# RBSP SPICE Workshop

August 18, 2010

Scott Turner

Grant Stephens

* This material borrows heavily from NAIF's Tutorials located at: http://naif.jpl.nasa.gov/naif/tutorials.html

# Topics

- Installing the SPICE Toolkit
- Testing the installation – a simple SPICE program
- Time conversions with SPICE
- Using SPICE for ephemeris computations
- Frame transformations in SPICE

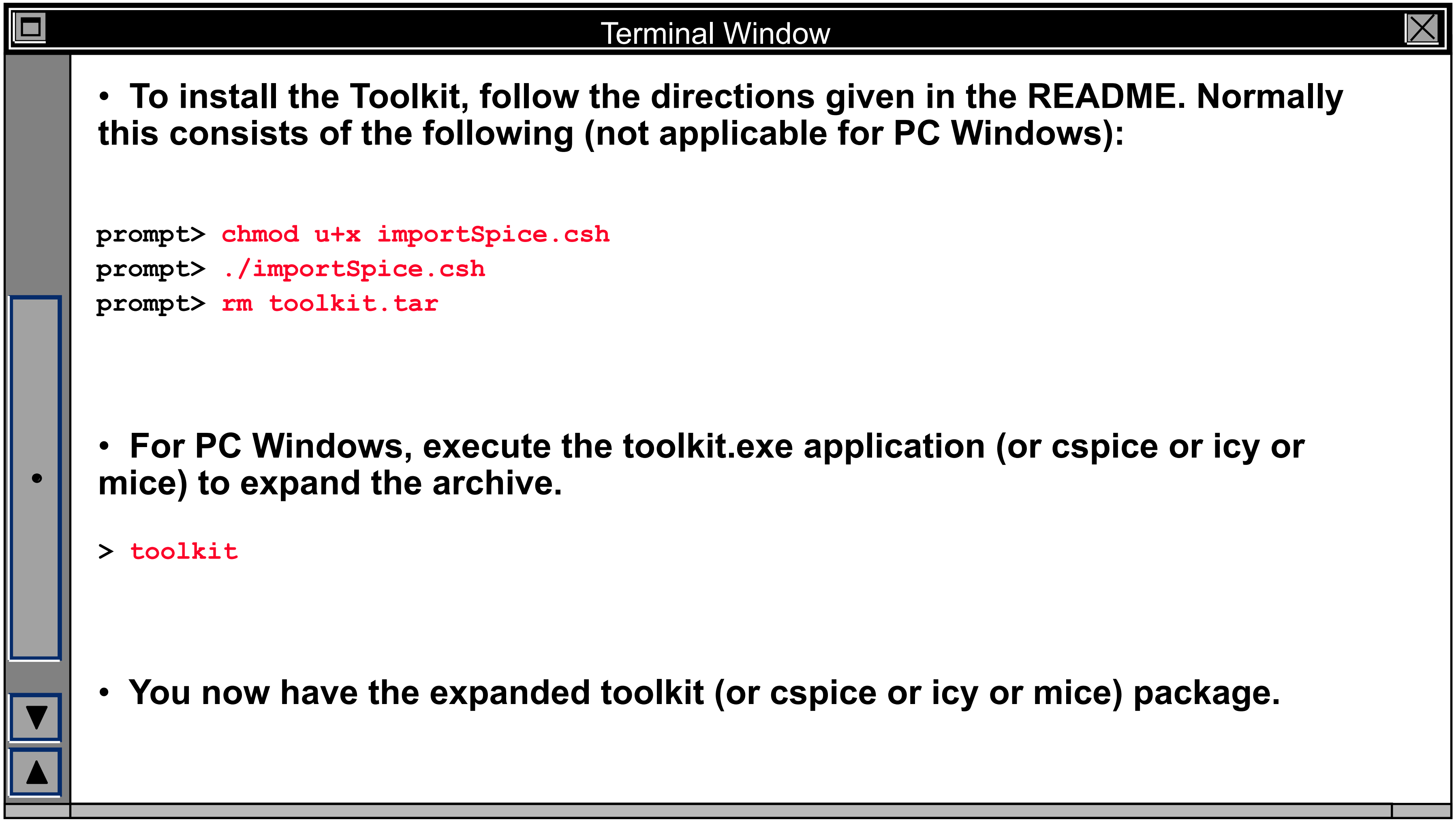
# Getting Toolkit

- All instances of the SPICE Toolkit are available 24x7 from the NAIF WWW server

    `http://naif.jpl.nasa.gov/naif/toolkit.html`

- No password or identification is needed
- To download a Toolkit package
    - Select language – FORTRAN, C, IDL, or MATLAB
    - Select computer platform/OS/compiler combination
    - Download all toolkit package components
        - » package file – toolkit.tar.Z (or toolkit.exe),<br>cspice.tar.Z (or cspice.exe),<br>icy.tar.Z (or icy.exe), or<br>mice.tar.Z (or mice.exe)
        - » Installation script (if present) – import*.csh
        - » Accompanying documents - README, dscriptn.txt, whats,new

# Installing Toolkit



- To install the Toolkit, follow the directions given in the README. Normally this consists of the following (not applicable for PC Windows):

```
prompt> chmod u+x importSpice.csh
prompt> ./importSpice.csh
prompt> rm toolkit.tar
```

- For PC Windows, execute the toolkit.exe application (or cspice or icy or mice) to expand the archive.

```
> toolkit
```

- You now have the expanded toolkit (or cspice or icy or mice) package.

# Programming Task

- Install the SPICE Toolkit onto your system now.

# Installed Directory Structure

- The top level directory name for each Toolkit is:
  - “toolkit” for Fortran Toolkits
  - “cspice” for C Toolkits
  - “icy” for IDL Toolkits
  - “mice” for MATLAB Toolkits

- Directory structures for the Toolkits are almost identical. However…
  - The CSPICE, Icy and Mice Toolkits also have a directory for include files
  - The names for application source code directories in CSPICE, Icy and Mice differ slightly from those in the Fortran toolkit
  - Icy and Mice include additional directories for
    - Icy/Mice source code
    - Icy/Mice cookbook programs

# Installed Directory Structure

- The next level is comprised of:
  - data
    - » Cookbook example kernels (use ONLY for training with cookbook programs)
  - doc
    - » Text documents — *.req, *.ug, spicelib.idx/cspice.idx, whats.new, dscriptn.txt, version.txt.
    - » Subdirectory containing HTML documentation, called “html”.
      - The “html” subdirectory contains a single file — the top level HTML documentation index called “index.html” — and a number of subdirectories, one for each of the various groups of documents in HTML format (API Reference Guide pages, User’s Guide pages, etc.)
  - etc
    - » In generic Toolkits this directory is empty.
  - exe
    - » Executables for  brief, chronos, ckbrief, commnt, inspekt, mkspk, msopck, spacit, spkdiff, frmdiff, spkmerge, tobin, toxfr, version.
    - » Executables for the several cookbook example programs.

# Installed Directory Structure

- include (applies to CSPICE, Icy, and Mice)
  - » API header files.
    - File to include in callers of CSPICE is SpiceUsr.h
- lib
  - » Toolkit libraries:
    - For Fortran SPICE Toolkits
      - spicelib.a or spicelib.lib (public modules; use these)
      - support.a or support.lib (private modules; don’t use these)
    - For CSPICE Toolkits
      - cspice.a or cspice.lib (public modules; use these)
      - csupport.a or csupport.lib (private modules; don’t use these)
    - For Icy Toolkits:
      - icy.so (shared object library)
      - icy.dlm (dynamically loadable module)
      - cspice.a or cspice.lib
      - csupport.a or csupport.lib
    - For Mice Toolkits:
      - mice.mex* (shared object library)
      - cspice.a or cspice.lib
      - csupport.a or csupport.lib
- src
  - » Source code directories for executables and libraries
    - Files have type *.f, *.for, *.inc, *.pgm, *.c, *.h, *.x, *.pro, *.m
    - *.h files appearing here are not part of the user API

# Toolkit Documentation

- All Toolkits include documentation in plain text and HTML formats
  - Plain text documents are located under the “doc” directory
  - HTML documents are located under the “<toolkit_name>/doc/html” (Unix) or “<toolkit name>\doc\html” (Windows) directory
    - » “<toolkit_name>/doc/html/index.html” or “<toolkit_name>\doc\html\index.html” is the top level index
- All Toolkits include the following kinds of documents
  - Module headers
    - » Act as primary functional specification: I/O, exceptions, particulars defining behavior of module
    - » Contain code examples
    - » A standard format is used for each routine or entry point
    - » Plain text Module Headers:
      - Fortran: the top comment block in the source code files under “src/spicelib”
      - C: the top comment block in the source code files under “src/cspice”
      - IDL: Icy Module Headers are not available in plain text format
      - MATLAB accessible via “`help` *`function_name`*” command
    - » HTML Module Headers are accessible using the “API Reference Guide” link from the top level index.

continues on next page

# Toolkit Documentation

- Required Reading
  - References for principal subsystems
  - Provide many low-level details
  - Provide code examples
  - Plain text versions are located under “doc” and have extension “.req”
  - HTML versions are are accessible using the “Required Reading Documents” link from the top level index.
  - Not all of Required Readings were adapted for all languages
    - Some of the Required Reading documents provided with CSPICE still cover Fortran SPICE
    - Some of the Required Readings for Icy or Mice toolkits still cover CSPICE
- User’s Guides
  - Interface specifications for the Toolkit utility programs and applications
  - Plain text versions are located under “doc” and have extension “.ug”
  - HTML versions are accessible using the “User’s Guide Documents” link from the top level index.

# Toolkit Documentation

- Other documents
  - Permuted Index
    - » Maps phrases describing functionality to corresponding module names and file names
    - » Shows names of all entry points in Fortran toolkit APIs
    - » Plain text version is located under “doc” and has extension “.idx”:

| | |
|---|---|
| • Fortran: | spicelib.idx |
| • C: | cspice.idx |
| • IDL: | icy.idx and cspice .idx |
| • MATLAB: | mice.idx and cspice.idx |

    - » HTML version isaccessible using the “Permuted Index” link from the top level index.
  - Toolkit Description
    - » Describes the directory structure and contents of an installed Toolkit
    - » Customized based on set of delivered products and platform
    - » Plain text version is “doc/dscriptn.txt”
    - » HTML version isaccessible using the “Toolkit Contents” link from the top level index.

# Toolkit Documentation

- Other documents (continued)
  - Introduction to SPICE
    - » Brief introduction to the Toolkit and SPICE system
    - » Not available in plain text
    - » HTML version isaccessible using the “Introduction to the SPICE System” link from the top level index.
  - What’s New in SPICE
    - » Describes new features and bug fixes
    - » Plain text version is “doc/whats.new”
    - » HTML version isaccessible using the “What’s New in SPICE” link from the top level index.
  - Toolkit Version Description
    - » Indicates Toolkit version
    - » Plain text version is “doc/version.txt”
    - » Not available in HTML

# Testing the Installation

- Retrieve the “start_programming” lesson from NAIF’s website:

  `http://naif.jpl.nasa.gov/naif/lessons.html`

- Unpack it onto your system and begin this basic SPICE programming lesson. It will verify that you have SPICE properly installed and are able to utilize successfully it with your language of choice.

# Programming Task

- **Build the sample SPICE program from the start programming lesson now.**

# Time Systems and Kernels

- Time inputs and outputs in users' SPICE-based programs are usually strings representing epochs in these three time systems:
  - Coordinated Universal Time (UTC)
  - Spacecraft Clock (SCLK)
  - Ephemeris Time (ET, also referred to as Barycentric Dynamical Time, TDB)
- Independent time variable in kernels, and time inputs and outputs to SPICE routines reading kernel data and computing derived geometry, are double precision numbers representing epochs in these two time systems:
  - Numeric Ephemeris Time (TDB), expressed as ephemeris seconds past J2000
  - Encoded Spacecraft Clock, expressed as clock ticks since the clock start
- SPICE provides routines to perform conversions between string and numeric times using data from these two kernels:
  - Leapseconds Kernel (LSK) containing data for UTC <=> ET conversion
  - Spacecraft Clock Kernel (SCLK) containing data for ET <=> SCLK conversion
- Caution: the long-term <u>future</u> relationships between UTC, TDB, and SCLK time systems cannot be accurately predicted

# Converting Time Strings

- UTC, TDB, or TDT (TT) String to numeric Ephemeris Time
  - STR2ET ( *string*, ET )
    - Converts virtually any time string, excepting SCLK. For example:

      ‘1996-12-18T12:28:28’ ‘1978/03/12 23:28:59.29’ ‘Mar 2, 1993 11:18:17.287 p.m. PDT’
      ‘1995-008T18:28:12’ ‘1993-321//12:28:28.287’
      ‘2451515.2981 JD’ ‘ jd 2451700.05 TDB’
      ‘1988-08-13, 12:29:48 TDB’ ‘1992 June 13, 12:29:48 TDT’
    - Requires LSK kernel
- Spacecraft Clock String to numeric Ephemeris Time
  - SCS2E ( *scid*, *string*, ET )
    - Converts SCLK strings consistent with SCLK parameters. For example:

      ‘5/65439:18:513’ (VGR1) ‘946814430.172’ (MRO) ‘1/0344476949-27365’ (MSL)
    - The “LSK and SCLK” tutorial discusses SCLK string formats in detail
    - Requires SCLK kernel, and usually LSK kernel (to handle a very small ~2 msec, difference between TDB and TT)
- Spacecraft Clock String to Encoded Spacecraft Clock (used in the mid-level interfaces of the C-kernel system)
  - SCENCD ( *scid*, *string*, SCLKDP )
    - Requires only SCLK kernel

# Converting Numeric Times - 1

- **Numeric Ephemeris Time to Calendar, DOY or Julian Date UTC, TDB, or TDT String**

  – **TIMOUT ( *et, fmtpic,* STRING )**
    - » ***fmtpic*** **is an output time string format specification, giving the user great flexibility in setting the appearance of the output time string and the time system used (UTC, TDB, TDT).**
      - **See next slide for examples of format pictures to produce a variety of output time strings**
      - **See the TIMOUT header for complete format picture syntax**
      - **The module TPICTR may be useful in constructing a format picture specification from a sample time string**
    - » **Requires LSK Kernel**

  – **ETCAL ( *et,* STRING )**
    - » **STRING, fixed format ephemeris calendar time string, for example**

      **‘2000 JAN 01 12:16:40.123’**
    - » **No LSK Kernel is required**

# Use of Format Picture

## Example Time Strings and the Corresponding Format Pictures

| Common Time Strings | Format Picture Used (*fmtpic*) |
|---|---|
| `1999-03-21T12:28:29.702` | `YYYY-MM-DDTHR:MN:SC.###` |
| `1999-283T12:29:33` | `YYYY-DOYTHR:MN:SC ::RND` |
| `1999-01-12, 12:00:01.342 TDB` | `YYYY-MM-DD, HR:MN:SC.### ::TDB TDB` |
| `2450297.19942145 JD TDB` | `JULIAND.######## ::TDB JD TDB` |

| Less Common Time Strings | Format Picture Used (*fmtpic*) |
|---|---|
| `465 B.C. Jan 12 03:15:23 p.m.` | `YYYY ERA Mon DD AP:MN:SC ampm` |
| `04:28:55 A.M. June 12, 1982` | `AP:MN:SC AMPM Month DD, YYYY` |
| `Thursday November 04, 1999` | `Weekday Month DD, YYYY` |
| `DEC 31, 15:59:60.12 1998 (PST)` | `MON DD, HR:MN:SC YYYY (PST)::UTC-8` |

# Converting Numeric Times - 2

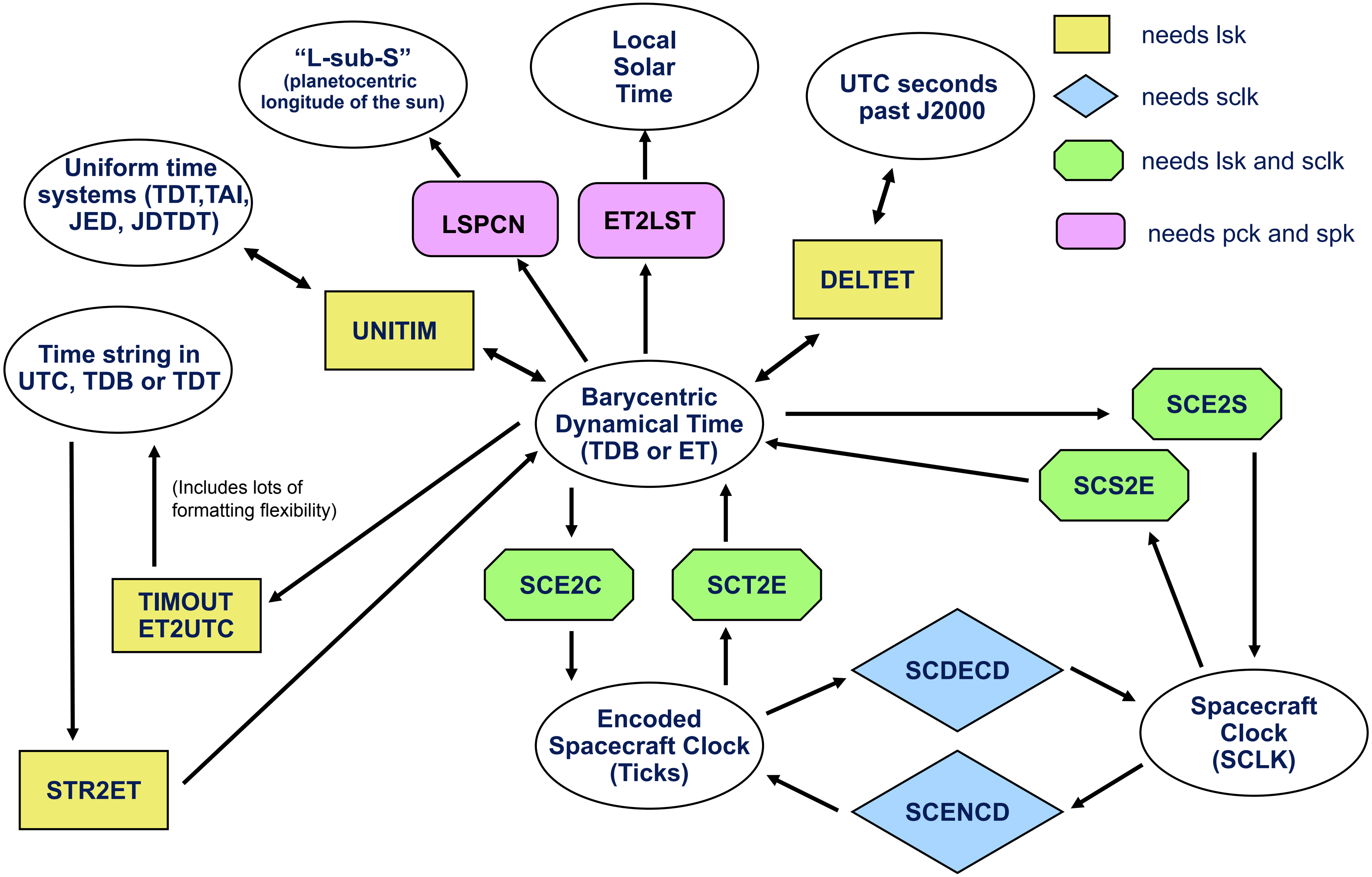
- **Numeric Ephemeris Time to Spacecraft Clock String**
  - **SCE2S (*scid*, *et*, SCLKCH )**
    - » **Requires both LSK and SCLK kernels**
    - » **Output SCLK string examples:**

```
‘1/1487147147.203’   (Cassini, MGS)
‘1/05812:00:001’     (Voyager 1 and 2)
```

- **Encoded Spacecraft Clock to Spacecraft Clock String**
  - **SCDECD (*scid*, *sclkdp*, SCLKCH )**
    - » **Requires only SCLK kernel**

# Principal Time System Interfaces

Navigation and Ancillary Information Facility



- 🟨 needs lsk
- 🔷 needs sclk
- 🟩 needs lsk and sclk
- 🟪 needs pck and spk

“L-sub-S” (planetocentric longitude of the sun)

Local Solar Time

UTC seconds past J2000

Uniform time systems (TDT,TAI, JED, JDTDT)

LSPCN

ET2LST

DELTET

UNITIM

Time string in UTC, TDB or TDT

Barycentric Dynamical Time (TDB or ET)

SCE2S

SCS2E

(Includes lots of formatting flexibility)

TIMOUT ET2UTC

SCE2C

SCT2E

SCDECD

STR2ET

Encoded Spacecraft Clock (Ticks)

Spacecraft Clock (SCLK)

SCENCD

# Module Header Purpose

- NAIF uses module "headers" to provide SPICE users with detailed information describing a module's function and design.
  - In FORTRAN, C and MATLAB the "headers" are comment blocks inserted in the source code
- All Toolkit distributions include HTML versions of the module headers.
- Using the HTML formats is usually the best approach because of hyperlinking with other NAIF documentation
- The next charts provide the header locations

# Fortran Module Header Locations

- In FORTRAN Toolkits:
  - <path to SPICELIB>/toolkit/src/spicelib/<name.f or <name>.for
  - In most cases there is a single "header" at the top of the source code. For cases where a FORTRAN module has multiple entry points, there are additional "headers" at each entry point. For example:
    - » "keeper.f" has entries for:
      - FURNSH, KTOTAL, KINFO, KDATA, KCLEAR, and UNLOAD
- HTML versions of the headers:
  - <path to SPICELIB>/toolkit/doc/html/spicelib/index.html

# C Module Header Locations

- In C Toolkits:
  - <path to CSPICE>/cspice/src/cspice/<name>_c.c
- HTML versions of the headers:
  - <path to CSPICE>/cspice/doc/html/cspice/index.html

# Icy Module Header Locations

- In IDL ("Icy") toolkits, two sets of headers are provided.
  - Icy headers in HTML format:
    - » <path to Icy>/icy/doc/html/icy/index.html
  - CSPICE headers, in text and HTML formats:
    - » <path to Icy>/icy/src/cspice/<name>_c.c
    - » <path to Icy>/icy/doc/html/cspice/index.html
- The information provided in an "Icy" wrapper is minimal in some cases; the corresponding CSPICE wrapper provides more detail.
  - A link to the corresponding CSPICE wrapper is provided in the Icy wrapper.

# Mice Module Header Locations

- In Matlab (“Mice”) toolkits, two sets of headers are provided.
  - Mice headers in HTML format:
    - » <path to Mice>/mice/doc/html/mice/index.html
    - » The user can also access the information presented in the HTML document via the Matlab `help` command, e.g.

```
>> help cspice_str2et
```

  - CSPICE headers, in text and HTML formats:
    - » <path to Mice>/mice/src/cspice/<name>_c.c
    - » <path to Mice>/mice/doc/html/cspice/index.html
- The information provided in a “Mice” wrapper is minimal in some cases; the corresponding CSPICE wrapper provides more detail.
  - A link to the corresponding CSPICE wrapper is provided in the Mice wrapper.

# Examine a Typical Header

- As example, look for and examine one of these headers:

| FORTRAN | C | IDL (Icy) | MATLAB (Mice) |
|---|---|---|---|
| TIMOUT | timout_c | cspice_timout | cspice_timout |
| STR2ET | str2et_c | cspice_str2et | cspice_str2et |

# Programming Task

Navigation and Ancillary Information Facility

- Install the in-situ programming lesson from:

  `http://sd-www.jhuapl.edu/MIDF/turner/rbsp/`

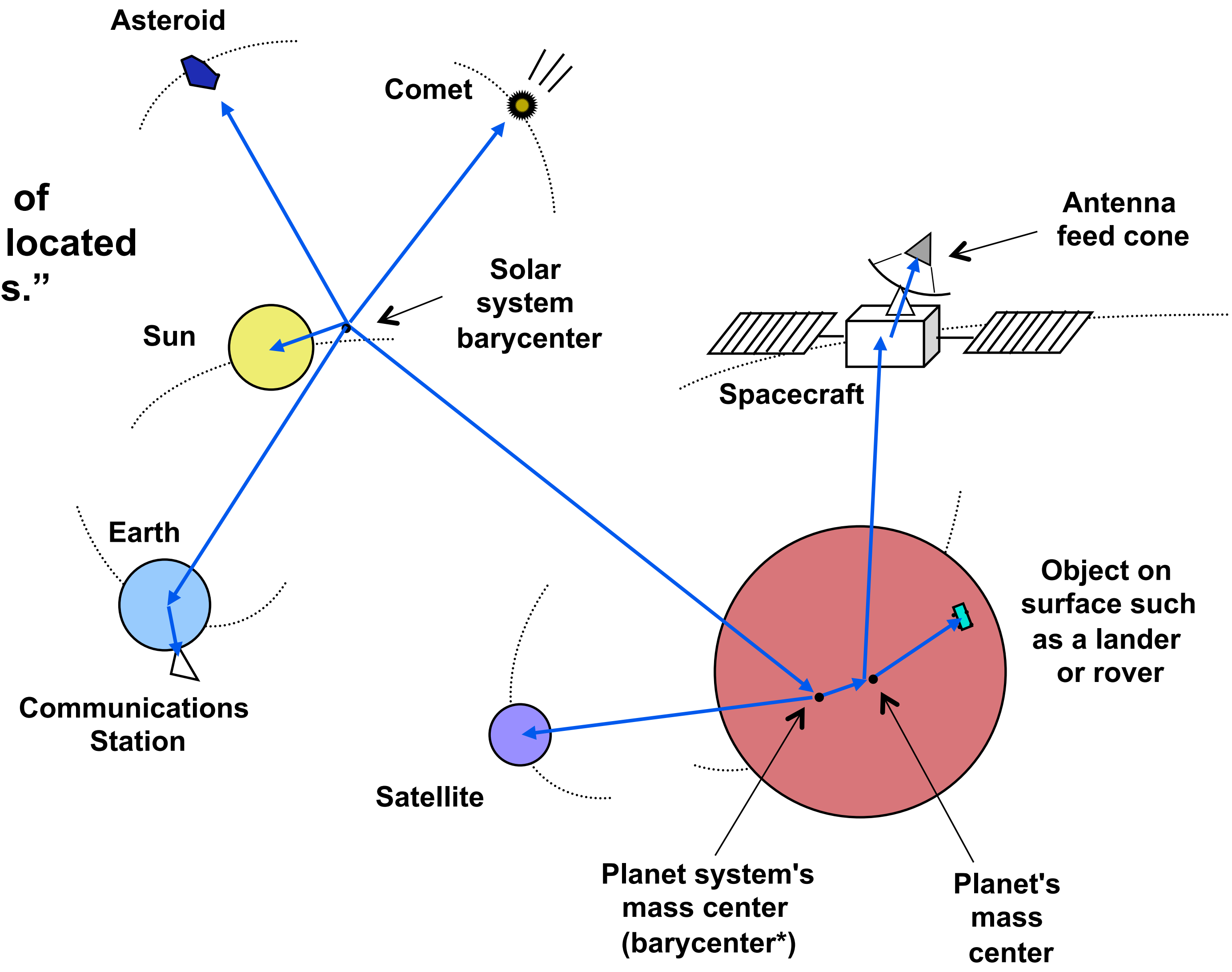
- Complete steps #1 and #2 in the in-situ programming lesson.
- In addition to the above, write code to convert the the RBSP_B MET: 98927737 with 22180 (out of 50000) subsecond counts to a UTC day of year format. Note: There are several ways to attack this, consider using the SPICE routine TIMOUT to solve this problem and exploring the encoded SCLK (SCENCD, SCDECD) concept.

# SPICE Ephemeris Data

- An SPK file contains ephemeris (trajectory) data for "ephemeris objects.”
  - “Ephemeris” means position and velocity as a function of time.
- Spacecraft, planets, satellites, comets and asteroids are the obvious kinds of "ephemeris objects," but many other possibilities exist, such as:
  - a rover on the surface of a body
  - a camera on top of a mast on a lander
  - a transmitter cone on a spacecraft
  - a deep space communications antenna on the earth
  - the center of mass of a planet/satellite system (planet barycenter)
  - the center of mass of our solar system (solar system barycenter)
- See the next page for a pictorial representation of some of these objects.

# Examples of Ephemeris Objects

The head and the tail of every blue arrow are located at “ephemeris objects.”



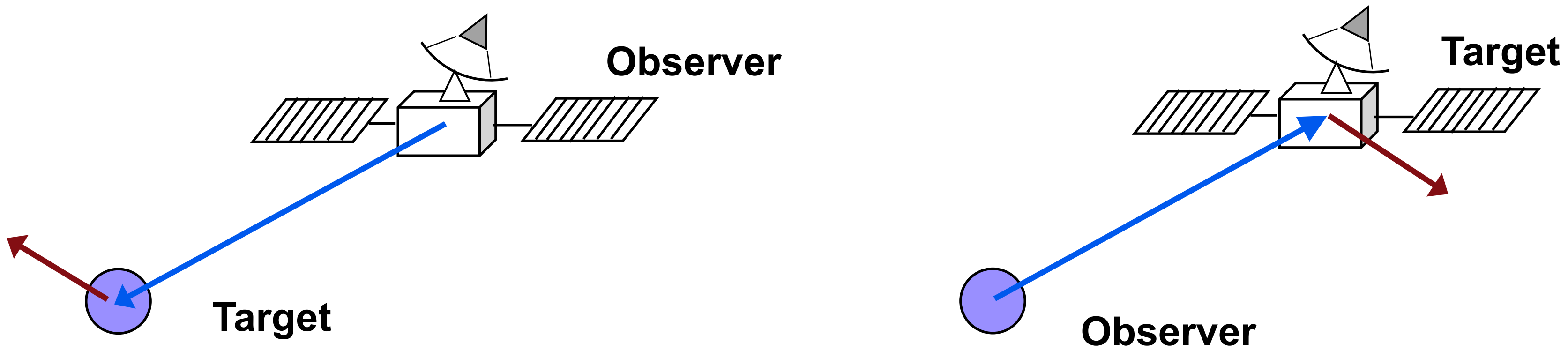
*A barycenter is the center of mass of a set of bodies, such as Saturn plus all of Saturn's satellites.

# Inside an SPK: Bodies and Centers of Motion

- Inside an SPK file ephemeris objects come in pairs: a “body” and its “center of motion.”
  - The ephemeris is given for the body moving relative to the center of motion.
    - » For the position component, the vector points TO the body FROM the center of motion.
  - There can be, and often are, multiple such pairs within an SPK file.

# Reading an SPK: Observers and Targets

- When you read an SPK file you specify which ephemeris object is to be the “target” and which is to be the “observer.”
- The SPK system returns the state of the target relative to the observer.
  - The position data point from the “observer” to the “target.”
  - The velocity is that of the “target” relative to the “observer.”



**Caution:** state (observer, target) ≠ - state (target, observer)

unless the state is geometric (no aberration corrections).

# SPK File Coverage

- The time period over which an SPK file provides data for an ephemeris object is called the “coverage” or “time coverage” for that object.
  - An SPK file’s coverage for an object consists of one or more time intervals.
  - Often the coverage for all objects in an SPK file is a single, common time interval.
    - » Example: a planetary SPK file such as de421.bsp
    - » Counterexample: Cassini tour SPK with merged Huygens probe ephemeris

- For any request time within any time interval comprising the coverage for an object, the SPK system can return a vector representing the state of that body relative to its center of motion.
  - The SPK system will automatically interpolate ephemeris data to produce a state vector at the request time.
  - To a user’s program, the ephemeris data appear to be continuous over each time interval, even if the data stored inside the SPK file are discrete.

# Reference Frames as Used in Writing and Reading SPKs

## On Writing

- All ephemeris data in an SPK file have an associated reference frame
  - There could be multiple such frames, each for a different portion of the data
  - For the ephemeris data to be useful, this/these frames must be "known" to any program that will subsequently read the ephemeris data

## On Reading

- The application "reading" an SPK file(s) must specify relative to what reference frame the output state or position vectors are to be given
  - This frame must be "known" to the SPICE-based program

- "Known" means either a built-in frame ("hard coded") or one fully specified at run-time
  - The user's program may need to have access to additional SPICE data in order to construct some of these frames

# Retrieving Position or State Vectors

- To retrieve position or state vectors of ephemeris objects from an SPK file one normally needs two kinds of SPICE kernels
  - Ephemeris kernel(s) (SPK)
    - » Sometimes just one is needed
    - » Sometimes two or more are needed to chain together the "target" and "observer" you have selected
  - Leapseconds kernel (LSK)
    - » Used to convert between Coordinated Universal Time (UTC) and Ephemeris Time (ET)
    - » Usually needed since most people work with UTC time
- Retrieving ephemeris data from an SPK file is usually called “reading” the file
  - This term is not very accurate since the SPK “reader” software also performs interpolation, and may chain together data from multiple sources and/or perform aberration corrections
- State and position vectors retrieved from an SPK file by the SPK “reader” routines are of the form:
  - X,Y, Z, dX, dY, dZ for a state vector
  - X, Y, Z for a position vector

# Retrieving a State Vector

Fortran syntax used here

Initialization…typically done once per program execution

Tell your program which SPICE files to use (“loading” files)

```
CALL FURNSH  ('spk_file_name')
CALL FURNSH  ('leapseconds_file_name')
```

Better yet, replace these two calls with a single call to a “furnsh kernel” containing the names of all kernel files to load.

Loop... do as many times as you need to

Convert UTC time to ephemeris time (TDB), if needed

```
CALL STR2ET ( 'utc_string', tdb)
```

Retrieve state vector from the SPK file at your requested time

```
CALL SPKEZR (target, tdb, 'frame', 'correction', observer, state, light time)
```

inputs

outputs

Use the returned state vector in other SPICE routines to compute observation geometry of interest.

# Arguments of SPKEZR - 1

## INPUTS

- **TARGET* and OBSERVER*: Character names or NAIF IDs for the end point and origin of the state vector (Cartesian position and velocity vectors) to be returned.**
  - **The position component of the requested state vector points from observer to target.**
- **TDB: The time at the observer at which the state vector is to be computed. The time system used is Ephemeris Time (ET), now generally called Barycentric Dynamical Time (TDB).**
- **FRAME: The SPICE name for the reference frame in which your output state vector is to be given. SPK software will automatically convert data to the frame you specify (if needed). SPICE must know the named frame. If it is not a built-in frame SPICE must have sufficient data at run time to construct it.**

* Character names work for the target and observer inputs only if built into SPICE or if registered using the SPICE ID-body name mapping facility. Otherwise use the SPICE numeric ID in quotes, as a character string.

# Arguments of SPKEZR - 2

- CORRECTION: Specification of what kind of aberration correction (s), if any, to apply in computing the output state vector.
  - Use 'LT+S' to obtain the apparent state of the target as seen by the observer. 'LT+S' invokes light time and stellar aberration corrections.
  - Use 'NONE' to obtain the uncorrected (aka "geometric") state, as given by the source SPK file or files.

  See the header for subroutine SPKEZR, the document SPK Required Reading, or the "Fundamental Concepts" tutorial for details. See the backup charts for examples of aberration correction magnitudes.

OUTPUTS

- *STATE*: This is the Cartesian state vector you requested. Contains 6 components: three for position (x,y,z) and three for velocity (dx, dy, dz) of the target with respect to the observer. The position component of the state vector points from the *observer* to the *target*.
- *LIGHT TIME*: The one-way light time between the (optionally aberration-corrected) position of target and the geometric position of the observer at the specified epoch.

# Chaining Data

- If needed, the SPK software will automatically chain together two or more state vectors needed to connect your "target" to your "observer." (See next chart.)
- SPK software can chain together state vectors provided by a single SPK file, or by multiple SPK files.
- In doing the chaining, if needed the SPK software will also transform the various state vectors into a common reference frame for addition or subtraction, then transform the result to the reference frame you have selected for output.
  - Your selected output reference frame must be one known to the SPICE system, and your application program must have available all needed SPICE data to construct this reference frame.
- See the chaining example on the next page.

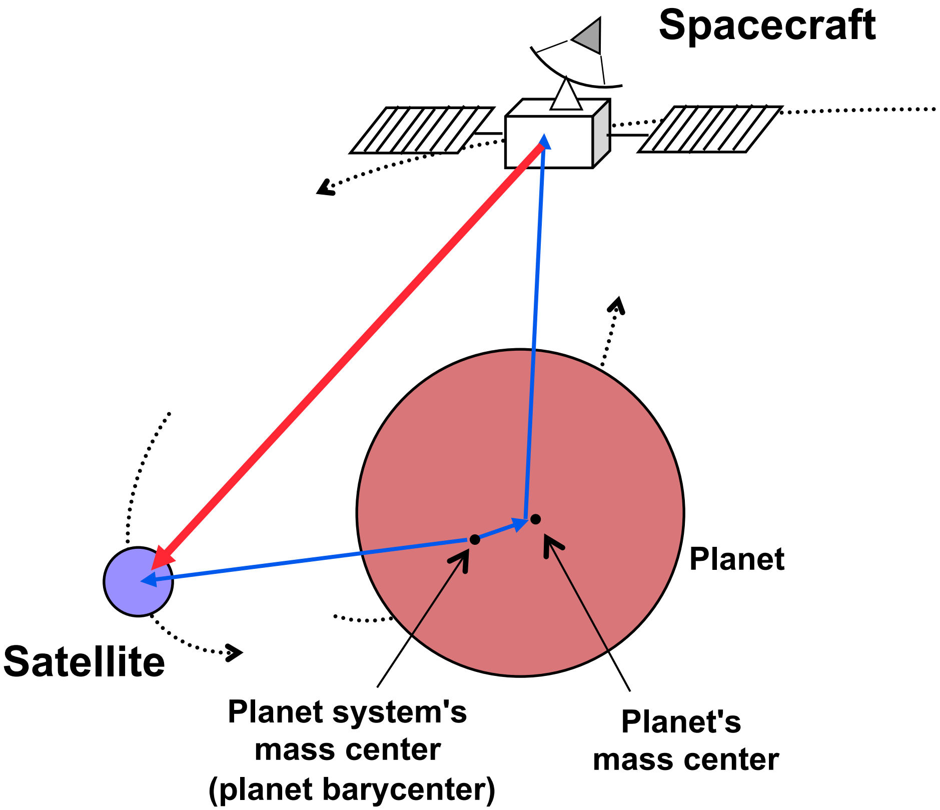
# Example of Chaining

## Suppose you ask for the position of the satellite relative to the spacecraft.

Your SPK may not contain exactly the ephemeris you want.

But, if all the needed data are available in your SPK file, the SPK subsystem will chain together the position vectors indicated by the three blue arrows–the data explicitly contained in an SPK file–to give you the position vector indicated by the red arrow–the one you asked for.

This might require the loading of two SPK files, one containing data for the spacecraft relative to the planet mass center, and another containing data for the planet mass center and the satellite relative to the planet barycenter.



"Planet system" = the planet and all of it's satellites

# Programming Task

- Complete steps #3 and #4 in the in-situ programming lesson.

# Reference Frames: Definition

- A reference frame is an ordered set of three mutually orthogonal (possibly time dependent) unit-length direction vectors, coupled with a location called the frame's "center" or "origin."
  - SPICE documentation frequently uses the shorthand "frame."
  - A reference frame is also called a "basis," but SPICE documentation very rarely uses this term.

# Reference Frame Center

- A frame's center is an ephemeris object whose location is coincident with the origin (0, 0, 0) of a reference frame.
  - The center of the IAU_<body> frame is <body>.
  - The center of any inertial frame is (in SPICE) the solar system barycenter.
    - » Even for frames naturally associated with accelerated bodies, such as MARSIAU.
- A frame's center plays little role in specification of states
  - Origin cancels out when doing vector arithmetic
    - » Whether positions of objects A and B are specified relative to centers C1 or C2 makes no difference:

$$(A - C1) - ( B - C1 ) = ( A - C2 ) - ( B - C2 ) = A - B$$

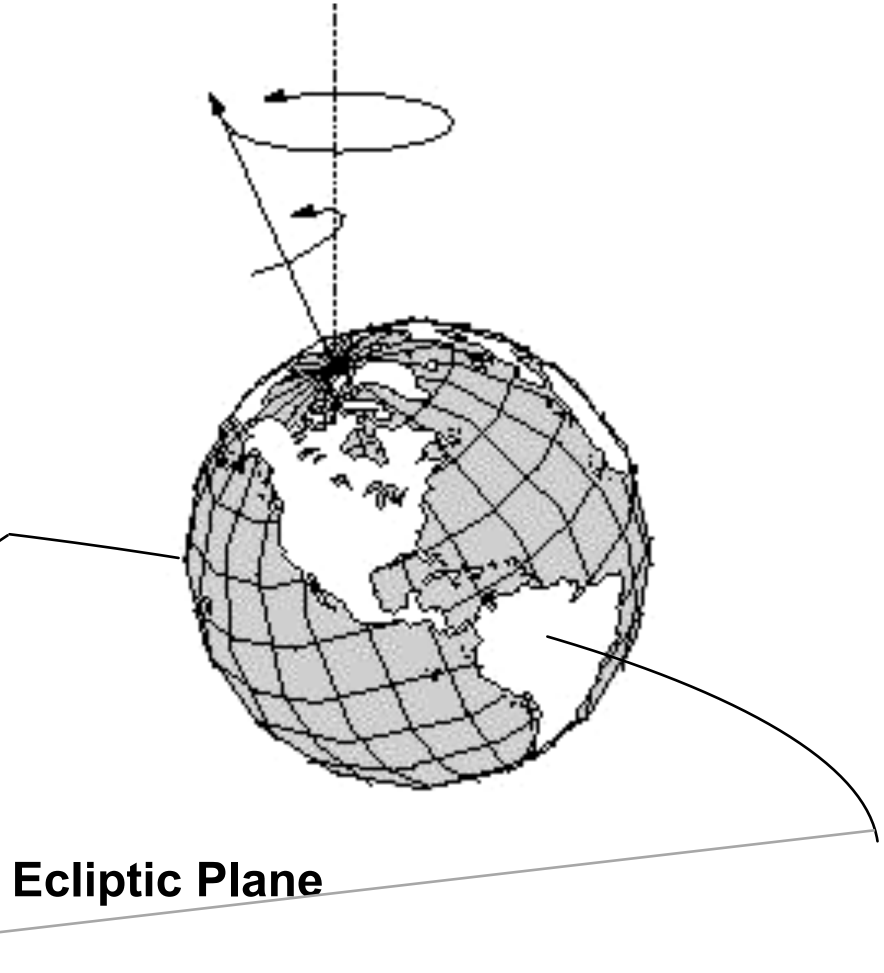
  - But the center *is* used in computing light time to centers of non-inertial frames
    - » When the aberration-corrected state of Titan as seen from the Cassini orbiter is computed in the body-fixed IAU_Titan frame, light time is computed from Titan's center to the Cassini orbiter, and this light time is used to correct both the state and orientation of Titan.

# Types of Reference Frames

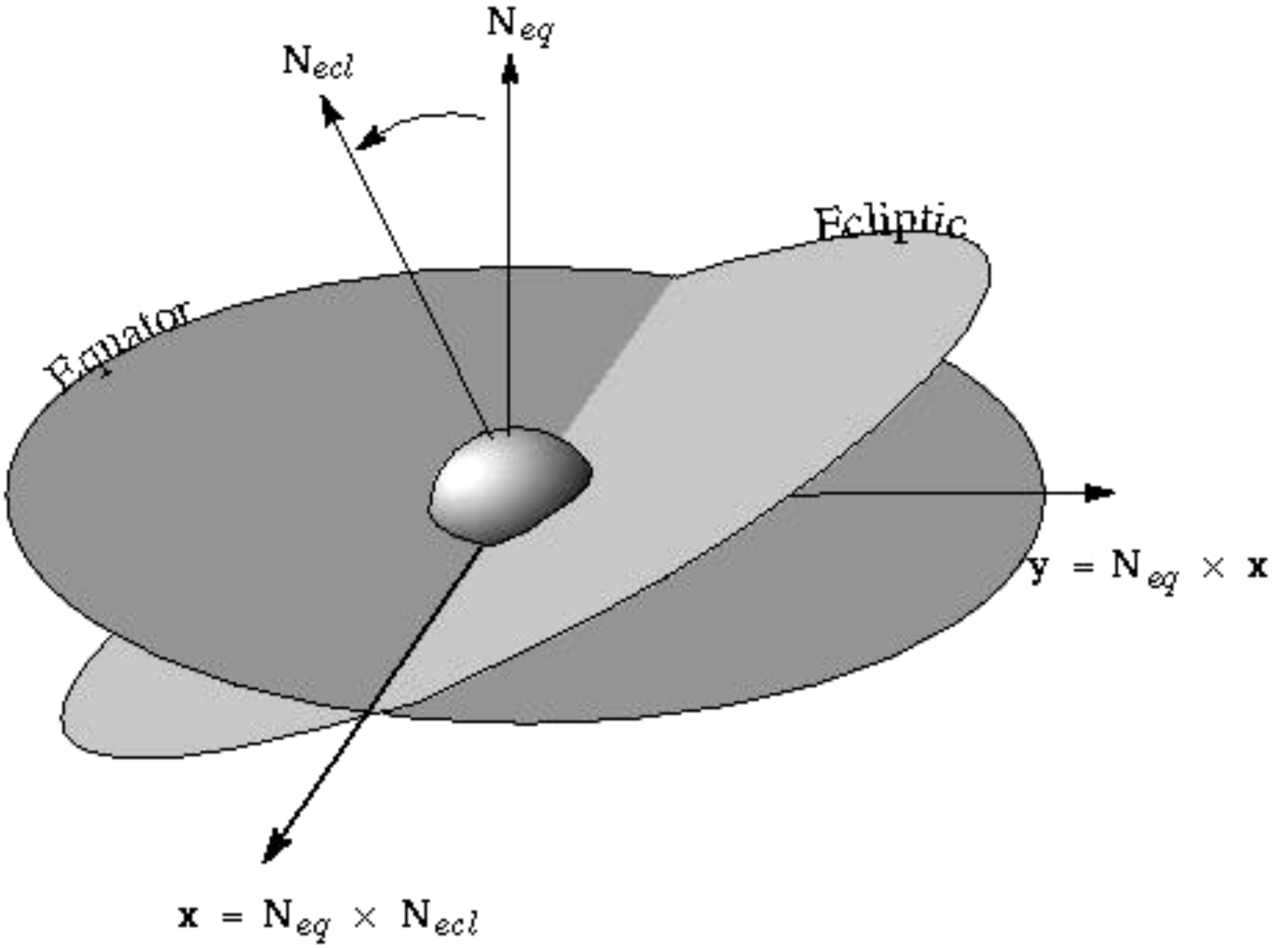
- **Inertial**
  - Non-rotating
    - » With respect to fixed stars
  - Non-accelerating origin
    - » Velocity is typically non-zero; acceleration is negligible
  - Examples:
    - » J2000 (also called ICRF), B1950
- **Non-Inertial**
  - Examples
    - » Body-fixed
      - Centered at body center
      - Topocentric
    - » Instrument
    - » Dynamic frames
      - For example, frames defined by time-dependent vectors

# Frames Defined by Dynamics

- **Mean Equator**
  - Model gives mean direction of north pole of earth accounting for precession
  - Defines z-axis of frame
  - Defines a mean plane of equator
- **Mean Ecliptic**
  - Model gives mean direction of the “pole” of the earth's orbit
  - Defines a mean plane of the ecliptic
- **Intersection of planes at a particular epoch determines x-axis**

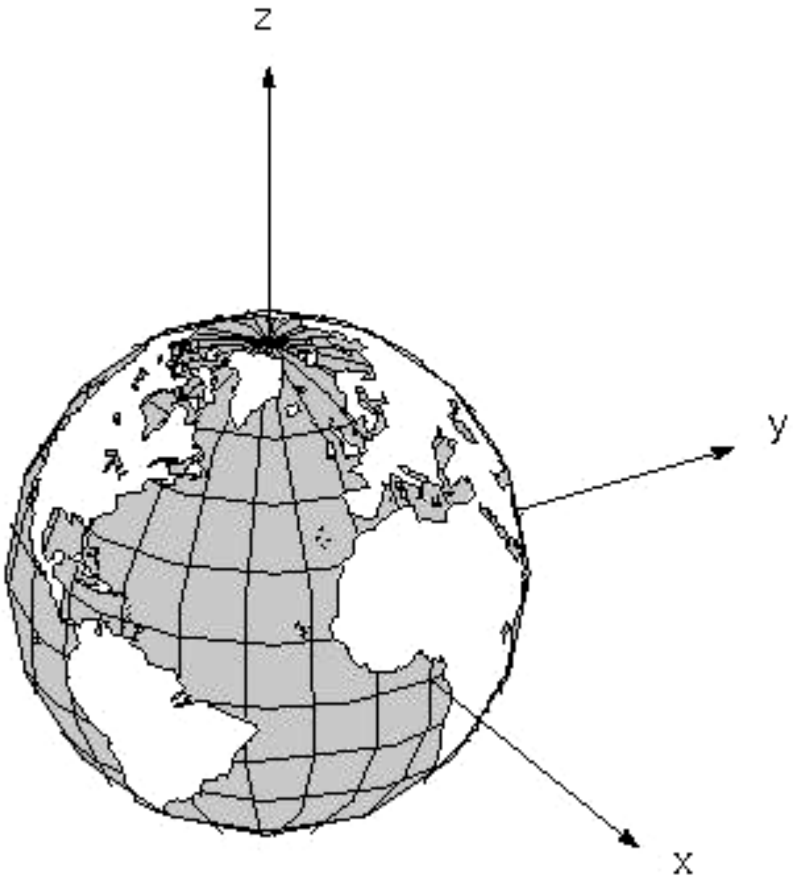


# J2000 (ICRF) Frame


$N_{eq}$
$N_{ecl}$
Ecliptic
Equator
$\mathbf{y} = N_{eq} \times \mathbf{x}$
$\mathbf{x} = N_{eq} \times N_{ecl}$

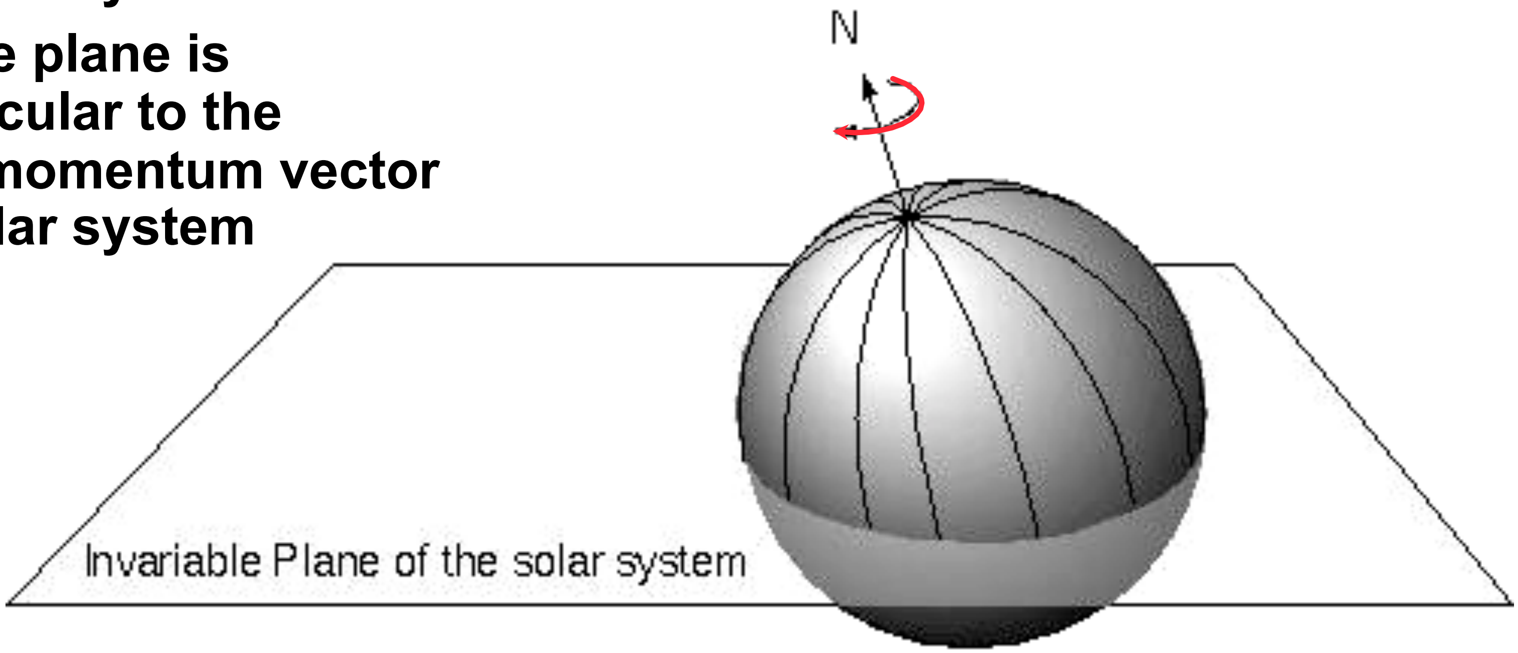
# Rotating Frames

- Rotating frames rotate with respect to Inertial Frames. Directions of axes are not constant w.r.t. inertial frames
- Centers may accelerate
- Examples:
  - Body-fixed frames are tied to the surface of a body and rotate with it.
  - Spacecraft-fixed frames are defined by the time-varying orientation of a spacecraft



# IAU Bodyfixed Frames

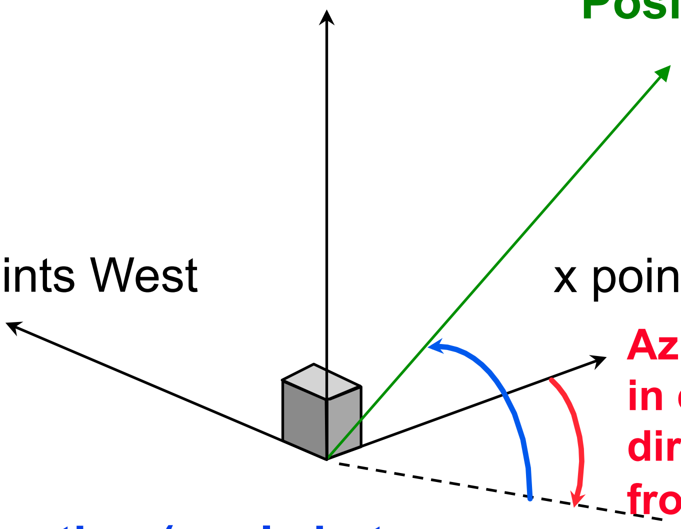
- Defined with simple models for position of spin axis and motion of prime meridian
- Z-axis points to the “north” side of the invariable plane of the solar system
- Invariable plane is perpendicular to the angular momentum vector of the solar system



IAU = International Astronomical Union

# Topocentric Frames

- **Topocentric frames are attached to a surface**
- **Z-axis is parallel to the gravity gradient or orthogonal to reference spheroid**



z points “up”

**Position Vector**

y points West

x points North

**Azimuth (increases in clockwise direction, measured from +x axis)**

**Elevation (angle between vector and x-y plane)**

**Orthogonal projection of vector onto x-y plane**

**One example of a topocentric frame. There are other types of topocentric frames: for example, the z-axis could point down, the x-axis North, and the y-axis East.**

# Spacecraft and Instrument Frames

- Defined relative to structures
  - Spacecraft
  - Scan platform
  - Instrument
    - » For example you might have:
      - z-axis lies along instrument boresight
      - x and y axes defined by instrument characteristics

# State Vectors

- **The state of an object is its position and velocity relative to a second object**
  - In SPICE, these objects are often referred to as “target” and “observer” or “center”
  - E.g. Saturn relative to Saturn barycenter; Titan relative to Huygens probe
- **In the SPK subsystem a state is a six dimensional vector**
  - First three components are Cartesian position: $x, y, z$
  - Second three components are Cartesian velocity: $dx/dt, dy/dt, dz/dt$
  - Units are km, km/sec
- **A state is specified relative to a reference frame**

# Transforming States

- **To perform algebraic operations on states they must be in the same frame.**
- **Position-only frame transformations require only a rotation* matrix given as a function of time.**
  - » $P_B\,(\boldsymbol{t}) = R_{A\text{ to }B}(\boldsymbol{t})\; P_A(t)$
- **Position and velocity frame transformations require that we differentiate the above equation**
  - » $d P_B\,(\boldsymbol{t})\;/d t = d R_{A\text{ to }B}(\boldsymbol{t})/d t\; P_A(\boldsymbol{t}) + R_{A\text{ to }B}(\boldsymbol{t})\; d\, P_A(\boldsymbol{t})/d t$
- **We can use a 6x6 matrix to combine these two transformations into a single equation**

* Assuming both frames are right-handed

# Transforming States

$$S_B(t) = T_{\text{A to B}}(t) S_A(t)$$

where

$$S_i(t) = \begin{pmatrix} P_i(t) \\ dP_i(t)/dt \end{pmatrix} \quad \text{i = A or B}$$

and

$$T_{\text{A to B}}(t) = \left( \begin{array}{c|c} R_{\text{A to B}}(t) & 0 \\ \hline dR_{\text{A to B}}(t)/dt & R_{\text{A to B}}(t) \end{array} \right)$$

The SPICELIB routines SXFORM and PXFORM return state transformation and position transformation matrices respectively.

# Coordinate Systems

- **Planetocentric**
  - Latitude: measured from X-Y plane
  - Longitude: increases counterclockwise w.r.t. the +Z axis
    - » +Z points to the north side of the invariable plane
  - Radius: measured from center of object
- **Planetographic, Geodetic, Planetodetic**
  - Tied to a reference surface
  - Latitude: for a point on a reference ellipsoid, angle measured from X-Y plane to the surface normal at the point of interest. For other points, equals latitude at the nearest point on the reference ellipsoid.
  - Longitude
    - » -odetic: same as for planetocentric
    - » -ographic: longitude of sub-observer point, for a distant, fixed observer in the J2000 frame, increases with time
  - Height above reference surface

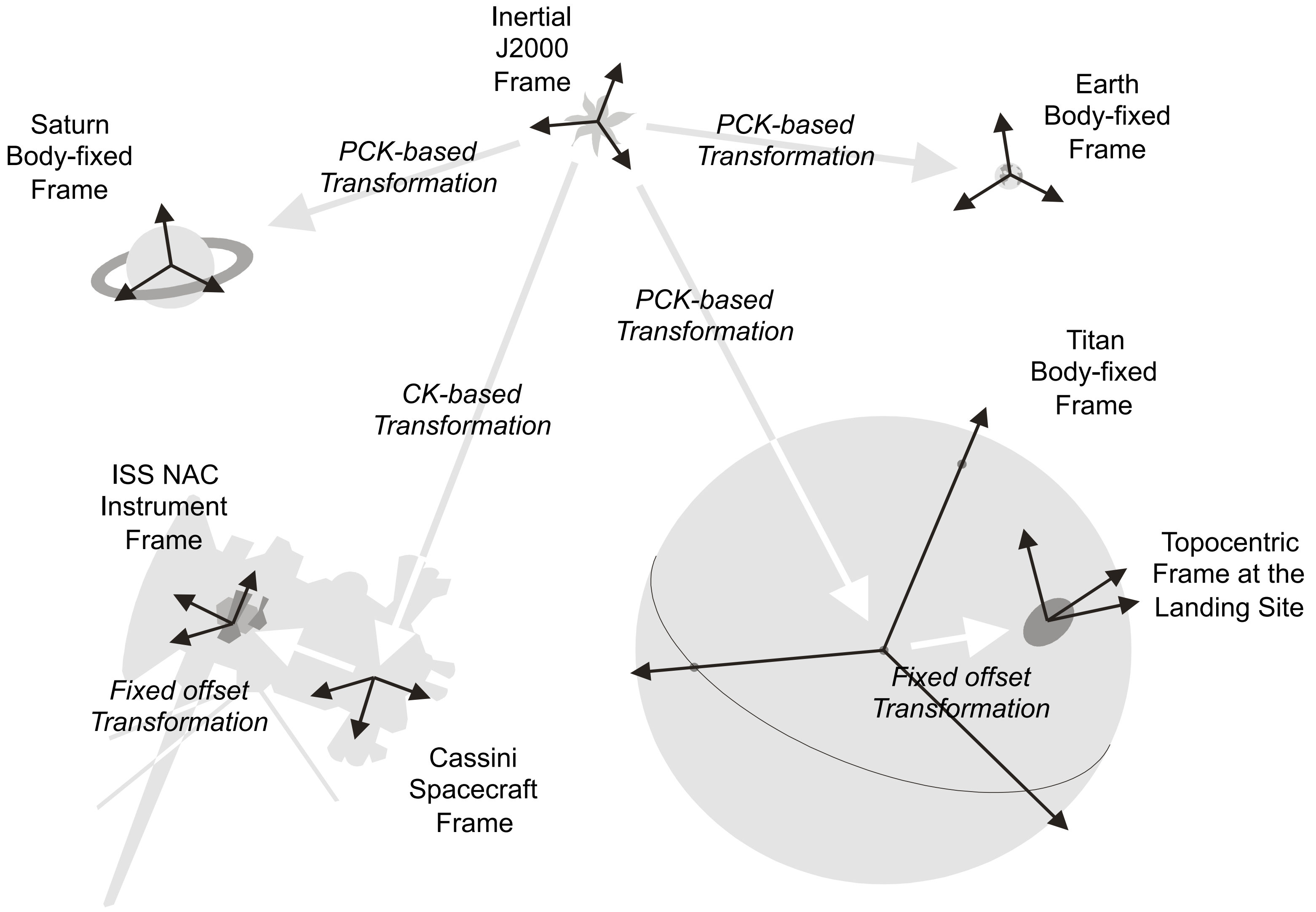
# Introduction

## What does the FRAMES subsystem do?

- It establishes relationships between reference frames used in geometry computations -- it "chains frames together.”
  - We often call this set of relationships a frame tree
- It connects frames with the sources of their orientation specifications.
- Based on these relationships and orientation source information, it allows SPICE software to compute transformations between neighboring frames in the "chain," and to combine these transformations in the right order, thus providing an ability to compute orientation of any frame in the chain with respect to any other frame in the chain at any time. *

* If the complete set of underlying SPICE data needed to compute the transformation is available.

# Sample Frame Tree



# Frame Classes

| ***Frame class*** | *Examples* |
|---|---|
| **Inertial** | • Earth Equator/Equinox of Epoch (J2000, …)<br>• Planet Equator/Equinox of Epoch (MARSIAU, ...)<br>• Ecliptic of Epoch (ECLIPJ2000, ...) |
| **Body-fixed** | • Solar system body IAU frames (IAU_SATURN, …)<br>• High accuracy Earth frames (ITRF93, …)<br>• High accuracy Moon frames (MOON_PA, MOON_ME) |
| **CK-based** | • Spacecraft (CASSINI_SC_BUS, …)<br>• Moving parts of an instrument (MPL_RA_JOINT1, ...) |
| **Fixed Offset** | • Instrument mounting alignment (CASSINI_ISS_NAC, …)<br>• Topocentric (DSS-14_TOPO, …) |
| **Dynamic** | • Geomagnetic<br>• Geocentric Solar Equatorial<br>• Planet true equator and equinox of date |

# Frames Class Specifications

Navigation and Ancillary Information Facility

| *Frame class* | *Frame Defined in* | *Orientation data provided in* |
|---|---|---|
| **Inertial** | **Toolkit** | **Toolkit** |
| **Body-fixed** | **Toolkit or FK** | **PCK** |
| **CK based** | **FK** | **CK** |
| **Fixed offset** | **FK** | **FK** |
| **Dynamic** | **FK** | **Toolkit, or computed using FK, SPK, CK, and/or PCK** |

# FRAMES Subsystem Interfaces

**SXFORM/PXFORM** **returns state or position transformation matrix**

```
CALL SXFORM ( ‘FROM_FRAME_NAME’, ‘TO_FRAME_NAME’, ET, MAT6x6 )
CALL PXFORM ( ‘FROM_FRAME_NAME’, ‘TO_FRAME_NAME’, ET, MAT3X3 )
```

**SPKEZR/SPKPOS** **returns state or position vector in specified frame**

```
CALL SPKEZR ( BOD, ET, ‘FRAME_NAME’, CORR, OBS, STATE,  LT )
CALL SPKPOS ( BOD, ET, ‘FRAME_NAME’, CORR, OBS, POSITN, LT )
```

**The above are FORTRAN examples, using SPICELIB modules.**
**The same interfaces exist for C, using CSPICE modules, and for Icy and Mice.**

# What are the Names of Frames?

- Refer to “NAIF IDs” Tutorial for an introduction to reference frame names and IDs

- Refer to FRAMES.REQ for the list of NAIF “built in” (hard coded) inertial and body-fixed frames

- Refer to a project’s Frames Kernel (FK) file for a list of frames defined for the spacecraft, its subsystems and instruments

- Refer to an earth stations FK for a list of frames defined for the DSN and other stations

- Refer to the moon FKs for descriptions of the body-fixed frames defined for the moon

# Programming Task

- Complete steps #5 and #6 from the in-situ programming lesson.
- Try step #6 with the RBSP_IGRF_MAG and RBSP_GSE frames instead. Note: these additional frames are defined in the RBSP dynamic frame kernel.

# Questions?

**This concludes the prepared lesson material for the workshop.**

# Additional Material

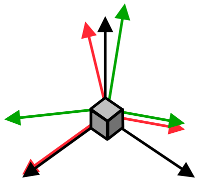
(if time permits)

# Matrix Conversions

Navigation and Ancillary Information Facility

## Function

## Routines



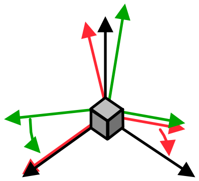
Euler angles

Transform between

$$\begin{array}{ccc} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{array}$$

3x3 rotation matrix

– EUL2M, M2EUL



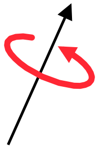
Euler angles and Euler angle rates
*or*
rotation matrix and angular velocity vector

Transform between

$$\begin{array}{cccccc} a_x & a_y & a_z & & & \\ b_x & b_y & b_z & & 0 & \\ c_x & c_y & c_z & & & \\ \alpha_x & \alpha_y & \alpha_z & a_x & a_y & a_z \\ \beta_x & \beta_y & \beta_z & b_x & b_y & b_z \\ \gamma_x & \gamma_y & \gamma_z & c_x & c_y & c_z \end{array}$$

6x6 state transformation matrix

– EUL2XF, XF2EUL
RAV2XF, XF2RAV



Rotation axis and angle

Transform between

$$\begin{array}{ccc} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{array}$$

3x3 rotation matrix

– RAXISA, AXISAR
ROTATE, ROTMAT

$(Q_0,Q_1,Q_2,Q_3)$

SPICE Style Quaternion

Transform between

$$\begin{array}{ccc} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{array}$$

3x3 rotation matrix

– Q2M, M2Q

# Geometry

| Function | Routine |
|---|---|
| • Ellipsoids | |
| – nearest point | – NEARPT, SUBPNT, DNEARP |
| – surface ray intercept | – SURFPT, SINCPT |
| – surface normal | – SURFNM |
| – limb | – EDLIMB |
| – slice with a plane | – INELPL |
| – altitude of ray w.r.t. to ellipsoid | – NPEDLN |
| • Planes | |
| – intersect ray and plane | – INRYPL |
| • Ellipses | |
| – project onto a plane | – PJELPL |
| – find semi-axes of an ellipse | – SAELGV |

# Position Coordinate Transformations

| Coordinate Transformation | Routine |
|---|---|
| – Latitudinal to/from Rectangular | – LATREC<br>RECLAT |
| – Planetographic to/from Rectangular | – PGRREC<br>RECPGR |
| – R.A. Dec to/from Rectangular | – RADREC<br>RECRAD |
| – Geodetic to/from Rectangular | – GEOREC<br>RECGEO |
| – Cylindrical to/from Rectangular | – CYLREC<br>RECCYL |
| – Spherical to/from Rectangular | – SPHREC<br>RECSPH |

# Velocity Coordinate Transformations - 1

- **Coordinate Transformation**
  - Latitudinal to/from Rectangular
  - Planetographic to/from Rectangular
  - R.A. Dec to/from Rectangular
  - Geodetic to/from Rectangular
  - Cylindrical to/from Rectangular
  - Spherical to/from Rectangular

- **Jacobian (Derivative) Matrix Routine**
  - `DRDLAT`  
    `DLATDR`
  - `DRDPGR`  
    `DPGRDR`
  - `DRDLAT*`  
    `DLATDR*`
  - `DRDGEO`  
    `DGEODR`
  - `DRDCYL`  
    `DCYLDR`
  - `DRDSPH`  
    `DSPHDR`

* Jacobian matrices for the R.A and Dec to/from rectangular mappings are identical to those for the latitudinal to/from rectangular mappings

# Velocity Coordinate Transformations - 2

- **Transform velocities from one coordinate system to another using the SPICE Jacobian matrix routines. Example:**
  - Let ( x, y, z ) be a time-dependent vector expressed in rectangular coordinates:

    $( x, y, z ) = \Gamma(\mathbf{t})$
  - Let $( \alpha, \beta, \gamma )$ be the same vector expressed in spherical coordinates:

    $( \alpha, \beta, \gamma ) = \Phi ( \Gamma(\mathbf{t}) )$
  - Then the chain rule gives us the time derivative of the position in spherical coordinates:

    $d (\Phi( \Gamma(\mathbf{t}) ) ) / dt \quad = \quad J (\Gamma(\mathbf{t})) \ * \ d\, \Gamma(\mathbf{t}) / dt$
  - The left hand side above is the velocity in spherical coordinates. The first factor on the right is the "Jacobian matrix" of the mapping from rectangular to spherical coordinates; the second factor on the right is the velocity in rectangular coordinates.

Continues on next page

# Velocity Coordinate Transformations - 3

- The SPICE calls that implement this computation are:

```
CALL SPKEZR  (  TARG,  ET,  REF,  CORR,  OBS,  STATE,  LT  )
CALL DSPHDR (  STATE(1), STATE(2),  STATE(3), JACOBI  )
CALL MXV        (  JACOBI,    STATE(4),  SPHVEL  )
```

- After these calls, the vector SPHVEL contains the velocity in spherical coordinates:  specifically, the derivatives

  ( d (r) / dt,    d (colatitude) / dt,    d (longitude) /dt  )

- Caution:  coordinate transformations often have singularities, so derivatives may not exist everywhere.
  - Exceptions are described in the headers of the SPICE Jacobian matrix routines.
  - SPICE Jacobian matrix routines signal errors if asked to perform an invalid computation.