catalog conventions, the object's right ascension and declination are freed from the effects of secular aberration. The frame, which is aligned to the catalog equator and equinox, is Lorentzian and centered on the SSB. The resulting position and velocity pv-vector is with respect to the same frame and, like the catalog coordinates, is freed from the effects of secular aberration. Should the "coordinate direction", where the object was located at the catalog epoch, be required, it may be obtained by calculating the magnitude of the position vector PV[0][0-2] dividing by the speed of light in AU per day to give the light-time, and then multiplying the space velocity PV[1][0-2] by this light-time and adding the result to PV[0][0-2]. Summarizing, the pv-vector returned is for most stars almost identical to the result of applying the standard geometrical "space motion" transformation. The differences, which are the subject of the Stumpff paper referenced below, are:

> In stars with significant radial velocity and proper motion, the constantly changing light-time distorts the apparent proper motion. Note that this is a classical, not a relativistic, effect.
>
> The transformation complies with special relativity.

Care is needed with units. The star coordinates are in radians and the proper motions in radians per Julian year, but the parallax is in arcseconds; the radial velocity is in kilometers per second, but the pv-vector result is in AU and AU per day. The RA proper motion is in terms of coordinate angle, not true angle. If the catalog uses arcseconds for both RA and DEC proper motions, the RA proper motion will need to be divided by cos(DEC) before use. Straight-line motion at constant speed, in the inertial frame, is assumed. An extremely small (or zero or negative) parallax is interpreted to mean that the object is on the "celestial sphere", the radius of which is an arbitrary (large) value (see the constant PXMIN). When the distance is overridden in this way, the ode GAL_DISTANCE_OVERRIDEN is set. If the space velocity is a significant fraction of c (see the constant VMAX), it is arbitrarily set to zero. When this action occurs the error code GAL_EXCESSIVE_VELOCITY is set. The relativistic adjustment involves an iterative calculation. If the process fails to converge within a set number (IMAX) of iterations, the error code GAL_NO_CONVERGENCE is set. The inverse transformation is performed by the routine gal_pvstar.

References:

Stumpff, P., Astronomy & Astrophysics 144, 232-240 (1985).

Chapter 9 - Ellipsoids

The routines detailed in this chapter are defined in the gal_frame_macros.h header file.

gal_ellipsoids.h [0.2]

This header file includes the header files of the routines that make up the ellipsoids sub-library, and defines the constants for the Ellipsoid Model identifiers.

Identifier	Ellipsoid Model
Earth	
GAL_EMEA_DEL1800	Delambre 1800
GAL_EMEA_AIRY1830	Airy 1830
GAL_EMEA_EVER1830	Everest 1830
GAL_EMEA_EVER1830BA	Everest 1830 Boni Alt
GAL_EMEA_BESL1841	Bessel 1841
GAL_EMEA_CL1866	Clarke 1866
GAL_EMEA_CL1880	Clarke 1880
GAL_EMEA_CLA1880M	Clarke 1880 Modified
GAL_EMEA_HEL1906	Helmert 1906
GAL_EMEA_INTL1909	International 1909
GAL_EMEA_KRSV	Krassovsky
GAL_EMEA_MERC1960	Mercury 1960
GAL_EMEA_WGS1960	World Geodetic System 1960
GAL_EMEA_IAU1964	IAU 1964
GAL_EMEA_AUSNAT1965	Australian National 1965
GAL_EMEA_WGS1966	World Geodetic System 1966
GAL_EMEA_MERC1968M	Modified Mercury 1968
GAL_EMEA_SA1969	South American 1969
GAL_EMEA_GRS1967	Geodetic Reference System 1967
GAL_EMEA_WGS1972	World Geodetic System 1972
GAL_EMEA_IAG1975	IAG 1975
GAL_EMEA_IAU1976	IAU 1976
GAL_EMEA_GRS1980	Geodetic Reference System 1980
GAL_EMEA_MERIT1983	MERIT 1983
GAL_EMEA_WGS1984	World Geodetic System 1984
GAL_EMEA_IERS1989	IERS 1989
GAL_EMEA_IAU1991	IAU/IAG/COSPAR 1991 Earth
GAL_EMEA_IERS2000	IERS 2000
Other Bodies	
GAL_EMME_IAU1991	IAU/IAG/COSPAR 1991 Mercury

GAL_EMVE_IAU1991	IAU/IAG/COSPAR 1991 Venus
GAL_EMMA_IAU1991	IAU/IAG/COSPAR 1991 Mars
GAL_EMJU_IAU1991	IAU/IAG/COSPAR 1991 Jupiter
GAL_EMSA_IAU1991	IAU/IAG/COSPAR 1991 Saturn
GAL_EMUR_IAU1991	IAU/IAG/COSPAR 1991 Uranus
GAL_EMNE_IAU1991	IAU/IAG/COSPAR 1991 Neptune
GAL_EMPL_IAU1991	IAU/IAG/COSPAR 1991 Pluto
GAL_EMSU_IAU1991	IAU/IAG/COSPAR 1991 Sun

It is important that the constants are used rather than the actual numerical value of the constants, as the numerical values may change between GAL releases.

Chapter 10 - Force Models

The routines detailed in this chapter are defined in the gal_gravity.h header file. Version 0.6.0 of GAL introduces many changes to the gravity sub-system. The gravity models that were previously defined in individual header files have been encapsulated into routines. The interfaces of many routines have been changed to use the GAL status recording mechanism.

gal_gm.h [0.3]

This header file defines the gravity model structures, and constants for the gravity model identifiers and status codes.

```
/* --------------------------------------------
 * Structure to store the gravity model details
 * --------------------------------------------
 */

typedef struct {

  int    body       ;  /* Solar System Body Identifier     */
  char   name[40]   ;  /* Gravity Model name               */
  double gm         ;  /* GM ( mu ) ( m^3 s^-2 )           */
  double sma        ;  /* Semi-Major Axis(meters)          */
  int    max_degree ;  /* Highest degree of coefficients   */
  int    max_order  ;  /* Highest order of coefficients    */
  int    normalized ;  /* 1 = Normalized, 0 = Unnormalized */
  double *terms     ;  /* Pointer to spherical terms       */

} gal_gm_t ;

/* -------------------------------------------------------
 * Structure to store the derivative parameters for derivs
 * -------------------------------------------------------
 */

typedef struct {
  gal_gm_t *gm        ; /* Gravity Model     */
  int      max_degree ; /* Max degree to use */
  int      max_order  ; /* Max order to use  */
} gal_derivsp_t ;

/*
 * -------------------------------------------
 * Constants for the gravity model identifiers
 * -------------------------------------------
 */

enum {

/*
```

```
 * Earth
 */

  GAL_GMEA_EGM96         = 0,
  GAL_GMEA_JGM3          = 1,
  GAL_GMEA_WGS72         = 2,
  GAL_GMEA_WGS66         = 3,

/*
 * The Moon
 */

  GAL_GMMO_GLGM1         = 4,
  GAL_GMMO_GLGM2         = 5,

/*
 * Venus
 */

  GAL_GMVE_MGNP180U      = 6,
  GAL_GMVE_MGNP120PSAAP  = 7,

/*
 * Mars
 */

  GAL_GMMA_GMM2B         = 8,
  GAL_GMMA_MGM1025       = 9,

} ;

/*
 * ------------------------------------------------------------
 * Constants for gravity model coefficients normalization state
 * ------------------------------------------------------------
 */

enum {
  GAL_UNNORMALIZED = 0,
  GAL_NORMALIZED   = 1,
} ;
```

gal_accdrag [0.5]

Computes the perturbational acceleration due to atmospheric drag

```
void
```

```
gal_accdrag
(
  double pv[2][3],
  double area,
  double mass,
  double cd,
  double p,
  double omega,
  double a[3]
) ;
```

On entry PV contains the satellite's position and velocity vectors (meters, meters per second), AREA the satellite's surface area (meters^2), MASS the satellite's mass (kilograms), CD the drag coefficient, P the atmospheric density (kilograms per meter^3), and OMEGA the planet's rotation rate (radians per second). On return A contains the acceleration vector (a=d^2r/dt^2).

References:

Satellite Orbits, Oliver Montenbruck, Eberhard Gill, Springer 2005, Pages 83-85

gal_acch [0.3]

Computes the body fixed acceleration due to the harmonic gravity field of the central body.

```
void
gal_acch
(
  double pbf[3],
  gal_gm_t *gm,
  int max_n,
  int max_m,
  double abf[3],
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

P	Position vector in body fixed frame (meters)
GM	Gravity Model
MAX_N	Maximum degree to use
MAX_M	Maximum order to use

On return ABF contains the body fixed acceleration vector. If the maximum degree and/or order to use exceeds that of the supplied gravity model then the error codes

GAL_INVALID_DEGREE or GAL_INVALID_ORDER are set in the status structure instance. If the routine fails to allocate memory for workspace then the error code GAL_ALLOC_FAILED is set in the status structure instance.

References:

Satellite Orbits, Oliver Montenbruck, Eberhard Gill, Springer 2005, Pages 61-68

gal_accpm [0.1]

Computes the perturbational acceleration due to a point mass

```
void
gal_accpm
(
  double ps[3],
  double ppm[3],
  double gm,
  double a[3]
) ;
```

On entry the parameters are set as follows:

PS	Position vector of satellite (meters)
PPM	Position vector of point mass (meters)
GM	Gravitational parameter of point mass (meters^3 per second^2)

On return A contains the acceleration vector (A = d^2r/dt^2).

References:

Satellite Orbits, Oliver Montenbruck, Eberhard Gill, Springer 2005, Pages 69-70

gal_accsrp [0.5]

Computes the perturbational acceleration due to solar radiation pressure

```
void
gal_accsrp
(
  double psat[3],
  double psun[3],
  double p,
  double au,
  double area,
  double mass,
  double eps,
  double v,
  double n[3],
  double a[3]
) ;
```

On entry the variables are set as follows:

PSAT	Position vector of satellite (meters)
PSUN	Position vector of the Sun (meters)
P	Solar radiation pressure at one AU (Newtons per meter)
AU	Length of Astronomical Unit (meters)
AREA	Satellite surface area (meters^2)
MASS	Satellite mass (kilograms)
EPS	Reflectivity
V	Shadow function (0 = eclipse, 1 = full sunlight)
N	Normal unit vector for the satellite's surface

On return A contains the acceleration vector (A = d^2r/dt^2). Reflectivity is usually in the range 0.2 to 0.9. The header file gal_astro.h defines the constant GAL_SRP96 for the solar radiation pressure at 1AU (IERS 1996).

Solar Panel	0.21
High Gain Antenna	0.30
Aluminum coated Mylar solar sail	0.88

References:

Satellite Orbits, Oliver Montenbruck, Eberhard Gill, Springer 2005, Pages 77-79

gal_accsrps [0.5]

Computes the perturbational acceleration due to solar radiation pressure using the simplified formula.

```
void
gal_accsrps
(
  double psat[3],
  double psun[3],
  double p,
  double au,
  double area,
  double mass,
  double eps,
  double v,
  double a[3]
) ;
```

On entry the variables are set as follows:

PSAT	Position vector of satellite (meters)
PSUN	Position vector of the Sun (meters)
P	Solar radiation pressure at one AU (Newtons per meter)
AU	Length of Astronomical Unit (meters)
AREA	Satellite surface area (meters^2)
MASS	Satellite mass (kilograms)
EPS	Reflectivity
V	Shadow function (0 = eclipse, 1 = full sunlight)

On return A contains the acceleration vector (A = d^2r/dt^2). Reflectivity is usually in the range 0.2 to 0.9. The header file gal_astro.h defines the constant GAL_SRP96 for the solar radiation pressure at 1AU (IERS 1996).

Solar Panel	0.21
High Gain Antenna	0.30
Aluminum coated Mylar solar sail	0.88

References:

Satellite Orbits, Oliver Montenbruck, Eberhard Gill, Springer 2005, Pages 77-79

gal_canpv [0.4]

This routine converts a pv-vector from regular to canonical units.

```
void
gal_canpv
(
  double pv1[2][3],
  double gm,
  double re,
  double pv2[2][3]
) ;
```

On entry PV1 contains the pv-vector to convert, GM contains the gravitational parameter, and RE contains the mean radius of the reference orbit. On return PV2 contains the converted position and velocity vectors in canonical units. GM and RE must be stated in consistent units, i.e. meters or kilometers based.

gal_eaadhp [0.5]

Computes the Earth's Atmospheric Density using the Harris-Priester Density Model

```
void
gal_eaadhp
(
  double p[3],
  double height,
  double alpha,
  double delta,
  double *ad,
  gal_status_t *status
) ;
```

On entry the variables are set as follows:

PV Position vector of satellite (meters, meters per second)
True-of-date inertial reference frame
HEIGHT Height of the satellite above mean sea level (meters)
ALPHA Right ascension of the Sun (radians)
DELTA Declination of the Sun (radians)

On return AD contains the atmospheric density (kilograms per meter^3). The routine sets the error code GAL_OUT_OF_RANGE if the height is out of the range model. In the out of range case AD is set to zero.

References:

Satellite Orbits, Oliver Montenbruck, Eberhard Gill, Springer 2005, Pages 89-91

gal_gmalloc [0.3]

This routine creates a blank gravity model of given degree.

```
gal_gm_t *
gal_gmalloc
(
  int n
) ;
```

On entry N contains the required degree. The routine returns a pointer to gravity model structure or NULL if failure.

gal_gmcpy [0.3]

This routine allocates memory and populates it with all or a subset of a gravity model.

```
gal_gm_t *
gal_gmcpy
(
  gal_gm_t *gm1,
  int maxn,
  int maxm,
  int norm,
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

GM1	Pointer to source gravity model structure to copy
MAXN	Maximum degree to be returned
MAXM	Maximum order to be returned
NORM	GAL_NORMALIZED means spherical terms are to be normalized GAL_UNNORMALIZED means spherical terms are to be un-normalized

The routine returns a pointer to the newly allocated model. This function additionally allows the user to limit the maximum degree and order of the coefficients to be included, useful when the full accuracy of the model is not required. If a maximum degree or order is requested greater than that provided for the base model then the higher unknown coefficients are set to zero. If the routine is unable to allocate memory then NULL is returned.

gal_gmdenorm [0.3]

This routine un-normalizes a gravity model's coefficients

```
gal_gm_t *
gal_gmdenorm
(
  gal_gm_t *gm1,
  gal_gm_t *gm2,
  gal_status_t *status
) ;
```

On entry GM1 contains the source gravity model. On return GM2 contains the unnormalized terms. The routine returns a pointer to GM2. If an internal error occurs then the applicable error code is set.

gal_gmfree [0.3]

This routine frees a gravity model previously allocated by gal_gmalloc or gal_gmcpy

```
void
gal_gmfree
(
  gal_gm_t *gm
) ;
```

On entry GM contains a pointer to the model to be deallocated.

gal_gmget [0.3]

This routine makes a copy of a selected gravity model.

```
gal_gm_t *
gal_gmget
(
  int gmi,
  int maxn,
  int maxm,
  int norm,
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

GMI Identifier of the required gravity model
MAXN Maximum degree to return
MAXM Maximum order to return
NORM GAL_NORMALIZED means spherical terms are to be normalized
GAL_UNNORMALIZED means spherical terms are to be un-normalized

The routine returns a pointer to a new copy of the required gravity model. If the gravity model identifier is unknown then the error code GAL_INVALID_ID is set. If the routine was unable to allocate memory then the error code GAL_ALLOC_FAILED is set. In the case of any error NULL is returned. The user must call gal_gmfree to de-allocate memory when finished with the gravity model. The header file gal_gm.h defines the constants for the gravity model identifiers.

gal_gmget_egm96 [0.6]

This file returns a gravity model containing the EGM96 Earth gravity model.

```
gal_gm_t *
```

```
gal_gmget_egm96
(
 int maxn,
 int maxm,
 int norm,
 gal_status_t *status
 ) ;
```

On entry the parameters are set as follows:

MAXN	Maximum degree to return
MAXM	Maximum order to return
NORM	GAL_NORMALIZED means spherical terms are to be normalized GAL_UNNORMALIZED means spherical terms are to be un-normalized

The routine returns a pointer to a new copy of the gravity model. If the routine was unable to allocate memory then NULL is returned, and the error code GAL_ALLOC_FAILED is set. The user must call gal_gmfree to de-allocate memory when finished with the gravity model. This routine replaces the functionality of gal_gmegm96.h that was included in releases of GAL prior to version 0.6.0.

References:

Lemoine F.G., Kenyon S.C., Factor J.K., Trimmer R.G., Pavlis N.K., Chinn D.S., Cox C.M., Klosko S.M., Luthcke S.B., Torrence M.H., Wang Y.M., Williamson R.G., Pavlis E.C., Rapp R.H., Olson T.R.; The Development of the Joint NASA GSFC and the National Imagery and Mapping Agency (NIMA) Geopotential Model EGM96; NASA Technical Paper NASA/TP1998206861, Goddard Space Flight Center, Greenbelt, USA, 1998

gal_gmget_glgm1 [0.6]

This file returns a gravity model containing the GLGM1 Lunar gravity model.

```
gal_gm_t *
gal_gmget_glgm1
(
 int maxn,
 int maxm,
 int norm,
 gal_status_t *status
 ) ;
```

On entry the parameters are set as follows:

MAXN	Maximum degree to return

MAXM Maximum order to return
NORM GAL_NORMALIZED means spherical terms are to be normalized
GAL_UNNORMALIZED means spherical terms are to be un-normalized

The routine returns a pointer to a new copy of the gravity model. If the routine was unable to allocate memory then NULL is returned, and the error code GAL_ALLOC_FAILED is set. The user must call gal_gmfree to de-allocate memory when finished with the gravity model. This routine replaces the functionality of gal_gmglgm1.h that was included in releases of GAL prior to version 0.6.0.

This model for the Lunar Gravity Field is derived from a tracking of Lunar Orbiters 1,2,3,4 & 5, the Apollo-15 subsatellite, and Clementine: 361,000 observations from Clementine, and 300,000 observations from the other spacecraft. The field was derived using the 1992 IAU Model for the Moon. Note that the IAU reference for this model has typographical errors for two the quantities describing the angular librations. The IAU "Table II" lists the IAU model for the orientation for the lunar pole and prime meridian. The quantities which read:

E3 = 260.008 - 13.012001*d
E5 = 357.529 - 0.985600*d

should instead read:

E3 = 260.008 + 13.012001*d
E5 = 357.529 + 0.985600*d

References:

Goddard Lunar Gravity Model-1 (GLGM-1): A 70th degree and order gravity model for the Moon, by F G Lemoine, D E Smith, and M T Zuber, P11A-9, EOS, Transactions of the American Geophysical Union Volume 75, No. 44, 1994.

Report of the IAU/IAG/COSPAR Working Group on Cartographic Coordinates and Rotational Elements of the Planets and Satellites 1991, by M E Davies, V K Abalakin, A. Brahic, M. Bursa, B H Chovitz, J H Lieske, P K Seidelmann, A T Sinclair, and Y S Tjuflin, Celestial Mechanics and Dynamical Astronomy, 53, 377-397, 1992.

gal_gmget_glgm2 [0.6]

This file returns a gravity model containing the GLGM2 Lunar gravity model.

```
gal_gm_t *
gal_gmget_glgm2
(
 int maxn,
 int maxm,
 int norm,
 gal_status_t *status
) ;
```

On entry the parameters are set as follows:

MAXN	Maximum degree to return
MAXM	Maximum order to return
NORM	GAL_NORMALIZED means spherical terms are to be normalized GAL_UNNORMALIZED means spherical terms are to be un-normalized

The routine returns a pointer to a new copy of the gravity model. If the routine was unable to allocate memory then NULL is returned, and the error code GAL_ALLOC_FAILED is set. The user must call gal_gmfree to de-allocate memory when finished with the gravity model. This routine replaces the functionality of gal_gmglgm2.h that was included in releases of GAL prior to version 0.6.0.

The field was derived using the 1992 IAU Model for the Moon. Note that the IAU reference for this model has typographical errors for two of the quantities describing the angular librations. The IAU reference "Table II" lists the IAU model for the orientation for the lunar pole and prime meridian. The quantities which read:

E3 = 260.008 - 13.012001*d
E5 = 357.529 - 0.985600*d

should instead read:

E3 = 260.008 + 13.012001*d
E5 = 357.529 + 0.985600*d

References:

Journal Geophysical Research, GLGM-2, A 70th Degree and Order Lunar Gravity Model from Clementine and Historical Data, Submitted, November 1995. by F. G. Lemoine, D. E. Smith, M.T. Zuber, G. A. Neumann, and D. D. Rowlands.

Report of the IAU/IAG/COSPAR Working Group on Cartographic Coordinates and Rotational Elements of the Planets and Satellites 1991, by M E Davies, V K Abalakin, A. Brahic, M. Bursa, B H Chovitz, J H Lieske, P K Seidelmann, A T Sinclair, and Y S Tjuflin, Celestial Mechanics and Dynamical Astronomy, 53, 377-397, 1992.

High Degree and Order Spherical Harmonic Models for the Moon from Clementine and Historic S-Band Doppler Data, 1995 XXI General Assembly, IUGG, Boulder, Colorado, July 12, 1995. by F. G. Lemoine, D. E. Smith, M. T. Zuber, and G. A. Neumann.

gal_gmget_gmm2b [0.6]

This file returns a gravity model containing the GMM2B Mars gravity model.

```
gal_gm_t *
gal_gmget_gmm2b
(
 int maxn,
 int maxm,
 int norm,
 gal_status_t *status
 ) ;
```

On entry the parameters are set as follows:

MAXN	Maximum degree to return
MAXM	Maximum order to return
NORM	GAL_NORMALIZED means spherical terms are to be normalized GAL_UNNORMALIZED means spherical terms are to be un-normalized

The routine returns a pointer to a new copy of the gravity model. If the routine was unable to allocate memory then NULL is returned, and the error code GAL_ALLOC_FAILED is set. The user must call gal_gmfree to de-allocate memory when finished with the gravity model. This routine replaces the functionality of gal_gmgmm2b.h that was included in releases of GAL prior to version 0.6.0.

This field is derived from radio tracking of the Mars Global Surveyor spacecraft; no Mariner 9 or Viking data are included. Coordinate system is IAU 1991 (Davies et al., Celestial Mechanics and Dynamical Astronomy, 53, 377-397, 1992). The model was constructed from 955,115 observations, summarized in the table below. MGS data are limited to tracking from the Aerobraking Hiatus and Science Phasing Orbit (SPO) subphases of the Orbit Insertion phase of the mission and to February 1999 to February 2000 after the orbit was circularized.

Time Periods	Arcs	Observations
Hiatus	2	24119
SPO-1	8	31001
SPO-2	16	157972
Feb-Mar 1999	9	76813
Apr 1999 - Feb 2000	47	665210
Total		955115

Orbit reconstruction was improved using Mars Orbiter Laser Altimeter (MOLA) data on 5 arcs between March and December 1999. Inter-arc and intra-arc crossovers at 21343 points were included in the orbit solutions. The gravity model was derived using a Kaula type constraint: sqrt(2)*13*10**(-5)/L**2. The analysis and results were described by F.G. Lemoine, D.D. Rowlands, D.E. Smith, D.S. Chinn, G.A. Neumann, and M.T. Zuber at the Spring Meeting of the American Geophysical Union, May 30 - June 3, 2000, Washington. DC. Further improvements to the model are expected as additional MGS data are incorporated. This Mars gravity model was produced by F.G. Lemoine under the direction of D.E. Smith of the MGS Radio Science Team.

References:

Kaula, W.M., Theory of Satellite Geodesy, Blaisdell, Waltham, MA, 1966

gal_gmget_jgm3 [0.6]

This file returns a gravity model containing the JGM-3 Earth gravity model.

```
gal_gm_t *
gal_gmget_jgm3
(
  int maxn,
  int maxm,
  int norm,
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

MAXN	Maximum degree to return
MAXM	Maximum order to return
NORM	GAL_NORMALIZED means spherical terms are to be normalized GAL_UNNORMALIZED means spherical terms are to be un-normalized

The routine returns a pointer to a new copy of the gravity model. If the routine was unable to allocate memory then NULL is returned, and the error code GAL_ALLOC_FAILED is set. The user must call gal_gmfree to de-allocate memory when finished with the gravity model. This routine replaces the functionality of gal_gmjgm3.h that was included in releases of GAL prior to version 0.6.0.

References:

Tapley B., Watkins M., Ries J., Davis G., Eanes R., Poole S., Rim H., Schutz B., Shum C., Nerem R., Lerch F., Marshall J.A., Klosko S.M., Pavlis N., Williamson R.; The Joint Gravity Model 3; Journal of Geophysical Research, Vol. 101, No. B12, S. 28029-28049, 1996

gal_gmget_mgm1025 [0.6]

This file returns a gravity model containing the MGM1025 Mars gravity model.

```
gal_gm_t *
gal_gmget_mgm1025
(
  int maxn,
  int maxm,
  int norm,
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

MAXN	Maximum degree to return
MAXM	Maximum order to return
NORM	GAL_NORMALIZED means spherical terms are to be normalized GAL_UNNORMALIZED means spherical terms are to be un-normalized

The routine returns a pointer to a new copy of the gravity model. If the routine was unable to allocate memory then NULL is returned, and the error code GAL_ALLOC_FAILED is set. The user must call gal_gmfree to de-allocate memory when finished with the gravity model. This routine replaces the functionality of gal_gmmgm1025.h that was included in releases of GAL prior to version 0.6.0.

This field is derived from radio tracking of the Mars Global Surveyor spacecraft; no Mariner 9 or Viking data are included. The MGM1025 gravity model is an update to the GMM-2B gravity model. It was determined from 155 arcs of MGS tracking data in Hiatus, SPO, GCO and Mapping. MGM1025 includes the same Mapping and GCO data as were in GMM2B; in addition, it includes data from the first half of 2001 (through July 21, 2001) when the MGS orbit orientation angle with respect to the line-of-sight (LOS) was optimum for gravity measurements. It excludes data in the vicinity of solar conjunction from May 8 to July 30 in 2000.

	GMM2B	MGM1025
Model Size	80x80	80x80
Coordinate System	IAU 1991	IAU 2000
Observations		
Hiatus	24,119	24,119
SPO-1	31,001	31,014
SPO-2	157,972	136,667
GCO	76,813	80,795

Mapping	665,210	1,352,661
TOTAL	955,155	1,625,276

Number of Arcs

Hiatus	2	2
SPO-1	8	8
SPO-2	16	14
GCO	9	9
Mapping	47	122

MGM1025 has improved correlation with topography compared with GMM-2B. The average correlation with MOLA derived topography (through degree 70) is 0.722 for GMM-2B and 0.756 for MGM1025. The new model has slightly greater power in the band from l=60 to 70. The average RMS of fit to the F2 (two-way) tracking data is 0.13 to 0.20 millimeters per second with this model, excluding arcs in the vicinity of solar conjunction. The average RMS of fit for the one-way (F1) Doppler tracking with this model is 0.10 to 0.15 millimeters per second. The one-way data contribute to solutions starting sporadically in February 2000 and more consistently in arcs starting in March of 2000. They are used solely to fill in what would otherwise be gaps in the two-way tracking Frequency biases are estimated for each pass of one-way data. The coordinate system for the model is IAU 2000 (Seidelman et al., Celestial Mechanics & Dynamical Astronomy, 82, 83-110, 2002), defined by the Mars Cartography Working Group. It includes updates to the orientation of the Mars Pole and rotation rate from a joint Pathfinder/Viking solution, and a re-determination of the location of the prime meridian (with respect to the crater Airy-0) from Mars Global Surveyor MOC and MOLA data. Pole right ascension (alpha) and declination (delta), prime meridian (Wo), and rotation rate (Wodot) in IAU 2000 are:

alpha	317.68143 deg	-0.1061 degrees per century
delta	52.88650 deg	-0.0609 degrees per century
Wo	176.630 deg	
Wdot	350.89198266 deg/day	

This Mars gravity model was produced by F.G. Lemoine under the direction of D.E. Smith of the MGS Radio Science Team.

References:

The analysis and results for MGM1025 were described by F.G. Lemoine, G.A. Neumann, D.S. Chinn, D.E. Smith, M.T. Zuber, D.D. Rowlands, D.P. Rubincam, and D.E. Pavlis in 'Solution for Mars Geophysical Parameters from Mars Global Surveyor Tracking Data', American Geophysical Union Fall Meeting 2001 (EOS, Trans. AGU 82(47), Fall Meeting Supplement, Abstract P42A-0545, F721, 2001). The GMM2B model was described by Lemoine et al., 'An Improved Solution of the Gravity Field of Mars (GMM-2B) from Mars Global Surveyor', Journal Geophysical Research, 106(E10), 23359-23376, October 25, 2001.

gal_gmget_mgnp120p [0.6]

This file returns a gravity model containing the MGNP120PSAAP Venus gravity model.

```
gal_gm_t *
gal_gmget_mgnp120p
(
  int maxn,
  int maxm,
  int norm,
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

MAXN	Maximum degree to return
MAXM	Maximum order to return
NORM	GAL_NORMALIZED means spherical terms are to be normalized GAL_UNNORMALIZED means spherical terms are to be un-normalized

The routine returns a pointer to a new copy of the gravity model. If the routine was unable to allocate memory then NULL is returned, and the error code GAL_ALLOC_FAILED is set. The user must call gal_gmfree to de-allocate memory when finished with the gravity model. This routine replaces the functionality of gal_gmmgnp120p.h that was included in releases of GAL prior to version 0.6.0.

This field is derived from radio tracking of the Magellan spacecraft. The Magellan Venus gravity model is produced by the Magellan Gravity Science Team at JPL under the direction of W.L. Sjogren. Orbits 5758 to 15019 used in the solution.

gal_gmget_mgnp180u [0.6]

This file returns a gravity model containing the MGNP180U Venus gravity model.

```
gal_gm_t *
gal_gmget_mgnp180u
(
  int maxn,
  int maxm,
  int norm,
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

MAXN	Maximum degree to return
MAXM	Maximum order to return
NORM	GAL_NORMALIZED means spherical terms are to be normalized GAL_UNNORMALIZED means spherical terms are to be un-normalized

The routine returns a pointer to a new copy of the gravity model. If the routine was unable to allocate memory then NULL is returned, and the error code GAL_ALLOC_FAILED is set. The user must call gal_gmfree to de-allocate memory when finished with the gravity model. This routine replaces the functionality of gal_gmmgnp180u.h that was included in releases of GAL prior to version 0.6.0.

This field is derived from radio tracking of the Magellan spacecraft. The Magellan Venus gravity model is produced by the Magellan Gravity Science Team at JPL under the direction of W.L. Sjogren. Orbits 5758 to 15019 used in the solution.

gal_gmget_wgs66 [0.6]

This file returns a gravity model containing the WGS66 Earth gravity model.

```
gal_gm_t *
gal_gmget_wgs66
(
  int maxn,
  int maxm,
  int norm,
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

MAXN	Maximum degree to return
MAXM	Maximum order to return
NORM	GAL_NORMALIZED means spherical terms are to be normalized GAL_UNNORMALIZED means spherical terms are to be un-normalized

The routine returns a pointer to a new copy of the gravity model. If the routine was unable to allocate memory then NULL is returned, and the error code GAL_ALLOC_FAILED is set. The user must call gal_gmfree to de-allocate memory when finished with the gravity model. This routine replaces the functionality of gal_gmwgs66.h that was included in releases of GAL prior to version 0.6.0.

The value for GM is unknown, so the value for WGS72 is used instead.

gal_gmget_wgs72 [0.6]

This file returns a gravity model containing the WGS72 Earth gravity model.

```
gal_gm_t *
gal_gmget_wgs72
(
  int maxn,
  int maxm,
  int norm,
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

MAXN	Maximum degree to return
MAXM	Maximum order to return
NORM	GAL_NORMALIZED means spherical terms are to be normalized GAL_UNNORMALIZED means spherical terms are to be un-normalized

The routine returns a pointer to a new copy of the gravity model. If the routine was unable to allocate memory then NULL is returned, and the error code GAL_ALLOC_FAILED is set. The user must call gal_gmfree to de-allocate memory when finished with the gravity model. This routine replaces the functionality of gal_gmwgs72.h that was included in releases of GAL prior to version 0.6.0.

gal_gmlda [0.6]

This routine loads a gravity model from an ASCII file in ICGEM format.

```
gal_gm_t *
gal_gmlda
(
  FILE *fp,
  int maxn,
  int maxm,
  int norm,
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

FP	Pointer to open file handle
MAXN	Maximum degree to return
MAXM	Maximum order to return
NORM	GAL_NORMALIZED means spherical terms are to be normalized GAL_UNNORMALIZED means spherical terms are to be un-normalized

The ICGEM-format accommodates Earth Gravity Field models in terms of spherical harmonic coefficients and Ocean and Atmosphere Tides. This routine only handles the spherical harmonic coefficients. Each individual data file consists of two sections: The header containing parameters which do not depend on degree and order. The end of the header is marked by the keyword "end_of_head" (as a separator between header and data section). The data section with the list of degree- and order-dependent parameters. The records have the following basic structure: The basic structure of the record lines is unformatted, i.e. separators are blanks and/or tabs. Each record consists of one keyword followed by one or more parameters (numbers or characters), which are separated by one or an arbitrary number of blanks and/or tabs. The number of parameters depends on the corresponding keyword as defined below. There are mandatory and optional records. All lines led by non-defined keywords are comments. In any line, additional characters and/or numbers beyond the last parameter are allowed as comments. Leading and trailing blanks are ignored. This routine extends the ICGEM format with the addition of the "body" keyword. Valid values for this keyword are the GAL Object Identifier codes. This routine assumes that the object is the Earth. If the file contains body identifier in the header then that will be used. Many gravity models can be downloaded from the International Centre for Global Earth Models (ICGEM).

http://icgem.gfz-potsdam.de/ICGEM/ICGEM.html

References:

The ICGEM-format by Franz Barthelmes and Christoph Förste, GFZ Potsdam, Department 1 “Geodesy and Remote Sensing” , 2006 February 26

gal_gmnorm [0.3]

This routine normalizes a gravity model's coefficients

```
gal_gm_t *
gal_gmnorm
(
  gal_gm_t *gm1,
  gal_gm_t *gm2,
  gal_status_t *status
) ;
```

On entry GM1 contains the source gravity model. On return GM2 contains the normalized coefficients. The routine returns a pointer to GM2. This routine depends upon the gal_factorial routine. Recent gravity models (e.g. EGM 2008) have degrees and orders that require the computation of factorials that exceed the largest possible value that can be represented as an IEEE long double type. Any errors recorded by gal_factorial are returned to the user in the status structure instance.

gal_gmsva [0.6]

This routine saves a gravity model to an ASCII file in ICGEM format.

```
void
gal_gmsva
(
  FILE *fp,
  gal_gm_t *gm,
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

FP	Pointer to open ASCII file
GM	Pointer to gravity model structure

The ICGEM-format accommodates Earth Gravity Field models in terms of spherical harmonic coefficients and Ocean and Atmosphere Tides. This routine only handles the spherical harmonic coefficients. Each individual data file consists of two sections: The header containing parameters which do not depend on degree and order. The end of the header is marked by the keyword "end_of_head" (as a separator between header and data section). The data section with the list of degree- and order-dependent parameters. The records have the following basic structure: The basic structure of the record lines is unformatted, i.e. separators are blanks and/or tabs. Each record consists of one keyword followed by one or more parameters (numbers or characters), which are separated by one or an arbitrary number of blanks and/or tabs. The number of parameters depends on the corresponding keyword as defined below. There are mandatory and optional records. All lines led by non-defined keywords are comments. In any line, additional characters and/or numbers beyond the last parameter are allowed as comments. Leading and trailing blanks are ignored. This routine extends the ICGEM format with the addition of the "body" keyword. Valid values for this keyword are the GAL Object Identifier codes.

References:

The ICGEM-format by Franz Barthelmes and Christoph Förste, GFZ Potsdam, Department 1 “Geodesy and Remote Sensing” , 2006 February 26

gal_gmuzh [0.3]

This routine calculates an un-normalized zonal harmonic

```
double
gal_gmuzh
(
  gal_gm_t *gm,
  gal_facexp_t *facexp,
  int harmonic,
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

GM	Pointer to gravity model
FACEXP	Pointer to factorial exponent lookup table
HARMONIC	Required harmonic

The routine returns the required un-normalized zonal harmonic. If the requested harmonic exceeds the maximum degree of the gravity model then the error code GAL_INVALID_DEGREE is set.

gal_stget [0.3]

This routine gets spherical terms C & S of degree n and order m from the given gravity model

```
void
gal_stget
(
  const int n,
  const int m,
  gal_gm_t *gm,
  double *c,
  double *s,
  gal_status_t *status
) ;
```

On entry N contains the required degree, and M the required order, gm is a pointer to the gravity model structure. On return C and S contain the C and S coefficients of degree N and order M. If the required degree or order exceed the maximum of the gravity model then the error code GAL_INVALID_DEGREE and/or GAL_INVALID_ORDER are set.

gal_stnf [0.3]

This function computes the spherical terms normalization factor.

```
double
gal_stnf
(
  gal_facexp_t *facexp,
  const int n,
  const int m,
  gal_status_t *status
) ;
```

On entry FACEXP contains a pointer to the factorial exponent lookup table, N the required degree, and M the required order. The routine returns the normalization factor. This routine depends upon the gal_factorial routine. Recent gravity models (e.g. EGM 2008) have degrees and orders that require the computation of factorials that exceed the largest possible value that can be represented as an IEEE long double type. Any errors recorded by gal_factorial are returned to the user in the status structure instance.

References:

Fundamentals of Astrodynamics and Applications by David A. Vallado, Second Section, Second Pressing Pages 519-520

gal_stset [0.3]

This routine sets spherical terms C & S of degree N and order M in the given gravity model

```
void
gal_stset
(
  const int n,
  const int m,
  const double c,
  const double s,
  gal_gm_t *gm,
  gal_status_t *status
) ;
```

On entry N contains the required degree, M the required order, C and S contain the values to store in the gravity model. On return the spherical terms of the gravity model GM have been updated. If the required degree or order exceed the maximum of the gravity model then the error code GAL_INVALID_DEGREE and / or GAL_INVALID_ORDER are set.

gal_stunf [0.3]

This function computes the spherical terms un-normalization factor.

```
double
```

```
gal_stunf
(
  gal_facexp_t *facexp,
  const int n,
  const int m,
  gal_status_t *status
) ;
```

On entry FACEXP contains a pointer to the factorial exponent lookup table, N contains the required degree, and M the required order. The routine returns the un-normalization factor of degree N and order M. Recent gravity models (e.g. EGM 2008) have degrees and orders that require the computation of factorials that exceed the largest possible value that can be represented as an IEEE long double type. Any errors recorded by gal_factorial are returned to the user in the status structure instance.

References:

Fundamentals of Astrodynamics and Applications by David A. Vallado, Second Section, Second Pressing Pages 519-520

gal_tu [0.4]

This routine computes the canonical unit TU factor from the mean radius and gravitational parameter.

```
double
gal_tu
(
  double gm,
  double re
) ;
```

On entry GM contains the gravitational parameter, and RE the mean radius of the reference orbit. The routine returns the TU factor. GM and RE must be stated in consistent units, i.e. meters or kilometers based.

gal_uncanpv [0.4]

This routine converts a pv-vector from canonical units to regular units

```
void
gal_uncanpv
(
  double pv1[2][3],
  double gm,
  double re,
  double pv2[2][3]
) ;
```

On entry PV1 contains the position and velocity vectors in canonical units, GM contains the gravitational parameter, and RE the mean radius of the reference orbit. On return PV2 contains the position and velocity vectors in regular units. GM and RE must be stated in consistent units, i.e. meters or kilometers based.

Chapter 11 – USSTRATCOM

The routines detailed in this chapter are defined in the gal_usstratcom.h header file. The header file also defines the following structures and constants:

```
/*
 * Two-Line Element Structure
 */

struct tle_node {
  int             satnum         ; /* US Space Command Object Number              */
  char            classification ; /* Security Classification                     */
  char            intldesg[12]   ; /* International Designator (COSPAR/WDC-A-R&S) */
  int             epochyr        ; /* Epoch Year                                  */
  double          epochdays      ; /* Epoch Day of Year (plus Fraction)           */
  double          ndot           ; /* Mean motion derivative (rev/day /2)         */
  double          nddot          ; /* Mean motion second derivative (rev/day2 /6) */
  double          bstar          ; /* Bstar / Drag Term                           */
  int             ephtype        ; /* Ephermeris Type                             */
  int             setnum         ; /* Element set number                          */
  double          inclo          ; /* Inclination                                 */
  double          nodeo          ; /* Right Ascension of Ascending Node (deg)     */
  double          ecco           ; /* Eccentricity                                */
  double          argpo          ; /* Argument of Perigee (deg)                   */
  double          mo             ; /* Mean Anamaly (deg)                          */
  double          no             ; /* Mean Motion (rev/day)                       */
  int             revnum         ; /* Epoch Revolution Number                     */
  struct tle_node *next          ; /* Pointer to next TLE                         */
} ;

typedef struct tle_node gal_tle_t ;

/*
 * Piece of launch definition structure
 */

struct piece_node {
  char              piece[4]     ; /* Piece of launch                             */
  int               satnum       ; /* US Space Command Object Number              */
  char              intldesg[12] ; /* International Designator (COSPAR/WDC-A-R&S) */
  char              name[40]     ; /* Description of piece of launch              */
  double            launch_date1 ; /* Launch date part 1                          */
  double            launch_date2 ; /* Launch date part 2                          */
  int               status       ; /* Orbital Status                              */
  double            decay_date1  ; /* Decay date part 1                           */
  double            decay_date2  ; /* Decay date part 2                           */
  double            period       ; /* Period                                      */
  double            inclo        ; /* Inclination                                 */
  double            apogee       ; /* Apogee                                      */
  double            perigee      ; /* Perigee                                     */
  double            rcs          ; /* RADAR cross section                         */
  int               owner_code   ; /* GAL Catalog Object Owner Code               */
  int               payload      ; /* Payload = 1, Debris = 0                     */
  gal_tle_t         *tle         ; /* Pointer to TLE list                         */
  int               tcount       ; /* Number of TLEs                              */
 int               tag          ; /* Tag; 1 = tagged, 0 = not tagged             */

  struct piece_node *next        ; /* Pointer to next piece of launch             */
} ;

typedef struct piece_node gal_piece_t ;
```

```
/*
 * Launch definition structure
 */

struct launch_node {
  int                year         ; /* Year of launch                              */
  int                number       ; /* Launch number of year                       */
  double             launch_date1 ; /* Date of launch part 1                       */
  double             launch_date2 ; /* Date of launch part 2                       */
  int                lv_code      ; /* GAL Launch Vehicle Code                     */
  gal_piece_t        *piece       ; /* Pointer to pieces of launch                 */
  int                pcount       ; /* Number of pieces of launch                  */
  struct launch_node *next        ; /* Pointer to next launch                      */
} ;

typedef struct launch_node gal_launch_t ;

/*
 * Object catalog structure
 */

typedef struct {
  gal_launch_t *launch           ; /* Pointer to launches                         */
  int          lcount            ; /* Count of number of launches                 */
  gal_piece_t  *index            ; /* Pointer to lookup table                     */
  int          icount            ; /* Number of entries in index                  */
} gal_objcat_t ;

/* ---------------------------
 * Catalog Object Status Codes
 * ---------------------------
 */

enum {
  GAL_COS_UNKNOWN               =  0, /* Unknown Status */
  GAL_COS_BARYCENTRIC_ORBIT_EMB =  1,
  GAL_COS_CIRCUMLUNAR           =  2,
  GAL_COS_DECAYED               =  3,
  GAL_COS_ESCAPED_SOLAR_SYSTEM  =  4,
  GAL_COS_GEOCENTRIC_ORBIT      =  5,
  GAL_COS_HELIOCENTRIC_ORBIT    =  6,
  GAL_COS_LUNAR_IMPACT          =  7,
  GAL_COS_LUNAR_LANDING         =  8,
  GAL_COS_MARS_IMPACT           =  9,
  GAL_COS_MARS_ORBIT            = 10,
  GAL_COS_NO_ELEMENTS_AVAILABLE = 11,
  GAL_COS_NO_INITIAL_ELEMENTS   = 12,
  GAL_COS_SELENOCENTRIC_ORBIT   = 13,
  GAL_COS_VENUS_IMPACT          = 14,
  GAL_COS_VENUS_LANDING         = 15,
  GAL_COS_VENUS_ORBIT           = 16
} ;

/* --------------------------
 * Catalog Object Owner Codes
 * --------------------------
 */

enum {
  GAL_COO_UNKNOWN               =  0,
  GAL_COO_ALGERIA               =  1,
  GAL_COO_ARABSAT               =  2,
  GAL_COO_ARGENTINA             =  3,
  GAL_COO_ASIASAT_CORP          =  4,
  GAL_COO_AUSTRALIA             =  5,
  GAL_COO_BERMUDA               =  6,
  GAL_COO_BRAZIL                =  7,
  GAL_COO_CANADA                =  8,
  GAL_COO_CHILE                 =  9,
  GAL_COO_CIS                   = 10,
  GAL_COO_COLUMBIA              = 11,
  GAL_COO_CZECHOSLOVAKIA        = 12,
  GAL_COO_DENMARK               = 13,
  GAL_COO_EGYPT                 = 14,
  GAL_COO_ESA                   = 15,
  GAL_COO_ESRO                  = 16,
  GAL_COO_EUMETSAT              = 17,
  GAL_COO_EUTELSAT              = 18,
  GAL_COO_FRANCE                = 19,
  GAL_COO_FRANCE_GERMANY        = 20,
  GAL_COO_GERMANY               = 21,
  GAL_COO_GLOBALSTAR            = 22,
  GAL_COO_GREECE                = 23,
  GAL_COO_INDIA                 = 24,
  GAL_COO_INDONESIA             = 25,
  GAL_COO_INMARSAT              = 26,
  GAL_COO_INTELSAT              = 27,
  GAL_COO_IRAN                  = 28,
  GAL_COO_IRIDIUM               = 29,
  GAL_COO_ISRAEL                = 30,
  GAL_COO_ISRO                  = 31,
  GAL_COO_ISS                   = 32,
  GAL_COO_ITALY                 = 33,
  GAL_COO_JAPAN                 = 34,
  GAL_COO_KOREA                 = 35,
  GAL_COO_LUXEMBOURG            = 36,
  GAL_COO_MALAYSIA              = 37,
  GAL_COO_MEXICO                = 38,
  GAL_COO_NATO                  = 39,
  GAL_COO_NETHERLANDS           = 40,
  GAL_COO_NEW_ICO               = 41,
  GAL_COO_NIGERIA               = 42,
  GAL_COO_NORTH_KOREA           = 43,
  GAL_COO_NORWAY                = 44,
  GAL_COO_PAKISTAN              = 45,
  GAL_COO_PHILIPPINES           = 46,
  GAL_COO_PORTUGAL              = 47,
  GAL_COO_PRC                   = 48,
  GAL_COO_PRC_BRAZIL            = 49,
  GAL_COO_PRC_ESA               = 50,
  GAL_COO_RASC                  = 51, /* Need to find out what this is */
  GAL_COO_SAUDI_ARABIA          = 52,
  GAL_COO_SEA_LAUNCH            = 53,
  GAL_COO_SINGPORE_TAIWAN       = 54,
  GAL_COO_SOUTH_AFRICA          = 55,
  GAL_COO_SOUTH_KOREA           = 56,
  GAL_COO_SPAIN                 = 57,
  GAL_COO_SWEDEN                = 58,
  GAL_COO_SWITZERLAND           = 59,
  GAL_COO_TAIWAN                = 60,
  GAL_COO_THAILAND              = 61,
  GAL_COO_TURKEY                = 62,
  GAL_COO_UNITED_ARAB_EMIRATES  = 63,
  GAL_COO_UNITED_KINGDOM        = 64,
  GAL_COO_USA                   = 65,
  GAL_COO_USA_BRAZIL            = 66,
  GAL_COO_VENEZUELA             = 67,
  GAL_COO_VIETNAM               = 68
} ;

#define GAL_MAX_OWNERS ( GAL_COO_VIETNAM + 1 )

/* ---------------------------
 * Launch vehicle identifiers
 * ---------------------------
 */

enum {
    GAL_LV_UNKNOWN                                  = 0,
    GAL_LV_APEX                                     = 1,
    GAL_LV_ARIANE_1                                 = 2,
    GAL_LV_ARIANE_2                                 = 3,
    GAL_LV_ARIANE_3                                  = 4,
    GAL_LV_ARIANE_40                                 = 5,
    GAL_LV_ARIANE_40PLUS                             = 6,
    GAL_LV_ARIANE_40PLUS3                            = 7,
    GAL_LV_ARIANE_42L                                = 8,
    GAL_LV_ARIANE_42LPLUS3                           = 9,
    GAL_LV_ARIANE_42P                                = 10,
    GAL_LV_ARIANE_42PPLUS                            = 11,
    GAL_LV_ARIANE_42PPLUS3                           = 12,
    GAL_LV_ARIANE_44L                                = 13,
    GAL_LV_ARIANE_44LPLUS                            = 14,
    GAL_LV_ARIANE_44LPLUS3                           = 15,
    GAL_LV_ARIANE_44LP                               = 16,
    GAL_LV_ARIANE_44LPPLUS                           = 17,
    GAL_LV_ARIANE_44LPPLUS3                          = 18,
    GAL_LV_ARIANE_44P                                = 19,
    GAL_LV_ARIANE_44PPUS3                            = 20,
    GAL_LV_ARIANE_5                                  = 21,
    GAL_LV_ASLV_D3                                   = 22,
    GAL_LV_ASLV_D4                                   = 23,
    GAL_LV_ATHENA_1_OAM                              = 24,
    GAL_LV_ATHENA_2                                  = 25,
    GAL_LV_ATLAS                                     = 26,
    GAL_LV_ATLAS_1_CENTAUR                           = 27,
    GAL_LV_ATLAS_14E                                 = 28,
    GAL_LV_ATLAS_2_CENTAUR                           = 29,
    GAL_LV_ATLAS_28E                                 = 30,
    GAL_LV_ATLAS_2A                                  = 31,
    GAL_LV_ATLAS_2A_CENTAUR                          = 32,
    GAL_LV_ATLAS_2AS_CENTAUR                         = 33,
    GAL_LV_ATLAS_34F                                 = 34,
    GAL_LV_ATLAS_35F                                 = 35,
    GAL_LV_ATLAS_3A_CENTAUR                          = 36,
    GAL_LV_ATLAS_3B_CENTAUR                          = 37,
    GAL_LV_ATLAS_41E                                 = 38,
    GAL_LV_ATLAS_42E                                 = 39,
    GAL_LV_ATLAS_5_CENTAUR                           = 40,
    GAL_LV_ATLAS_5_STAR_48_B                         = 41,
    GAL_LV_ATLAS_55E                                 = 42,
    GAL_LV_ATLAS_67F                                 = 43,
    GAL_LV_ATLAS_75E                                 = 44,
    GAL_LV_ATLAS_AGENA                               = 45,
    GAL_LV_ATLAS_AGENA_A                             = 46,
    GAL_LV_ATLAS_AGENA_B                             = 47,
    GAL_LV_ATLAS_AGENA_D                             = 48,
    GAL_LV_ATLAS_CENTAUR                             = 49,
    GAL_LV_ATLAS_CENTAUR_2                           = 50,
    GAL_LV_ATLAS_D                                   = 51,
    GAL_LV_ATLAS_D_5                                 = 52,
    GAL_LV_ATLAS_D_7                                 = 53,
    GAL_LV_ATLAS_F                                   = 54,
    GAL_LV_ATLAS_F_BURNER_2                          = 55,
    GAL_LV_ATLAS_H                                   = 56,
    GAL_LV_BLACK_ARROW                               = 57,
    GAL_LV_BLOCK_DM                                  = 58,
    GAL_LV_BLOCK_DM_SL                               = 59,
    GAL_LV_BREEZE_KM                                 = 60,
    GAL_LV_BREEZE_M                                  = 61,
    GAL_LV_CELESTIS_01_PEGASUS                       = 62,
    GAL_LV_CELESTIS_02_TAURUS                        = 63,
    GAL_LV_CELESTIS_03_TAURUS                        = 64,
    GAL_LV_CZ_1                                      = 65,
    GAL_LV_CZ_2                                      = 66,
    GAL_LV_CZ_2B                                     = 67,
    GAL_LV_CZ_2C                                     = 68,
    GAL_LV_CZ_2D                                     = 69,
    GAL_LV_CZ_2E                                     = 70,
    GAL_LV_CZ_2F                                     = 71,
    GAL_LV_CZ_3                                      = 72,
    GAL_LV_CZ_3A                                     = 73,
    GAL_LV_CZ_3B                                     = 74,
    GAL_LV_CZ_3C                                    = 75,
    GAL_LV_CZ_4                                     = 76,
    GAL_LV_CZ_4B                                    = 77,
    GAL_LV_CZ_4C                                    = 78,
    GAL_LV_CZ_5                                     = 79,
    GAL_LV_DELTA                                    = 80,
    GAL_LV_DELTA_1                                  = 81,
    GAL_LV_DELTA_1_PLUSCAMEO                        = 82,
    GAL_LV_DELTA_1_CEP_1                            = 83,
    GAL_LV_DELTA_1_ITOS_B                           = 84,
    GAL_LV_DELTA_2                                  = 85,
    GAL_LV_DELTA_2_DUVE                             = 86,
    GAL_LV_DELTA_3                                  = 87,
    GAL_LV_DELTA_4                                  = 88,
    GAL_LV_DELTA_4H                                 = 89,
    GAL_LV_DIAMANT                                  = 90,
    GAL_LV_DIAMANT_MIKA                             = 91,
    GAL_LV_DIAMANT_B                                = 92,
    GAL_LV_DIAMANT_B_P4                             = 93,
    GAL_LV_DNEPR_1                                  = 94,
    GAL_LV_FALCON_1                                 = 95,
    GAL_LV_FREGAT                                   = 96,
    GAL_LV_GSLV                                     = 97,
    GAL_LV_H_1                                      = 98,
    GAL_LV_H_1_MABES                                = 99,
    GAL_LV_H_2                                      = 100,
    GAL_LV_H_2A                                     = 101,
    GAL_LV_H_2B                                     = 102,
    GAL_LV_IABS                                     = 103,
    GAL_LV_IRIS                                     = 104,
    GAL_LV_IRS_P2                                   = 105,
    GAL_LV_IUS                                      = 106,
    GAL_LV_JUNO_II                                  = 107,
    GAL_LV_M_3C                                     = 108,
    GAL_LV_M_3H                                     = 109,
    GAL_LV_M_3S                                     = 110,
    GAL_LV_M_3S2                                    = 111,
    GAL_LV_M_4S                                     = 112,
    GAL_LV_M_5                                      = 113,
    GAL_LV_MARS_96                                  = 114,
    GAL_LV_MINOTAUR                                 = 115,
    GAL_LV_N_1                                      = 116,
    GAL_LV_N_2                                      = 117,
    GAL_LV_OPTUS_B1_STAR_63F                        = 118,
    GAL_LV_PAM_S                                    = 119,
    GAL_LV_PEGASUS                                  = 120,
    GAL_LV_PSLV                                     = 121,
    GAL_LV_PSLV_1C                                  = 122,
    GAL_LV_SAFIR_2                                  = 123,
    GAL_LV_SATURN_1                                 = 124,
    GAL_LV_SATURN_1B                                = 125,
    GAL_LV_SATURN_5                                 = 126,
    GAL_LV_SCOUT_A                                  = 127,
    GAL_LV_SCOUT_A_1                                = 128,
    GAL_LV_SCOUT_B                                  = 129,
    GAL_LV_SCOUT_B_1                                = 130,
    GAL_LV_SCOUT_D_1                                = 131,
    GAL_LV_SCOUT_D_1_D                              = 132,
    GAL_LV_SCOUT_E_1                                = 133,
    GAL_LV_SCOUT_F_1                                = 134,
    GAL_LV_SCOUT_G_1                                = 135,
    GAL_LV_SCOUT_X_1                                = 136,
    GAL_LV_SCOUT_X_2M                               = 137,
    GAL_LV_SCOUT_X_3                                = 138,
    GAL_LV_SCOUT_X_3M                               = 139,
    GAL_LV_SCOUT_X_4                                = 140,
    GAL_LV_SHAVIT                                   = 141,
    GAL_LV_SL_04                                    = 142,
    GAL_LV_SL_1                                     = 143,
    GAL_LV_SL_11                                    = 144,
    GAL_LV_SL_12                                    = 145,
    GAL_LV_SL_13                                    = 146,
    GAL_LV_SL_14                                    = 147,
    GAL_LV_SL_16                                    = 148,
    GAL_LV_SL_18                                    = 149,
    GAL_LV_SL_19                                    = 150,
    GAL_LV_SL_21                                    = 151,
    GAL_LV_SL_23                                    = 152,
    GAL_LV_SL_24                                    = 153,
    GAL_LV_SL_26                                    = 154,
    GAL_LV_SL_3                                     = 155,
    GAL_LV_SL_4                                     = 156,
    GAL_LV_SL_5                                     = 157,
    GAL_LV_SL_6                                     = 158,
    GAL_LV_SL_7                                     = 159,
    GAL_LV_SL_8                                     = 160,
    GAL_LV_SL_9                                     = 161,
    GAL_LV_SLV_3                                    = 162,
    GAL_LV_SS_18                                    = 163,
    GAL_LV_SSN_23                                   = 164,
    GAL_LV_STRELA                                   = 165,
    GAL_LV_TAURUS                                   = 166,
    GAL_LV_THOR_ABLE                                = 167,
    GAL_LV_THOR_ABLESTAR                            = 168,
    GAL_LV_THOR_AGENA_B                             = 169,
    GAL_LV_THOR_AGENA_D                             = 170,
    GAL_LV_THOR_ALTAIR                              = 171,
    GAL_LV_THOR_BURNER_2                            = 172,
    GAL_LV_THOR_BURNER_2A                           = 173,
    GAL_LV_THOR_DELTA_1                             = 174,
    GAL_LV_THORAD_AGENA_D                           = 175,
    GAL_LV_THORAD_DELTA_1                           = 176,
    GAL_LV_TITAN_2                                  = 177,
    GAL_LV_TITAN_2G                                 = 178,
    GAL_LV_TITAN_3                                  = 179,
    GAL_LV_TITAN_34B                                = 180,
    GAL_LV_TITAN_34B_AGENA                          = 181,
    GAL_LV_TITAN_34B_AGENA_D                        = 182,
    GAL_LV_TITAN_34D                                = 183,
    GAL_LV_TITAN_34D_AGENA_D                        = 184,
    GAL_LV_TITAN_34D_IUS                            = 185,
    GAL_LV_TITAN_3A_TRANSTAGE                       = 186,
    GAL_LV_TITAN_3B_AGENA                           = 187,
    GAL_LV_TITAN_3C                                 = 188,
    GAL_LV_TITAN_3C_TRANSTAGE                       = 189,
    GAL_LV_TITAN_3D                                 = 190,
    GAL_LV_TITAN_3E                                 = 191,
    GAL_LV_TITAN_3E_CENTAUR                         = 192,
    GAL_LV_TITAN_4                                  = 193,
    GAL_LV_TITAN_4_CENTAUR                          = 194,
    GAL_LV_TITAN_401_CENTAUR                        = 195,
    GAL_LV_TITAN_4A                                 = 196,
    GAL_LV_TITAN_4A_CENTAUR                         = 197,
    GAL_LV_TITAN_4B                                 = 198,
    GAL_LV_TITAN_4B_CENTAUR                         = 199,
    GAL_LV_TOS                                      = 200,
    GAL_LV_VANGUARD                                 = 201
} ;

#define GAL_MAX_LAUNCHERS ( GAL_LV_VANGUARD + 1 )
```

gal_launch_ins [0.6]

This routine inserts a launch structure into an object catalog.

```
gal_launch_t *
gal_launch_ins
(
  gal_objcat_t *objcat,
  int year,
  int number,
  gal_status_t *status
) ;
```

On entry OBJCAT points to an object catalog structure created previously by gal_objcat_alloc. YEAR is set to the year of the launch, and NUMBER to the number of the launch within the year. The routine returns a pointer to the launch structure. If the launch is already in the catalog then a pointer is returned to the existing structure. i.e. no duplicates are created. If an internal error occurs then the error code GAL_ALLOC_FAILED is set and the routine returns NULL. The launches are stored in a linked list within the catalog, the order is most recent first.

gal_ldssr [0.6]

This routine loads an object catalog from a US Strategic Command Satellite Situation Report file.

```
void
gal_ldssr
(
  FILE         *fp,
  gal_objcat_t *objcat,
  gal_status_t *status
) ;
```

On entry FP points to an open file, and OBJCAT points to an object catalog structure created previously by gal_objcat_alloc. If an error occurs then the applicable error code is set. The Satellite Situation Report is a listing of those satellites (objects) currently in orbit and those which have previously orbited the Earth. Some objects are too small or too far from the Earth's surface to be detected; therefore, the Satellite Situation Report does not include all man-made objects orbiting the Earth. The latest report can be downloaded from here:

http://www.space-track.org/perl/login.pl

gal_lvcode [0.6]

This routine returns the GAL identifer code of a launch vehicle.

```
int
gal_lvcode
(
  char *name
) ;
```

On entry NAME points to an object name, the routine returns the GAL launch vehicle identifier associated with the object name. If the object name is unknown then the routine returns GAL_LV_UNKNOWN. The header file “gal_usstratcom.h” defines the GAL_LV_* constants.

gal_lvname [0.6]

This routine returns the name of a launch vehicle.

```
char *
gal_lvname
(
  int launcher,
  char *name,
  gal_status_t *status
) ;
```

On entry LAUNCHER contains the GAL identifier code of the required launch vehicle. On return NAME contains the name of the launch vehicle, and the routine returns a pointer to NAME. The header file "gal_usstratcom.h" defines the applicable GAL_LV_* constants. If the specified identifier code is not known to the routine then the error code GAL_INVALID_ID is set, and the routine returns NULL.

gal_objcat_alloc [0.6]

This routine creates an empty object catalog instance.

```
gal_objcat_t *
gal_objcat_alloc
(
) ;
```

If the allocation fails then NULL is returned. Object catalog instances must be de-allocated using the gal_objcat_free routine.

gal_objcat_export [0.6]

This routine exports the contents of an object catalog structure to a fixed format ASCII file.

```
void
gal_objcat_export
(
  FILE *fp,
  gal_objcat_t *objcat,
  gal_status_t *status
) ;
```

On entry FP points to an open file, and OBJCAT points to an object catalog instance. The routine exports the contents of the object catalog to a fixed format ASCII file. The sort order is most recent launch first, then piece of launch. The format is as follows:

%-11s	International designation
%8i	US Strategic Command Object Number
%-40s	Object name
%8i	GAL Launch Vehicle Identifier Code
%8i	GAL Owner Identifier Code
%8i	Payload flag; 1 means payload, 0 means debris
%8i	Status code
%20.12e	Launch date part 1 (JD UTC)
%20.12e	Launch date part 2 (JD UTC)
%20.12e	Decay date part 1 (JD UTC)
%20.12e	Decay date part 2 (JD UTC)
%20.12e	Period
%20.12e	Inclination
%20.12e	Apogee
%20.12e	Perigee
%20.12e	RADAR cross section

Constants for the identifier codes are defined in “gal_usstratcom.h”.

gal_objcat_free [0.6]

This routine frees an object catalog instance.

```
void
gal_objcat_free
(
  gal_objcat_t *objcat
) ;
```

On entry OBJCAT points to the object catalog instance to de-allocate.

gal_objcat_init [0.6]

This routine initializes an object catalog instance.

```
void
gal_objcat_free
(
  gal_objcat_t *objcat
) ;
```

On entry OBJCAT points to the object catalog instance to initialize. If the object catalog contains any launches, piece of launches, or TLEs then these are removed and de-allocated. The object catalog instance is returned to the same state as it was when created by gal_objcat_alloc.

gal_objowner_code [0.6]

Returns the code of a catalog object's owner organization or country for the specified US Stategic Command abbreviation code.

```
int
gal_objowner_code
(
  char *owner
) ;
```

On entry OWNER contains a USSTRATCOM object owner abbreviation code. The routine returns the GAL Object Owner Code. If the OWNER abbrevation is unknown to the routine then GAL_UNKNOWN_OWNER is returned. The header file “gal_usstratcom.h” contains the GAL_COO_* constants.

gal_objowner_name [0.6]

Returns the name of a catalog object's owner organization or country for the specified GAL Object Owner Code.

```
char *
gal_objowner_name
(
  int owner,
  char *name,
  gal_status_t *status
) ;
```

On entry OWNER contains the GAL Object Owner Code for the required country or organization. The routine returns a pointer to NAME which now contains the owner name. The header file "gal_usstratcom.h" defines the applicable GAL_COO_* constants. If the specified identifier code is not known to the routine then the error code

GAL_INVALID_ID is set, and the routine returns NULL.

gal_osc2mean [0.6]

This routine converts osculating Keplerian elements to mean elements suitable for use with the SGP4 orbit propagator.

```
void
gal_osc2mean
(
  double epoch1,
  double epoch2,
  double osc[6],
  gal_gm_t *gm,
  double mean[6],
  gal_status_t *status
) ;
```

On entry EPOCH1 and EPOCH2 contain the UTC date in standard SOFA two-piece format. OSC contains the osculating elements; the elements are stored as follows:

[0] - Argument of perigee (degrees)
[1] - Right ascension of ascending node (degrees)
[2] - Inclination (degrees)
[3] - Mean motion (rev / day)
[4] - Eccentricity
[5] - Mean anomaly (degrees)

On return MEAN contains the mean elements.

The conversion process will not always converge to a solution. If it does not converge then the error code GAL_NO_CONVERGENCE is set. Testing against the complete unclassified catalog (as of January 2009) it was found that out of 11766 objects, 11103 (~94%) converged and matched the NORAD values exactly, 9 (~0.07%) objects converged but did not exactly match the NORAD values, and 654 (~5%) objects failed to converge. Propagating the 9 objects that converged but do not match NORAD over a 10 day period, shows that the velocities match within 1 m/s to those NORAD keps. x and y positions match within 1 km, and the worst difference value for z position is 11km after 10 days.

gal_piece_ins [0.6]

This routine inserts a piece of launch structure into an object catalog.

```
gal_piece_t *
gal_piece_ins
(
  gal_objcat_t *objcat,
  int satnum,
  char *intldesg,
  int *year,
  int *number,
  gal_status_t *status
) ;
```

On entry OBJCAT points to an object catalog structure created previously by gal_objcat_alloc, SATNUM contains the US Strategic Command Object Number, and INTLDESG contains the international designation (COSPAR/WDC-A-R&S). On return YEAR is set to the year of the launch, and NUMBER to the number of the launch within the year. The routine returns a pointer to the piece of launch structure. If the piece of launch is already in the catalog then a pointer is returned to the existing structure. i.e. no duplicates are created. If an internal error occurs then the error code GAL_ALLOC_FAILED is set and the routine returns NULL. The piece of launches are stored in a linked list within the catalog, the piece order is “A”, “B”, … “AA”, “AB”, … “AAA”, ... The international Designator must be in the format:

"YYYY-NNNPPP" e.g. "1997-051VX "

gal_settle [0.6]

This routine sets up a TLE structure from given mean Keplerian elememts suitable for use with the SGP4 propagator.

```
void
gal_settle
(
  double epoch1,
  double epoch2,
  double mean[6],
  gal_tle_t *tle,
  gal_status_t *status
) ;
```

On entry EPOCH1 and EPOCH2 contain a UTC date in standard SOFA two-piece format. MEAN contains the keplerian elements stored as follows:

[0] - Argument of perigee (degrees)
[1] - Right ascension of ascending node (degrees)
[2] - Inclination (degrees)
[3] - Mean motion (rev / day)
[4] - Eccentricity
[5] - Mean anomaly (degrees)

On return TLE is populated suitably for initializing the SGP4 propagator.

gal_sgp4 [0.2]

This routine is the SGP4 propagation model from US Strategic Command.

```
void
gal_sgp4
(
  gal_sgp4_t *sgp4,
  double date1,
  double date2,
  double pv[2][3],
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

SGP4 Initialized structure from gal_sgp4init call.
DATE1 Date part 1 (UTC)
DATE2 Date part 2 (UTC)

On return the variables are set as follows:

SGP4 Common values for subsequent calls
PV Geocentric position/velocity (meters, meters per second)
True Equator Mean Equinox (TEME)

If an error occurs then the applicable error code is set. This is an updated and combined version of SGP4 and SDP4, which were originally published separately in Spacetrack Report #3. This version follows the methodology from the AIAA paper (2006) describing the history and development of the code. This routine is a translation from c++ to c of David Vallado's SGP4UNIT.sgp4 routine (2007 November 16).

References:

NORAD Spacetrack Report #3 1980, Hoots, Roehrich

NORAD Spacetrack Report #6 1986, Hoots

Hoots, Schumacher and Glover 2004

Revisiting Spacetrack Report #3, Vallado, David, Crawford, Paul, Hujsak, Richard, Kelso, T.S., AIAA 2006-6753

gal_sgp4gm [0.2]

This routine gets the gravity model parameters required by SGP4

```
void
gal_sgp4gm
(
  gal_gm_t *gm,
  double *tumin,
  double *mu,
  double *re,
  double *xke,
  double *j2,
  double *j3,
  double *j4,
  double *j3oj2,
  gal_status_t *status
) ;
```

On entry GM points to the gravity model structure. On return the variables are set as follows:

TUMIN	One time unit (minutes)
MU	Earth gravitational parameter (meters^3 per second^2)
RE	Radius of the Earth (kilometers)
XKE	Reciprocal of tumin
J2	Un-normalized second zonal harmonic value
J3	Un-normalized third zonal harmonic value
J4	Un-normalized fourth zonal harmonic value
J3OJ2	J3 divided by J2

If an error occurs then the applicable error code is set.

References:

NORAD Spacetrack Report #3 1980, Hoots, Roehrich

NORAD Spacetrack Report #6 1986, Hoots

Hoots, Schumacher and Glover 2004

Revisiting Spacetrack Report #3, Vallado, David, Crawford, Paul, Hujsak, Richard, Kelso, T.S., AIAA 2006-6753

gal_sgp4init [0.2]

This routine initializes the variables for gal_sgp4.

```
void
gal_sgp4init
(
  gal_gm_t *gm,
  gal_tle_t *tle,
  gal_sgp4_t *sgp4,
  gal_status_t *status
) ;
```

On entry GM points to the gravity model structure, and TLE points to the two-line-elements parameters structure. On return SGP4 is initialized to its start state. If an error occurs then the applicable error code is set. This routine is based on a translation from c++ to c of David Vallado's SGP4UNIT.sgp4init routine (2007 November 16).

References:

NORAD Spacetrack Report #3 1980, Hoots, Roehrich

NORAD Spacetrack Report #6 1986, Hoots

Hoots, Schumacher and Glover 2004

Revisiting Spacetrack Report #3, Vallado, David, Crawford, Paul, Hujsak, Richard, Kelso, T.S., AIAA 2006-6753

gal_sgp4rs [0.5]

This routine computes the rise and set parameters for an Earth orbiting spacecraft and terrestrial observer using the Ernandes algorithm and the SGP4 propagator

```
void
gal_sgp4rs
(
  double utc1,
  double utc2,
  double latitude,
  double longitude,
  double height,
  double minel,
  double timefwd,
  double gm,
  double re,
  double inf,
  gal_sgp4_t *sgp4,
  double pass[3][3],
  gal_status_t *status
) ;
```

On entry the parameters are set as follows:

UTC1	Date part 1 (UTC)
UTC2	Date part 2 (UTC)
LATITUDE	Observer latitude (radians)
LONGITUDE	Observer longitude (radians)
HEIGHT	Observer height ASL (meters)
MINEL	Minimum elevation (radians)
TIMEFWD	Max time forward (days)
GM	Gravitational parameter (meters^{3} per second^{2})
RE	Earth equatorial radius (meters)
INF	Earth inverse flattening factor
SGP4	Initialized SGP4 structure

On return pass contains the AOS, CUL, & LOS details. UTC1 and UTC2 contain a Coordinated Universal Time (UTC) Julian Date in standard SOFA two-piece format. If the satellite never sets then the warning code GAL_NEVER_SETS is set. If an error occurs then the applicable error code is set. The rise and set parameters are the time, elevation, and azimuth for Acquisition of Signal (AOS), Loss of Signal (LOS), and Culmination (CUL) (maximum elevation). All angles are in radians. On return the array PASS is populated as follows:

PASS[0][0]	AOS JD (UTC)
PASS[0][1]	AOS Elevation
PASS[0][2]	AOS Azimuth
PASS[1][0]	CUL JD (UTC)
PASS[1][1]	CUL Elevation
PASS[1][2]	CUL Azimuth
PASS[2][0]	LOS JD (UTC)
PASS[2][1]	LOS Elevation
PASS[2][2]	LOS Azimuth

This routine is based upon the rise/set algorithm devised and developed by Ernandes.

References:

Kenneth J. Ernandes, private communication, December 17, 1997

gal_sgp4t.h [0.2]

This header file defines the SGP4 data structure that is used to store interim results between successive calls to gal_sgp4.

```
typedef struct {

/*
 * Internal Control Variables
 */

  int    error          ;
  char   init           ;
  char   method         ;

/*
 * TLE Parameters
 */

  int    satnum         ;  /* NORAD Catalog Number                        */
  char   classification ;  /* Security Classification                     */
  char   intldesg[10]   ;  /* International Designator (COSPAR/WDC-A-R&S) */
  int    epochyr        ;  /* Epoch Year                                  */
  double epochdays      ;  /* Epoch Day of Year (plus Fraction)           */
  double epoch          ;  /* Epoch Date ( days since 1950-01-01 0h )     */
  double ndot           ;  /* Mean motion derivative                      */
  double nddot          ;  /* Mean motion second derivative               */
  double bstar          ;  /* Bstar / Drag Term                           */
  int    ephtype        ;  /* Ephemeris Type                              */
  int    setnum         ;  /* Element set number                          */
  double inclo          ;  /* Inclination (rad)                           */
  double nodeo          ;  /* Right Ascension of Ascending Node (rad)     */
  double ecco           ;  /* Eccentricity                                */
  double argpo          ;  /* Argument of Perigee (rad)                   */
  double mo             ;  /* Mean Anomaly (rad)                          */
  double no             ;  /* Mean Motion ( rad/min )                     */
  int    revnum         ;  /* Epoch Revolution Number                     */

  double a              ;
  double altp           ;
  double alta           ;
  double jdepoch1       ;  /* Julian Date of Epoch Part 1                 */
  double jdepoch2       ;  /* Julian Date of Epoch Part 2                 */
  double rcse           ;
  int    epochtynumrev  ;

/*
 * Gravity Model Parameters
 */

  double tumin          ;  /* Minutes in one time unit                    */
  double mu             ;  /* Earth gravitational parameter               */
  double radiusearthkm  ;  /* Radius of the earth in kilometers
*/
  double xke            ;  /* Reciprocal of tumin                         */
  double j2             ;  /* Un-normalized second zonal harmonic value   */
  double j3             ;  /* Un-normalized third  zonal harmonic value   */
  double j4             ;  /* Un-normalized fourth zonal harmonic value   */
  double j3oj2          ;  /* j3 divided by j2                            */
  double vkmpersec      ;

/*
 * Near Earth
 */

  int    isimp          ;
  double aycof          ;
  double con41          ;
  double cc1            ;
  double cc4            ;
  double cc5            ;
  double d2             ;
  double d3             ;
  double d4             ;
  double delmo          ;
  double eta            ;
  double argpdot        ;
  double omgcof         ;
  double sinmao         ;
  double t              ;
  double t2cof          ;
  double t3cof          ;
  double t4cof          ;
  double t5cof          ;
  double x1mth2         ;
  double x7thm1         ;
  double mdot           ;
  double nodedot        ;
  double xlcof          ;
  double xmcof          ;
  double nodecf         ;

/*
 * Deep Space
 */

  int    irez           ;
  double d2201          ;
  double d2211          ;
  double d3210          ;
  double d3222          ;
  double d4410          ;
  double d4422          ;
  double d5220          ;
  double d5232          ;
  double d5421          ;
  double d5433          ;
  double dedt           ;
  double del1           ;
  double del2           ;
  double del3           ;
  double didt           ;
  double dmdt           ;
  double dnodt          ;
  double domdt          ;
  double e3             ;
  double ee2            ;
  double peo            ;
  double pgho           ;
  double pho            ;
  double pinco          ;
  double plo            ;
  double se2            ;
  double se3            ;
  double sgh2           ;
  double sgh3           ;
  double sgh4           ;
  double sh2            ;
  double sh3            ;
  double si2            ;
  double si3            ;
  double sl2            ;
  double sl3            ;
  double sl4            ;
  double gsto           ;
  double xfact          ;
  double xgh2           ;
  double xgh3           ;
  double xgh4           ;
  double xh2            ;
  double xh3            ;
  double xi2            ;
  double xi3            ;
  double xl2            ;
  double xl3            ;
  double xl4            ;
  double xlamo          ;
  double zmol           ;
  double zmos           ;
  double atime          ;
  double xli            ;
  double xni            ;

/*
 * The following parameters are for the use of the gal_sgp4rs routine
 */

  double time           ; /* UTC JD of last computation */
  double pv[2][3]       ; /* pv-vector ( m, m/s )       */
  double pos            ; /* modulus of p-vector        */
  double vel            ; /* modulus of v-vector        */

} gal_sgp4_t ;
```

References:

Revisiting Spacetrack Report #3, Vallado, David, Crawford, Paul, Hujsak, Richard,

Kelso, T.S., AIAA 2006-6753

gal_tlechksum [0.2]

This routine calculates the US Strategic Command two line element (TLE) card checksum character

```
char
gal_tlechksum
(
  char *card
) ;
```

On entry card points to the string containing the line for which the checksum is required. The routine returns the checksum character. The checksum is modulo 10, letters, blanks, periods, plus signs equal 0; minus signs equal 1.

gal_tledec [0.2]

This routine decodes the packed two line element (TLE) cards into the tle structure

```
void
gal_tledec
(
  char *card1,
  char *card2,
  gal_tle_t *tle,
  gal_status_t *status
) ;
```

On entry CARD1 and CARD2 point to the first and second TLE lines respectively. On return the structure pointed to by TLE contains the decoded TLE parameters. If an error occurs then the applicable error code is set.

References:

Revisiting Spacetrack Report #3, Vallado, David, Crawford, Paul, Hujsak, Richard, Kelso, T.S., AIAA 2006-6753

Chapter 12 – Keplerian Propagation

The routines detailed in this chapter are defined in the gal_kepler.h header file.

gal_ea2ta [0.5]

This routine calculates the true anomaly from eccentric anomaly.

```
double
gal_ea2ta
(
  double ea,
  double ecc
) ;
```

On entry EA contains the eccentric anomaly (radians), and ECC the eccentricity. The routine returns the true anomaly (radians). This routine is valid for elliptical orbits only.

References:

Fundamentals of Astrodynamics and Applications, Vallado, David A. Second Edition 2004, Page 85

gal_ha2ta [0.5]

This routine calculates the true anomaly from hyperbolic anomaly.

```
double
gal_ha2ta
(
  double ha,
  double ecc
) ;
```

On entry HA contains the hyperbolic anomaly (radians), and ECC the eccentricity. The routine returns the true anomaly (radians). This routine is valid for hyperbolic orbits only.

References:

Fundamentals of Astrodynamics and Applications, Vallado, David A. Second Edition 2004, Page 85

gal_kep2pv [0.4]

This routine computes position and velocity from the classical orbital elements.

```
void
gal_kep2pv
```

```
(
  double gm,
  double ecc,
  double raan,
  double argp,
  double inc,
  double p,
  double v,
  double truelon,
  double u,
  double lonper,
  double pv[2][3]
) ;
```

On entry the parameters are set as follows, if any parameter is unknown then the constant GAL_UNDEFINED (defined in gal_math.h) should be passed as parameter:

GM	Gravitational parameter (meters^3 per second^2)
ECC	Eccentricity
RAAN	Longitude of the ascending mode (radians)
ARGP	Argument of Pericenter (radians)
INC	Inclination (radians)
P	Semi-Latus Rectum (meters)
V	True Anomaly (radians)
TRUELON	True Longitude (radians)
U	Argument of Latitude (radians)
LONPER	True Longitude of Periapsis (radians)

On return PV contains the position and velocity vectors (meters, meters per second).

References:

Fundamentals of Astrodynamics and Applications, Vallado, David A. Second Edition 2004, Pages 118-122

Satellite Orbits, Oliver Montenbruck, Eberhard Gill, Springer 2005, Pages 28-32

Methods of Orbit Determination for the Micro Computer, Boulet, Dan, Willmann-Bell 1991, Pages 149-157

gal_ma2ea [0.5]

This routine calculates the eccentric anomaly from mean anomaly.

```
double
```

```
gal_ma2ea
(
  double ma,
  double ecc
) ;
```

On entry MA contains the mean anomaly (radians), and ECC the eccentricity. The routine returns the eccentric anomaly (radians). This routine is valid for elliptical orbits only.

References:

Fundamentals of Astrodynamics and Applications, Vallado, David A. Second Edition 2004, Pages 72-75

gal_ma2ha [0.5]

This routine calculates the hyperbolic anomaly from mean anomaly.

```
double
gal_ma2ha
(
  double ma,
  double ecc
) ;
```

On entry MA contains the mean anomaly (radians), and ECC the eccentricity. The routine returns the hyperbolic anomaly (radians). This routine is valid for hyperbolic orbits only.

References:

Fundamentals of Astrodynamics and Applications, Vallado, David A. Second Edition 2004, Pages 78-80

gal_pv2kep [0.4]

This routine computes the classical orbital elements from position and velocity.

```
void
gal_pv2kep
(
  double gm,
  double pv[2][3],
  double *sma,
  double *ecc,
  double *raan,
  double *argp,
  double *ma,
  double *inc,
  double *p,
  double *v,
  double *truelon,
  double *u,
  double *lonper
) ;
```

On entry GM contains the gravitational parameter (meters^3 per second^2), and PV the position and velocity vectors (meters, meters per second). On return the variables are set as follows, if any result cannot be calculated then GAL_UNDEFINED (defined in gal_math.h) is returned:

SMA	Semi-Major Axis (meters)
ECC	Eccentricity
RAAN	Longitude of the ascending mode (radians)
ARGP	Argument of Pericenter (radians)
MA	Mean Anomaly (radians)
INC	Inclination (radians)
P	Semi-Latus Rectum (meters)
V	True Anomaly (radians)
TRUELON	True Longitude (radians)
U	Argument of Latitude (radians)
LONPER	True Longitude of Periapsis (radians)

References:

Fundamentals of Astrodynamics and Applications, Vallado, David A. Second Edition 2004, Pages 118-122

Satellite Orbits, Oliver Montenbruck, Eberhard Gill, Springer 2005, Pages 28-32

Methods of Orbit Determination for the Micro Computer, Boulet, Dan, Willmann-Bell 1991, Pages 149-157

gal_pvt2pv [0.4]

This routine calculates position and velocity from starting position and velocity at given time using Universal variables. This routine is valid for all orbit types.

```
void
```

```
gal_pvt2pv
(
  double gm,
  double pv0[2][3],
  double ed0,
  double ed1,
  double tt0,
  double tt1,
  double pv[2][3]
) ;
```

On entry the variables are set as follows:

GM	Gravitational parameter (meters^3 per second^2)
PV0	Epoch position & velocity vectors (meters, meters per second)
ED0, ED1	Epoch date (TT)
TT0, TT1	Required date (TT)

Both Terrestrial Time (TT) Julian Dates are in standard SOFA two-piece format. On return PV contains the position and velocity vectors (meters, meters per second). The iteration method is the Laguerre's method described in Chobotov page 58. It was selected as it converged faster than the Newton-Raphson method described by Vallado, and the choice of initial value is simpler. The calculations of SN and CN are from Vallado as they are simpler to implement.

References:

Orbital Mechanics Third Edition, AIAA Education Series, Chobotov, Vladimir A. Ed., Pages 55-61

Fundamentals of Astrodynamics and Applications, Vallado, David A. Second Edition 2004, Pages 101-103

gal_t2pa [0.5]

This routine calculates the parabolic anomaly from time dt.

```
double
gal_t2pa
  (
    double gm,
    double dt,
    double p
  ) ;
```

On entry the variables are set as follows:

gm	Gravitational parameter (meters^3 per second^2)
dt	Time since periapsis (seconds)
p	Semi parameter (meters)

The routine returns the parabolic anomaly (radians). This routine is valid for parabolic trajectories only.

References:

Fundamentals of Astrodynamics and Applications, Vallado, David A. Second Edition 2004, Pages 75-78

gal_ta2ea [0.5]

This routine calculates the eccentric anomaly from true anomaly.

```
double
gal_ta2ea
  (
    double ta,
    double ecc
  ) ;
```

On entry ta contains the true anomaly (radians), and ecc the eccentricity. The routine returns the eccentric anomaly (radians). This routine is valid for elliptical orbits only.

References:

Fundamentals of Astrodynamics and Applications, Vallado, David A. Second Edition 2004, Page 85

gal_ta2ha [0.5]

This routine calculates the hyperbolic anomaly from true anomaly.

```
double
gal_ta2ha
  (
    double ta,
    double ecc
  ) ;
```

On entry ta contains the true anomaly (radians), and ecc the eccentricity. The routine returns the hyperbolic anomaly (radians). This routine is valid for hyperbolic orbits only.

References:

Fundamentals of Astrodynamics and Applications, Vallado, David A. Second Edition 2004, Page 85

Chapter 13 - Ephemerides

The routines detailed in this chapter are defined in the gal_ephemerides.h header file.

gal_beapv87 [0.4]

Earth Barycentric position and velocity.

```
void
gal_beapv87
 (
    double tt1,
    double tt2,
    int ref,
    double pv[2][3]
 ) ;
```

On entry the parameters are set as follows:

tt1	Epoch part 1 (TT)
tt2	Epoch part 2 (TT)
ref	Reference frame 0 = dynamical equinox and ecliptic J2000. 1 = FK5 (VSOP87)

On return pv contains the position and velocity vectors (AU, AU per day).

pv[0][0]	x
pv[0][1]	y
pv[0][2]	z
pv[1][0]	xdot
pv[1][1]	ydot
pv[1][2]	zdot

tt1 and tt2 contain a Terrestrial Time (TT) Julian Date in standard SOFA two-piece format. The vectors are Barycentric with respect to the FK5 Reference Frame. The routine is a solution from the planetary theory VSOP87. The main version of VSOP87 is similar to the previous theory VSOP82. In the both cases the constants of integration have been determined by fitting to the numerical integration DE200 of the Jet Propulsion Laboratory. The differences between VSOP87 and VSOP82 mainly improve the validity time-span for Mercury, Venus, Earth-Moon Barycenter and Mars with a precision of 1" for 4000 years before and after J2000. The same precision is ensured for Jupiter and Saturn over 2000 years and for Uranus and Neptune over 6000 years before and after J2000. The size of the relative precision p0 of VSOP87 solutions is given hereunder. That means that the actual precision is close by p0*a0 AU for the distances (a0 being the semi-major axis) and close by p0 radian for the other variables. By derivation with respect to time expressed in day (d), the precision of the velocities is close by p0*a0 AU per day for the distances and close by p0 radians per day for the other variables.

Body	a0 (au)	p0 (10^{-8})
Mercury	0.3871	0.6
Venus	0.7233	2.5
Earth	1.0000	2.5
Mars	1.5237	10.0
Jupiter	5.2026	35.0
Saturn	9.5547	70.0
Uranus	19.2181	8.0
Neptune	30.1096	42.0

References:

Bretagnon P., Francou G., : 1988, Astronomy & Astrophysics, 202, 309.

gal_bebpv87 [0.4]

Earth-Moon Barycenter Barycentric position and velocity.

```
void
gal_bebpv87
 (
    double tt1,
    double tt2,
    int ref,
    double pv[2][3]
 ) ;
```

See gal_beapv87 for details.

gal_bjupv87 [0.4]

Jupiter Barycentric position and velocity.

```
void
gal_bjupv87
 (
    double tt1,
    double tt2,
    int ref,
    double pv[2][3]
 ) ;
```

See gal_beapv87 for details.

gal_bmapv87 [0.4]

Mars Barycentric position and velocity.

```
void
gal_bmapv87
 (
    double tt1,
    double tt2,
    int ref,
    double pv[2][3]
 ) ;
```

See gal_beapv87 for details.

gal_bmepv87 [0.4]

Mercury Barycentric position and velocity.

```
void
gal_bmepv87
 (
    double tt1,
    double tt2,
    int ref,
    double pv[2][3]
 ) ;
```

See gal_beapv87 for details.

gal_bnepv87 [0.4]

Neptune Barycentric position and velocity.

```
void
gal_bnepv87
 (
    double tt1,
    double tt2,
    int ref,
    double pv[2][3]
 ) ;
```

See gal_beapv87 for details.

gal_bplpv87 [0.4]

Pluto Barycentric position and velocity.

```
void
gal_bplpv87
 (
    double tt1,
    double tt2,
    int ref,
    double pv[2][3]
 ) ;
```

See gal_beapv87, and gal_hplpv87 for details.

gal_bsapv87 [0.4]

Saturn Barycentric position and velocity.

```
void
gal_bsapv87
 (
    double tt1,
    double tt2,
    int ref,
    double pv[2][3]
 ) ;
```

See gal_beapv87 for details.

gal_bsupv87 [0.4]

Sun Barycentric position and velocity.

```
void
gal_bsupv87
 (
    double tt1,
    double tt2,
    int ref,
    double pv[2][3]
 ) ;
```

See gal_beapv87 for details.

gal_burpv87 [0.4]

Uranus Barycentric position and velocity.

```
void
gal_burpv87
 (
    double tt1,
    double tt2,
    int ref,
    double pv[2][3]
 ) ;
```

See gal_beapv87 for details.

gal_bvepv87 [0.4]

Venus Barycentric position and velocity.

```
void
gal_bvepv87
 (
    double tt1,
    double tt2,
    int ref,
    double pv[2][3]
 ) ;
```

See gal_beapv87 for details.

gal_epv00 [0.1]

Earth position and velocity, heliocentric and Barycentric, with respect to the International Celestial Reference Frame.

```
int
gal_epv00
 (
    double epoch1,
    double epoch2,
    double pvh[2][3],
    double pvb[2][3]
 ) ;
```

On entry epoch1+epoch2 contain the Barycentric Dynamical Time (TDB) Julian Date in standard SOFA two-piece format. On return pvh contains the heliocentric Earth position and velocity, and pvb the Barycentric Earth position and velocity (AU, AU per day). The routine returns one of the following status codes:

0	success
1	warning: date outside 1900-2100CE

The vectors are with respect to the International Celestial Reference Frame. The routine is a simplified solution from the planetary theory VSOP2000 (X. Moisson, P. Bretagnon, 2001, Celestial Mechanics & Dynamical Astronomy, 80, 3/4, 205-213) and is an adaptation of original Fortran code supplied by P. Bretagnon (private communication, 2000). Comparisons over the time span 1900-2100 with this simplified solution and the JPL DE405 ephemeris give the following results:

	RMS	max	
Heliocentric:			
position error	3.7	11.2	kilometers
velocity error	1.4	5.0	millimeters per second
Barycentric:			
position error	4.6	13.4	kilometers
velocity error	1.4	4.9	millimeters per second

gal_gmopv02 [0.3]

Moon geocentric position and velocity.

```
int
gal_gmopv02
 (
    double epoch1,
    double epoch2,
    int icor,
    double pv[2][3]
 ) ;
```

On entry the parameters are set as follows:

epoch1	Epoch part 1 (TDB)
epoch2	Epoch part 2 (TDB)
icor	correction type 0: the constants are fitted to LLR observations provided from 1970 to 2001; it is the default value;

1: the constants are fitted to DE405 ephemeris over one also additive corrections to the secular coefficients.

On return pv contains the geocentric Moon position & velocity (meters, meters per second). The routine returns one of the following status codes:

0	success
1	warning: date outside 1940-2060 CE

epoch1 and epoch2 contain the Barycentric Dynamical Time (TDB) Julian Date is in standard SOFA two-piece format. The algorithm used is the Lunar Solution ELP/MPP02.

References:

Lunar Solution ELP version ELP/MPP02, Jean Chapront and Gerard Francou, Observatoire de Paris -SYRTE department - UMR 8630/CNRS, October 2002

gal_gsupv00 [0.3]

Sun position and velocity, with respect to the Geocentric Celestial Reference Frame (GCRF).

```
int
gal_gsupv00
 (
    double epoch1,
    double epoch2,
    double pv[2][3]
 ) ;
```

On entry epoch1 and epoch2 contain the Barycentric Dynamical Time (TDB) Julian Date in standard SOFA two-piece format. On return pv contains the geocentric Sun position & velocity (meters, meters per second). The routine returns one of the following status codes:

0	success
1	warning: date outside 1900-2100CE range

References:

IERS Technical Note 32, IERS Conventions 2003, Dennis D. McCarthy et al., Page 12

gal_heapv87 [0.4]

Earth heliocentric position and velocity, with respect to the FK5 Reference Frame.

```
void
gal_heapv87
 (
    double tt1,
    double tt2,
    int ref,
    double pv[2][3]
 ) ;
```

On entry the parameters are set as follows:

tt1	Epoch part 1 (TT)
tt2	Epoch part 2 (TT)
ref	Reference frame 0 dynamical equinox and ecliptic J2000. 1 FK5 (VSOP87)

On return pv contains the position and velocity vectors (AU, AU per day).

pv[0][0]	x
pv[0][1]	y
pv[0][2]	z
pv[1][0]	xdot
pv[1][1]	ydot
pv[1][2]	zdot

tt1 and tt2 contain the Terrestrial Time Julian Date is standard SOFA two-piece format. The vectors are heliocentric with respect to the FK5 Reference Frame. The routine is a solution from the planetary theory VSOP87. The main version of VSOP87 is similar to the previous theory VSOP82. In the both cases the constants of integration have been determined by fitting to the numerical integration DE200 of the Jet Propulsion Laboratory. The differences between VSOP87 and VSOP82 mainly improve the validity time-span for Mercury, Venus, Earth-Moon Barycenter and Mars with a precision of 1" for 4000 years before and after J2000. The same precision is ensured for Jupiter and Saturn over 2000 years and for Uranus and Neptune over 6000 years before and after J2000. The size of the relative precision p0 of VSOP87 solutions is given hereunder. That means that the actual precision is close by p0*a0 AU for the distances (a0 being the semi-major axis) and close by p0 radian for the other variables. By derivation with respect to time expressed in day (d), the precision of the velocities is close by p0*a0 AU per day for the distances and close by p0 radians per day for the other variables.

Body	a0 (AU)	p0 (10^{-8})

Mercury	0.3871	0.6
Venus	0.7233	2.5
Earth	1.0000	2.5
Mars	1.5237	10.0
Jupiter	5.2026	35.0
Saturn	9.5547	70.0
Uranus	19.2181	8.0
Neptune	30.1096	42.0

References:

Bretagnon P., Francou G., : 1988, Astronomy & Astrophysics, 202, 309.

gal_hebpv87 [0.4]

Earth-Moon Barycenter heliocentric position and velocity.

```
void
gal_hebpv87
 (
    double tt1,
    double tt2,
    int ref,
    double pv[2][3]
 ) ;
```

See gal_heapv87 for details.

gal_hjupv87 [0.4]

Jupiter heliocentric position and velocity.

```
void
gal_hjupv87
 (
    double tt1,
    double tt2,
    int ref,
    double pv[2][3]
 ) ;
```

See gal_heapv87 for details.

gal_hmapv87 [0.4]

Mars heliocentric position and velocity.

```
void
gal_hmapv87
 (
    double tt1,
    double tt2,
    int ref,
    double pv[2][3]
 ) ;
```

See gal_heapv87 for details.

gal_hmepv87 [0.4]

Mercury heliocentric position and velocity.

```
void
gal_hmepv87
 (
    double tt1,
    double tt2,
    int ref,
    double pv[2][3]
 ) ;
```

See gal_heapv87 for details.

gal_hnepv87 [0.4]

Neptune heliocentric position and velocity.

```
void
gal_hnepv87
 (
    double tt1,
    double tt2,
    int ref,
    double pv[2][3]
 ) ;
```

See gal_heapv87 for details.

gal_hplpv87 [0.4]

Pluto heliocentric position and velocity.

```
void
gal_hplpv87
 (
    double tt1,
    double tt2,
    int ref,
    double pv[2][3]
 ) ;
```

On entry the parameters are set as follows:

tt1	epoch part 1 (TDB)
tt2	epoch part 2 (TDB)
ref	Reference frame
	0 dynamical equinox and ecliptic J2000.
	1 FK5 (VSOP87)

tt1 and tt2 contain a Barycentric Dynamical Time Julian Date in standard SOFA two-piece format. On return pv contains the position and velocity vectors (AU, AU per day).

pv[0][0]	x
pv[0][1]	y
pv[0][2]	z
pv[1][0]	xdot
pv[1][1]	ydot
pv[1][2]	zdot

The tables of Pluto were constructed by J. Chapront (BDL) with a method of approximation using frequency analysis as described in the referenced paper.

This representation uses the result of numerical integration DE200 of Jet Propulsion Laboratory as a source.

The interval of validity is 146120 days, from January 1 1700 0h JD2341972.5 to January 24 2100 0h JD2488092.5 The tables contain series which represent the heliocentric rectangular coordinates of Pluto as functions of time. The reference frame is defined with dynamical equinox and equator J2000 (DE200). The time scale is Barycentric Dynamical Time (TDB).

References:

Representation of planetary ephemerides by frequency analysis. Application to the five outer planets. Astronomy & Astrophysics Supplement Series, 109, 191 (1995)

Standish E. M., 1990, The observational basis for JPL's DE200, the planetary ephemerides of the Astronomical Almanac. Astronomy & Astrophysics, 233, 252

gal_hsapv87 [0.4]

Saturn heliocentric position and velocity.

```
void
gal_hsapv87
 (
    double tt1,
    double tt2,
    int ref,
    double pv[2][3]
 ) ;
```

See gal_heapv87 for details.

gal_hurpv87 [0.4]

Uranus heliocentric position and velocity.

```
void
gal_hurpv87
 (
    double tt1,
    double tt2,
    int ref,
    double pv[2][3]
 ) ;
```

See gal_heapv87 for details.

gal_hvepv87 [0.4]

Venus heliocentric position and velocity.

```
void
gal_hvepv87
 (
    double tt1,
    double tt2,
    int ref,
    double pv[2][3]
 ) ;
```

See gal_heapv87 for details.

gal_plan94 [0.1]

Approximate heliocentric position and velocity of a nominated major planet: Mercury, Venus, Earth-Moon Barycenter, Mars, Jupiter, Saturn, Uranus or Neptune.

```
int
gal_plan94
 (
    double date1,
    double date2,
    int np,
    double pv[2][3]
 ) ;
```

On entry date1 and date2 contain the Barycentric Dynamical Time (TDB) Julian Date in standard SOFA two-piece format, np contains the number of the required planet (1=Mercury, 2=Venus, 3=EMB ... 8=Neptune). This routine uses non-GAL standard constants for the planet's ID. This is for compatibility with the SOFA library. However, it is planned for this to be changed in a future release of GAL, so that the planet IDs match those used by the other GAL routines. On return pv contains the planet's heliocentric J2000 position and velocity vectors (AU, AU per day). The routine returns one of the following status codes:

-1	illegal NP (outside 1-8)
0	success
+1	warning: date outside 1000-3000 CE
+2	warning: solution failed to converge

If an np value outside the range 1-8 is supplied, an error status (-1) is returned and the pv vector set to zeroes. For np=3 the result is for the Earth-Moon Barycenter. To obtain the heliocentric position and velocity of the Earth, use instead the routine gal_epv00. The reference frame is equatorial and is with respect to the mean equator and equinox of epoch J2000. The algorithm is due to J.L. Simon, P. Bretagnon, J. Chapront, M. Chapront-Touze, G. Francou and J. Laskar (Bureau des Longitudes, Paris, France). From comparisons with JPL ephemeris DE102, they quote the following maximum errors over the interval 1800-2050:

	L (arcseconds)	B (arcseconds)	R (kilometers)
Mercury	4	1	300
Venus	5	1	800
EMB	6	1	1000

Mars	17	1	7700
Jupiter	71	5	76000
Saturn	81	13	267000
Uranus	86	7	712000
Neptune	11	1	253000

Over the interval 1000-3000, they report that the accuracy is no worse than 1.5 times that over 1800-2050. Outside 1000-3000 the accuracy declines.

Comparisons of this routine with the JPL DE200 ephemeris give the following RMS errors over the interval 1960-2025:

	position (kilometers)	velocity (meters per second)
Mercury	334	0.437
Venus	1060	0.855
EMB	2010	0.815
Mars	7690	1.98
Jupiter	71700	7.70
Saturn	199000	19.4
Uranus	564000	16.4
Neptune	158000	14.4

Comparisons against DE200 over the interval 1800-2100 gave the following maximum absolute differences. (The results using DE406 were essentially the same.)

	L	B	R	Rdot
Mercury	7	1	500	0.7
Venus	7	1	1100	0.9
EMB	9	1	1300	1.0
Mars	26	1	9000	2.5
Jupiter	78	6	82000	8.2
Saturn	87	14	263000	24.6
Uranus	86	7	661000	27.4
Neptune	11	2	248000	21.4

References:

Simon, J.L, Bretagnon, P., Chapront, J., Chapront-Touze, M., Francou, G., and Laskar, J., Astronomy & Astrophysics 282, 663 (1994).

Chapter 13 – Astromomical Macros

The header file gal_astro.h defines the following constants and macros:

```
/*
 * ----------------------
 * Astronomical Constants
 * ----------------------
 */

#define GAL_KM2M   (1000.0)           /* Kilometers to meters                          */

#define GAL_AU03   (149597870691.0)   /* Astronomical Unit IERS 2003 / DE405 meters    */

#define GAL_AU09   (149597870700.0)   /* Astronomical Unit IERS 2010 / IAU 2009 meters */

#define GAL_AUR    (149597870000.0)   /* Astronomical Unit rounded meters              */
                                      /* Same as SOFA's December 2010 DAU              */

#define GAL_RESU76 (6.96e8)           /* Equatorial Radius of the Sun IAU76 meters     */

#define GAL_SRP96  (4.56e-6)          /* Solar Radiation Pressure @ 1 AU IERS 1996     */
                                      /* Newtons per meter^2                           */

#define GAL_CM     (299792458.0)      /* Speed of Light meters per second              */

/*
 * -------------------
 * Astronomical Macros
 * -------------------
 */

/*
 * Macro to calculate Speed of Light in AU per second
 * Requires the desired value for AU as parameter
 */

#define GAL_CAUS( AU ) ( ( AU ) / GAL_CM )

/*
 * Macro to calculate Speed of Light in AU per day
 * Requires the desired value for AU as parameter
 */

#define GAL_CAUD( AU ) ( GAL_D2S / GAL_CAUS( ( AU ) ) )

/*
 * Speed of light (AU per day) rounded
 * This is the same as the SOFA December 2010 constant DC
 */

#define GAL_CAUDR ( GAL_D2S / 499.004782 )
```

Appendix A – GNU Free Documentation License

Version 1.2, November 2002

1. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software. We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

2. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language. A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them. The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none. The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words. A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque". Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only. The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text. A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition. The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

3. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3. You may also lend copies, under the same conditions stated above, and you may publicly display copies.

4. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects. If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages. If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public. It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

5. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher. D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers. If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles. You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard. You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one. The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

6. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers. The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of

Invariant Sections in the license notice of the combined work. In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

7. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects. You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

8. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document. If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

9. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail. If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

10. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

11. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/. Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

12. ADDENDUM:

How to use this License for your documents To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page: Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License". If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this: with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation. If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index