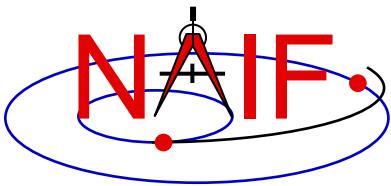


---

Navigation and Ancillary Information Facility

# Welcome to the **SPICE Tutorials**

**January 2020**

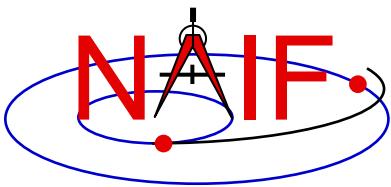


# Objectives

---

Navigation and Ancillary Information Facility

- Provide an **overview** of the entire SPICE system
- Provide a sense of the purpose and **uses** of SPICE
- Provide an introduction to the primary SPICE **components**
- Provide **examples** of how to use SPICE software and data files
- Provide some insight into **conventions** and **common problems**
- Provide a substantive **programming example**
- Provide a **peek at new capabilities** being worked on or considered
- Familiarize you with available **SPICE resources**
  
- Going through these extensive tutorials and the associated programming lessons will not be enough to make you well versed in SPICE – it's just a significant start towards that goal.

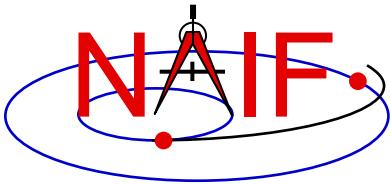


# Tutorials Scope

---

Navigation and Ancillary Information Facility

- **Broad coverage**
  - Begins at a high level, but quickly drills down to details
  - Touches on many SPICE-related topics that could be of interest to science and engineering teams
    - » Depth of discussion varies somewhat amongst topics
- **Provide information for FORTRAN, C, IDL and MATLAB programmers**
  - Does not cover Java Native Interface (JNISpice), or the Python (e.g. SpiceyPy), Ruby, Swift and Julia 3rd party offerings.
- **Some tutorials have important material provided in a “Backup” section: you should read those charts too!**
- **Some topics are addressed rather little or not at all**
  - Kernel production
  - Archiving SPICE data



# Repeated Material

---

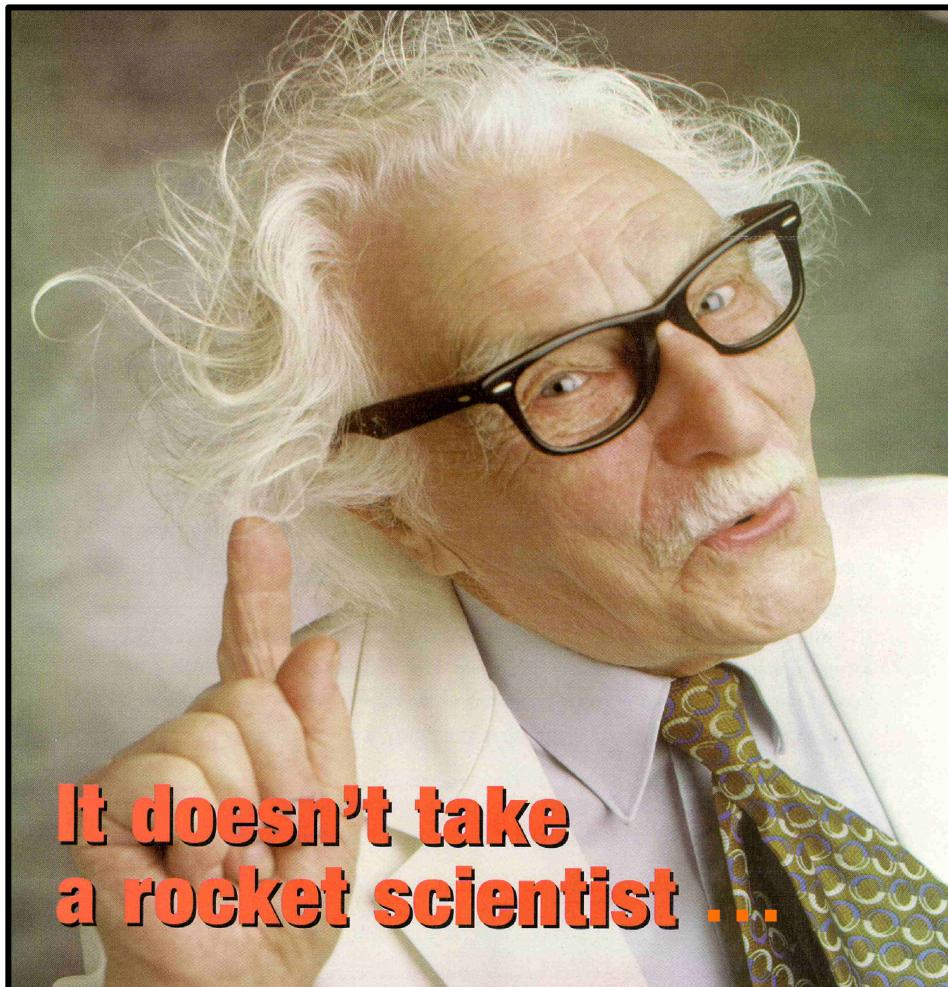
Navigation and Ancillary Information Facility

- **Some topics will be repeated in two or more tutorials**
  - We're not trying to bore you, but...
    - » we don't wish to assume people will read all of the tutorials at the same time
    - » we think some items are sufficiently important to mention them more than once



# Your SPICE Odyssey Begins Here

Navigation and Ancillary Information Facility



**... but it does take a modest amount of effort to learn enough about SPICE to begin to use its features with good success.**

**It helps to have some math skills, some innate sense of spatial orientation, and some familiarity with your computer's operating system, a code editor, and a compiler or Integrated Development Environment (IDE).**

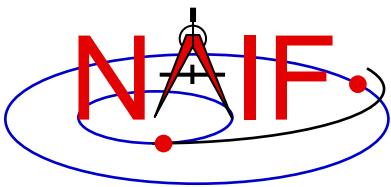


# SPICE Seems “Large”

---

Navigation and Ancillary Information Facility

- **The generic SPICE Toolkit contains:**
  - several hundred public APIs (“modules” or “subroutines”)
  - about 17 utility and application executables
  - about 26 subsystem technical reference documents
  - and more
- **Don’t let this size bother you... just work your way into it bit by bit**
- **Most customers use only a handful of these APIs**
- **Hundreds of others just like you are using SPICE today**



# The NAIF Team at JPL

---

Navigation and Ancillary Information Facility



**Chuck Acton**



**Nat Bachman**



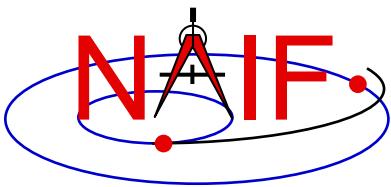
**Fraser Thomson**



**Boris Semenov**



**Ed Wright**

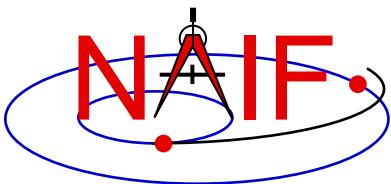


---

Navigation and Ancillary Information Facility

# Motivation for Developing SPICE

January 2020

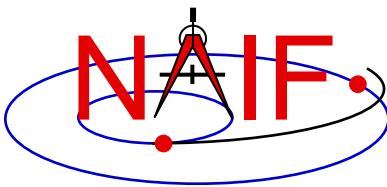


# Why Did NAIF Build SPICE?

---

Navigation and Ancillary Information Facility

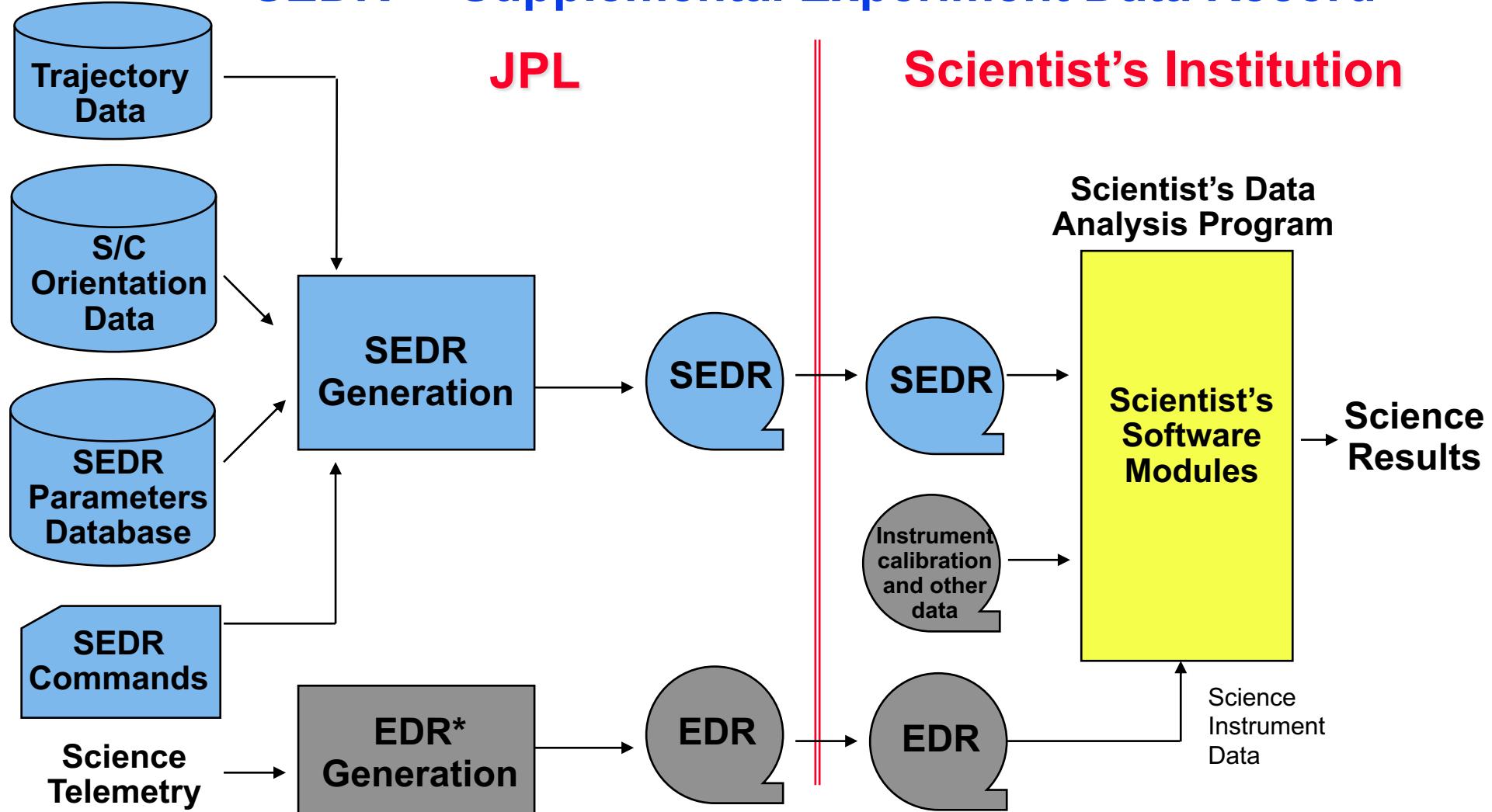
- **Scientists said they would like to:**
  - use common observation geometry computation tools and methods throughout a project's lifecycle, and for all projects (national and international)
  - be able to produce custom observation geometry calculations themselves, whenever and however they want
  - understand the calculations and data used to produce observation geometry data
  - have the ability to revise the data and software tools used to produce their own observation geometry data



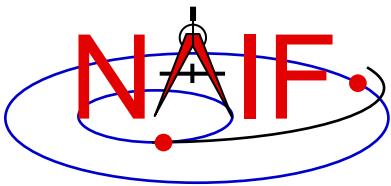
# What Existed Prior to SPICE ?

Navigation and Ancillary Information Facility

## “SEDR” - Supplemental Experiment Data Record



\* EDR = Experiment Data Record = "raw" science instrument data

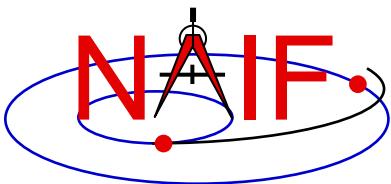


# SEDR System Characteristics

---

Navigation and Ancillary Information Facility

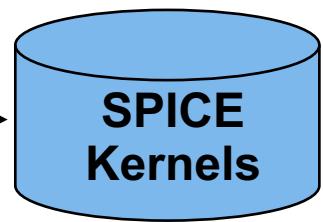
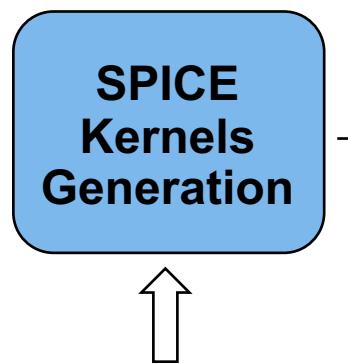
- **The SEDR generation program was built and operated at JPL**
  - Scientist's requirements on SEDR had to be provided long before launch
    - » Late or post-launch updates were hard/expensive to accommodate
      - Difficult to change WHAT gets computed
      - Difficult to change HOW items are computed (algorithms, parameters)
      - Difficult to change the TIMES at which items get computed
    - » Result: the scientist can't get better observation geometry data if/when better inputs (e.g. spacecraft trajectory or orientation, etc.) become available
  - SEDR generation was done “in the blind”
    - » Operators were not familiar with processes used to make the inputs
    - » Operators were not familiar with scientist's processing schemes
    - » Result: SEDR often did not fully meet science team's expectations
  - The SEDR system was not exportable to other institutions



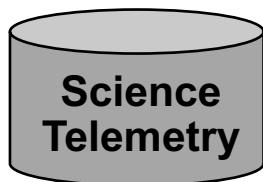
# The SPICE Idea

Navigation and Ancillary Information Facility

## Any Mission Operations Center

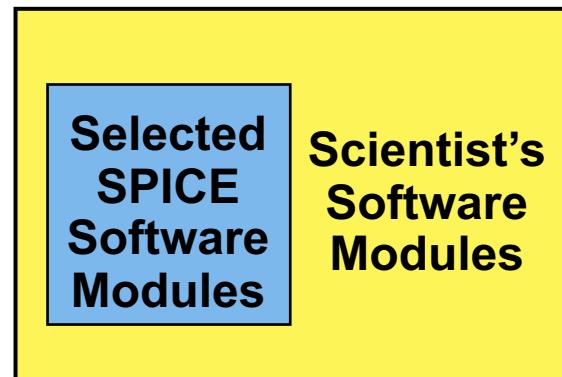


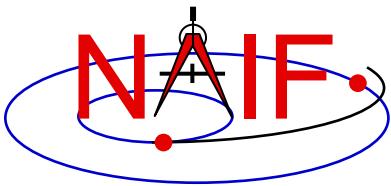
Target body ephemeris  
Target body size/shape/orientation  
S/C trajectory  
S/C orientation  
Instrument geometry  
Reference frame specs



## Scientist's Institution

Scientist's Data Analysis Program



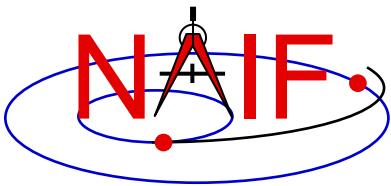


# SPICE Benefits vs. SEDR

---

Navigation and Ancillary Information Facility

- **The customer has great flexibility in deciding:**
  - what observation geometry parameters are computed
  - at what times or at what frequency these parameters are computed
  - for what time span(s) these parameters are computed
  - electing if/when to re-do parameter computations using new (better) or otherwise different data as inputs
- **The customer also has:**
  - multimission tools and methods that can be reused on many tasks
  - full visibility into algorithms and data used in geometry calculations
- **The flight project operations center can:**
  - concentrate on producing better ancillary data, rather than on producing lots of SEDRs and frequently updating the SEDR software
- **The SPICE process may be replicated anywhere**

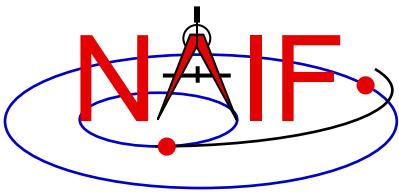


# SPICE “Challenges” vs. SEDR

---

Navigation and Ancillary Information Facility

- There are often many SPICE data files produced by the mission
  - It can be a challenge to select the “correct” ones for a particular job
- Customers must do some non-trivial programming to read SPICE data and compute whatever is needed
- If the mission operations center is other than JPL, the appropriate project people need to learn how to produce SPICE data
- In some areas of SPICE, the offering of choices to allow correct handling of different situations may present complexity that is unwarranted for “simple” problems



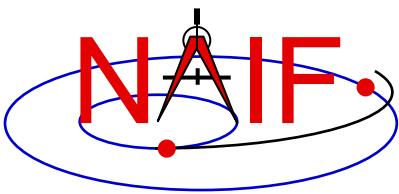
---

Navigation and Ancillary Information Facility

# An Overview of SPICE

**NASA's Observation Geometry System  
for Space Science Missions**

**January 2020**

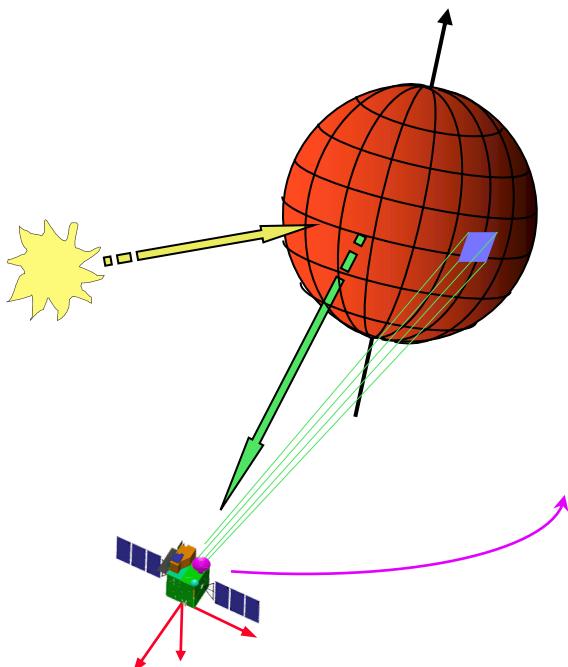


# What Can One Do With SPICE?

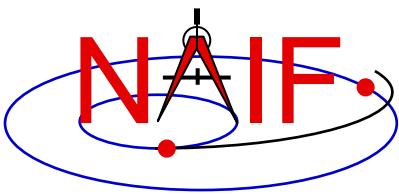
Navigation and Ancillary Information Facility

**Compute many kinds of observation geometry parameters at selected times**

## *Examples*



- Positions and velocities of planets, satellites, comets, asteroids and spacecraft
- Size, shape and orientation of planets, satellites, comets and asteroids
- Orientation of a spacecraft and its various moving structures
- Instrument field-of-view location on a planet's surface or atmosphere

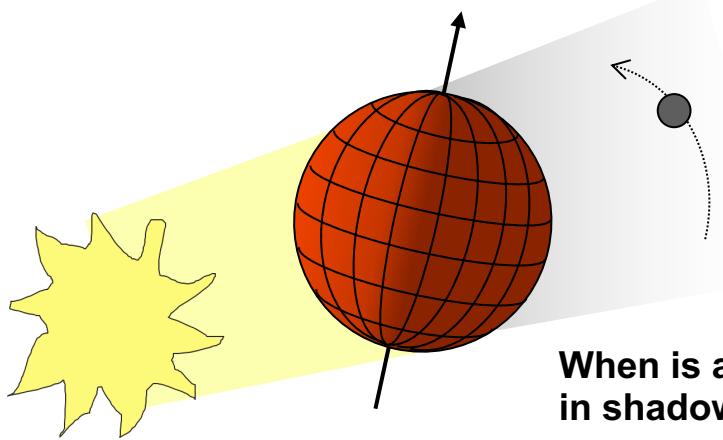


# What One Can Do With SPICE

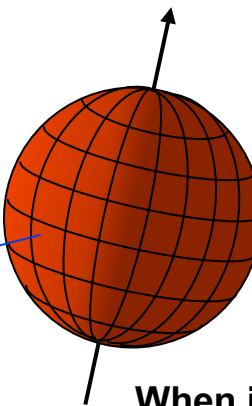
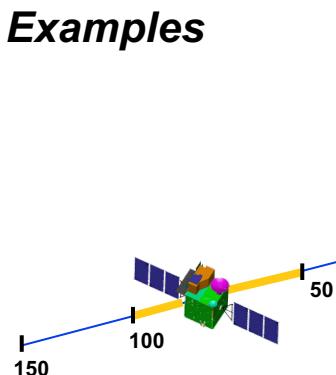
Navigation and Ancillary Information Facility

Find times when a specified “geometric event” occurs

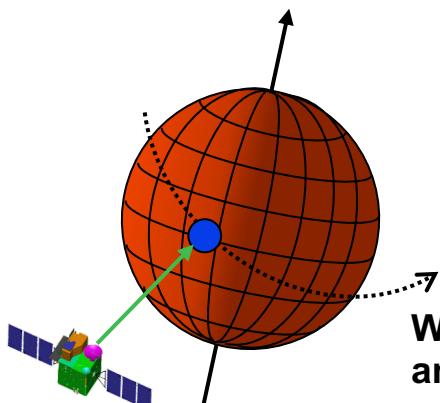
## Examples



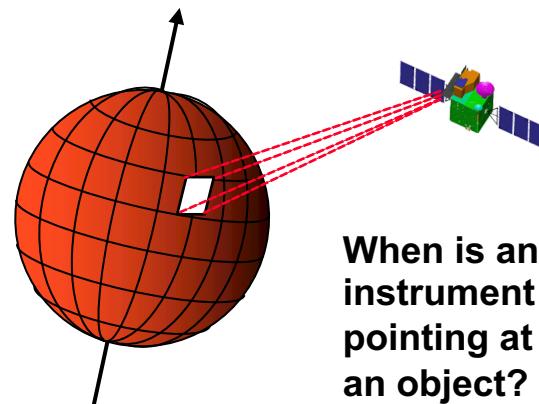
When is an object  
in shadow (occultation) ?



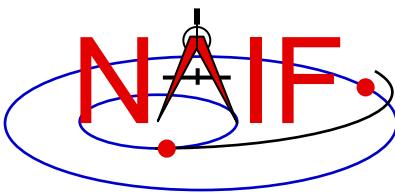
When is the spacecraft’s  
altitude within a given  
range (say 50 to 100 km)?



When is an object in front of  
another, as seen from a  
spacecraft (transit) ?

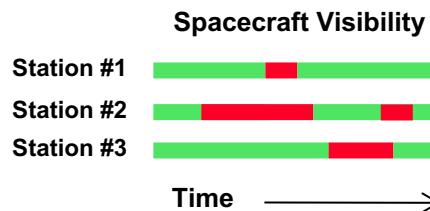


When is an  
instrument  
pointing at  
an object?

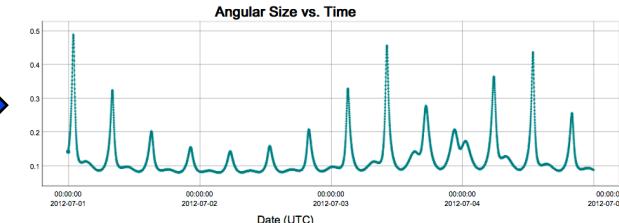


# Examples of How SPICE Is Used

Navigation and Ancillary Information Facility

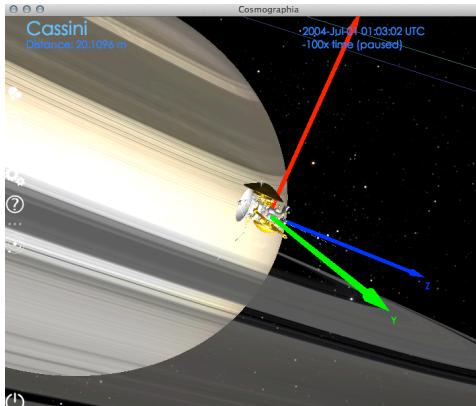


Evaluation of a planned trajectory

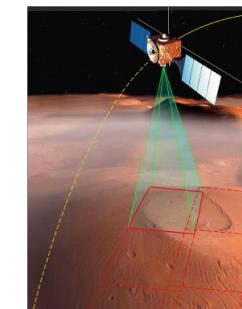


Angular size of Phobos as seen from the MEX spacecraft

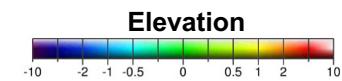
Mission engineering analyses



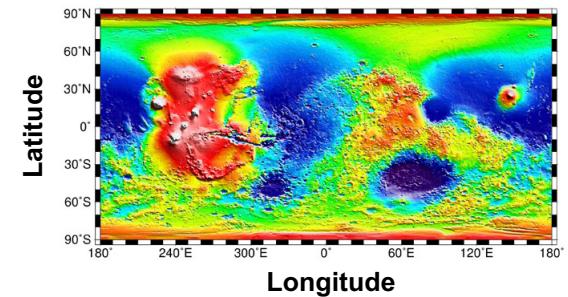
Planning an instrument pointing profile

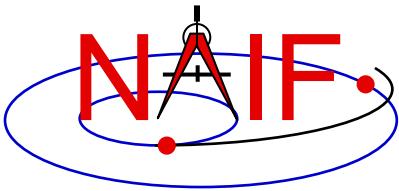


Observation geometry visualization



Science data archiving and analysis





# SPICE Pictorial Summary

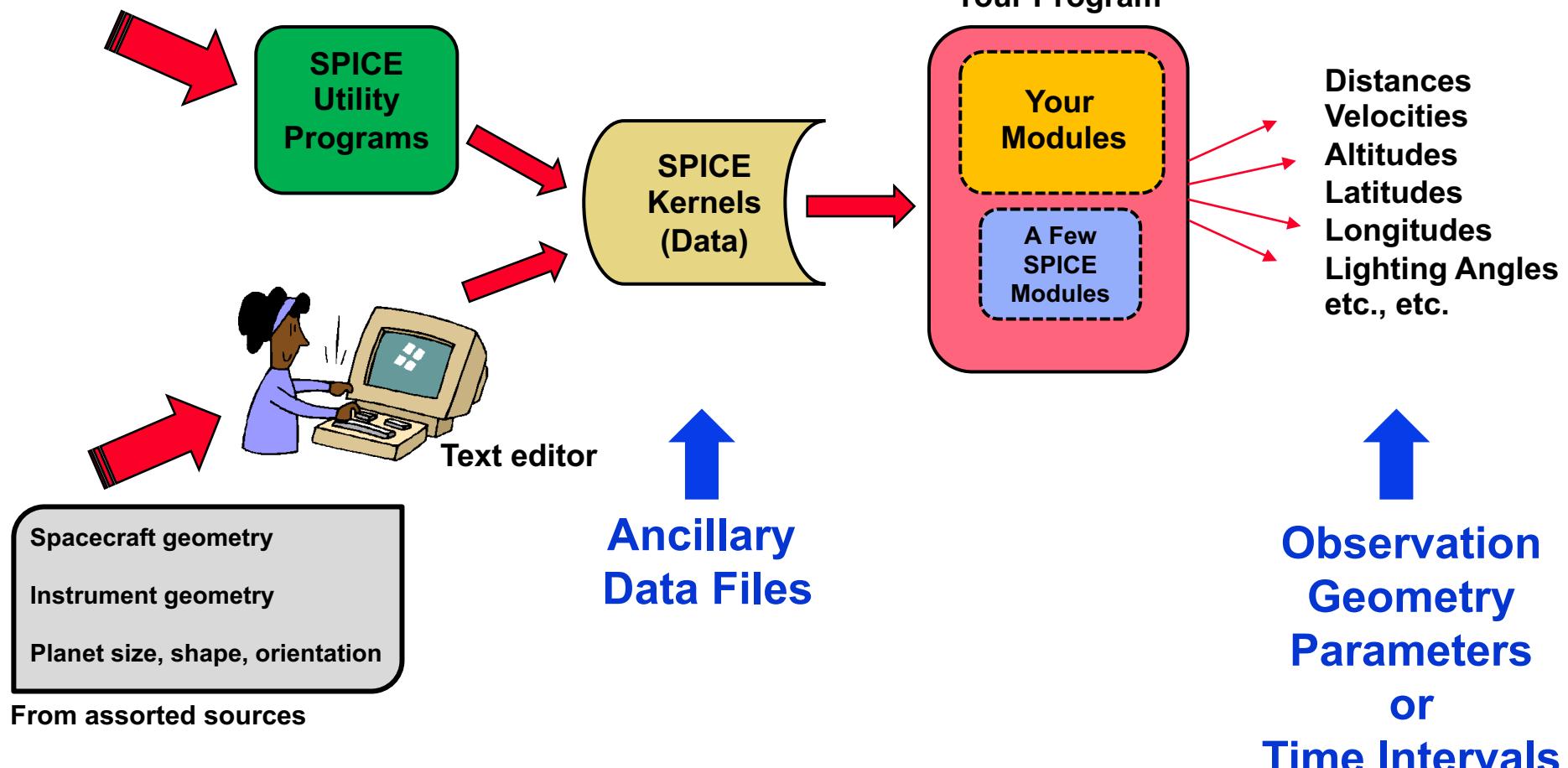
Navigation and Ancillary Information Facility

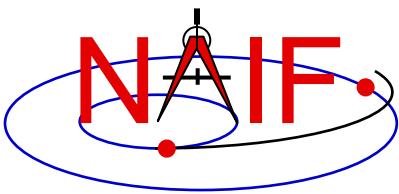
From assorted sources

Planet ephemeris

S/C trajectory

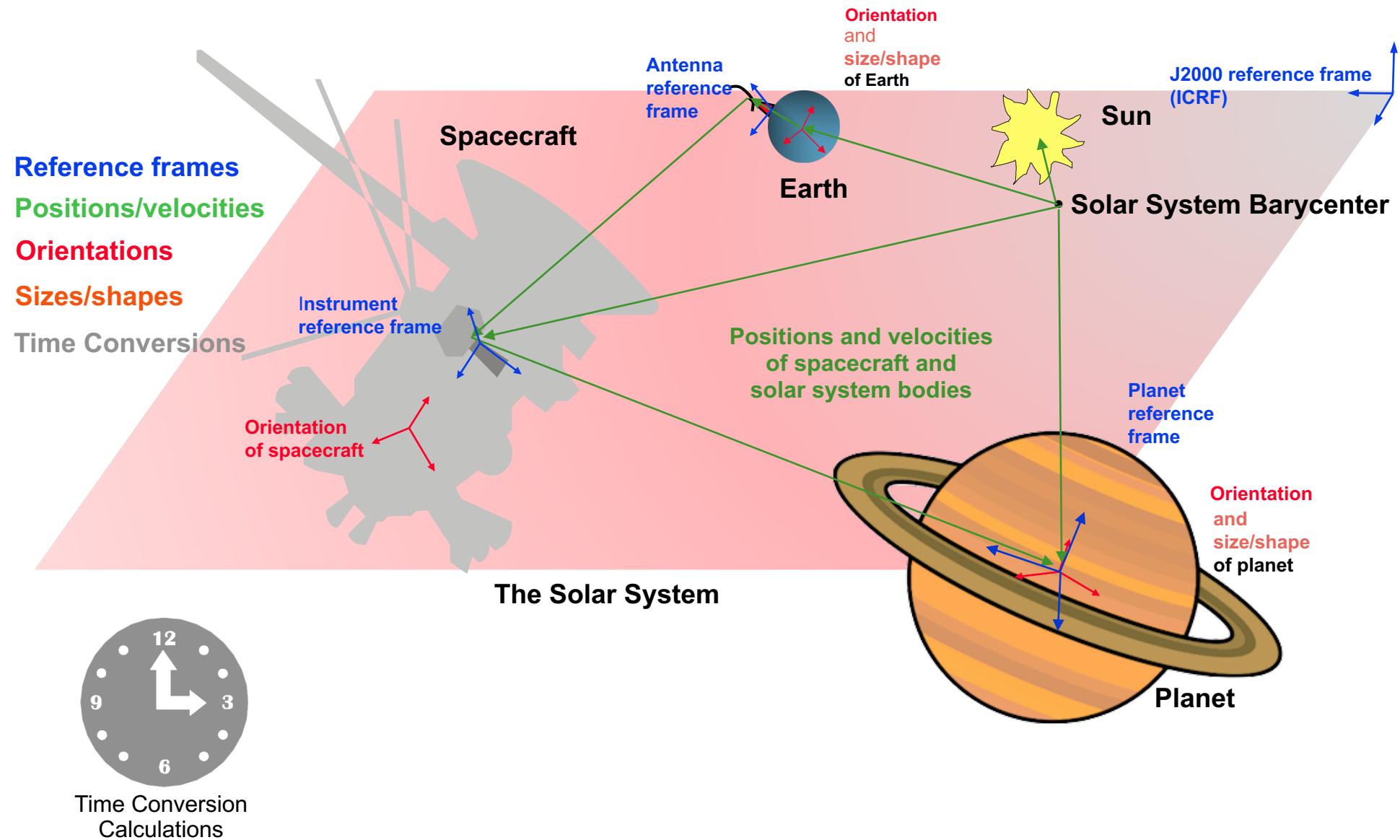
S/C orientation



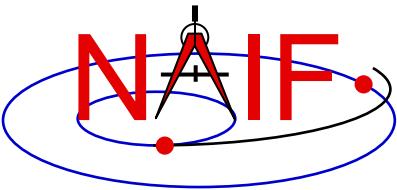


# What are “Ancillary Data?”

Navigation and Ancillary Information Facility



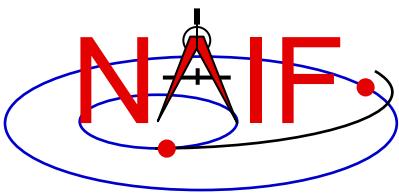
Time Conversion  
Calculations



# How Use Ancillary Data?

Navigation and Ancillary Information Facility

- **Ancillary data** are those that help scientists and engineers determine **observation geometry**, such as:
  - where the spacecraft was located
  - how the spacecraft and its instruments were oriented (pointed)
  - what was the location, size, shape and orientation of the target being observed
  - where on the surface the instrument was looking
- The text above uses past tense, but doing the same functions for future times to support mission planning is equally applicable



# From Where do Ancillary Data Come?

Navigation and Ancillary Information Facility

- From the spacecraft



- From the mission control center



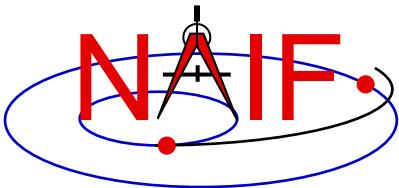
- From the spacecraft and instrument builders



- From science organizations



- SPICE is used to organize and package these data in a collection of stable file types—called “kernels”—used by scientists and engineers

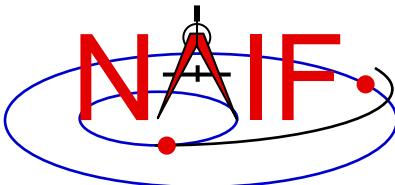


# Why Use SPICE?

---

Navigation and Ancillary Information Facility

- **Knowing observation geometry and geometric events is an important element of:**
  - space mission design,
  - selection of observation opportunities,
  - analysis of the science data returned from the instruments,
  - mission engineering activities, and
  - preparation of science data archives.
- **Having a proven, extensive and reusable means for producing and using ancillary data reduces cost and risk, and can help scientists and engineers achieve more substantive, accurate and timely results.**



# SPICE System Components

Navigation and Ancillary Information Facility

**Ancillary data files (“kernels”)** .....

1100  
1010  
0101



**Software (SPICE Toolkit)** .....



**Documentation** .....



**Tutorials** .....



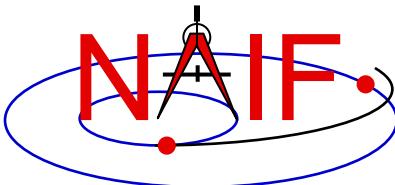
**Programming lessons** .....



**Training classes** .....



**User consultation** .....



# Origin of the SPICE Acronym\*

Navigation and Ancillary Information Facility

S

Spacecraft

P

Planet

I

Instrument

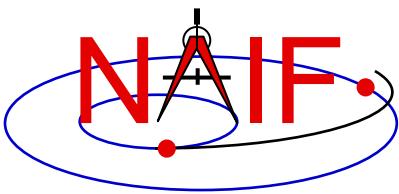
C

C-matrix (“Camera matrix”)

E

Events

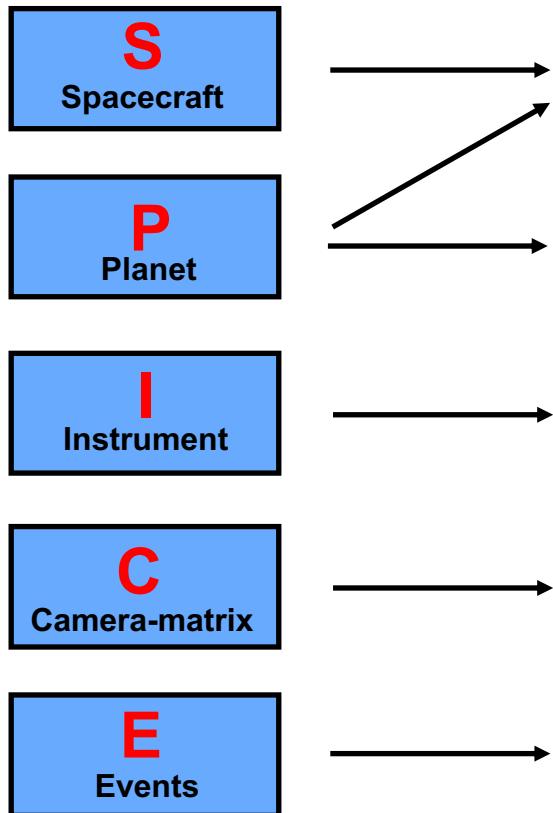
\* Coined by Dr. Hugh Kieffer, USGS Astrogeology Branch, Flagstaff AZ, circa 1985



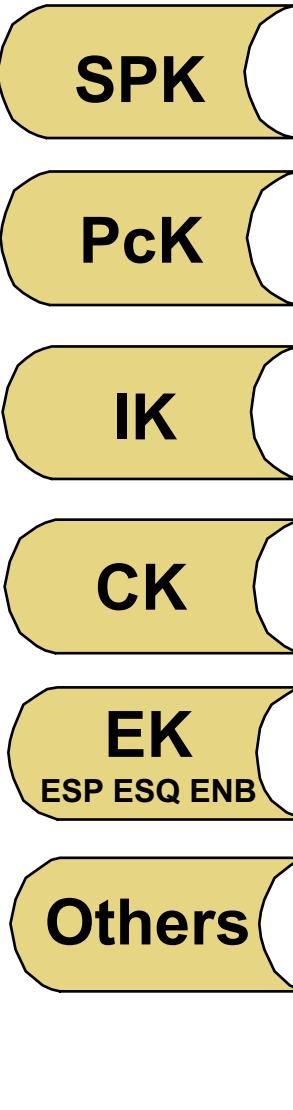
# SPICE Data Overview

Navigation and Ancillary Information Facility

## Logical Components



## Kernels



## Contents

Space vehicle or target body trajectory (ephemeris)

Target body size, shape and orientation

Instrument field-of-view size, shape and orientation

Orientation of space vehicle or any articulating structure on it

Events information:  
- Science Plan (ESP)  
- Sequence of events (ESQ)  
- Experimenter's Notebook (ENB)

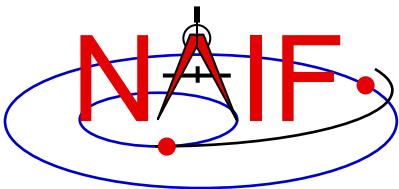
Reference frame specifications

Leapseconds tabulation

Spacecraft clock coefficients

Digital shape models

Meta-kernel



# SPICE Kernels Details- 1

Navigation and Ancillary Information Facility

**SPK**

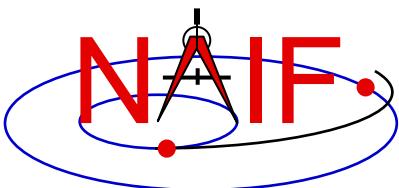
- Space vehicle ephemeris (trajectory)
- Planet, satellite, comet and asteroid ephemerides
- More generally, position of something relative to something else

**PcK**

- Planet, satellite, comet and asteroid orientations, sizes, shapes
  - See also **DSK**
- Possibly other similar “constants” such as parameters for gravitational model, atmospheric model or rings model

**IK**

- Instrument field-of-view size, shape, orientation
- Possibly additional information, such as internal timing



# SPICE Kernels Details- 2

Navigation and Ancillary Information Facility

CK

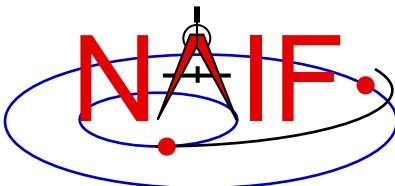
- Instrument platform (e.g. spacecraft) attitude
- More generally, orientation of something relative to a specified reference frame

EK

3 components

- “Events,” broken into three components:
  - ESP: Science observation plans
  - ESQ: Spacecraft & instrument commands
  - ENB: Experiment “notebooks” and ground data system logs

EK is not much used



# SPICE System Data - 3

Navigation and Ancillary Information Facility

**FK**

- **Frames**

- Definitions of and specification of relationships between reference frames (coordinate systems)
  - Both “fixed” and “dynamic” frames are available

**LSK**

- **Leapseconds Tabulation**

- Used for UTC <--> TDB (ET) time conversions

**SCLK**

- **Spacecraft Clock Coefficients**

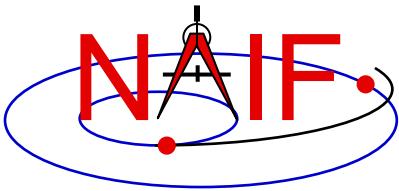
- Used for SCLK <--> TDB (ET) time conversions

**DSK**

- **Shape models (tessellated plate model and digital elevation model\*) (DSK)**

\*DEM portion under development

UTC = Coordinated Universal Time    TDB = Barycentric Dynamical Time    ET = Ephemeris Time    SCLK = Spacecraft Clock Time

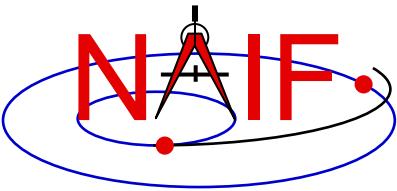


# SPICE System Data - 4

Navigation and Ancillary Information Facility



- **Meta-kernel**
  - A means to conveniently reference a collection of real kernels you would like to use together



# SPICE Toolkit Software

Navigation and Ancillary Information Facility

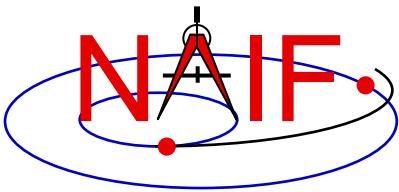
## Contents

- **Library of subroutines**
  - But typically just a few are used within a customer's program to compute quantities derived from SPICE data files
- **Programs**
  - SPICE data production
  - SPICE data management
- **Documentation**
  - Highly annotated source code
  - Technical Reference Manuals
  - User Guides

## Versions

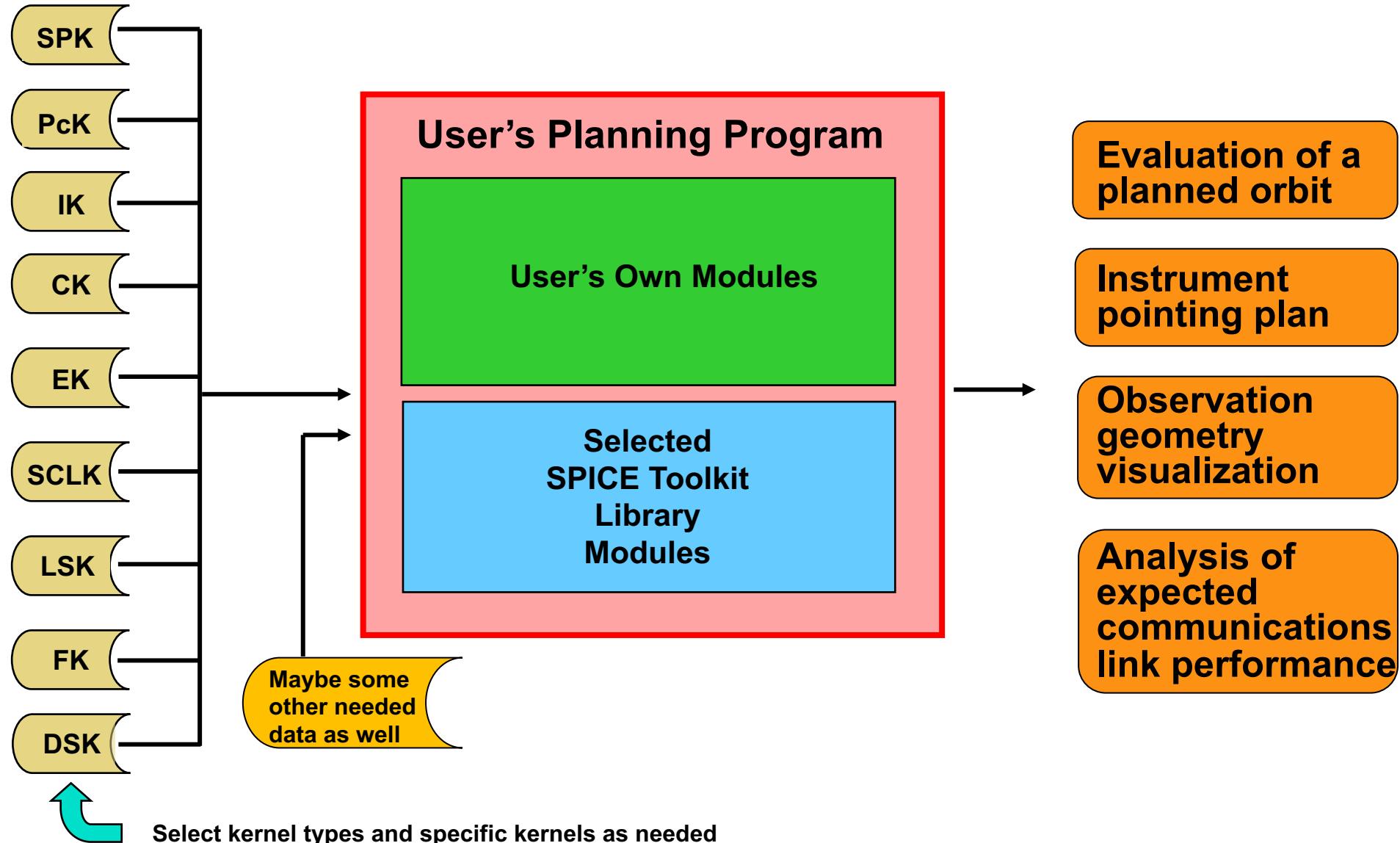
- **Nine languages**
  - Fortran 77
  - C
  - IDL
  - MATLAB
  - Java Native Interface (JNI)
  - Python, Ruby, Swift, Julia  
(provided by 3<sup>rd</sup> parties)
- **Four platforms**
  - PC/Linux
  - PC/Windows
  - Sun/Solaris
  - Mac/OSX
- **Several compilers**
  - For the Fortran and C Toolkits

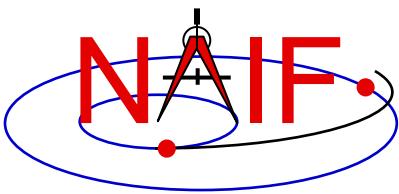
All combinations provided by NAIF are fully built and individually tested before being made available to customers



# Using SPICE: Mission Planning Example

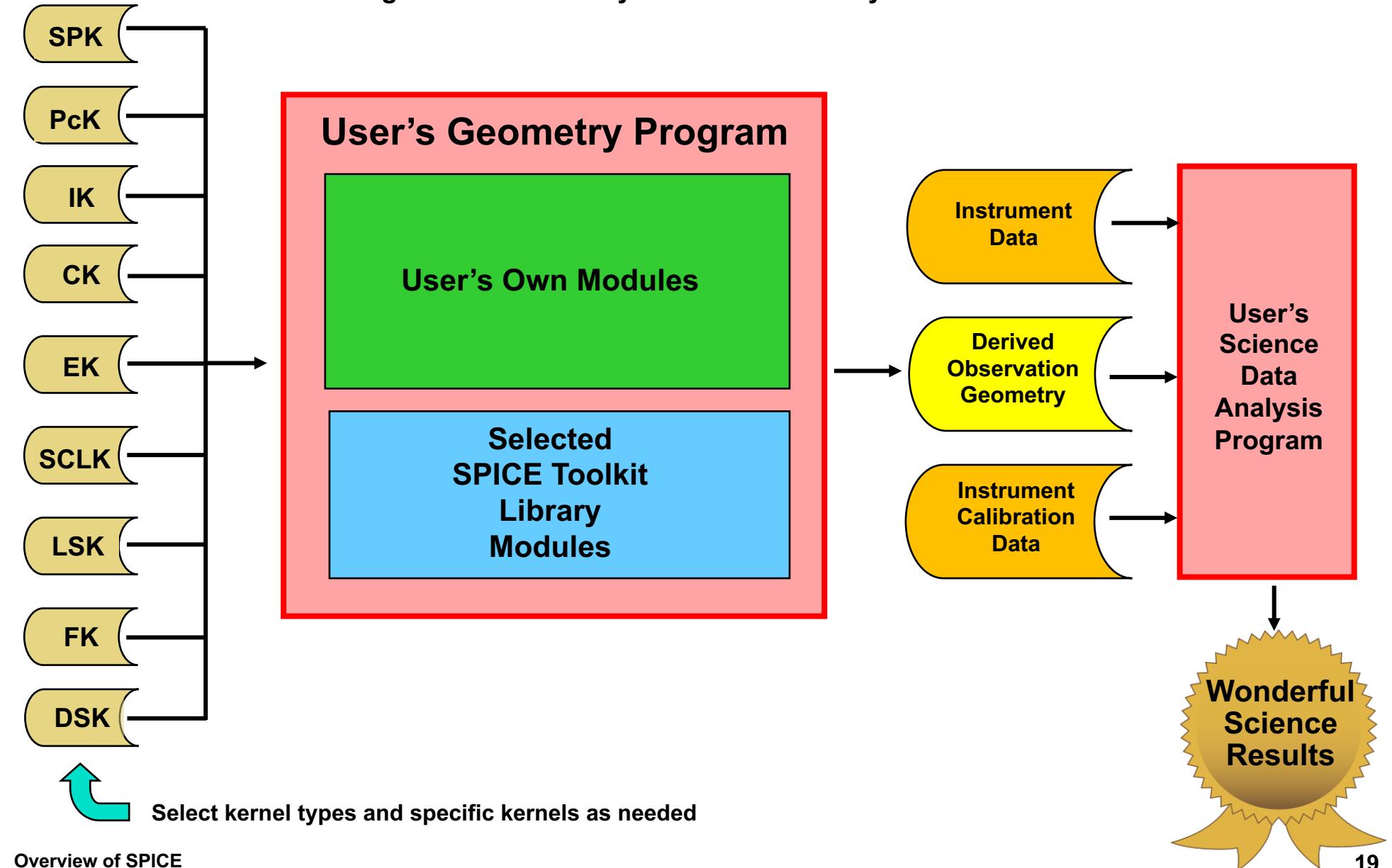
Navigation and Ancillary Information Facility

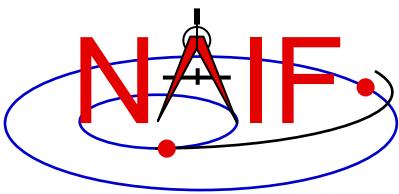




# Using SPICE: Science Data Analysis Example

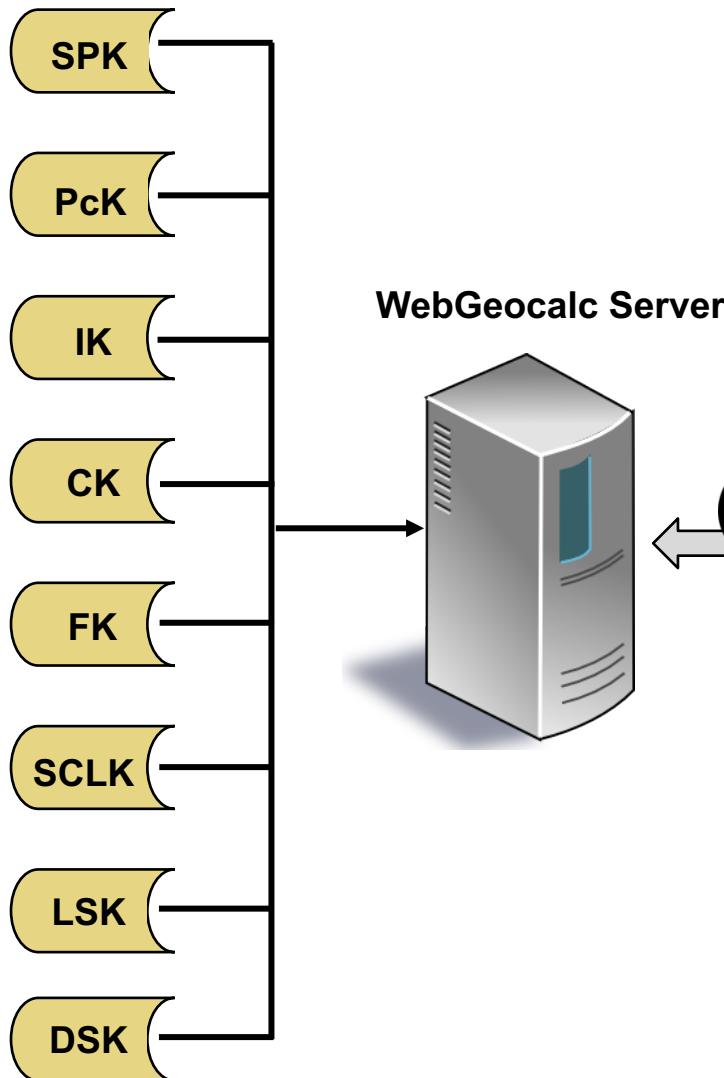
Navigation and Ancillary Information Facility





# Using SPICE: Science Data Peer Review Example

Navigation and Ancillary Information Facility



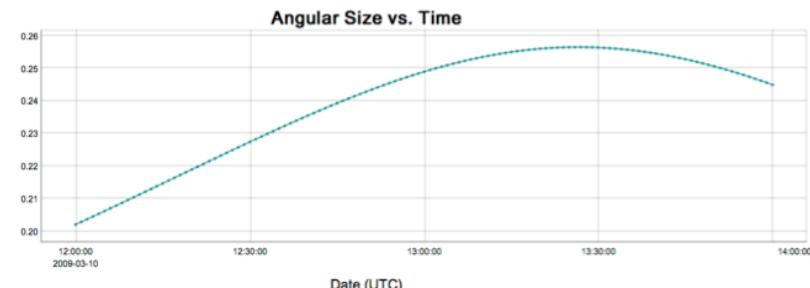
## Tabular Results

Click a value to save it for a subsequent calculation.

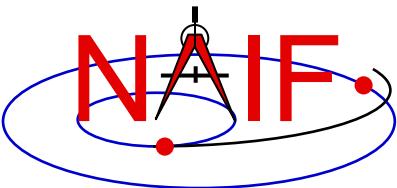
	UTC calendar date	Angular Size (deg)
1	2009-03-10 12:00:00.000000 UTC	0.20212256
2	2009-03-10 12:01:00.000000 UTC	0.20294481
3	2009-03-10 12:02:00.000000 UTC	0.20377024
4	2009-03-10 12:03:00.000000 UTC	0.20459871
5	2009-03-10 12:04:00.000000 UTC	0.20543007
6	2009-03-10 12:05:00.000000 UTC	0.20626418
7	2009-03-10 12:06:00.000000 UTC	0.20710088
8	2009-03-10 12:07:00.000000 UTC	0.20794000
9	2009-03-10 12:08:00.000000 UTC	0.20878138
10	2009-03-10 12:09:00.000000 UTC	0.20962484
11	2009-03-10 12:10:00.000000 UTC	0.21047019
12	2009-03-10 12:11:00.000000 UTC	0.21131725
13	2009-03-10 12:12:00.000000 UTC	0.21216581

## Numeric Results

## Graphic Results



Angular size of Phobos as seen from the Mars rover "SPIRIT"

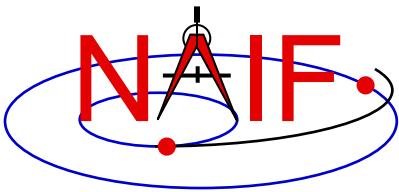


# SPICE System Characteristics - 1

---

Navigation and Ancillary Information Facility

- SPICE Toolkit software is portable between computers
- New Toolkits are released irregularly, when enough new capability warrants it
- Code is very well tested before being released to users
- New Toolkits are always 100% backwards compatible
- Source code is provided, and is well documented
- Extensive user-oriented documentation is provided
- Software includes built-in exception handling
  - Catches most invalid inputs

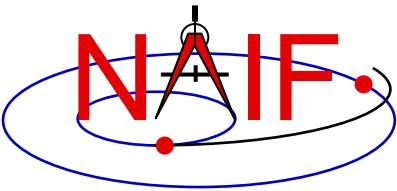


# SPICE System Characteristics - 2

Navigation and Ancillary Information Facility

- All numeric computations are double precision
- Kernel files are portable between computers
- Kernel files are separable
  - Use only those you need for a particular application
- SPICE kernels and software are free of licensing and U.S. ITAR restrictions
  - Everyone is free to use SPICE
- No cost to individual end users

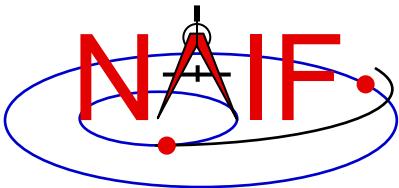




# Supported Environments

Navigation and Ancillary Information Facility

- **The SPICE Toolkit has been ported to many popular “environments”**
  - Each environment is characterized by...
    - » Language
    - » Hardware type (platform)
    - » Operating System
    - » Compiler (where applicable)
    - » Selected compilation options (32-bit or 64-bit)
- **NAIF provides separate, ready-built SPICE Toolkit packages for each supported environment**
  - If you need to port the Toolkit to a new environment yourself, consult with NAIF staff first

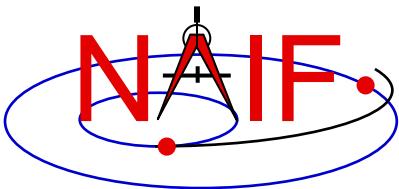


# What “Vehicle” Types Can Be Supported?

Navigation and Ancillary Information Facility

- **Cruise/Flyby**
  - Remote sensing
  - In-situ measurement
  - Instrument calibration
- **Orbiters**
  - Remote sensing
  - In-situ measurement
  - Communications relay
- **Balloons\***
  - Remote sensing
  - In-situ measurements
- **Landers**
  - Remote sensing
  - In-situ measurements
  - Rover or balloon relay
- **Rovers**
  - Remote sensing
  - In-situ sensing
  - Local terrain characterization
- **Terrestrial applications**
  - Ephemerides for telescopes
  - Radiometric tracking & comm
  - Optical tracking & comm

\* Not yet demonstrated

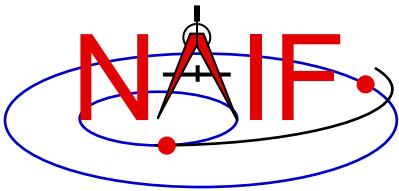


# More Than Just Planetary Science

---

Navigation and Ancillary Information Facility

- Today SPICE is used well beyond just planetary science missions.
  - Heliophysics
  - Earth science
  - Observations from terrestrial observatories
  - Space technology demos
  - Planetariums
  - Probably still more...?

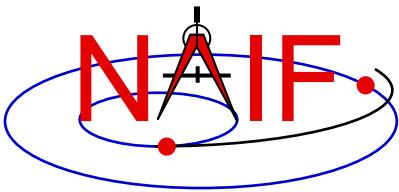


# History

---

Navigation and Ancillary Information Facility

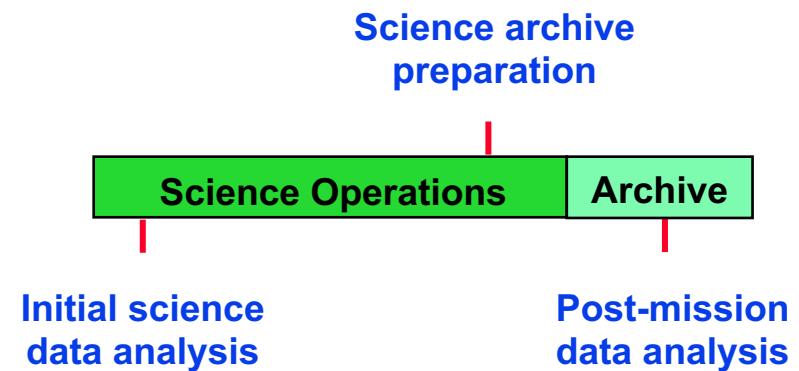
- A SPICE precursor was initiated in 1984 as part of a major initiative to improve archiving and distribution of space science data in all NASA disciplines
- Responsibility for leading SPICE development was assigned to the newly-created Navigation and Ancillary Information Facility (NAIF), located at the Jet Propulsion Laboratory
- Today's SPICE system dates from about 1991



# Original Purpose for SPICE

Navigation and Ancillary Information Facility

- The original focus of SPICE was on ancillary data and associated software needed by planetary scientists for:
  - science data analysis, both during and after the mission operations
  - science archive preparation

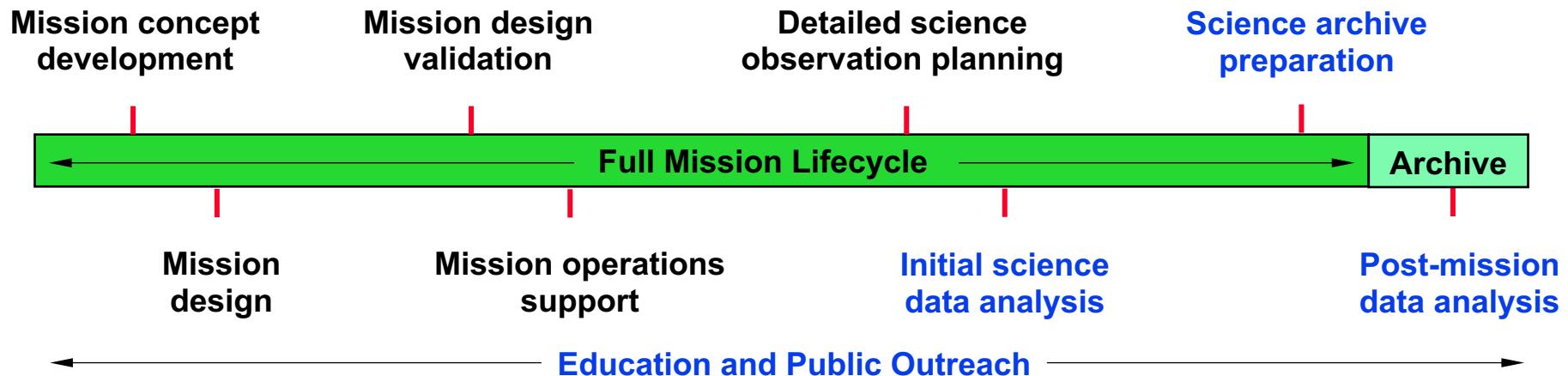


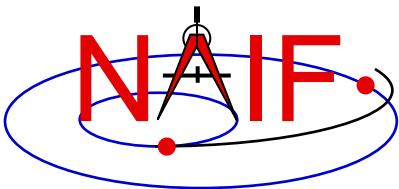


# Large Breadth of Use

Navigation and Ancillary Information Facility

- The original focus of SPICE was on ancillary data and associated software needed by planetary scientists for:
  - science data analysis, both during and after the mission operations
  - science archive preparation
- The scope of SPICE usage has grown to cover the full mission lifecycle.
- Also education and public outreach.



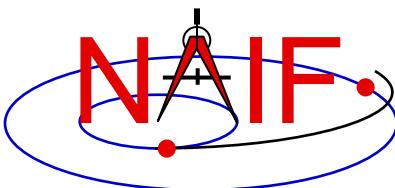


# Ancillary Data Archives

---

Navigation and Ancillary Information Facility

- **SPICE is the U.S. Planetary Data System's recommendation for archiving ancillary data**
- **Use of SPICE is recommended by the International Planetary Data Alliance**
- **SPICE data for European planetary missions are archived in ESA's Planetary Science Archive**
  - Some of these data are also mirrored on the NAIF server
- **SPICE data for some Japanese, Indian and Russian missions may be available from their local archives**



# SPICE Users

## Navigation and Ancillary Information Facility

Data Restorations	Selected Past Users	Current/Pending Users	Examples of Possible Future Users
Apollo 15, 16 [L]	Magellan [L]	Cassini Orbiter	NASA Discovery Program
Mariner 2 [L]	Clementine (NRL)	Mars Odyssey	NASA New Frontiers Program
Mariner 9 [L]	Mars 96 (RSA)	Mars Exploration Rover	Lunar IceCube (Moorehead State)
Mariner 10 [L]	Mars Pathfinder	Mars Reconnaissance Orbiter	LunaH-Map (Arizona State)
Viking Orbiters [L]	NEAR	Mars Science Laboratory	Luna-Glob (RSA)
Viking Landers [L]	Deep Space 1	Juno	Aditya-L1 (ISRO)
Pioneer 10/11/12 [L]	Galileo	MAVEN	
Haley armada [L]	Genesis	SMAP (Earth Science)	<i>Examples of Users not Requesting NAIF Help</i>
Phobos 2 [L] (RSA)	Deep Impact	OSIRIS REx	Hera (ESA)
Ulysses [L]	Huygens Probe (ESA) [L]	InSight	ExoMars RSP (ESA, RSA)
Voyagers [L]	Stardust/NExT	Mars 2020	Emmirates Mars Mission (UAE via LASP)
Lunar Orbiter [L]	Mars Global Surveyor	Europa Clipper	Hayabusa-2 (JAXA)
Helios 1,2 [L]	Phoenix	NISAR (NASA and ISRO)	Proba-3 (ESA)
	EPOXI	Psyche	Parker Solar Probe
	GRAIL	Lucy	EUMETSAT GEO satellites [L]
	DAWN	Lunar Reconnaissance Orbiter	MOM (ISRO)
	Messenger	Mars Express (ESA)	Chandrayan-2 (ISRO)
	Phobos Sample Return (RSA)	ExoMars 2016 (ESA, RSA)	Solar Orbiter (ESA)
	Venus Express (ESA)	Akatsuki (JAXA)	STEREO [L]
	Rosetta (ESA)	Korean Pathfinder Lunar Orbiter (KARI)	Spitzer Space Telescope [L]
[L] = limited use	Chandrayaan-1 (ISRO)	New Horizons	Kepler [L]
[S] = special services	Hayabusa (JAXA)	JUICE (ESA)	Hubble Space Telescope [S][L]
	Kaguya (JAXA)	Bepicolombo (ESA, JAXA)	James Webb Space Telescope [S][L]
	LADEE		Altius (Belgian earth science satellite)
	ISO [S] (ESA)		Armadillo (CubeSat, by UT at Austin)
Last updated: 11/19/19	Smart-1 (ESA)	Deep Space Network	

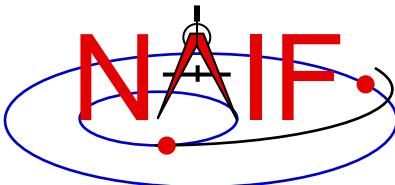
█ NAIF has or had project-supplied funding to support mission operations, consultation for flight team members, and SPICE data archive preparation. NAIF also has PDS funding to help scientists use SPICE data that have been officially archived at the NAIF Node of the PDS.

█ NAIF has NASA funding to support a foreign partner in SPICE deployment and archive review, and to consult with flight team SPICE users.

█ NAIF has token funding to consult with kernel producers at APL. APL provides support to science teams.

█ NAIF has or had modest PDS-supplied funding to consult on assembly of a SPICE archive.

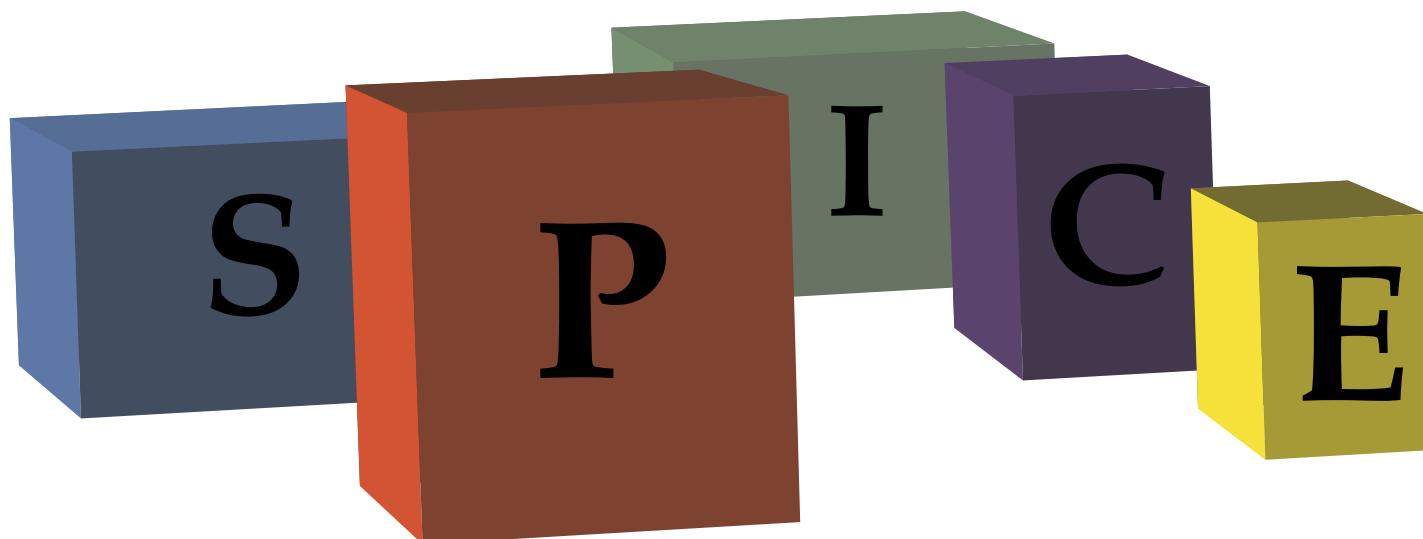
█ NAIF has PDS funding to help NASA funded scientists use SPICE data archived at the NAIF Node of the PDS.

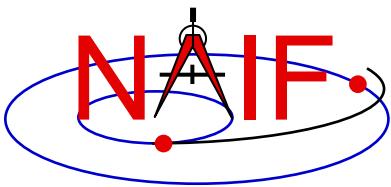


# Building Blocks for Your Applications

Navigation and Ancillary Information Facility

The “SPICE” observation geometry system can serve as a set of building blocks for constructing tools supporting multi-mission, international space exploration programs.



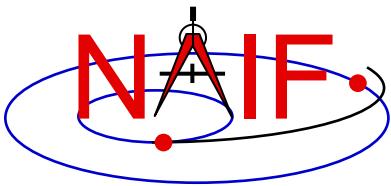


---

Navigation and Ancillary Information Facility

# Fundamental Concepts

January 2020

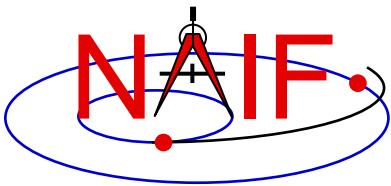


# Topics

---

Navigation and Ancillary Information Facility

- Preface
- Time
- Reference Frames
- Coordinate Systems
- Positions and States
- Aberration Corrections

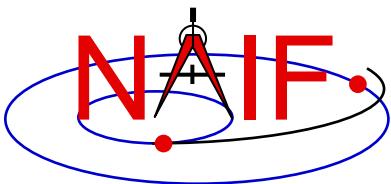


# Preface

---

Navigation and Ancillary Information Facility

- **This tutorial introduces terminology and concepts used in the later SPICE tutorials.**
- **Some of this material is more difficult than what follows in later presentations.**
  - A complete understanding of this material is *not* essential in order to use SPICE.
- **Still, we think this information may be helpful, so... on we go!**

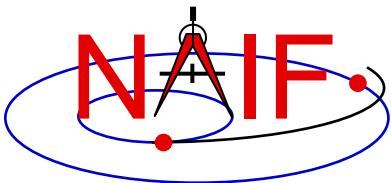


# Time

---

Navigation and Ancillary Information Facility

- **An epoch is an instant in time specified by some singular event**
  - Passage of a star across your zenith meridian
  - Eclipse of a spacecraft signal as it passes behind a solid body
- **Clocks**
  - Clocks count epochs specified by events such as: “regular” oscillations of a pendulum, quartz crystal, or electromagnetic radiation from a specified source, measured from an agreed upon reference epoch.
  - Careful specification of epochs using clocks requires reference to the particular clock and the location of that clock.
- **Time Systems**
  - Are agreed upon standards for “naming” epochs, measuring time, and synchronizing clocks

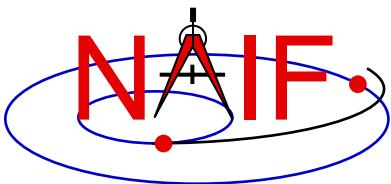


# Atomic Time and UTC

---

Navigation and Ancillary Information Facility

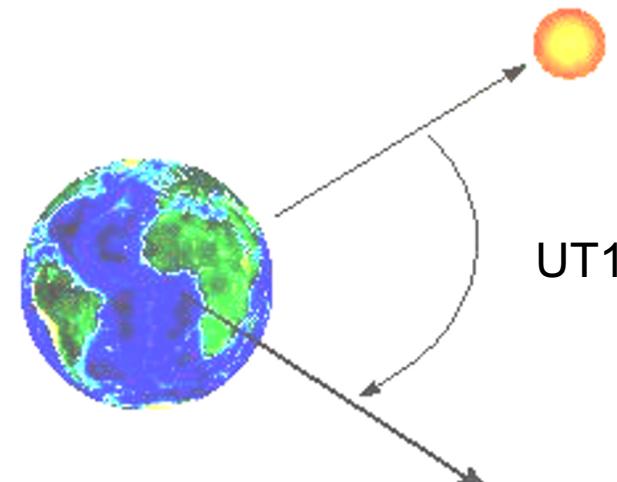
- **International Atomic Time (TAI)**
  - Statistical time scale
    - » Based on data from ~200 atomic clocks in over 50 national laboratories
  - Maintained by Bureau International des Poids et Mesures (BIPM)
  - Unit is the SI (System International) second
    - » duration of 9192631770 periods of the radiation corresponding to the transition between two hyperfine levels of the ground state of the cesium 133 atom
  - TAI is expressed as a count of atomic seconds past the astronomically determined instant of midnight 1 Jan 1958 00:00:00
- **Coordinated Universal Time (UTC)**
  - Civil Time at Greenwich England (~GMT)
  - Usual Calendar Formats plus Hour:Minute:Second.fraction
- **Relationship between TAI and UTC**
  - $\text{UTC} + 10 \text{ seconds} + \text{number of leap seconds} = \text{TAI}$ 
    - » Valid only after Jan 01, 1972

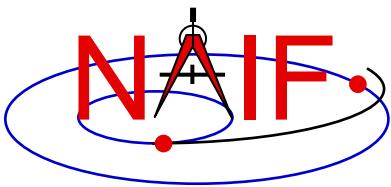


# Astronomical Time – UT1

Navigation and Ancillary Information Facility

- **Astronomical Time (UT1)** is an hour representation of the angle between the Greenwich zenith meridian and the location of the “computed mean sun.”
- Used prior to atomic time for civil time keeping

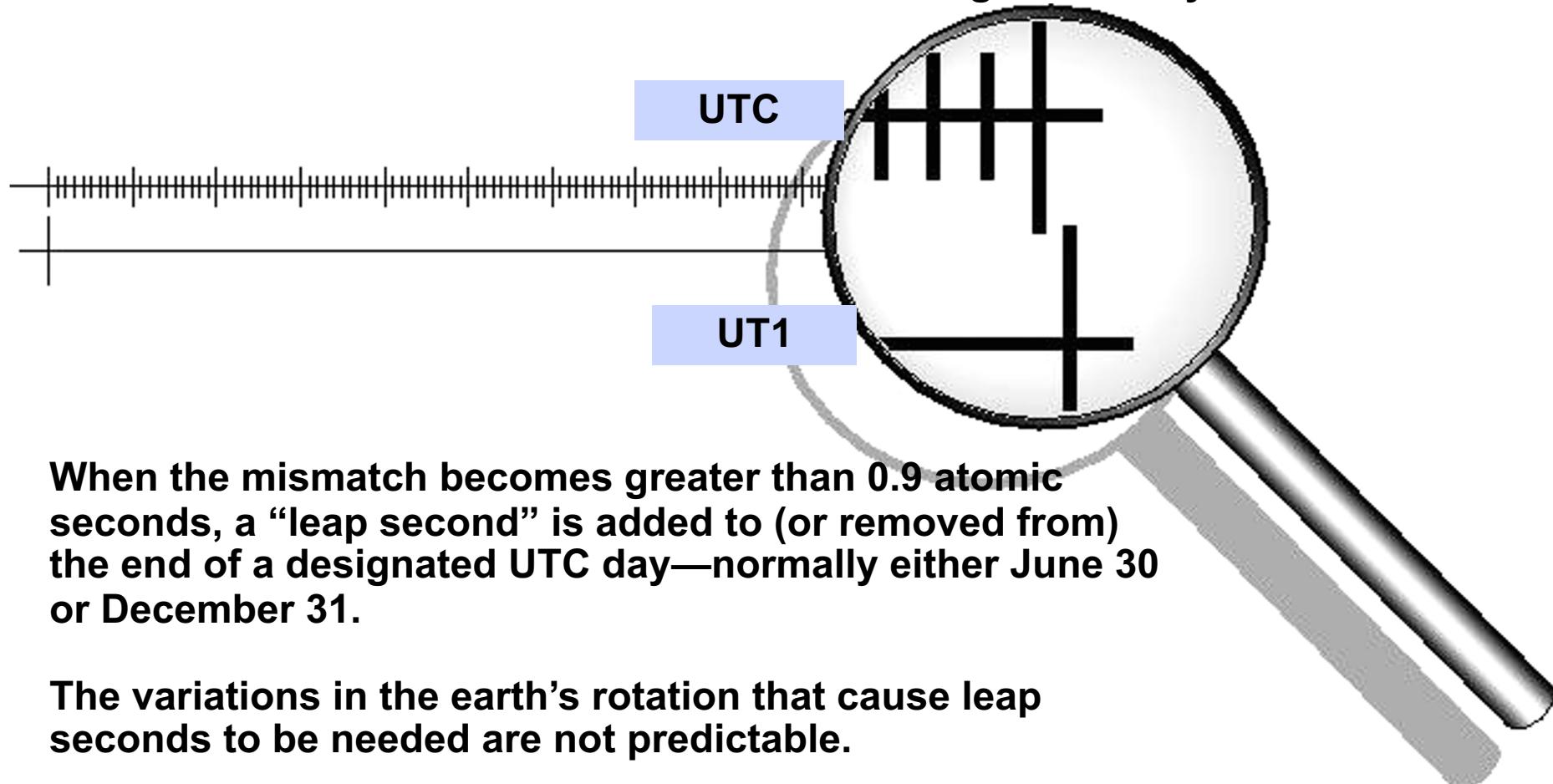


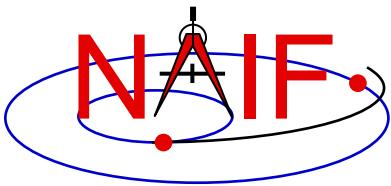


# Tying UTC to Earth's Rotation (UT1)

Navigation and Ancillary Information Facility

Ideally, UTC noon and astronomical noon at Greenwich (UT1) should occur simultaneously. However, the earth's rotation is not uniform. Eventually, UTC noon and astronomical noon at Greenwich get out of sync.





# Leapseconds (+ and -)

Navigation and Ancillary Information Facility

- “Normal” sequence of UTC time tags

- 1998 Dec 31 23:59:58.0
- 1998 Dec 31 23:59:59.0
- 1999 Jan 01 00:00:00.0
- 1999 Jan 01 00:00:01.0

Leap seconds complicate the task of finding the duration between two UTC epochs:

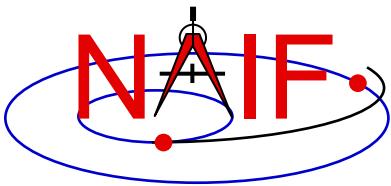
- You need to know when past leap seconds occurred to compute durations defined by pairs of past UTC epochs.
- Durations defined by pairs of future UTC epochs are indeterminate if leap seconds could occur in the interim.

- Sequence with a Positive Leapsecond

- 1998 Dec 31 23:59:58.0
- 1998 Dec 31 23:59:59.0
- 1998 Dec 31 23:59:60.0
- 1999 Jan 01 00:00:00.0
- 1999 Jan 01 00:00:01.0

- Sequence with a Negative Leapsecond

- 1998 Dec 31 23:59:57.0
- 1998 Dec 31 23:59:58.0
- 1999 Jan 01 00:00:00.0
- 1999 Jan 01 00:00:01.0

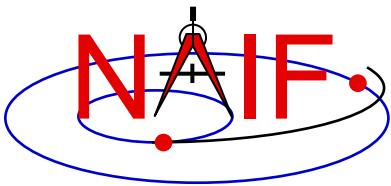


# Barycentric Dynamical Time

---

Navigation and Ancillary Information Facility

- **Barycentric Dynamical Time (TDB) and Ephemeris Time (ET) are synonyms in SPICE documentation.**
- **TDB is**
  - a mathematical ideal used in the equations of motion.
  - used as the independent time variable for many SPICE subroutine interfaces.
  - related to Barycentric Coordinate Time (TCB) by an offset and a scale factor.
  - TDB advances on average at very close to the same rate as TAI---the difference is nearly periodic.

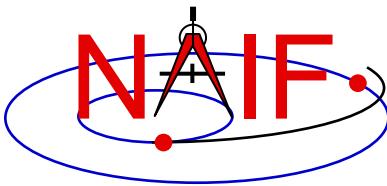


# Terrestrial Dynamical Time

---

Navigation and Ancillary Information Facility

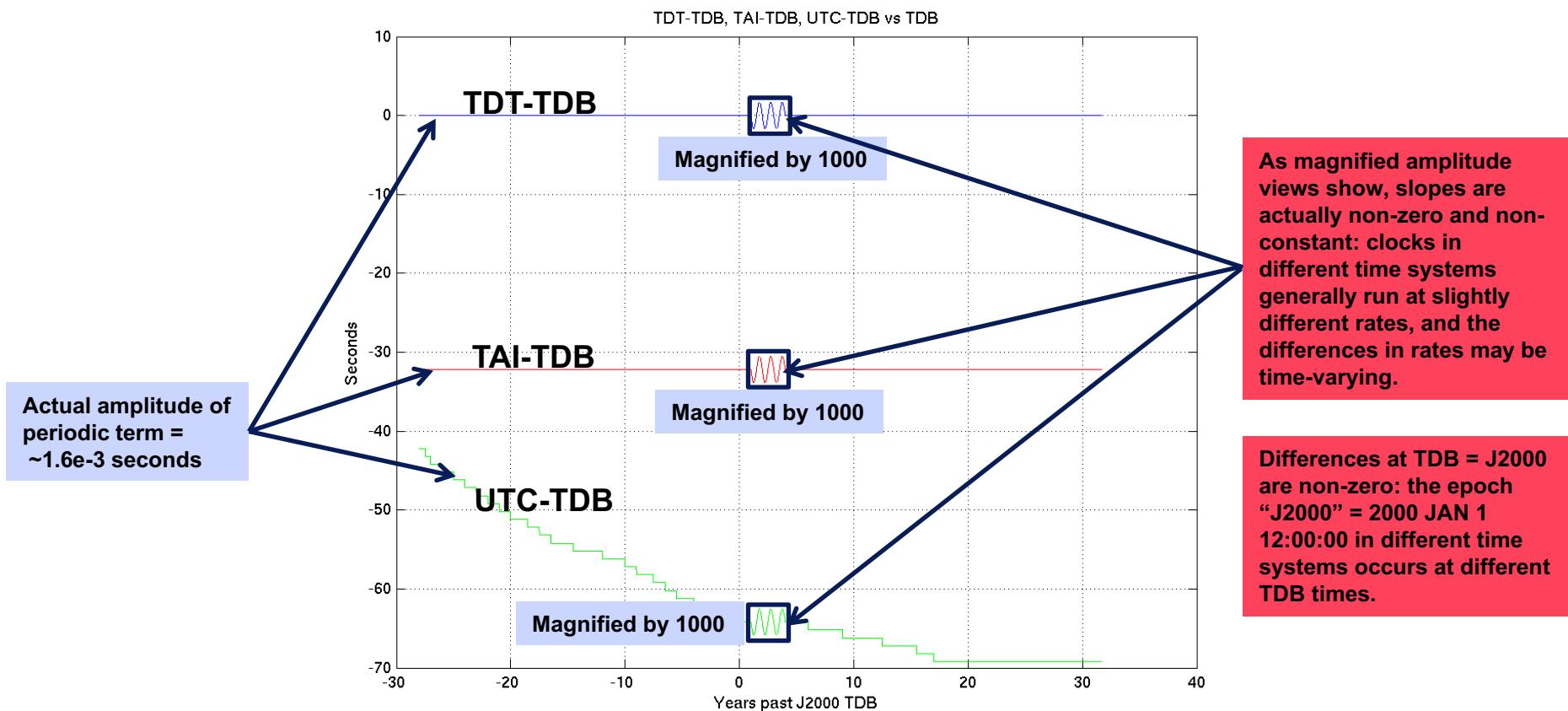
- **Terrestrial Dynamical Time (TDT)**
  - TDT is the Ideal Time (proper time) on Earth at sea level
  - $TDT = TAI + 32.184$  seconds
  - The IAU has adopted the name “Terrestrial Time” (TT)
    - » But this is called TDT throughout SPICE documentation
- **TDB and TDT have nearly the same reference epoch (approximately 1 Jan 2000, 12:00:00 at Greenwich England), called “J2000.”**
- **TDB and TDT advance at different rates.**
  - Variations are small: ~ 1.6 milliseconds
  - Variations are almost periodic with a period of 1 sidereal year (to first order)
  - Variations are due to relativistic effects
    - »  $TDB = TDT + 0.001657 \sin(E + 0.01671\sin(E))$
- **Use of TDT in the SPICE system is quite limited.**
  - SCLK kernels
  - Duration computations involving UTC

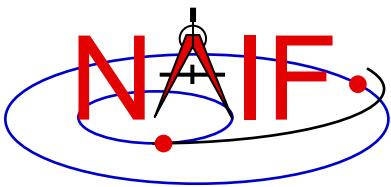


# Offsets between Time Systems

Navigation and Ancillary Information Facility

Difference between seconds past J2000 in a given time system and TDB seconds past J2000 TDB. Systems used for comparison are TDT, TAI, and UTC:



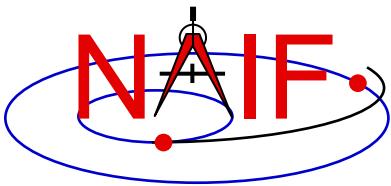


# Spacecraft Clocks

---

Navigation and Ancillary Information Facility

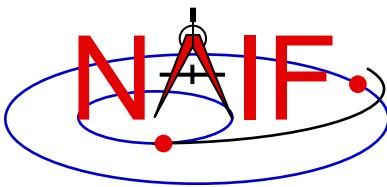
- **Spacecraft have onboard clocks to control scheduling of observations, maneuvers, attitude adjustments, etc.**
- **Used to time stamp data**
- **Fundamental unit of time is the “tick”**
  - Smallest increment possible for a spacecraft clock
  - Nominal tick duration is spacecraft clock dependent
- **Spacecraft clock time is a count of ticks since some reference tick.**
- **The duration of the tick drifts with respect to other time systems because spacecraft clocks are not very stable**



# More about Spacecraft Clocks

Navigation and Ancillary Information Facility

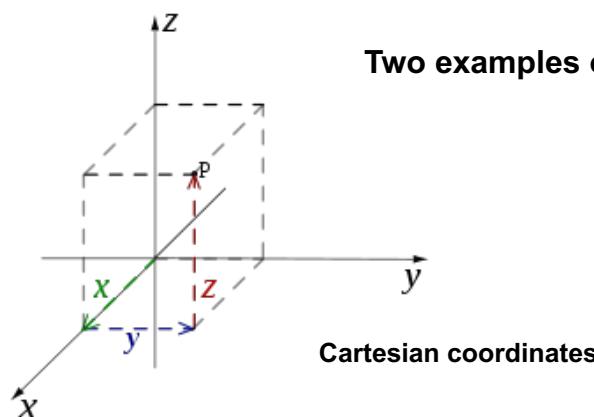
- **SCLK string formats vary from one spacecraft clock to the next.**
  - Cassini: Maximum reading for partition 1 = 1/4294967295.255
    - » Partition number: 1
    - » Seconds: 4294967295
    - » Ticks (for Cassini, unit = 1/256 second): 255
  - Galileo: Maximum reading for partition 1 = 1/16777215:90:09:07
    - » Partition number: 1
    - » "RIM" count (unit = 60 2/3 seconds): 16777215
    - » "Mod 91" count (unit = 2/3 second): 90
    - » "RTI" count (unit = 1/15 second): 9
    - » "Mod 8" count (unit = 1/120 second): 7
- The format of spacecraft clock and the relationship between tick count and other time systems (usually UTC) is captured in a SPICE SCLK kernel



# SPICE Definitions: Reference Frames & Coordinate Systems

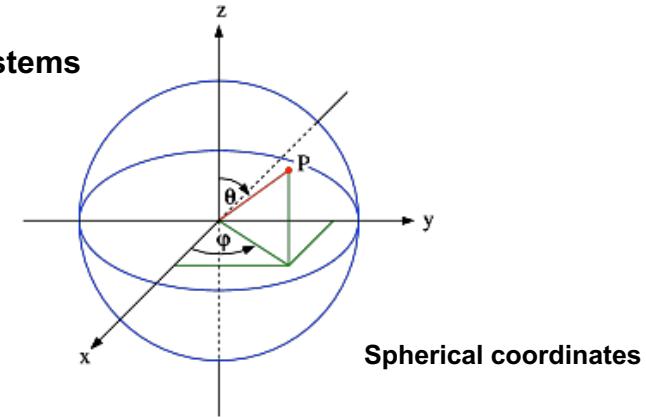
Navigation and Ancillary Information Facility

- 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” instead of “reference frame.”
  - The ordered set of axes of a reference frame is also called a “basis.”
- A **coordinate system** specifies the method of locating a point within a reference frame.

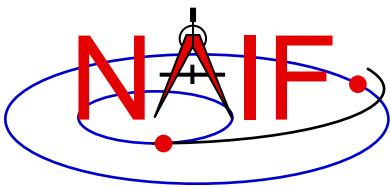


Two examples of coordinate systems

Cartesian coordinates



Spherical coordinates

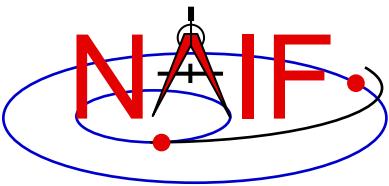


# Reference Frame Center

---

Navigation and Ancillary Information Facility

- **A reference frame's center is an ephemeris object whose location is coincident with the origin (0, 0, 0) of the frame.**
  - The center of the IAU\_<body> frame is center of mass of <body>.
  - The center of any inertial frame is (in SPICE) the solar system barycenter.
    - » True 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.
    - » (Light time and aberration corrections are discussed later on.)

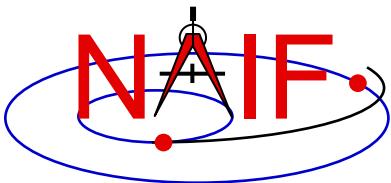


# Types of Reference Frames

---

Navigation and Ancillary Information Facility

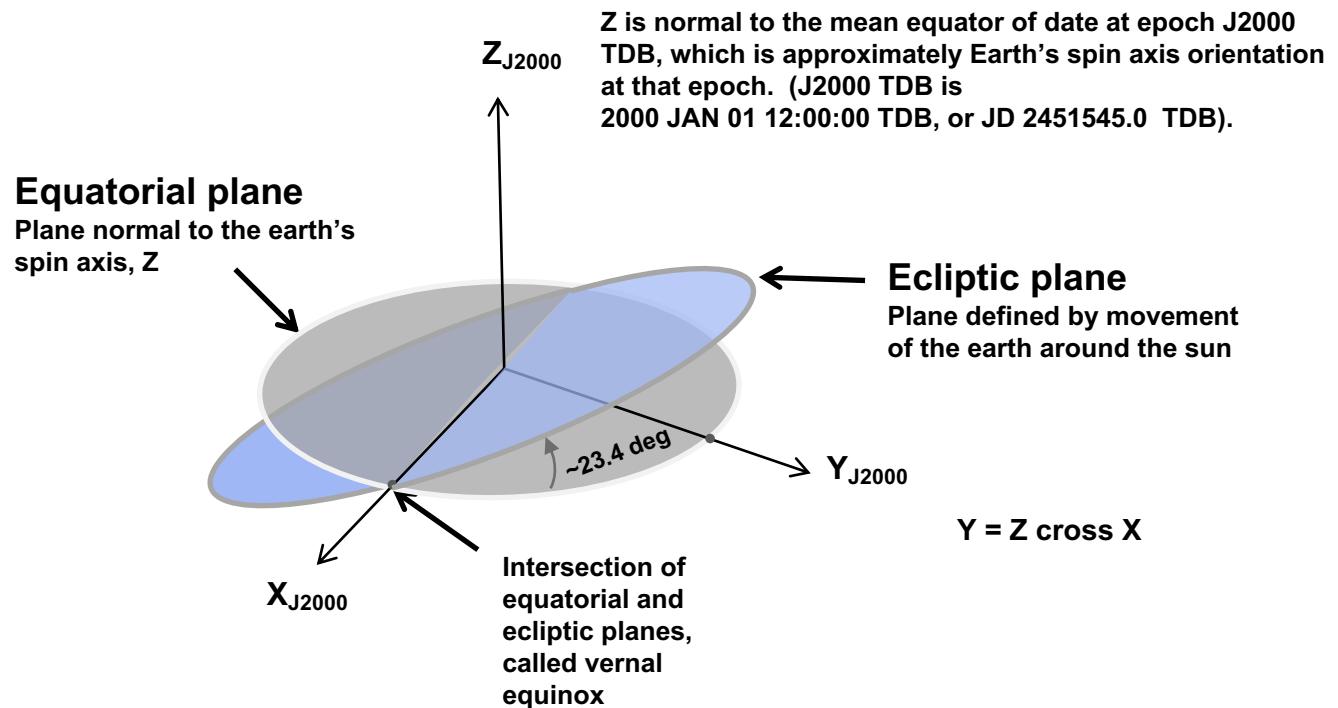
- **Inertial**
  - Non-rotating
    - » With respect to fixed stars
  - Non-accelerating origin
    - » Velocity is typically non-zero; acceleration is negligible
  - Examples:
    - » J2000 (also called ICRF), B1950
- **Non-Inertial**
  - Examples
    - » Body-fixed
      - Centered at body center
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



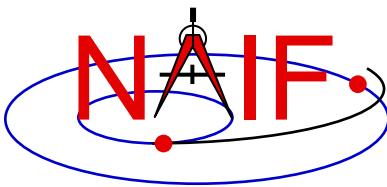
# J2000 Frame

Navigation and Ancillary Information Facility

- The J2000\* (aka EME2000) frame definition is based on the earth's equator and equinox, determined from observations of planetary motions, plus other data.



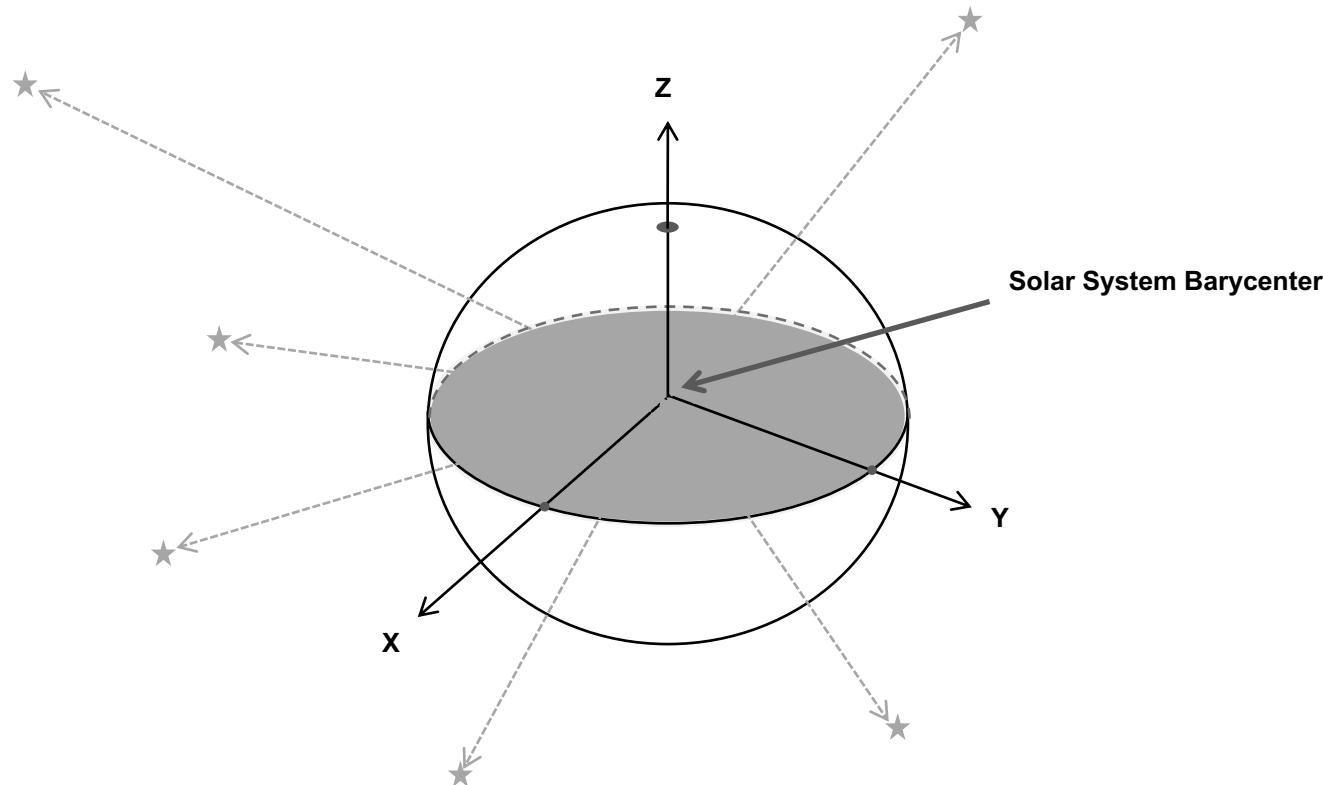
\*Caution: The name "J2000" is also used to refer to the zero epoch of the ephemeris time system (ET, also known as TDB).



# The ICRF Frame

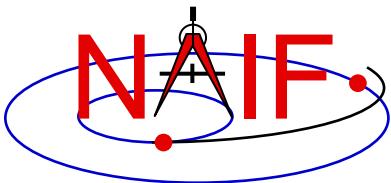
Navigation and Ancillary Information Facility

- The ICRF\* frame is defined by the adopted locations of 295 extragalactic radio sources



\*ICRF = International Celestial Reference Frame

The ICRF is managed by the International Earth Rotation Service (IERS)



# J2000 vs. ICRF

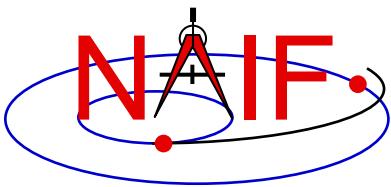
---

Navigation and Ancillary Information Facility

- **The realization of ICRF was made to coincide almost exactly with the J2000 frame.**
  - The difference is very small—a rotation of less than 0.1 arc second
  - These two frame names are treated synonymously in SPICE
    - » In reality, any SPICE data said to be referenced to the J2000 frame are actually referenced to the ICRF frame
    - » For historical and backwards compatibility reasons, only the name “J2000” is recognized by SPICE software as a frame name—not “ICRF”
- **No transformation is required to convert SPICE state vectors or orientation data from the J2000 frame to the ICRF.**

ICRF = International Celestial Reference Frame, defined by the IERS

IERS = International Earth Rotation Service

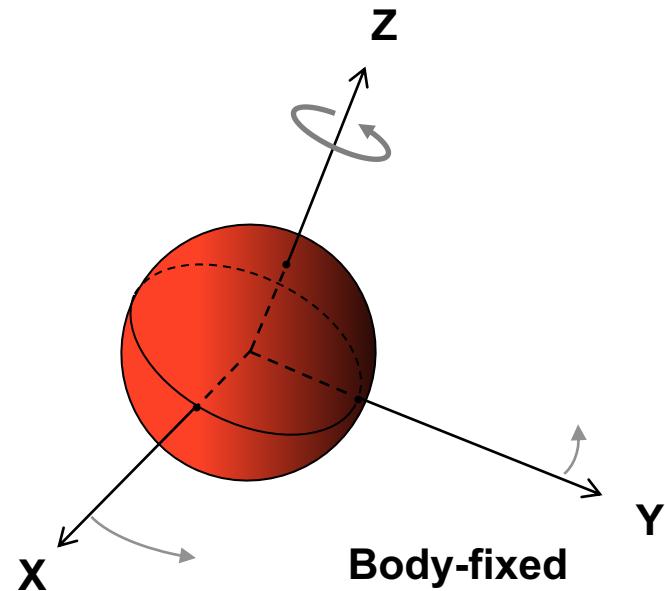


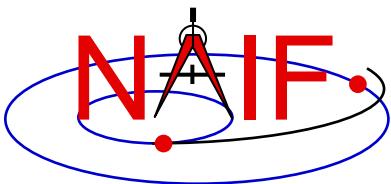
# Body-fixed Frames

Navigation and Ancillary Information Facility

- **Body-fixed frames are tied to a named body and rotate with it**

- The most common names, those for planets, satellites, the sun, and a few asteroids and comets, are hard-coded in SPICE software
  - » Frame name style is “IAU\_body name”
    - Examples: IAU\_MARS, IAU\_SATURN
  - » To see all such names, see:
    - Frames Required Reading document, or
    - Latest generic PCK file
- The rotation state (the orientation at time  $T$ ) is usually determined using a SPICE text PCK containing data published by the IAU
- The earth and moon are special cases!
  - » IAU\_EARTH and IAU\_MOON both exist but generally should NOT be used
  - » See the SPICE tutorial named “lunar-earth\_pck-fk” for the best frames to be used for those bodies
- On very rare occasions a CK is used to provide a body’s rotation state

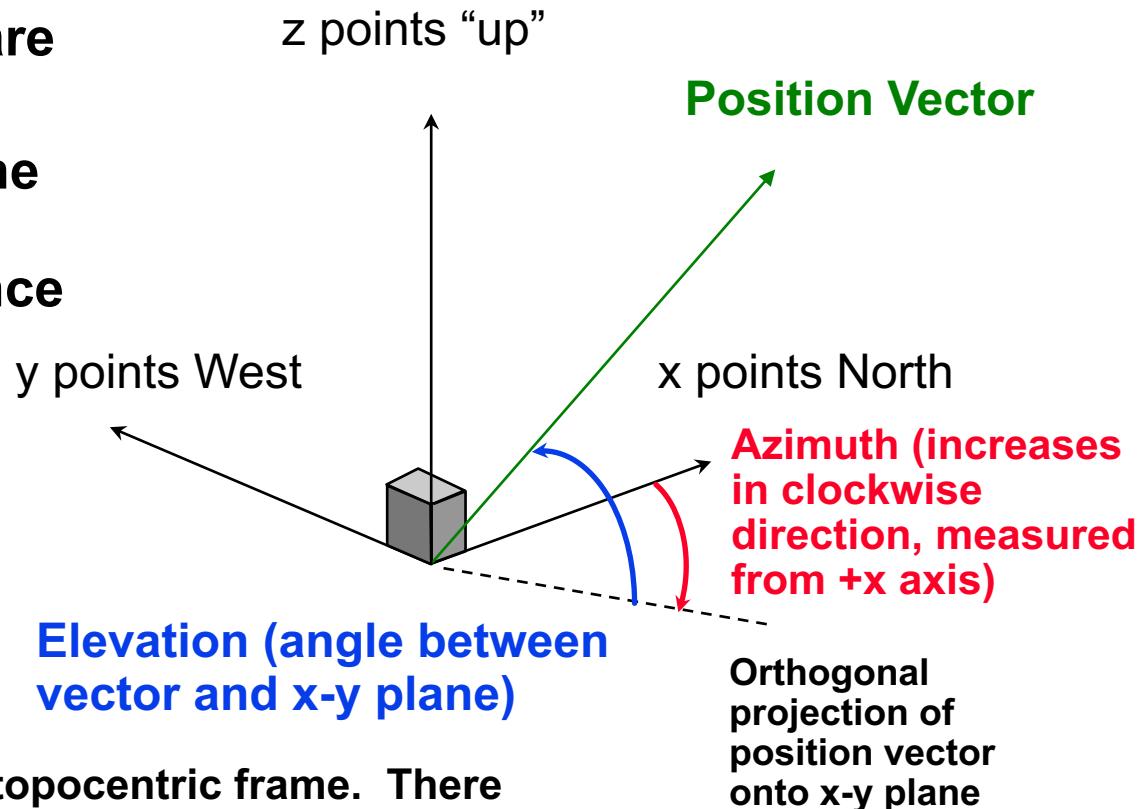




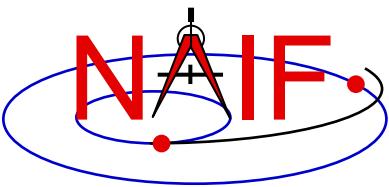
# Topocentric Frames

Navigation and Ancillary Information Facility

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



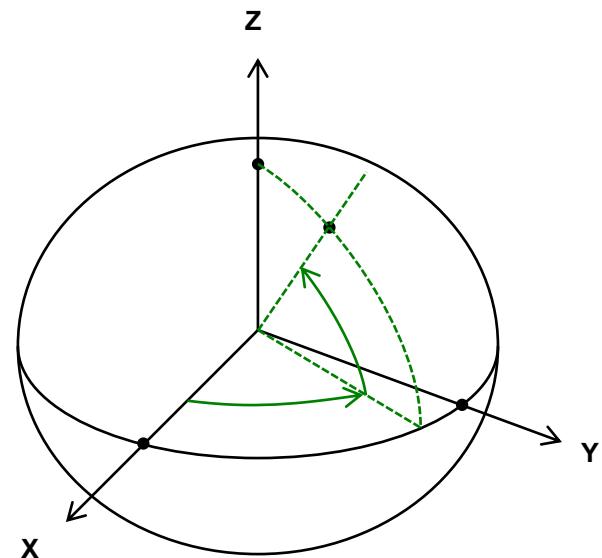
One example of a topocentric frame. There are other types of topocentric frames: for example, the z-axis could point down, the x-axis North, and the y-axis East.



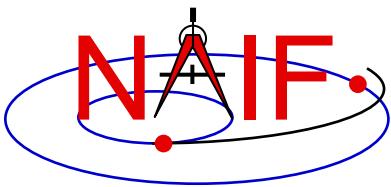
# Coordinate System Conventions - 1

Navigation and Ancillary Information Facility

- **Planetocentric coordinate systems (also known as latitudinal)**
  - For planets and their satellites the +Z axis (+90 LAT) always points to the north side of the invariable plane (the plane whose normal vector is the angular momentum vector of the solar system)
    - » Planetocentric longitude increases positively eastward
    - » Planetocentric latitude increases positively northward
  - Dwarf planets\*, asteroids and comets spin in the right hand sense about their “positive pole.”
    - » What the IAU now calls the “positive pole” is still referred to as the “north pole” in SPICE documentation.
    - » The “positive pole” may point above or below the invariable plane of the solar system (see above).
    - » This revision by the IAU Working Group (2006) inverts what had been the direction of the north pole for Pluto, Charon and Ida.
  - Planetocentric routines are RECLAT/ LATREC, RADREC/RECRAD, DRDLAT/DLATDR



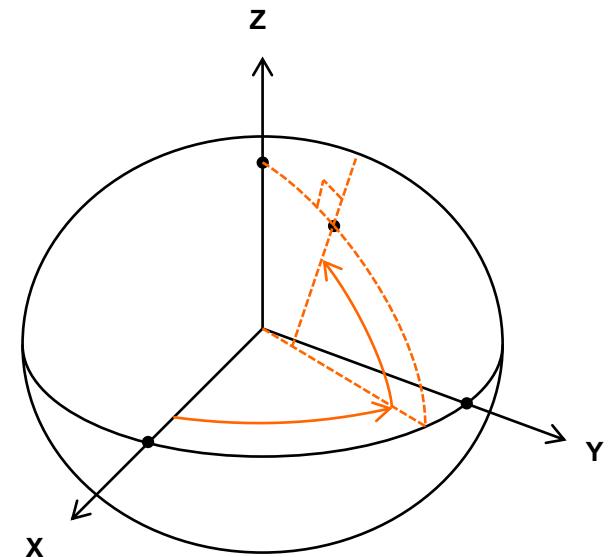
\*The dwarf planets are: Ceres, Eris, Haumea, Makemake, Pluto

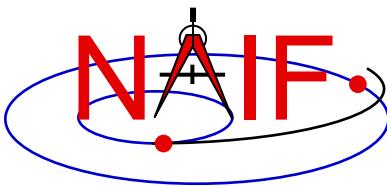


# Coordinate System Conventions - 2

Navigation and Ancillary Information Facility

- **Planetodetic coordinate systems**
  - Planetodetic longitude is planetocentric longitude
    - » increases positively eastward
  - Planetodetic latitude
    - » Tied to a reference ellipsoid
    - » For a point on a reference ellipsoid, the angle measured from the 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
    - » Latitude increases positively in the +Z direction
      - +Z is defined the same as for planetocentric coordinates
  - Planetodetic routines are REC GEO/GEO REC,  
DRD GEO/DGEO DR





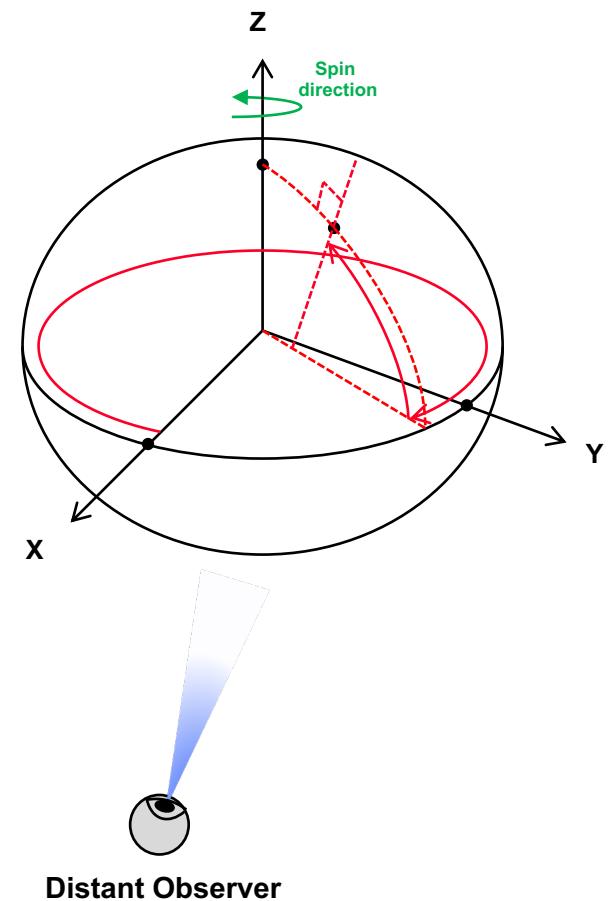
# Coordinate System Conventions - 3

Navigation and Ancillary Information Facility

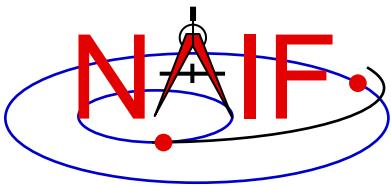
- **Planetographic coordinate systems**
  - For planet and satellite planetographic coordinate systems:
    - » Planetographic latitude is planetodetic latitude
    - » Planetographic longitude is usually defined such that the sub-observer longitude increases with time as seen by a distant, fixed observer in an inertial reference frame
      - The earth, moon and sun are exceptions; planetographic longitude is positive east by default
      - Planetographic routines are PGRREC/RECPGR

- For dwarf planets, asteroids and comets:

- » There are multiple, inconsistent standards! (USNO, IAU, PDS)
- » NAIF strongly suggests you use only planetocentric or planetodetic coordinates
  - Planetocentric routines are RECLAT/ LATREC, RADREC/RECRAD, DRDLAT/DLATDR
  - Planetodetic routines are REC GEO/GEOREC, DRD GEO/DGEODR



Distant Observer

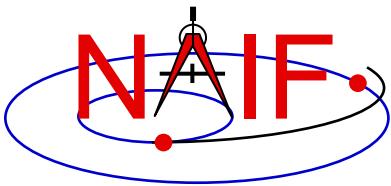


# State Vectors

---

Navigation and Ancillary Information Facility

- **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
  - A state is always given in the TDB time system (also known as ET)
- **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**

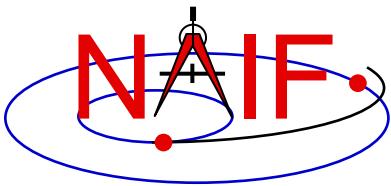


# Transforming States

Navigation and Ancillary Information Facility

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

\* Assuming both frames are right-handed



# Transforming States

Navigation and Ancillary Information Facility

$$\mathbf{S}_B(t) = \mathbf{T}_{A \text{ to } B}(t) \mathbf{S}_A(t)$$

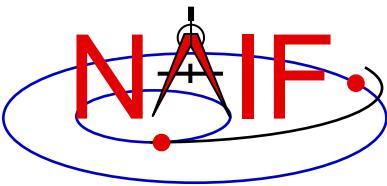
where

$$\mathbf{S}_i(t) = \begin{pmatrix} \mathbf{P}_i(t) \\ d\mathbf{P}_i(t)/dt \end{pmatrix} \quad i = A \text{ or } B$$

and

$$\mathbf{T}_{A \text{ to } B}(t) = \left( \begin{array}{c|c} \mathbf{R}_{A \text{ to } B}(t) & \mathbf{0} \\ \hline d\mathbf{R}_{A \text{ to } B}(t)/dt & \mathbf{R}_{A \text{ to } B}(t) \end{array} \right)$$

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

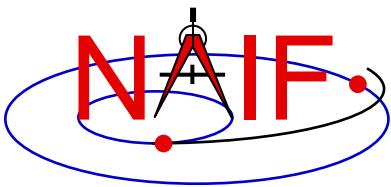


# Aberration Corrections: Introduction

---

Navigation and Ancillary Information Facility

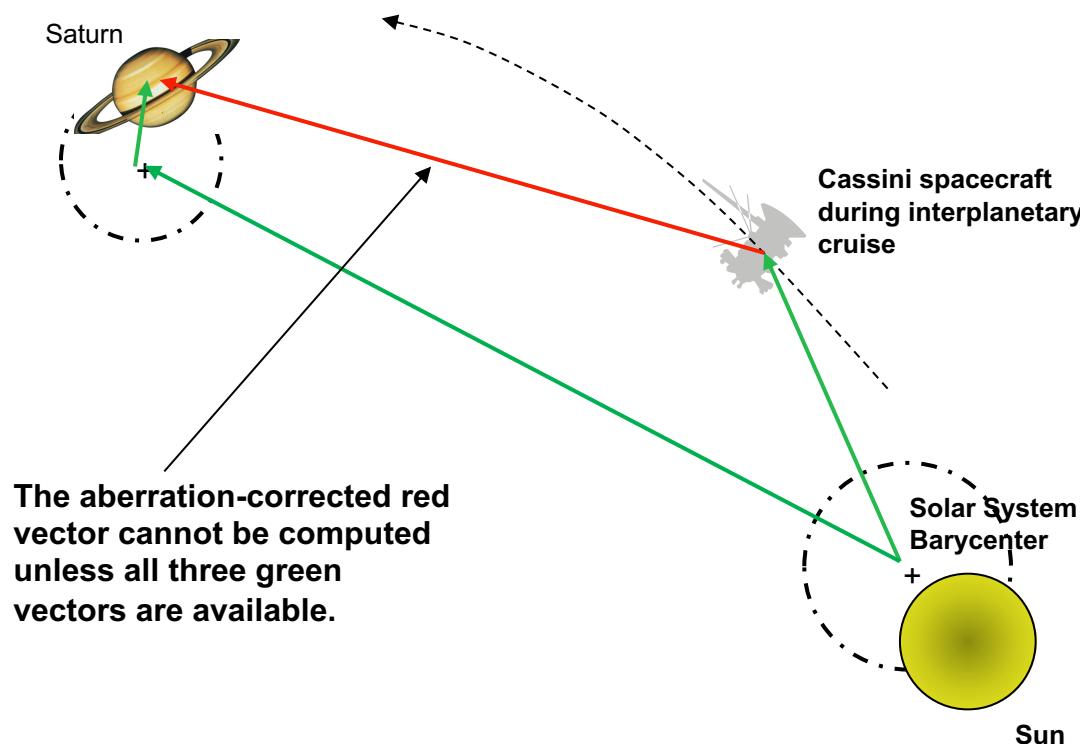
- **Within the SPICE system, “aberration corrections” are adjustments made to state vectors and time-dependent reference frames to accurately reflect the apparent—as opposed to the actual—state and attitude of a target object as seen from a specified observer at a specified time.**
  - Actual, uncorrected states from an ephemeris are called “geometric” states.
  - When computing state vectors, SPICE users may request geometric or aberration-corrected states.
- **Aberration corrections are needed to accurately answer questions such as:**
  - In which direction must a remote sensing instrument be pointed to observe a target of interest?
  - For a given pointing direction and observation time, what target body surface location would be observed by a remote sensing instrument?
  - In which direction must an antenna be pointed to transmit a signal to a specified target?

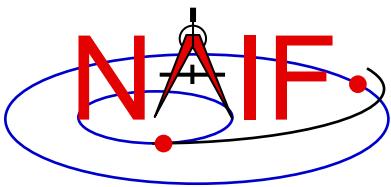


# Computing Aberration-corrected States

Navigation and Ancillary Information Facility

- In order to compute an aberration-corrected state, the state vectors used in the computation must all be chained back to the Solar System Barycenter



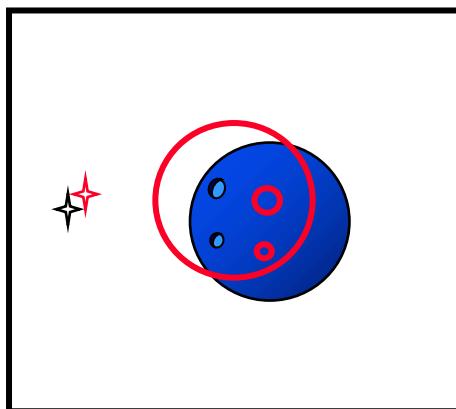


# Example: Predicted vs Actual Photo

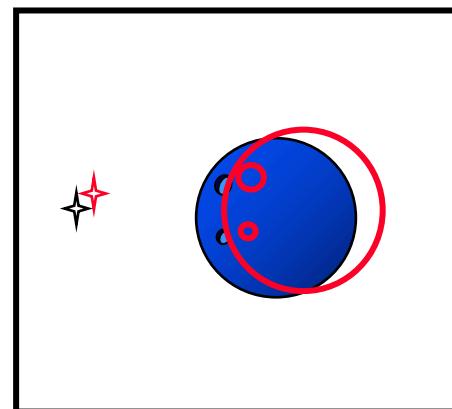
Navigation and Ancillary Information Facility

We compare the predicted appearance of a photograph from an optical camera against the actual photograph. We show three predictions derived using different aberration corrections: NONE, LT ("light time only"), and LT+S ("light time plus stellar aberration").

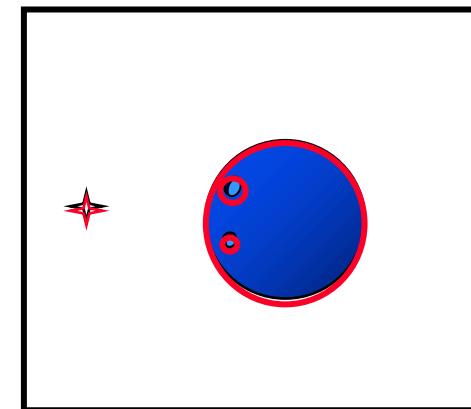
For each prediction, we use red overlays to indicate the expected location in the photo of the images of an extended target body (for example, a natural satellite), of features on the surface of the target body, and of a star.



NONE



LT

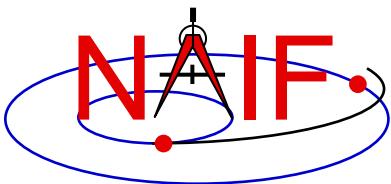


LT+S

Predicted images using uncorrected target position and orientation and uncorrected star direction vector

Predicted images using light time-corrected target position and orientation and uncorrected star direction vector

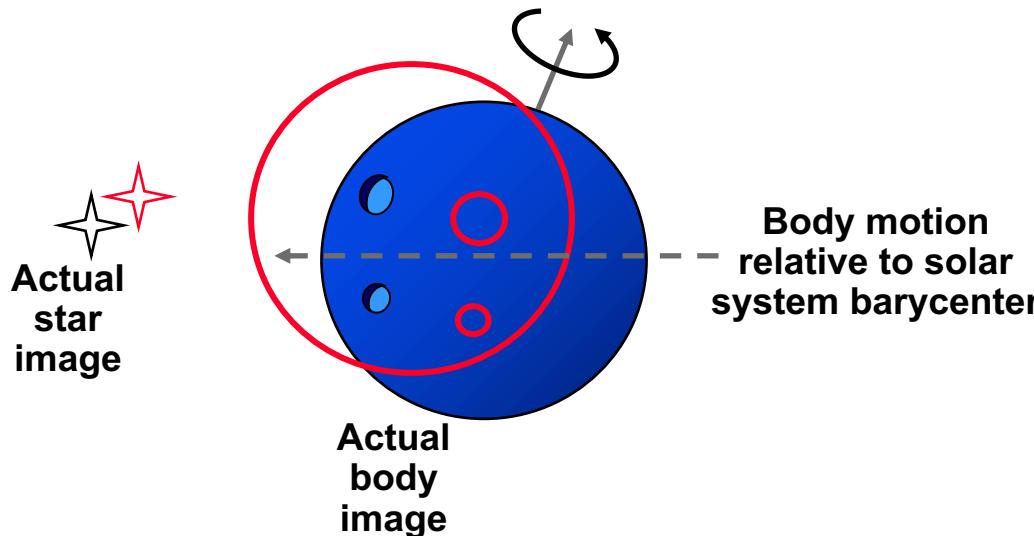
Predicted images using light time and stellar aberration-corrected target position, light time-corrected target orientation, and stellar aberration-corrected star direction vector



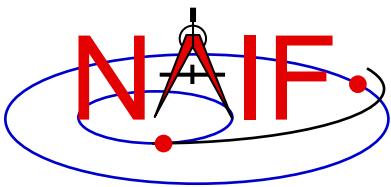
# Prediction Without Corrections

Navigation and Ancillary Information Facility

Predicted target body, surface feature, and star image locations (in red). Actual locations are in blue and black.

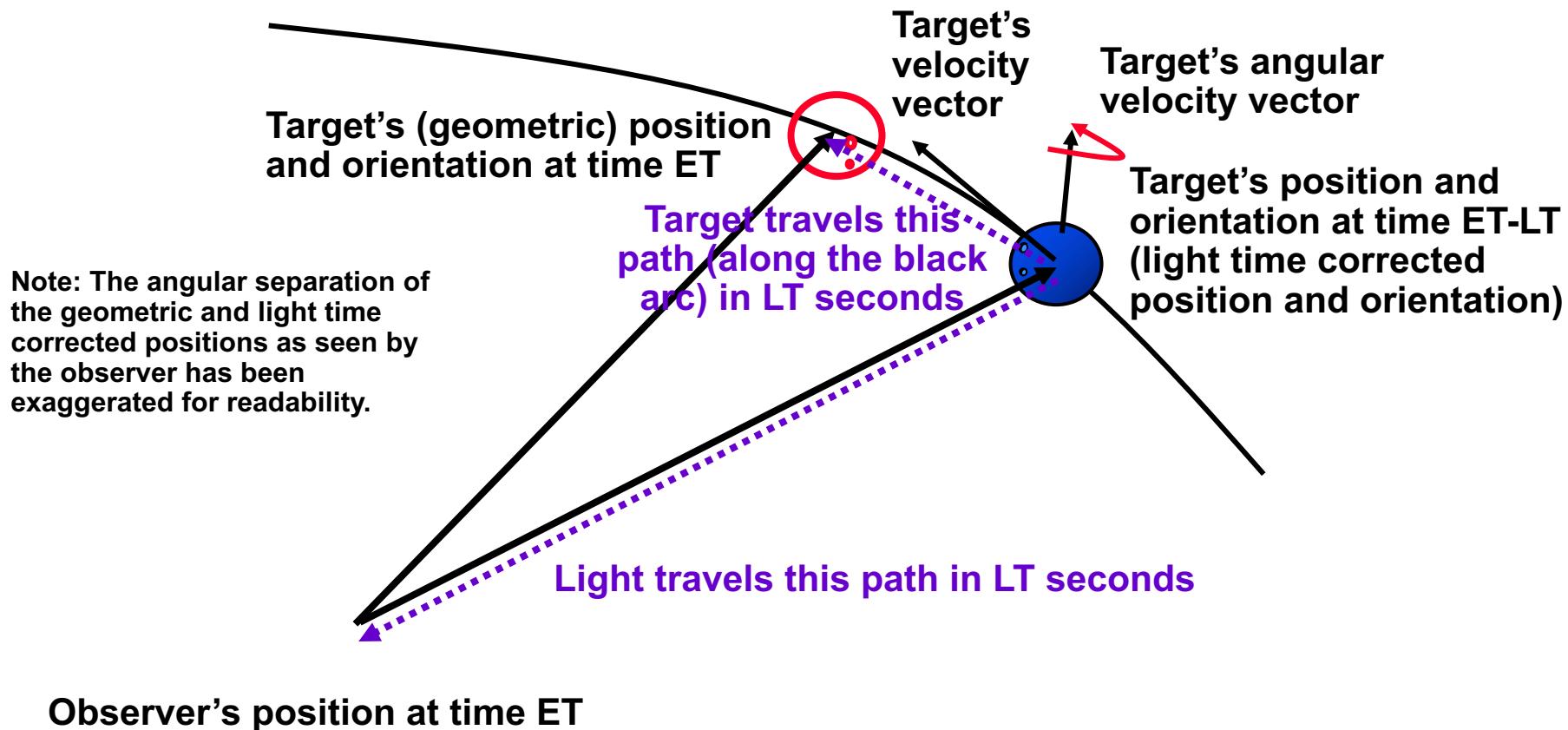


Using geometric target positions, target images in photos or other remote-sensing observations don't appear at their predicted locations.

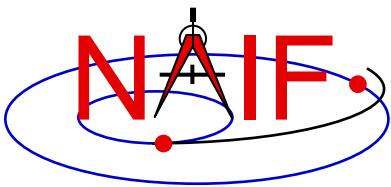


# Light Time Corrections

Navigation and Ancillary Information Facility



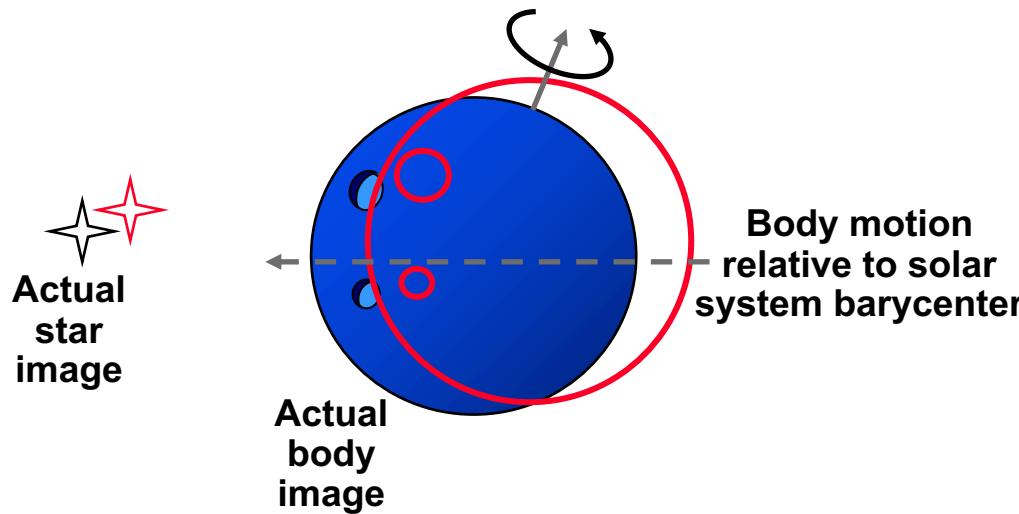
At time ET, the observer's camera records photons emitted from the target at time ET-LT, where LT is the one-way light time. The camera "sees" the target's position and orientation at ET-LT.



# Prediction Using Light Time Corrections

Navigation and Ancillary Information Facility

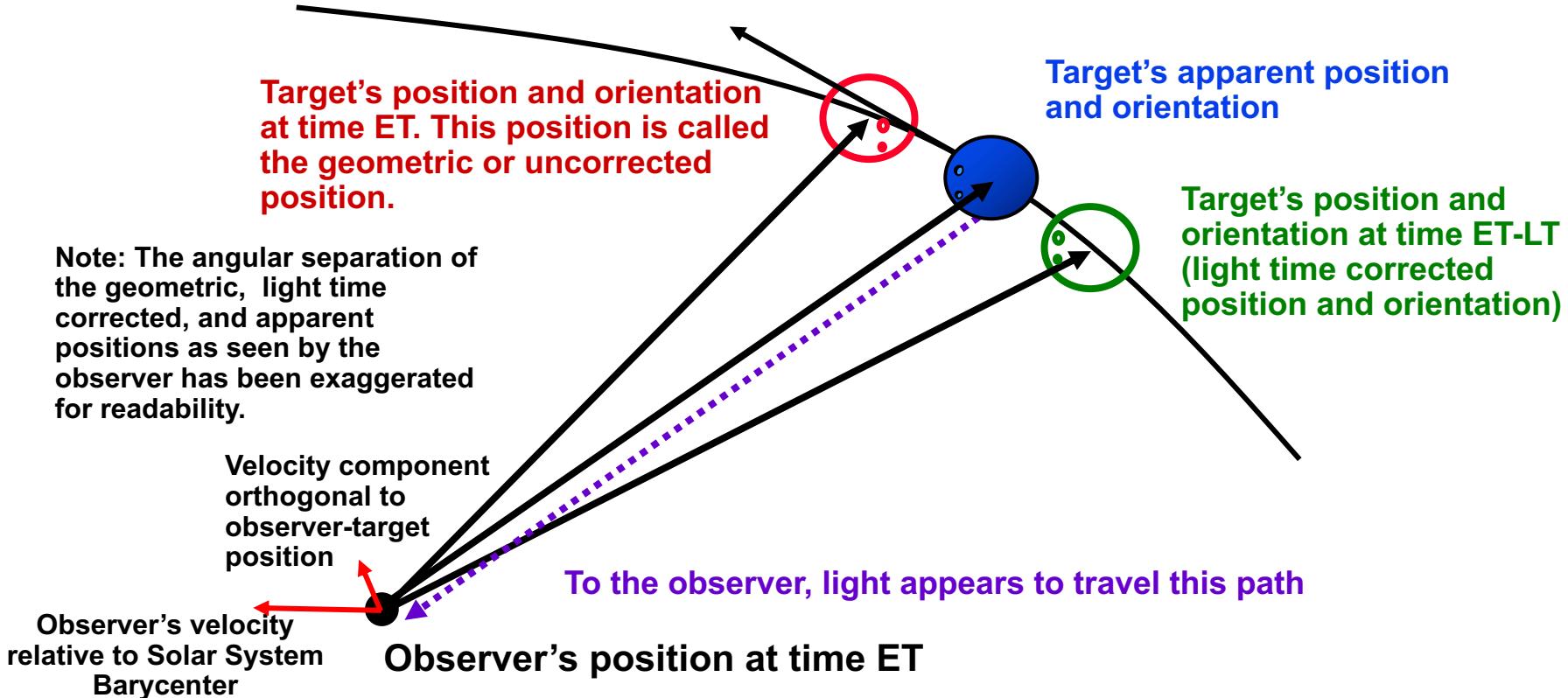
Predicted target body, surface feature, and star image locations (in red). Actual locations are in blue and black.



Using the light time corrected target position and orientation, the predicted locations in the photo of the target image and surface features have changed, but the accuracy of the prediction may still be poor.

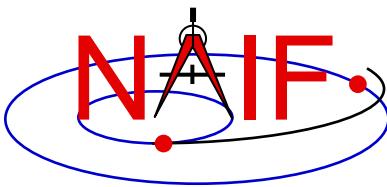
# Stellar Aberration Correction

Navigation and Ancillary Information Facility



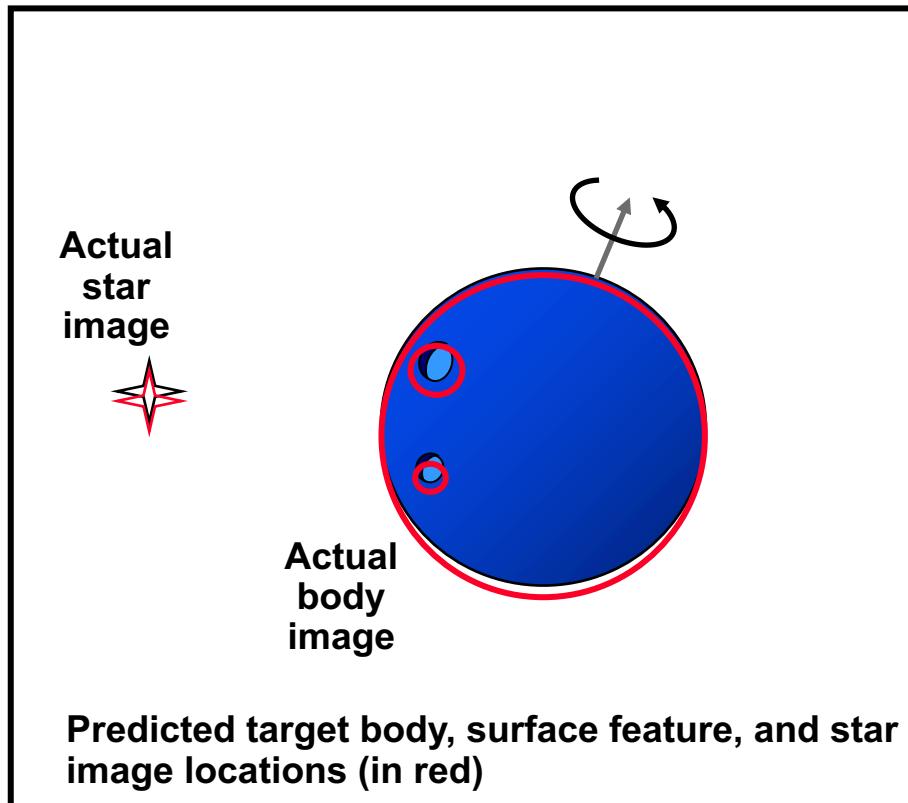
At time ET, the observer's camera records photons emitted from the target at time ET-LT, where LT is the one-way light time.

The vector from the observer at ET to the location of the target at ET-LT is displaced by a physical phenomenon called stellar aberration. The displaced vector yields the apparent position of the target.



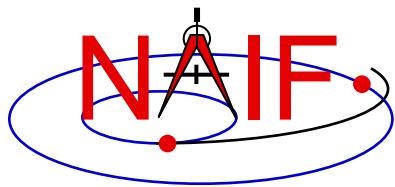
# Prediction Using "LT+S" Corrections

Navigation and Ancillary Information Facility



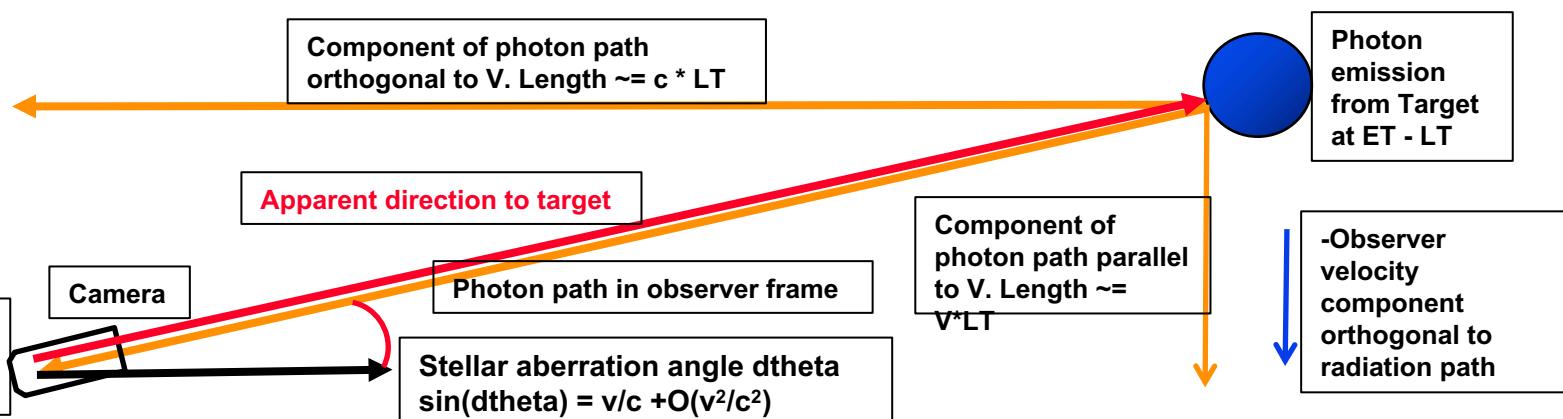
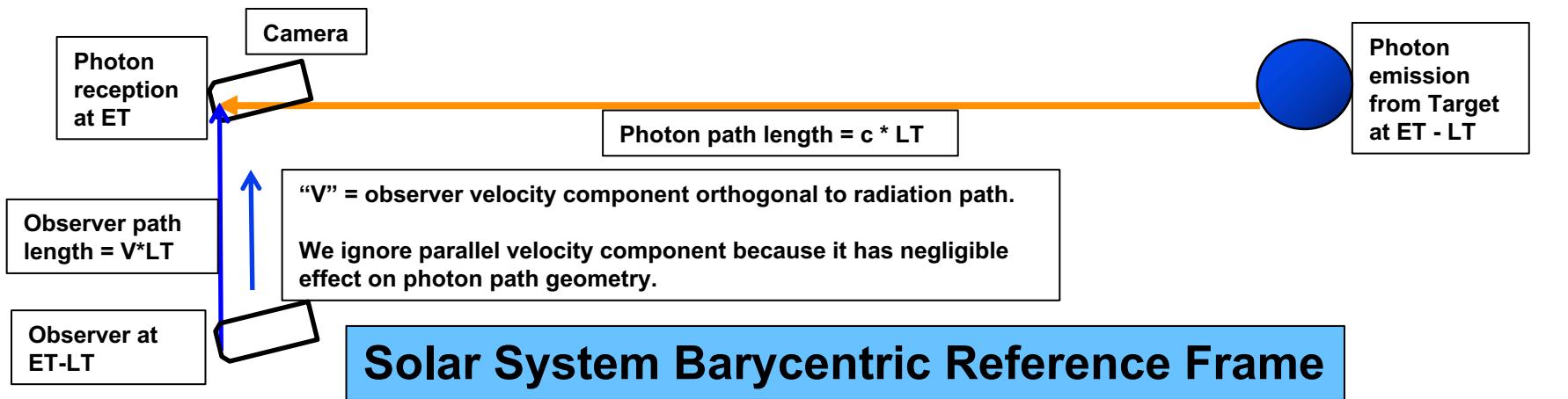
Using the light time and stellar aberration-corrected target position, light time-corrected target orientation, and stellar aberration-corrected star direction, we obtain a significantly improved image location predictions.

Remaining prediction errors may be due to, among other causes, pointing error, spacecraft and target ephemeris errors, and timing errors.

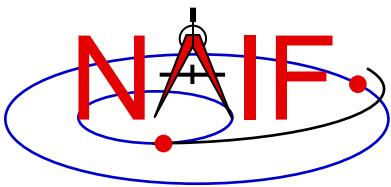


# Newtonian Stellar Aberration Effect

Navigation and Ancillary Information Facility



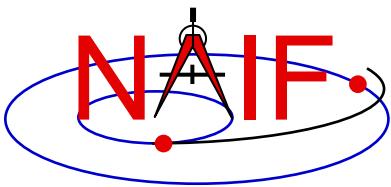
Observer Reference Frame



# Effect of Aberration Corrections - 1

Navigation and Ancillary Information Facility

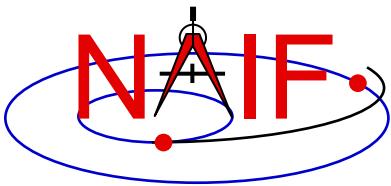
- **Angular offsets between corrected and uncorrected position vectors over the time span 2004 Jan 1--2005 Jan 1**
  - Mars as seen from MEX:
    - » LT+S vs NONE: .0002 to .0008 degrees
    - » LT vs NONE: .0006 to .0047 degrees
  - Earth as seen from MEX:
    - » LT+S vs NONE: .0035 to .0106 degrees
    - » LT vs NONE: .0000 to .0057 degrees
  - MEX as seen from Earth:
    - » LT+S vs NONE: .0035 to .0104 degrees
    - » LT vs NONE: .0033 to .0048 degrees
  - Sun as seen from Mars:
    - » LT+S vs NONE: .0042 to .0047 degrees
    - » LT vs NONE: .0000 to .0000 degrees



# Effect of Aberration Corrections - 2

Navigation and Ancillary Information Facility

- Angular offsets between corrected and uncorrected position vectors over the time span 2004 Jan 1--2008 Jan1
  - Saturn as seen from CASSINI:
    - » LT+S vs NONE: .0000 to .0058 degrees
    - » LT vs NONE: .0001 to .0019 degrees
  - Titan as seen from CASSINI:
    - » LT+S vs NONE: .0000 to .0057 degrees
    - » LT vs NONE: .0000 to .0030 degrees
  - Earth as seen from CASSINI:
    - » LT+S vs NONE: .0000 to .0107 degrees
    - » LT vs NONE: .0000 to .0058 degrees
  - CASSINI as seen from Earth:
    - » LT+S vs NONE: .0000 to .0107 degrees
    - » LT vs NONE: .0000 to .0059 degrees
  - Sun as seen from CASSINI:
    - » LT+S vs NONE: .0000 to .0059 degrees
    - » LT vs NONE: .0000 to .0000 degrees



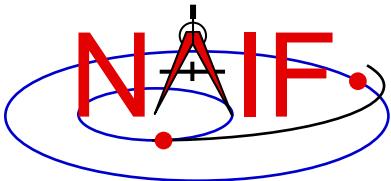
---

Navigation and Ancillary Information Facility

# SPICE Conventions

**A broad summary of standards, terms  
and synanons used within SPICE**

**January 2020**



# SPICE Lexicon - 1

---

Navigation and Ancillary Information Facility

**SPICE**

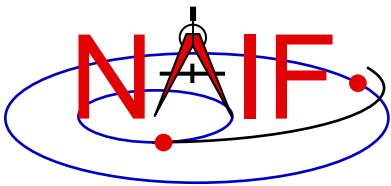
- **The name of this ancillary information system**

**NAIF**

- **The name of the team of people at JPL who lead development of the SPICE system.**
- **Also the name of the ancillary data node of NASA's Planetary Data System (PDS).**

**SPICE-based (code)**

- **A program incorporating some SPICE APIs (a.k.a. subroutines or modules) to compute some geometric quantities.**



# SPICE Lexicon - 2

Navigation and Ancillary Information Facility

## SPICE Toolkit

### The Toolkit

- Names that refer to the principal collection of software produced by JPL's NAIF Team as part of the SPICE information system.

## Toolkit

- The Fortran 77 version of the Toolkit.

## SPICELIB

- The principal user library found within Fortran versions of the Toolkit.

## CSPICE

- Refers to the entire C Toolkit, and also to the principal user library found within C versions of the Toolkit.

## Icy

- An IDL interface to CSPICE

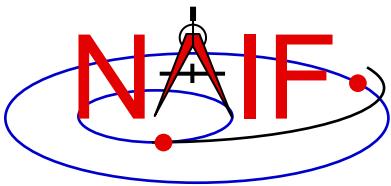
## Mice

- A MATLAB interface to CSPICE

## Kernel

- A SPICE data file

Sorry for this rather confusing terminology!

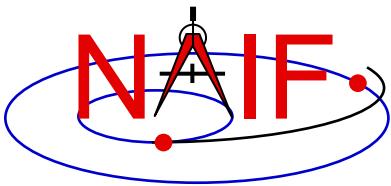


# SPICE Lexicon - 3

---

Navigation and Ancillary Information Facility

- **Text kernel**
  - Any kernel type consisting entirely of ASCII information, with each line terminated using the local operating system convention (CR, LF, or CR+LF)
  - Text kernel types are FK, IK, text Pck, LSK, SCLK, MK
  - Any set of text kernels, excepting MKs, could be combined in a single file.
    - » But this is certainly not recommended!
- **Binary kernel**
  - Any kernel type using a binary file format
  - Binary types are SPK, binary Pck, CK, DBK and DSK
  - Different binary kernel types cannot be combined together
- **Transfer format kernel**
  - A hexadecimal (ASCII) version of a binary kernel, used ONLY for porting a binary kernel between incompatible computers.
  - Not as important as it was prior to the addition of the “run-time translation” capability added in Toolkit version N0052 (January 2002).
    - » But still has a role in making native binary kernels required for some operations.

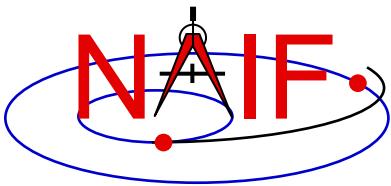


# SPICE Lexicon - 4

---

Navigation and Ancillary Information Facility

- “**Command file**”
  - Many SPICE application and utility programs either require, or optionally accept, an input file containing program directives, and sometimes input data.
  - The following names are used synonymously:
    - » setup file
    - » preferences file
    - » command file
    - » specifications file
    - » definitions file
- “**Found flag**”
  - A Boolean output (“True” or “False”) from a SPICE API that informs your program whether or not data were found that match your request
- **Database Kernel (DBK)**
  - A SPICE kernel that, in conjunction with Toolkit DBK software, provides a self-contained SQL-like database capability.

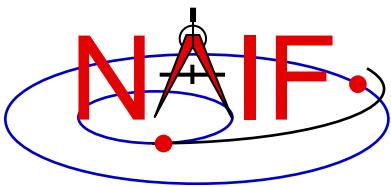


# SPICE Lexicon - 5

---

Navigation and Ancillary Information Facility

- **Coverage**
  - The period(s) of time for which a time-based kernel contains data
- **Deprecated software**
  - Toolkit code that, while still useable, has been superseded with a newer and presumably better version
  - We encourage you to not use deprecated SPICE software
  - Deprecated modules are so marked in their headers
- **Toolkit version naming**
  - "Nxxxx" e.g. N0066 is Version 66
    - » Often shortened to just Nxx (e.g. N66)
  - Used for all instances of a given toolkit release
    - » Fortran ("Toolkit"), C ("CSPICE"), IDL ("Icy"), MATLAB ("Mice")
- **“Satellite” is used to refer only to a natural satellite, never to a spacecraft.**
- **“Run-time” occurs when you execute a program**



# SPICE Lexicon - 6

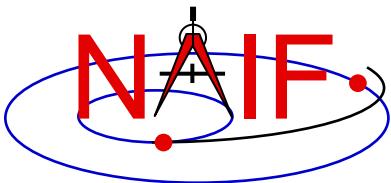
Navigation and Ancillary Information Facility

## Names used synonymously

- Kernel, SPICE file, SPICE kernel, SPICE kernel file
- Meta-kernel, Furnsh Kernel
- Module, routine, subroutine, procedure, function, application program interface (API)
- Application, program, utility, executable
- Metadata, comments
- Time, Epoch
- Encoded SCLK, ticks\*
- Frame, Reference Frame\*\* ( ≠ Coordinate System) (See the "Frames & Coordinate Systems tutorial")
- Ephemeris, trajectory
- Rectangular coordinates, Cartesian coordinates\*\*
- Geodetic, Planetodetic (coordinate system)
- Ephemeris time (ET), Barycentric Dynamical Time (TDB)
- Attitude, orientation
- International Celestial Reference Frame (ICRF) and Earth Mean Equator and Equinox of 2000 reference frame (J2000)
- “Body”, “solar system object,” “ephemeris object”

\* Encoded SCLK always refers to absolute time; “ticks” is used to refer to both durations and absolute times.

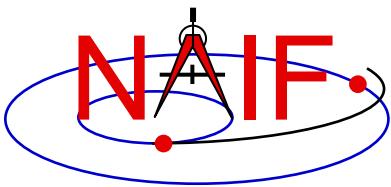
\*\* Outside of SPICE the term “coordinate system” is often used synonymously with “frame” or “reference frame.” We prefer to use “coordinate system” in the sense of describing how coordinates are measured (e.g. cylindrical coordinate system, rectangular coordinate system, polar coordinate system, etc) within a frame, and to use “frame” in the sense of a set of three orthogonal vectors that define an orientation.



# Kernel File Names

Navigation and Ancillary Information Facility

- **SPICE imposes some restrictions on kernel file names**
  - No white space allowed within a name
  - Maximum length of a name (including any path specifications) is 255 characters
    - » See the tutorial “Intro\_to\_kernels” for limitations on file name specifications contained within meta-kernels (“furnsh kernels”)
- **NAIF suggests names conform to the PDS3 standard: “36.3”**
  - <1 to 36 alphanumeric characters>.<1 to 3 chars>
- **Common usage within NAIF for SPICE kernel file name extensions is listed on the next page, with the following general style used:**
  - t\* text format (e.g. pck00010.tpc)
  - b\* binary format (e.g. de430.bsp)
  - x\* transfer format (e.g. de430.xsp)



# Common SPICE Kernel File Name Extensions

Navigation and Ancillary Information Facility

## SPK:

.bsp	<b>binary SPK file</b>
.xsp	<b>transfer format SPK file</b>

## PcK:

.tpc	<b>text PcK file</b> <small>(The most common type PcK)</small>
.bpc	<b>binary PcK file</b> <small>(Very few instances of this)</small>
.xpc	<b>transfer format PcK file</b>

## IK:

.ti	<b>text IK file</b>
-----	---------------------

## FK:

.tf	<b>text FK file</b>
-----	---------------------

## LSK:

.tls	<b>text LSK file</b>
------	----------------------

## CK:

.bc	<b>binary CK file</b>
.xc	<b>transfer format CK file</b>

## SCLK:

.tsc	<b>text SCLK file</b>
------	-----------------------

## MK:

.tm	<b>text meta-kernel file ("FURNSH kernel")</b>
-----	--

## DSK:

.bds	<b>binary DSK file</b>
------	------------------------

## DBK:

.bdb	<b>binary database kernel</b>
.xdb	<b>transfer format database kernel</b>

## EK Family (ESP, ESQ, ENB)

### ESP:

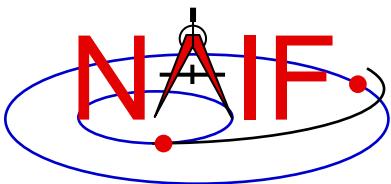
.bep	<b>binary Science Plan EK file</b>
.xep	<b>transfer format Science Plan EK file</b>

### ESQ:

.bes	<b>binary Sequence Component EK file</b>
.xes	<b>transfer format Sequence Component EK file</b>

### ENB:

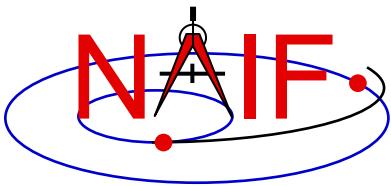
.ten	<b>text Experimenter's Notebook EK file</b>
------	---



# Common Document Name Extensions

Navigation and Ancillary Information Facility

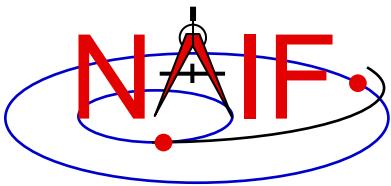
- These extensions are used for plain ASCII documents included with each Toolkit delivery
  - .ug User's Guide
  - .req “Required Reading” technical reference document
  - .txt Used for a few miscellaneous documents
  - .idx Used only for the permuted index document
- All HTML documents included in the Toolkit have extension .html



# Public and Private Modules

Navigation and Ancillary Information Facility

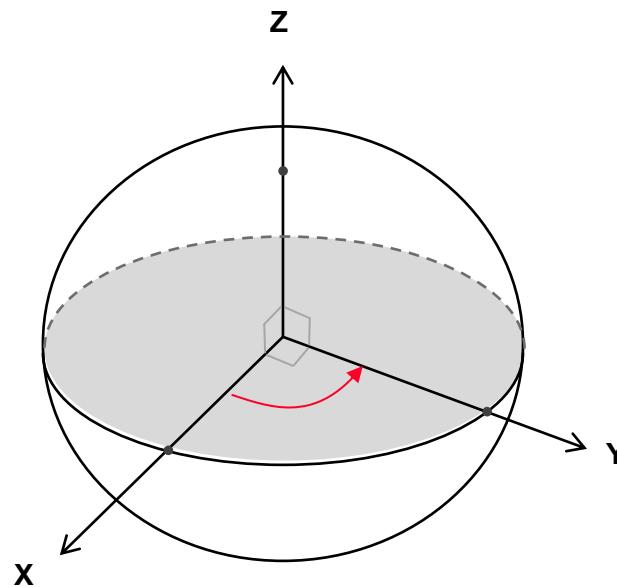
- All Toolkits include public and private modules
- Public modules are for you to use
  - Names of public APIs are different in the four SPICE library implementations. For example, the top level SPK reader SPKEZR has the following names
    - » in SPICELIB (FORTRAN) [SPKEZR](#)
    - » In CSPICE (C) [spkezr\\_c](#)
    - » ICY (IDL) [cspice\\_spkezr](#)
    - » MICE (MATLAB) [cspice\\_spkezr](#) and [mice\\_spkezr](#)
- Private modules are for NAIF staff use only
  - Names of private modules start with “ZZ”
  - They are present in the Toolkit only to support operations of “public” modules
  - Do not use “private” modules in your code – they may be changed by NAIF without notice

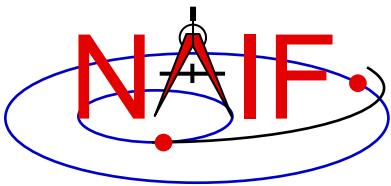


# Reference Frame Conventions

Navigation and Ancillary Information Facility

- All reference frames used within SPICE are right handed systems: this means  $X \times Y = Z$



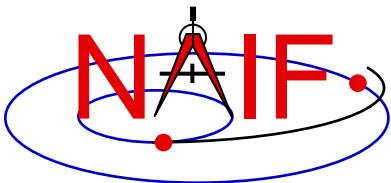


# Quaternions

---

Navigation and Ancillary Information Facility

- The SPICE system uses quaternions to provide orientation in C-kernels
- There are different “styles” of quaternions used in science and engineering applications. Styles are characterized by
  - The order of the quaternion elements
  - The quaternion multiplication formula
  - The convention for associating quaternions with rotation matrices
- Two of the commonly used styles are
  - “SPICE”
    - » Used by Sir William Rowan Hamilton (discoverer of quaternions)
    - » Used in math and physics textbooks
  - “Engineering” or “MSOP”
    - » Widely used in attitude control and other aerospace applications
- Details about SPICE quaternions are found in:
  - Rotations Required Reading document
  - NAIF white paper on quaternions: [https://naif.jpl.nasa.gov/pub/naif/mis/Quaternion\\_White\\_Paper/](https://naif.jpl.nasa.gov/pub/naif/mis/Quaternion_White_Paper/)
  - SPICE quaternion conversion routines: M2Q, Q2M

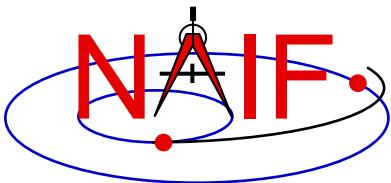


# Names and IDs

---

Navigation and Ancillary Information Facility

- Many items within SPICE have assigned names (text strings) and IDs (integer numbers)
- The rules, standards, practices and exceptions regarding these names and IDs are discussed in a separate tutorial (“NAIF IDs and Names”)



# Use of Quotes

---

Navigation and Ancillary Information Facility

- **Reminder of language-specific rules for quoting strings used as values in API arguments**
  - Use **double** quotes in C and Java Native Interface (JNI) codes  
`"this is a string"`
  - Use **single** quotes in Fortran, IDL, MATLAB, and Python codes  
`'this is a string'`
- **Regardless of the language version of the SPICE Toolkit you're using, in all SPICE text kernels, string values are enclosed in **single** quotes. For example:**

```
INS-43012_FOV_SHAPE = 'CIRCLE'
```

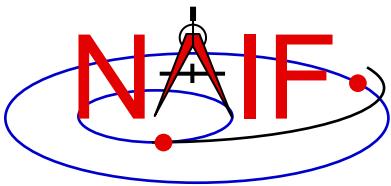


# Pluto is a Special Case in SPICE

---

Navigation and Ancillary Information Facility

- **For practical and historical reasons, Pluto is treated as a planet when speaking about ephemerides (SPK).**
- **But Pluto is treated as a “dwarf planet” when speaking about orientation and rotational state (PCK).**

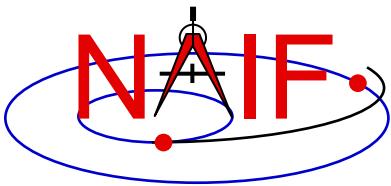


---

Navigation and Ancillary Information Facility

# IDs and Names

January 2020

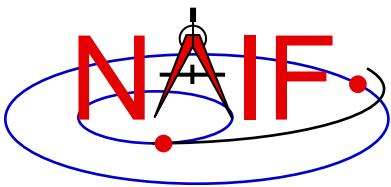


# Topics

---

Navigation and Ancillary Information Facility

- **Summary of naming/numbering schemes used in SPICE**
- **Naming/numbering of objects**
- **Naming/numbering of reference frames**
- **Naming/numbering of DSK surfaces**
- **Connection between the schemes**
- **Oddball cases: SCLK and CK IDs**
- **Some examples**

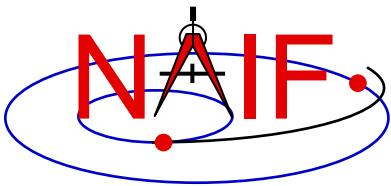


# Overview - 1

---

Navigation and Ancillary Information Facility

- **SPICE uses IDs and names to identify:**
    - objects
    - reference frames
    - digital shape kernel (DSK) surfaces
  - An ID is an integer number
  - A name is a text string
- 
- **IDs are used primarily as data identifiers inside SPICE kernels**
    - Users rarely have to use IDs
  - **Names are used primarily as input and output arguments in SPICE software interfaces (APIs)**
    - Users deal with lots of names

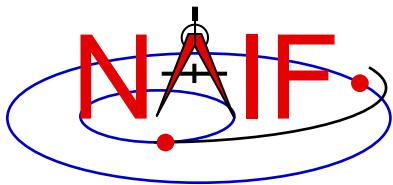


# Overview - 2

---

Navigation and Ancillary Information Facility

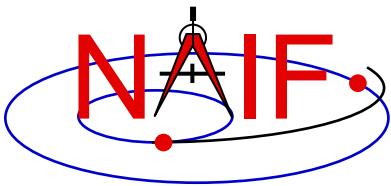
- **The schemes used for assigning IDs and names to objects and to reference frames are independent!**
  - This means that, in general, SPICE does not make any assumptions about reference frame names and IDs based on associated object names and IDs
    - » There are some exceptions; they will be mentioned later



---

**Navigation and Ancillary Information Facility**

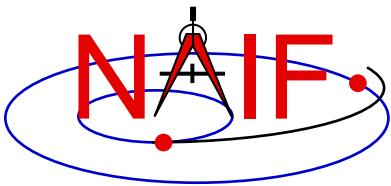
# **Names and IDs associated with Objects**



# Object IDs and Names

Navigation and Ancillary Information Facility

- **A single ID is assigned to an object of any of the following types:**
  - Natural bodies -- planets, satellites, comets, asteroids
  - Artificial bodies -- spacecraft, spacecraft structures, science instruments, individual detectors within science instruments, DSN stations
  - Any other point, the location of which can be known within the SPICE context, such as:
    - » barycenters of solar system and planetary systems, landing sites, corners of solar arrays, focal points of antennas, etc.
- **One or more names can be assigned to that same object**
- **Within SPICE software there is a 1-to-MANY mapping between the ID and the object's name(s)**
  - On input, the names are treated as synonyms
  - On output, the name that was last associated with the ID is returned

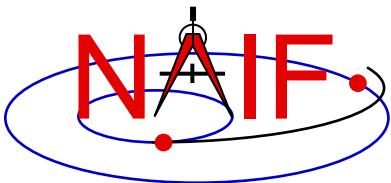


# Object IDs: Where Used? - 1

---

Navigation and Ancillary Information Facility

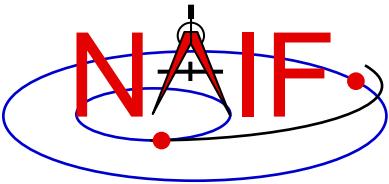
- **Object IDs are used in kernels as data identifiers:**
  - » in SPKs -- to identify a body and its center of motion
  - » in text PCKs -- in keywords associated with a body
  - » in DSKs -- to identify a body
  - » in IKs -- in keywords associated with an instrument
  - » in FKs -- to specify the center used in computing light-time correction, and to identify the body in PCK-based frames
  - » in FKs -- to identify target and observer in dynamic frame specifications
  - » in SCLKs -- normally the SCLK ID used in keywords is the negative of the spacecraft's ID (thus is a positive integer)
  - » ... and more...



# Object IDs: Where Used? - 2

Navigation and Ancillary Information Facility

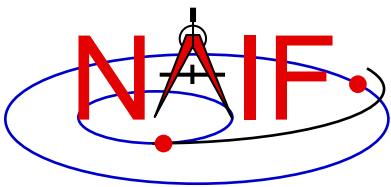
- Object IDs are used in some APIs as input and/or output arguments:
  - » in older SPK routines -- SPKEZ, SPKEZP, SPKGEO, ...
  - » in older derived geometry routines -- ET2LST, ...
  - » in older PCK routines -- BODVAR, BODMAT, ...
  - » in IK routines -- GETFOV, indirectly in G\*POOL, ...
  - » in SCLK routines -- SCE2C, SCT2E, ...
  - » in coverage routines -- SPKOBJ, SPKCOV, CKOBJ, CKCOV, DSKOBJ, DSKSRF
  - » ... and more...



# Object Names – Where Used?

Navigation and Ancillary Information Facility

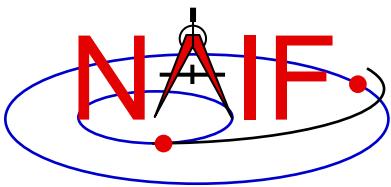
- Object names are used in the high-level user APIs as input and/or output arguments:
  - » in newer SPK routines -- SPKEZR, SPKPOS
  - » in newer derived geometry routines -- SINCPT, ILUMIN, SUBPNT, SUBSLR, LIMBPT, TERMPT, LATSRF ...
  - » in high-level Geometry Finder routines – GFPOS, GFDIST, GFSEP, GFILUM, ...
  - » in newer PCK routines -- BODVRD, ...
- Object names are **not** used as data identifiers within kernels



# Object IDs and Names – Where Defined?

Navigation and Ancillary Information Facility

- Object name-to-ID mappings used by SPICE may be defined in two places
  - Built into Toolkit software: hard-coded in source code
    - » See NAIF\_IDS.REQ for a complete listing of these built-in (default) assignments
  - In text kernels
    - » Normally used to define name/ID mappings for instruments, their subsystems/detectors and spacecraft structures
      - See comments and the actual data sections in a text kernel for the complete listing of the names/IDs defined in that kernel
    - » These assignments exist most often in FKs (e.g. MER, MEX, JUNO, MSL), sometimes in IKs (e.g. CASSINI, MGS), but could be placed in any text kernel
  - Mappings defined in text kernels take precedence over those defined in Toolkit source code

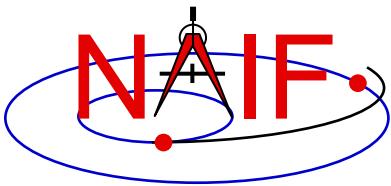


# Examples of Object IDs and Names

## Spacecraft and Ground Stations

Navigation and Ancillary Information Facility

- **Spacecraft (negative numbers)**
  - Within NASA, this number is generally the negative of the numeric ID assigned by the NASA control authority at GSFC
    - -6                    'PIONEER-6', 'P6'
    - -64                 'OSIRIS-REX', 'ORX'
    - -74                 'MARS RECON ORBITER', 'MRO'
    - -82                 'CASSINI', 'CAS'
    - ...
  - Unfortunately sometimes NASA re-uses a number
    - » For example -18 for MGN and LCROSS, -53 for MPF and M01
    - » This will happen with increasing frequency in the future
    - » Probably a new scheme is needed
- **DSN ground stations (399000 + station number)**
  - 399005            'DSS-05'
  - ...
  - 399066            'DSS-66'
- **Non-DSN stations (398000 + some integer 0 to 999)**
  - 398990            'NEW\_NORCIA'
  - ...



# Examples of Object IDs and Names

## Planets

---

Navigation and Ancillary Information Facility

- **Solar System Barycenter and Sun\* (0 and 10)**

- 0                    'SOLAR SYSTEM BARYCENTER', 'SSB'
- 10                  'SUN'

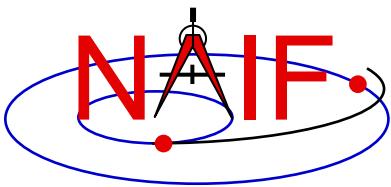
- **Planetary system barycenters (numbers from 1 to 9)**

- 1                    'MERCURY BARYCENTER'
- 2                    'VENUS BARYCENTER'
- 3                    'EARTH MOON BARYCENTER', 'EMB', ...
- 4                    'MARS BARYCENTER'
- ...
- 9                    'PLUTO BARYCENTER' (Within SPICE Pluto is still treated as a planet!)

- **Planet-only mass centers (planet barycenter ID \* 100 + 99)**

- 199                'MERCURY'
- 299                'VENUS'
- 399                'EARTH'
- 499                'MARS'
- ...
- 999                'PLUTO' (Within SPICE Pluto is still treated as a planet!)

\* Barycenter: the center of mass of a system (collection) of two or more bodies, each of which orbits that point.  
See the SPK tutorial for details.



# Examples of Object IDs and Names

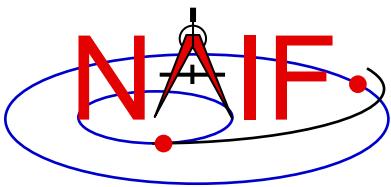
## Satellites

---

Navigation and Ancillary Information Facility

- **Satellites (planet barycenter ID\*100 + number <1... 98>)**

- 301            'MOON'
- 401            'PHOBOS'
- 402            'DEIMOS'
- 501            'IO'
- 502            'EUROPA'
- ...
- 901            'CHARON', '1978P1'
- 902            'NIX'



# Examples of Object IDs and Names

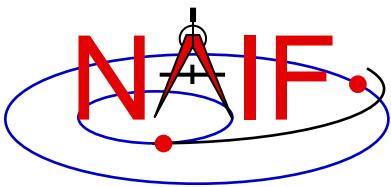
## Comets & Asteroids

---

Navigation and Ancillary Information Facility

- **Periodic Comets (1000000 + sequence number \*)**
  - 1000001            'AREN'D'
  - 1000002            'AREN'D-REGAUX'
  - ...
  - 1000032            'HALE-BOPP'
- **Numbered Asteroids (2000000 + IAU asteroid number)**
  - 2000001            'CERES'
  - 2000004            'VESTA'
  - ...
  - 2009969            'BRAILLE', '1992KD'
  - There are three exceptions, for Gaspra, Ida and Dactyl
    - » See NAIF\_IDS.REQ
  - One can search here for a name or ID and find the corresponding item: <http://ssd.jpl.nasa.gov/sbdb.cgi>

\*Sequence number is assigned by JPL's Solar System Dynamics Group



# Examples of Object IDs and Names

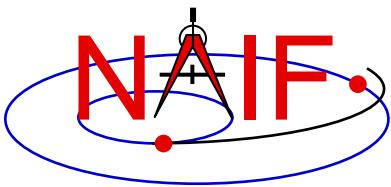
## Instruments

Navigation and Ancillary Information Facility

*the “minus” sign*

- **Science Instruments (s/c ID\*1000 - instrument number)**

- An instrument number should be picked for **EVERY** instrument, instrument subsystem or detector, or spacecraft structure, the parameters for which are to be stored in IKs, or the location of which is to be stored in SPKs
- Instrument numbers are picked from the range 0...999. The only requirement is that they must be unique within each mission
  - ...
  - -82760        ‘CASSINI\_MIMI\_CHEMS’
  - -82761        ‘CASSINI\_MIMI\_INCA’
  - -82762        ‘CASSINI\_MIMI\_LEMMS1’
  - -82763        ‘CASSINI\_MIMI\_LEMMS2’
  - ...
  - -82001        ‘CASSINI\_SRU-A’
  - -82002        ‘CASSINI\_SRU-B’
  - -82008        ‘CASSINI\_SRU-A\_RAD’
  - -82009        ‘CASSINI\_SRU-B\_RAD’
  - ...



# Object IDs/Names -- Mapping APIs

Navigation and Ancillary Information Facility

- **SPICE provides two routines to map object IDs to names, and vice versa**

- To get the ID for a given object name:

**CALL BODN2C ( NAME, ID, FOUND )**

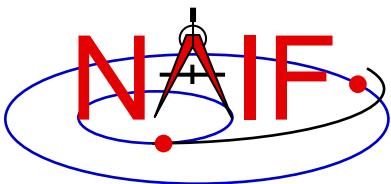
**CALL BODS2C ( NAME, ID, FOUND )**

(This is a more general version as compared to BODN2C. Use this one.)

- To get the name for a given object ID:

**CALL BODC2N( ID, NAME, FOUND )**

- If the “FOUND” flag returned by either of these routines comes back FALSE, then the input ID or name cannot be mapped



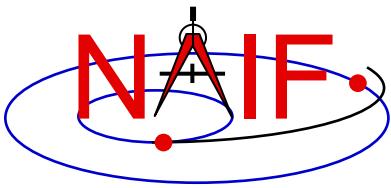
# Adding New or Additional Object Name-to-ID Mappings

Navigation and Ancillary Information Facility

- You may define new or additional name-to-ID mappings using assignments inside any text kernel.
- For example, for a spacecraft:

```
NAIF_BODY_NAME    += ( 'my_spacecraft_name'      )
NAIF_BODY_CODE    += ( my_spacecraft_ID          )
↑
Note the combination of + and =
```

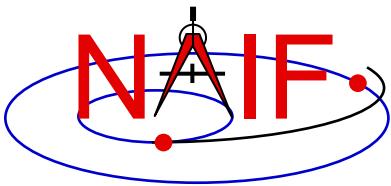
- See “NAIF\_IDs Required Reading” for details
- Caution: the object name length is limited to 36 characters



---

Navigation and Ancillary Information Facility

# **Names and IDs associated with Reference Frames**

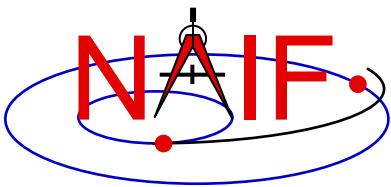


# Frame IDs and Names

---

Navigation and Ancillary Information Facility

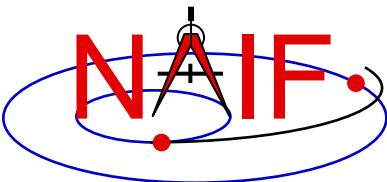
- A single ID and a single name are assigned to a **reference frame** of any of the following types
  - Inertial frames
  - Body-fixed frames
  - Spacecraft and instrument frames
  - Topocentric frames
  - Any other reference frame for which the orientation may be needed to compute observation geometry



# Frame IDs and Names – Where Defined?

Navigation and Ancillary Information Facility

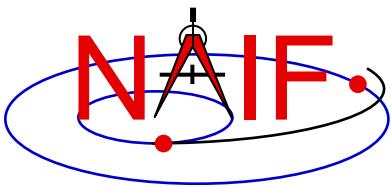
- The reference frame name-to-ID mappings used by the SPICE system are defined in two places
  - Built into Toolkit software: hard-coded in source code
    - » For inertial frames
    - » For body-fixed frames defining the orientation for planets and most satellites
    - » See FRAMES REQUIRED READING for a complete listing
  - In text kernels: provided by KEYWORD=VALUE sets
    - » Almost always placed in FKs
    - » Rarely placed in other kernels, but could be in any text kernel
      - (For example during operations MGS frames were defined in IKs and SCLK)
- Unlike for objects, only one name may be directly associated with a reference frame ID
  - However, an “alias” for a given reference frame can be established by defining a new, zero-offset frame with its own unique name and ID



# Frame IDs and Names – Where Used?

Navigation and Ancillary Information Facility

- **Reference frame IDs** are used in the following kernels as data identifiers
  - » in FKs -- to “glue” frame definition keywords together
  - » in SPKs -- to identify base reference frames
  - » in PCKs -- to identify base reference frames
  - » in CKs -- to identify base reference frames
  - » in DSKs -- to identify reference frames
- Reference frame IDs are **not** used as input and/or output arguments in any high level user APIs
- **Reference frame names** are used
  - as arguments in all high level APIs that require a reference frame to be specified as an input
    - » in derived geometry routines -- SINCPT, ILUMIN, SUBPNT, ...
    - » in frame transformation routines -- PXFORM, SXFORM
    - » In SPK routines -- SPKEZR, SPKPOS, ...
  - Frame names are **not** used as data identifiers within kernels

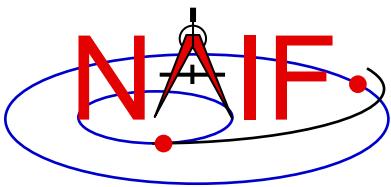


# Examples of Frame IDs and Names Inertial and Body-fixed

---

Navigation and Ancillary Information Facility

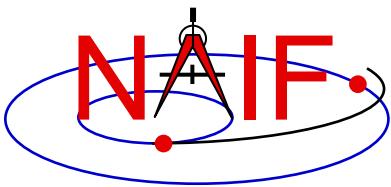
- Inertial frames (positive integers starting at 1)
  - 1                    'J2000'
  - ...
  - 17                  'ECLIPJ2000'
  - ...
- Body-fixed frames (positive integers starting at 10001)
  - ...
  - 10012              'IAU\_VENUS'
  - 10013              'IAU\_EARTH'
  - 10014              'IAU\_MARS'
  - ...
  - 10020              'IAU\_MOON'
  - ...
  - 13000              'ITRF93'
  - ...
- **NOTE: SPICE users would rarely if ever need to know or use the frame IDs; you use only the frame names**



# Examples of Frame IDs and Names Spacecraft and Instrument

Navigation and Ancillary Information Facility

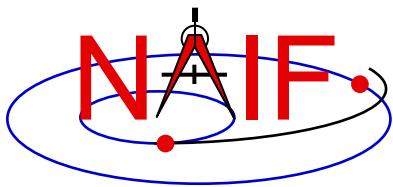
- IDs for frames associated with spacecraft, spacecraft structures, and instruments are usually defined as:  
s/c ID times 1000 minus an arbitrary number
- Examples based on Cassini:
  - Spacecraft frame (ID and name)  
-82000 ‘CASSINI\_SC\_COORD’
  - Spacecraft structure frame (ID and name)  
-82001 ‘CASSINI\_SRU-A’
  - Instrument frames (ID and name)  
-82760 ‘CASSINI\_MIMI\_CHEMS’  
-82761 ‘CASSINI\_MIMI\_LEMMS\_INCA’  
-82762 ‘CASSINI\_MIMI\_LEMMS1’  
-82763 ‘CASSINI\_MIMI\_LEMMS2’  
-82764 ‘CASSINI\_MIMI\_LEMMS\_BASE’  
-83765 ‘CASSINI\_MIMI\_LEMMS\_ART’  
...
- **NOTE: SPICE users would rarely if ever need to know or use the frame IDs; you use only the frame names**



# Frame IDs/Names -- Mapping APIs

Navigation and Ancillary Information Facility

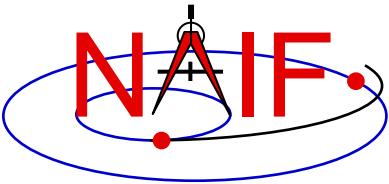
- **SPICE provides three routines to convert (map) reference frame IDs to names, and vice versa**
  - To get the ID for a given reference frame name:  
`CALL NAMFRM( NAME, ID )`
  - To get the name for a given reference frame ID:  
`CALL FRMNAM( ID, NAME )`
  - If the ID or name cannot be mapped, these routines return zero and an empty(blank string respectively.
- **Note: SPICE users will rarely if ever need to call these routines**



---

Navigation and Ancillary Information Facility

# **Names and IDs associated with DSK Surfaces**

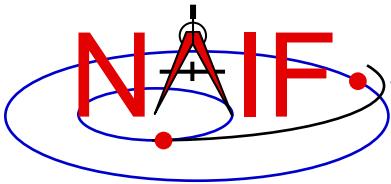


# Surface IDs and Names

---

Navigation and Ancillary Information Facility

- A single ID is assigned to each DSK topography data set for a particular body
  - this ID is called **DSK surface ID** or simply **surface ID**
- One or more names can be associated with that surface ID
  - These names are called **DSK surface names** or simply **surface names**
- Within SPICE software there is a 1-to-MANY mapping between the surface IDs and names
  - On input, the names are treated as synonyms
  - On output, the name that was last associated with the ID is returned



# Surface IDs and Names – Where Defined?

Navigation and Ancillary Information Facility

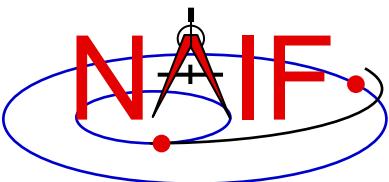
- The surface name-to-ID mappings used by the SPICE system are defined in text kernels
  - usually in FKS
  - using these triplets of keywords that include the ID of the body, placing the name-to-ID mapping in the body's "namespace"

```
NAIF_SURFACE_NAME += ( 'my_surface_name' )  
NAIF_SURFACE_CODE += ( my_surface_ID )  
NAIF_SURFACE_BODY += ( body_ID )
```



Note the combination of + and =

- Since surface name-to-ID mappings are defined within a body's "namespace", the same surface names and IDs can be used for other bodies if desired

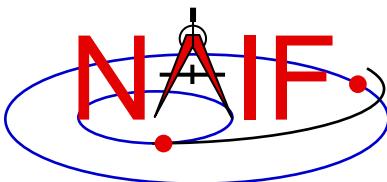


# Surface IDs and Names – Where Used?

---

Navigation and Ancillary Information Facility

- **Surface IDs are used as data identifiers:**
  - inside DSK files
  - in some mid-level DSK APIs -- DSKXV, DSKXSI
  - in place of surface names in the METHOD argument in derived geometry routines
- **Surface names are used as topography data set identifiers in all high-level, DSK-enabled APIs:**
  - in derived geometry routines -- SINCPT, ILUMIN, SUBPNT, ...
  - in topography sampling routine -- LATSRF
  - in Geometry Finder routines -- GFOCLT



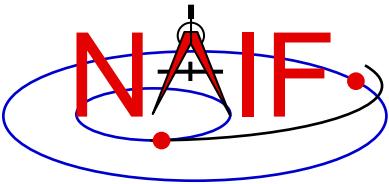
# Surface IDs and Names – Example

Navigation and Ancillary Information Facility

- Surface name-ID mappings for ROSETTA targets

DSK Surface Name	ID	Body ID	
ROS(CG)_M004_NSPCESA_N_V1	11000	1000012	(comet C-G)
ROS(CG)_K250_NSPCESA_N_V1	11001	1000012	
...			
ROS(CG)_M001_OSPGDLR_N_V1	24003	1000012	
ROS(CG)_M004_OSPGDLR_N_V1	24004	1000012	
ROS(LU)_K003_OSPCLAM_N_V1	1000	2000021	(aasteroid Lutetia)
ROS(LU)_K006_OSPCLAM_N_V1	1001	2000021	
...			
ROS(LU)_M003_OSPCLAM_N_V1	1011	2000021	
ROS(ST)_K020_OSPCLAM_N_V1	1000	2002867	(asteroid Steins)

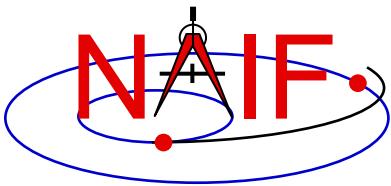
Note: the ROSETTA DSK surface name-ID schema appears rather cryptic because of the need to accommodate multiple shape model versions from multiple producers for three different targets.



# Surface Name/ID Mapping APIs

Navigation and Ancillary Information Facility

- **SPICE provides four routines to convert (map) surface IDs to names, and vice versa**
  - To get the surface ID for a given surface name and body name:  
`CALL SRFS2C( SRFSTR, BODSTR, CODE, FOUND )`
  - To get the surface ID for a given surface name and body ID:  
`CALL SRFSCC( SRFSTR, BODYID, CODE, FOUND )`
  - To get the surface name for a given surface ID and body name:  
`CALL SRFCSS( CODE, BODSTR, SRFSTR, FOUND )`
  - To get the surface name for a given surface ID and body ID:  
`CALL SRFC2S( CODE, BODYID, SRFSTR, FOUND )`
- **If the “FOUND” flag returned by either of these routines comes back FALSE, then the input ID or name cannot be mapped**

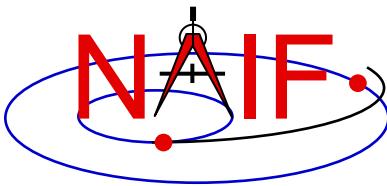


# Connections between Objects and Frames

---

Navigation and Ancillary Information Facility

- Although object and reference frame naming/numbering schemes are independent, there is nevertheless much overlap in the way **objects** and **frames** are named and numbered
- This overlap is due to the following reasons
  - Conventions adopted over the course of SPICE implementation
    - » Example: PCK-based body-fixed frames for planets and satellites are named ‘IAU\_<body name>’
      - However, the IDs of these **frames** have nothing in common with the IDs of the **objects** (bodies) for which these frames are defined
    - The need for the object and frame IDs to be unique
      - » For this reason both the instrument (**object**) IDs and the instrument **frame** IDs are derived from the ID of the spacecraft on which the instrument is flown
    - The need for the object and frame names to be meaningful
      - » For this reason the instrument **frame** names normally contain both the name of the spacecraft and the name of the instrument



# “Odd Ball” Cases

---

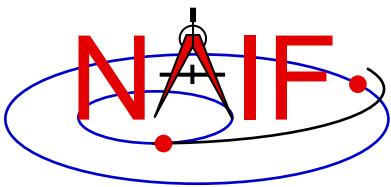
Navigation and Ancillary Information Facility

- **CK IDs**

- Historically IDs used in CKs are called structure IDs but in reality they are much more closely related to frames than to objects
- To find which frame is associated with a particular CK ID, look through the FK for a frame whose `_CLASS_ID` keyword is set to the CK ID
  - » For CK-based frames both the frame ID and frame `CLASS_ID` are set equal to the CK ID

- **SCLK IDs**

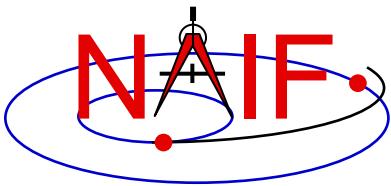
- Because most spacecraft have only one on-board clock, the SCLK ID of that clock is simply the spacecraft ID. This SCLK ID is used by SPICE APIs. **Caution: the negative of the SCLK ID is used in SCLK kernel keywords.**
- Should a spacecraft carry more than one independent clock, unique SCLK IDs for these other clocks would be needed
  - » Normally the ID of an additional clock will be set to the ID of the instrument, of which that clock is a part
- SCLK IDs are used in SCLK APIs (must be provided by the user) and by the frames subsystem when it reads CKs to determine orientation of CK-based frames (gets SCLK ID from CK `* SCLK` keyword provided in the frame definition or computes it by dividing CK ID by 1000)



# Name/IDs Example -- CASSINI (1)

Navigation and Ancillary Information Facility

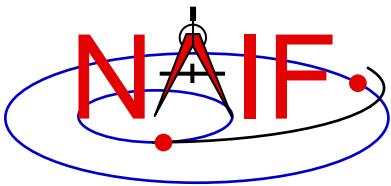
	<u>Objects</u> IDs/Names	<u>Frames</u> IDs/Names
Ephemeris objects	10 'SUN' 399 'EARTH' 699 'SATURN' 601 'MIMAS' 602 'ENCELADUS'	1 'J2000' 10013 'IAU_EARTH' 10016 'IAU_SATURN' 10039 'IAU_MIMAS' 10040 'IAU_ENCELADUS'
Spacecraft and its structures	-82 'CASSINI' -82001 'CASSINI_SRU-A'	-82000 'CASSINI_SC_COORD' -82001 'CASSINI_SRU-A'
CDA instrument	-82790 'CASSINI_CDA'	-82790 'CASSINI_CDA' -82791 'CASSINI_CDA_ART' -82792 'CASSINI_CDA_BASE'
CAPS instrument	-82820 'CASSINI_CAPS_IMS' -82821 'CASSINI_CAPS_ELS' -82822 'CASSINI_CAPS_IBS_DT1' -82823 'CASSINI_CAPS_IBS_DT2' -82824 'CASSINI_CAPS_IBS_DT3'	-82820 'CASSINI_CAPS' -82821 'CASSINI_CAPS_ART' -82822 'CASSINI_CAPS_BASE'



# Name/IDs Example -- CASSINI (2)

Navigation and Ancillary Information Facility

- The lists provided on the previous page are by no means complete
  - There are many more Saturnian satellites and other natural bodies of interest to the Cassini mission, each having an associated frame
  - There are many more instruments on the Cassini spacecraft, with multiple frames associated with each of them
- To find names and IDs associated with these objects and frames, users should refer as follows
  - For names/IDs of Cassini instruments: Cassini IKs
    - » For other missions this information is in the mission's FK
  - For names of the reference frames associated with the Cassini spacecraft, its subsystems and instruments: the Cassini FK
  - For names of inertial frames and body-fixed frames associated with natural bodies: FRAMES.REQ
  - For names/IDs of natural objects: NAIF\_IDS.REQ



---

Navigation and Ancillary Information Facility

# Getting, Installing and Verifying the SPICE Toolkit

January 2020

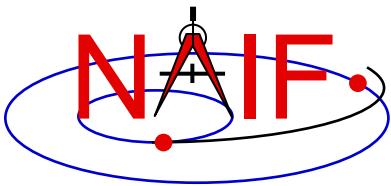


# Getting the Toolkit

---

Navigation and Ancillary Information Facility

- All official NAIF-supported instances of the SPICE Toolkit are freely available from the NAIF server
  - <https://naif.jpl.nasa.gov/naif/toolkit.html>
  - No password or identification is needed
- To download a Toolkit package
  - Select language – FORTRAN, C, IDL, MATLAB or Java Native Interface
  - Select computer platform/OS/compiler combination
    - » Be careful to pick the right architecture: 64 or 32 bit
  - Download all toolkit package components
    - » Package file – toolkit.tar.Z (or toolkit.zip),  
cspice.tar.Z (or cspice.zip),  
icy.tar.Z (or icy.zip),  
mice.tar.Z (or mice.zip),  
JNISpice.tar.Z (or JNISpice.zip)
    - » Installation script (if present) – import\*.csh
    - » Accompanying documents - README, dscriptn.txt, whats.new



# Installing the Toolkit

---

Navigation and Ancillary Information Facility

- To install the Toolkit on Linux or Mac platform, follow the directions given in the README. Normally this consists of the following:

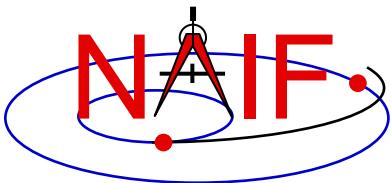
```
prompt> chmod u+x importSpice.csh ( or chmod u+x import<language>.csh )
prompt> ./importSpice.csh   ( or ./import<language>.csh )
prompt> rm toolkit.tar     ( or rm <toolkit_name>.tar )
```

- To install the Toolkit on a PC running Windows, do the following:

- unzip the toolkit (or cspice or icy or mice) to expand the archive.

```
prompt> unzip toolkit.zip
```

- You now have the expanded toolkit (or cspice or icy or mice or JNISpice) package. In it the APIs are already compiled into object modules, the needed libraries have been assembled, and the several Toolkit utility executables have been built. In most cases you need NOT re-do any of this build work! But read on about some special circumstances.

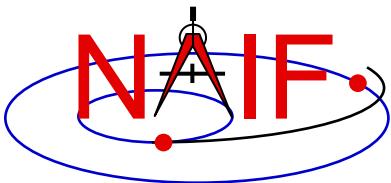


# Configuring Your Computer

---

Navigation and Ancillary Information Facility

- For some programming environments there are **required** additional steps to prepare for programming using SPICE.
- For some programming environments there are recommended additional steps to make program development easier.
- Read the “Preparing for Programming” tutorial and the “README” file found in the Toolkit download directory for more information!

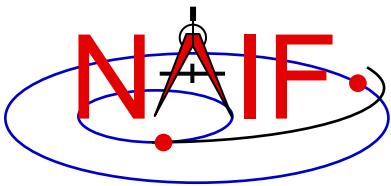


# Checking It Out

Navigation and Ancillary Information Facility

- Try an executable: **tobin**
  - Use the Toolkit's *tobin* utility to convert the SPICE transfer format SPK files supplied with the Toolkit into binary format.
  - The available transfer format files, `cook_01.tsp` and `cook_02.tsp`, are found in the `../data` directory\*
    - » For example try this:  
`prompt> tobin cook_01.tsp`
    - » This should produce an output file named `cook_01.bsp`
  - Then try using *brief* to summarize the converted SPK kernel  
`prompt> brief cook_01.bsp`

\* According to modern SPICE conventions, the file name extension ".tsp" seen above should be ".xsp". The ".tsp" extension is kept for historical reasons.

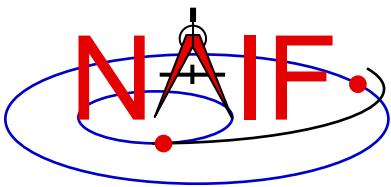


# Installation Problems?

---

Navigation and Ancillary Information Facility

- **New versions of operating systems, compilers, and MATLAB and IDL processors are released rather frequently as compared to the frequency of new SPICE Toolkit releases**
  - Sometimes this results in incompatibility issues with SPICE
- **Sometimes a customer wants to use the Toolkit in an environment not (officially) supported by NAIF**
  - » Example: Octave instead of Matlab
  - » Example: Ubuntu instead of Linux
  - » Example: clang instead of gcc
  - Porting a Toolkit to an unsupported environment might be straightforward, but could be problematic
- **See the next charts for a bit more information regarding Toolkit installation issues**

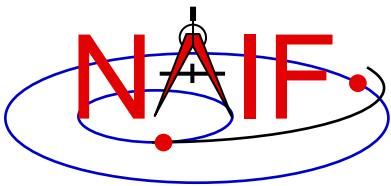


# Compatibility Issues

---

Navigation and Ancillary Information Facility

- **Problems may occur if your version of the operating system, compiler, or IDL or MATLAB is substantially newer or older than what NAIF used in making its build**
  - First examine NAIF's “Bugs” webpage for any relevant info:
    - » <https://naif.jpl.nasa.gov/naif/bugs.html>
  - Then try rebuilding the Toolkit using the script “makeall.csh” (or “makeall.bat”) located in the “top level” directory (toolkit or cspice or icy or mice)
- **If this doesn't seem to work, contact NAIF, providing error messages observed and version numbers for your OS and your compiler or your MATLAB or IDL app**

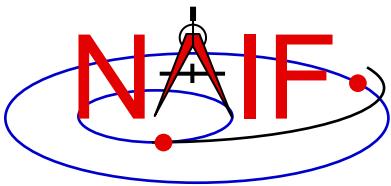


# Porting the Toolkit

---

Navigation and Ancillary Information Facility

- The packages provided on the NAIF server have been built and tested by NAIF using the particular environments shown at the end of this tutorial.
- If you try porting an instance of the Toolkit to an unsupported environment there are numeric and possibly compiler optimization issues that must be carefully dealt with.
  - You should definitely run NAIF's test harness (e.g. `tspice`, for Fortran, or `tspice_c`, for cspice/Mice/Icy) as part of your porting confirmation process.
  - Contact one of the NAIF team members for access to and instructions on running the appropriate `tspice` test harness.
    - » <https://naif.jpl.nasa.gov/naif/contactinfo.html>

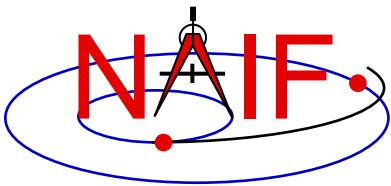


---

Navigation and Ancillary Information Facility

# Introduction to the Family of SPICE Toolkits

January 2020

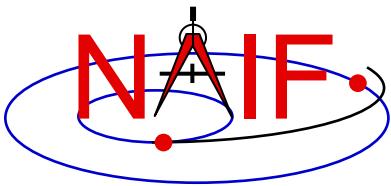


# Topics

---

Navigation and Ancillary Information Facility

- **Architecture**
- **Contents**
- **Characteristics**
- **Versions**
- **Capabilities**
- **Directory Structure**
- **Application Programs**
- **Utility Programs**
- **Documentation**
- **Backup: Currently Supported Environments**



# Toolkit Architecture

---

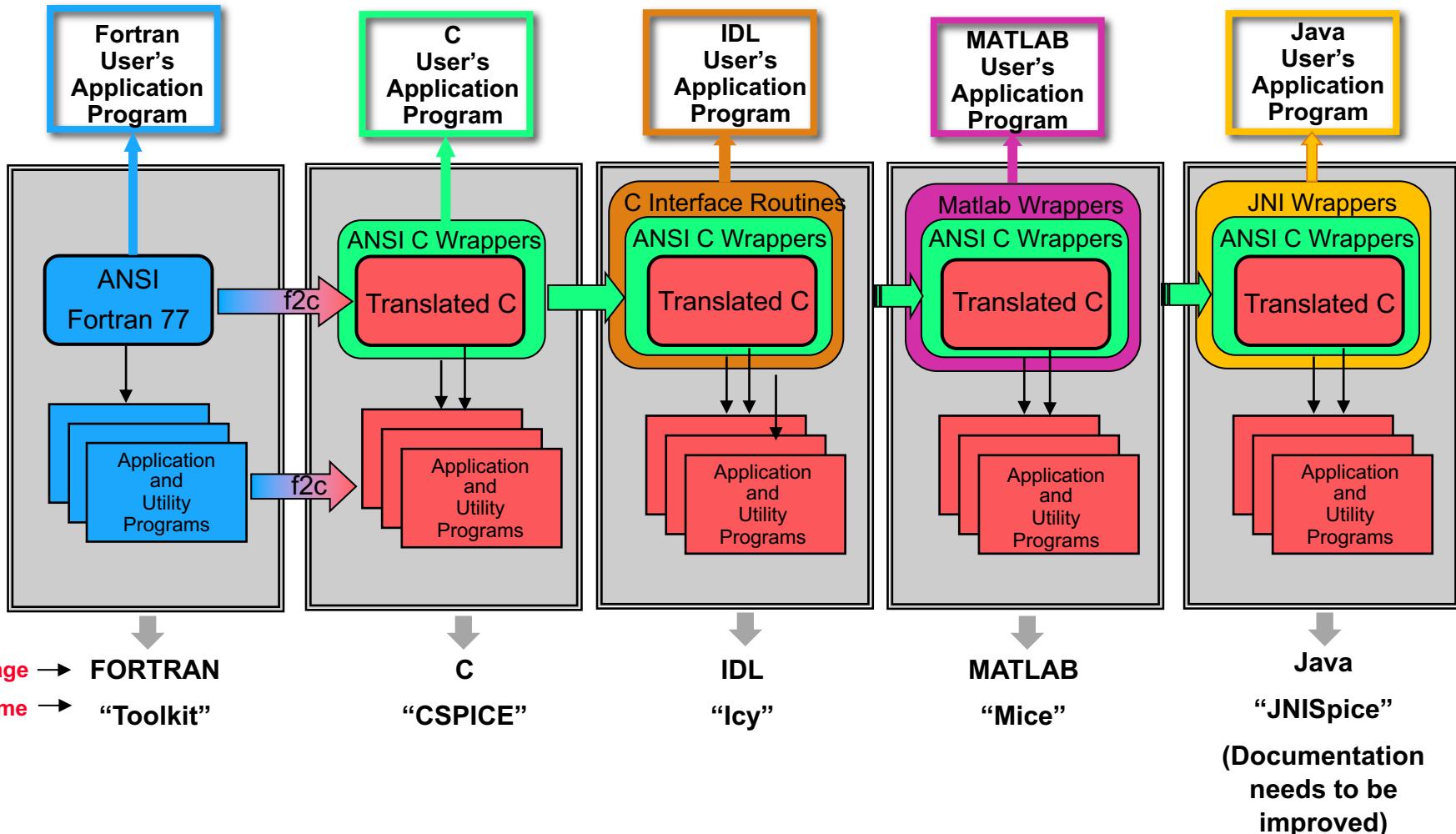
Navigation and Ancillary Information Facility

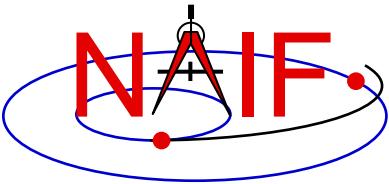
- The SPICE Toolkit is officially available in Fortran 77, C, IDL and MATLAB.
  - <https://naif.jpl.nasa.gov/naif/toolkit.html>
  - A beta Java Native Interface version (JNISpice) is also available
    - » [https://naif.jpl.nasa.gov/pub/naif/misc/JNISpice\\_N0066/](https://naif.jpl.nasa.gov/pub/naif/misc/JNISpice_N0066/)
- The Toolkits are packaged and delivered as standalone products.
  - The IDL, MATLAB and JNISpice Toolkits by necessity also include the complete C Toolkit.
- Other people have created Python, Ruby, Swift and Julia toolkits, available from their own websites.
  - <https://naif.jpl.nasa.gov/naif/links.html>
  - NAIF has NOT been involved in creating, testing or documenting these. Check with their authors about functionality and details.



# Toolkit Architecture Pictorial

Navigation and Ancillary Information Facility





# Fortran Toolkit

---

Navigation and Ancillary Information Facility

- “Toolkit,” the Fortran 77 Toolkit.
  - Developed first: in use since February 1990.
  - Contains code written in ANSI Standard Fortran 77.
    - » A few widely supported non-ANSI extensions are used, for example DO WHILE, DO...END DO.
  - Compiles under a wide variety of Fortran compilers.
    - » While NAIF cannot guarantee proper functioning of SPICE under F90/F95 compilers except on officially supported environments, those compilers might properly compile SPICELIB with the resulting libraries being callable from F90/F95 code if that compiler supports the F77 standard.



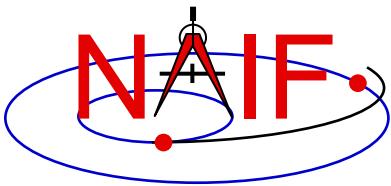
# C Toolkit

---

Navigation and Ancillary Information Facility

- “**CSPICE**,” the C-language Toolkit
  - Designed to duplicate the functionality of the Fortran Toolkit.
  - All CSPICE source code is in ANSI C.
    - » The Fortran SPICE Toolkit code is converted to ANSI C using the automatic translation program f2c.
    - » High-level functions have been hand-coded in C and documented in C style in order to provide a natural C-style API. These functions are called “wrappers.”
    - » Most wrappers encapsulate calls to C functions generated by f2c
      - The simpler wrappers do their work in-line to boost performance
    - » f2c'd functions may be called directly, but this is strongly discouraged since f2c'd functions emulate Fortran functionality:
      - Call by reference
      - Fortran-style array indexing
      - Fortran-style strings

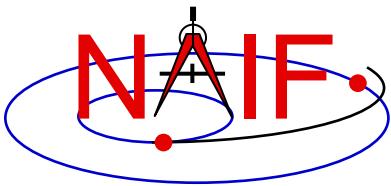
continued on next page



# C Toolkit, continued

## Navigation and Ancillary Information Facility

- CSPICE runs under a wide variety of ANSI C compilers.
- CSPICE functions may be called from within C++ source code.
  - » CSPICE prototypes are protected from name mangling.
- Current CSPICE Limitations
  - » Not all “Required Reading” reference documents have been converted to C style, with C examples.
    - Eventually all will be converted.
  - » CSPICE wrappers do not exist for every API provided in the Fortran toolkits.
    - But CSPICE does include all the most commonly used modules.
    - More will be added as time permits.
  - » In some very limited cases, code generated by f2c fails to emulate Fortran accurately. Should not be a problem.
    - List-directed I/O has some problems (not consequential for CSPICE).
    - Treatment of white space in text output is slightly different in CSPICE.
    - Logical unit-to-file name translation does not handle file name “synonyms” properly under Linux: once opened with a specified name, a file must be referred to using the same name throughout a program run.

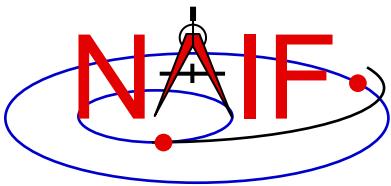


# IDL Toolkit

---

Navigation and Ancillary Information Facility

- “Icy,” the Interactive Data Language Toolkit
  - Provides an IDL-callable “wrapper” interface for many CSPICE wrapper routines.
    - » Example:
      - CSPICE: `spkezr_c ( targ, et, ref, abcorr, obs, state, &lttime );`
      - Icy: `cspice_spkezr, targ, et, ref, abcorr, obs, state, ltime`
    - » NAIF will add additional interfaces to Icy as time permits.
  - By necessity all Icy Toolkit packages include the complete CSPICE Toolkit.
    - » Additional Icy software components are:
      - IDL interface wrappers (implemented in ANSI C)
      - Icy cookbook programs (implemented in IDL)
  - Icy Documentation
    - » Icy Reference Guide
      - Principal documentation showing how to call Icy wrappers.
      - Each Icy wrapper has an HTML page containing usage examples serving as the Icy “module header”.
    - » Icy Required Reading
      - Provides background information essential for programming with Icy.
- See the “IDL\_Interface” tutorial for details

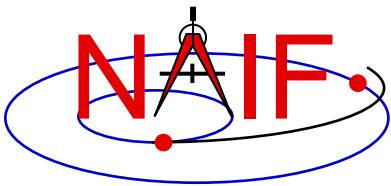


# Matlab Toolkit

---

Navigation and Ancillary Information Facility

- “Mice,” the Matlab Toolkit
  - Mice provides a Matlab-callable “wrapper” interface for many CSPICE wrapper routines
    - » Example:
      - CSPICE: `spkezr_c ( targ, et, ref, abcorr, obs, state, &lttime );`
      - Mice: `[state, ltime] = cspice_spkezr( targ, et, ref, abcorr, obs)`
    - By necessity all Mice Toolkit packages include the complete CSPICE Toolkit.
      - » Additional Mice software components are:
        - Matlab interface wrappers (implemented in Matlab wrapper scripts calling the ANSI C based interface library)
        - Mice cookbook programs (implemented in Matlab script)
      - Mice Documentation
        - » Mice Reference Guide
          - Principal documentation showing how to call Mice wrappers
          - Each Mice wrapper script has a documentation header containing usage examples, serving as SPICE “module header”, available from the `help` command. This documentation also exists as an HTML page.
        - » Mice Required Reading
          - Provides background information essential for programming with Mice
  - See the “Matlab\_Interface” tutorial for details

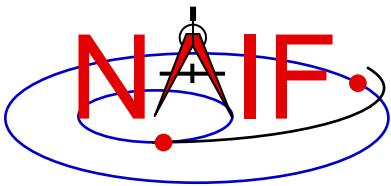


# Toolkit Contents

---

Navigation and Ancillary Information Facility

- **Software**
  - Subroutine libraries, with source code
    - » SPICELIB (Fortran)
    - » CSPICE (C)
    - » Icy (C)
    - » Mice (C and Matlab scripts)
  - Executable programs
    - » Application and utility programs
    - » A few example programs (called “cookbook” programs)
  - Installation/build scripts (normally you do NOT need to use these)
- **Documentation**
  - Available in plain text and HTML
- **Example Data**
  - Sample kernel files (supplied only for use with cookbook example programs, **not valid for general use**).

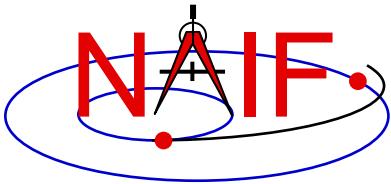


# Toolkit Characteristics

---

Navigation and Ancillary Information Facility

- **Computations are identical in all languages.**
- **For a given computer and operating system, all Toolkits use identical kernel files.**
  - Refer to the “Porting Kernels” tutorial for information about using kernels received from a machine different from what you are using.
- **Code is well tested before being released to users.**
- **New Toolkits are always backwards compatible.**
  - An application that worked when linked against an older Toolkit will link and work, without need for changes, using a new Toolkit.
  - Past functionality is never changed or removed, except that:
    - » enhancements of existing routines are allowed.
    - » NAIF reserves the right to fix bugs.
- **Extensive user-oriented documentation is provided.**
  - Includes highly documented source code.



# Toolkit Versions

---

Navigation and Ancillary Information Facility

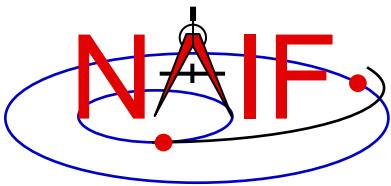
- **Toolkit Version**
  - SPICE Toolkits have an associated Version number
    - » Example: “N0066” (also written as “N66”)
  - The version number applies to all language implementations for all supported platforms.
- **When does NAIF release a new SPICE Toolkit version?**
  - » Not according to a fixed schedule
  - » Primarily driven by availability of significant new capabilities
    - For example, addition of the digital shape kernel subsystem (DSK)
  - » On rare occasion a Toolkit update is released to fix bugs, improve documentation, or satisfy an urgent request from a flight project.



# Toolkit Library Overview

Navigation and Ancillary Information Facility

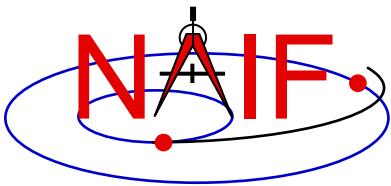
- **Toolkit libraries contain a broad set of capabilities related to the computations needed for determining “observation geometry” and time conversions.**
  - Examples appear on the next several pages
- **Not all functionality is present in all four language versions of the Toolkit library.**
  - The Fortran (Toolkit) and C (CSPICE) Toolkits provide almost identical functionality.
  - The IDL (Icy) and Matlab (Mice) Toolkits duplicate most but not all of the functionality available in the C Toolkits.
    - » We add additional interfaces as time permits.



# Toolkit Library Capabilities - 1

Navigation and Ancillary Information Facility

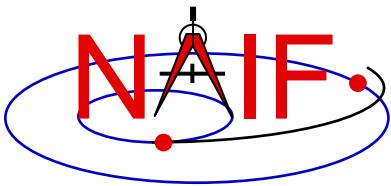
- **Kernel read access**
  - “Load” kernels
  - Get state or position vectors (SPK)
  - Get orientation of planets, natural satellites, etc. (PCK)
  - Get body shape parameters or physical constants (PCK)
  - Get orientation of spacecraft or spacecraft instruments or structures (CK, FK)
  - Get instrument parameters (e.g., FOV) (IK)
  - Get digital shape data (DSK)
  - Query binary EK files (EK-ESQ)
- **Kernel write access for binary kernels**
  - SPK writers
  - CK writers
  - PCK writers (only for binary PCK files)
  - DSK writers



# Toolkit Library Capabilities - 2

Navigation and Ancillary Information Facility

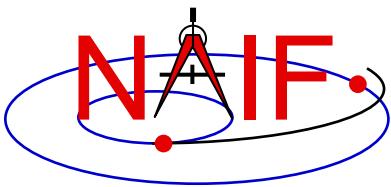
- **Additional ephemeris functions**
  - Classical osculating elements
  - Two-body Keplerian propagation
  - NORAD two line elements sets (TLE) propagation
  - Light time and Stellar aberration computation
- **Frame transformations**
  - Obtain 3x3 matrices for frame transformations of positions
  - Obtain 6x6 matrices for frame transformations of states
- **Time conversions**
  - Conversion between standard systems: TDB, TT (TDT), UTC
  - Conversion between SCLK and other systems
  - Parsing and formatting
- **Geometry finder calculations**
  - Find times or time spans when a specified geometric situation is true
  - Find times or time spans when a specified geometric parameter is within a given range, or is at a maximum or minimum



# Toolkit Library Capabilities - 3

Navigation and Ancillary Information Facility

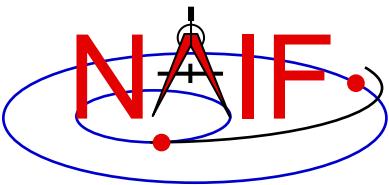
- **Math**
  - Vector/Matrix operations
  - Rotations, Euler angles, quaternions
  - Coordinate conversion (systems: latitudinal, cylindrical, rectangular, RA and DEC, spherical, geodetic, planetographic)
  - Geometry: ellipsoids, ellipses, planes
  - High-level functions: illumination angles, sub-observer point, sub-solar point, surface intercept point.
- **Constants**
  - Julian date of epoch J2000, SPD (seconds per day), PI, etc.
- **Strings**
  - Parsing: find tokens, words
  - Numeric conversion
  - Pattern matching
  - Replace marker, substring
  - Suffix, prefix
  - Case conversion
  - Find first/last non-blank character, first/last printing character



# Toolkit Library Capabilities - 4

Navigation and Ancillary Information Facility

- **Arrays**
  - Sorting, finding order vector, reordering
  - Searching: linear, binary
  - Insertion and deletion
- **Name/ID code conversion**
  - Bodies
  - Frames
  - Surfaces
- **I/O support**
  - Logical unit management (for Fortran toolkits)
  - Open, read, write text files
  - Kernel pool API
- **Exception handling**
  - Control exception handling behavior: mode, set message, assign output device.
- **Advanced data types**
  - Cells, Sets
  - Windows (sometimes called schedules)
  - Symbol Tables
  - Planes, Ellipses

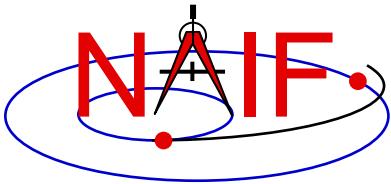


# Toolkit Directory Structure

---

Navigation and Ancillary Information Facility

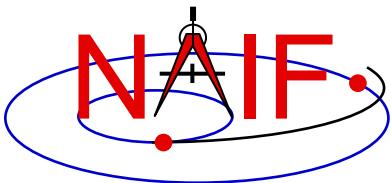
- **The directory structures for the four kinds of Toolkits are almost identical. However...**
  - The CSPICE, Icy and Mice Toolkits also have a directory for include files.
  - The names for application source code directories in CSPICE, Icy and Mice differ slightly from those in the Fortran toolkit.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



# Toolkit Directory Structure

Navigation and Ancillary Information Facility

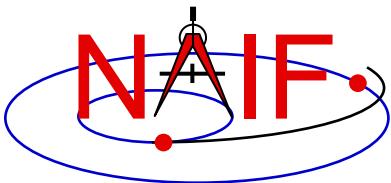
- The next level is comprised of:
  - data
    - » Cookbook example kernels (use **ONLY** for training with cookbook programs).
  - doc
    - » Text documents — \*.req, \*.ug, spicelib.idx/cspice.idx, whats.new, dscriptn.txt, version.txt.
    - » Subdirectory containing HTML documentation, called “html”.
      - 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 most Toolkits this directory is empty.
  - exe
    - » Executables for some SPICE application and utility programs:
      - brief, chronos, ckbrief, commnt, dskbrief, dskexp, frmdiff, inspekt, mkdsk, mkspk, msopck, spacit, spkdiff, spkmerge, tobin, toxfr, version.
    - » Executables for the several cookbook example programs:
      - simple, states, subpt, tictoc



# Toolkit Directory Structure

Navigation and Ancillary Information Facility

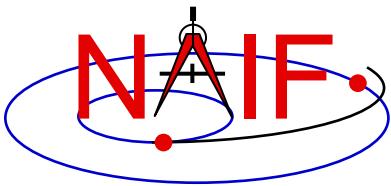
- **include** (applies only to CSPICE, Icy, and Mice)
  - » API header files.
    - File to include in callers of CSPICE is SpiceUsr.h
- **lib**
  - » Toolkit libraries:
    - For Fortran Toolkits
      - spicelib.a or spicelib.lib (public modules; use these)
      - support.a or support.lib (private modules; don't use these)
    - For C 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 or icy.dll (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 Application Programs

Navigation and Ancillary Information Facility

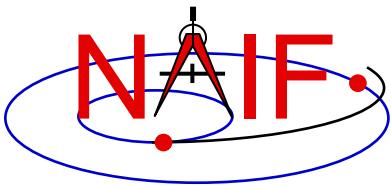
- **SPICE Toolkit application programs are available to:**
  - create most binary kernel types
  - compare or analyze certain kernel types
  - do various kinds of time conversions
  - and more...
  - See the `toolkit_apps` tutorial for details
- **Some additional application programs are available only from the NAIF website:**
  - <http://naif.jpl.nasa.gov/naif/utilities.html>
  - See the `non_toolkit_apps` tutorial for details



# Toolkit Utility Programs

Navigation and Ancillary Information Facility

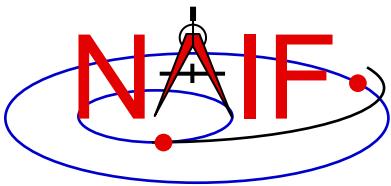
- **SPICE Toolkit utility programs are available to:**
  - add comments to binary kernels
    - » commnt
  - read comments from binary kernels
    - » commnt, spacit
    - » inspekt (only for EK/ESQ files)
  - summarize coverage of binary kernels
    - » brief, ckbrief, dskbrief, spacit
  - merge or subset SPK files
    - » spkmerge
  - indicate current Toolkit version
    - » version
  - port binary SPICE kernels between incompatible systems (infrequently needed)
    - » tobin, toxfr, spacit
    - » bingo (available only from the NAIF webpage)
  - port text SPICE kernels between incompatible systems
    - » bingo (available only from the NAIF webpage)
- See the `toolkit_apps` tutorial for details



# Toolkit Documentation - 1

Navigation and Ancillary Information Facility

- 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
    - » “index.html” is the top level index... your starting point
- All Toolkits include the following kinds of documents
  - Module headers
    - » They act as primary functional specification: I/O, exceptions, particulars defining behavior of module
    - » They contain code examples
    - » A standard format is used for each routine or entry point
    - » Location of HTML Module Headers:
      - Use the “API Reference Guide” link from the top level index
    - » Location of 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

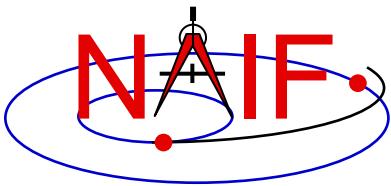


# Toolkit Documentation - 2

---

## Navigation and Ancillary Information Facility

- “Required Reading” documents
  - » Extensive technical references for principal subsystems
    - Provide many low-level details
    - Provide code examples
  - » HTML versions are accessible using the “Required Reading Documents” link from the top level index.
  - » Plain text versions are located under “doc” and have extension “.req”
  - » Not all Required Readings were adapted for all languages
    - Some of the Required Reading documents provided with CSPICE are based upon Fortran SPICE
    - Some of the Required Readings for Icy or Mice toolkits are based upon CSPICE
- User’s Guides
  - » Tell how to use the utility and application programs.
  - » HTML versions are accessible using the “User’s Guide Documents” link from the top level index.
  - » Plain text versions are located under “doc” and have extension “.ug.”



# Toolkit Documentation - 3

---

Navigation and Ancillary Information Facility

- Other documents

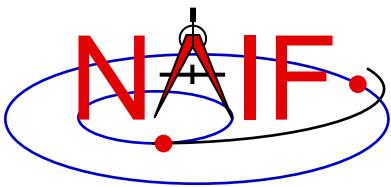
- Permutated Index

- » Maps phrases describing functionality to corresponding module names and file names
    - » Shows names of all entry points in Fortran toolkit APIs
    - » HTML version is accessible using the “Permutated Index” link from the top level index.
    - » 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

- Toolkit Description

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

continues on next page

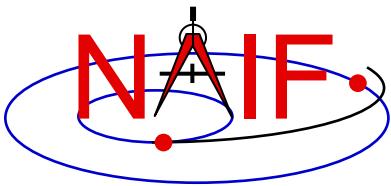


# Toolkit Documentation - 4

---

## Navigation and Ancillary Information Facility

- Introduction to SPICE
  - » HTML document containing a brief introduction to the Toolkit and SPICE system; accessible using the “Introduction to the SPICE System” link from the top level index.
- What’s New in SPICE
  - » Describes new features and bug fixes in each Toolkit release, covering the last 20 years.
  - » Plain text version is “doc/whats.new”.
  - » HTML version is accessible 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
- You can also use Google to view any of the Toolkit documents
  - E.g., search on the API name (e.g. spkezr, spkezr\_c, or cspice\_spkezr (for Icy and Mice))
  - E.g. search on the document name (e.g. CK Required Reading, spkdiff user’s guide)

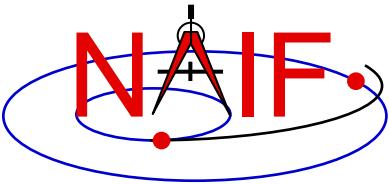


---

Navigation and Ancillary Information Facility

## Backup

## Supported Environments



# Supported Environments

---

Navigation and Ancillary Information Facility

- NAIF ports the SPICE Toolkit to many popular environments.
  - Each environment is characterized by
    - » Language
    - » Hardware type (platform)
    - » Operating System
    - » Compiler
    - » Selected compilation options
- NAIF provides SPICE Toolkit packages for each supported environment.
  - If you cannot find a package built for the environment of interest to you, contact NAIF.
- The list of supported environments slowly evolves.
  - Old ones no longer supportable are terminated.
  - New ones are added based on user interest and available NAIF resources.



# Supported Environments - Fortran

## Navigation and Ancillary Information Facility

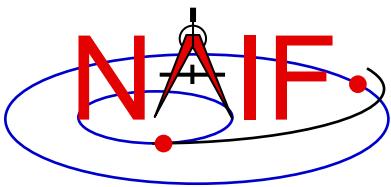
Product Name	Operating System	Compiler
Mac/Intel, OS-X, Intel FORTRAN, 32bit	OS X 10.11	Intel Fortran 15.0
Mac/Intel, OS-X, Intel FORTRAN, 64bit	OS X 10.11	Intel Fortran 15.0
Mac/Intel, OS-X, gfortran, 32bit	OS X 10.11	gfortran 5.3
Mac/Intel, OS-X, gfortran, 64bit	OS X 10.11	gfortran 5.3
PC, CYGWIN, gfortran, 32bit	Windows/Cygwin 2.5	gfortran 5.4
PC, CYGWIN, gfortran, 64bit	Windows/Cygwin 2.7	gfortran 5.4
PC, Linux, Intel FORTRAN, 32bit	Red Hat Linux (RHE5)	Intel Fortran 10.0
PC, Linux, Intel FORTRAN, 64bit	Red Hat Linux (RHE5)	Intel Fortran 10.0
PC, Linux, g77, 32bit	Red Hat Linux (RHE5)	g77 3.4
PC, Linux, gfortran, 32bit	Red Hat Linux (RHE5)	gfortran 4.3
PC, Linux, gfortran, 64bit	Red Hat Linux (RHE5)	gfortran 4.3
PC, Windows, Intel FORTRAN, 32bit	Windows 7 and above	Intel Fortran 11.1
PC, Windows, Intel FORTRAN, 64bit	Windows 7 and above	Intel Fortran 11.1
Sun/Intel, Solaris, SUN FORTRAN, 32bit	Solaris 10	Sun FORTRAN 95 12.1
Sun/Intel, Solaris, SUN FORTRAN, 64bit	Solaris 10	Sun FORTRAN 95 12.1
Sun/SPARC, Solaris, SUN FORTRAN, 32bit	Solaris 9	Sun FORTRAN 95 8.2



# Supported Environments - C

Navigation and Ancillary Information Facility

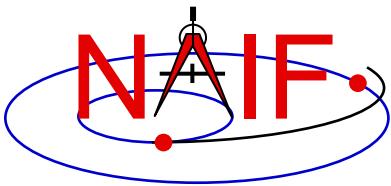
Product Name	Operating System	Compiler
Mac/Intel, OS-X, Apple C, 32bit	OS X 10.11	Apple C 8.0
Mac/Intel, OS-X, Apple C, 64bit	OS X 10.11	Apple C 8.0
PC, CYGWIN, gCC, 32bit	Windows/Cygwin 2.5	gcc 5.4
PC, CYGWIN, gCC, 64bit	Windows/Cygwin 2.7	gcc 5.4
PC, Linux, gCC, 32bit	Red Hat Linux (RHE5)	gcc 4.3
PC, Linux, gCC, 64bit	Red Hat Linux (RHE5)	gcc 4.3
PC, Windows, Microsoft Visual C, 32bit	Windows 7 and above	MS Visual Studio 15.0 C
PC, Windows, Microsoft Visual C, 64bit	Windows 7 and above	MS Visual Studio 15.0 C
Sun/Intel, Solaris, SunC, 32bit	Solaris 10	Sun C 12.1
Sun/Intel, Solaris, SunC, 64bit	Solaris 10	Sun C 12.1
Sun/SPARC, Solaris, gCC, 32bit	Solaris 9	gcc 3.4
Sun/SPARC, Solaris, gCC, 64bit	Solaris 9	gcc 3.4
Sun/SPARC, Solaris, SUN C, 32bit	Solaris 9	Sun C 5.8
Sun/SPARC, Solaris, SUN C, 64bit	Solaris 10	Sun C 5.13



# Supported Environments - IDL

Navigation and Ancillary Information Facility

Product Name	Operating System	Compiler, IDL
Mac/Intel, OS-X, Apple C/IDL, 64bit	OS X 10.11	Apple C 8.0, IDL 8.3
PC, Linux, gcc/IDL, 32bit	Red Hat Linux (RHE5)	gcc 4.3, IDL 8.1
PC, Linux, gcc/IDL, 64bit	Red Hat Linux (RHE5)	gcc 4.3, IDL 8.1
PC, Windows, Microsoft Visual C/IDL, 32bit	Windows 7 and above	MS Visual Studio 15.0 C, IDL 8.1
PC, Windows, Microsoft Visual C/IDL, 64bit	Windows 7 and above	MS Visual Studio 15.0 C, IDL 8.1
Sun/Intel, Solaris, SUN C/IDL, 64bit	Solaris 10	Sun C 12.1, IDL 8.3
Sun/SPARC, Solaris, gcc/IDL, 32bit	Solaris 9	gcc 3.4, IDL 7.1
Sun/SPARC, Solaris, gcc/IDL, 64bit	Solaris 9	gcc 3.4, IDL 7.1
Sun/SPARC, Solaris, SUN C/IDL, 32bit	Solaris 9	Sun C 5.8, IDL 7.1



# Supported Environments - MATLAB

Navigation and Ancillary Information Facility

Product Name	Operating System	Compiler, MATLAB
Mac/Intel, OS-X, Apple C, 64bit	OS X 10.11	Apple C 8.0.0, MATLAB R2015a
PC, Linux, gCC, 64bit	Red Hat Linux (RHE5)	gcc 4.3, MATLAB R2010a
PC, Windows, Microsoft Visual C/Matlab, 64bit	Windows 7 and above	MS Visual Studio 15.0 C, MATLAB R2014b
Sun/SPARC, Solaris, SUN C/Matlab, 64bit	Solaris 10	Sun C 5.13, MATLAB R2009b

**The N66 version of Mice will run on Matlab R2016a or later as long as the Mice codebase has not been recompiled. If it has been recompiled, Mice will not work.**

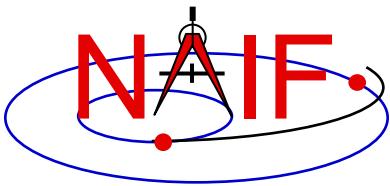


# Status for Other Environments

---

Navigation and Ancillary Information Facility

- **NAIF is unable to support environments other than those listed on the previous set of charts.**
  - Except an alpha-test version of a Java Native Interface Toolkit is available upon request.
- **The SPICE and CSPICE packages should function as expected on platforms running any Linux OS (Ubuntu, Fedora, etc.), BSD OS (OpenBSD, FreeBSD, etc.), or a Linux based OS environment (minGW) using a standard GCC tool-chain (gfortran or gcc compiler).**
  - Version 4.2 or later for gfortran; 4.0 or later for gcc
- **The Mice package has been successfully built against the octave environment (version > 3.4) on Linux and OS X. Contact NAIF if you have questions concerning use with Octave.**
- **Apple OS9 and earlier computers are no longer supported.**

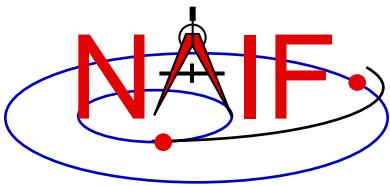


---

Navigation and Ancillary Information Facility

# “Icy” The IDL<sup>©</sup> Interface to CSPICE

January 2020

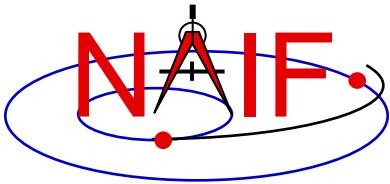


# Topics

---

Navigation and Ancillary Information Facility

- **Icy Benefits**
- **How does it work?**
- **Distribution**
- **Icy Operation**
- **Vectorization**
- **Simple Icy Example**

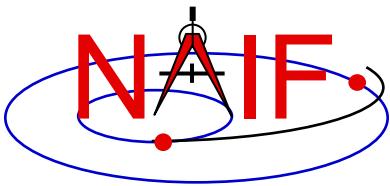


# Icy Benefits

---

Navigation and Ancillary Information Facility

- Ease of use: Icy operates as an extension to the IDL language regime.
- Icy supports more than four-hundred CSPICE routines.
- Icy calls usually correspond to the call format of the underlying CSPICE routine, returning IDL native data types.
- Icy has some capability not available in CSPICE such as vectorization.
- CSPICE error messages return to IDL in a form usable by the *catch* error handler construct.

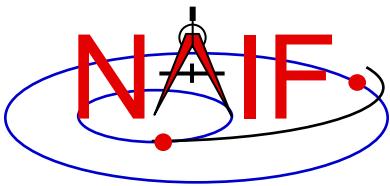


# How Does It Work? (1)

---

Navigation and Ancillary Information Facility

- The IDL environment includes an intrinsic capability to use external routines.
  - Icy functions as an IDL Dynamically Loadable Module (DLM). A DLM consists of a shared object library (icy.so/.dll) and a DLM text definition file (icy.dlm).
    - » The shared library contains a set of IDL callable C interface routines that wrap a subset of CSPICE wrapper calls.
    - » The text definition file lists the routines within the shared library and the format for the routine's call parameters.



# How Does It Work? (2)

Navigation and Ancillary Information Facility

- **Using Icy from IDL requires you register the Icy DLM with IDL to access the interface routines. Several means exist to do so.**
  - On Unix/Linux, start IDL from the directory containing `icy.dlm` and `icy.so`
  - **From the IDL interpreter (or from a command script), execute the `dlm_register` command:** IDL> `dlm_register,'path_to_directory_containing_icy.dlm'`
    - » Examples (Unix and Windows):
      - » IDL> `dlm_register, '/naif/icy/lib/icy.dlm'`
      - » IDL> `dlm_register, 'c:\naif\icy\lib\icy.dlm'`
  - **Copy `icy.dlm` and `icy.so` or `icy.dll` to IDL's binary directory:**

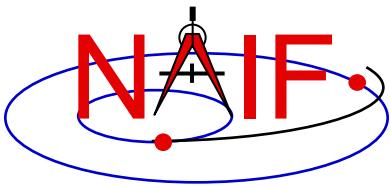
{The IDL install directory}/bin/bin.*user\_architecture*

    - » Examples (Unix and Windows):
      - » `cp icy.dlm icy.so /Applications/exelis/idl/bin/bin.darwin.x86_64/`
      - » `cp icy.dlm icy.dll C:\Program Files\Exelis\idl83\bin\bin.x86_64\`
  - **Append to the `IDL_DLM_PATH` environment variable the directory name containing `icy.dlm` and `icy.so` or `icy.dll`:**

```
setenv IDL_DLM_PATH "<IDL_DEFAULT>:path_to_directory_containing_icy.dlm"
```

**Warning: with regards to the Icy source directory, `icy/src/icy`, do not invoke IDL from the directory, do not register the directory, and do not append `IDL_DLM_PATH` to the directory. This directory contains an “`icy.dlm`” but no “`icy.so`.”**

continued on next page



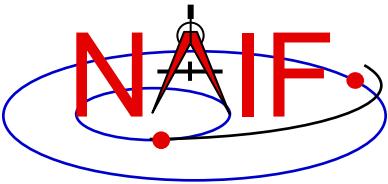
# How Does It Work? (3)

---

Navigation and Ancillary Information Facility

**When a user invokes a call to a DLM routine:**

- 1. IDL calls...**
  - 2. the interface routine in the shared object library, linked against...**
  - 3. CSPICE, which performs its function and returns the result...**
  - 4. to IDL...**
- ... transparent from the user's perspective.**

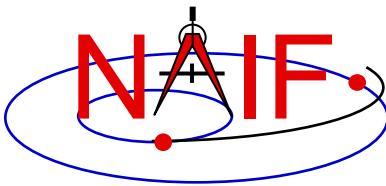


# Icy Distribution

---

Navigation and Ancillary Information Facility

- NAIF distributes the Icy package as an independent product analogous to SPICELIB and CSPICE.
- The package includes:
  - the CSPICE source files
  - the Icy interface source code
  - platform specific build scripts for Icy and CSPICE
  - IDL versions of the SPICE cookbook programs, *states*, *tictoc*, *subpt*, and *simple*
  - an HTML based help system for both Icy and CSPICE, with the Icy help cross-linked to CSPICE
  - the Icy shared library and DLM file. The system is ready for use after installation of these files
- You do not need a C compiler to use Icy.



# Icy Operation (1)

---

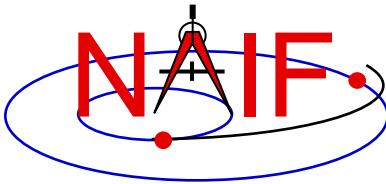
Navigation and Ancillary Information Facility

- A user may occasionally encounter an IDL math exception:

```
% Program caused arithmetic error: Floating underflow
```

- This warning occurs most often as a consequence of CSPICE math operations.
- In all known cases, the SIGFPE exceptions caused by CSPICE can be ignored. CSPICE assumes numeric underflow as zero.
  - A user can adjust IDL's response to math exceptions by setting the !EXCEPT variable:
    - » !EXCEPT = 0 suppresses the SIGFPE messages, and even more (e.g. a fatal error).
    - » !EXCEPT = 1, the default, reports math exceptions on return to the interactive prompt.
      - NAIF recommends this be used.
    - » !EXCEPT = 2 reports exceptions immediately after executing the command.

continued on next page



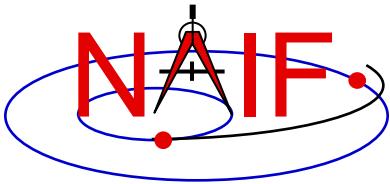
# Icy Operation (2)

---

Navigation and Ancillary Information Facility

- A possible irritant exists in loading kernels using the `cspice_furnsh` function.
  - Kernels are loaded into your IDL session, not into your IDL scripts. This means:
    - » loaded binary kernels remain accessible (“active”) throughout your IDL session
    - » data from loaded text kernels remain in the kernel pool (in the IDL memory space) throughout your IDL session
  - Consequence: some kernel data may be available to one of your scripts even though not intended to be so.
    - » You could get **incorrect results!**
    - » If you run only one script during your IDL session, there's no problem.

continued on next page

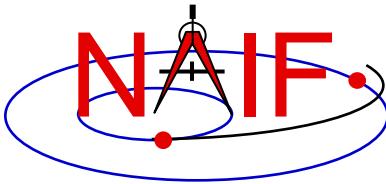


# Icy Operation (3)

---

Navigation and Ancillary Information Facility

- **Mitigation: two approaches**
  - Load all needed SPICE kernels for your IDL session at the beginning of the session, paying careful attention to the files loaded and the loading order (loading order affects precedence)
    - » Convince yourself that this approach will provide ALL of the scripts you will run during this IDL session with the appropriate SPICE data
  - At or near the end of every IDL script:
    - » include a call to `cspice_unload` for each kernel loaded using `cspice_furnsh`
    - » or include a call to `cspice_kclear` to remove ALL kernel data from the kernel pool loaded using `cspice_furnsh`



# Icy Vectorization (1)

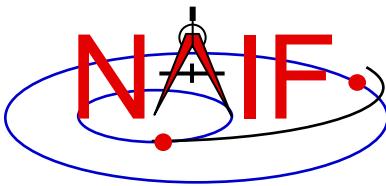
---

Navigation and Ancillary Information Facility

- Several common Icy functions include use of vectorized arguments, a capability not available in C or FORTRAN toolkits.
  - Note: IDL indexes arrays using a base value of zero as opposed to FORTRAN, which uses a base value of one.
    - » Example: access the first element of an IDL 1xN array using array[0], the second element using array[1], etc.
- Example: use Icy to retrieve state vectors and light-time values for 1000 ephemeris times.
  - Create an array of 1000 ephemeris times with step size of 10 hours, starting from July 1, 2005.

```
cspice_str2et, 'July 1, 2005', start  
et = dindgen( 1000 )*36000.d + start
```

continued on next page



# Icy Vectorization (2)

---

Navigation and Ancillary Information Facility

- Retrieve the state vectors and corresponding light times from Mars to earth at each `et`, in the J2000 frame, using LT+S aberration correction:

```
cspice_spkezr, 'Earth', et, 'J2000', 'LT+S', 'MARS', state, ltime
```

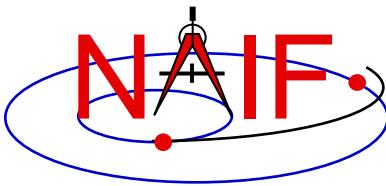
- Access the *i*th state 6-vector corresponding to the *i*th ephemeris time with the expression

```
state_i = state[*,i]
```

- Convert the ephemeris time vector `et` from the previous example to UTC calendar strings with three decimal places accuracy.

```
format = 'C'  
prec   = 3  
cspice_et2utc, et, format, prec, utcstr
```

continued on next page



# Icy Vectorization (3)

---

Navigation and Ancillary Information Facility

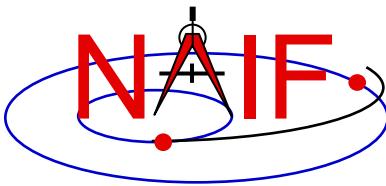
- The call returns `utcstr`, an array of 1000 strings each *i*th string the calendar date corresponding to `et[i]`. Access the *i*th string of `utcstr` corresponding to the *i*th ephemeris time with the expression

```
utcstr_i = utcstr[i]
```

- Convert the position components of the N state vectors to latitudinal coordinates (the first three components of a state vector - IDL uses a zero based vector index).

```
cspice_reclat, state[0:2,*], radius, latitude, longitude
```

- The call returns three double precision variables of type Array[1000] (vectorized scalars): `radius`, `latitude`, `longitude`.



# Simple Icy Example

Navigation and Ancillary Information Facility

- As an example of using Icy with vectorization, calculate and plot, in the J2000 inertial frame, the trajectory of the Cassini spacecraft from June 20 2004 to December 1 2005.

```
;; Construct a meta kernel, "standard.tm", which will be used to load the needed
;; generic kernels: "naif0011.tls," "de421.bsp," and "pck00010.tpc."

;; Load the generic kernels using the meta kernel, and a Cassini spk.

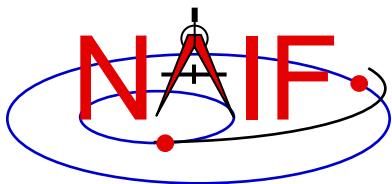
cspice_furnsh, ['standard.tm', '/kernels/cassini/spk/030201AP SK SM546 T45.bsp' ]

;; Define the number of divisions of the time interval and the time interval.
STEP = 10000
cspice_str2et, [ 'Jun 20, 2004', 'Dec 1, 2005' ] , et
times = dindgen(STEP)*(et[1]-et[0])/STEP + et[0]

cspice_spkpos, 'Cassini', times, 'J2000', 'NONE', 'SATURN BARYCENTER', pos, ltime

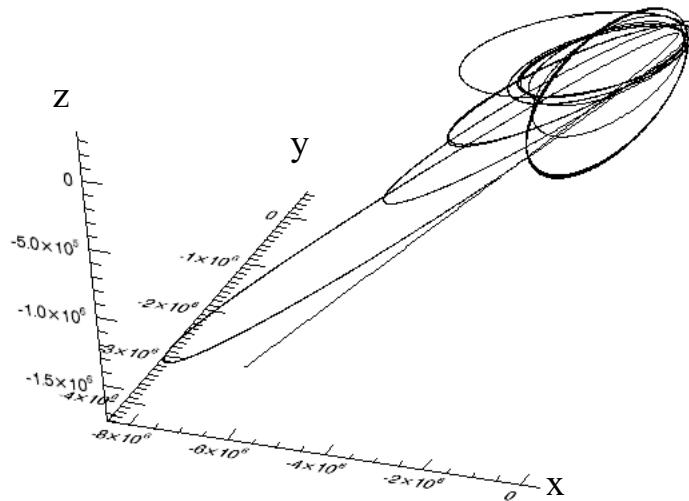
;; Plot the resulting trajectory.
x = pos[0,*]
y = pos[1,*]
z = pos[2,*]
iplot, x, y, z

cspice_kclear
```

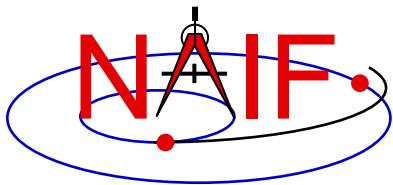


# Icy Example Graphic Output

Navigation and Ancillary Information Facility



Trajectory of the Cassini spacecraft, in the J2000 frame, from June 20 2004 to Dec 1 2005

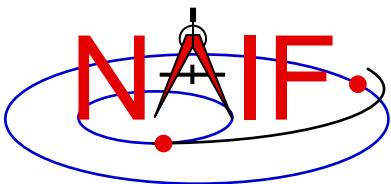


---

Navigation and Ancillary Information Facility

# “Mice” The MATLAB<sup>©</sup> Interface to CSPICE

January 2020

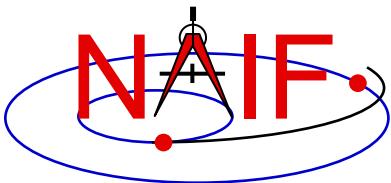


# Topics

---

Navigation and Ancillary Information Facility

- **Mice Benefits**
- **How does it work?**
- **Distribution**
- **Mice Operation**
- **Vectorization**
- **Simple Mice Examples**

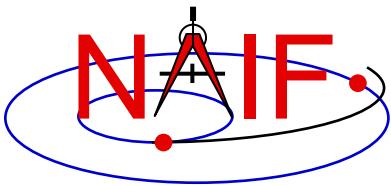


# Mice Benefits

---

Navigation and Ancillary Information Facility

- Mice operates as an extension to the MATLAB environment.
- All Mice calls are functions regardless of the call format of the underlying CSPICE routine, returning MATLAB native data types.
- Mice has some capability not available in CSPICE such as vectorization.
- CSPICE error messages return to MATLAB in the form usable by the *try...catch* construct.

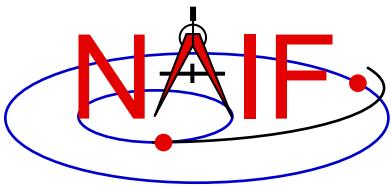


# How Does It Work? (1)

Navigation and Ancillary Information Facility

- The MATLAB environment includes an intrinsic capability to use external routines.
  - Mice functions as a MATLAB executable, MEX, consisting of the Mice MEX shared object library and a set of .m wrapper files.
    - » The Mice library contains the MATLAB callable C interface routines that wrap a subset of CSPICE wrapper calls.
    - » The wrapper files, named `cspice_*.m` and `mice_*.m`, provide the MATLAB calls to the interface functions.
      - » A function prefixed with ‘`cspice_`’ retains essentially the same argument list as the CSPICE counterpart.
      - » An interface prefixed with ‘`mice_`’ returns a structure, with the fields of the structure corresponding to the output arguments of the CSPICE counterpart.
    - » The wrappers include a header section describing the function call, displayable by the MATLAB `help` command.

continued on next page



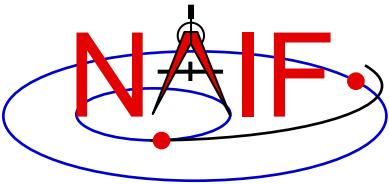
# How Does It Work? (2)

---

Navigation and Ancillary Information Facility

**When a user invokes a call to a Mice function:**

- 1. MATLAB calls...**
  - 2. the function's wrapper, which calls...**
  - 3. the Mice MEX shared object library, which performs its function then returns the result...**
  - 4. to the wrapper, which...**
  - 5. returns the result to the user**
- ... transparent from the user's perspective.**

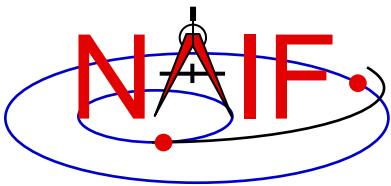


# Mice Distribution

---

Navigation and Ancillary Information Facility

- NAIF distributes Mice as a complete, standalone package.
- The package includes:
  - the CSPICE source files
  - the Mice interface source code
  - platform specific build scripts for Mice and CSPICE
  - MATLAB versions of the SPICE cookbook programs, *states*, *tictoc*, *subpt*, and *simple*
  - an HTML-based help system for both Mice and CSPICE, with the Mice help cross-linked to CSPICE
  - the Mice MEX shared library and the M wrapper files. The system is ready for use after installation of the library and wrapper files.
- You do not need a C compiler to use Mice.



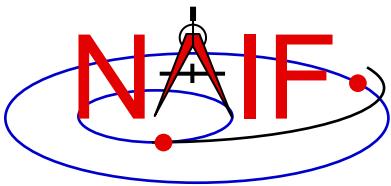
# Mice Operation (1)

---

Navigation and Ancillary Information Facility

- A possible irritant exists in loading kernels using the `cspice_furnsh` function.
  - Kernels load into your MATLAB session, not into your MATLAB scripts. This means:
    - » loaded binary kernels remain accessible (“active”) throughout your MATLAB session
    - » data from loaded text kernels remain in the kernel pool (in the memory space used by CSPICE) throughout your MATLAB session
  - Consequence: some kernel data may be available to one of your scripts even though not intended to be so.
    - » You could get **incorrect results!**
    - » If you run only one script during your MATLAB session, there’s no problem.

continued on next page

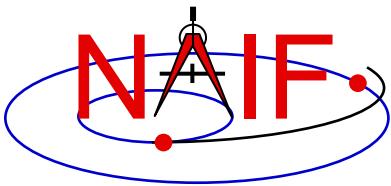


# Mice Operation (2)

---

Navigation and Ancillary Information Facility

- Mitigation: two approaches
  - Load all needed SPICE kernels for your **MATLAB session** at the beginning of the session, paying careful attention to the files loaded and the loading order (loading order affects precedence)
    - » Convince yourself that this approach will provide **ALL** of the scripts you will run during this MATLAB session with the appropriate SPICE data
  - At or near the end of every **MATLAB script**:
    - » include a call to `cspice_unload` for each kernel loaded using `cspice_furnsh`
    - » or include a call to `cspice_kclear` to remove **ALL** kernel data from the kernel pool loaded using `cspice_furnsh`



# Mice Vectorization (1)

Navigation and Ancillary Information Facility

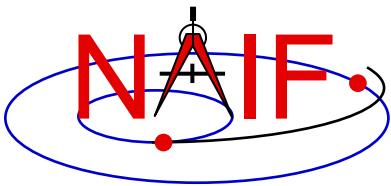
- **Most Mice functions include use of vectorized arguments, a capability not available in C or Fortran toolkits.**
- **Example: use Mice to retrieve state vectors and light-time values for 1000 ephemeris times.**
  - **Create an array of 1000 ephemeris times with a step size of 10 hours, starting from July 1, 2005:**

```
start = cspice_str2et('July 1 2005');  
et      = (0:999)*36000 + start;
```

- **Retrieve the state vectors and corresponding light times from Mars to earth at each et in the J2000 frame with LT+S aberration correction:**

```
[state, ltime] = cspice_spkezr( 'Earth', et, 'J2000', 'LT+S', 'MARS');  
or  
starg = mice_spkezr( 'Earth', et, 'J2000', 'LT+S', 'MARS');
```

[continued on next page](#)



# Mice Vectorization (2)

Navigation and Ancillary Information Facility

- Access the *i*th state 6-vector (6x1 array) corresponding to the *i*th ephemeris time with the expression

```
state_i = state(:,i)
```

or

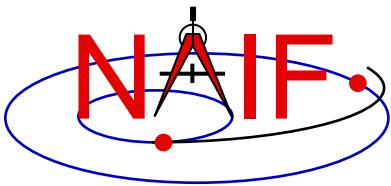
```
state_i = starg(i).state
```

- Convert the ephemeris time vector `et` from the previous example to UTC calendar strings with three decimal places of precision in the seconds field.

```
format = 'C';
prec   = 3;
utcstr = cspice_et2utc( et, format, prec );
```

- The call returns `utcstr`, an array of 1000 strings (dimensioned 1000x24), where each *i*th string is the calendar date corresponding to `et(i)`.

[continued on next page](#)



# Mice Vectorization (3)

Navigation and Ancillary Information Facility

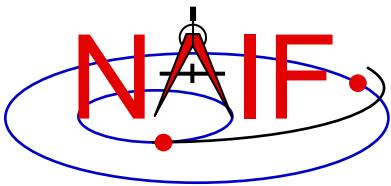
- Access the *i*th string of utcstr corresponding to the *i*th ephemeris time with the expression

```
utcstr_i = utcstr(i,:)
```

- Convert the position components (the first three components in a state vector) of the N state vectors returned in state by the cspice\_spkezr function to latitudinal coordinates.

```
[radius, latitude, longitude] = cspice_reclat( state(1:3,:) );
```

- The call returns three double precision 1x1000 arrays (vectorized scalars): radius, latitude, longitude.



# Simple Mice Example (1)

Navigation and Ancillary Information Facility

- As an example of using Mice, calculate and plot the trajectory of the Cassini spacecraft, in the J2000 inertial frame, from June 20 2004 to December 1 2005. This example uses the `cspice_spkpos` function to retrieve position data.

```
% Construct a meta kernel, "standard.tm", which will be used to load the needed
% generic kernels: "naif0011.tls," "de421.bsp," and "pck00010.tpc."

% Load the generic kernels using the meta kernel, and a Cassini spk.

cspice_furnsh( { 'standard.tm', '/kernels/cassini/spk/030201AP SK SM546 T45.bsp' } )

% Define the number of divisions of the time interval.
STEP      = 1000;
et        = cspice_str2et( {'Jun 20, 2004', 'Dec 1, 2005'} );
times     = (0:STEP-1) * ( et(2) - et(1) )/STEP + et(1);

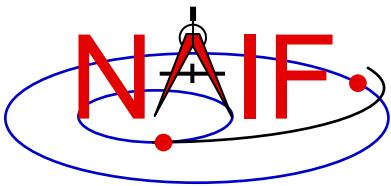
[pos,ltime]= cspice_spkpos( 'Cassini', times, 'J2000', 'NONE', 'SATURN BARYCENTER' );

% Plot the resulting trajectory.
x = pos(1,:);
y = pos(2,:);
z = pos(3,:);

plot3(x,y,z)

cspice_kclear
```

continued on next page

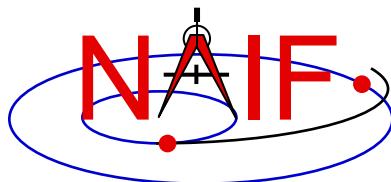


# Simple Mice Example (2)

Navigation and Ancillary Information Facility

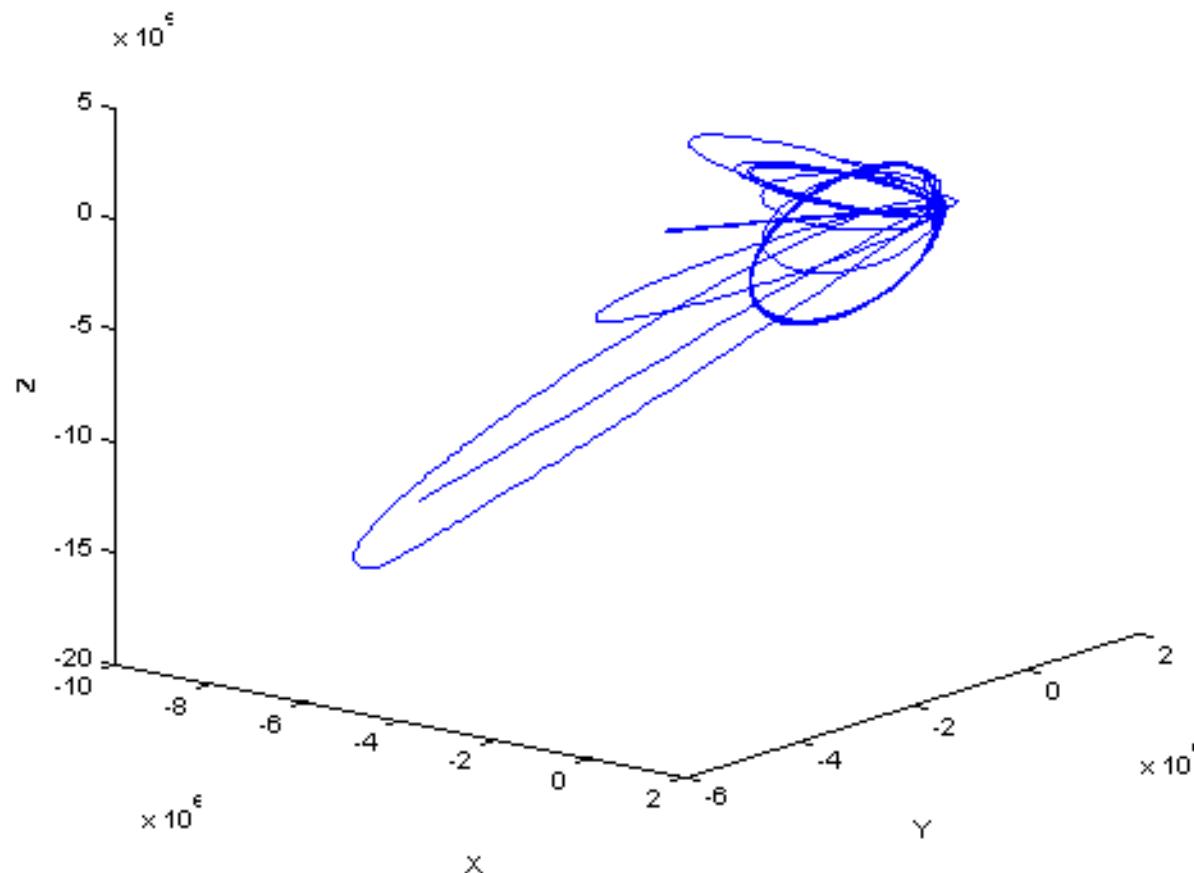
- Repeat the example of the previous page, except use the `mice_spkezr` function to retrieve full state vectors.

```
% Define the number of divisions of the time interval.  
STEP = 1000;  
  
% Construct a meta kernel, "standard.tm", which will be used to load the needed  
% generic kernels: "naif0009.tls," "de421.bsp," and "pck00009.tpc."  
  
% Load the generic kernels using the meta kernel, and a Cassini spk.  
  
cspice_furnsh( { 'standard.tm', '/kernels/cassini/spk/030201AP SK SM546 T45.bsp' } )  
  
et      = cspice_str2et( {'Jun 20, 2004', 'Dec 1, 2005'} );  
times = (0:STEP-1) * ( et(2) - et(1) )/STEP + et(1);  
  
ptarg = mice_spkpos( 'Cassini', times, 'J2000', 'NONE', 'SATURN BARYCENTER' );  
pos   = [ptarg.pos];  
  
% Plot the resulting trajectory.  
x = pos(1,:);  
y = pos(2,:);  
z = pos(3,:);  
  
plot3(x,y,z)  
  
cspice_kclear
```

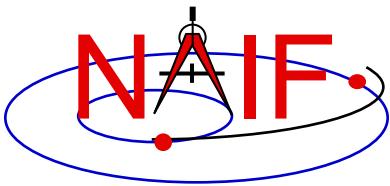


# Mice Example Graphic Output

Navigation and Ancillary Information Facility



Trajectory of the Cassini spacecraft, in the J2000 frame, from June 20 2004 to Dec 1 2005



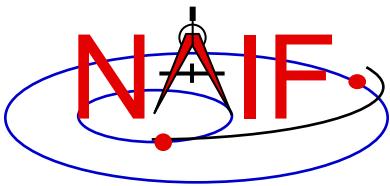
---

Navigation and Ancillary Information Facility

# Preparing for Programming Using the SPICE Toolkits

This tutorial contains important requirements and recommendations.

January 2020



# Setting Path to Toolkit Executables (1)

Navigation and Ancillary Information Facility

## Recommended for all Toolkits

- Unix (OS X, Linux, BSD, execute the command echo \$SHELL to determine your shell name)

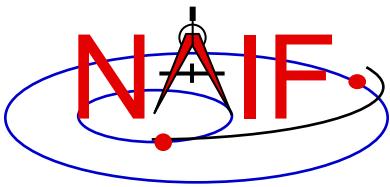
- csh, tcsh: Use the set command to add the location of toolkit executables to your path.

```
» set path = ($path /my_directory/toolkit/exe)
» set path = ($path /my_directory/cspice/exe)
» set path = ($path /my_directory/icy/exe)
» set path = ($path /my_directory/mice/exe)
```

- sh, bash, zsh, dash, ksh: Assign the \$PATH environment variable.

```
» PATH=$PATH:/my_directory/toolkit/exe
» PATH=$PATH:/my_directory/cspice/exe
» PATH=$PATH:/my_directory/icy/exe
» PATH=$PATH:/my_directory/mice/exe
```

Replace `my_directory` with the path in which you installed the toolkit on your computer.



# Setting Path to Toolkit Executables (2)

Navigation and Ancillary Information Facility

## Recommended for all Toolkits

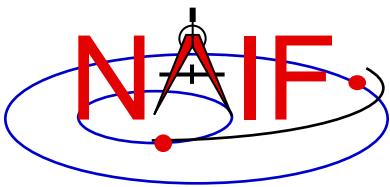
- **Windows**

- DOS shell: Use the set command to add the location of toolkit executables to your path. Use setx for a persistent setting.

```
» set PATH=drive:\my_directory\toolkit\exe;%PATH%
» set PATH=drive:\my_directory\cspice\exe;%PATH%
» set PATH=drive:\my_directory\icy\exe;%PATH%
» set PATH=drive:\my_directory\mice\exe;%PATH%
```

- Or edit the environment variable PATH from the Advanced pane on the System Control Panel (Control Panel->System->Advanced).

Replace `drive:\my_directory` with the path in which you installed the toolkit on your computer.



# Unix: Build a SPICE Executable

Navigation and Ancillary Information Facility

- Assume your Toolkit distribution is installed at:

`/naif/cspice/` for CSPICE (C toolkits)

`/naif/toolkit/` for SPICE (Fortran toolkits)

- Compile and link an application—let's pretend it's named ***program***—against the CSPICE or SPICELIB library.

- For C:

```
$ gcc program.c -I/naif/cspice/include /naif/cspice/lib/cspice.a -lm
```

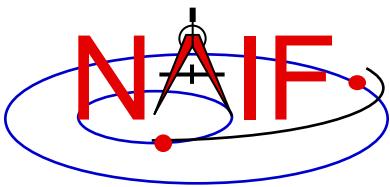
- For FORTRAN:

```
$ gfortran program.f /naif/toolkit/spicelib.a
```

- The default SPICE library names do not conform to the UNIX convention `libname.a`. So you cannot use the conventional library path/name options `-L` and `-l`, e.g.

```
$ gcc ... -L/path_to_libs/ -lname
```

unless you rename the SPICE library.



# Windows: C compiler settings

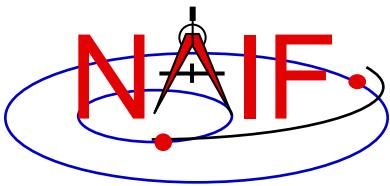
Navigation and Ancillary Information Facility

- The standard installation of Microsoft Visual Studio may not update environment variables needed to use the C compiler (cl) from the standard DOS shell. This depends on your version of the Microsoft development environment.
  - If programming in an XP 32-bit environment, you can set the environment variables by executing from a DOS shell one of the “vars32” batch scripts supplied with Microsoft compilers:
    - » `vars32.bat`, `vcvars32.bat`, `vsvars32.bat`
  - Recent versions of Visual Studio include scripts to spawn a DOS shell with the needed environment. The scripts exist under *Visual Studio “version”* found in the *Programs* menu:

*Programs* → *Visual Studio “version”*

The scripts' names for a 64bit (x64) environment or a 32bit (x86) are:

- » **VS*version* x64 Native Tools Command Prompt**
  - Example, *VS2015 x64 Native Tools Command Prompt*
- » **VS*version* x86 Native Tools Command Prompt**
  - Example, *VS2015 x86 Native Tools Command Prompt*



# Windows: ifort compiler settings

Navigation and Ancillary Information Facility

- The standard installation of Intel ifort may not update environment variables needed to use the Fortran compiler (ifort) from the standard DOS shell.
  - Intel provides batch scripts to spawn DOS shells properly configured for 32-bit or 64-bit Fortran development. Find the scripts by navigating to the menu

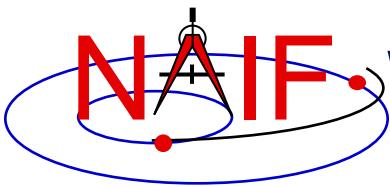
*Programs -> Intel Software Development Tools -> Intel Visual Fortran Compiler (version)*

The script for a 32-bit ifort environment is:

Fortran Build Environment for applications running on IA-32

The script for a 64-bit ifort environment is:

Fortran Build Environment for applications running on Intel 64



# Windows: Build a SPICE Executable

Navigation and Ancillary Information Facility

- Assume the SPICE distribution is installed at:

`C:\naif\cspice\` for C toolkits

`C:\naif\toolkit\` for Fortran toolkits

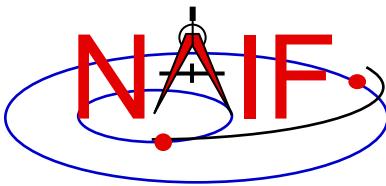
- Compile and link an application, say *program*, against the CSPICE or SPICELIB library.

- For C toolkits:

```
> cl program.c -IC:\naif\cspice\include C:\naif\cspice\lib\cspice.lib
```

- For FORTRAN toolkits:

```
> ifort program.f C:\naif\toolkit\lib\SPICELIB.LIB
```



# Icy: Register the Icy DLM to IDL (1)

Navigation and Ancillary Information Facility  
Required for “Icy”

- **Unix and Windows**

- Use the IDL register command:

```
IDL> dlm_register, '_path_to_directory_containing_icy.dlm_'
```

e.g.

```
IDL > dlm_register, '/naif/icy/lib/icy.dlm'
```

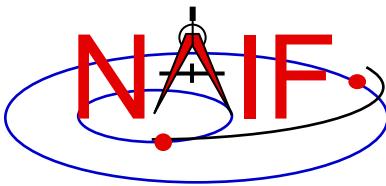
- Or, copy **icy.dlm** and **icy.so** (or **icy.dll**) to IDL's binary directory located at *{The IDL install directory}/bin/bin.user\_architecture*, e.g.
    - » For Unix, X86 architecture

```
cp icy.dlm icy.so /Applications/exelis/idl/bin/bin.darwin.x86_64/
```

» For Windows, X86 architecture

```
cp icy.dlm icy.dll C:\Program Files\Exelis\idl83\bin\bin.x86_64\
```

continued on next page



# Icy: Register the Icy DLM to IDL (2)

Navigation and Ancillary Information Facility

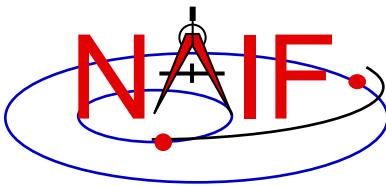
- **Unix specific:**

- Start the IDL application from a shell in the directory containing both `icy.dlm` and `icy.so`.
- Append the path to your `icy.dlm` to the `IDL_DLM_PATH` environment variable to include the directory containing `icy.dlm` and `icy.so`, e.g.:

```
setenv IDL_DLM_PATH "<IDL_DEFAULT>:_path_to_directoryContaining_icy.dlm_"
```

**Warning: do not invoke IDL from the Icy source directory, `icy/src/icy`, nor register that directory, and do not append that directory to `IDL_DLM_PATH`. This directory contains an “`icy.dlm`” but not “`icy.so`.”**

continued on next page



# Icy: Register the Icy DLM to IDL (3)

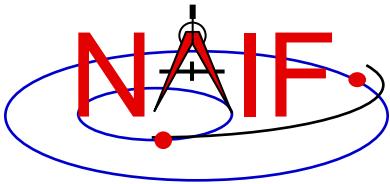
Navigation and Ancillary Information Facility

- **Windows specific:**
  - Set environment variable **IDL\_DLM\_PATH** from the *Advanced* pane of the *System Control Panel*.
- Once registered as specified on earlier pages, confirm IDL recognizes and can access Icy.
  - Using the help command:

```
IDL> help, 'icy', /DL
**ICY - IDL/CSPICE interface from JPL/NAIF (not loaded)
```

- » Appearance of the words “not loaded” might suggest something is wrong, but this is expected state until you execute an Icy command.
- Execute a trivial Icy command:

```
IDL> print, cspice_icy('version')
% Loaded DLM: ICY.
Icy 1.4.20 25-DEC-2008 (EDW)
```



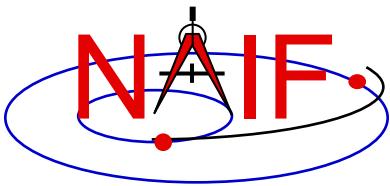
# Icy: Using the IDL IDE

---

Navigation and Ancillary Information Facility

**Recommended for “Icy”**

- Use the IDL IDE’s preferences panel to set the current working directory to the location where you will be developing your code.
- Optional: Place your `dlm_register` command in a start up script. Specify the script using the IDL IDE’s preferences panel.



# Mice

---

Navigation and Ancillary Information Facility

## Required for “Mice”

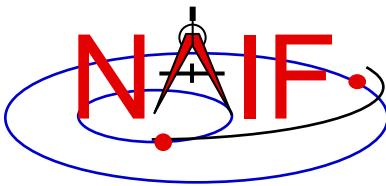
- Assume the Mice distribution is installed at `C:\naif\mice\` on Windows, or `/naif/mice/` on Unix/Linux. Use of Mice from Matlab requires the Mice source and library directories exist in the Matlab search path. The easiest way to update the Matlab path is with the “addpath” command.

- On Windows:

```
>> addpath('C:\naif\mice\lib')
>> addpath('C:\naif\mice\src\mice')
```

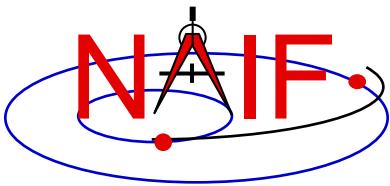
- On Unix/Linux:

```
>> addpath('/naif/mice/lib')
>> addpath('/naif/mice/src/mice')
```



# Backup

- **Icy programming example**
- **Mice programming example**
- **References**
- **Matlab 2016a MEX Change**



# Simple Icy Example

---

Navigation and Ancillary Information Facility

- As an example of Icy use with vectorization, calculate and plot the trajectory in the J2000 inertial frame of the Cassini spacecraft from June 20, 2004 to December 1, 2005.

```
;; Construct a meta kernel, "standard.tm", which will be used to load the needed
;; generic kernels: "naif0009.tls," "de421.bsp," and "pck0009.tpc."

;; Load the generic kernels using the meta kernel, and a Cassini spk.

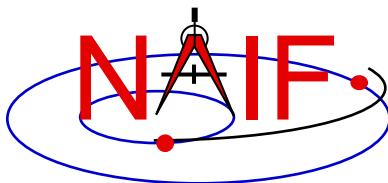
cspice_furnsh, 'standard.tm'
cspice_furnsh, '/kernels/cassini/spk/030201AP_SK_SM546_T45.bsp'

;; Define the number of divisions of the time interval and the time interval.
STEP = 10000
utc = [ 'Jun 20, 2004', 'Dec 1, 2005' ]
cspice_str2et, utc, et
times = dindgen(STEP)*(et[1]-et[0])/STEP + et[0]

cspice_spkpos, 'Cassini', times, 'J2000', 'NONE', 'SATURN BARYCENTER', pos, ltime

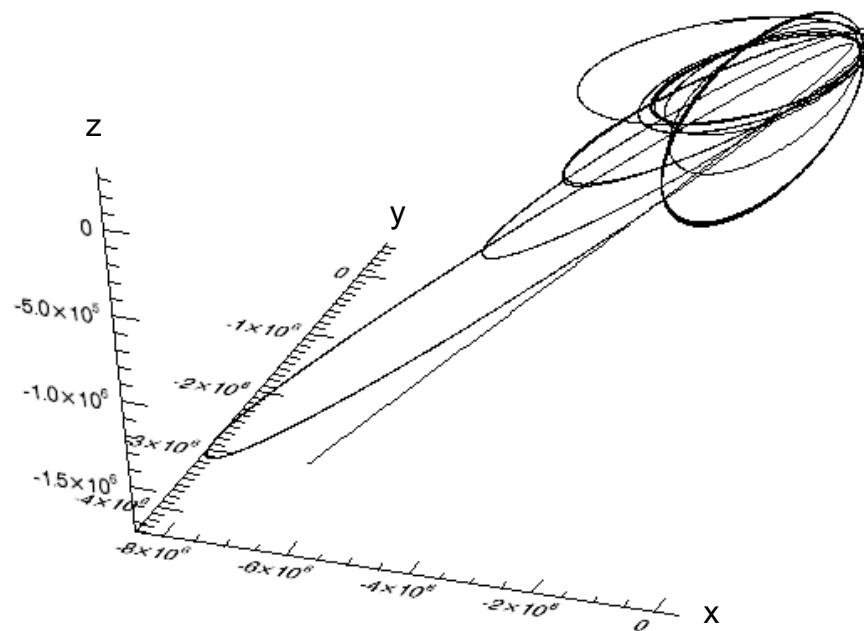
;; Plot the resulting trajectory.
x = pos[0,*]
y = pos[1,*]
z = pos[2,*]
iplot, x, y, z

cspice_kclear
```

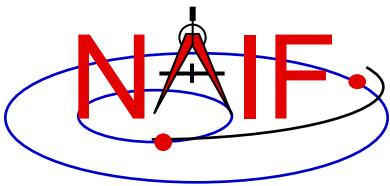


# Graphic Output

Navigation and Ancillary Information Facility



Trajectory of the Cassini vehicle in the J2000 frame, for June 20, 2004 to Dec 1, 2005



# Simple Mice Example

---

Navigation and Ancillary Information Facility

- As an example of Mice use with vectorization, calculate and plot the trajectory in the J2000 inertial frame of the Cassini spacecraft from June 20, 2004 to December 1, 2005

```
% Construct a meta kernel, "standard.tm", which will be used to load the needed
% generic kernels: "naif0009.tls," "de421.bsp," and "pck0009.tpc."

% Load the generic kernels using the meta kernel, and a Cassini spk.

cspice_furnsh( { 'standard.tm', '/kernels/cassini/spk/030201AP_SK_SM546_T45.bsp' } )

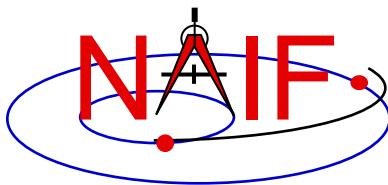
% Define the number of divisions of the time interval and the time interval.
STEP      = 1000;
et        = cspice_str2et( {'Jun 20, 2004', 'Dec 1, 2005'} );
times     = (0:STEP-1) * ( et(2) - et(1) )/STEP + et(1);

[pos, ltime]= cspice_spkpos( 'Cassini', times, 'J2000', 'NONE', 'SATURN BARYCENTER' );

% Plot the resulting trajectory.
x = pos(1,:);
y = pos(2,:);
z = pos(3,:);

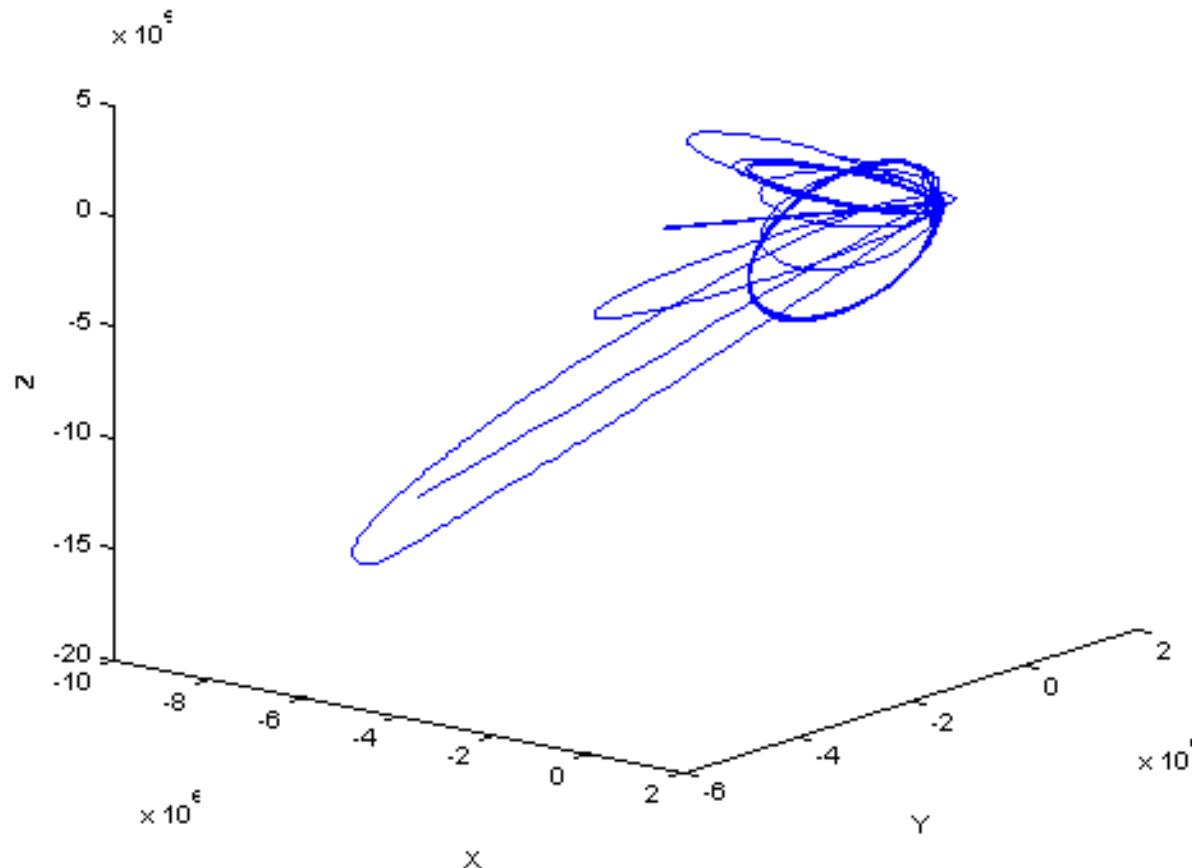
plot3(x,y,z)

cspice_kclear
```

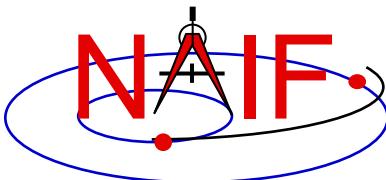


# Graphic Output

Navigation and Ancillary Information Facility



Trajectory of the Cassini vehicle in the J2000 frame, for June 20, 2004 to Dec 1, 2005

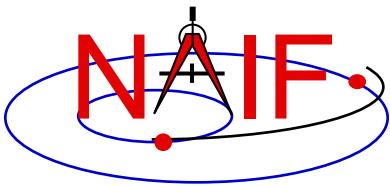


# References

---

Navigation and Ancillary Information Facility

- **NAIF documents providing more information concerning SPICE programing:**
  - “**icy.req**,” **Icy Required Reading**
    - » [icy/doc/icy.req](#)
    - » [icy/doc/html/req/icy.html](#)
  - “**mice.req**,” **Mice Required Reading**
    - » [mice/doc/mice.req](#)
    - » [mice/doc/html/req/mice.html](#)
  - “**cspice.req**,” **CSPICE Required Reading**
    - » [cspice/doc/cspice.req](#)
    - » [cspice/doc/html/req/cspice.html](#)
  - “**Introduction to the Family of SPICE Toolkits**” tutorial

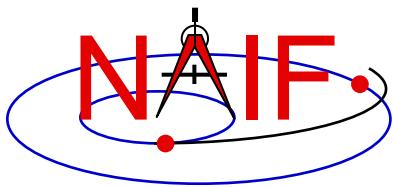


# Matlab 2016a MEX Change

---

Navigation and Ancillary Information Facility

- **Mathworks changed the operation of the MEX utility in the Matlab 2016a. Use of the mkprodct.csh/mkprodct.bat build script included with SPICE Toolkit N66 or earlier will fail to build Mice against 2016a or later. NAIF will include a modified version of the build scripts in the version N67 Toolkits.**
- **Most Mice users should \*NOT\* rebuild the Mice Toolkit.**

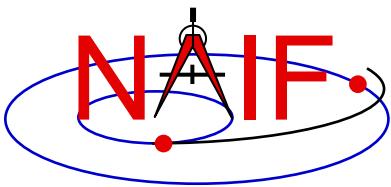


---

Navigation and Ancillary Information Facility

# Introduction to Kernels

January 2020

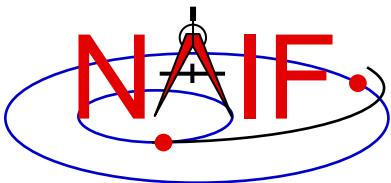


# Agenda

---

Navigation and Ancillary Information Facility

- **Overview**
- **Kernel architecture**
- **Producing kernels**
- **Using kernels**



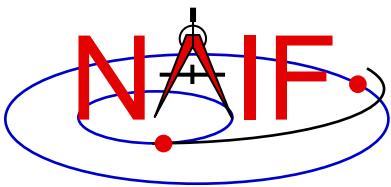
# What is a SPICE “Kernel”

Navigation and Ancillary Information Facility

## “Kernel” means file

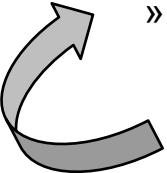
## “Kernel” means a file containing ancillary data

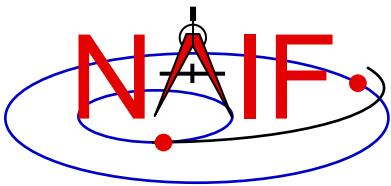
“Kernel” means a file containing “low level” ancillary data that may be used, along with other data and SPICE Toolkit software, to determine higher level observation geometry parameters of use to scientists and engineers in planning and carrying out space missions, and analyzing data returned from missions.



# The Family of SPICE Kernels

Navigation and Ancillary Information Facility

- **SPK**
    - Spacecraft and Planet Ephemeris
  - **PCK**
    - Planetary Constants, for natural bodies
      - » Orientation
      - » Size and shape
  - **IK**
    - Instrument
  - **CK**
    - Orientation (“Camera-matrix”)
  - **EK**
    - Events, up to three distinct components
      - » ESP: science plan
      - » ESQ: sequence
      - » ENB: experimenter’s notebook
-  EK is rarely used
- **FK**
    - Reference frame specifications
  - **SCLK**
    - Spacecraft clock correlation data
  - **LSK**
    - Leapseconds
  - **MK**
    - Meta-Kernel (a.k.a. “FURNSH kernel”)
    - Mechanism for aggregating and easily loading a collection of kernel files
  - **DSK**
    - Digital shape kernel
      - » Tesselated plate model
      - » Digital elevation model (under development)
  - **DBK**
    - Database mechanism
      - » Primarily used to support the ESQ

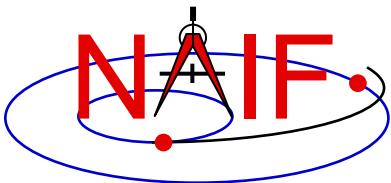


# SPICE Kernel Forms

---

Navigation and Ancillary Information Facility

- **Binary form**
  - Files containing mostly data encoded in binary form
    - » They also contain a small amount of ASCII text
  - Provide rapid access to large amounts of numeric data
  - Require the use of SPICE Toolkit software to produce them
  - Require the use of SPICE Toolkit software to utilize their contents
- **Text form**
  - Files containing only printing characters (ASCII values 32-126), i.e. human-readable text.
  - Produced using a text editor
  - Require the use of SPICE Toolkit software to utilize their contents
- **“Transfer” form of a binary kernel**
  - An ASCII representation of a binary kernel
  - Was used for porting the file between computers with incompatible binary representations (e.g. PC and UNIX); no longer needed
    - » But it is one way to convert a non-native binary kernel into native format, needed for modifying the kernel or improving read efficiency



# Text and Binary Kernels

Navigation and Ancillary Information Facility

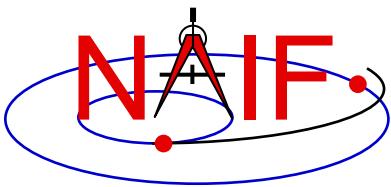
## SPICE **text** kernels are:

- text PCK (the most common type of PCK)
- IK
- FK
- LSK
- SCLK
- MK

## SPICE **binary** kernels are:

- SPK
- binary PCK (has been used only for Earth, Moon and Eros)
- CK
- DSK
- ESQ (part of the E-kernel)
- DBK (database kernel)

Rarely used



---

Navigation and Ancillary Information Facility

# Kernel Architecture

- **Text kernels**
- **Binary kernels**
- **Comments in kernels**



# Text Kernel Contents

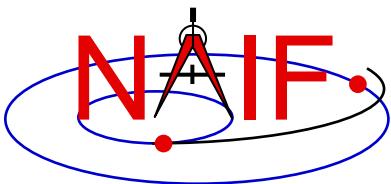
---

Navigation and Ancillary Information Facility

- **A text kernel is a plain text file of ASCII data**
- **It contains assignments of the form:**

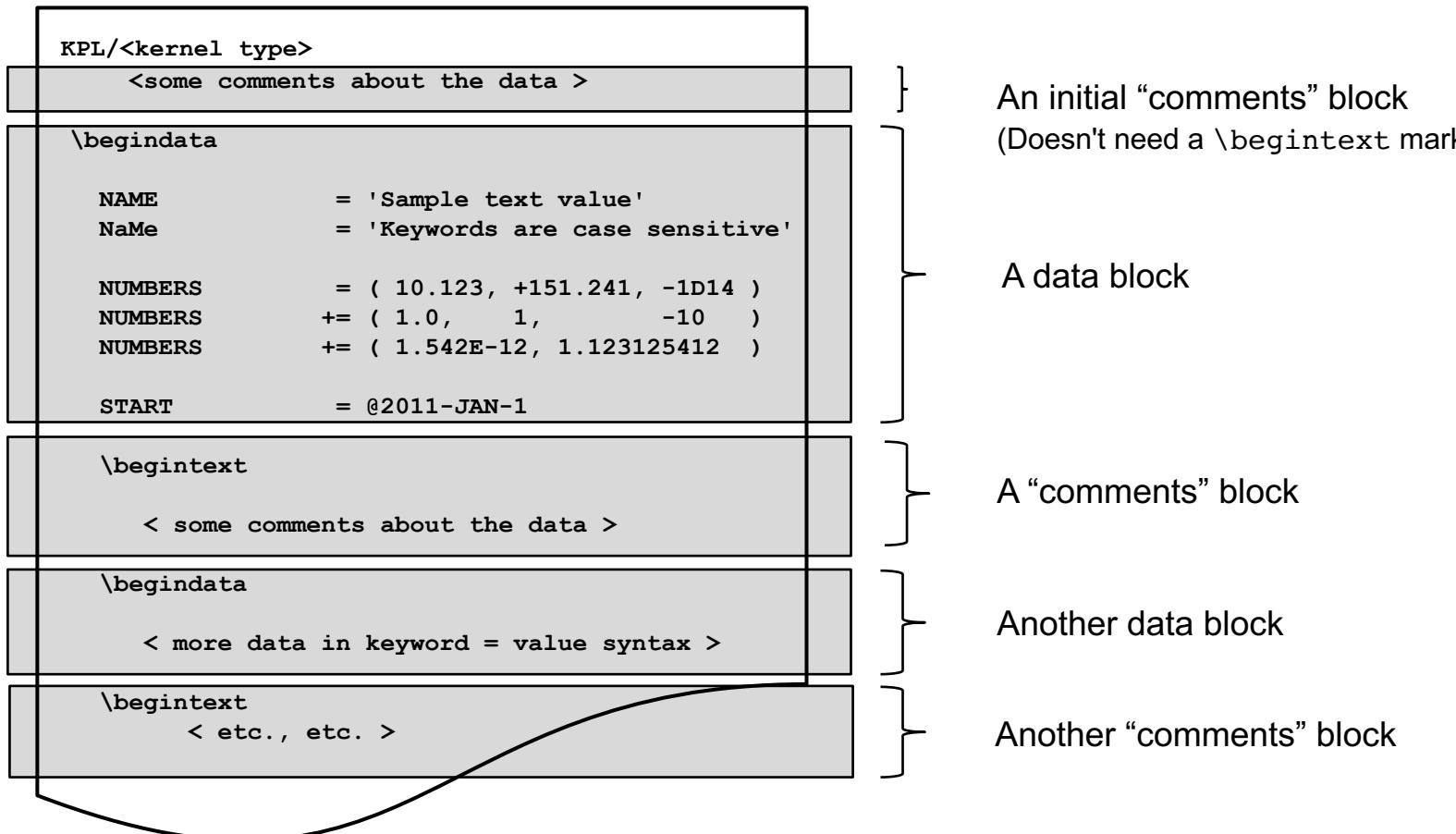
```
variable_name = value(s)
```

- **A text kernel should also contain descriptive comments that describe the assignments**
  - Comments are sometimes referred to as “meta-data”
    - » Don’t confuse this usage with the “meta-kernel” described later in this tutorial

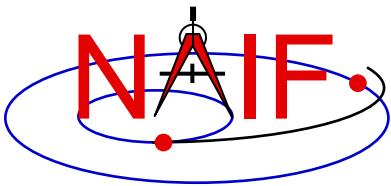


# Example Text Kernel

## Navigation and Ancillary Information Facility



- The next several pages describe what you see above
- See the “Kernel Required Reading” document for details

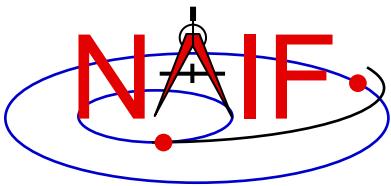


# Text Kernel Formatting

---

Navigation and Ancillary Information Facility

- **KPL/<text kernel type>**
  - Its use is optional, but is highly recommended
  - Must appear on the first line, starting in column 1
  - Tells SPICE software what kind of kernel it is
  - Text kernel types are FK, IK, PCK, SCLK, MK
- **\begindata and \begin{text}**
  - Markers, on lines by themselves, which set off the beginning of data and the beginning of comment (metadata) blocks respectively
    - They need not begin in column 1
    - An initial set of comments need not be preceded by a \begin{text} marker
- **<LF> for Unix/Linux/Mac or <CR><LF> for Windows**
  - End of line marker (usually not visible when displaying a text kernel)
  - Must be present on EVERY line in the text kernel
- **Max line length, including any white space is 132 characters**

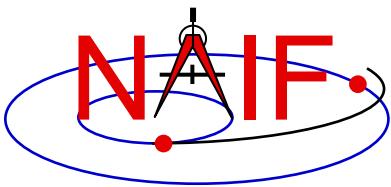


# Text Kernel Operators

---

Navigation and Ancillary Information Facility

- An assignment using the “=” operator associates one or more values with a variable name.
- An assignment using the “+=” operator associates additional values with an existing variable name.
- An assignment using the “@” symbol associates a calendar date with a variable name.
  - The string will be parsed and converted to an internal double precision representation of that epoch as seconds past the J2000 epoch
    - » There is no time system implied
    - » This conversion does not need a leap seconds kernel

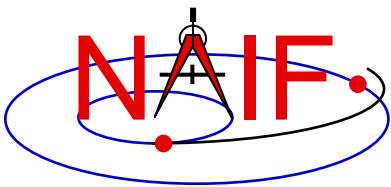


# Variable Names and Values

---

Navigation and Ancillary Information Facility

- **Variable names**
  - Max of 32 characters
  - Are case sensitive (**recommendation: use only upper case**)
  - Cannot include space, period, parenthesis, equals sign or tab
  - Recommendation: don't use the “+” sign as the last character
- **Values**
  - Numeric: integer, fixed point and scientific notation are allowed
  - String:
    - » enclosed in single quotes
    - » maximum length of 80 characters on a given line
      - SPICE has means to concatenate multiple string values to allow for values exceeding 80 characters
    - » string values may contain any printing ASCII character, including blank
  - Time: identified by the “@” character
  - Any of these three types can be provided as an n-dimensional vector of values
    - » Components are separated by commas or white space (but not TABs)
    - » Parentheses are used to enclose the vector
    - » Each string value in a vector is contained in single quotes
    - » Values in a vector must all be of the same type (numeric, string or time)
- See “Kernel Required Reading” for more information

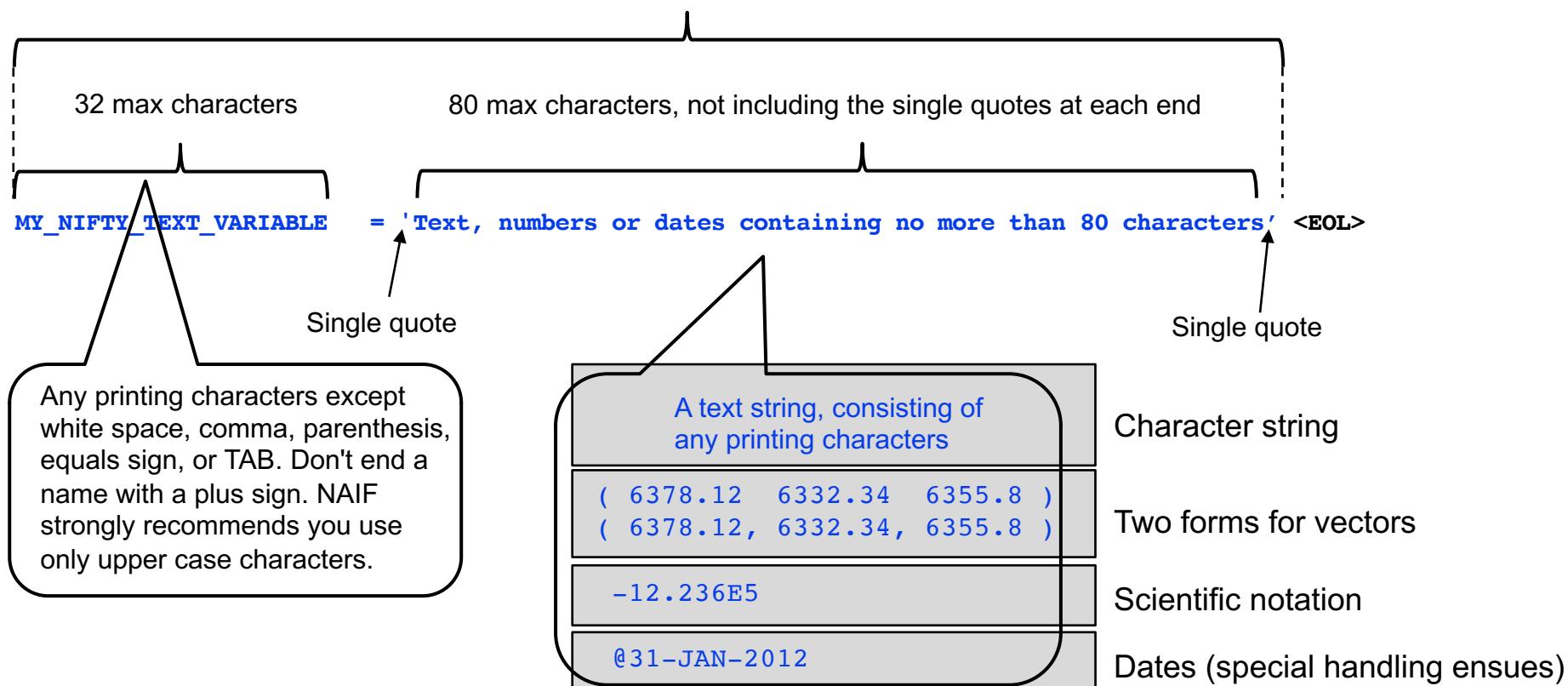


# Variable Names and Values

Navigation and Ancillary Information Facility

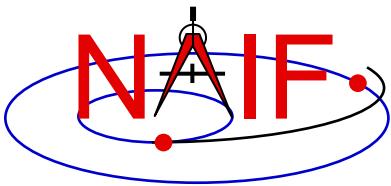
- A “picture” of the most basic text kernel assignment rules

132 max characters, with the non-printing system-dependent end-of-line indicator at the end\*



\*Unix, Linux, OSX EOL symbol: <LF>

\*DOS/Windows EOL symbol: <CR><LF>



# Example Binary Kernel

Navigation and Ancillary Information Facility

**A binary kernel contains lots  
of non-printing data.**

**Includes a “comment area”  
where descriptive meta-data  
provided as ASCII text  
should be placed.**

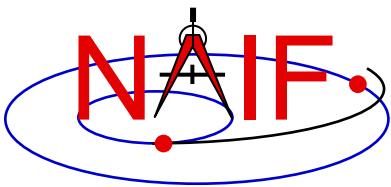


# Comments In SPICE Kernels

---

Navigation and Ancillary Information Facility

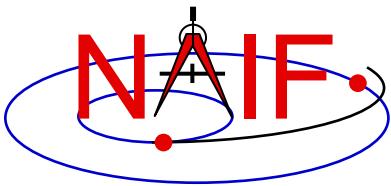
- **All SPICE kernels should contain comments—descriptive information about the data contained in the file.**
  - “Comments” are also known as “meta-data”
- **See the tutorial on comments for more information.**



---

Navigation and Ancillary Information Facility

# Producing Kernels

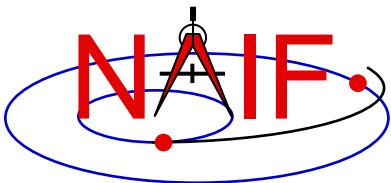


# Making a Text Kernel

---

Navigation and Ancillary Information Facility

- **Text kernels may be produced using a text editor**
  - Text kernels must contain only printing characters (ASCII values 32-126), i.e. human-readable text
    - » TAB characters are allowed but **HIGHLY DISCOURAGED**
    - » Caution: some text editors insert non-printing characters
  - Text kernels must have each line terminated with the end-of-line indicator appropriate for the operating system you are using
    - » For Unix, PC/Linux, Mac OSX: <LF>
    - » For PC/Windows: <CR><LF>
    - » **Don't forget to insert the end-of-line indicator on the very last line of the kernel!**
  - Fortran toolkit software will detect and warn you if trying to read a non-native text kernel. (Not needed for other languages.)
    - » Caution: this warning doesn't work for a file smaller than 132 bytes
  - See the BACKUP for information on converting text kernels between these two line termination techniques

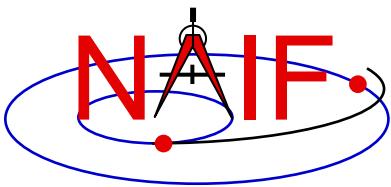


# Making a Binary Kernel

---

Navigation and Ancillary Information Facility

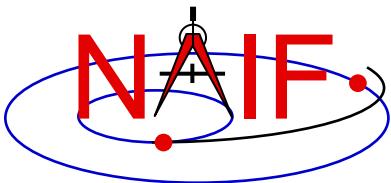
- **Binary kernels are made using Toolkit utility programs, or by using Toolkit APIs built into your own application program**
- **See “How Kernels are Made and Used” in the BACKUP section for a bit more information**
- **See the “Making an SPK” and “Making a CK” tutorials**



---

Navigation and Ancillary Information Facility

# Using Kernels

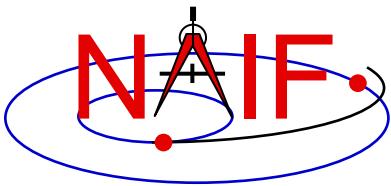


# Loading Kernels - 1

---

Navigation and Ancillary Information Facility

- To make kernels available to a program you “load” them
- When you load a text kernel:
  - the file is opened
  - the kernel contents are read into memory
    - » variable names and associated values are stored in a data structure called the “kernel pool”
  - the file is closed
- When you load a binary kernel:
  - the file is opened
  - for SPK, CK, binary PCK and DSK files, no data are read until a read request is made by Toolkit software
  - for ESQ files, the schema description is read, checked, and stored in memory at load time, but no data are read until a query/fetch is made
  - for all practical purposes the binary file remains open unless specifically unloaded by you



# Loading Kernels - 2

Navigation and Ancillary Information Facility

- Use the **FURNSH** routine to load all kernels – text and binary
  - `CALL FURNSH ( 'name.ext' )` (Fortran)
  - `furnsh_c ( "name.ext" );` (C)
  - `cspice_furnsh, 'name.ext'` (IDL)
  - `cspice_furnsh ( 'name.ext' )` (MATLAB)
  - `spiceypy.furnsh ( 'name.ext' )` (Python using SpiceyPy)
- Best practice: don't hard code filenames as shown above— instead, list them in a “meta-kernel” and load the meta-kernel using **FURNSH**
  - `CALL FURNSH ( 'meta-kernel_name' )` (Fortran example)
  - Look further down for more information on meta-kernels
- Caution: “Transfer format” versions of binary kernels can not be loaded; they must first be converted to binary with the Toolkit utility program *tobin* or *spacit*

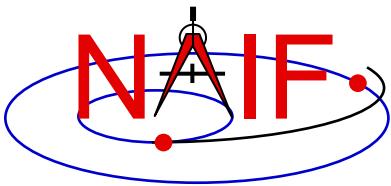


# Run-time Translation

---

Navigation and Ancillary Information Facility

- **Binary kernels, whether or not in native binary format, may be read by any of the toolkits**
  - Accomplished by run-time translation built into Toolkit code
  - Run-time translation does NOT apply to writing to an existing binary kernel
- **Text kernels may be read by any of the C, IDL and Matlab Toolkits no matter if the end-of-line terminator is Windows style (<CR><LF>) or OSX/Linux style (<LF>)**
  - Accomplished by run-time translation built into Toolkit code
  - **Run-time translation does NOT work for Fortran Toolkits: these Toolkits read text kernels only in native format**

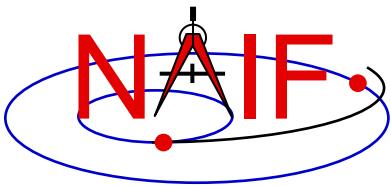


---

Navigation and Ancillary Information Facility

## Meta-Kernels

**These help make kernel management easy!**

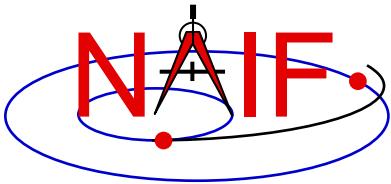


# What is a “Meta-Kernel”

---

Navigation and Ancillary Information Facility

- **A meta-kernel is a file that lists names (and locations) of a collection of SPICE kernels that are to be used together in a SPICE-based application**
  - Loading the meta-kernel causes all of the kernels listed in it to be loaded
- **Using a meta-kernel makes it easy to manage which SPICE files are loaded into your program. You don't need to revise your code—just edit your meta-kernel**
- **A meta-kernel is implemented using the SPICE text kernel standards**
  - Refer to the Kernel Required Reading technical reference for details
- **The terms “meta-kernel” and “FURNISH kernel” are used synonymously**



# Sample Meta-Kernel

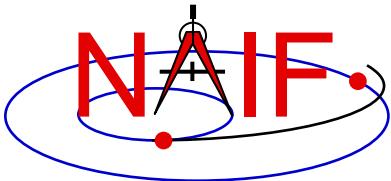
Navigation and Ancillary Information Facility

KPL/MK

```
\begindata
```

```
KERNELS_TO_LOAD = (
    '/home/mydir/kernels/lowest_priority.bsp',
    '/home/mydir/kernels/next_priority.bsp',
    '/home/mydir/kernels/highest_priority.bsp',
    '/home/mydir/kernels/leapseconds.tls',
    '/home/mydir/kernels/sclk.tsc',
    '/home/mydir/kernels/c-kernel.bc',
    '/home/mydir/kernels+',
    '/custom/kernel_data/p_constants.tpc',
)
```

All the commas  
are optional



# Sample Meta-Kernel

Navigation and Ancillary Information Facility

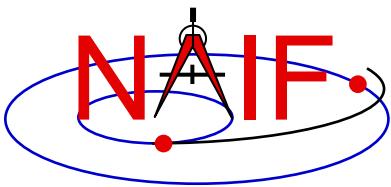
KPL/MK

```
\begindata
```

```
KERNELS_TO_LOAD = (
    '/home/mydir/kernels/lowest_priority.bsp',
    '/home/mydir/kernels/next_priority.bsp',
    '/home/mydir/kernels/highest_priority.bsp',
    '/home/mydir/kernels/leapseconds.tls',
    '/home/mydir/kernels/sclk.tsc',
    '/home/mydir/kernels/c-kernel.bc',
    '/home/mydir/kernels+' ,
    '/custom/kernel_data/p_constants.tpc' ,
)
```

All the commas  
are optional

- The last file listed in this example (`p_constants.tpc`) demonstrates how to use the continuation character, '+', to work around the 80 character limitation imposed on string lengths by the text kernel standards.
- See the next two pages for some important OS-specific details!



# Unix/Mac Sample Meta-Kernel

Navigation and Ancillary Information Facility

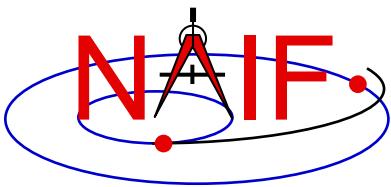
- This meta-kernel uses the **PATH\_VALUES** and **PATH\_SYMBOLS** keywords to specify the directory where the kernels are located.

KPL/MK

```
\begindata
  PATH_VALUES      = ( '/home/mydir/kernels' )
  PATH_SYMBOLS     = ( 'KERNELS' )
  KERNELS_TO_LOAD = (
    '$KERNELS/lowest_priority.bsp',
    '$KERNELS/next_priority.bsp',
    '$KERNELS/highest_priority.bsp',
    '$KERNELS/leapseconds.tls',
    '$KERNELS/sclk.tsc',
    '$KERNELS/c-kernel.bc',
    '$KERNELS/custom/kernel_data/p_constants.tpc'
  )
```

UNIX/MAC style path notation, using forward slashes

- Although the OS environment variable notation \$<name> is used to refer to the symbols specified using the **PATH\_VALUES** and **PATH\_SYMBOLS** keywords, these symbols are NOT operating system environment variables and are set and used for substitution by SPICE only in the context of this particular meta-kernel.
- The '+' continuation character described on the previous page may be used to handle path strings that exceed 80 characters.



# Windows Sample Meta-Kernel

Navigation and Ancillary Information Facility

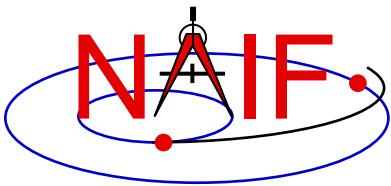
- This meta-kernel uses the PATH\_VALUES and PATH\_SYMBOLS keywords to specify the directory where the kernels are located.

KPL/MK

```
\begindata
  PATH_VALUES      = ( 'c:\home\mydir\kernels' )
  PATH_SYMBOLS     = ( 'KERNELS' )
  KERNELS_TO_LOAD = (
    '$KERNELS\lowest_priority.bsp',
    '$KERNELS\next_priority.bsp',
    '$KERNELS\highest_priority.bsp',
    '$KERNELS\leapseconds.tls',
    '$KERNELS\sclk.tsc',
    '$KERNELS\c-kernel.bc',
    '$KERNELS\custom\kernel_data\p_constants.tpc'
  )
```

Windows style path notation, using backwards slashes

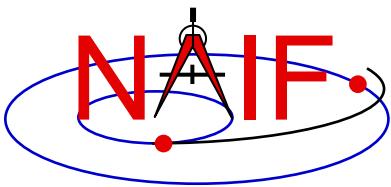
- Although the OS environment variable notation \$<name> is used to refer to the symbols specified using the PATH\_VALUES and PATH\_SYMBOLS keywords, these symbols are NOT operating system environment variables and are set and used for substitution by SPICE only in the context of this particular meta-kernel.
- The '+' continuation character described on the previous page may be used to handle path strings that exceed 80 characters.



# Limits on Loaded Kernels (N66)

Navigation and Ancillary Information Facility

- The number of all types of kernels that may be loaded at any time is large, but limited.
  - As of the version N66 Toolkits it is limited to 5,300
    - » Assumes each kernel has been loaded only once, and not unloaded.
- As of the version N66 Toolkits the number of binary kernels that may be loaded at the same time is limited to 5000
  - Binary kernel types are: SPK, binary PCK, CK and DSK
    - » Also the rarely used ESQ
- There are also limits on the number of keywords and values for all loaded text kernels:
  - Maximum number of keywords is 26,003
  - Maximum number of numeric data items is 400,000
  - Maximum number of character data items is 15,000

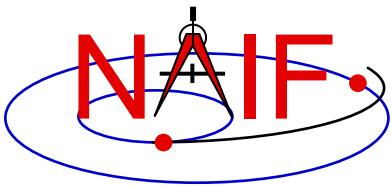


# Kernel Precedence Rule

---

Navigation and Ancillary Information Facility

- The order in which SPICE kernels are loaded at run-time determines their priority when requests for data are made
  - For binary kernels, data from a higher priority file will be used in the case when two or more files contain data overlapping in time for a given object.
    - » For SPKs, CKs, and binary PCKs the file loaded **last** takes precedence (has higher priority).
    - » For DSKs, use of priority will be specified via API calls
      - » Not yet supported as of N66 Toolkits
    - » Priority doesn't apply to ESQ files – all data from all loaded files are available.
  - If two (or more) text kernels assign value(s) to a single keyword using the “=” operator, the data value(s) associated with the last loaded occurrence of the keyword are used—all earlier values are replaced with the last loaded value(s).
  - Orientation data from a binary PCK always supersedes orientation data (for the same object) obtained from a text PCK, no matter the order in which the kernels are loaded.

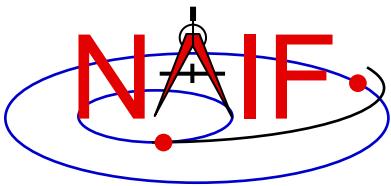


# Unloading Kernels

---

Navigation and Ancillary Information Facility

- The unloading of a kernel is infrequently needed for FORTRAN or CSPICE applications but is **essential** for Icy, Mice, Python and similar interpreter scripts.
  - Because of the way IDL and MATLAB interact with external shared object libraries, any kernels loaded during an IDL or MATLAB session will stay loaded until the end of the session unless they are specifically unloaded.
- The routines KCLEAR and UNLOAD may be used to unload kernels containing data you wish to be no longer available to your program.
  - KCLEAR unloads all kernels and clears the kernel pool
  - UNLOAD unloads specified kernels
  - KCLEAR and UNLOAD are only capable of unloading kernels that have been loaded with the routine FURNISH. They will not unload any files that have been loaded with older load routines such as SPKLEF (those used prior to availability of FURNISH).
- Caution: unloading text kernels with UNLOAD will also remove any kernel pool data provided through the kernel pool run-time data insertion/update APIs (PCPOOL, PDPOOL, PIPOOL).

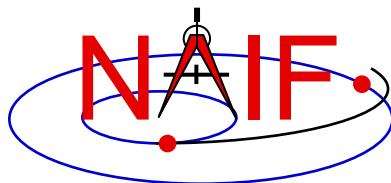


# Backup

---

Navigation and Ancillary Information Facility

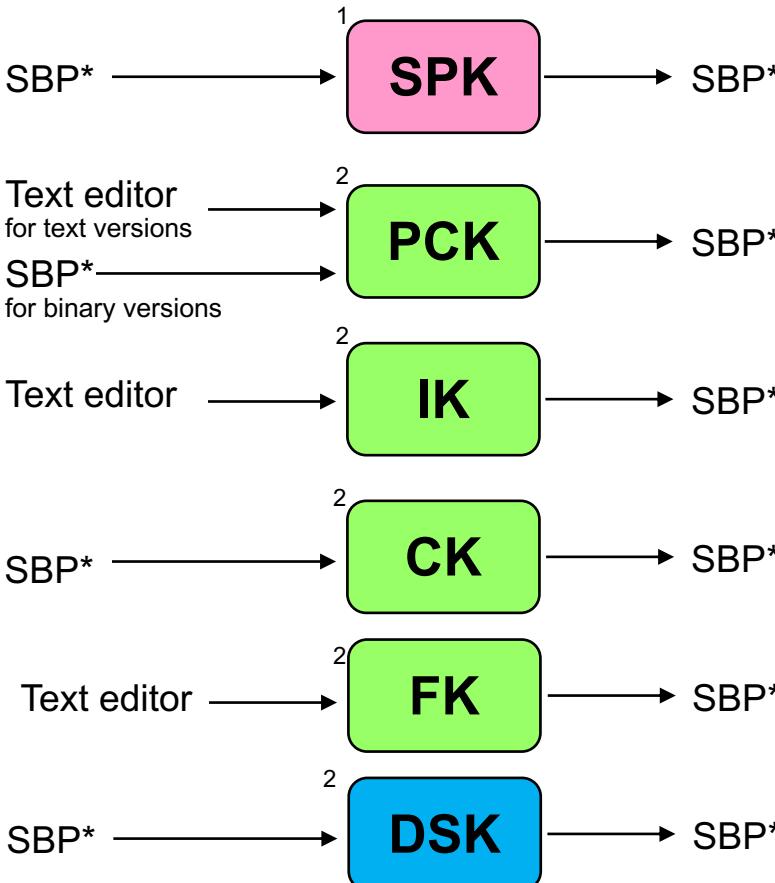
- **How kernels are made and used**
- **Why and how kernels are modified**
- **SPICE data structures hierarchy**
- **Problems making text kernels**



# How Kernels are Made and Used at JPL

## Navigation and Ancillary Information Facility

### How Made?



### How Used?

SBP\*

### How Made?

Text editor or existing file, input via ESQ or ENB

SBP\*

Browser or e-mail

Text editor

SBP\*

Text editor

### How Used?

Web browser or SBP\*, depending on implementation

SBP\*

SBP\*

SBP\*

SBP\*

SBP\*

### The EK family

ESP

ESQ

ENB

LSK

SCLK

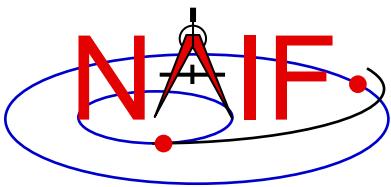
Meta-kernel  
(FURNISH)

### Who usually makes the kernels at JPL?

- 1 NAV and NAIF
- 2 NAIFF
- 3 NAIFF or other

This represents current practice for most JPL missions, but is by no means a requirement. **Anyone can make SPICE files.**

**\*SBP** = SPICE-based program that uses modules from the SPICE Toolkit. In some cases the Toolkit contains such a program already built. In some cases NAIFF may have such a ready-built program that is not in the SPICE Toolkit.



# Why & How Kernels are “Modified” - 1

## Navigation and Ancillary Information Facility

### File Type

### Why Modified

### How Modified

**SPK**

- To add comments
- To merge files or subset a file
- To correct/revise an object ID

- COMMNT, SPACIT or SPICELIB module
- SPKMERGE, DAFCAT
- BSPIDMOD

**PCK**

Text version

- To revise data values
- To add additional data items and values

- Text editor
- Text editor

**IK**

- To revise data values
- To add additional data items and values

- Text editor
- Text editor

**CK**

- To add comments
- To merge files
- To revise the interpolation interval
- To subset a file

- COMMNT, SPACIT, or SPICELIB module
- DAFCAT, CKSMRG
- CKSPANIT, CKSMRG
- CKSLICER

**FK**

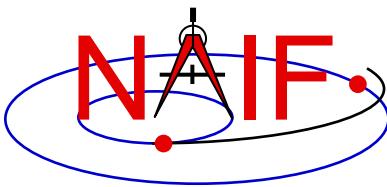
- To revise data values
- To add additional data items and values

- Text editor
- Text editor

**DSK**

- To add comments
- To merge files

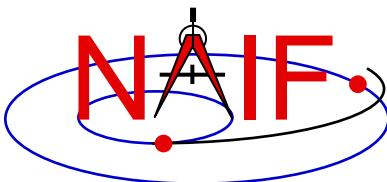
- COMMNT, SPACIT or SPICELIB module
- DLACAT



# Why & How Kernels are “Modified” - 2

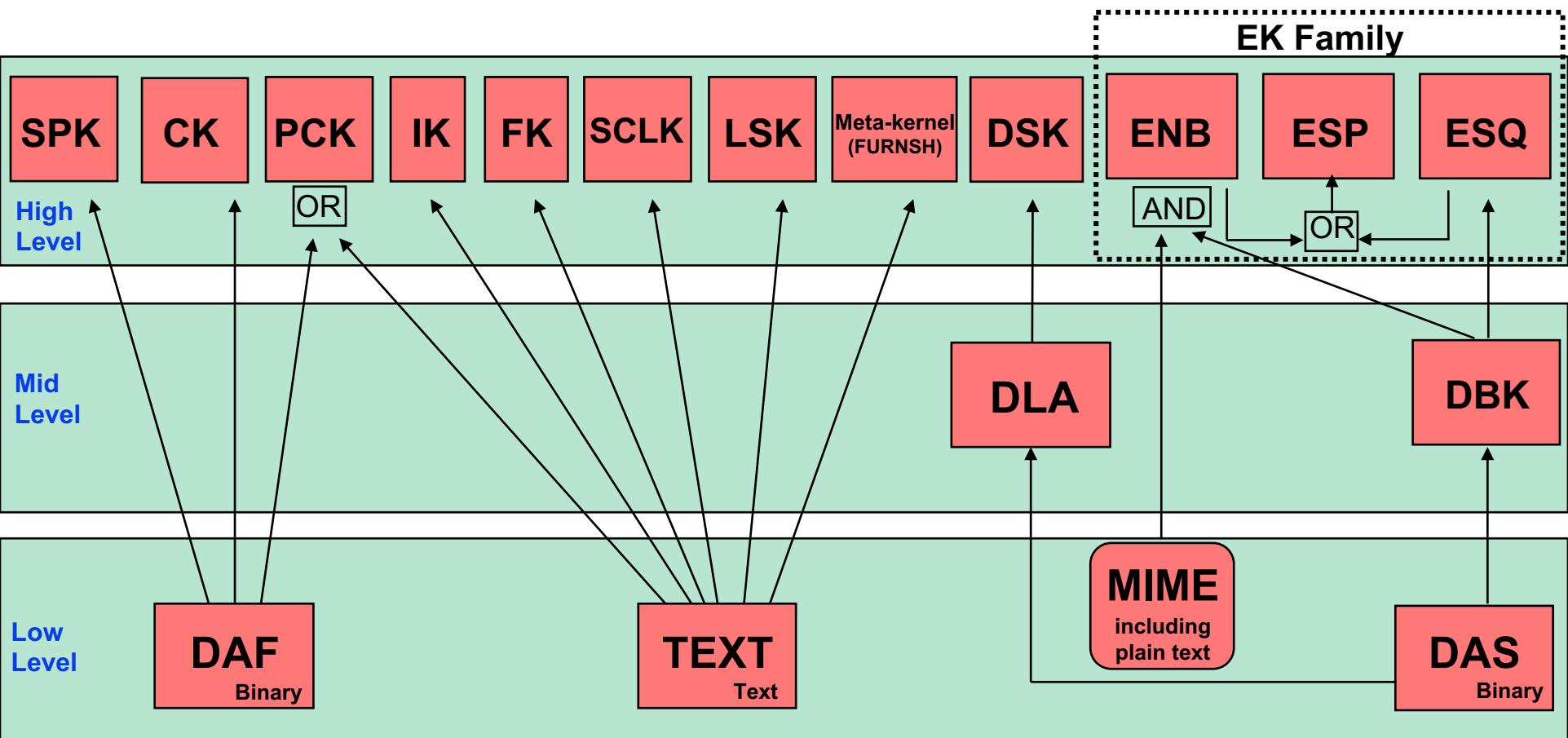
## Navigation and Ancillary Information Facility

<u>File Type</u>	<u>Why Modified</u>	<u>How Modified</u>
The EK family		
ESP	- To add, revise or delete “data” - To add comments	- (Depends on implementation) - (Depends on implementation)
ESQ	- To add additional data - To revise data - To delete data - To add comments - To merge files	- Toolkit modules - Toolkit modules - Toolkit modules - COMMNT, SPACIT or SPICELIB module - (under development)
ENB	- To change entry status (public <--> private) - To delete an entry	- WWW - WWW
LSK	- To add a new leapsecond	- Text editor
SCLK	- To add comments	- Text editor
Meta-kernel (FURNSH)	- To add or remove kernels to be used in a program	- Text editor



# SPICE Data Structures Hierarchy

Navigation and Ancillary Information Facility



DAF = Double Precision Array File

DBK = Data Base Kernel

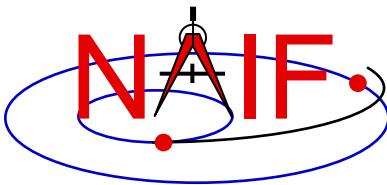
DAS = Direct Access, Segregated

DSK = Digital Shape Kernel (under development)

DLA = DAS Linked Array (under development)

Excepting MIME, each of these data structures is built entirely of SPICE components.

PCK files are usually text-based, but binary versions exist for the earth and moon. The ESP has been implemented using both the ENB and ESQ mechanisms. The DBK is a SQL-like, homebrew database.

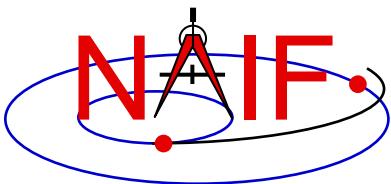


# Problems Making Text Kernels

---

Navigation and Ancillary Information Facility

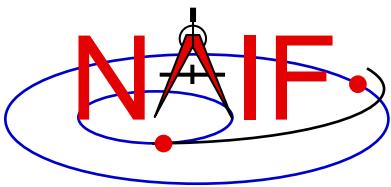
- **Cutting/pasting complete, or pieces of, data assignments or \begindata or \begin{text} markers into a text kernel can cause a problem**
  - It may result in insertion of non-printing characters or incorrect end-of-line terminators
  - This is not a problem for comments, but it is probably best to treat all portions of a text kernel the same
- **If creating a text kernel by editing an existing one:**
  - first save a backup copy
  - be sure you are starting with a file in native format for the computer you are using: either Unix/Linux/Mac or Windows
  - use single quotes around any string (character) values
  - be sure to insert a final end-of-line marker at the end of your last line of data or text
    - » Press the “return” key to accomplish this



# Some Useful Tools - 1

Navigation and Ancillary Information Facility

- For a Unix or Linux (including Mac) environment
  - In order to display all non-printing characters, display tab characters as “`^I`”, and place a “`$`” character at the end of each line:
    - » `cat -et <file name>`
  - How do end-of-line markers appear when displayed in a text file using the `cat -et` command?
    - » Unix/Linux/Mac: `$` (line feed)
    - » Windows: `^M$` (carriage return followed by line feed)
  - In order to display the file type, language used, and end of line marker
    - » `file <file name>`
    - » Examples using Unix and Windows (“PC”) versions of the SPICE leapseconds kernel:
      - `file naif0010.tls`  
`naif0010.tls: ASCII English text`
      - `file naif0010.tls.pc`  
`naif0010.tls.pc: ASCII English text, with CRLF line terminators`



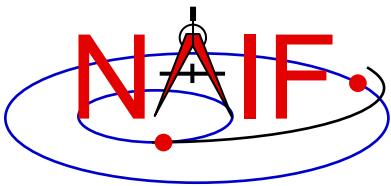
# Some Useful Tools - 2

---

Navigation and Ancillary Information Facility

- **For a Unix or Linux (including Mac) environment**
  - To convert a Unix/Linux/Mac text kernel to Windows (“DOS”) style:
    - » `unix2dos <filename>`
  - To convert a Windows (“DOS”) style text kernel to Unix/Linux/Mac style:
    - » `dos2unix <filename>`
  - Unix2dos and dos2unix are often available on Unix-based computers, and may be easily obtained from the www
  - Alternatively, to convert either style text kernel to the other style, use the SPICE “bingo” program
    - » The bingo program and User Guide are available only from the NAIF/Toolkit/Utilities web page:
      - <http://naif.jpl.nasa.gov/naif/utilities.html>
- **More information**

In Wikipedia, search on “newline” or “unix2dos”



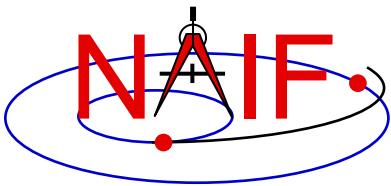
---

Navigation and Ancillary Information Facility

# “Comments” In SPICE Kernels

Also known as “meta-data”

January 2020

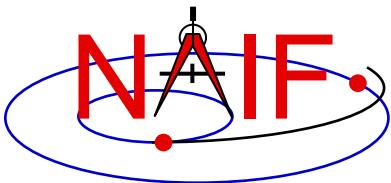


# What are Comments?

---

Navigation and Ancillary Information Facility

- **Comments, also called “meta-data,” are information that describe the context of kernel data, i.e. “data about data”**
- **Comments are provided inside kernels as plain text (prose)**
- **Examples of uses for comments:**
  - Data descriptions
    - » **“This file contains representations of the trajectories for bodies X, Y and Z over the interval from launch to landing”**
  - Data accuracy comments
  - Data pedigree
    - » **How and by whom the kernel was created**
      - The program(s) and/or steps used in creation
      - Contact information for user’s questions
        - email address
        - phone numbers
    - » **Data sources used as inputs when creating the kernel**
  - Intended kernel usage
  - Names of companion files

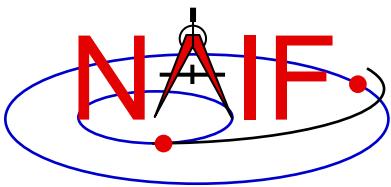


# Where are Comments Stored?

Navigation and Ancillary Information Facility

- **Binary kernels contain a reserved “comment area” to hold comments**
- **Text kernels have comments interleaved with the data**
  - Comments may be placed at the beginning of the text kernel, before any data, and ...
  - Comments may be inserted between blocks of data using `\begintext` and `\begindata` as start and end markers:

```
\begintext
    Some comments
\begin{data}
    Some data
```



# Adding Comments to Kernels

Navigation and Ancillary Information Facility

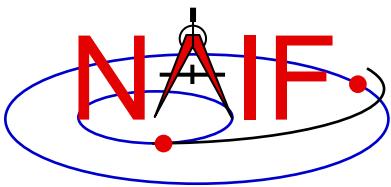
- **Binary Kernels**
  - Use the *commnt* utility program, available in the Toolkit
  - Include comment information at the time of kernel creation using SPICE APIs (subroutines)
- **Text Kernels**
  - Use a text editor
    - » Begin comment sections with a “\begintext” marker, placed alone on a line
      - (The marker is not needed for comments occurring before any data)
    - » End comment sections with a “\begindata” marker, placed alone on a line
      - (The marker is not needed if there are no data following the comments)
- **Restrictions**
  - For both binary and text kernels
    - » Comment line length limit is 255 characters. However, **NAIF recommends using no more than 80 characters per line** as this makes your comments far more readable!
    - » Use only printing characters (ASCII 32 - 126)
    - » Manipulating binary kernel comments requires the kernel be in the native binary format for the machine being used
  - For text kernels
    - » Refer to “Kernel Required Reading” (*kernel.req*) for details



# Viewing Comments in Kernels

Navigation and Ancillary Information Facility

- **Binary kernels:**
  - Use either the *commnt* or *spacit* utility program
    - » Both are available in all Toolkits
- **Text kernels:**
  - Use any available text file utility, such as:
    - » more, cat, vi, emacs
    - » Notepad,TextEdit, BBEdit, Word, etc.



# Viewing Comments in Binary Kernels

Navigation and Ancillary Information Facility

This example shows reading the comments in an SPK file using the “commnt” utility program

Terminal Window

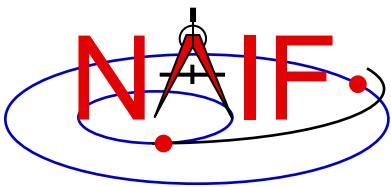
```
Prompt> commnt -r de421.bsp | more

...
DE 421 JPL Planetary Ephemeris SPK
=====
Original file name: de421.bsp
Creation date: Feb. 13, 2008
File created by: Dr. William Folkner (SSD/JPL)
Comments added by: Nat Bachman (NAIF/JPL)

This SPK file was released on February 13, 2008 by the Solar System Dynamics Group of JPL's Guidance, Navigation, and Control section.

The DE 421 planetary ephemeris is described in JPL IOM 343R-08-002, dated Feb. 13, 2008. The introduction of that memo states, in part, that this ephemeris "represents an overall update for all

--More--
```



# Viewing Comments in Text Kernels

## Navigation and Ancillary Information Facility

This example shows use of the unix “more” processor to show some of the comments at the beginning of a text kernel.

Terminal Window

```
prompt> more naif0008.tls

KPL/LSK

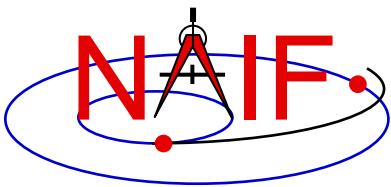
LEAPSECONDS KERNEL FILE
=====
Modifications:
-----
2005, Aug. 3   NJB   Modified file to account for the leapsecond
                      that will occur on December 31, 2005.

• 1998, Jun 17   WLT   Modified file to account for the leapsecond
                      that will occur on December 31, 1998.

1997, Feb 22   WLT   Modified file to account for the leapsecond
                      that will occur on June 30, 1997.

...etc.

-More-- (19%)
```



---

Navigation and Ancillary Information Facility

# Using Module Headers

January 2020



# Topics

---

Navigation and Ancillary Information Facility

- **Module\* Header Purpose**
- **FORTRAN Module Header Locations**
- **C Module Header Locations**
- **Icy Module Header Locations**
- **Mice Module Header Locations**
- **Examine a Typical Header**

\* “Module” = API, routine, subroutine, procedure, function

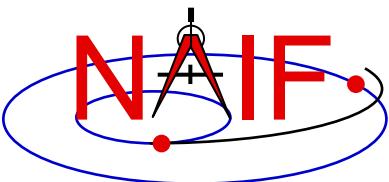


# Module Header Purpose

---

Navigation and Ancillary Information Facility

- **NAIF uses module “headers” to provide detailed information describing how to use the module**
  - In FORTRAN, C and MATLAB Toolkits the “headers” are comment blocks inserted in the source code
  - In IDL Toolkits, where there are (currently) no source code files, the “headers” exist as independent files
- **All Toolkit distributions include hyperlinked HTML versions of the module headers.**
  - All but ICY also include plain text versions
- **The next charts provide the header contents and locations**



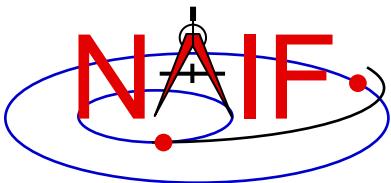
# Module Header Contents

## Navigation and Ancillary Information Facility

- Procedure or subroutine name
- Brief abstract
- Disclaimer (legalese required for JPL code)
- Required Reading (names of any related SPICE technical reference documents)
- Keywords (single relevant words; not really used)
- Argument type declarations, or Include files (for C and Fortran toolkits)
- Brief Input and Output descriptions
- Detailed Input descriptions
- Detailed Output descriptions
- Parameter definitions, if any
- Exceptions (what happens if a problem is detected)
- Descriptions of any files used
- Particulars (details about what the module does, how it works, any limitations)
- Code usage example(s)
- Restrictions in usage of the module
- Literature references
- Author
- Version
- Index entries (brief phrases used to generate entries for the Permuted Index document)
- Revision history (only in Fortran headers)

The source code goes here!

ICY and MICE headers contain only the items shown in blue;  
see the corresponding CSPICE header for full details.



# Fortran Module Header Locations

Navigation and Ancillary Information Facility

- **Plain text versions:**

- <[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:**

- <[path to SPICELIB](#)>/toolkit/doc/html/spicelib/index.html

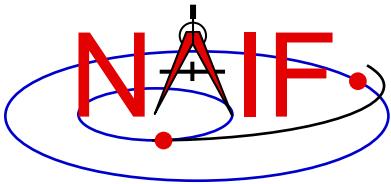


# C Module Header Locations

---

Navigation and Ancillary Information Facility

- Plain text versions:
  - <path to CSPICE>/cspice/src/cspice/<name>\_c.c
- HTML versions:
  - <path to CSPICE>/cspice/doc/html/cspice/index.html



# IDL Module Header Locations

---

Navigation and Ancillary Information Facility

- **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” header is minimal in some cases; the corresponding CSPICE header provides more detail**
  - A link to the corresponding CSPICE header is provided in the Icy header

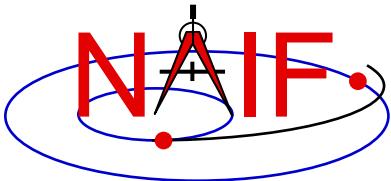


# Matlab Module Header Locations

---

Navigation and Ancillary Information Facility

- **Two sets of headers are provided**
  - Mice headers in HTML format:
    - » <path to Mice>/mice/doc/html/mice/index.html
    - » Also available using 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” header is minimal in some cases; the corresponding CSPICE header provides more detail**
  - A link to the corresponding CSPICE header is provided in the Mice header



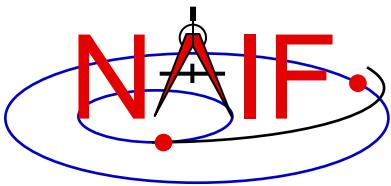
# Examine a Typical Header

Navigation and Ancillary Information Facility

- As example, look for and examine the headers for the modules named **spkezr** and **str2et**

FORTRAN	C	IDL (Icy)	MATLAB (Mice)
SPKEZR	spkezr_c	cspice_spkezr	cspice_spkezr
STR2ET	str2et_c	cspice_str2et	cspice_str2et

spkezr is the principal ephemeris access module  
str2et is a key time conversion module

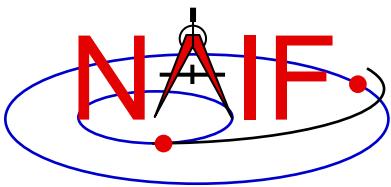


---

Navigation and Ancillary Information Facility

# Time Conversion and Time Formats

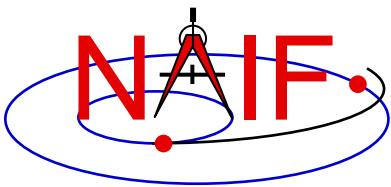
January 2020



# Time Systems and Kernels

Navigation and Ancillary Information Facility

- Time inputs to and outputs from user's programs are usually **strings** representing epochs in these three time systems:
  - Ephemeris Time (**ET**, also referred to as Barycentric Dynamical Time, **TDB**)
  - Coordinated Universal Time (**UTC**). This is the default for calendar strings.
  - Spacecraft Clock (**SCLK**)
- Time stamps in kernel files, and time inputs to and outputs from 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
    - » J2000 = 2000 Jan 1 12:00:00 TDB
  - Encoded Spacecraft Clock, expressed as clock ticks since the clock start
- SPICE provides routines to convert between these string and numeric representations.
- A time string used as an argument in a SPICE API must be provided in quotes.
  - Fortran, Matlab, IDL and Python: use single quotes
  - C and JNI: use double quotes



# Converting Time Strings

Navigation and Ancillary Information Facility

- UTC, TDB, or TDT (TT) String to numeric Ephemeris Time
  - STR2ET ( *string*, ET )
    - » Converts virtually any time string format known to the SPICE Time subsystem, excepting SCLK.
    - » Examples of acceptable string inputs:

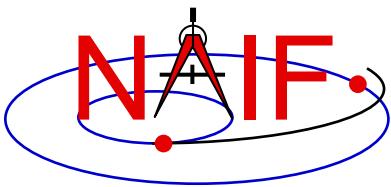
‘1996-12-18T12:28:28’	‘1978/03/12 23:28:59.29’	‘Mar 2, 1993 11:18:17.287 p.m. PDT’
‘1995-008T18:28:12’	‘1993-321//12:28:28.287’	
‘2451515.2981 JD’	‘ jd 2451700.05 TDB’	
‘1988-08-13, 12:29:48 TDB’	‘1992 June 13, 12:29:48 TDT’	

- » Requires the LSK kernel

These example inputs all use the single quote required by Fortran, IDL MATLAB and Python APIs. Use double quotes for C and JNI APIs.

- Spacecraft Clock String to numeric Ephemeris Time

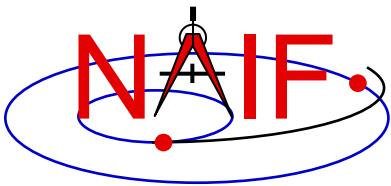
- SCS2E ( *scid*, *string*, ET )
  - » Converts SCLK strings consistent with SCLK parameters.
  - » Examples of acceptable clock string inputs:
    - ‘5/65439:18:513’ (VGR1)
    - ‘946814430.172’ (MRO)
    - ‘1/0344476949-27365’ (MSL)
  - » Requires a SCLK kernel and the LSK kernel



# Converting Numeric Times

Navigation and Ancillary Information Facility

- Numeric Ephemeris Time to a string, where the format is Calendar, DOY or Julian Date, and the time system is *UTC*, *TDB* or *TDT*
  - **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 the 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
- Numeric Ephemeris Time to Spacecraft Clock String
  - **SCE2S ( *scid*, *et*, *SCLKCH* )**
    - » Requires the LSK and a SCLK kernel
    - » Output SCLK string examples:
      - 1/05812:00:001 (Voyager 1 and 2)
      - 1/1487147147.203 (Cassini, MRO)
      - 1/0101519975.65186 (MEX, VEX, Rosetta)



# Use of Time Format Picture

Navigation and Ancillary Information Facility

## Example Time Strings and the Corresponding Format Pictures

### Common Time Strings

1999-03-21T12:28:29.702

1999-283T12:29:33

1999-01-12, 12:00:01.342 TDB

2450297.19942145 JD TDB

### Format Picture Used (*fmpic*)

YYYY-MM-DDTHR:MN:SC.###

YYYY-DOYTHR:MN:SC ::RND

YYYY-MM-DD, HR:MN:SC.### ::TDB TDB

JULIAND.##### ::TDB JD TDB

### Less Common Time Strings

465 B.C. Jan 12 03:15:23 p.m.

04:28:55 A.M. June 12, 1982

Thursday November 04, 1999

DEC 31, 15:59:60.12 1998 (PST)

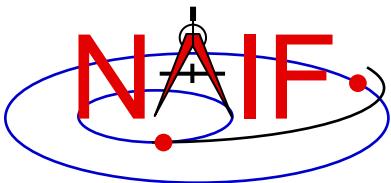
### Format Picture Used (*fmpic*)

YYYY ERA Mon DD AP:MN:SC ampm

AP:MN:SC AMPM Month DD, YYYY

Weekday Month DD, YYYY

MON DD, HR:MN:SC.## YYYY (PST) ::UTC-8

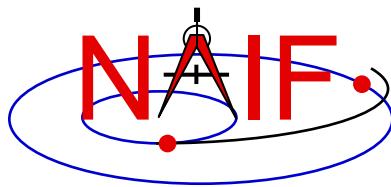


# Additional Time Conversions

Navigation and Ancillary Information Facility

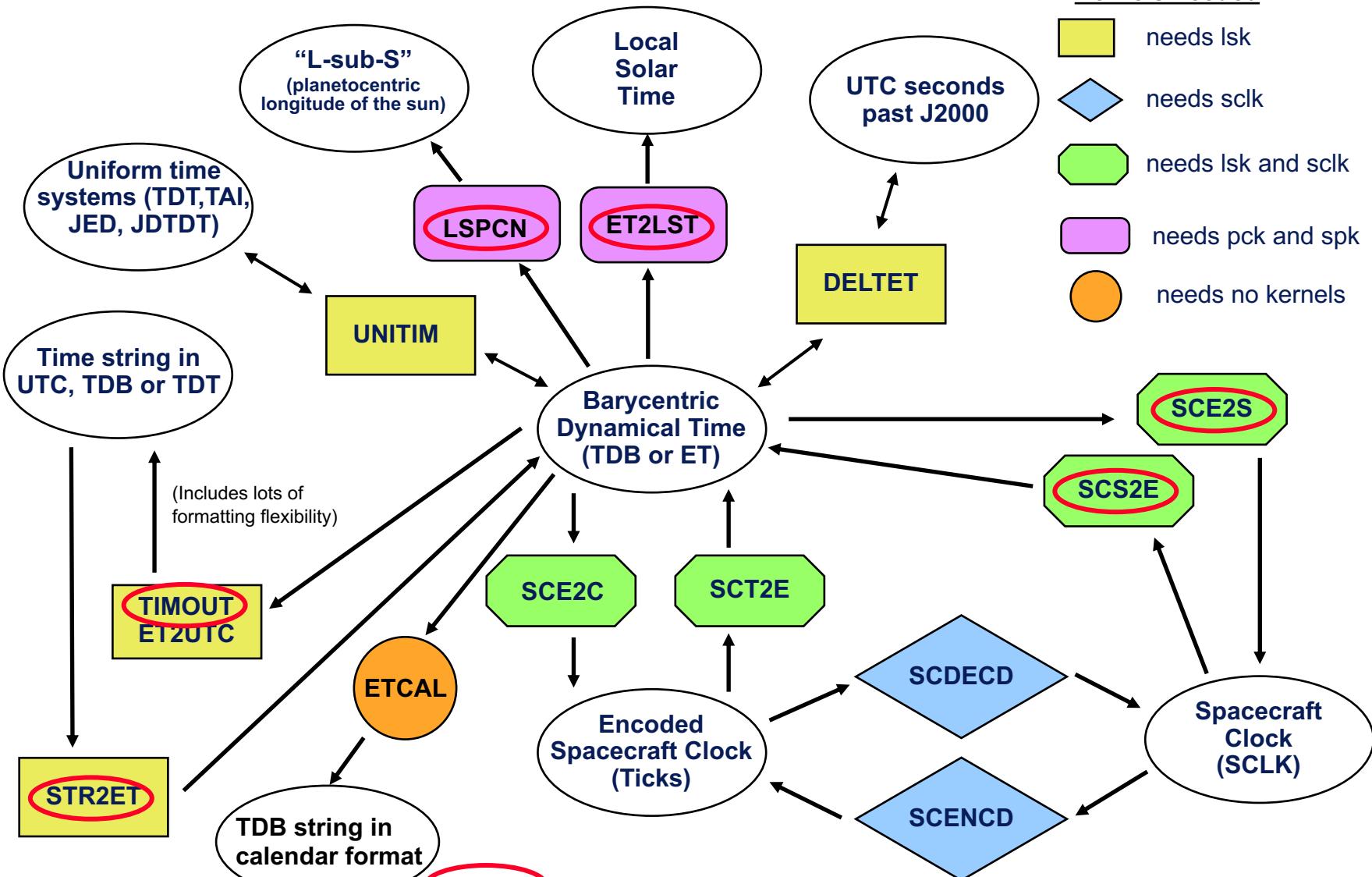
- Numeric Ephemeris Time to Local Solar Time String
  - ET2LST( *et, body, long, type, HR, MN, SC, TIME, AMPM* )
    - » Requires SPK (to compute *body* position relative to the Sun) and PCK (to compute *body* rotation) kernels
- Numeric Ephemeris Time to planetocentric longitude of the Sun (*Ls*)
  - LS = LSPCN (*body, et, abcorr* )
    - » While *Ls* is not a time system, it is frequently used to determine *body* season for a given epoch
      - LS =  $0^\circ$  , Spring
      - LS =  $90^\circ$  , Summer
      - LS =  $180^\circ$  , Autumn
      - LS =  $270^\circ$  , Winter
    - » The *Ls* calculation requires SPK and PCK kernels

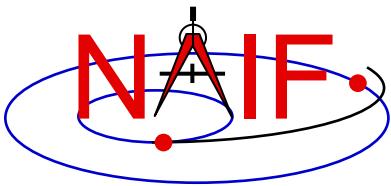
} For the northern hemisphere



# Principal Time System Interfaces

Navigation and Ancillary Information Facility





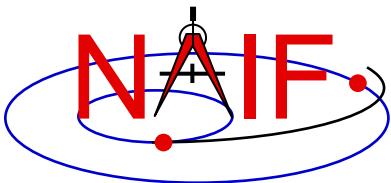
---

Navigation and Ancillary Information Facility

# Leapseconds and Spacecraft Clock Kernels

## LSK and SCLK

January 2020

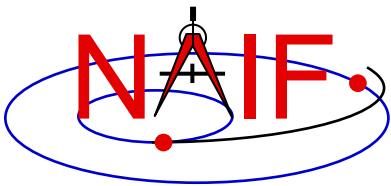


# Topics

---

Navigation and Ancillary Information Facility

- **Kernels Supporting Time Conversions**
  - LSK
  - SCLK
- **Forms of SCLK Time Within SPICE**
- **Backup**



# SPICE Time Conversion Kernels

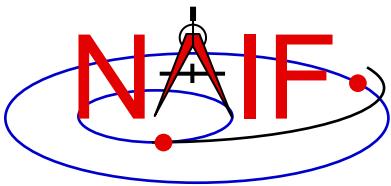
---

Navigation and Ancillary Information Facility

In most cases one or two kernel files are needed to perform conversions between supported time systems.

- **LSK** - The leapseconds kernel is used in conversions between ephemeris time (ET/TDB) and Coordinated Universal Time (UTC).
- **SCLK** - The spacecraft clock kernel is used in conversions between spacecraft clock time (SCLK) and ephemeris time (ET/TDB).
  - It's possible there could be two or more clocks associated with a given spacecraft.

Ephemeris Time, ET and Barycentric Dynamical Time, TDB, are the same

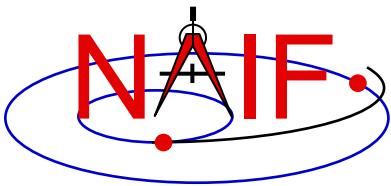


# The Leapseconds Kernel (LSK)

Navigation and Ancillary Information Facility

**The leapseconds kernel contains a tabulation of all the leapseconds that have occurred, plus additional terms.**

- **Used in ET↔UTC and in ET↔SCLK conversions.**
  - Subroutines using LSK: STR2ET, TIMOUT, ET2UTC, etc.
  - Utility programs using LSK: *spkmerge*, *chronos*, *spacit*, etc.
- **Use FURNSH to load it.**
- **NAIF updates the LSK when a new leap second is announced by the International Earth Rotation Service (IERS).**
  - The latest LSK file is always available from the NAIF server.
    - » The latest is LSK always the best one to use.
  - Announcement of each new LSK is typically made months in advance using the “spice\_announce” system.
    - » [http://naif.jpl.nasa.gov/mailman/listinfo/spice\\_announce](http://naif.jpl.nasa.gov/mailman/listinfo/spice_announce)
  - New LSKs take effect ONLY on January 1<sup>st</sup> and July 1<sup>st</sup>



# LSK File Example

---

Navigation and Ancillary Information Facility

KPL/LSK

. . . <comments> . . .

\begindata

DELTET/DELTA\_T\_A = 32.184  
DELTET/K = 1.657D-3  
DELTET/EB = 1.671D-2  
DELTET/M = ( 6.239996D0 1.99096871D-7 )

DELTET/DELTA\_AT = ( 10, @1972-JAN-1  
11, @1972-JUL-1  
12, @1973-JAN-1  
13, @1974-JAN-1  
14, @1975-JAN-1

. . . <more leapsecond records> . . .

35, @2012-JUL-1  
36, @2015-JUL-1  
37, @2017-JAN-1 )

\begintext

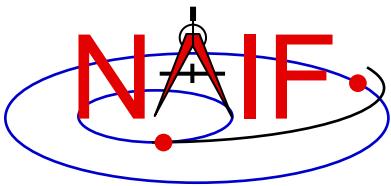


# Out of Date LSKs

---

Navigation and Ancillary Information Facility

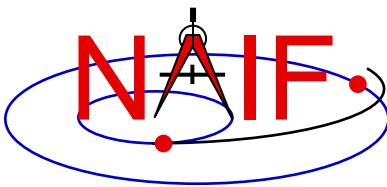
- An out-of-date leapseconds kernel can be used successfully for conversions that occur at epochs **prior** to the epoch of the first missing leapsecond.
  - Any conversions of epochs occurring after the epoch of a missing leapsecond will introduce inaccuracies in multiples of one second per missed leapsecond.
- Using the latest leapseconds kernel to perform conversions at epochs more than six months ahead of the last leapsecond listed may result in an error if, later on, a new leapsecond is declared for a time prior to the epochs you processed.



# The Spacecraft Clock Kernel (SCLK)

Navigation and Ancillary Information Facility

- The spacecraft clock kernel is required by Toolkit utilities and routines that utilize SCLK time.
  - For example, the SPICE CK subsystem makes heavy use of spacecraft clock time.
- Use FURNSH to load it.
- Ensure you have the correct version of the SCLK file for your spacecraft since this kernel may be updated rather frequently.
  - SCLK files are usually maintained on a flight project server.
    - » For JPL operated missions they can always be found on the NAIF server as well.
  - When using a CK, “correct SCLK” means compatible with that CK.
    - » For reconstructed CKs, this is most likely the latest version of the SCLK.
    - » For “predict” CKs, this is probably the SCLK kernel used when the CK was produced.



# SCLK File Example

## Navigation and Ancillary Information Facility

KPL/SCLK

```
. . . <comments> . . .

\begin{data

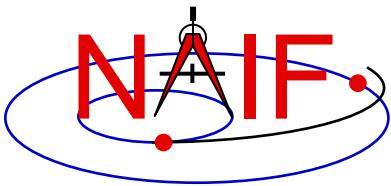
SCLK_KERNEL_ID          = ( @2009-12-07/18:03:04.00 )
SCLK_DATA_TYPE_74        = ( 1 )
SCLK01_TIME_SYSTEM_74   = ( 2 )
SCLK01_N_FIELDS_74      = ( 2 )
SCLK01_MODULI_74         = ( 4294967296 256 )
SCLK01_OFFSETS_74        = ( 0 0 )
SCLK01_OUTPUT_DELIM_74  = ( 1 )

SCLK_PARTITION_START_74  = ( 0.000000000000E+00
    . . . <more partition start records> . . .
    2.4179319500800E+11 )

SCLK_PARTITION_END_74    = ( 2.0692822929300E+11
    . . . <more partition end records> . . .
    1.0995116277750E+12 )

SCLK01_COEFFICIENTS_74 = (
    0.000000000000E+00    -6.3119514881600E+08    1.000000000000E+00
    1.2098765056000E+10   -5.8393434781600E+08    1.000000000000E+00
    . . . <more coefficient records> . . .
    2.4179319365000E+11   3.1330950356800E+08    9.999999750000E-01 )

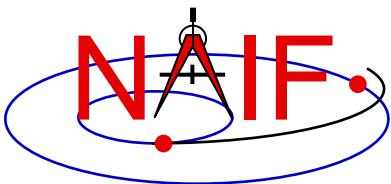
\begin{text
```



# Forms of SCLK Time Within SPICE

Navigation and Ancillary Information Facility

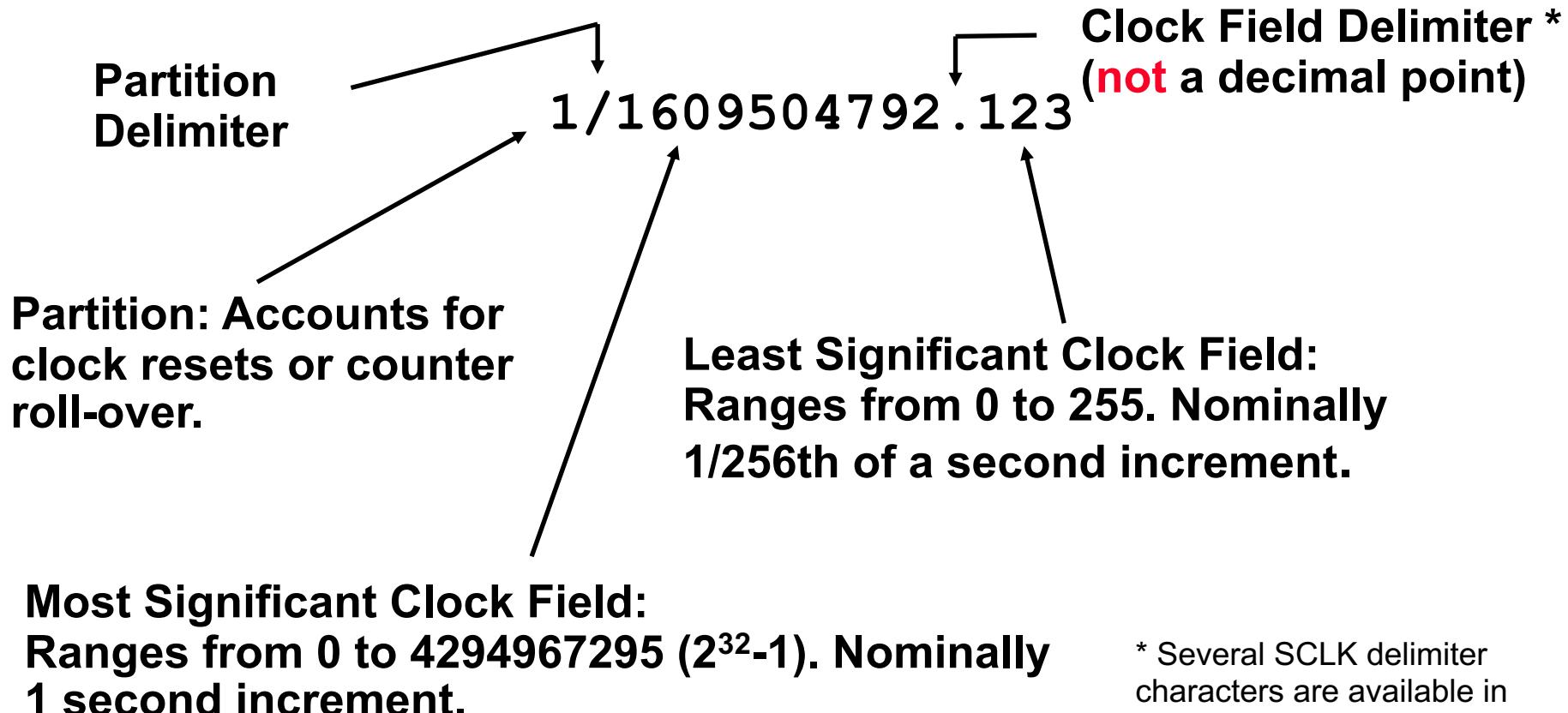
- SCLK time in SPICE is represented in two different ways:
  - a character string
  - a double precision number called “ticks”
- A SCLK character string is composed of one or more cascading integer numbers – similar to a digital clock.
  - This form is derived from clock values represented by sets of bits or bytes found in downlinked telemetry.
- A SCLK value encoded as a double precision number (called “ticks”) is used within SPICE because it’s easy to convert this to other time systems, such as ephemeris time.



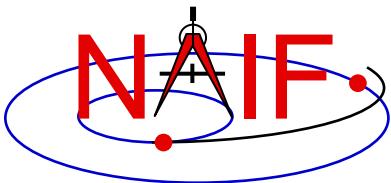
# Sample SCLK String

Navigation and Ancillary Information Facility

The Cassini orbiter SCLK time **string** consists of three fields separated by delimiters.



\* Several SCLK delimiter characters are available in SPICE. See “SCLK Required Reading” for details.



# What is a Partition?

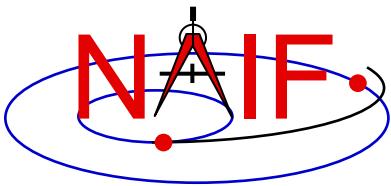
---

Navigation and Ancillary Information Facility

**1/1609504792.123**

**The portion of the SCLK string circled above indicates the partition to which the remaining portion of the string is related.**

- A partition is a NAIF-created construct to handle spacecraft clock rollovers or resets.
- SCLK strings not having a partition number are treated as belonging to the first partition in which they occur.

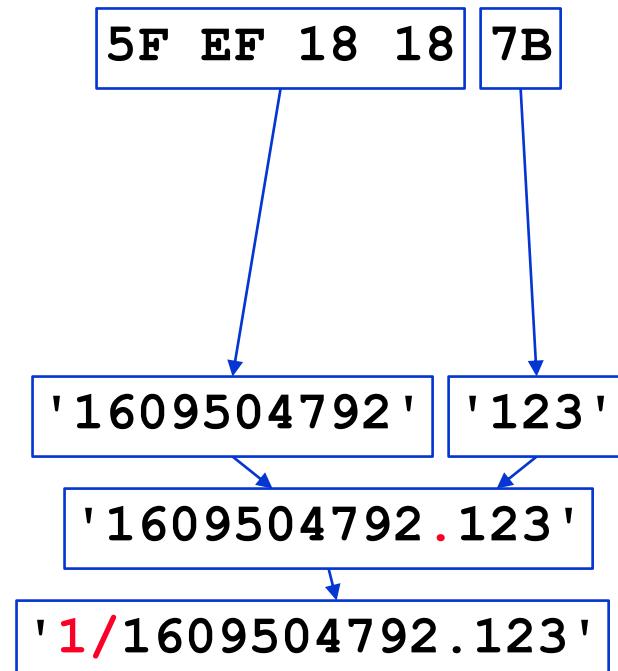


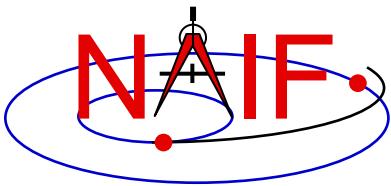
# Constructing a SCLK String

Navigation and Ancillary Information Facility

Usually SCLK tags in raw telemetry are represented by sets of bits or bytes. Such tags must be converted to SCLK strings used in SPICE. This is an example of how it is done for the sample CASSINI SCLK string from previous slides.

- Start with a 5-byte CASSINI TLM SCLK
  - The first four bytes are an unsigned integer representing seconds
  - The last byte is an unsigned byte representing fractional seconds (as a count of 1/256 second ticks)
- Convert integer and fractional seconds to two strings
- Concatenate strings together using a recognized delimiter ('.', ':', etc)
- Add the partition number and delimiter
  - Optional; for most modern missions it may be omitted (not so for Chandrayaan-1 and MESSENGER)





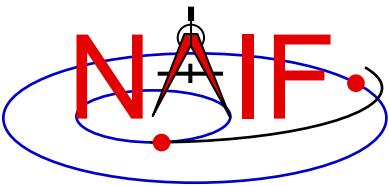
# Encoded SCLK (Ticks)

---

Navigation and Ancillary Information Facility

**The representation of SCLK time in the SPICE system is a double precision encoding of a SCLK string.**

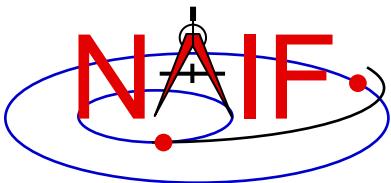
- **Encoded spacecraft clock values represent “ticks since spacecraft clock start.”**
  - The time corresponding to tick “0” is mission dependent and does not necessarily relate to launch time. It is often an arbitrary epoch occurring before launch.
- **A tick is the smallest increment of time that a spacecraft clock measures.**
  - For example, in the case of the Cassini orbiter this is nominally 1/256th of a second.
- **Encoded SCLK increases continuously independent of leapseconds, clock resets, and clock rollovers.**



# Additional Info on LSK and SCLK

Navigation and Ancillary Information Facility

- For more information about LSK, SCLK, and time conversions, look at the following documents
  - Time Required Reading
  - SCLK Required Reading
  - Time tutorial (at the end it has a nice graphic depicting time APIs)
  - Most Useful SPICELIB Routines
  - headers for the routines mentioned in this tutorial
  - CHRONOS User's Guide
  - Porting\_kernels tutorial
- Related documents
  - Kernel Required Reading
  - CK Required Reading



# SCLK Interface Routines

Navigation and Ancillary Information Facility

## Convert SCLK times using the following routines

**SCS2E** (SC, SCLKCH, ET)

(SCLK String $\Rightarrow$  ET)

**SCE2S** (SC, ET, SCLKCH)

(ET $\Rightarrow$  SCLK String)

**SCT2E** (SC, SCLKDP, ET)

(Encoded SCLK $\Rightarrow$  ET)

**SCE2C**<sup>1</sup> (SC, ET, SCLKDP)

(ET $\Rightarrow$  Continuous Encoded SCLK)

**SCE2T** (SC, ET, SCLKDP)

(ET $\Rightarrow$  Discrete Encoded SCLK)

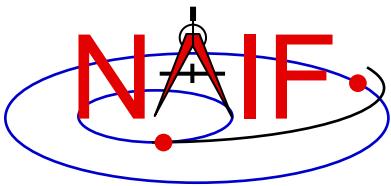
**SCENCD** (SC, SCLKCH, SCLKDP)

(Encode SCLK string to number)

**SCDECD** (SC, SCLKDP, SCLKCH)

(Decode SCLK number to string)

<sup>1</sup> Use SCE2C (not SCE2T) for C-kernel data access.

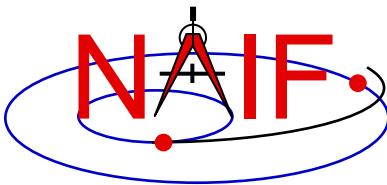


# Backup

---

Navigation and Ancillary Information Facility

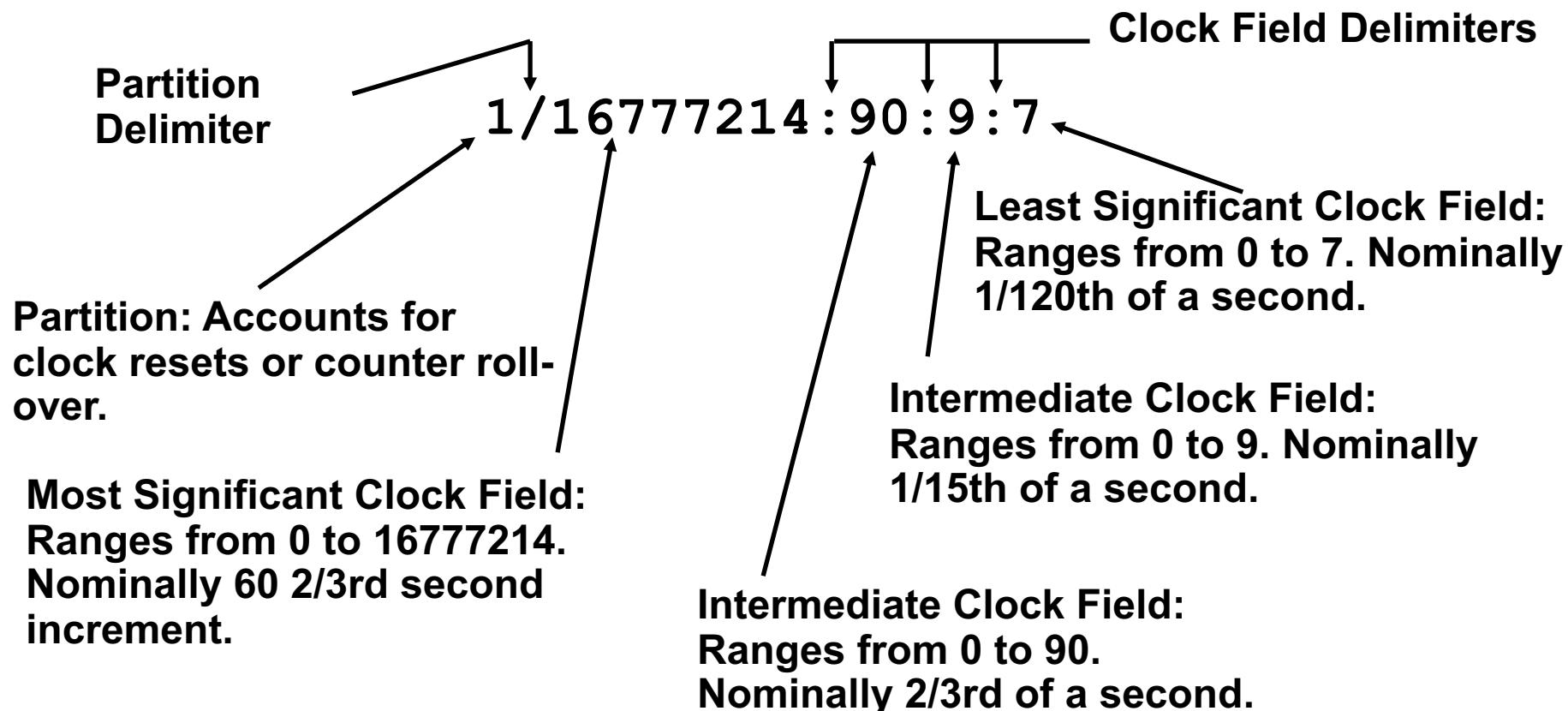
- **Examples of SCLK strings**
- **Dates of "recent" leapseconds**

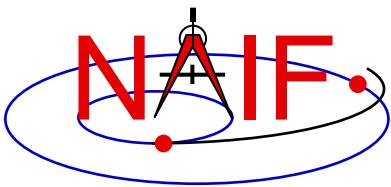


# Sample Galileo SCLK String

Navigation and Ancillary Information Facility

The Galileo spacecraft SCLK time **string** consists of five fields separated by delimiters.





# More Sample SCLK Strings

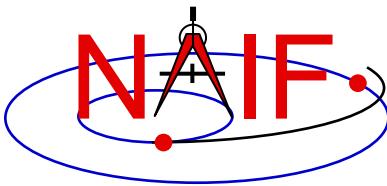
Navigation and Ancillary Information Facility

The following are examples of SCLK strings\* from missions using SPICE.

- |                                       |  |  |
|---------------------------------------|--|--|
| • <b>Cassini</b><br>1/1334314108.134  | • <b>MPF</b><br>1/559627908.058          | • <b>Viking 1&amp;2</b><br>1/32233616        |
| • <b>DS1</b><br>1/67532406.010        | • <b>Mariner 9</b><br>1/11542909         | • <b>Voyager 1&amp;2</b><br>1/05812:00:001   |
| • <b>Galileo</b><br>1/16777214:90:9:7 | • <b>Mars Odyssey</b><br>1/687231994.091 | • <b>Mars Express</b><br>1/0090979196.29713  |
| • <b>Genesis</b><br>1/666230496.204   | • <b>NEAR</b><br>1/40409721942           | • <b>Venus Express</b><br>1/0033264000.50826 |
| • <b>MGS</b><br>1/655931592.103       | • <b>Stardust</b><br>1/697451990.042     | • <b>Rosetta</b><br>1/0101519975.65186       |

\* When clock strings are used as arguments in modules they must be contained in quotes:

- Single quotes for Fortran
- Double quotes for C
- Single quotes for IDL and MATLAB



# Dates of New Leapseconds<sup>+</sup>

---

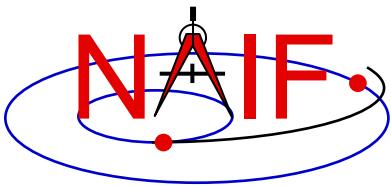
Navigation and Ancillary Information Facility

1972-JAN-1  
1972-JUL-1  
1973-JAN-1  
1974-JAN-1  
1975-JAN-1  
1976-JAN-1  
1977-JAN-1  
1978-JAN-1  
1979-JAN-1  
1980-JAN-1  
1981-JUL-1  
1982-JUL-1  
1983-JUL-1  
1985-JUL-1  
1988-JAN-1  
1990-JAN-1  
1991-JAN-1  
1992-JUL-1  
1993-JUL-1  
1994-JUL-1  
1996-JAN-1  
1997-JUL-1  
1999-JAN-1  
2006-JAN-1  
2009-JAN-1  
2012-JUL-1  
2015-JUL-1  
2017-JAN-1

- **New leapseconds become effective only on January 1<sup>st</sup> or July 1<sup>st</sup>.**
- **NAIF announces every new leapsecond using the "spice\_announce" Mailman system**
- **NAIF provides a new leapseconds kernels\* (LSK) several months before the new leapsecond becomes effective**

<sup>+</sup> Current as of January 1, 2020

\* NAIF provides both Linux/OSX and Windows versions of the LSK

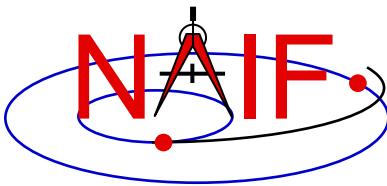


---

Navigation and Ancillary Information Facility

# An Overview of Reference Frames and Coordinate Systems in the SPICE Context

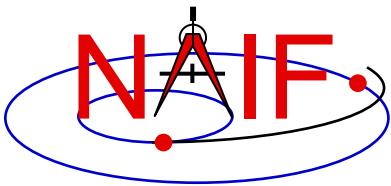
January 2020



# Purpose of this Tutorial

Navigation and Ancillary Information Facility

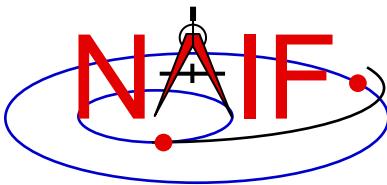
- This tutorial provides an overview of reference frames and coordinate systems.
  - It contains conventions specific to SPICE.
- Details about the SPICE Frames Subsystem are found in other tutorials and one document:
  - FK (tutorial)
  - Using Frames (tutorial)
  - Dynamic Frames (advanced tutorial)
  - Frames Required Reading (technical reference)
- Details about SPICE coordinate systems are found in API module headers for coordinate conversion routines.



# A Challenge

Navigation and Ancillary Information Facility

- Next to “time,” the topics of reference frames and coordinate systems present some of the largest challenges to documenting and understanding observation geometry. Contributing factors are ...
  - differences in definitions, lack of concise definitions, and special cases
  - evolution of the frames subsystem within SPICE
  - the substantial frames management capabilities within SPICE
- NAIF hopes this tutorial will provide some clarity on these subjects within the SPICE context.
  - Definitions and terminology used herein may not be consistent with those found elsewhere.

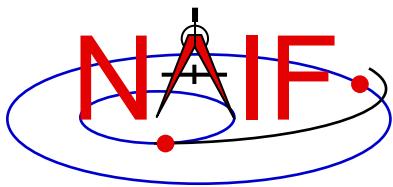


# SPICE Definitions

---

Navigation and Ancillary Information Facility

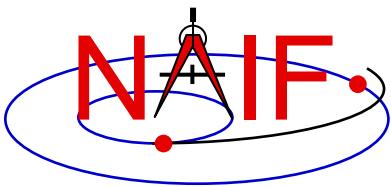
- The definitions below are used within SPICE.
- A **reference frame** (or simply “**frame**”) is specified by an ordered set of three mutually orthogonal, possibly time dependent, unit-length direction vectors.
  - A reference frame has an associated center.
  - In some documentation external to SPICE, this is called a “coordinate frame.”
- A **coordinate system** specifies a mechanism for locating points within a reference frame.
- When producing or using state (position and velocity) or orientation (pointing) data, one needs to understand both the reference frame and the coordinate system being used.



---

**Navigation and Ancillary Information Facility**

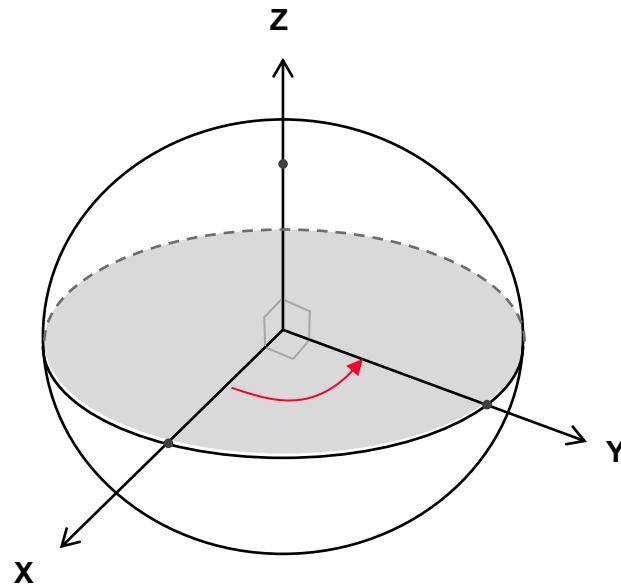
# **Reference Frames**

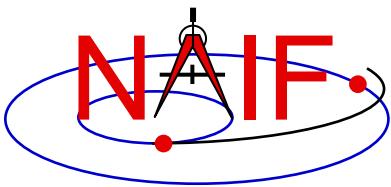


# Reference Frame Conventions

Navigation and Ancillary Information Facility

- All reference frames used within SPICE are right handed: this means  $X \times Y = Z$



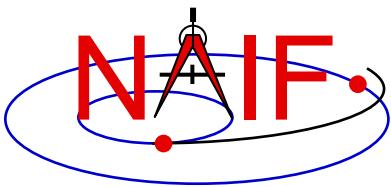


# Reference Frame Center

Navigation and Ancillary Information Facility

- A reference frame's center must be a SPICE ephemeris object whose location is coincident with the origin  $(0, 0, 0)$  of the frame.
  - The center of any inertial frame is ALWAYS the solar system barycenter.\*
  - The center of a body-fixed frame is the center of the body.
    - » “Body” means a natural body: sun, planet, satellite, comet, asteroid.
    - » The location of the “body” center is specified using an SPK file.
  - The center of a topocentric, spacecraft or instrument frame is also an object for which the location is specified by an SPK file.
- A frame's center may play a role in specification of states.
  - The location of the origin cancels out when doing vector subtraction, but the center is used in computing light time to the center of any non-inertial frame being used

\*True even for inertial frames associated with accelerated bodies, such as the MARSIAU frame.

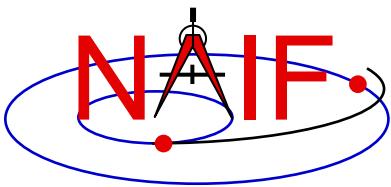


# Types of Reference Frames - 1

Navigation and Ancillary Information Facility

- **Inertial**

- Non-rotating with respect to stars
- Non-accelerating origin
  - » Velocity is typically non-zero, but acceleration is negligible
- Examples:
  - » J2000 (also known as EME 2000, and is actually ICRF)
  - » ECLIPJ2000



# Types of Reference Frames - 2

Navigation and Ancillary Information Facility

- **Non-Inertial**

- Accelerating, including by rotation
- Examples

- » **Body-fixed**

- Associated with a natural body (e.g. planets, satellites)

- » **Topocentric**

- Associated with an object on or near the surface of a natural body (e.g. DSN station, rover)

- » **Spacecraft**

- Associated with the main spacecraft structure (the “bus”)

- » **Instrument**

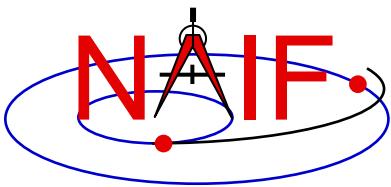
- One or more frames are usually associated with each instrument
  - Also applicable to a spacecraft antenna, solar array, etc.

- » **Dynamic**

- A special family of frames unique to SPICE
  - These have time-dependent orientation

- But this category does not include frames for which the orientation is provided using a C-kernel (CK) or a PC-kernel (PCK)

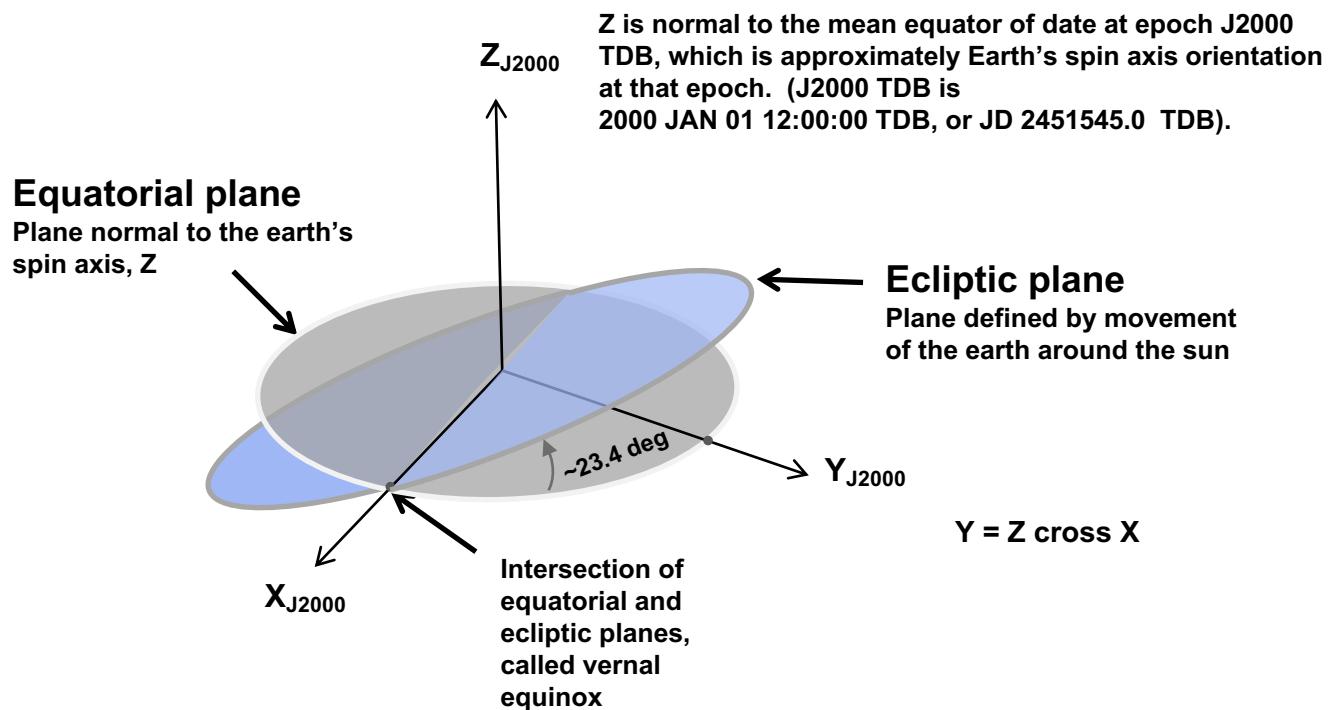
CK ≈ spacecraft orientation; PCK ≈ natural body orientation



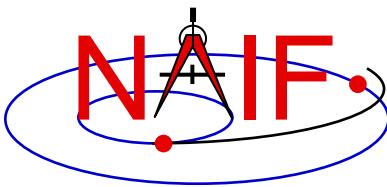
# The J2000 Inertial Frame

Navigation and Ancillary Information Facility

- The J2000\* (aka EME2000) frame definition is based on the earth's equator and equinox, determined from observations of planetary motions, plus other data.



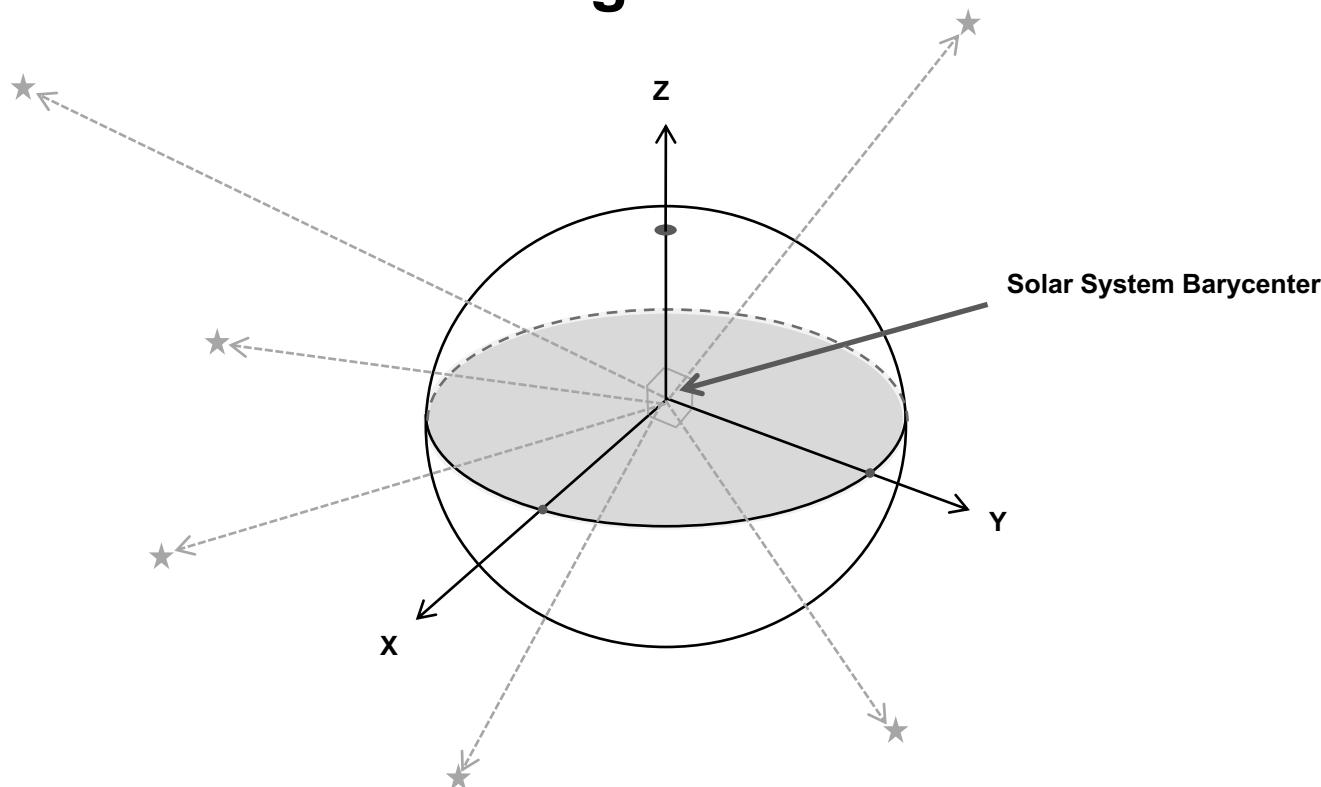
\*Caution: The name "J2000" is also used to refer to the zero epoch of the ephemeris time system (ET, also known as TDB).



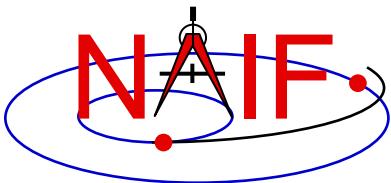
# The ICRF Inertial Frame

Navigation and Ancillary Information Facility

- The ICRF\* frame is defined by the adopted locations of 295 extragalactic radio sources.



ICRF = International Celestial Reference Frame

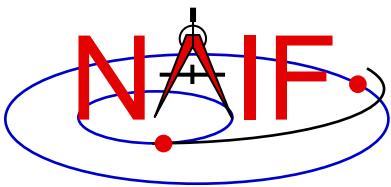


# J2000 versus ICRF

---

Navigation and Ancillary Information Facility

- **The realization of ICRF was made to coincide almost exactly with the J2000 frame.**
  - The difference is very small—a rotation of less than 0.1 arc second.
  - **These two frames are considered the same in SPICE.**
    - » In reality, any SPICE data said to be referenced to the J2000 frame are actually referenced to the ICRF frame.
    - » For historical and backwards compatibility reasons, only the name “J2000” is recognized by SPICE software as a frame name—not “ICRF.”
  - No transformation is required to convert SPICE state vectors or orientation data from the J2000 frame to the ICRF.



# Body-fixed Frames

Navigation and Ancillary Information Facility

- **Body-fixed frames are tied to a named body and rotate with it**

- Specifications for the most common body-fixed frames, those for the sun, the planets, many satellites, and a few asteroids and comets, are hard-coded in SPICE software

- » Frame name style is “IAU\_body name”

- Examples: IAU\_MARS, IAU\_SATURN

- » To see all such names, see:

- Frames Required Reading document, or
    - Latest generic PCK file

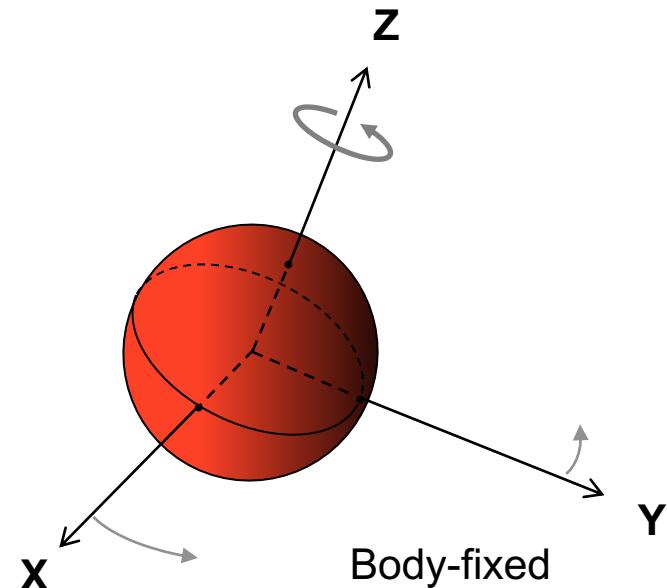
- The rotation state (the orientation at time  $T$ ) is usually determined using a SPICE text PCK containing data published by the IAU

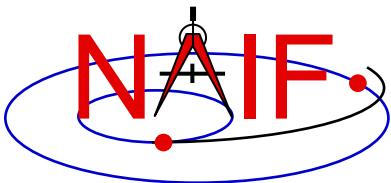
- » The earth and moon are special cases!

- IAU\_EARTH and IAU\_MOON both exist but generally should NOT be used

- See the SPICE tutorial named “lunar-earth\_pck-fk” for the best frames to be used for those bodies

- On very rare occasions a CK is used to provide a body’s rotation state



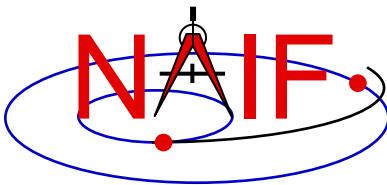


# A Caution for Mars

---

Navigation and Ancillary Information Facility

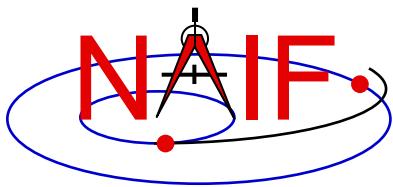
- **The body-fixed frame for Mars is named IAU\_MARS**
  - This follows the SPICE naming standard for such frames
- **However, there also exists in SPICE an inertial frame associated with Mars, named “MARSIAU”**
  - This frame was defined 20 years ago based on old Mars rotation constants, for use by the Mars Observer and Mars Global Surveyor projects
  - This frame has NO relationship to the similarly sounding IAU\_MARS frame, other than that they both relate to Mars



# Spacecraft and Instrument Frames

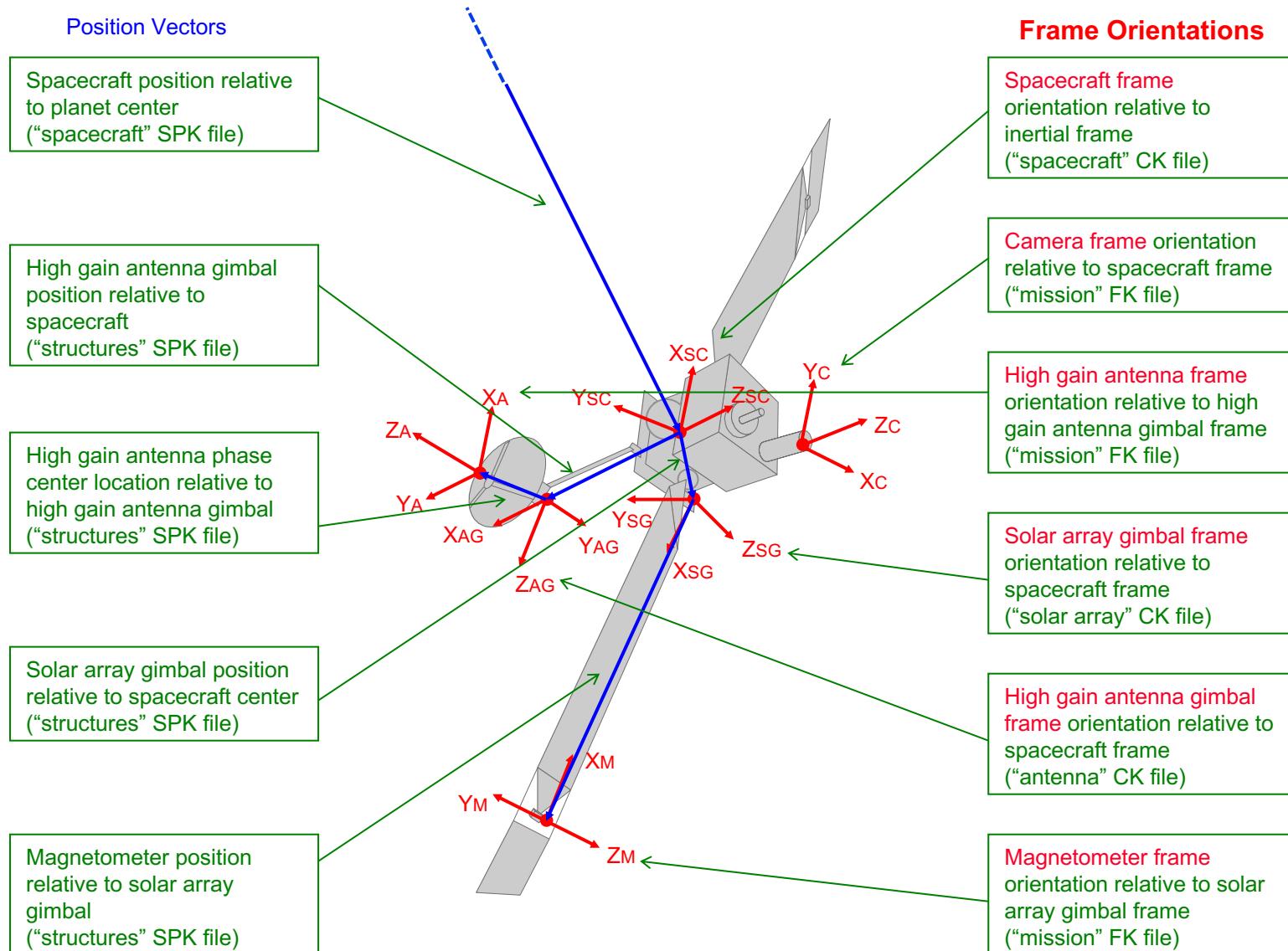
Navigation and Ancillary Information Facility

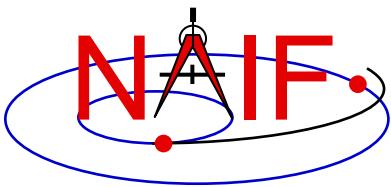
- **Defined for spacecraft, and items attached to a spacecraft, such as antennas, solar arrays, scan platforms, instruments and moving parts of an instrument (e.g. a scanning mirror)**
- **For those frames that are time varying (“moving”), the frame name is usually defined in an FK and the frame orientation data are usually provided by a CK**
- **For those frames that are not moving (what we call “fixed offset”) both the frame name and the actual data defining the fixed orientation of the frame are provided in an FK**



# Some Examples of Spacecraft and “Instrument” Frames

Navigation and Ancillary Information Facility

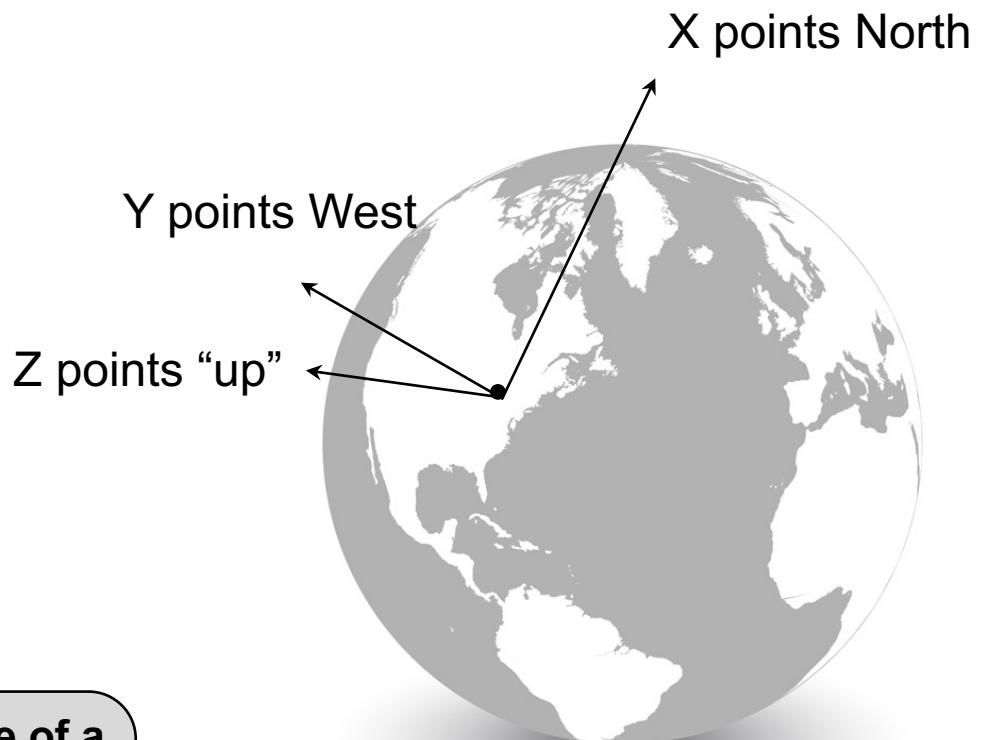




# Topocentric Frames

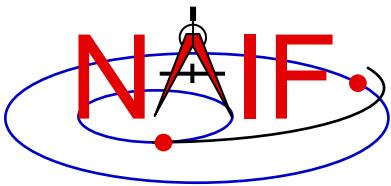
Navigation and Ancillary Information Facility

- **Topocentric frames are located at or near to a body's surface**
- **One axis is normal to a reference spheroid, or parallel to the gravity gradient\***
- **Examples: frames defined for telecommunications stations, or for landers or rovers**



The graphic illustrates one example of a topocentric frame. There is not a standard definition—for example, the z-axis could point down, the x-axis North, and the y-axis East.

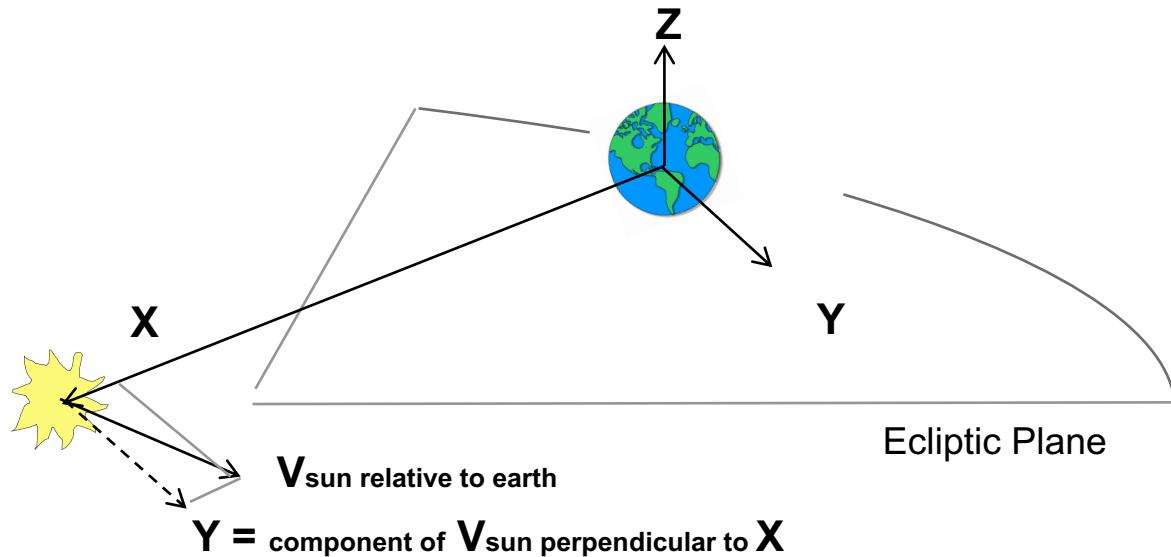
\*SPICE tools always have the “up” or “down” axis being normal to the spheroid. But one could use external data to determine the local gravity gradient and construct a frame based on that.

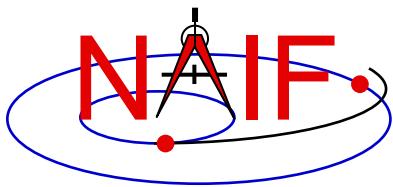


# Dynamic Frames

Navigation and Ancillary Information Facility

- In a dynamic frame the orientation changes with time
  - Families: Two-vector, Euler, and Of-date (refer to Dynamic Frames tutorial)
  - This category excludes frames for which the orientation is determined by a PCK or CK
  - Example of a two-vector dynamic frame: Geocentric Solar Ecliptic (GSE)
    - » X = earth – sun vector
    - » Y = component of the sun's velocity perpendicular to X
    - » Z = X cross Y

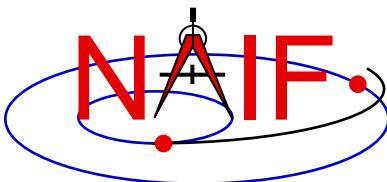




---

Navigation and Ancillary Information Facility

# Coordinate Systems

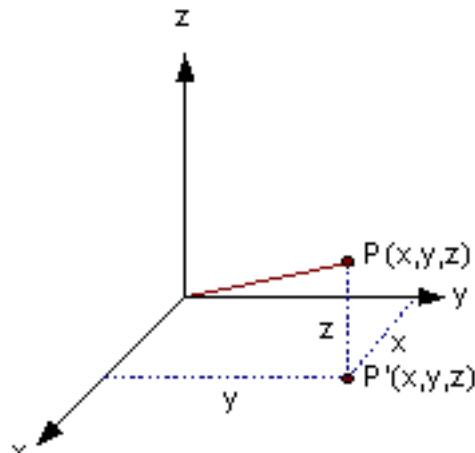


# SPICE Coordinate Systems

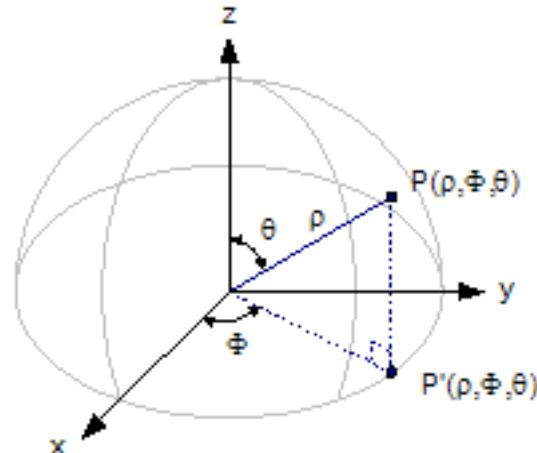
Navigation and Ancillary Information Facility

- A coordinate system specifies the method used to locate a point within a particular reference frame.

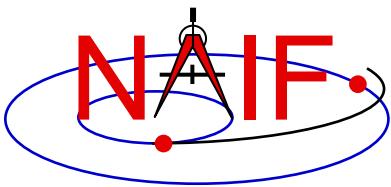
Two examples of coordinate systems used to locate point “P”



Rectangular or Cartesian coordinates:  
X, Y, Z



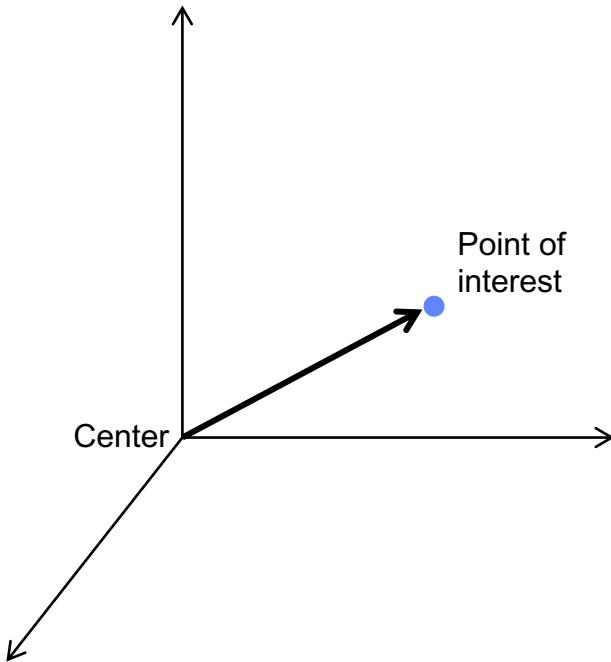
Spherical coordinates:  
 $\Phi, \theta, \rho$



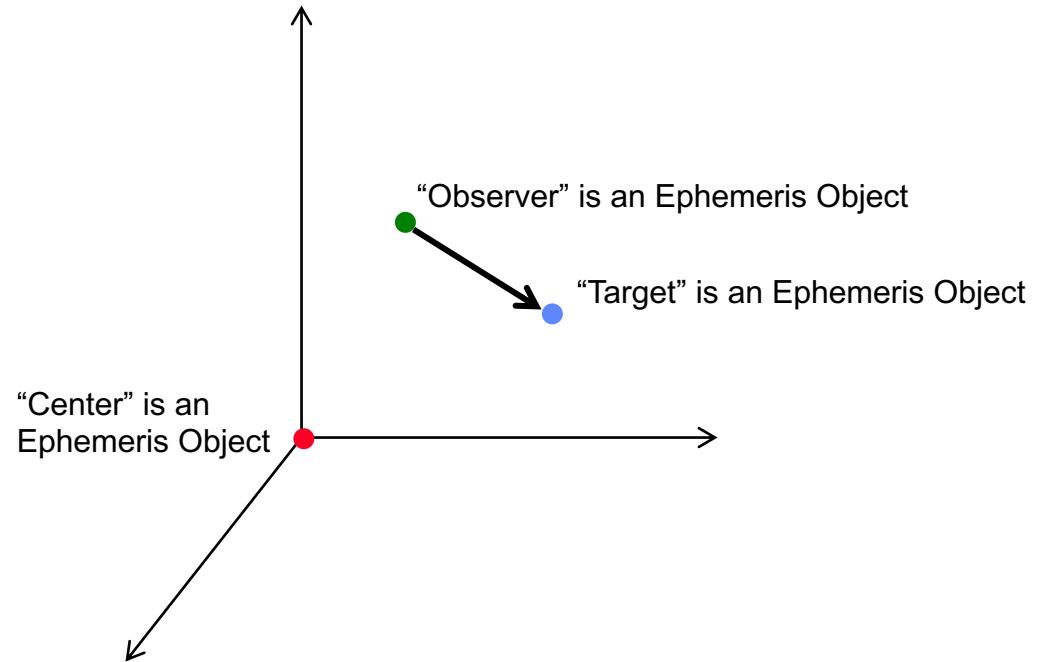
# Specifying Positions

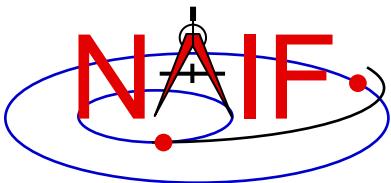
Navigation and Ancillary Information Facility

## Common Style



## SPICE Style

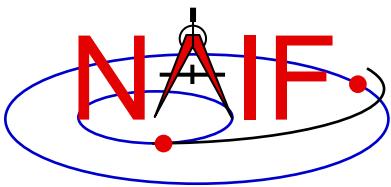




# Many Coordinate Systems Used

Navigation and Ancillary Information Facility

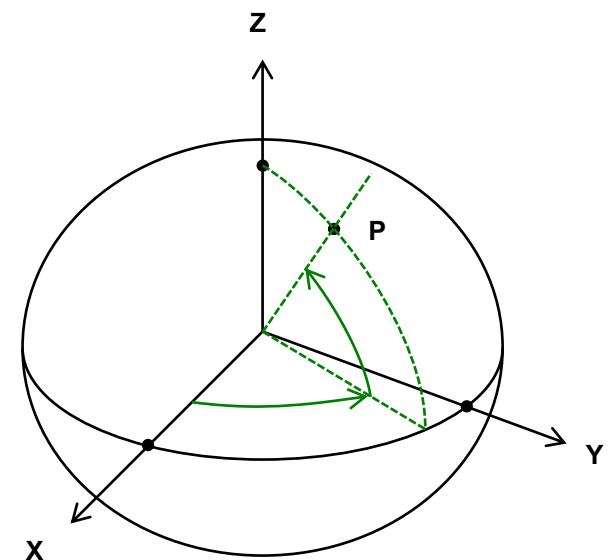
- In the Planetary Science discipline there are a number of coordinate systems in use, just as there are quite a few reference frames in use.
- Some of these coordinate systems have well accepted standard definitions, while others are anything but standard.
  - This means data producers and especially data users need to pay close attention to what they are doing!



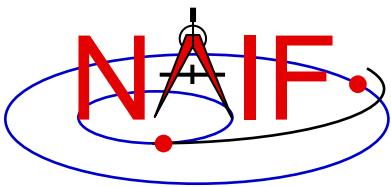
# Planetocentric Coordinate System

Navigation and Ancillary Information Facility

- For planets and their satellites the +Z axis (+90 latitude) always points to the north side of the invariable plane (the plane whose normal vector is the angular momentum vector of the solar system)
  - Planetocentric longitude increases positively eastward (-180 to +180)
  - Planetocentric latitude increases positively northward (-90 to +90)
- Dwarf planets\*, asteroids and comets spin in the right hand sense about their “positive pole.”
  - What the IAU now calls the “positive pole” is still referred to as the “north pole” in SPICE documentation.
  - The “positive pole” may point above or below the invariable plane of the solar system (see above).
  - This revision by the IAU Working Group (2006) inverts what had been the direction of the north pole for Pluto, Charon and Ida.
- Toolkit planetocentric APIs:
  - LATREC, RECLAT, DRDLAT, DLATDR, XFMSTA



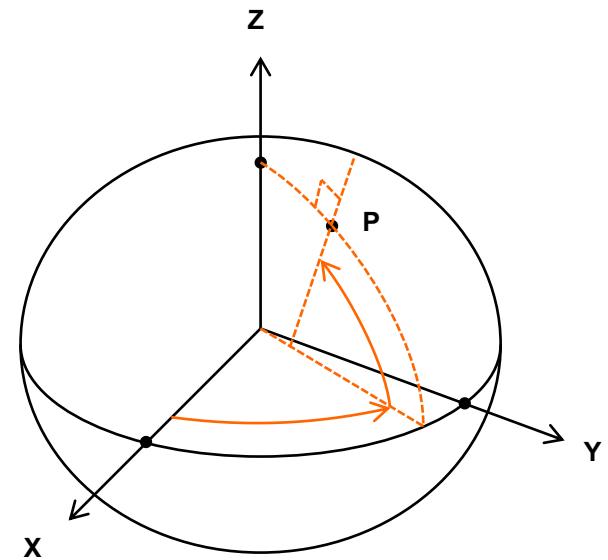
\*The dwarf planets are: Ceres, Eris, Haumea, Makemake, Pluto

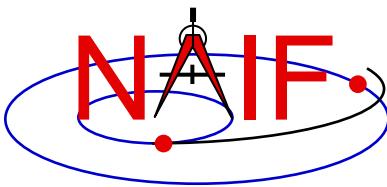


# Planetodetic Coordinate System

Navigation and Ancillary Information Facility

- **Planetodetic longitude is the same as planetocentric longitude**
  - Increases positively eastward (-180 to +180)
- **Planetodetic latitude**
  - Tied to a reference ellipsoid
  - For a point, P, on a reference ellipsoid, the angle measured from the X-Y plane to the surface normal at the point of interest. For points not on the ellipsoid, equals latitude at the nearest point on the reference ellipsoid
  - Increases positively northward (-90 to +90)
- **Toolkit planetodetic APIs are:**
  - GEOREC, RECGEO, DRDGEO, DGEODR, XFMSTA





# Planetographic Coordinate System

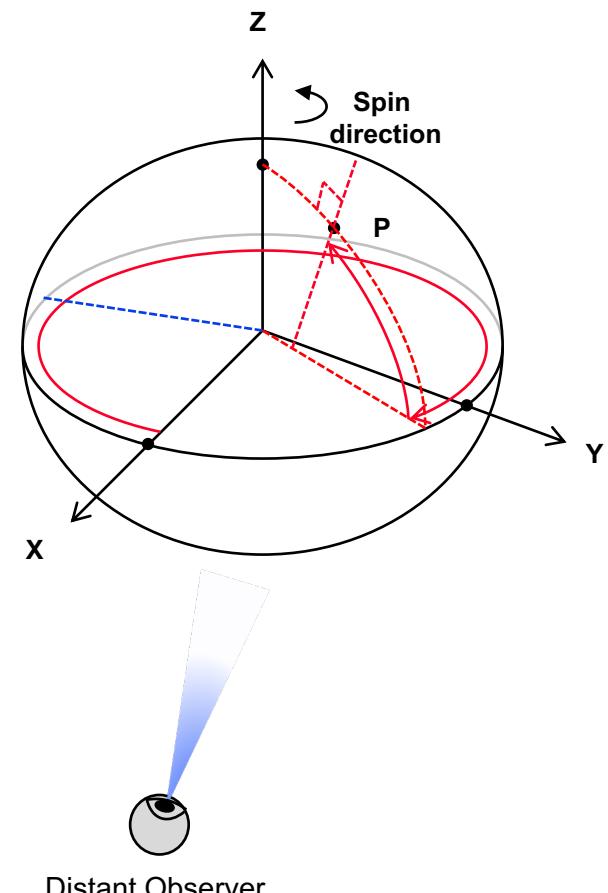
Navigation and Ancillary Information Facility

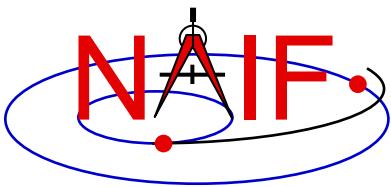
- **For planet and satellite planetographic coordinate systems:**

- Planetographic longitude is usually defined such that the sub-observer longitude increases with time as seen by a distant, fixed observer (0 to 360)
- The earth, moon and sun are exceptions; planetographic longitude is positive east by default (0 to 360)
- Planetographic latitude is planetodetic latitude (-90 to +90)
- Toolkit planetographic APIs are:
  - » PGRREC, RECPGR, DRDPGR, DPGRDR, XFMSTA

- **For dwarf planets, asteroids and comets:**

- There are multiple, inconsistent standards! (USNO, IAU, PDS)
- **NAIF strongly suggests you use only planetocentric or planetodetic coordinates for these objects**





# Spherical Coordinates

Navigation and Ancillary Information Facility

- **Longitude:**

- angle from +X axis to projection of position vector on X-Y plane
- increases in counter-clockwise direction
- see the API header for restrictions on ranges

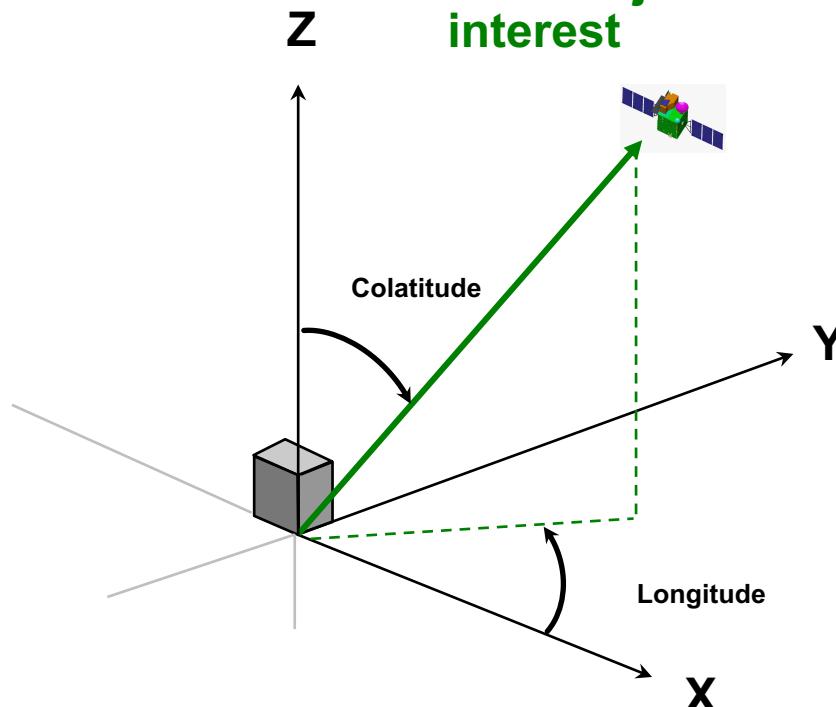
- **Colatitude:**

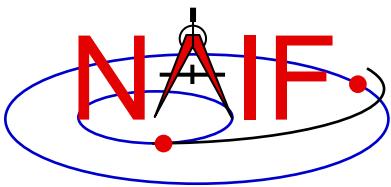
- Angle between +Z axis and position vector (0 to 180)
- Other names used elsewhere are zenith angle, inclination angle and polar angle.

- **Toolkit spherical APIs :**

- SPHREC, RECSPH,  
DRDSPH, DSPHDR, XFMSTA

Position Vector  
of an object of  
interest





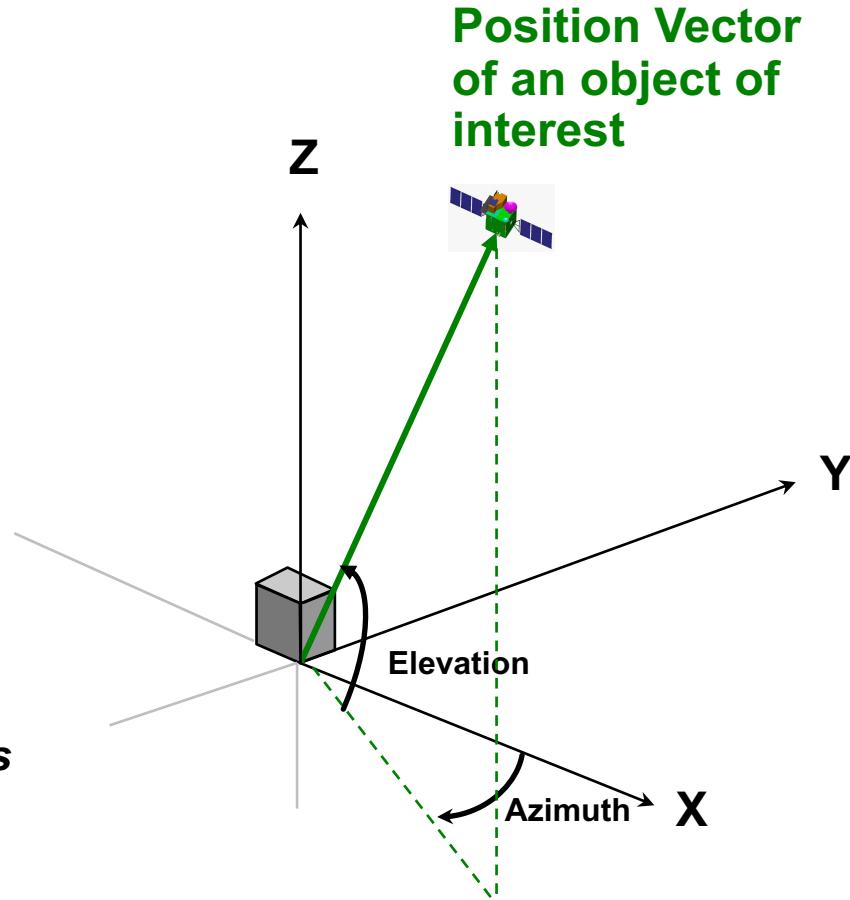
# An Example of Azimuth-Elevation Coordinates

Navigation and Ancillary Information Facility

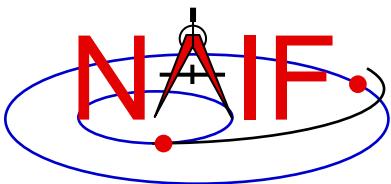
- **Azimuth:**
  - Angle from +X axis to projection of position vector on x-y plane
  - Increases in clockwise direction (0 to 360)

- **Elevation:**
  - Angle between position vector and x-y plane (-90 to +90)
  - In this example, +Z is in the “up” direction, which might not be true for you.

*SPICE does not currently contain APIs specific to converting between AZ-EL and other coordinate systems due to lack of standard definitions for AZ-EL. See the next page for methods for doing this conversion.*



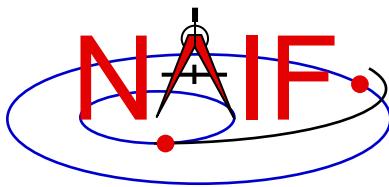
Position Vector  
of an object of  
interest



# Converting Rectangular to AZ-EL Coordinates

Navigation and Ancillary Information Facility

- **Rectangular to AZ-EL**
  - Using RA-DEC as intermediary
    - » Convert rectangular to RA-DEC using RECRAD (where the range for RA is  $[0, 2\pi]$ )
    - » Then map RA-DEC to whatever is the AZ-EL convention you are using (how does DEC compare with your definition of EL?)
  - Using LAT-LON as intermediary
    - » Convert rectangular to LAT-LON using RECLAT, where the range for LON is  $[-\pi, \pi]$
    - » Then map LAT-LON to whatever is the AZ-EL convention you are using (e.g. you could negate LAT to achieve positive EL being “up” in a frame having Z pointed “down.”)

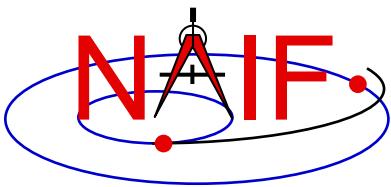


# Summary of SPICE Coordinate Transformation APIs

Navigation and Ancillary Information Facility

Coordinate Systems	APIs for Position Transformation	APIs for Velocity Transformation	Notes
Latitudinal to/from Rectangular	LATREC RECLAT	DRDLAT DLATDR	More commonly called Planetocentric. Use these APIs for Azimuth/Elevation as well.
R.A. & Dec. to/from Rectangular	RADREC RECRAD	DRDLAT DLATDR	Same as for latitudinal except for range of LON and RA when converting rectangular to angular. LON: -Pi to +Pi RA: 0 to 2 Pi
Planetographic to/from Rectangular	PGRREC RECPGR	DRDPGR DPGRDR	Best restricted to planets, satellites and the sun. Requires a text PCK to be loaded to determine body spin direction.
Geodetic to/from Rectangular	GEOREC REC GEO	DRDGEO DGEODR	
Cylindrical to/from Rectangular	CYLREC RECCYL	DRDCYL DCYLD R	
Spherical to/from Rectangular	SPHREC RECSPH	DRDSPH DSPHDR	Shape must be a true sphere.
AZ-EL to/from Rectangular	none	none	See earlier chart titled "Converting Rectangular to AZ-EL Coordinates"

See also the next page re XFMSTA



# Examples of Velocity Coordinate Transformations

Navigation and Ancillary Information Facility

This example is for rectangular to spherical

Fortran examples

- **Using full state vector transformation API**

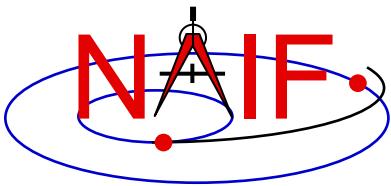
```
CALL SPKEZR ( TARG, ET, REF, CORR, OBS, STATE, LT )
CALL XFMSTA ( STATE, 'RECTANGULAR', 'SPHERICAL', ' ', OUTSTATE )
```

- **Using velocity-only (Jacobian) APIs**

- Transform velocities from rectangular to spherical coordinates using the SPICE Jacobian matrix routines. 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
    - (  $d(r)/dt$ ,  $d(\text{colatitude})/dt$ ,  $d(\text{longitude})/dt$  )
  - 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.
- **Note: Using XFMSTA for velocity transformations is slower than using the Jacobian API**



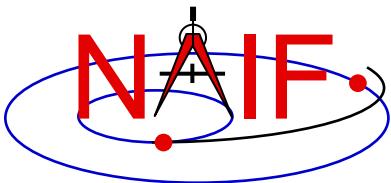
---

Navigation and Ancillary Information Facility

# Ephemeris Subsystem SPK

Focused on reading SPK files

January 2020



# First... clear your mind!

---

Navigation and Ancillary Information Facility

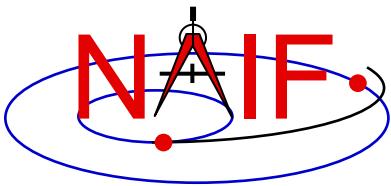
- **SPK is probably unlike any previous ephemeris or trajectory representation you've used or heard about.**
- We think you'll find it to be more capable than other ephemeris system architectures.
  - As such, it's also a bit more complicated to grasp.
- *Don't panic! Shortly you'll be reading SPK files like a pro.*



---

Navigation and Ancillary Information Facility

# Overview of SPICE Ephemeris Data

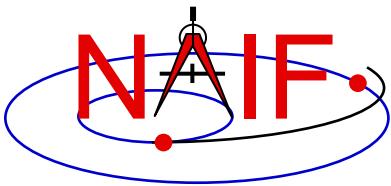


# A Picture is worth ...

---

Navigation and Ancillary Information Facility

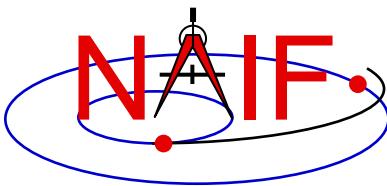
- We'll start with a mostly pictorial view of ephemeris data and SPK files, just to ease you into this topic.



# SPICE Ephemeris Data

Navigation and Ancillary Information Facility

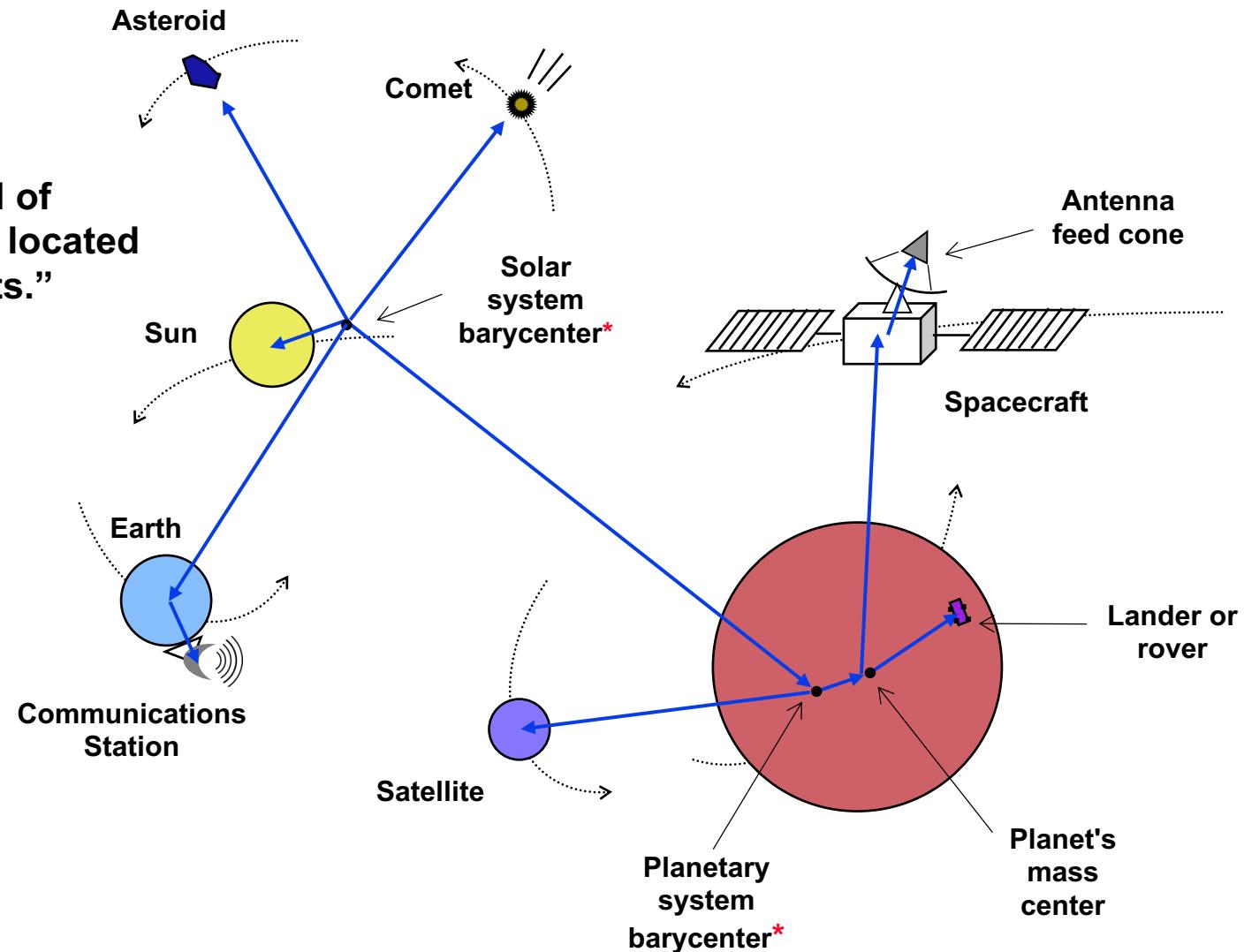
- An SPK file contains ephemeris (trajectory) data for "ephemeris objects."
  - "Ephemeris" means position and velocity as a function of time
    - » Position + velocity is often referred to as "state"
- "Ephemeris objects" are spacecraft, planets, satellites, comets and asteroids.
  - But the following are also ephemeris objects:
    - » the center of mass of our solar system (solar system barycenter)
    - » the center of mass of a planet/satellite system (planet barycenter)
    - » 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

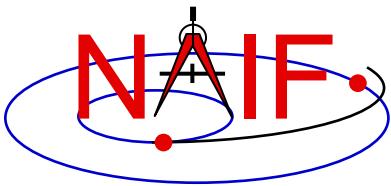


# Examples of SPICE Ephemeris Objects

Navigation and Ancillary Information Facility

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





# Imagine Some Ephemeris Data

Navigation and Ancillary Information Facility

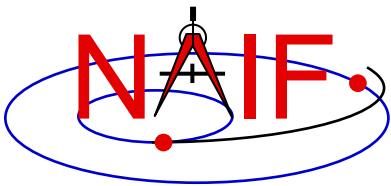
```
epoch_1, x1, y1, z1, vx1, vy1, vz1  
epoch_2, x2, y2, z2, vx2, vy2, vz2  
epoch_3, x3, y3, z3, vx3, vy3, vz3  
epoch_4, x4, y4, z4, vx4, vy4, vz4  
..... etc. ....  
..... etc. ....  
epoch_n, xn, yn, zn, vxn, vyn, vzn
```

Perhaps this is an ASCII table or an Excel spreadsheet containing rows of time-tagged Cartesian state vectors.

“epoch” = time

**It may not be written inside the table or spreadsheet, but perhaps an interface agreement somehow tells you:**

- what object this ephemeris is for
- what is the name of the reference frame in which the data are given
- what is the center of motion of the object
- what time system is being used for the epochs
- what are the start and stop times of the file
  - » meaning, what are “epoch\_1” and “epoch\_n”



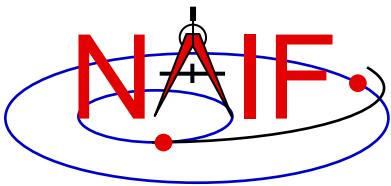
# Imagine a Simple Ephemeris File

Navigation and Ancillary Information Facility

```
epoch_1, x1, y1, z1, vx1, vy1, vz1  
epoch_2, x2, y2, z2, vx2, vy2, vz2  
epoch_3, x3, y3, z3, vx3, vy3, vz3  
epoch_4, x4, y4, z4, vx4, vy4, vz4  
..... etc. ....  
..... etc. ....  
epoch_n, xn, yn, zn, vxn, vyn, vzn
```



We'll represent that simple ephemeris data shown on the previous page as a "block" like this.



# Imagine a Simple Ephemeris File

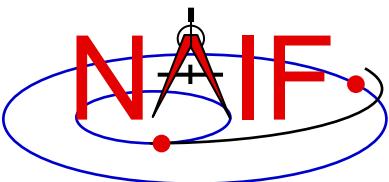
Navigation and Ancillary Information Facility

```
epoch_1, x1, y1, z1, vx1, vy1, vz1  
epoch_2, x2, y2, z2, vx2, vy2, vz2  
epoch_3, x3, y3, z3, vx3, vy3, vz3  
epoch_4, x4, y4, z4, vx4, vy4, vz4  
..... etc. ....  
..... etc. ....  
epoch_n, xn, yn, zn, vxn, vyn, vzn
```



We'll represent that simple ephemeris file as a "block" like this.

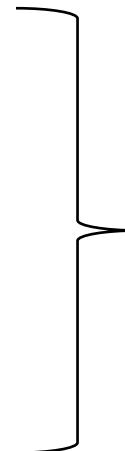
This becomes the basis of a "segment" in an SPK file.



# An SPK “Segment”

Navigation and Ancillary Information Facility

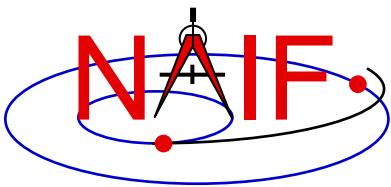
Target, Ref Frame ID, Center of Motion, $T_{start}$ , $T_{stop}$					
epoch_1, x1, y1, z1, vx1, vy1, vz1					
epoch_2, x2, y2, z2, vx2, vy2, vz2					
epoch_3, x3, y3, z3, vx3, vy3, vz3					
epoch_4, x4, y4, z4, vx4, vy4, vz4					
..... etc.					
..... etc.					
epoch_n, xn, yn, zn, vxn, vyn, vzn					



One segment

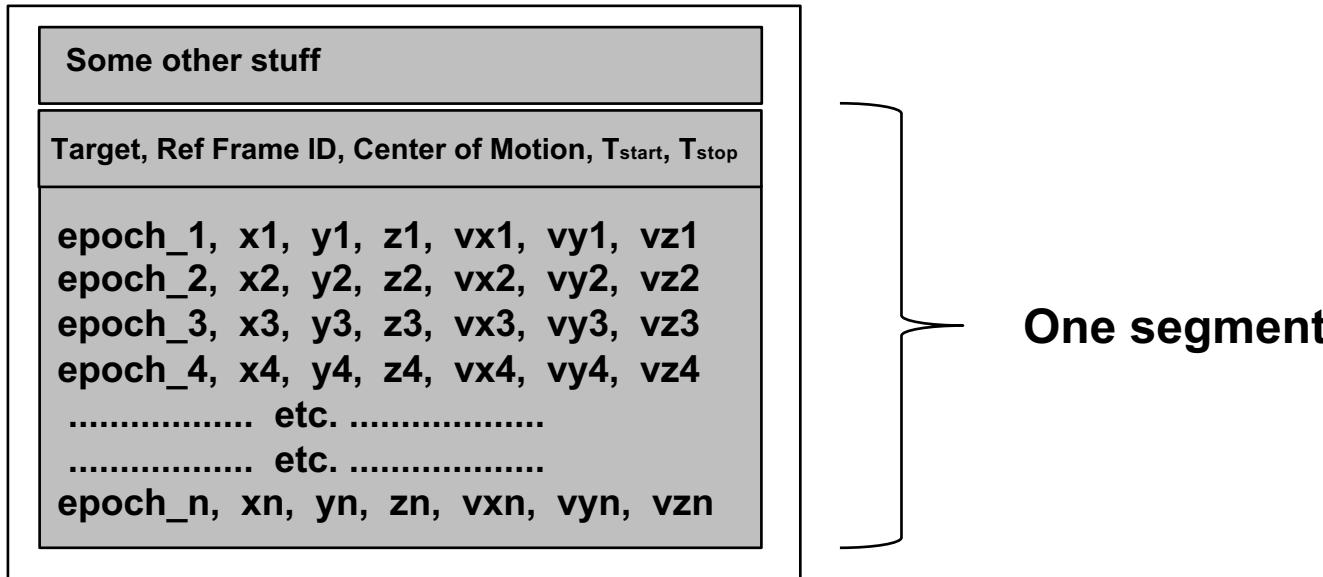
We insert some meta-data into the segment:

- what is the object this ephemeris is for – SPICE calls this the “target”
- what is the ID of the reference frame in which the data are given
- what is the center of motion for the target
- the start and stop times of the file,  $T_{start}$  and  $T_{stop}$ 
  - » meaning, what are “epoch\_1” and “epoch\_n”

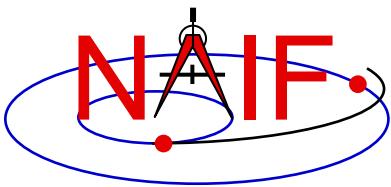


# A Simple SPK File

Navigation and Ancillary Information Facility

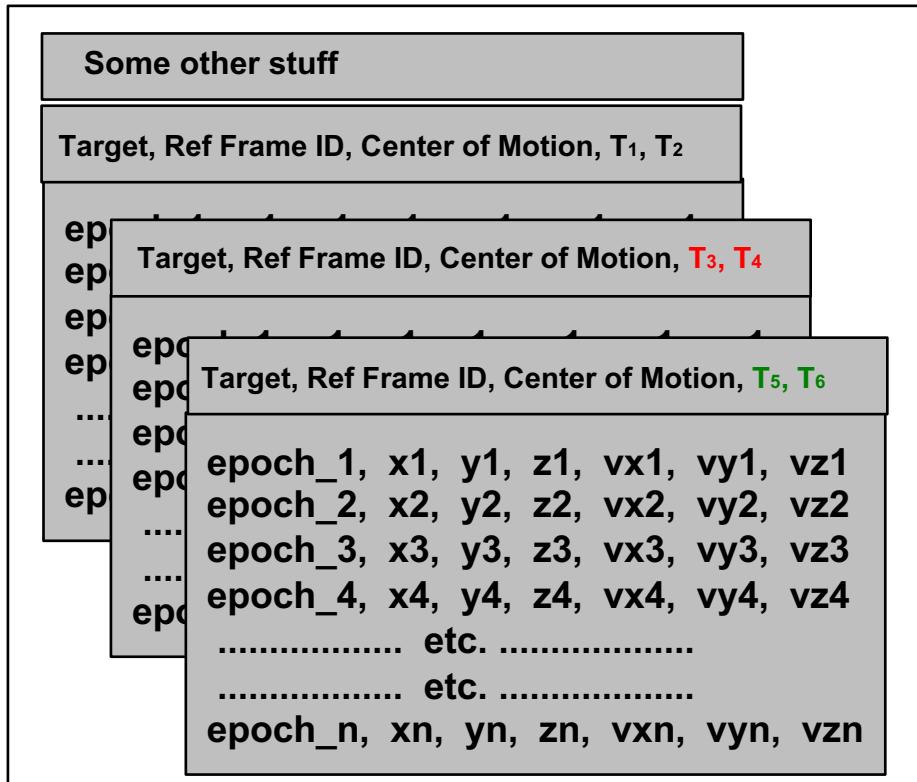


- **This very simple SPK file is made up of a single segment containing ephemeris data:**
  - for a single object (perhaps a spacecraft, an asteroid, or ...whatever),
  - given in a single reference frame,
  - having a single center of motion,
  - with data spanning from T<sub>start</sub> to T<sub>stop</sub>

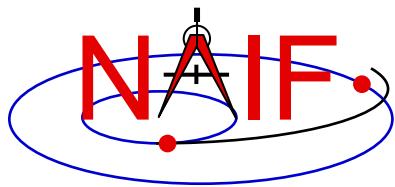


# A More Substantial SPK File

Navigation and Ancillary Information Facility

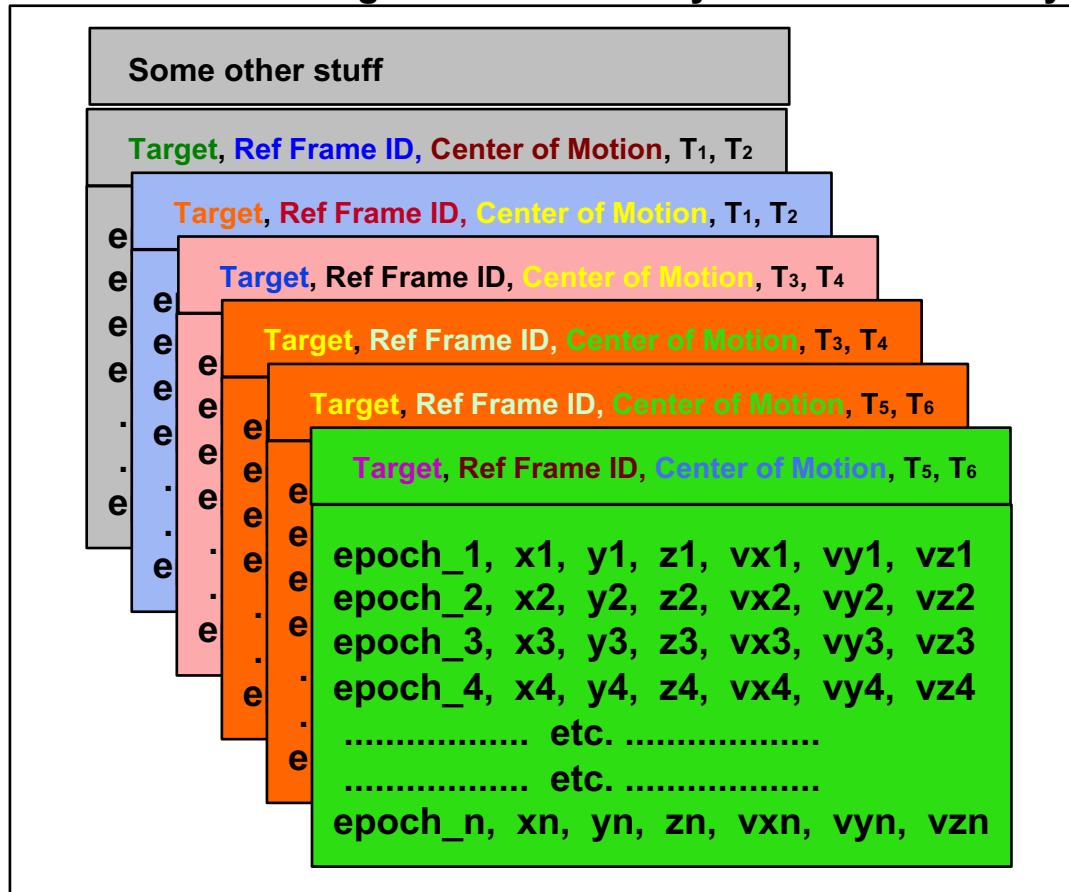


- This more substantial SPK is made up of multiple segments containing ephemeris data:
  - for a single object (perhaps a spacecraft, an asteroid, or ...???)
  - given in a single reference frame (“coordinate frame”),
  - having a single center of motion,
  - with data spanning from T<sub>1</sub> to T<sub>6</sub>

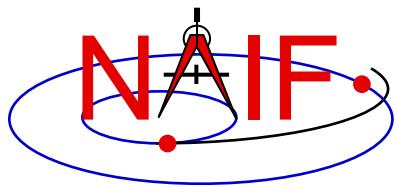


# An Even More Substantial SPK File

## Navigation and Ancillary Information Facility



- **This even more substantial SPK contains multiple segments having:**
  - **several objects (targets)**
  - **several reference frames**
  - **several centers of motion**
  - **several pairs of start and stop times**

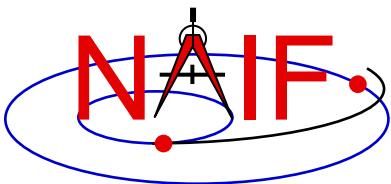


# SPK “Type” Info in each Segment

Navigation and Ancillary Information Facility

Some other stuff
Target, Ref Frame ID, Center of Motion, T <sub>1</sub> , T <sub>2</sub> , <b>Type 13</b>
epoch_1, x <sub>1</sub> , y <sub>1</sub> , z <sub>1</sub> , vx <sub>1</sub> , vy <sub>1</sub> , vz <sub>1</sub> epoch_2, x <sub>2</sub> , y <sub>2</sub> , z <sub>2</sub> , vx <sub>2</sub> , vy <sub>2</sub> , vz <sub>2</sub> epoch_3, x <sub>3</sub> , y <sub>3</sub> , z <sub>3</sub> , vx <sub>3</sub> , vy <sub>3</sub> , vz <sub>3</sub> epoch_4, x <sub>4</sub> , y <sub>4</sub> , z <sub>4</sub> , vx <sub>4</sub> , vy <sub>4</sub> , vz <sub>4</sub>
Target, Ref Frame ID, Center of Motion, T <sub>3</sub> , T <sub>4</sub> , <b>Type 2</b>
MID, RADIUS, X coefs, Y coefs, Z coefs MID, RADIUS, X coefs, Y coefs, Z coefs MID, RADIUS, X coefs, Y coefs, Z coefs (some time tag info)
Target, Ref Frame ID, Center of Motion, T <sub>5</sub> , T <sub>6</sub> , <b>Type 1</b>
First set of difference line coeffs Second set of difference line coeffs epoch_1 epoch_2

- Each segment can contain a different type of ephemeris data (as long as it's been built into the SPK subsystem). Examples:
  - Discrete state vectors
  - Chebyshev polynomials
  - Difference lines (unique to JPL)
  - Etc., etc.
- Each segment has the **SPK Type** stored in its meta-data record.
- Toolkit software knows how to evaluate each Type – no worries for you!



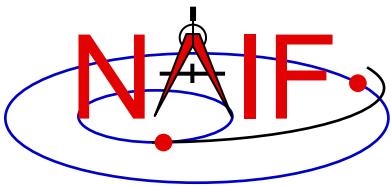
# SPK Data are Continuous Within a Segment

Navigation and Ancillary Information Facility

Cassini, Ref Frame ID, Saturn bc, T<sub>1</sub>, T<sub>2</sub>, Type 13

```
epoch_1, x1, y1, z1, vx1, vy1, vz1  
epoch_2, x2, y2, z2, vx2, vy2, vz2  
epoch_3, x3, y3, z3, vx3, vy3, vz3  
epoch_4, x4, y4, z4, vx4, vy4, vz4
```

- Within the time bounds (T<sub>1</sub>, T<sub>2</sub>) of a single segment, SPICE software will return a result—a state vector consisting of position and velocity—at any epoch... not just at the discrete epochs of the ephemeris records (epoch\_1, epoch\_2, epoch\_3, epoch\_4)
- In the example above, SPICE will return the position and velocity (the state) of the Cassini spacecraft relative to the Saturn barycenter at any time  $t$  where:  $T_1 \leq t \leq T_2$



# Chaining and Frame Transformation

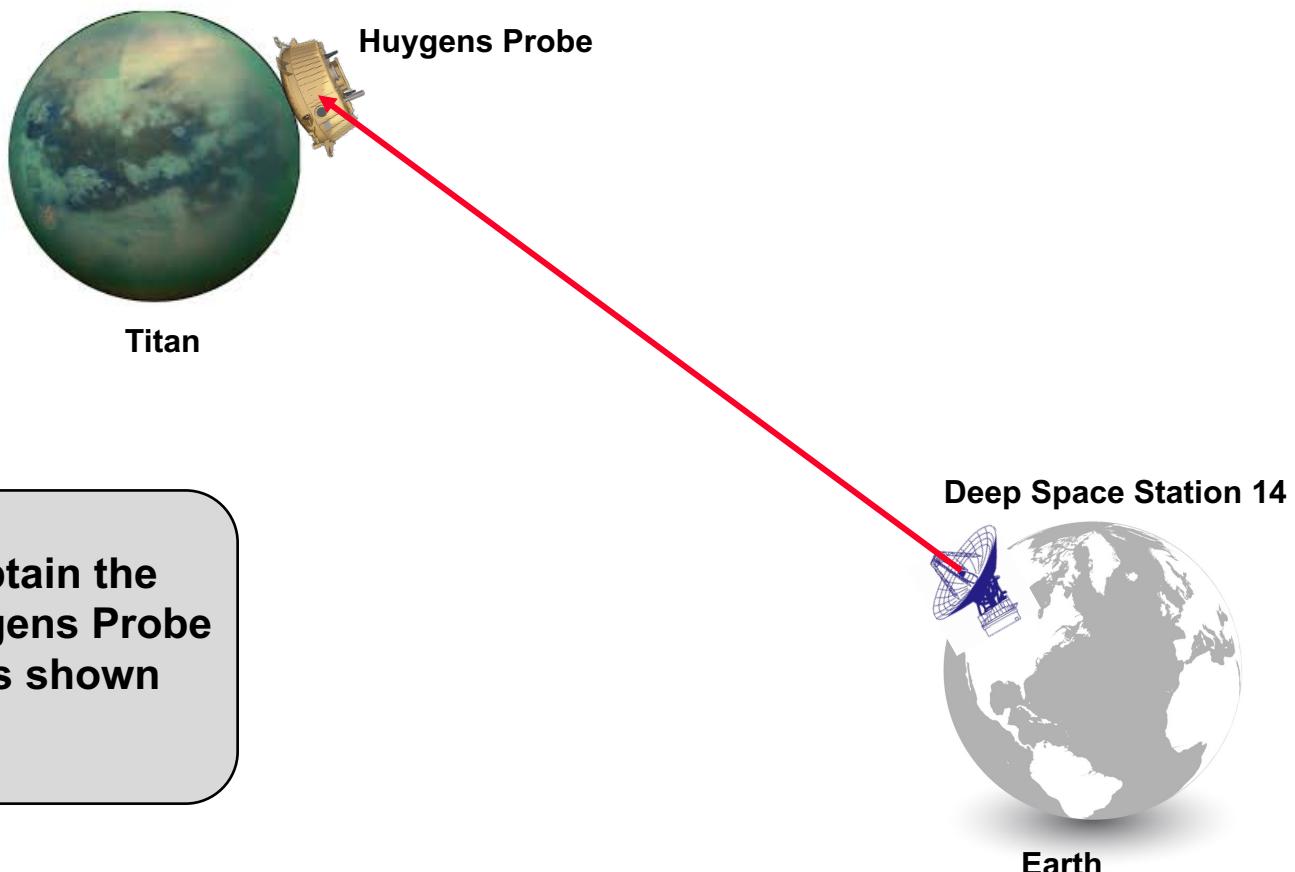
Navigation and Ancillary Information Facility

- Next we'll discuss “chaining” and “frame transformations”... features of the SPK subsystem that make it rather unique.



# Your Question

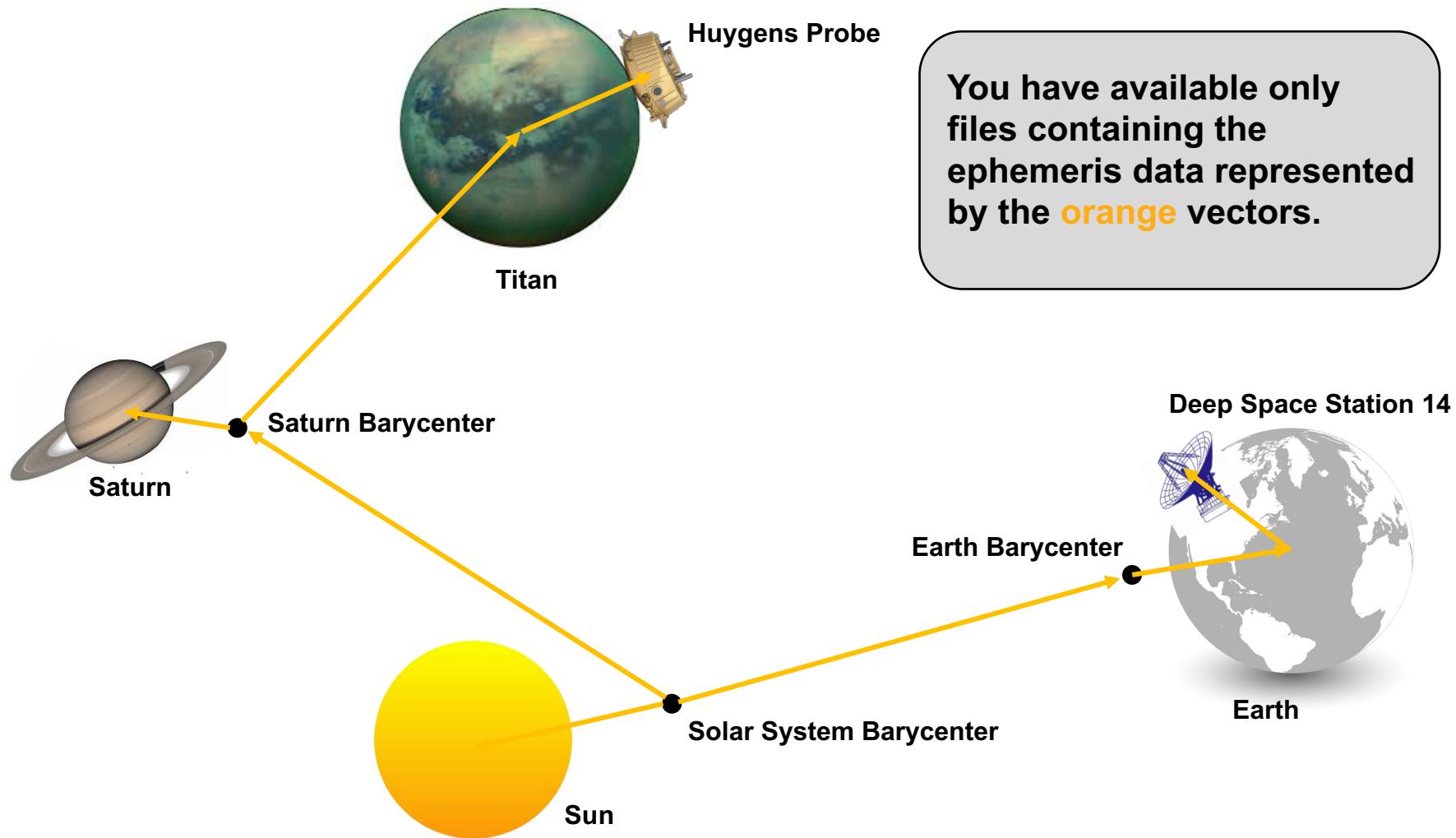
Navigation and Ancillary Information Facility





# Your Dilemma

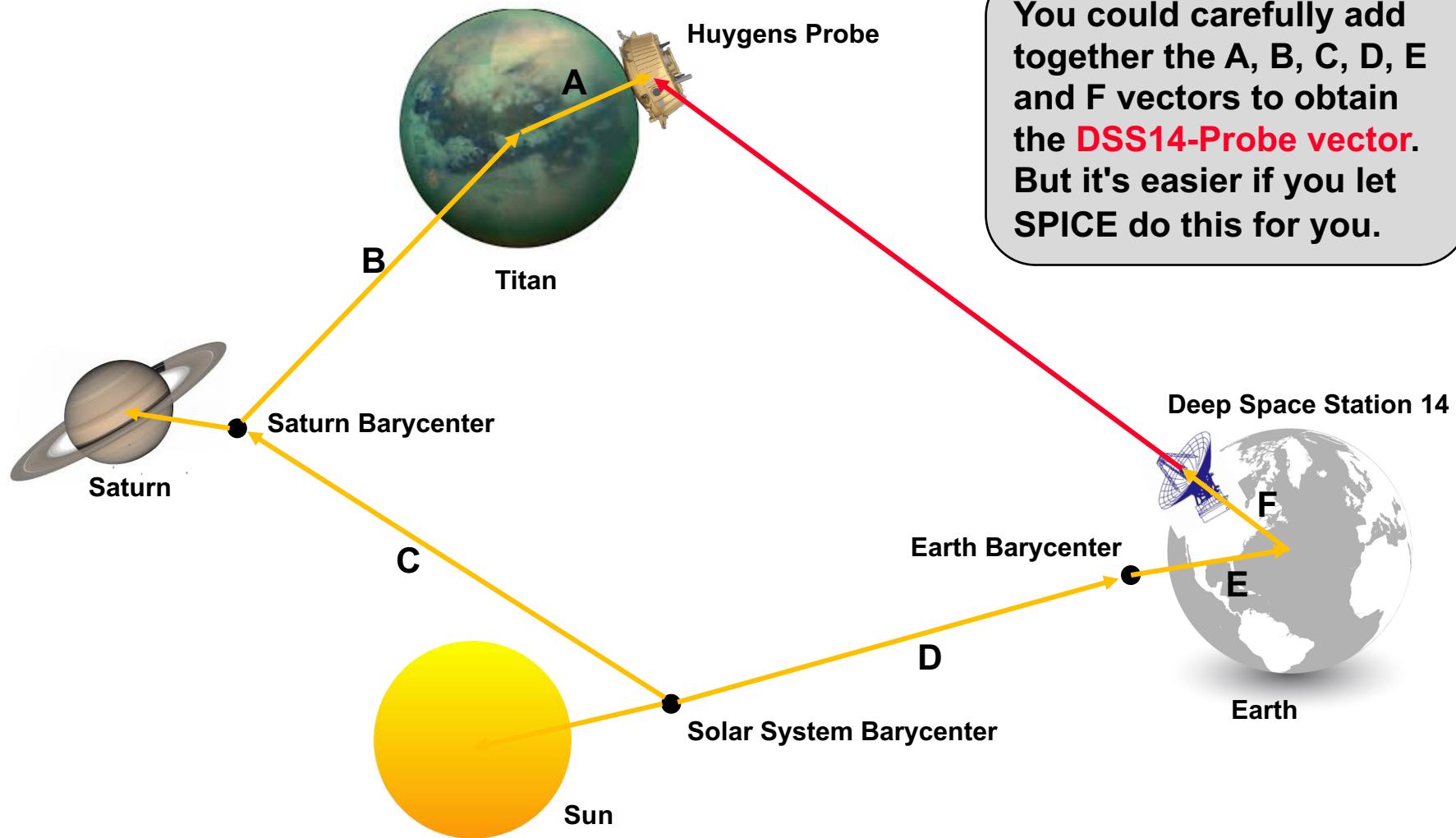
Navigation and Ancillary Information Facility

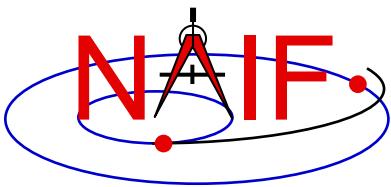




# Not a Problem

Navigation and Ancillary Information Facility

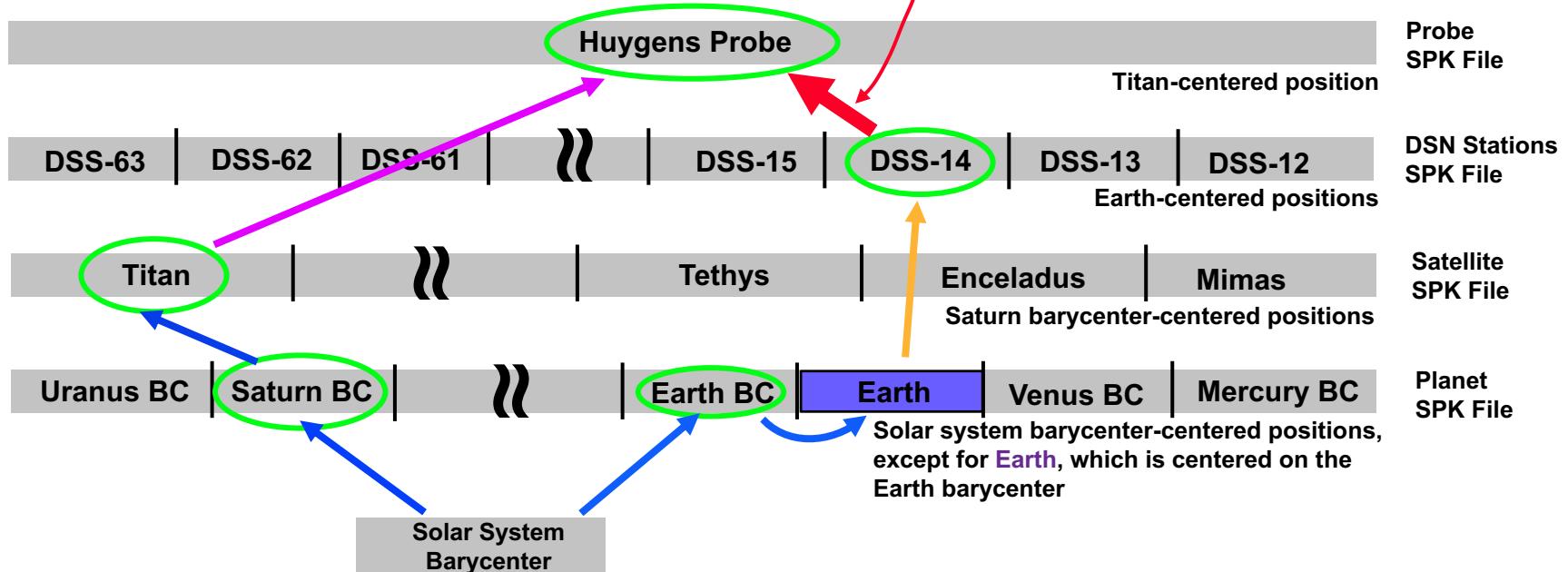




# SPICE Chains SPK Data

Navigation and Ancillary Information Facility

- SPICE automatically searches across all loaded SPK files to find the segments needed to compute the vectors needed to obtain the result the customer has asked for. SPICE chains these together using addition and subtraction.
  - In this example the user wants the **position** of the Huygens probe sitting on the surface of Titan as seen from Deep Space Station 14.
  - SPICE computes this by chaining the **gold**, **blue** and **violet** chunks.

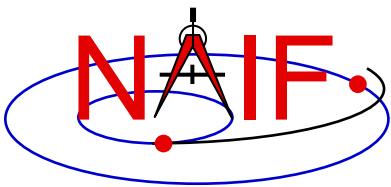




# Maybe It's Not "Simple Addition"

Navigation and Ancillary Information Facility

- **What if the A, B, C, D, E and F vectors shown two pages ago are given with respect to several different reference frames? (This is the normal situation!)**
  - Before you can add the vectors together you need to rotate them into a common reference frame.
  - This may not be very easy to accomplish on your own.
  - Once the addition is complete you may need to rotate the resultant vector into the reference frame appropriate for the job you are doing.



# SPICE Automates Frame Transformation

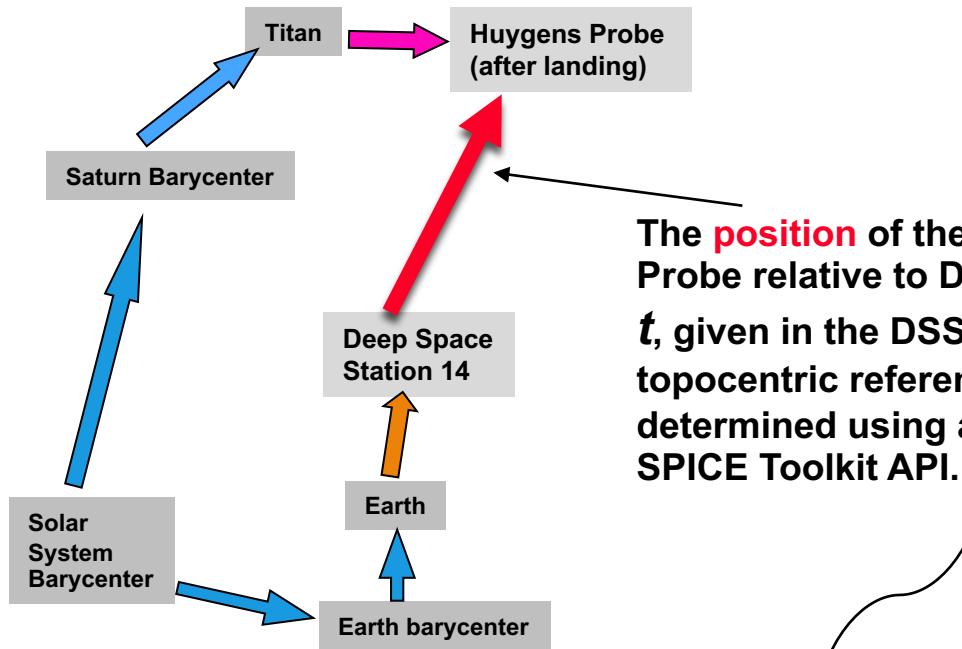
Navigation and Ancillary Information Facility

- As part of the “chaining” process just mentioned...
  - position vectors are automatically rotated into a consistent reference frame
  - the final vector is rotated into the output reference frame requested by the user

## Reference Frames Used

- International Celestial Reference Frame (J2000)
- Titan body-fixed frame (IAU\_TITAN)
- International Terrestrial Reference Frame (ITRF93)
- DSS-14 topocentric reference frame (DSS-14\_TOPO)

## Ephemeris Segments Used

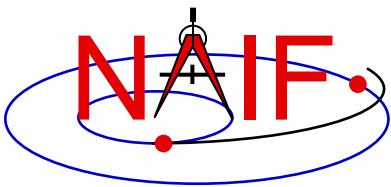


The **position** of the Huygens Probe relative to DSS-14, at time  $t$ , given in the DSS-14 topocentric reference frame, is determined using a single SPICE Toolkit API.

*A single API does it all!*

CALL SPKPOS ('HUYGENS\_PROBE',  $t$ , 'DSS-14\_TOPO', 'CN+S', 'DSS-14', POSITION, LT)

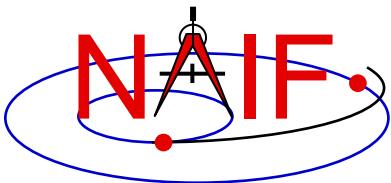
Fortran syntax  
used here



# SPK Segment Order and Priority

Navigation and Ancillary Information Facility

- **Within a single SPK file...**
  - The segments in an SPK file **need not be ordered according to time or body**.
  - But segment order **does imply priority**. If two segments from the same SPK file both contain data for a given target and time that satisfy a request, the SPK system selects the segment for which the **physical location** is positioned **later** in the file.
    - » The centers of motion, frames and SPK types are irrelevant to this selection.
- **If using two or more SPK files...**
  - Segments from SPK files loaded **later** have higher priority: if two segments from two different SPK files both contain data for a given target and time, the SPK system selects the segment from the SPK file that was loaded later.
    - » The centers of motion, frames and SPK types are irrelevant to this selection.



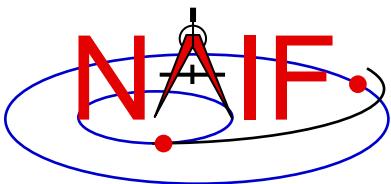
# Details

---

Navigation and Ancillary Information Facility

- Now for some details.
- There's quite a lot... don't feel you need to grasp all of this immediately.

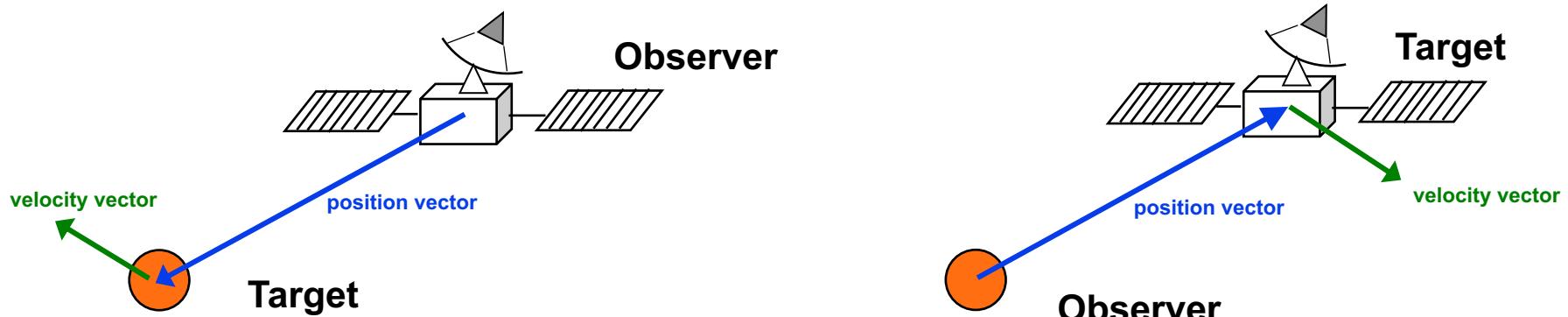




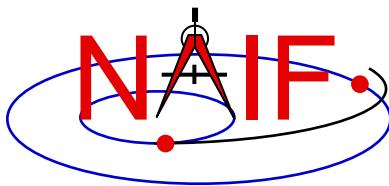
# Reading an SPK: Observers and Targets

Navigation and Ancillary Information Facility

- 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 computed **position** data point from the “observer” to the “target.”
  - The computed **velocity** is that of the “target” relative to the “observer.”



- Any ephemeris object can be a target **or** an observer!



# SPK File Coverage - 1

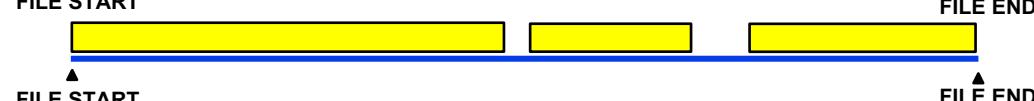
Navigation and Ancillary Information Facility

- 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.

SPK file containing data for one object with no data gaps



SPK file containing data for one object, with two data gaps



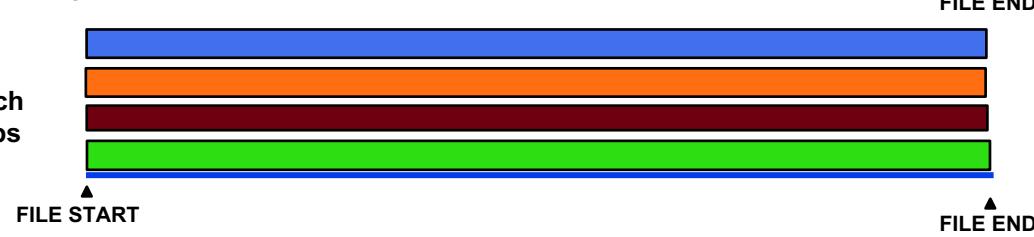
SPK file containing data for three objects, each having different data gaps

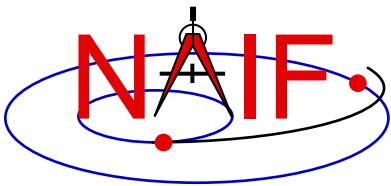


SPK file containing data for three objects, each having different coverage but with no data gaps



SPK file containing data for several objects, each having the same coverage and with no data gaps



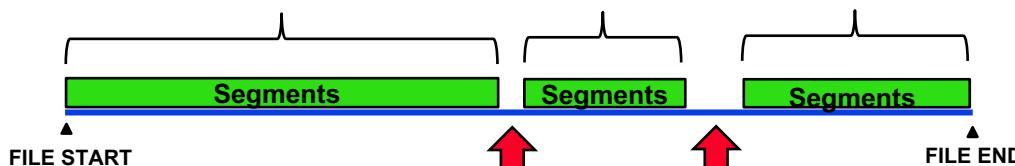


# SPK File Coverage - 2

Navigation and Ancillary Information Facility

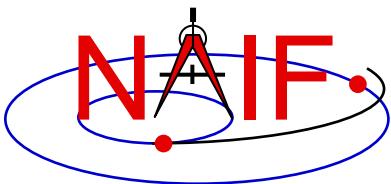
- For any request time within any time interval comprising the coverage for an object (i.e. the three green stripes shown below), the SPK subsystem can return a vector representing the state of that object relative to its center of motion.
  - The SPK system will automatically interpolate ephemeris data to produce a Cartesian 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.
- The SPK subsystem will *not* return a result for a request time falling within a data gap.
  - Data gaps can only occur between segments.

“Results” will be returned by the SPK reader API for any request time falling within these three intervals.



Note: each of the green stripes above consists of one or more segments.

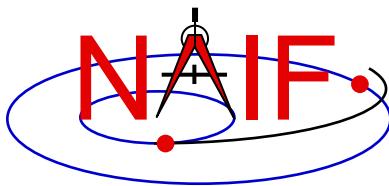
A SPICE error message will be returned by the SPK subsystem for any request time falling within these two data gaps



# Reference Frames Used in Writing and Reading SPKs

Navigation and Ancillary Information Facility

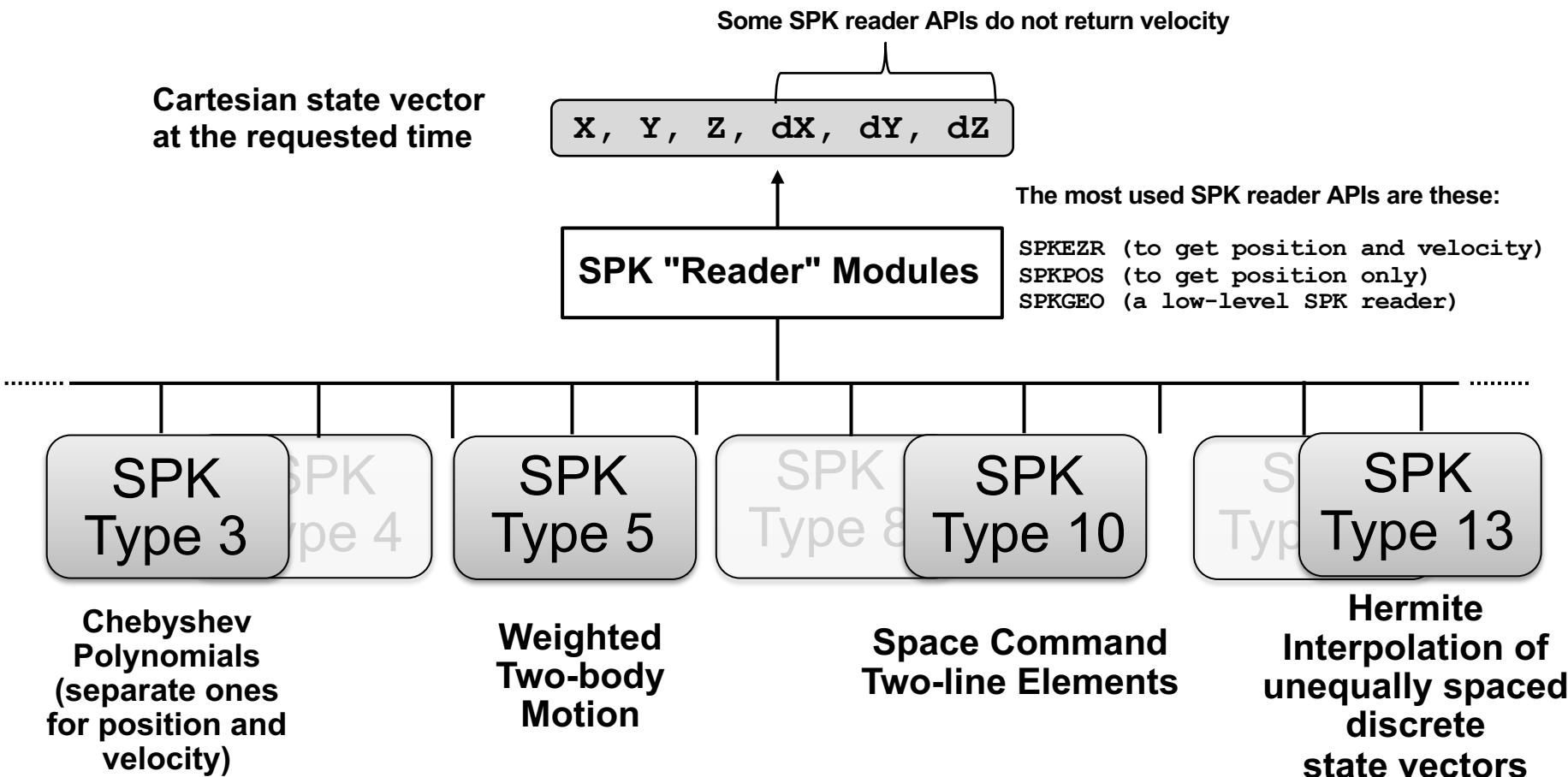
- All ephemeris data have an associated reference frame
  - The frame specification is input by the SPK producer and is stored in the segment's meta-data
    - » This input frame must be one “known” to the SPICE system
  - The frame may be different across segments
- A program reading an SPK file specifies relative to what reference frame the output state or position vectors are to be given; you're not stuck with using the frame the SPK producer used
  - This output frame you select must be known to your program
    - » “Known” means either a built-in frame (hard coded in SPICE) or one specified in a Frames Kernel
    - » The user's program may need to have access to additional SPICE data in order to construct the specified frame

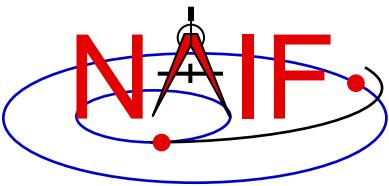


# SPK Data Type Concept

Navigation and Ancillary Information Facility

SPK files may contain various mathematical representations of ephemeris data ("data types"), but the high-level user interfaces (SPK "readers") are type-independent:

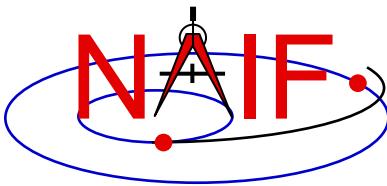




# Why Have Multiple Data Types?

Navigation and Ancillary Information Facility

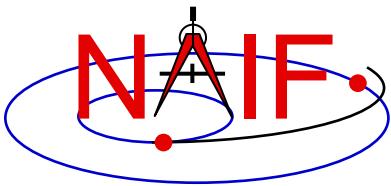
- To allow an SPK producer to choose an ephemeris representation well-suited for her/his application. For example:
  - Weighted two-body extrapolation (Type 5) yields compact files; may be used with sparse data. Only accurate for motion that is well approximated by the two-body model.
  - SPK files based on sliding-window Lagrange and Hermite interpolation (Types 9 and 13) are easy to create. Position can be made arbitrarily accurate with sufficiently small time separation of states.
  - Chebyshev polynomials (Types 2, 3, 14) yield the best combination of evaluation speed and accuracy. But the file creator must do more work to use these data types.
- To replicate data originally provided in another format.
  - Types 1, 2, 3, 8, 10, 14, 15, 17 and 18 were developed to enable accurate duplication of data obtained from original ephemeris developers.



# Widely Used SPK Data Types

Navigation and Ancillary Information Facility

- **Type 1 (Modified divided difference arrays)**
  - Used by JPL orbit determination software for spacecraft ephemerides
- **Type 2 (Chebyshev polynomials for position, velocity given by differentiation)**
  - Used for JPL planetary ephemerides
- **Type 3 (Separate Chebyshev polynomials for position and velocity)**
  - Used for JPL satellite ephemerides
- **Type 5 (Weighted two-body extrapolation)**
  - Used for comets and asteroids, as well as for sparse data sets where a piecewise two-body approximation is acceptable
- **Type 10 (Space command two-line elements)**
  - Used for some earth orbiters
- **Types 9 and 13 (Sliding-window Lagrange and Hermite interpolation of unequally-spaced states)**
  - Used by many non-JPL ephemeris producers and by MKSPK users
- **Type 18, 19 (Sliding window Hermite or Lagrange interpolation)**
  - Used in SPKs made by ESA planetary missions (through Rosetta)



# Barycenters

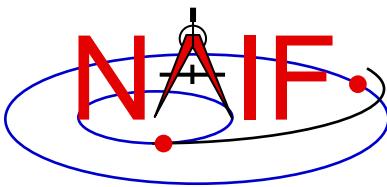
Navigation and Ancillary Information Facility

- **For planets**

- A planet and its satellites orbit the planet system's barycenter
  - » For example, the planet Jupiter (599) and each of Jupiter's satellites (501 - 5xx) orbit the Jupiter system barycenter (5)
- Because Mercury and Venus have no satellites, their barycenters (1 and 2) are at exactly the same locations as their mass centers (199 and 299)
  - » Therefore SPICE ephemeris objects 199 and 299 as well as 1 and 2 are found in a planet ephemeris file
- Because the masses of Phobos and Deimos are so small compared to the mass of Mars, the mass center for Mars (499) was treated as being located at the Mars barycenter (4)
  - » Starting in 2013 with the JPL planetary ephemeris named DE430 this is no longer the case; there is a very small offset of about 20 cm

- **For the solar system**

- Planet system barycenters (i.e. 1 through 9) and the sun (10) orbit the solar system barycenter (0)



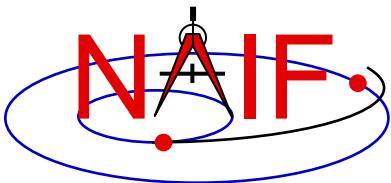
# Barycenter Offset Magnitude

Navigation and Ancillary Information Facility

<u>Body Mass Center</u>	<u>System Barycenter</u>	<u>Barycenter offset from body mass center (km)*</u>	<u>Offset as % of body radius*</u>
Sun (10)	SSB (0)	1,378,196	198%
Mercury (199)	M. BC (1)	0	0
Venus (299)	V. BC (2)	0	0
Earth (399)	E. BC (3)	4942	77%
Mars (499)	M. BC (4)	0.0002	~ 0
Jupiter (599)	J. BC (5)	220	0.3%
Saturn (699)	S. BC (6)	312	0.5%
Uranus (799)	U. BC (7)	43	0.17%
Neptune (899)	N. BC (8)	74	0.3%
Pluto (999)**	P. BC (9)	2056	172%

\* Estimated maximum values over the time range 2000-2050.

\*\* For ephemeris purposes, Pluto is still treated as a planet.



# SPK File Contents

Navigation and Ancillary Information Facility

- A single SPK file can hold data for one ephemeris object, or for many ephemeris objects
- The ephemeris objects in a given SPK file need not all be of the same type
  - One might find data for a spacecraft, some planets, and some satellites all in one file, split across multiple segments
- This is illustrated in the next three charts



# Examples of Generic SPK File Contents

Navigation and Ancillary Information Facility

Planet Ephemeris

Asteroid Ephemeris

Merged Planet<sup>4</sup> and Satellite Ephemeris

<b>0</b>	<b>Solar System BC</b>
1	Merc. BC
199 <sup>1</sup>	Mercury
2	Venus BC
299 <sup>1</sup>	Venus
3	Earth BC
301 <sup>2</sup>	Moon
399 <sup>2</sup>	Earth
4	Mars BC
5	Jupiter BC
6	Saturn BC
7	Uranus BC
8	Neptune BC
9	Pluto BC*
10 <sup>3</sup>	Sun

\* For ephemeris purposes, Pluto is still treated as a planet.

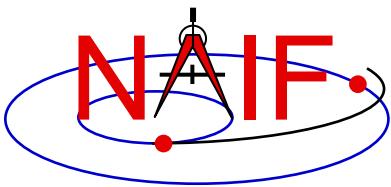
**10 Sun**  
2000001 Ceres

Notes:

- (1) Mercury and Venus planet locations are included in planet ephemerides since there are no satellite ephemerides for these planets.
- (2) The Moon and Earth locations are included in each planetary ephemeris because of historical ephemeris production techniques.
- (3) The Sun's location is included in each planetary ephemeris because of historical ephemeris production techniques.
- (4) For user convenience, NAIF often merges into a planet's satellite ephemeris files the locations of the earth, the earth barycenter and the sun.

**5 Jupiter BC**  
3<sup>4</sup> Earth BC  
10<sup>4</sup> Sun  
399<sup>4</sup> Earth  
501 Io  
502 Europa  
503 Ganymede  
504 Callisto  
505 Amalthea  
514 Thebe  
515 Adrastea  
516 Metis  
599 Jupiter

*The objects in blue italic font are the center of motion for the remaining objects in each file. These is no ephemeris data present for these centers of motion.*



# Examples of a Flight Project's SPK File Contents

Navigation and Ancillary Information Facility

This made-up example shows four collections of SPK files for the Cassini mission

## Cassini Orbiter

6 = Saturn bc

-82 = Cassini S/C

One object

## Huygens Probe

6 = Saturn bc

-150 = Huygens Probe

One object

## Planets

0 = solar system bc

3 = Earth barycenter

6 = Saturn barycenter

399 = Earth mass center

301 = Moon

Multiple objects

## Satellites - 1

6 = Saturn bc

601 = Mimas

602 = Enceladus

603 = Tethys

604 = Dione

605 = Rhea

606 = Titan

607 = Hyperion

608 = Iapetus

609 = Phoebe

699 = Saturn mass center

Multiple objects

610 = Janus

611 = Epimetheus

:

617 = Pandora

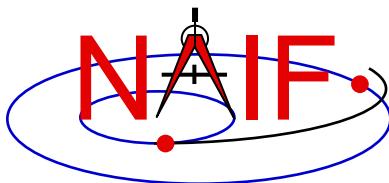
699 = Saturn mass center

Multiple objects

## Satellites - 2

- The user's program must "load" as many of these SPK files as needed to satisfy her/his requirements.
- Sometimes a project NAV team combines (merges) several of these collections before releasing them, making the user's job easier.
- Objects in blue font are the centers of motion for the remaining objects; these don't have ephemeris data included in the file.

See the next page for a graphical representation of this collection of SPKs



# Possible\* SPK File Time Coverages for the Previous Example

Navigation and Ancillary Information Facility

Each bar represents a separate file (SPK kernel)

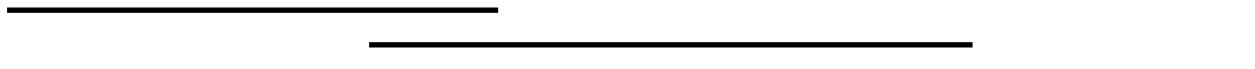
Planet:



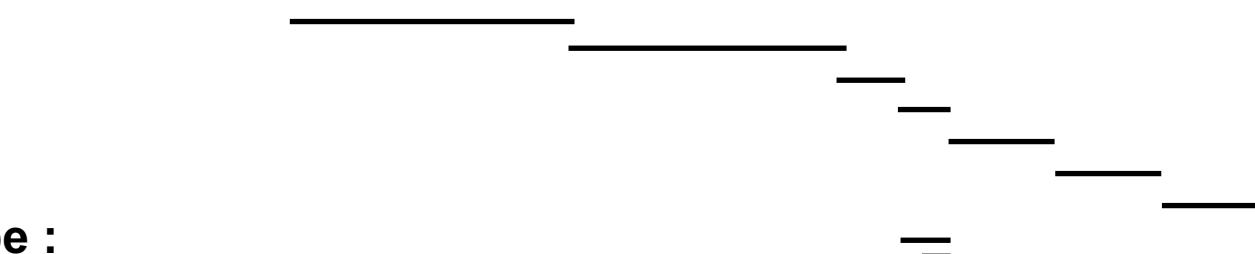
**Satellite - 1:**  
(Major satellites)



**Satellite - 2:**  
(Minor satellites)



**Orbiter :**



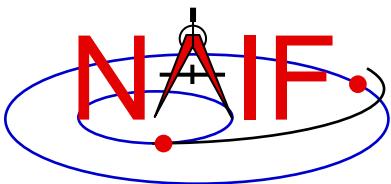
**Huygens Probe :**



Time line:



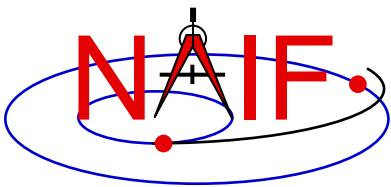
\* Note: This was not the real Cassini scenario—it is simply an illustration  
of some of the possibilities for ephemeris delivery on a planetary mission.



# SPKs for Objects Located on the Surface of a Natural Body

Navigation and Ancillary Information Facility

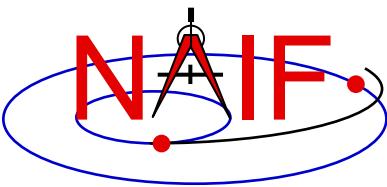
- An SPK file may contain positions of tracking stations, observatories, rovers, etc.
  - The object could be stationary or moving
  - Usually such SPKs contain ephemeris data given in the body-fixed reference frame
- One reads this file the same as for any other SPK file
  - Use the name or NAIF ID of the antenna, observatory or rover as the “target” or “observer” in an SPK reader argument list
  - Also requires use of a SPICE PCK file if you request vectors to be returned in an inertial frame such as J2000; the PCK is needed to rotate body-fixed vectors to the J2000 frame



---

Navigation and Ancillary Information Facility

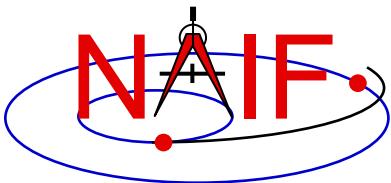
# Using SPK Files



# Retrieving Position or State Vectors

Navigation and Ancillary Information Facility

- To retrieve position or state vectors of ephemeris objects one often needs two kinds of SPICE kernels
  - Ephemeris kernel(s) (SPK)
  - Leapseconds kernel (LSK)
    - » The LSK is used to convert between Coordinated Universal Time (UTC) and Barycentric Dynamical Time (TDB, also called Ephemeris Time, ET)
      - This conversion is done outside of the SPK subsystem
- 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, do frame transformations, and perform aberration corrections
- State and position vectors retrieved from an SPK file by the SPK “reader” routines are Cartesian (rectangular) vectors of the form:
  - X, Y, Z, dX, dY, dZ for a state vector (km, km/sec)
  - X, Y, Z for a position vector (km)



# Retrieving a State Vector

Navigation and Ancillary Information Facility

Initialization...typically done once per program execution

Fortran syntax  
used here

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

`CALL FURNSH ('spk_file_name')`

`CALL FURNSH ('leapseconds_file_name')`



It's better to 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

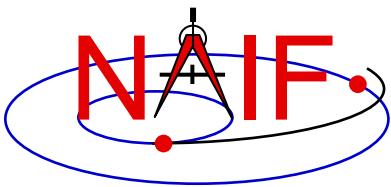
`CALL STR2ET ( 'utc_string', tdb )`

Retrieve state vector from the SPK file at your requested time

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



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



# Arguments of SPKEZR - 1

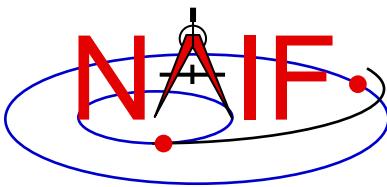
---

Navigation and Ancillary Information Facility

## 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 the observer to the target.
- **TDB:** The time at the observer's location 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 the output state vector is to be given. SPK software will automatically convert ephemeris data to the frame you specified, if needed. SPICE must know the named frame, either built-in or specified using a Frames Kernel.

\* 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 enclosed in quotes.



# Arguments of SPKEZR - 2

---

Navigation and Ancillary Information Facility

- **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. (CN+S is better in some cases.)
  - Use NONE to obtain the uncorrected (aka “geometric”) state, as given by the source SPK file or files.

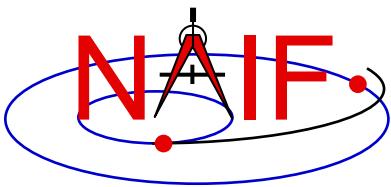
See the header for subroutine SPKEZR, the document SPK Required Reading, or the “Fundamental Concepts” tutorial for details. See the backup charts for examples of aberration correction magnitudes.

## OUTPUTS

- **STATE:** This is the Cartesian state vector you requested. It 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. (Generally not needed.)

LT + S = light time plus stellar aberration

CN + S = converged Newtonian light time plus stellar aberration



# A Simple Example of Retrieving a State Vector

Navigation and Ancillary Information Facility

Fortran syntax  
used here

Initialization - typically do this just once per program execution

```
CALL FURNSH ( 'NAIF0012.TLS' )  
CALL FURNSH ( 'CASSINI_MERGED.BSP' )
```

It's better to replace these two calls with a single call loading a "furnsh kernel" containing the names of all kernel files to load.

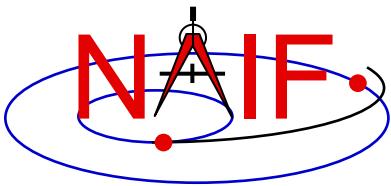
Repeat in a loop if/as needed to solve your particular problem

```
CALL STR2ET ('2004 NOV 21 02:40:21.3', TDB )  
CALL SPKEZR ('TITAN', TDB, 'J2000', 'LT+S', 'CASSINI',  
             STATE, LT )
```

(Insert additional code here to make derived computations such as spacecraft sub-latitude and longitude, lighting angles, etc. Use more SPICE subroutines to help.)

In this example we get the state (STATE) of Titan as seen from the Cassini spacecraft at the UTC epoch 2004 NOV 21 02:40:21.3. The state vector is returned in the J2000 inertial reference frame, which in SPICE is the same as the ICRF frame. The state vector has been corrected for both light time and stellar aberration (LT+S). The one-way light time (LT) is also returned, just in case it could be useful.

A SPICE leapseconds file (NAIF0012.TLS) is used, as is a SPICE ephemeris file (CASSINI\_MERGED.BSP) containing ephemeris data for Cassini (-82), Saturn barycenter (6), Saturn mass center (699), Saturn's satellites (6xx) and the sun (10).

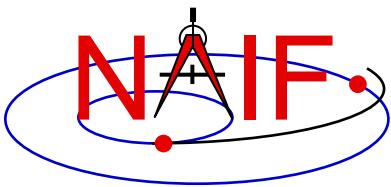


# Retrieving a Position Vector

---

Navigation and Ancillary Information Facility

- **SPKPOS is the position-only analog of SPKEZR**
  - The arguments of SPKPOS are identical to those of SPKEZR, except that SPKPOS returns a 3-component position vector instead of a 6-component state vector
  - SPKPOS executes more quickly than SPKEZR when stellar aberration corrections are used
  - SPKPOS can be used when reference frame transformations of velocity are not possible due to absence of C-kernel angular velocity data

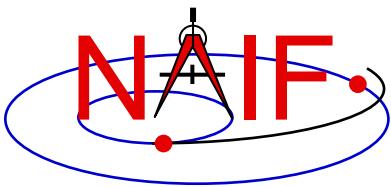


# A Slightly More Complex Example

## Kernel Data Needed

Navigation and Ancillary Information Facility

- To get state vectors referenced to a non-inertial reference frame, or when the data within the SPK file are provided in a non-inertial frame, typically more kernels will be needed.
  - To get the state of an object relative to a body in the body's IAU body-fixed reference frame you'll need:
    - » PCK file containing orientation data for the body
    - » SPK(s) for the object and body
    - » LSK
  - To get the state of an object in a spacecraft-fixed reference frame you'll need:
    - » FK, CK and SCLK for the spacecraft
    - » SPK(s) for the spacecraft and object
    - » LSK



# A Slightly More Complex Example Retrieving a State Vector

Navigation and Ancillary Information Facility

Obtain the state of Titan relative to Cassini in the Titan body-fixed reference frame

Initialization...typically once per program execution

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

```
CALL FURNSH  ('CASSINI_MERGED.BSP')
CALL FURNSH  ('NAIF0012.TLS')
CALL FURNSH  ('NAIF0011.TPC')
```

It's better to replace these three calls with a single call loading a “furnsh kernel” containing the names of all kernel files to load.

Fortran syntax used here

Loop... do as many times as you need

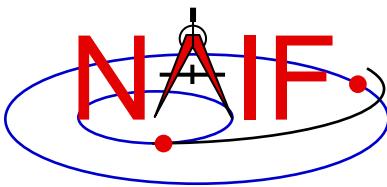
Convert UTC time to ephemeris time (TDB), if needed

```
CALL STR2ET ( '2004 NOV 21 02:40:21.3' , TDB)
```

Get state vector from SPK file at requested time, in satellite's IAU body-fixed frame

```
CALL SPKEZR ('TITAN', TDB, 'IAU_TITAN', 'LT+S', 'CASSINI',
STATE, LT)
```

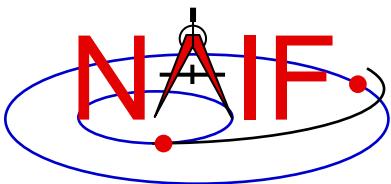
Insert additional code here to make derived computations such as spacecraft sub-latitude and longitude, lighting angles, etc. Use more SPICE subroutines to help.



# Constant-velocity objects: Additional SPK State Computation APIs

Navigation and Ancillary Information Facility

- The SPK subsystem contains routines for computing the state of an ephemeris object with respect to a fixed point or one moving with constant, non-zero velocity, in a specified reference frame.
  - The point may act as either the target or the observer.
  - The center of motion of the point must be an ephemeris object.
- The SPK routines providing this capability are:
  - SPKCPT (SPK, constant position target)
  - SPKCPO (SPK, constant position observer)
  - SPKCVT (SPK, constant velocity target)
  - SPKCVO (SPK, constant velocity observer)
- These routines may provide a convenient alternative to creating SPK files for surface points:
  - when there's a need to compute the locations on the fly
  - when the number of surface points is large

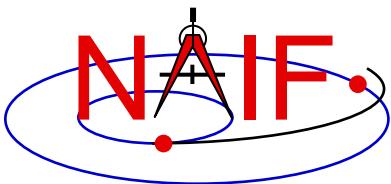


# Manipulating and Using SPK Files

Navigation and Ancillary Information Facility

- You can subset an SPK, or merge two or more SPKs
  - The subset or merge may be keyed off of objects, or time, or both
  - Merging only portions of SPKs is possible
- You can read data from just one, or many\* SPK files in your application program
  - Don't forget the precedence rule: data in a later loaded file take precedence over data from an earlier loaded file
- You can convert an SPK that is in non-native binary format to native binary format if you need to add data or comments

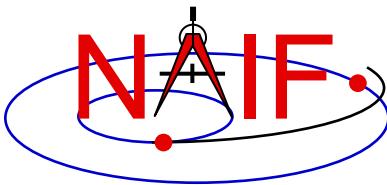
\* The allowed number of simultaneously loaded DAF- and DAS-based files is set to 5000 in N66 Toolkits. “DAF” is the acronym for Double Precision Array File. SPKs are based on the DAF architecture.



# Understanding an SPK File

Navigation and Ancillary Information Facility

- The SPK producer should have provided descriptive meta-data inside an SPK file, in the “comment area”
  - The comments should say when, why, how and for what purpose the file was made
  - Additional useful information could also be provided by the producer
    - » Example: when and why any data gaps are present
- These comments may be extracted using an API (subroutine) or viewed using a SPICE utility program.
  - API: DAFEC
  - Utility program: commnt –r <spk\_file\_name>



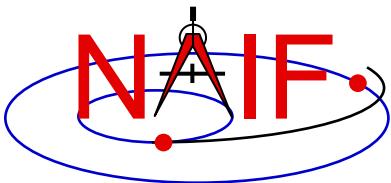
# SPK Utility Programs

Navigation and Ancillary Information Facility

- The following SPK utility programs are included in the Toolkit:

BRIEF	summarizes coverage for one or more SPK files
SPACIT	generates segment-by-segment summary of an SPK file
COMMNT	reads, appends, or deletes comments in an SPK file
MKSPK	converts ephemeris data provided in a text file into an SPK file
SPKDIFF	compares two SPK files
SPKMERGE	subsets an spk, or merges one or more SPK files or pieces
- These additional SPK utility programs are provided on the NAIF Web site (<http://naif.jpl.nasa.gov/naif/utilities.html>)

SPY	validates, inspects, and analyses SPK files
PINPOINT	creates an SPK file for fixed locations (ground stations, etc)
BSPIDMOD	alters body IDs in an SPK file
DAFMOD	alters body or frame IDs in an SPK file
DAFCAT	concatenates together SPK files
BFF	displays binary file format of an SPK file
BINGO	converts SPK files between big- and little-endian formats



# Summarizing an SPK File - 1

Navigation and Ancillary Information Facility

- A summary of the contents and time coverage of an SPK file can be made using the SPICE Toolkit utility “brief”
  - See the brief User’s Guide for details

```
% brief 070413BP SCPSE 07097 07121.bsp
```

```
Summary for: 070413BP SCPSE 07097 07121.bsp
```

Bodies:	CASSINI (-82)	PLUTO BARYCENTER (9)	TETHYS (603)
	MERCURY BARYCENTER (1)	SUN (10)	DIONE (604)
	VENUS BARYCENTER (2)	MERCURY (199)	RHEA (605)
	EARTH BARYCENTER (3)	VENUS (299)	TITAN (606)
	MARS BARYCENTER (4)	MOON (301)	HYPERION (607)
	JUPITER BARYCENTER (5)	EARTH (399)	IAPETUS (608)
	SATURN BARYCENTER (6)	MARS (499)	PHOEBE (609)
	URANUS BARYCENTER (7)	MIMAS (601)	SATURN (699)
	NEPTUNE BARYCENTER (8)	ENCELADUS (602)	

Start of Interval (ET)

-----

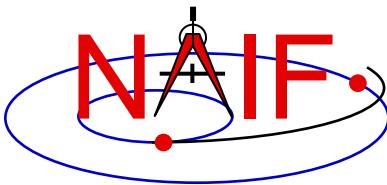
2007 APR 07 16:22:23.000

End of Interval (ET)

-----

2007 MAY 01 09:34:03.000

Note, the default  
time system is  
ET, not UTC!



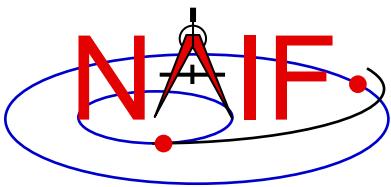
# Summarizing an SPK File - 2

Navigation and Ancillary Information Facility

- **Use of the “-c” option when using the *brief* utility will show you the center of motion for each object**
  - This is often useful in diagnosing an SPK chaining problem
    - » See the “Problems” section at the end of this tutorial for more information

```
% brief -c 070413BP_SCPSSE_07097_07121.bsp

Bodies: CASSINI (-82) w.r.t. SATURN BARYCENTER (6)
        MERCURY BARYCENTER (1) w.r.t. SOLAR SYSTEM BARYCENTER (0)
        VENUS BARYCENTER (2) w.r.t. SOLAR SYSTEM BARYCENTER (0)
        EARTH BARYCENTER (3) w.r.t. SOLAR SYSTEM BARYCENTER (0)
        MARS BARYCENTER (4) w.r.t. SOLAR SYSTEM BARYCENTER (0)
        JUPITER BARYCENTER (5) w.r.t. SOLAR SYSTEM BARYCENTER (0)
        SATURN BARYCENTER (6) w.r.t. SOLAR SYSTEM BARYCENTER (0)
        URANUS BARYCENTER (7) w.r.t. SOLAR SYSTEM BARYCENTER (0)
        NEPTUNE BARYCENTER (8) w.r.t. SOLAR SYSTEM BARYCENTER (0)
        PLUTO BARYCENTER (9) w.r.t. SOLAR SYSTEM BARYCENTER (0)
        SUN (10) w.r.t. SOLAR SYSTEM BARYCENTER (0)
        MERCURY (199) w.r.t. MERCURY BARYCENTER (1)
        VENUS (299) w.r.t. VENUS BARYCENTER (2)
        MOON (301) w.r.t. EARTH BARYCENTER (3)
        EARTH (399) w.r.t. EARTH BARYCENTER (3)
        MARS (499) w.r.t. MARS BARYCENTER (4)
        MIMAS (601) w.r.t. SATURN BARYCENTER (6)
        ENCELADUS (602) w.r.t. SATURN BARYCENTER (6)
        TETHYS (603) w.r.t. SATURN BARYCENTER (6)
        DIONE (604) w.r.t. SATURN BARYCENTER (6)
        RHEA (605) w.r.t. SATURN BARYCENTER (6)
        TITAN (606) w.r.t. SATURN BARYCENTER (6)
        HYPERION (607) w.r.t. SATURN BARYCENTER (6)
        IAPETUS (608) w.r.t. SATURN BARYCENTER (6)
        PHOEBE (609) w.r.t. SATURN BARYCENTER (6)
        SATURN (699) w.r.t. SATURN BARYCENTER (6)
        Start of Interval (ET)           End of Interval (ET)
-----
```



# Summarizing an SPK File - 3

Navigation and Ancillary Information Facility

- A detailed summary of an SPK can be made using the Toolkit utility named “**SPACIT**”
- See the **SPACIT User’s Guide** for details

```
Summary for SPK file: sat240.bsp
Leapseconds File      : /kernels/gen/lsk/leapseconds.ker
Summary Type          : Entire File
```

---

```
Segment ID      : SAT240
Target Body     : Body 601, MIMAS
Center Body     : Body 6, SATURN BARYCENTER
Reference frame: Frame 1, J2000
SPK Data Type   : Type 3
Description     : Fixed Width, Fixed Order Chebyshev Polynomials: Pos, Vel
UTC Start Time : 1969 DEC 31 00:00:00.000
UTC Stop Time  : 2019 DEC 02 00:00:00.000
ET Start Time  : 1969 DEC 31 00:00:41.183
ET Stop time   : 2019 DEC 02 00:01:05.183
```

---

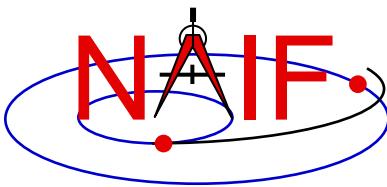
---

```
Segment ID      : SAT240
Target Body     : Body 602, ENCELADUS
Center Body     : Body 6, SATURN BARYCENTER
Reference frame: Frame 1, J2000
SPK Data Type   : Type 3
Description     : Fixed Width, Fixed Order Chebyshev Polynomials: Pos, Vel
UTC Start Time : 1969 DEC 31 00:00:00.000
UTC Stop Time  : 2019 DEC 02 00:00:00.000
ET Start Time  : 1969 DEC 31 00:00:41.183
ET Stop time   : 2019 DEC 02 00:01:05.183
```

---

:

(This is a partial output; not all data could be displayed on this chart)



# Summarizing an SPK File - 4

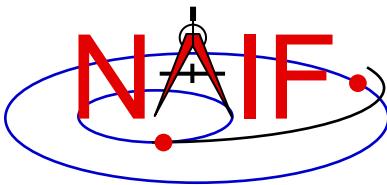
Navigation and Ancillary Information Facility

## Summarizing an SPK at the API Level

- Call **SPKOBJ** to find the set of objects for which a specified SPK provides data.
  - INPUT: an SPK file name and initialized SPICE integer “Set” data structure. The set may optionally contain ID codes obtained from previous calls.
  - OUTPUT: the input set, to which have been added (via set union) the ID codes of objects for which the specified SPK provides coverage.

```
CALL SPKOBJ ( SPK, IDSET )
```
- Call **SPKCOV** to find the window of times for which a specified SPK file provides coverage for a specified body:
  - INPUT: an SPK file name, body ID code and initialized SPICE double precision “Window” data structure. The window may optionally contain coverage data from previous calls.
  - OUTPUT: the input window, to which have been added (via window union) the sequence of start and stop times of segment coverage intervals of the specified SPK, expressed as seconds past J2000 TDB.

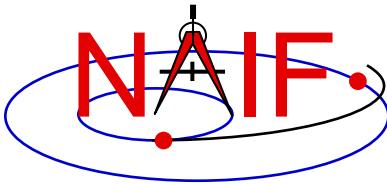
```
CALL SPKCOV ( SPK, IDCODE, COVER )
```
- See the headers of these routines for example programs.
- Also see the CELLS, SETS and WINDOWS Required Reading for background information on these SPICE data types.



# Additional Information on SPK

Navigation and Ancillary Information Facility

- **For more information about SPK, look at the following:**
  - The remainder of this tutorial (the BACKUP section)
  - Most Useful Routines document
  - SPK Required Reading document
  - Headers of the subroutines mentioned
  - Using Frames tutorial
  - BRIEF and SPKDIFF User's Guides
- **Related documents:**
  - NAIF\_IDS Required Reading
  - Frames Required Reading
  - Time Required Reading
  - Kernel Required Reading

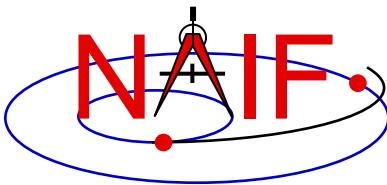


# Backup

---

Navigation and Ancillary Information Facility

- **Problems Using SPK Files**
- **Don't Mix Planet Ephemerides**
- **Effect of Aberration Corrections**
- **Examples of Retrieving State Vectors**
- **SPK File Structure**



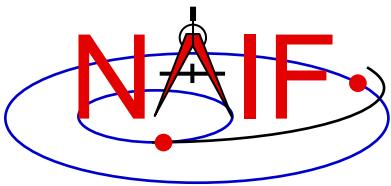
# Problems Using SPK Files - 1

Navigation and Ancillary Information Facility

- **The file, or files, you loaded do not contain data for both your target and observer bodies**
  - You may have loaded the wrong file, or assumed the file contains data that it doesn't
  - You may not have loaded all the files needed
- **The file, or files, you loaded do not cover the time at which you requested a state vector**
  - This could occur if you've been given a file coverage summary in calendar ET form and you mistook this for UTC
    - (  $ET = UTC + DELTAET$ , where  $DELTAEET$  is about 69 seconds as of 1/20)
  - This could occur if you are requesting a light-time corrected state and the SPK files being used do not have data at the time that is one-way light-time away\* from your ET epoch of interest
    - » \* Earlier, for the receive case; later, for the transmission case
- **In the above situations you'll get an error message like the following:**

SPICE (SPKINSUFFDATA) - -

Insufficient ephemeris data has been loaded to compute the state of xxx relative to yyy.

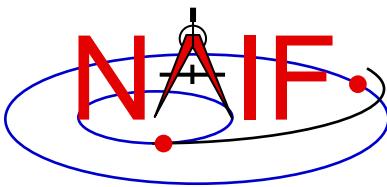


# Problems Using SPK Files - 2

Navigation and Ancillary Information Facility

- You have requested aberration-corrected states but the SPK file or files you loaded do not contain sufficient data to relate both your target and observer bodies back to the solar system barycenter, which is required for this calculation.
  - You may not have loaded all the files needed
  - You may have assumed the file contains data that it doesn't
- In the above situations you'll get an error message like the following:

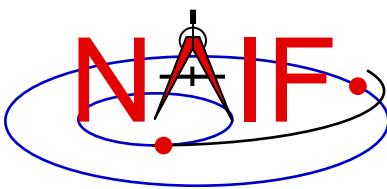
```
SPICE(SPKINSUFFDATA) - -
Insufficient ephemeris data has been loaded to
compute the state of xxx relative to yyy.
```



# Problems Using SPK Files – 3a

Navigation and Ancillary Information Facility

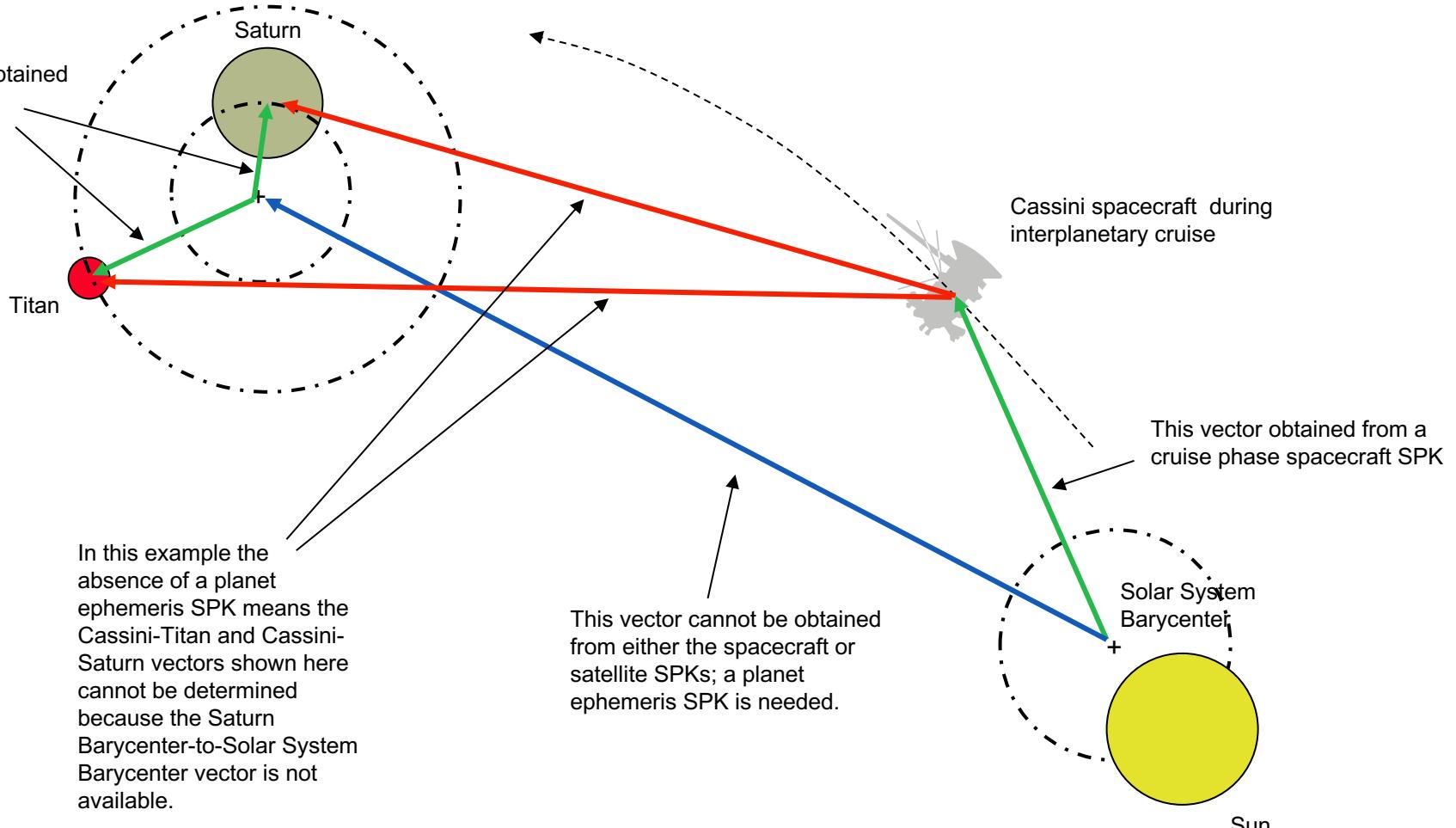
- An infrequent problem occurs when your SPK file(s) contain data for both target and observer, and cover the period of interest, but ephemeris data for an intermediate body needed to chain the target and observer together is missing.
  - Example: You load a spacecraft SPK containing ephemeris for Cassini (-82) relative to the solar system barycenter (0), and you load a satellite SPK containing the ephemeris for Titan (606) and Saturn (699) relative to the Saturn barycenter (6). But you forgot to load a planet SPK file that contains data for the Saturn barycenter (6) relative to the solar system barycenter (0). The SPK software cannot “connect” Cassini to Titan or to Saturn. (See the drawing on the next page.)
  - In this case, knowing what is the “Center Body” of movement for each target body is important; this is shown in SPACIT, and also in BRIEF summaries when the -c command line option is used.
  - This problem is illustrated on the next page.



# Problems Using SPK Files - 3b (drawing)

## Navigation and Ancillary Information Facility

These vectors obtained from a satellite ephemeris SPK

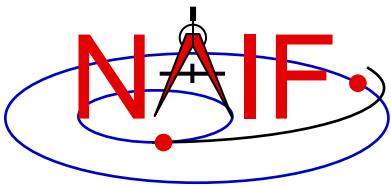


Given...

- available in SPK files
- not available in SPK files

Therefore...

- can't be computed

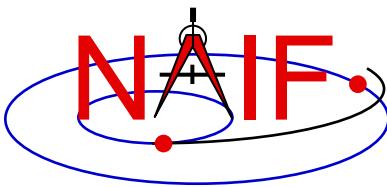


# Problems Using SPK Files - 4

---

Navigation and Ancillary Information Facility

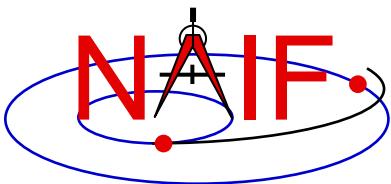
- You see an error message to the effect that pole RA (right ascension) data cannot be found
  - You are requesting results in a body-fixed frame, but you have not loaded a SPICE PCK file that defines this frame.



# Problems Using SPK Files - 5

Navigation and Ancillary Information Facility

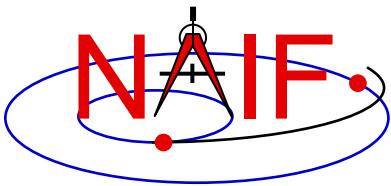
- **Segment Masking: You've loaded sufficient data to chain together the target and observer, but the SPK subsystem can't make the connection.**
  - This can happen when a high-priority segment that can't be connected to both target and observer "masks" a lower-priority segment that can be connected.
  - Example: you want the state of earth as seen from the Galileo orbiter at a specified ephemeris time ET1.
    - » You have loaded SPK files providing:
      - the state of the Galileo orbiter relative to the asteroid Gaspra
      - the state of the orbiter relative to the sun
      - the state of the earth relative to the earth-moon barycenter
      - the states of the sun and earth-moon barycenter relative to the solar system barycenter
    - » If an SPK segment for the orbiter relative to Gaspra covering ET1 has higher priority than the segment for the orbiter relative to the sun covering ET1, no connection between the orbiter and the earth will be made.
    - » Solution:
      - Load an SPK file providing the ephemeris of Gaspra relative to the sun or the solar system barycenter (for a time interval containing ET1)



# Problems Using SPK Files - 6

Navigation and Ancillary Information Facility

- Other missing data... not obvious.
  - You may need CK (and SCLK), FK or PCK data to construct your requested output frame.
- Mistaking ET for UTC, or vice-versa: **very common**
- Using light time corrections requires target ephemeris data at the light time-corrected epoch.
  - If you're working near the beginning of an SPK file, the light time-corrected epoch may occur earlier than available data.



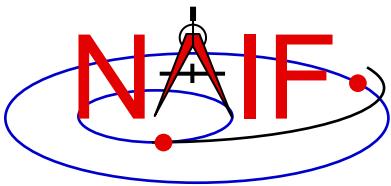
# Problems Using SPK Files - 7

Navigation and Ancillary Information Facility

- You've assumed that:

$$\text{state (observer, target)} = \overset{\text{negative sign}}{-} \text{state (target, observer)}$$

- This is NOT true unless you have requested geometric states in both cases (i.e. no light time or stellar aberration corrections are applied)

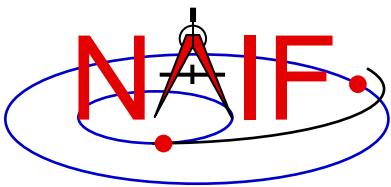


# Problems Using SPK Files - 8

---

Navigation and Ancillary Information Facility

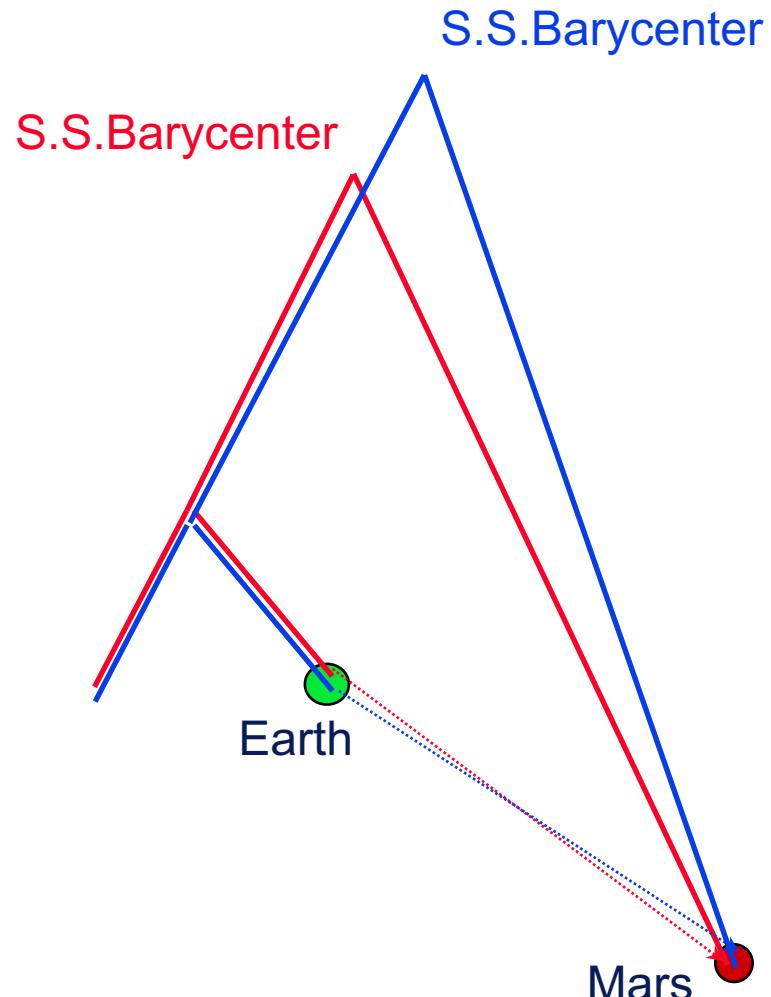
- **SPK reading efficiency can be impacted by any of several circumstances.**
  - This won't result in erroneous results... just slow performance.
  - This sort of problem does not occur very often.
  - For details, read some of the backup charts in the tutorial named **Making an SPK**—those titled "**SPK Reading: Efficiency Issues**."

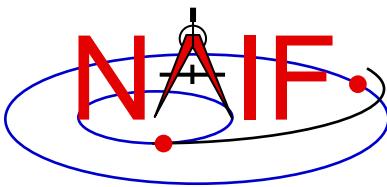


# Don't Mix Planet Ephemerides-1

Navigation and Ancillary Information Facility

- With each new version of the JPL planetary ephemeris, the solar system barycenter moves with respect to the planets
- Changes in relative planet positions are much smaller than changes in the planet locations relative to the solar system barycenter

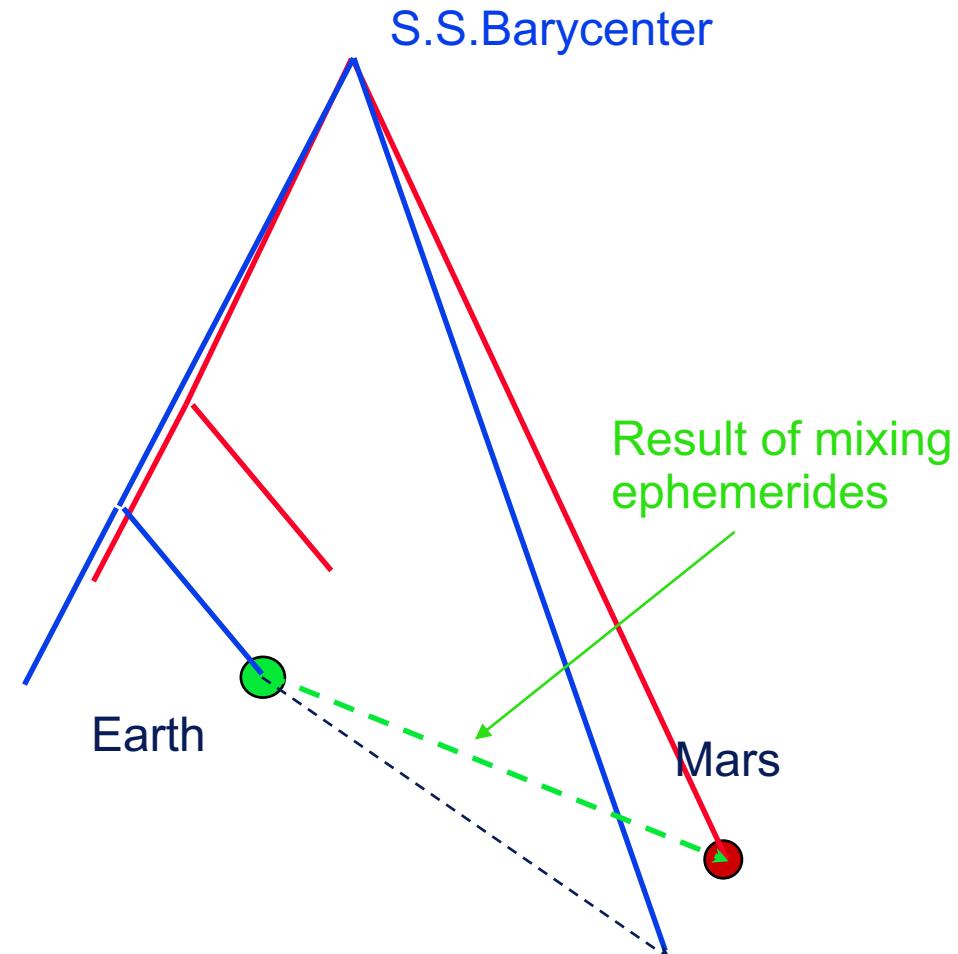


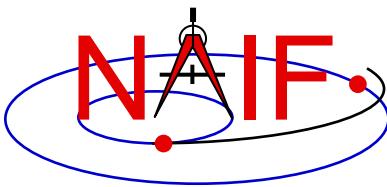


# Don't Mix Planet Ephemerides-2

Navigation and Ancillary Information Facility

- SPICE allows you to “load” different planetary ephemerides (or portions of them)
  - » You can potentially subtract the solar system barycenter-relative positions from different ephemerides to get relative states
- Don’t mix planetary ephemerides
- For JPL flight projects, a consistent set of ephemerides is provided to the mission team



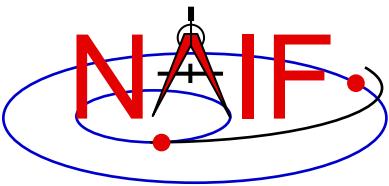


# Effect of Aberration Corrections - 1

---

Navigation and Ancillary Information Facility

- **Angular offsets between corrected and uncorrected position vectors over the time span 2004 Jan 1 to 2005 Jan 1**
  - Mars as seen from MEX:
    - » LT+S vs NONE: .0002 to .0008 degrees
    - » LT vs NONE: .0006 to .0047 degrees
  - Earth as seen from MEX:
    - » LT+S vs NONE: .0035 to .0106 degrees
    - » LT vs NONE: .0000 to .0057 degrees
  - MEX as seen from Earth:
    - » LT+S vs NONE: .0035 to .0104 degrees
    - » LT vs NONE: .0033 to .0048 degrees
  - Sun as seen from Mars:
    - » LT+S vs NONE: .0042 to .0047 degrees
    - » LT vs NONE: .0000 to .0000 degrees

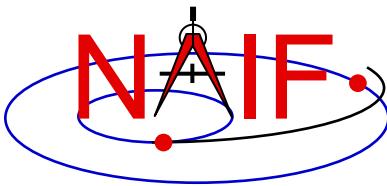


# Effect of Aberration Corrections - 2

---

Navigation and Ancillary Information Facility

- Angular offsets between corrected and uncorrected position vectors over the time span 2004 Jan 1 to 2008 Jan1
  - Saturn as seen from CASSINI:
    - » LT+S vs NONE: .0000 to .0058 degrees
    - » LT vs NONE: .0001 to .0019 degrees
  - Titan as seen from CASSINI:
    - » LT+S vs NONE: .0000 to .0057 degrees
    - » LT vs NONE: .0000 to .0030 degrees
  - Earth as seen from CASSINI:
    - » LT+S vs NONE: .0000 to .0107 degrees
    - » LT vs NONE: .0000 to .0058 degrees
  - CASSINI as seen from Earth:
    - » LT+S vs NONE: .0000 to .0107 degrees
    - » LT vs NONE: .0000 to .0059 degrees
  - Sun as seen from CASSINI:
    - » LT+S vs NONE: .0000 to .0059 degrees
    - » LT vs NONE: .0000 to .0000 degrees

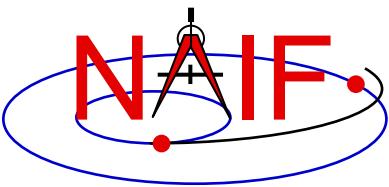


# Examples of Retrieving State Vectors – 1

Navigation and Ancillary Information Facility

- Example: find the geometric state of the MGS orbiter relative to Mars at the observation epoch ET, expressed in the J2000 reference frame.
  - CALL SPKEZR ( 'MGS', ET, 'J2000', 'NONE', 'MARS', STATE, LT )
  - The SPK subsystem locates an SPK segment containing the ephemeris of the orbiter relative to Mars covering epoch ET, interpolates the ephemeris data at ET, and returns the interpolated state vector.
- Example: find the geometric state of Titan relative to the earth at epoch ET, expressed in the J2000 reference frame.
  - CALL SPKEZR ( 'TITAN', ET, 'J2000', 'NONE', 'EARTH', STATE, LT )
  - The SPK subsystem looks up and interpolates ephemeris data to yield:
    - » The state of the earth relative to the earth-moon barycenter (A)
    - » The state of the earth-moon barycenter relative to the solar system barycenter (B)
    - » The state of Titan relative to the Saturn system barycenter at ET (C)
    - » The state of the Saturn system barycenter relative to the solar system barycenter at ET (D)
  - SPKEZR then returns the state vector
    - »  $C + D - (A + B)$

Fortran syntax  
used here

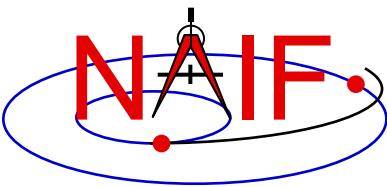


## Examples of Retrieving State Vectors – 2

Navigation and Ancillary Information Facility

- **Example: find the apparent state of the Cassini orbiter relative to the DSN station DSS-14, expressed in the topocentric reference frame centered at DSS-14, at a specified observation epoch ET.**
  - CALL SPKEZR ( 'CASSINI', ET, 'DSS-14\_TOPO',  
                  'LT+S',   'DSS-14', STATE, LT )
  - The SPK subsystem looks up and interpolates ephemeris data to yield:
    - » The state of DSS-14 relative to the earth in the ITRF93 terrestrial reference frame (A)
    - » The state at ET of the earth relative to the earth-moon barycenter in the J2000 reference frame (B)
    - » The state at ET of the earth-moon barycenter relative to the solar system barycenter in the J2000 frame (C)
    - » The state at the light time-corrected epoch ET-LT of the Cassini orbiter relative to the Saturn system barycenter (other centers are possible) in the J2000 frame (D)

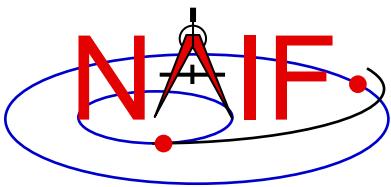
*continued on next page*



## Examples of Retrieving State Vectors – 3

Navigation and Ancillary Information Facility

- » The state at ET-LT of the Saturn system barycenter relative to the solar system barycenter in the J2000 frame (E)
- The SPK subsystem also looks up transformation matrices to map states:
  - » From the J2000 frame to the ITRF93 terrestrial (earth body-fixed) frame at the observation epoch ET (T1)
  - » From the ITRF93 terrestrial frame to the DSS-14-centered topocentric frame (T2)
- SPKEZR then computes the J2000-relative, light-time corrected observer-target state vector
  - »  $E + D - ( (T1)^{-1} * A + B + C )$
- SPKEZR corrects this vector for stellar aberration
  - » Call the result "V\_J2000\_apparent"
- and finally returns the requested state vector in the DSS-14 topocentric reference frame
  - » **STATE = T2 \* T1 \* V\_J2000\_apparent**

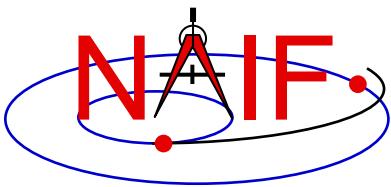


# SPK File Structure

---

Navigation and Ancillary Information Facility

- A description of the SPK file structure is shown near the beginning of the “Making an SPK” tutorial.

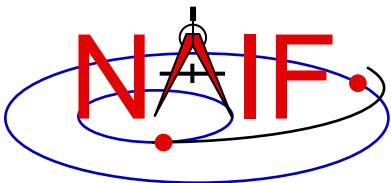


---

Navigation and Ancillary Information Facility

# Planetary Constants Kernel PCK

January 2020

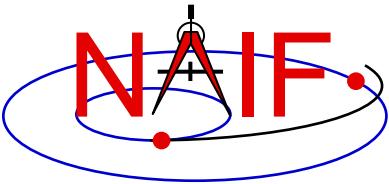


# Topics

---

Navigation and Ancillary Information Facility

- **Overview**
- **Text PCK Orientation Models**
- **Binary PCK Orientation Models**
- **PCK reference frames**
- **PCK Shape Models**
- **Using PCKs**
- **Interface Routines**
- **PCK Precedence Rules**
- **PCK Utility Programs**

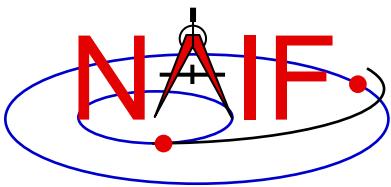


# Overview

---

Navigation and Ancillary Information Facility

- **The Planetary Constants Kernel (PCK) subsystem comprises both text and binary kernels.**
  - **Text PCKs provide orientation and shape models for the sun, planets, natural satellites and a few asteroids and comets.**
  - **Binary PCKs are used only when very high accuracy orientation data are available.**
    - » Currently available only for the earth and the moon
    - » One still needs to use a text-style PCK to get shape data



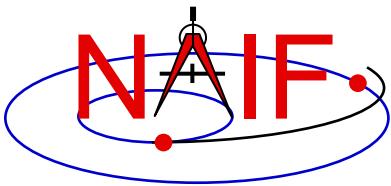
# Text PCKs - 1

---

Navigation and Ancillary Information Facility

- **Text PCK files contain size, shape and orientation data associated with natural solar system bodies: planets, satellites, and a few comets and asteroids.**
  - Some additional kinds of data might also be included.
- **NAIF creates and distributes a “generic” text PCK based on the latest IAU/IAG Report.\***
  - The reports are issued about once every three years, and so might not contain the very latest available results.
- **SPICE PCK software is designed to use these data to compute orientation of body-fixed, body-centered frames.**
  - These frames have a name style of “IAU\_*body-name*”
- **NAIF also provides a “masses” PCK, containing GM values for the Sun and planetary systems.**
  - Values from this file are typically used with SPICE osculating element routines, and in using the MKSPK application to make a Type 5 SPK file.
- **Text PCKs are sometimes produced by flight projects and others—not only by NAIF.**

\* “Report of the IAU/IAG Working Group on cartographic coordinates and rotational elements: <year issued>”; published in *Celestial Mechanics and Dynamical Astronomy*



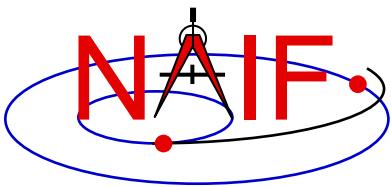
# Text PCKs - 2

---

Navigation and Ancillary Information Facility

- **The SPICE text kernel mechanism is used to implement PCK files.**
  - Kernel variables contain the mathematical terms appearing in rotation or shape models. For example:

```
BODY699_POLE_RA = ( 40.589   -0.036      0. )
BODY699_POLE_DEC = ( 83.537   -0.004      0. )
BODY699_PM =        ( 38.90    810.7939024 0. )
BODY699_RADII     = ( 60268    60268    54364  )
```
  - Users may easily inspect data in text PCKs.
  - Users may (carefully!) modify text PCKs with a text editor.
    - » Data or comments may be added, deleted, or changed.
    - » Comments should be added to explain changes .
  - The user may include additional kernel variables to change the base frame or reference epoch.
  - Kernel variable names are **case-sensitive**.
    - » NAIF uses only upper case for variable names; we suggest you do the same.



# Text PCK Orientation Models - 1

Navigation and Ancillary Information Facility

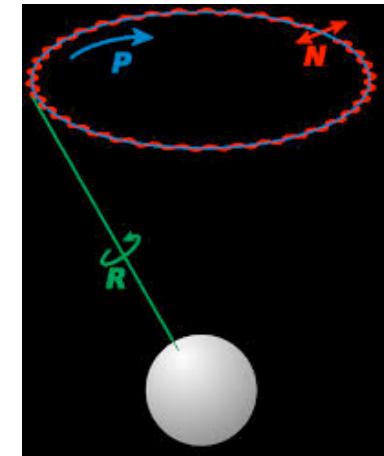
- **For the sun, planets and a few major asteroids:**

- PCK models use low-degree (typically linear) polynomials to represent RA and DEC of the pole (body-fixed +Z-axis) as a function of time.
- The prime meridian is also represented by a low-degree polynomial.
- For a few planets, trigonometric polynomial terms are used to more accurately represent precession and nutation of the pole.

R = rotation of the body about its rotational axis

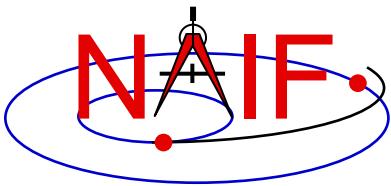
P = precession of the bodies' rotational axis

N = nutation of the bodies' rotational axis



- **For natural satellites:**

- In addition to low-degree polynomials for the spin axis and prime meridian, trigonometric polynomial terms are used to more accurately represent precession and nutation.
- A few satellites have chaotic rotation and so are not modeled.

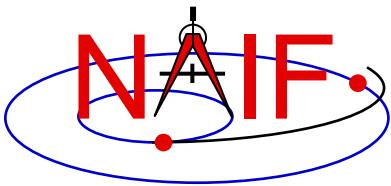


# Text PCK Orientation Models - 2

---

Navigation and Ancillary Information Facility

- The base frame for PCK orientation models is the International Celestial Reference Frame (ICRF), as defined by the International Earth Rotation Service (IERS).
  - For historical and backwards compatibility reasons, SPICE uses the name “J2000” as a synonym for the ICRF inertial reference frame, even though J2000 and ICRF are, in fact, not identical. (The difference is well under 0.1 arc second.)

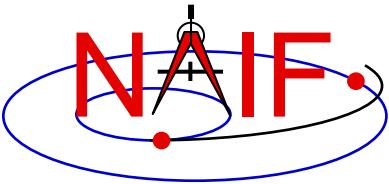


# Text PCK Orientation Models - 3

Navigation and Ancillary Information Facility

- **Body-fixed frames provided in text PCKs have +Z axes consistent with planetocentric coordinate systems. The +X axes of these frames coincide with planetocentric longitude 0.**
- **For planets and satellites the +Z axis (+90 LAT) always points to the north side of the invariable plane – the plane whose normal vector is the angular momentum vector of the solar system.**
  - Planetocentric longitude increases positively eastward
  - Planetocentric latitude increases positively northward
- **Dwarf planets\*, asteroids and comets spin in the right hand sense about their “positive pole.”**
  - What the IAU now calls the “positive pole” is still referred to as the “north pole” in SPICE documentation.
  - The “positive pole” may point above or below the invariable plane of the solar system (see above).
  - This revision by the IAU Working Group (2006) inverts what had been the direction of the north pole for Pluto, Charon and Ida.

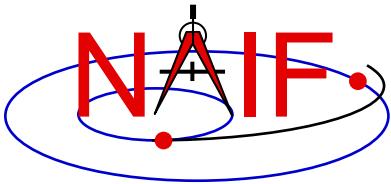
\*The dwarf planets are: Ceres, Pluto, Haumea, Makemake, Eris



# Binary PCK Orientation Models

Navigation and Ancillary Information Facility

- When available, the SPICE system can store high-accuracy orientation model data in binary PCKs.
- Binary PCKs are limited to storing orientation data.
  - Applications that require shape data must also load a text PCK.
- Orientation data from a binary PCK always supersedes orientation data for the same object obtained from a text PCK, no matter the order in which the kernels are loaded.
- Binary PCKs for the earth and the moon are available from the NAIF server.
  - The accuracy of these is much better than what is provided in the generic text PCK.
  - See the tutorial "lunar-earth\_pck-fk" for details.

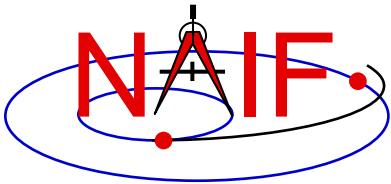


# Location of Text PCK Reference Frame Specifications

---

Navigation and Ancillary Information Facility

- Many PCK reference frame specifications are built into SPICE. Examples are IAU\_SATURN and IAU\_TITAN.
  - To use these, load a text PCK file containing orientation data for the body of interest.
    - » Typically this is the current generic text PCK
  - Be very cautious about using IAU\_EARTH and IAU\_MOON; the binary PCKs for these two bodies offer much more accuracy.
  - Data for a small number of comets and asteroids are included.
- Other PCK frames are not built-in and must be defined in a frames kernel that is loaded by your program. Examples are body fixed frames for asteroids or “newer” natural satellites.
  - See the Frames Required Reading technical reference for information on creating frame kernels that specify PCK reference frames.

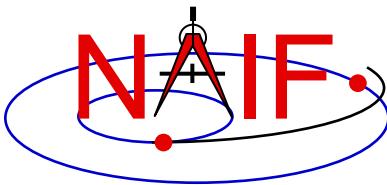


# Location of Binary PCK Reference Frame Specifications

---

Navigation and Ancillary Information Facility

- **Special high-accuracy earth and lunar body-fixed frames are realized using binary PCKs.**
  - These frames are named:
    - » For the earth: ITRF93
    - » For the moon: MOON\_PA and MOON\_ME
- **To use high-accuracy earth or moon orientation, load the appropriate binary PCK and allied FK.**
  - See the special tutorial “lunar-earth\_pck-fk” for details on these.

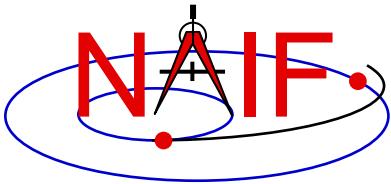


# PCK Shape Models

---

Navigation and Ancillary Information Facility

- **PCK shape models are nominally triaxial ellipsoids**
  - For many bodies, the two equatorial axes have the same value; these bodies have a spheroidal shape.
  - For some bodies, one or more radii have not been determined.
  - See the DSK tutorial for information about other kinds of shape models available within SPICE.
- **Although many bodies are in fact modeled as spheres or spheroids, SPICE usually deals with the general, triaxial case.**
  - **Exception: SPICE supports geodetic coordinate transformations only for bodies modeled as spheres or spheroids.**
    - » RECGEO, GEOREC, DGEODR, DRDGEO and XFMSTA are the modules performing these transformations.
  - **Exception: SPICE supports planetographic coordinate transformations only for bodies modeled as spheres or spheroids.**
    - » PGRREC, RECPGR, DPGRDR, DRDPGR and XFMSTA are the modules supporting these transformations.

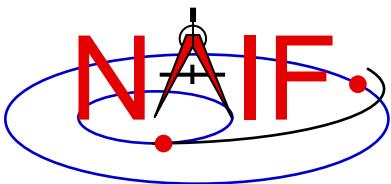


# Using PCK Data

Navigation and Ancillary Information Facility

- PCK orientation data are usually accessed using frame subsystem or ephemeris subsystem APIs.
  - Example: Get the IAU\_SATURN body-fixed reference frame to J2000 position or state transformation matrix at ET:
    - » CALL PXFORM ( 'IAU\_SATURN', 'J2000', ET, RMAT )
    - » CALL SXFORM ( 'IAU\_SATURN', 'J2000', ET, XFORM )
  - Example: Get the state of Saturn relative to Cassini in the IAU\_SATURN body-fixed reference frame:
    - » CALL SPKEZR ( 'SATURN', ET, 'IAU\_SATURN', 'LT+S', 'CASSINI', STATE, LT )
- PCK shape data are usually accessed using APIs needing size and shape data such as SUBPT, SUBSLR, ILUMIN, etc.

Fortran  
examples



# Interface Routines - 1

Navigation and Ancillary Information Facility

- Call FURNISH to load PCKs.
  - CALL UNLOAD or KCLEAR to unload them.
- Call SXFORM to return a state transformation.
  - Returns 6x6 matrix (attitude and angular velocity)
    - » CALL SXFORM ( FROM, TO, ET, XFORM )
- Call PXFORM to return a position transformation.
  - Returns 3x3 matrix (attitude only)
    - » CALL PXFORM ( FROM, TO, ET, RMAT )
- Get state of Saturn relative to Cassini in the IAU\_SATURN body-fixed reference frame:
  - CALL SPKEZR ( 'SATURN', ET, 'IAU\_SATURN', 'LT+S', 'CASSINI', STATE, LT )
- Get state of Cassini relative to the DSN station DSS-13 in the J2000 inertial reference frame:
  - CALL SPKEZR ( 'CASSINI', ET, 'J2000', 'LT+S', 'DSS-13', STATE, LT )
    - » An Earth PCK **must** be loaded in order for this call to work, even though the requested output reference frame is inertial.
      - That's because, in the course of its work, this call must convert the position of the DSN station relative to the Earth's center from an Earth-fixed, earth-centered frame to the J2000 frame.

Fortran  
examples

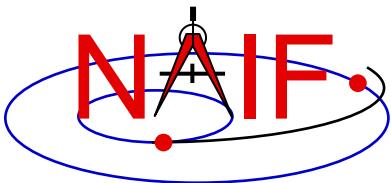


# Interface Routines - 2

Navigation and Ancillary Information Facility

Fortran  
examples

- Call **BODVRD** or **BODVCD** to retrieve constants associated with a body. For example:
  - `CALL BODVRD ( 'SATURN' , 'RADII' , 3, N, RADII )`
  - `CALL BODVCD ( 699 , 'RADII' , 3, N, RADII )`
  - These calls retrieve values associated with the variable `BODY699_RADII`.
  - The variable name is **case-sensitive**, so the string, `RADII`, above must be in upper case.
- You can use general kernel pool fetch routines to fetch data assigned to any non-standard names.
  - `GCPOOL`, for character data
  - `GDPOOL`, for double precision data
  - `GIPOOL`, for integer data

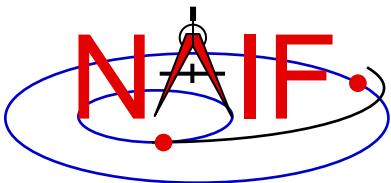


# PCK Precedence Rules

---

Navigation and Ancillary Information Facility

- In text PCKs, assignments are of two types:
  - » “Direct”: variable name = value(s)
  - » “Incremental”: variable name += value(s)
- The last direct assignment made to a given variable replaces any/all previous assignments for that variable.
- Incremental assignments simply add additional values to an existing variable.
  - » The variable will be newly created if it didn’t already exist.
- Orientation data from a binary PCK always supersedes orientation data (for the same object) obtained from a text PCK, no matter the order in which the kernels have been loaded.



# PCK Utility Programs

---

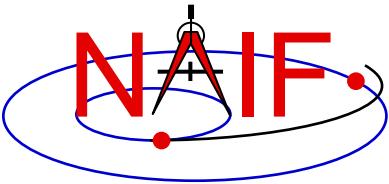
Navigation and Ancillary Information Facility

- These utilities are included in the Toolkit.

BRIEF	summarizes coverage for one or more <u>binary</u> PCK files
SPACIT	generates segment-by-segment summary of a <u>binary</u> PCK file
COMMNT	reads, appends, or deletes comments in a <u>binary</u> PCK file
FRMDIFF	samples a PCK-based frame or compares orientation of two PCK-based frames (binary or text PCKs)

- These additional utilities are provided on the NAIF Web site (<http://naif.jpl.nasa.gov/naif/utilities.html>).

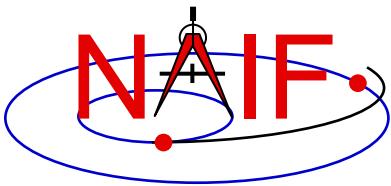
BFF	displays <u>binary</u> file format of a binary PCK file
BINGO	converts <u>binary</u> PCK files between big-endian and little-endian formats



# Additional Information on PCK

Navigation and Ancillary Information Facility

- **For more information about PCKs, look at the following:**
  - Most Useful Routines document
  - PCK Required Reading document
  - Headers of the routines mentioned
  - Lunar/Earth High-Precision PCK/FK tutorial
  - BRIEF and FRMDIFF User's Guides
- **Related documents:**
  - Frames Required Reading
  - Kernel Required Reading
  - NAIF\_IDS Required Reading
  - Time Required Reading



---

Navigation and Ancillary Information Facility

# “Camera-matrix” Kernel CK

(Orientation or Attitude Kernel)

Focused on reading CK files

January 2020



# CK File Contents - 1

---

Navigation and Ancillary Information Facility

- **A CK file holds orientation data for a spacecraft or a moving structure on the spacecraft**
  - “Orientation data”  $\Rightarrow$  quaternions, from which orientation matrices are formed by SPICE software. These matrices are used to rotate position vectors from a base reference frame (the “from” frame) into a second reference frame (the “to” frame)
    - » In SPICE this is often called the “C-matrix or “Camera matrix”
  - Optionally may include angular velocity of the “to” frame with respect to the “from” frame
    - » Angular velocity vectors are expressed relative to the “from” frame.
- **A CK file should also contain comments—sometimes called metadata—that provide some details about the CK such as:**
  - the purpose for this particular CK
  - when and how it was made
  - what time span(s) the data cover

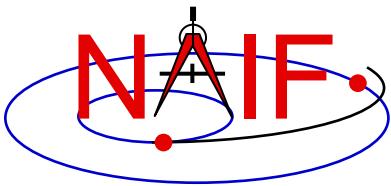


# CK File Contents - 2

---

Navigation and Ancillary Information Facility

- **A single CK file can hold orientation data for one, or for any combination of spacecraft or spacecraft moving structures**
  - Some examples
    1. Huygens Probe
    2. Cassini Orbiter and its CDA instrument mirror
    3. Mars Express Orbiter, PFS scanner, Beagle Lander
    4. MRO orbiter, MRO high gain antenna, MRO solar arrays
- **But in most cases CKs contain data for just one structure**

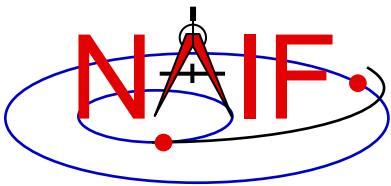


# C-Kernel Varieties - 1

---

Navigation and Ancillary Information Facility

- “Reconstruction” CK (also called “definitive” CK)
  - A CK file can be made from downlinked orientation telemetry returned from a spacecraft or other structure
  - A CK might also be made from some process that improves upon the pointing determined from downlinked telemetry (“C-smithing”)
  - Most often used for science data analysis or spacecraft performance analysis
- “Predict” CK
  - A CK file can be made using information that predicts what the orientation will be some time in the future
    - » Input data usually come from a modeling program, or a set of orientation rules
  - Most often used for science observation planning, quick-look science data analysis, engineering assessments and software testing
    - » If of known high quality, it might be used to substitute for any data gaps in reconstruction CKs
    - » In some cases (ESA in particular) the predict meets the reconstruction accuracy requirements; thus a true reconstruction CK is not generally produced

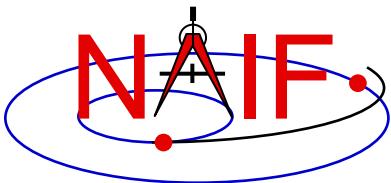


# C-Kernel Varieties - 2

---

Navigation and Ancillary Information Facility

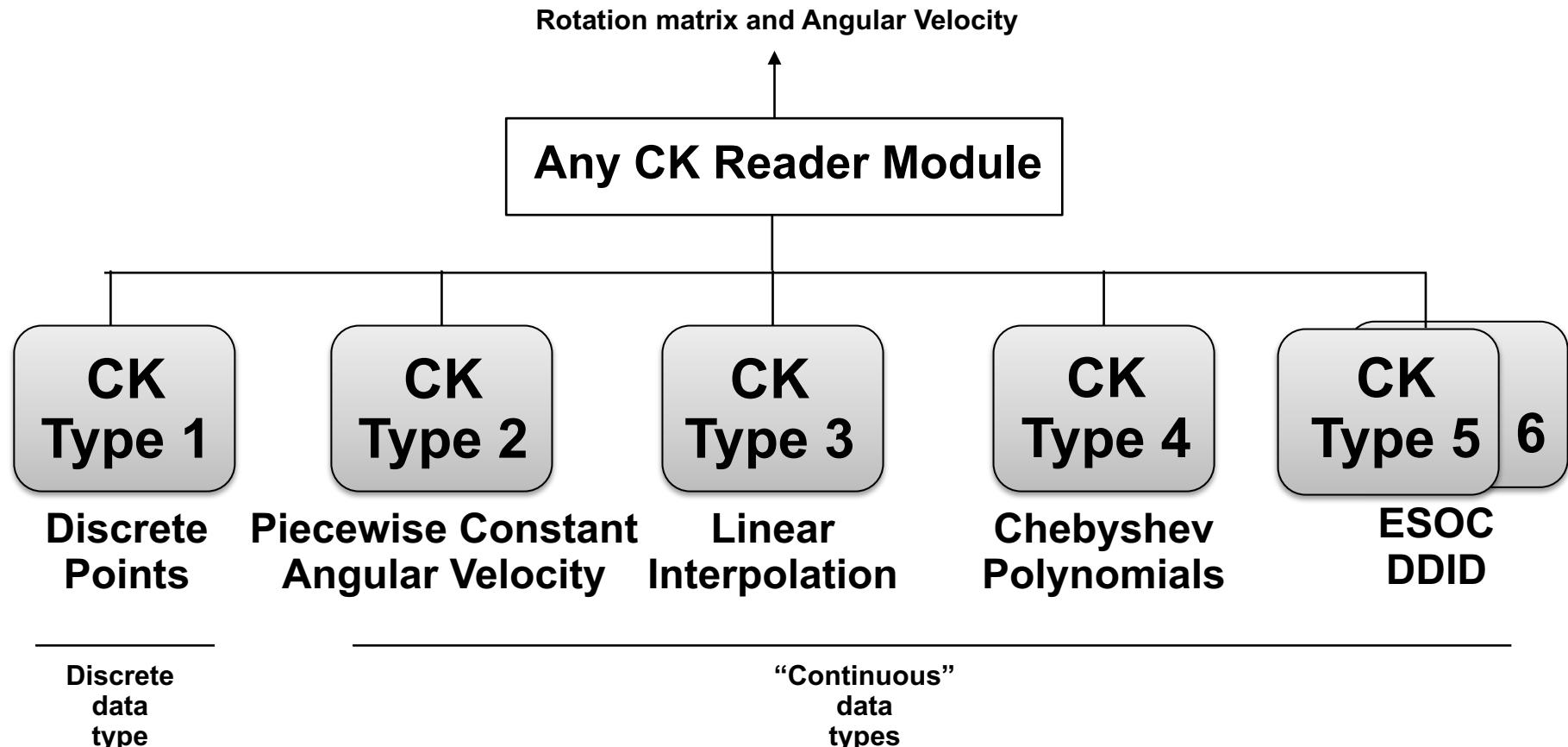
- **Knowledge of CK variety**—“reconstruction” or “predict”—might be implicit in the file naming schema, and/or might be provided in the comment section, but is not available using a SPICE API
- **It is inadvisable that both “reconstruction” and “predict” data be combined in a single file**



# CK Data Types

Navigation and Ancillary Information Facility

The underlying orientation data are of varying types, but the user interface to each of these CK types is the same.



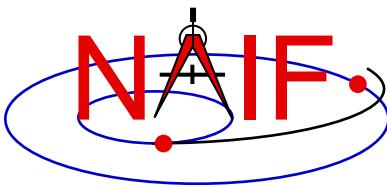


# Kernel Data needed

---

Navigation and Ancillary Information Facility

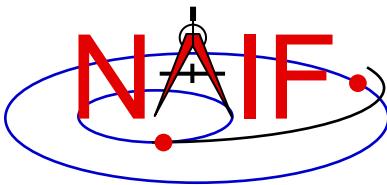
- To obtain orientation one needs at least three SPICE kernel types: **CK, SCLK, and LSK**.
  - CK contains spacecraft or other structure orientation.
  - SCLK and LSK contain time correlation coefficients used to convert between encoded spacecraft clock time (SCLK) and ephemeris time (ET).
    - » Sometimes an LSK is not needed in this conversion, but it's best to have it available as it is usually needed for other purposes.
- One may also need an **FK** if planning to access CK data via high level SPICE interfaces.
  - FK associates reference frames with CK data via CK IDs.



# What SPICE Routines Access CKs?

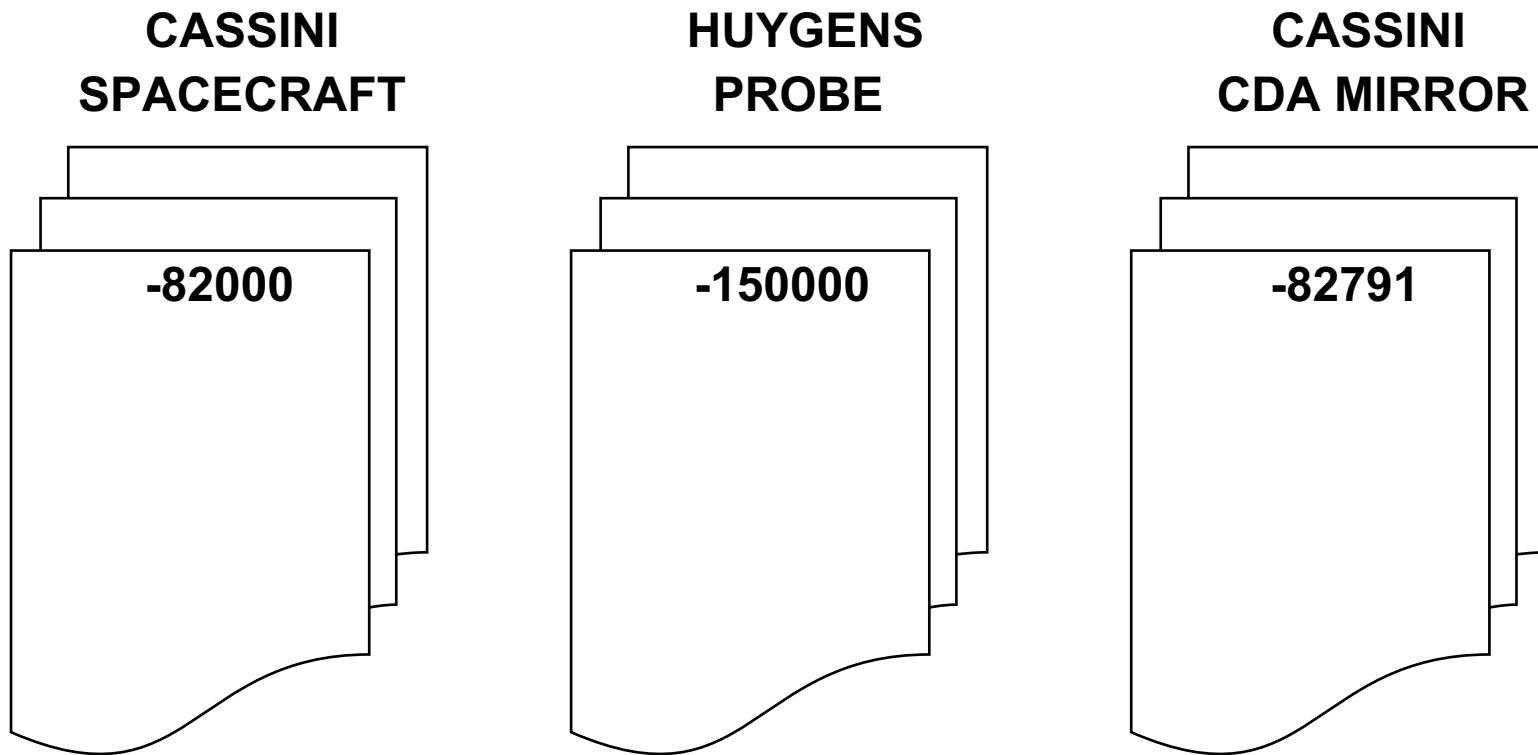
Navigation and Ancillary Information Facility

- **High-level SPICELIB routines are used more often than the “original” CK readers to access CK data. These high-level routines are:**
  - Position or state transformation matrix determination
    - » **PXFORM, PXFRM2:** return a rotation matrix (3x3) from one frame to another, either of which can be a CK-based frame or have CK-based frames as “links” in its chain
    - » **SXFORM:** return a state transformation matrix (6x6) from one frame to another, either of which can be a CK-based frame or have CK-based frames as “links” in its chain
  - Position or state vector determination
    - » **SPKPOS:** return a position vector (3x1) in a specified frame, which can be a CK-based frame or have CK-based frames as “links” in its chain
    - » **SPKEZR:** return a state vector (6x1) in a specified frame, which can be a CK-based frame or have CK-based frames as “links” in its chain
- **Use of the above mentioned routines is discussed in the FK, Using Frames, and SPK tutorials**
- **The “original” CK access routines are CKGP and CKGPAV**
  - Use of these routines is described in the BACKUP section of this tutorial

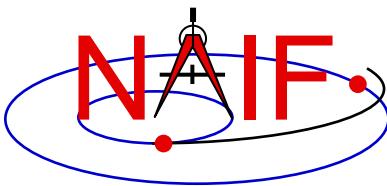


# An Example of Project CK Files

Navigation and Ancillary Information Facility



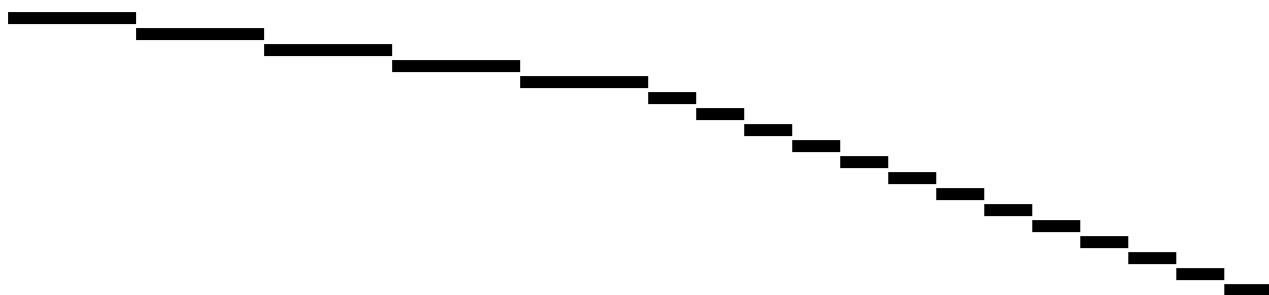
A user's program must be able to load as many of these files as needed to satisfy his/her requirements. It is strongly recommended that users' programs have the flexibility to load a list of CK files provided to the program at run time; this is easily accomplished using the Toolkit's FURNISH routine.



# Sample\* CK File Coverage - 1

Navigation and Ancillary Information Facility

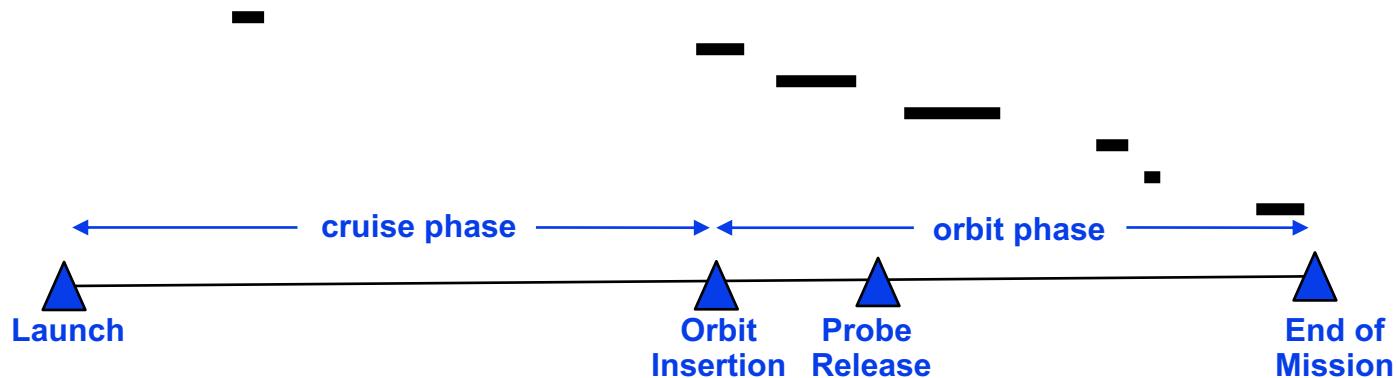
Cassini  
orbiter CK:



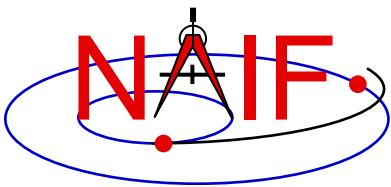
Huygens  
probe CK:



Cassini  
instrument  
mirror CK:



\* Note: This is not an actual Cassini/Huygens scenario; it is a highly simplified illustration of some of the possibilities for orientation delivery on a planetary mission.

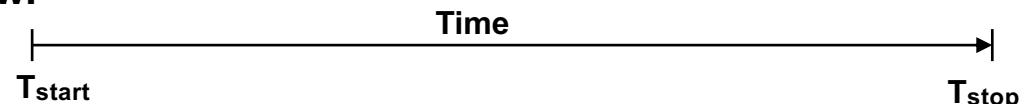


# Sample CK Data Coverage - 2

Navigation and Ancillary Information Facility

Even though a project's CK production process may suggest that CK files provide continuous coverage for the interval of time for which they were generated, in reality this is rarely the case. CK files very often contain gaps in coverage! An example of this is depicted below.

Coverage of ...



a CK file:  
as appears in file name/comments



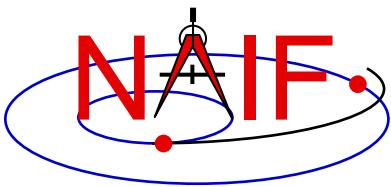
CK file segments:  
as appears in ckbrief/spacit summary



CK segments having data gaps:  
(CK Types 2 - 6)



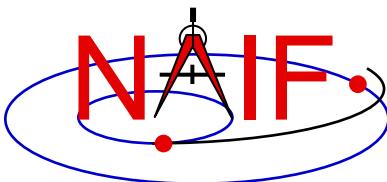
The blue line segments represent interpolation intervals – times when pointing will be returned and the FOUND flag returns as “TRUE.”



# What is an Interpolation interval?

Navigation and Ancillary Information Facility

- An interpolation interval is a time period for which routines that access CKs can compute and return pointing.
  - For CK Types 3, 5 and 6 the pointing is computed by interpolating between the attitude data points that fall within the interval.
  - For CK Type 2 the pointing within each interval is computed by extrapolating from a single attitude and associated angular velocity.
  - For CK Type 4 the pointing is computed by evaluating polynomials covering the interval.
  - For CK Type 1 (discrete pointing instances) the notion of an interpolation interval is not relevant.
- The time periods between interpolation intervals are gaps during which CK access routines are not able to compute pointing.
- The interpolation intervals in Type 3 CK segments can be modified without changing the actual pointing data.
  - The CKSPANIT and CKSMRG programs are used to make these changes.



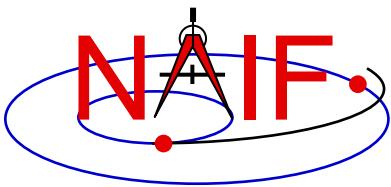
# How are Requests Falling in Gaps Handled?

---

Navigation and Ancillary Information Facility

- When using high-level SPICE routines\* in a fashion that requires CK data, if the time of your requested computation falls within a CK gap (thus, outside of a CK interpolation interval), the routine will signal an error.
  - Usually this is a frame-related error
- If using a low-level CK reader, CKGP or CKGPAV, valid pointing data will not be returned and the reader's "found flag" will be set to false.
  - Be sure to check the "found flag!"

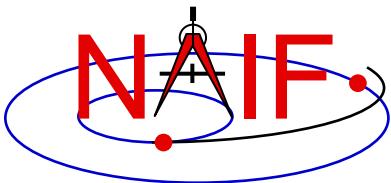
\* For example: SPKEZR, SPKPOS, SXFORM, PXFORM, SINCPT



# Obtaining Coverage of a CK File

Navigation and Ancillary Information Facility

- **High-level subroutine interfaces allow for obtaining CK coverage information.**
  - Call CKOBJ to find the set of structures for which a specified CK provides data.
    - » INPUT: a CK file name and initialized SPICE integer “Set” data structure. The set may optionally contain ID codes obtained from previous calls.
    - » OUTPUT: the input set, to which have been added (via set union) the ID codes of structures for which the specified CK provides coverage.
  - CALL CKOBJ ( CK, IDS )
  - Call CKCOV to find the window of times for which a specified CK file provides coverage for a specified structure:
    - » INPUT: a CK file name, structure ID code, flag indicating whether angular rates are needed, flag indicating whether coverage on segment or interval level is to be returned, tolerance, output time system, and initialized SPICE double precision “Window” data structure. The window may optionally contain coverage data from previous calls.
    - » OUTPUT: the input window, to which have been added (via window union) the sequence of start and stop times of coverage intervals of the specified CK, expressed as encoded SCLK or ET seconds past J2000.
  - CALL CKCOV ( CK, ID, NEEDAV, LEVEL, TOL, TIMSYS, COVER)
- See the headers of these routines for example programs.
- Also see the CELLS, SETS and WINDOWS Required Reading for background information on these SPICE data types.



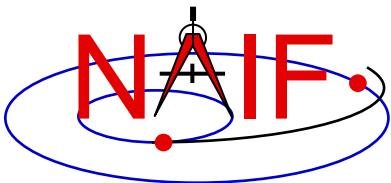
# Using CKCOV

---

Navigation and Ancillary Information Facility

- When using high-level routines\* that need orientation data from a C-kernel, it's often a good idea to first determine what are the valid interpolation intervals in your CK using CKCOV.
  - If using multiple CKs, all of which are needed to construct a frame chain, call CKCOV for each one and then intersect the coverage windows. (wnintd is the SPICE intersection API.)
- Then check each time of interest for your geometry calculations against the window of valid intervals before proceeding onwards.

\* For example: SPKEZR, SPKPOS, SXFORM, PXFORM, SINCPT

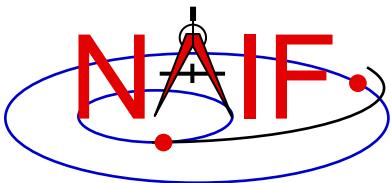


# Determine Coverage Using Utilities

---

Navigation and Ancillary Information Facility

- Three Toolkit utility programs can provide various kinds of CK summaries, including listings of gaps or of interpolation intervals
  - CKBRIEF
  - FRMDIFF
  - SPACIT



# Summarizing a CK File - 1

Navigation and Ancillary Information Facility

- A summary of a CK can be made using CKBRIEF
  - At your command prompt, type the program name followed by the names of CK, LSK and SCLK files (given in any order)

```
% ckbrie f 07102_07107ra.bc naif0008.tls cas00106.tsc
```

CKBRIEF Ver 3.2.0, 2006-11-02. SPICE Toolkit Version: N0061.

Summary for: 07102\_07107ra.bc

Object: -82000

Interval Begin ET

Interval End ET

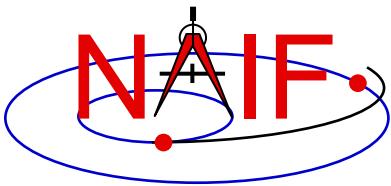
AV

-----

2007-APR-12 00:01:06.462

-----

2007-APR-17 00:01:03.726 Y



# Summarizing a CK File - 2

---

Navigation and Ancillary Information Facility

- A summary of interpolation intervals within a CK can be made with CKBRIEF, using its '-dump' option

```
% ckbrieft -dump 07102_07107ra.bc naif0008.tls cas00106.tsc
```

CKBRIEF Ver 3.2.0, 2006-11-02. SPICE Toolkit Version: N0061.

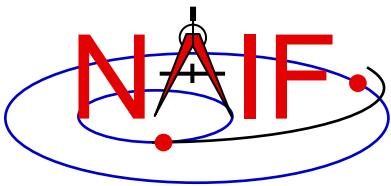
Summary for: 07102\_07107ra.bc

Segment No.: 1

Object: -82000

Interval Begin ET	Interval End ET	AV
-----	-----	-----
2007-APR-12 00:01:06.462	2007-APR-12 05:58:02.576	Y
2007-APR-12 05:58:22.576	2007-APR-12 21:34:26.221	Y
. . .		

continued on next page



# Summarizing a CK File - 3

Navigation and Ancillary Information Facility

continued from previous page

- A summary of interpolation intervals within a CK can also be made using FRMDIFF, using its '-t dumpc' option
- A summary of gaps between interpolation intervals in a CK can be made using FRMDIFF, using its '-t dumpg' option

```
% frmdiff -t dumpg \
           -k cas_v40.tf naif0008.tls cas00106.tsc \
           -f 'YYYY-DOYTHR:MN:SC ::RND' \
           07102_07107ra.bc
#
# . . . <FRMDIFF report header> . . .
#
# gap_start, gap_stop, gap_duration_sec, gap_duration_string
2007-102T05:56:57 2007-102T05:57:17 19.999 0:00:00:19.999
2007-102T21:33:21 2007-102T21:33:41 19.999 0:00:00:19.999
2007-102T21:34:57 2007-102T21:35:25 27.999 0:00:00:27.999
. . .
```



# Summarizing a CK file - 4

---

Navigation and Ancillary Information Facility

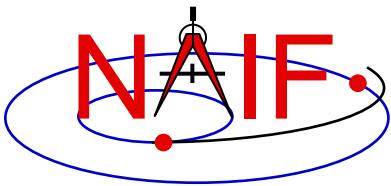
- A detailed summary of a CK can be made using SPACIT. See the SPACIT User's Guide for details.

---

```
Segment ID      : CASSINI ATT: -Y TO TITAN, +Z - VELCTY
Instrument Code: -82000
Spacecraft      : Body -82, CAS
Reference Frame: Frame 1, J2000
CK Data Type   : Type 3
Description     : Continuous Pointing: Linear Interpolation
Available Data  : Pointing and Angular Velocity
UTC Start Time : 2005 FEB 15 07:59:59.999
UTC Stop Time  : 2005 FEB 15 08:59:59.998
SCLK Start Time: 1/1487147147.203
SCLK Stop Time : 1/1487150747.209
```

---

. .  
etc. etc.



# CK Utility Programs

Navigation and Ancillary Information Facility

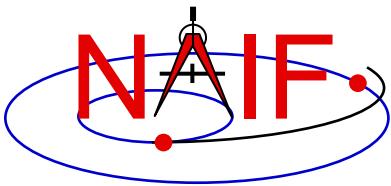
- **The following CK utility programs are included in the Toolkit:**

<b>CKBRIEF</b>	summarizes coverage for one or more CK files
<b>SPACIT</b>	generates segment-by-segment summary of a CK file
<b>COMMNT</b>	reads, appends, or deletes comments in an CK file
<b>MSOPCK</b>	converts attitude data provided in a text file into a CK file
<b>FRMDIFF</b>	samples or compares orientation of CK-based frames

- **These additional CK utility programs are provided on the NAIF Web site**

<b>CKSLICER</b>	subsets a CK file
<b>CKSMRG</b>	merges segments in a type 3 CK file (*)
<b>DAFCAT</b>	concatenates together CK files (*)
<b>CKSPANIT</b>	modifies interpolation interval information in a Type 3 CK file
<b>DAFMOD</b>	alters structure or frame IDs in a CK file
<b>PREDICKT</b>	creates a CK file representing an orientation profile described by a set of orientation rules and a schedule
<b>BFF</b>	displays binary file format of a CK file
<b>BINGO</b>	converts CK files between IEEE and PC binary formats

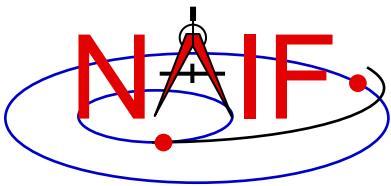
(\*) DAFCAT and CKSMRG are frequently used together to first merge many CK files into a single file using DAFCAT and then merge segments inside the merged file using CKSMRG.



# Additional Information on CK

Navigation and Ancillary Information Facility

- For more information about CK, look at the following documents
  - CK Required Reading
  - headers for the CKGP and CKGPAV routines
  - Most Useful SPICELIB Routines
  - CKBRIEF and FRMDIFF User's Guides
  - Frames tutorials: FK and Using Frames ← *don't miss these*
  - Porting\_kernels tutorial
- Related documents
  - SCLK Required Reading
  - Time Required Reading
  - Frames Required Reading
  - NAIF\_IDS Required Reading
  - Rotations Required Reading

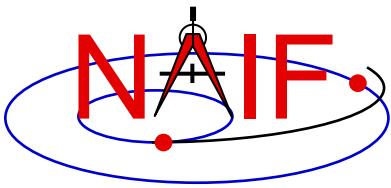


# Backup

---

Navigation and Ancillary Information Facility

- **The meaning of Tolerance**
- **Examples of Problems Encountered When Using the CK Subsystem**
- **Using the CK “reader” APIs**

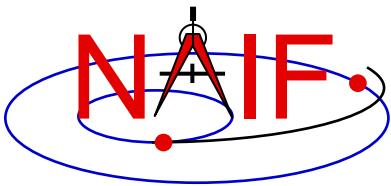


# The Meaning of Tolerance - 1

Navigation and Ancillary Information Facility

- The low level CKGP and CKGPAV routines use a time tolerance, “tol,” measured in ticks, in executing pointing lookups.
  - No matter whether your CK is a discrete type (Type 1) or a continuous type (Types 2 - 6), if pointing information is not found within +/- tol of your pointing request time, no pointing will be returned and the “found flag” will return as “FALSE.”
  - For Type 1 (discrete) CKs, the pointing instance **nearest\*** to your request time will be returned, as long as it is within tol of your request time.
    - » If the nearest pointing instances on each side of your request time are equidistant from your request time, the instance with the later time tag will be selected.
  - For Types 2 - 6 (continuous pointing), pointing for **exactly** your request time will be returned if this time falls anywhere within an interpolation interval.
  - For all Types, the time tag associated with the pointing data will also be returned.
- See the next three charts for graphic depictions.

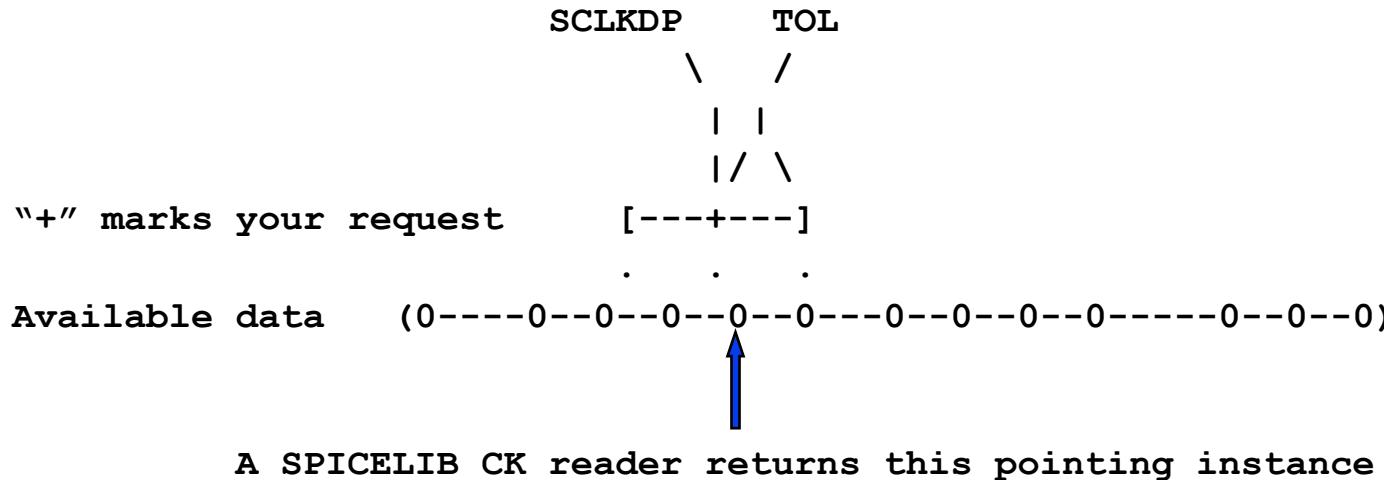
\*Ignoring segment priority



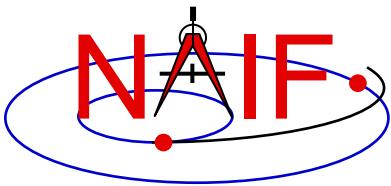
# The Meaning of Tolerance - 2

Navigation and Ancillary Information Facility

When reading a Type 1 CK containing **discrete** pointing instances



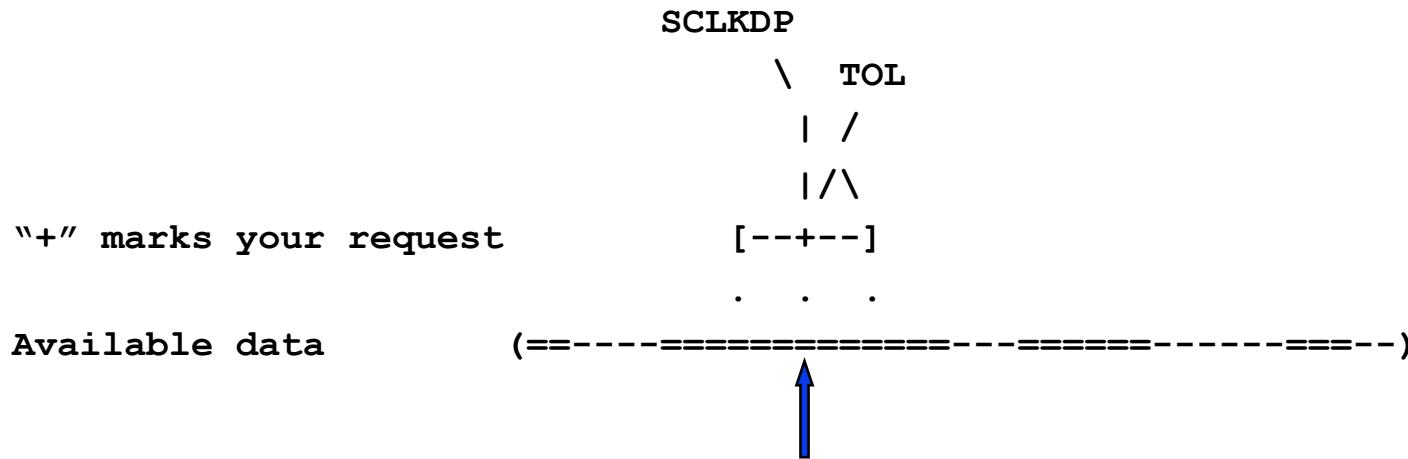
- “0” is used to represent discrete pointing instances (quaternions)
- “( )” are used to represent the end points of a segment within a CK file
- SCLKDP is the time at which you have requested pointing
- TOL is the time tolerance you have specified in your pointing request
- The quaternions occurring in the segment need not be evenly spaced in time



# The Meaning of Tolerance - 3

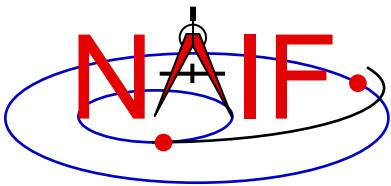
Navigation and Ancillary Information Facility

When reading a Type 2, 3, 4, 5 or 6 CK (**continuous pointing**), with a “pointing request” that falls within a span of continuous pointing (an “interpolation interval”)



A SPICELIB CK reader returns pointing at precisely the requested epoch

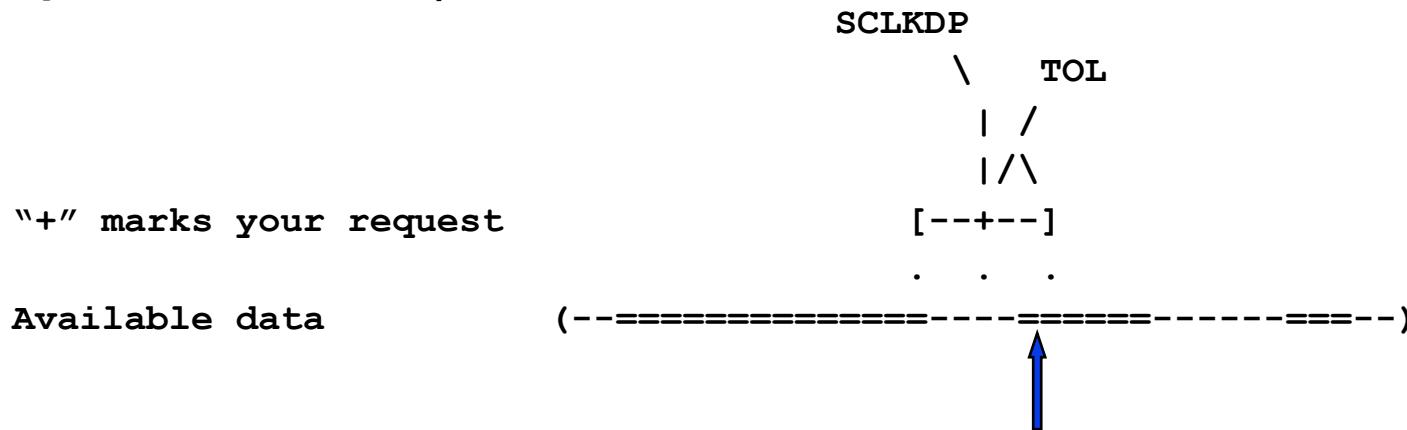
- “==” is used to indicate interpolation intervals of continuous pointing
- “( )” are used to represent the end points of a segment within a CK file
- SCLKDP is the time at which you have requested pointing
- TOL is the time tolerance you have specified in your pointing request; for this particular case it does not come into play
- The quaternions occurring in the periods of continuous pointing need not be evenly spaced in time



# The Meaning of Tolerance - 4

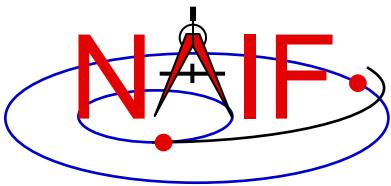
Navigation and Ancillary Information Facility

When reading a Type 2, 3, 4, 5 or 6 CK (continuous pointing), with a “pointing request” that is NOT within a span of continuous pointing (an “interpolation interval”)



A SPICELIB CK reader returns pointing at the epoch closest to the request time, if this is within TOL of that request time.

- “==” is used to indicate interpolation intervals of continuous pointing
- “( )” are used to represent the end points of a segment within a CK file
- SCLKDP is the time at which you have requested pointing
- TOL is the time tolerance you have specified in your pointing request
- The quaternions occurring in the periods of continuous pointing need not be evenly spaced in time

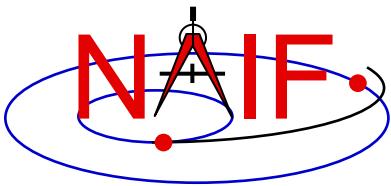


# Problems using CK - 1

---

Navigation and Ancillary Information Facility

- **The file or files you loaded do not contain orientation data for the object of interest.**
  - Make sure the ID that you use in a call to CKGP or CKGPAV matches one in the CK file(s) you have loaded.
  - Make sure the frame that you specify in a call to SXFORM, PXFORM, SPKEZR, or SPKPOS is transformable to one available in the loaded CK files.
- **One of the low-level routines, CKGP or CKGPAV, returns a transformation matrix and/or angular velocity that does not appear correct.**
  - Probably the FOUND flag is “FALSE” and you are using data left over from a previous query. Remember to **always check the FOUND flag!** (If the FOUND flag is “TRUE” but the data seem bad, contact the data producer.)

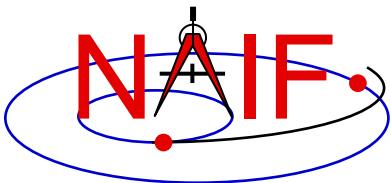


# Problems using CK - 2

---

Navigation and Ancillary Information Facility

- **The file, or files, you loaded do not cover the time at which you requested orientation**
  - Check file coverage on the segment level by summarizing the file(s) using CKBRIEF or SPACIT
  - Check interpolation interval coverage using CKBRIEF with option “-dump,” or by examining comments provided in the comment area of the file - you may be asking for data within a coverage gap (your request time is outside of interpolation intervals)
- **One of the high-level routines (SPKEZR, SPKPOS, SXFORM, PXFORM, SINCPT) signals an error**
  - These routines read CK files using tolerance = 0
    - » For discrete CKs (Type 1) the orientation of a CK-based frame will be computed only if the time provided to a Frames routine exactly matches one of the times stored in the CK file; otherwise an error will be signaled.
    - » For continuous CKs (all but Type 1) the orientation of a CK-based frame will be computed only if the time provided to a frame routine falls within one of the interpolation intervals defined by the CK file; otherwise an error will be signaled.

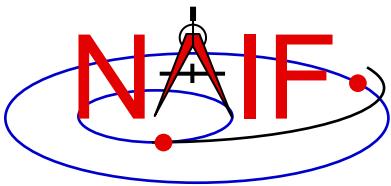


# Problems using CK - 3

---

Navigation and Ancillary Information Facility

- You've confirmed not having any of the previously described problems, but the **FOUND** flag returns as "FALSE" when using **CKGPAV**, or **SXFORM** or **SPKEZR** signals a frame related error.
  - You are using a SPICE routine that requires angular velocity as well as orientation, but the CK segments available at your requested epoch don't contain angular velocity.
    - » Routines requiring AV are: **CKGPAV**, **SXFORM**, **SPKEZR**
    - » Routines not requiring AV are: **CKGP**, **PXFORM**, **SPKPOS**

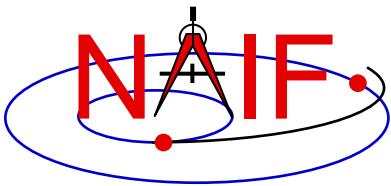


# Problems using CK - 4

---

Navigation and Ancillary Information Facility

- The FOUND flag returns as “TRUE” when using CKGPAV but returned angular velocity does not appear correct.
  - While many sources of angular rate data, for example spacecraft telemetry, specify it relative to the spacecraft frame, SPICE CK files store it, and CKGPAV returns it, with respect to the base frame. So the sense of returned angular velocity in a CK may be unexpected.
- The FOUND flag returns as “TRUE” when using CKGP/CKPGAV but the quaternion computed from the returned transformation matrix via a call to M2Q does not appear correct.
  - The quaternion returned by M2Q follows the SPICE style, which is different from the quaternion styles used by some other sources of orientation data, for example most spacecraft telemetry.
    - » See the headers of the M2Q and Q2M routines, and the Rotations Required Reading technical reference document for more details.
    - » NAIF prepared a “white paper” explaining differences between various quaternion styles commonly used in space applications:
      - [https://naif.jpl.nasa.gov/pub/naif/misc/Quaternion\\_White\\_Paper/Quaternions\\_White\\_Paper.pdf](https://naif.jpl.nasa.gov/pub/naif/misc/Quaternion_White_Paper/Quaternions_White_Paper.pdf)

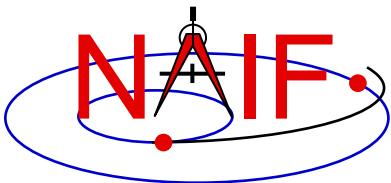


# Problems using CK - 5

---

Navigation and Ancillary Information Facility

- You're trying to use a CK file that is not properly formatted for your computer
  - You can read only a binary CK file with the CK subroutines; you can't read a “transfer format” file
    - » Although not required, binary CK files often have a name like “xxxxxx.bc”
    - » Although not required, transfer format CK files often have a name like “xxxxxx.xc”
  - If using Toolkit Version N0051 or earlier you must have the proper kind of CK binary file for your computer (a native binary file)
    - » PC (Windows or Linux) and Mac OSX use the IEEE Little-endian binary standard
    - » Sun and very old Mac (Motorola cpu) use the IEEE Big-endian binary standard
    - » The pair of utility programs named TOBIN and TOXFR, or the utility program SPACIT, can be used to port CK files between computers having incompatible binary standards



# One Example of How To Read a CK File

Navigation and Ancillary Information Facility

Initialization ... typically once per program run

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

```
CALL FURNSH( 'lsk_file_name' )
CALL FURNSH( 'sclk_file_name' )
CALL FURNSH( 'ck_file_name' )
```

}

Better yet, use a “furnsh kernel”  
to load all the needed files.

Loop ... do as often as you need

**Convert UTC to SCLK ticks \***

```
CALL STR2ET( 'utc_string', tdb )
CALL SCE2C ( spacecraft_id, tdb, sclkdp )
```

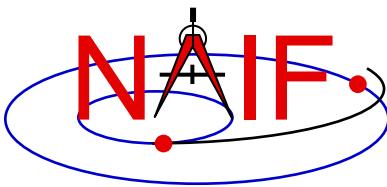
**Get rotation matrix, or rotation matrix and angular velocity at requested time**

or CALL CKGP (instid,sclkdp,tol,'ref\_frame',cmat, clkout,found)  
CALL CKGPAV (instid,sclkdp,tol,'ref\_frame',cmat,av,clkout,found)

inputs

outputs

\* Although most spacecraft have a single on-board clock and this clock has the same ID as the spacecraft, the user should know which SCLK was used to tag data in a CK file in order to specify the correct ID in a call to SCE2C.



# Arguments of CKGPAV

Navigation and Ancillary Information Facility

## Inputs

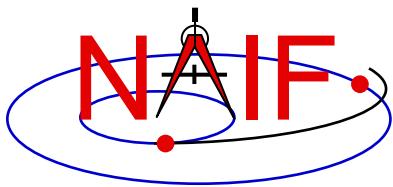
instid	NAIF ID for the spacecraft, instrument or other structure for which the orientation is to be returned
sclkdp	the time at which the orientation matrix and angular velocity are to be computed. The time system used is encoded spacecraft clock time (SCLK). The units are ticks since the zero epoch of the clock
tol*	the tolerance, expressed as number of SCLK ticks, to be used in searching for and computing the orientation data
ref_frame	the name of the reference frame with respect to which the orientation is to be computed. This is also called the “base” or “from” frame.

## Outputs

cmat	the 3x3 rotation matrix that you requested
av	the angular velocity that you requested
clkout	the exact time for which the orientation and angular velocity was computed
found	the logical flag indicating whether the orientation and angular velocity data were found. Note that if the loaded CK file(s) do not contain angular velocity data, CKGPAV will return a FALSE found flag even if orientation could have been computed. If “found” is .FALSE., then the values of the output arguments “cmat” and “av” are undefined and <b>should not be used!</b>

**Always check the FOUND flag returned by CKGPAV and CKGP!**

\* tol is explained in detail earlier in these backup slides

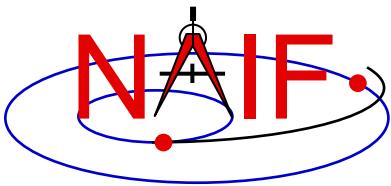


---

Navigation and Ancillary Information Facility

# Frames Kernel FK

January 2020

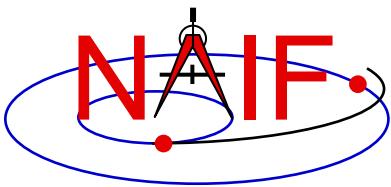


# Background

---

Navigation and Ancillary Information Facility

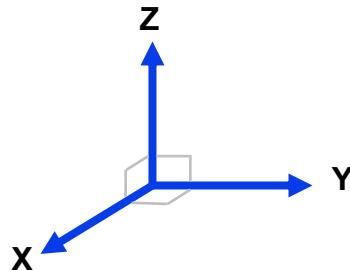
- **First, reminders of what SPICE means by:**
  - reference frame
  - coordinate system



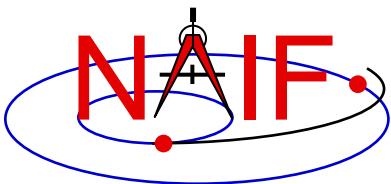
# What is a Reference Frame?

Navigation and Ancillary Information Facility

- Within SPICE, a reference frame is specified by three orthogonal axes



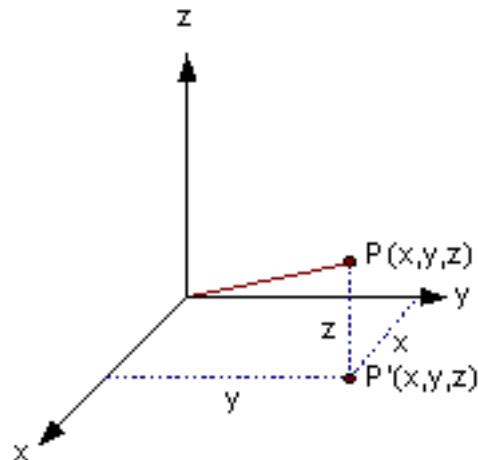
- It may be fixed in space (not rotating, not accelerating)
  - This is called an inertial frame
- It may be changing its orientation in space
  - This is called a non-inertial frame
- Every frame has a name and a numeric ID assigned to it
  - You'll use the name as an argument in calling Toolkit APIs
- Every frame has an associated center location



# What is a Coordinate System?

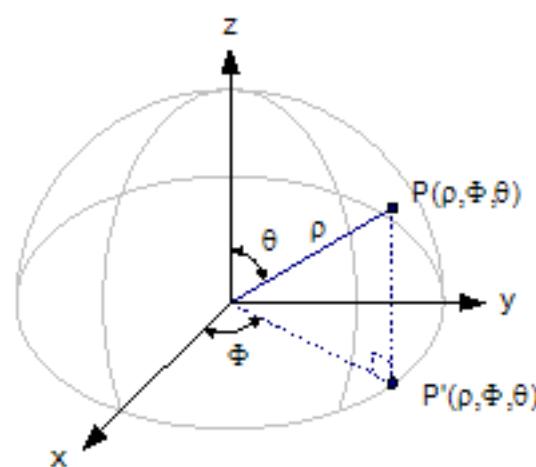
Navigation and Ancillary Information Facility

- Within SPICE, a coordinate system is the method used to specify a vector within a reference frame. Examples:



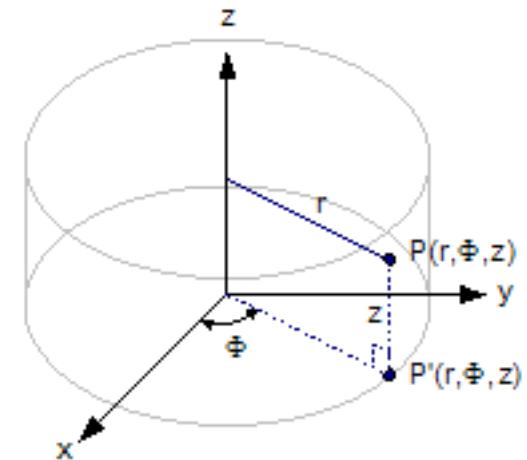
Rectangular  
coordinates

$$X, Y, Z$$



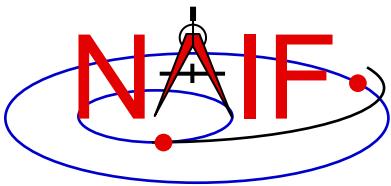
Spherical  
coordinates

$$\Phi, \theta, \rho$$



Cylindrical  
coordinates

$$\phi, Z, R$$



# Introduction

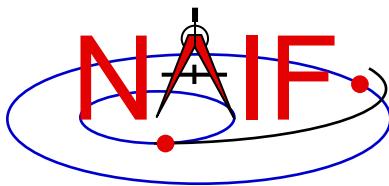
---

Navigation and Ancillary Information Facility

## What does the SPICE FRAMES subsystem do?

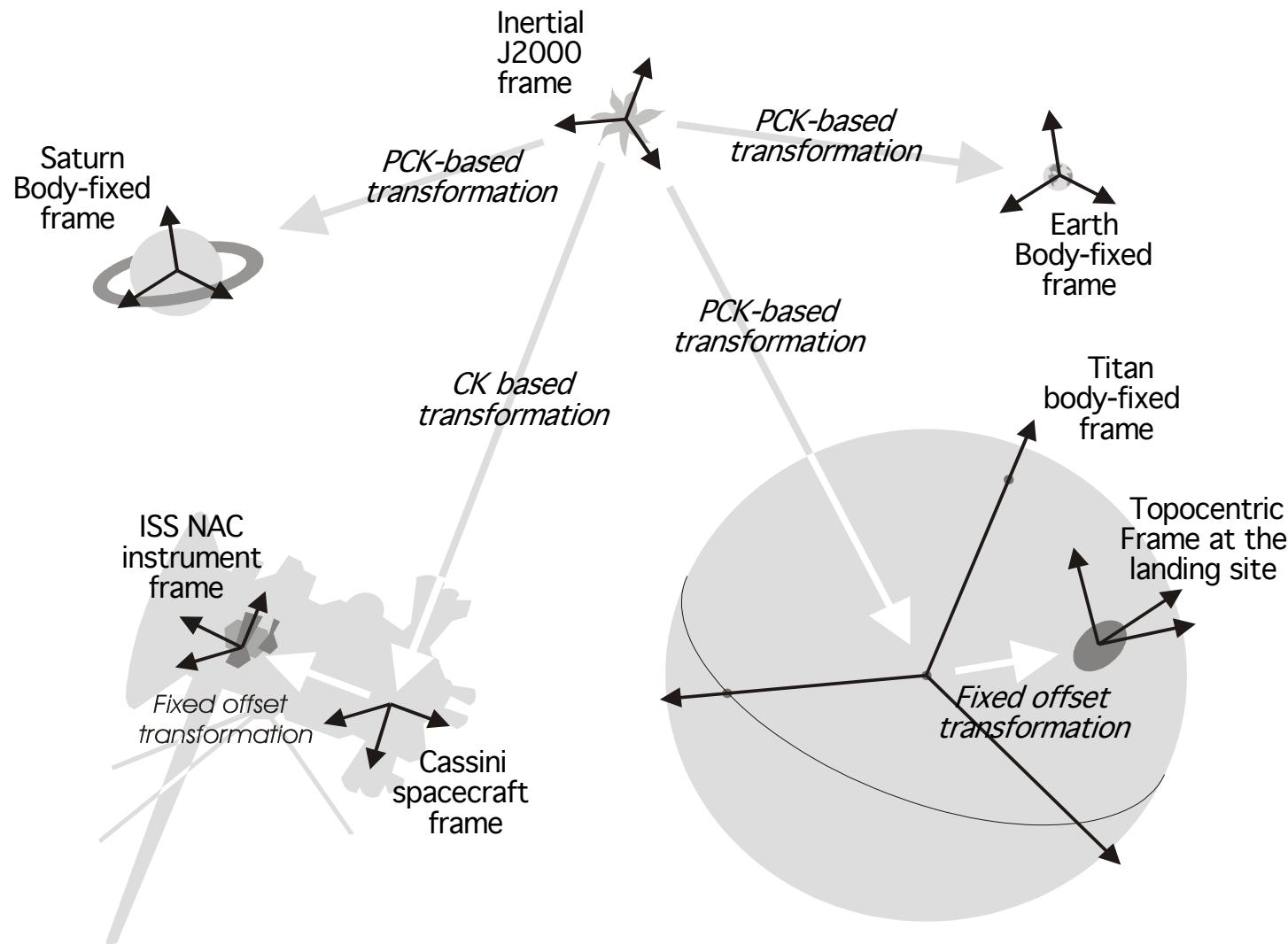
1. It establishes relationships between reference frames used in geometry computations – it "chains frames together" in a frame tree.
  
2. It connects frames with the sources of their orientation specifications.
  - In some cases those data are included in the Frames kernel itself.

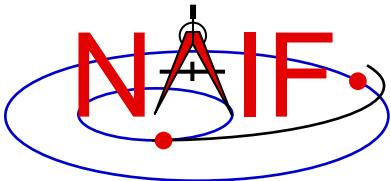
Based on these relationships and the orientation source information, the frames subsystem allows SPICE software to compute rotations between neighboring frames in the frame tree, and to combine these rotations in the right order, thus providing an ability to compute the orientation of any frame in the tree with respect to any other frame in the tree, at any time.



# Sample Frame Tree

Navigation and Ancillary Information Facility





# Frame Names

---

Navigation and Ancillary Information Facility

- Frame names are character strings used to identify frames to Toolkit APIs
- Examples of frame names:
  - J2000
  - IAU\_MARS
  - DAWN\_SPACECRAFT
  - MEX\_OMEGA
  - DSS-14\_TOPO

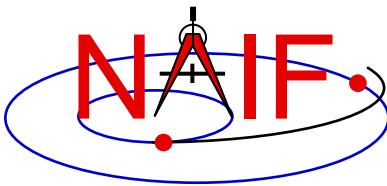


# Where to Find the Names of Frames?

---

Navigation and Ancillary Information Facility

- Refer to the “NAIF IDs” Tutorial for an introduction to reference frame names and IDs
- Refer to the FRAMES.REQ document for the list of NAIF “built in” (hard coded) inertial and body-fixed frames
- Refer to a mission’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 names and descriptions of the body-fixed frames defined for the moon



# Frame Classes and Examples

Navigation and Ancillary Information Facility

## Frame class

## Frame Examples (with real frame names)

### Inertial

- Earth Equator/Equinox of Epoch (ICRF, also called J2000 in SPICE)
- Planet Equator/Equinox of Epoch (MARSIAU, ...)
- Ecliptic of Epoch (ECLIPJ2000, ...)

### Body-fixed

- Solar system body IAU frames (IAU\_MARS, IAU\_SATURN, ...)
- High accuracy Earth frames (ITRF93, ...)
- High accuracy Moon frames (MOON\_PA, MOON\_ME)

### CK-based

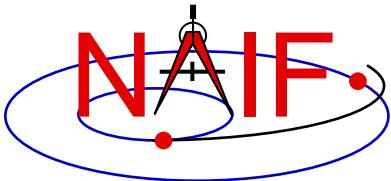
- Spacecraft (CASSINI\_SC\_BUS, ...)
- Moving parts of an instrument (MPL\_RA\_JOINT1, ...)

### Fixed Offset

- Instrument mounting alignment (CASSINI\_ISS\_NAC, ...)
- Topocentric (DSS-14\_TOPO, ...)

### Dynamic

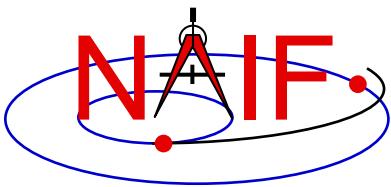
- Geomagnetic
- Geocentric Solar Equatorial
- Planet true equator and equinox of date



# Frame Class Specifications

Navigation and Ancillary Information Facility

<u>Frame class</u>	<u>Frame Defined in:</u>	<u>Orientation data provided in:</u>
Inertial	Toolkit software	Toolkit software
Body-fixed	Toolkit software or FK	PCK (text or binary style)
CK based	FK	CK
Fixed offset	FK	FK
Dynamic	FK	Toolkit software, or computed using FK, SPK, CK, and/or PCK



# Frames Kernel File Overview

Navigation and Ancillary Information Facility

- Uses the SPICE text kernel standards
- Loaded using the FURNISH routine
- Usually contains comprehensive information about the defined frames in the comment section(s) of the file
- Contains frame definition information consisting of a set of keywords in the data sections of the file. Below are examples defining a CK-based frame and a fixed-offset frame.

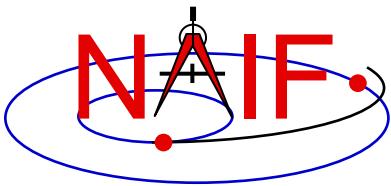
## CK-based Frame Example

```
FRAME_DAWN_SPACECRAFT = -203000
FRAME_-203000_NAME    = 'DAWN_SPACECRAFT'
FRAME_-203000_CLASS   = 3
FRAME_-203000_CLASS_ID = -203000
FRAME_-203000_CENTER  = -203
CK_-203000_SCLK       = -203
CK_-203000_SPK        = -203
```

## Fixed-offset Frame Example

```
FRAME_DAWN_FC1          = -203110
FRAME_-203110_NAME      = 'DAWN_FC1'
FRAME_-203110_CLASS     = 4
FRAME_-203110_CLASS_ID  = -203110
FRAME_-203110_CENTER    = -203
TKFRAME_-203110_RELATIVE = 'DAWN_SPACECRAFT'
TKFRAME_-203110_SPEC    = 'ANGLES'
TKFRAME_-203110_UNITS   = 'DEGREES'
TKFRAME_-203110_ANGLES  = ( 0.0, 0.0, 0.0 )
TKFRAME_-203110_AXES    = ( 1, 2, 3 )
```

- These examples are discussed in detail in the next few slides



# Frame Definition Details - 1

Navigation and Ancillary Information Facility

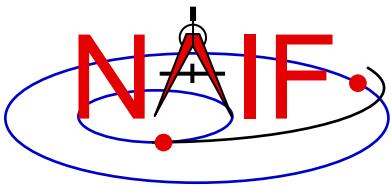
## Frame definition for the DAWN spacecraft

```
FRAME_DAWN_SPACECRAFT = -203000
FRAME_-203000_NAME    = 'DAWN_SPACECRAFT'
FRAME_-203000_CLASS   = 3
FRAME_-203000_CLASS_ID = -203000
FRAME_-203000_CENTER  = -203
CK_-203000_SCLK       = -203
CK_-203000_SPK        = -203
```

## Frame definition for the DAWN Framing Camera #1

```
FRAME_DAWN_FC1          = -203110
FRAME_-203110_NAME      = 'DAWN_FC1'
FRAME_-203110_CLASS     = 4
FRAME_-203110_CLASS_ID  = -203110
FRAME_-203110_CENTER    = -203
TKFRAME_-203110_RELATIVE = 'DAWN_SPACECRAFT'
TKFRAME_-203110_SPEC    = 'ANGLES'
TKFRAME_-203110_UNITS   = 'DEGREES'
TKFRAME_-203110_ANGLES  = ( 0.0, 0.0, 0.0 )
TKFRAME_-203110_AXES    = ( 1, 2, 3 )
```

- The Frame ID, shown in red in the two examples above, is an integer number used by the SPICE system as a “handle” in buffering and retrieving various parameters associated with a frame. In an FK it “glues together” the keywords defining the frame.



# Frame Definition Details - 2

## Navigation and Ancillary Information Facility

### Frame definition for the DAWN spacecraft

```
FRAME_DAWN_SPACECRAFT = -203000
FRAME_-203000_NAME    = 'DAWN_SPACECRAFT'
FRAME_-203000_CLASS   = 3
FRAME_-203000_CLASS_ID = -203000
FRAME_-203000_CENTER  = -203
CK_-203000_SCLK       = -203
CK_-203000_SPK        = -203
```

### Frame definition for the DAWN Framing Camera #1

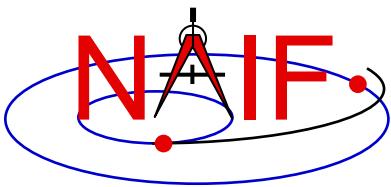
```
FRAME_DAWN_FC1          = -203110
FRAME_-203110_NAME      = 'DAWN_FC1'
FRAME_-203110_CLASS     = 4
FRAME_-203110_CLASS_ID  = -203110
FRAME_-203110_CENTER    = -203
TKFRAME_-203110_RELATIVE = 'DAWN_SPACECRAFT'
TKFRAME_-203110_SPEC    = 'ANGLES'
TKFRAME_-203110_UNITS   = 'DEGREES'
TKFRAME_-203110_ANGLES  = ( 0.0, 0.0, 0.0 )
TKFRAME_-203110_AXES    = ( 1, 2, 3 )
```

- These keywords

`FRAME_<name> = <id>`

`FRAME_<id>_NAME = <name>`

establish the association between the name and the ID of the frame.



# Frame Definition Details - 3

Navigation and Ancillary Information Facility

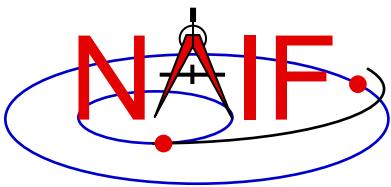
## Frame definition for the DAWN spacecraft

```
FRAME_DAWN_SPACECRAFT = -203000
FRAME_-203000_NAME    = 'DAWN_SPACECRAFT'
FRAME_-203000_CLASS = 3
FRAME_-203000_CLASS_ID = -203000
FRAME_-203000_CENTER   = -203
CK_-203000_SCLK        = -203
CK_-203000_SPK         = -203
```

## Frame definition for the DAWN Framing Camera #1

```
FRAME_DAWN_FC1          = -203110
FRAME_-203110_NAME      = 'DAWN_FC1'
FRAME_-203110_CLASS    = 4
FRAME_-203110_CLASS_ID  = -203110
FRAME_-203110_CENTER    = -203
TKFRAME_-203110_RELATIVE = 'DAWN_SPACECRAFT'
TKFRAME_-203110_SPEC     = 'ANGLES'
TKFRAME_-203110_UNITS    = 'DEGREES'
TKFRAME_-203110_ANGLES   = ( 0.0, 0.0, 0.0 )
TKFRAME_-203110_AXES     = ( 1, 2, 3 )
```

- The **FRAME...CLASS** keyword specifies the method by which the frame is related to its base frame
- This keyword is set to:
  - 1, for inertial frames
  - 2, for PCK-based frames
  - 3, for CK-based frames
  - 4, for fixed-offset frames
  - 5, for dynamic frames



# Frame Definition Details - 4

## Navigation and Ancillary Information Facility

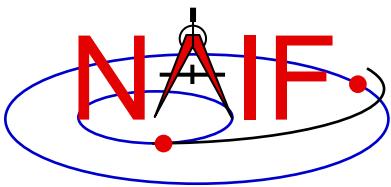
### Frame definition for the DAWN spacecraft

```
FRAME_DAWN_SPACECRAFT = -203000
FRAME_-203000_NAME    = 'DAWN_SPACECRAFT'
FRAME_-203000_CLASS   = 3
FRAME_-203000_CLASS_ID = -203000
FRAME_-203000_CENTER  = -203
CK_-203000_SCLK      = -203
CK_-203000_SPK       = -203
```

### Frame definition for the DAWN Framing Camera #1

```
FRAME_DAWN_FC1          = -203110
FRAME_-203110_NAME      = 'DAWN_FC1'
FRAME_-203110_CLASS     = 4
FRAME_-203110_CLASS_ID = -203110
FRAME_-203110_CENTER    = -203
TKFRAME_-203110_RELATIVE = 'DAWN_SPACECRAFT'
TKFRAME_-203110_SPEC    = 'ANGLES'
TKFRAME_-203110_UNITS   = 'DEGREES'
TKFRAME_-203110_ANGLES  = ( 0.0, 0.0, 0.0 )
TKFRAME_-203110_AXES    = ( 1, 2, 3 )
```

- The FRAME...CLASS\_ID is the number that connects a frame with the orientation data for it.
  - For body-fixed frames the CLASS\_ID is the ID of the natural body. It is used as input to PCK routines called by the Frame subsystem to compute orientation of the frame.
    - » The Frame ID and CLASS\_ID are not the same for the body-fixed frames defined in the Toolkit but they can be the same for frames defined in FK files.
  - For CK-based frames the CLASS\_ID is the CK structure ID. It is used as input to CK routines called by the Frame subsystem to compute orientation of the frame.
    - » Usually the CLASS\_ID of a CK-based frame is the same as the frame ID, but this is not required.
  - For fixed offset and dynamic frames the CLASS\_ID is the ID that is used to retrieve the frame definition keywords.
    - » The CLASS\_ID of a fixed offset or dynamic frame is the same as the frame ID.



# Frame Definition Details - 5

## Navigation and Ancillary Information Facility

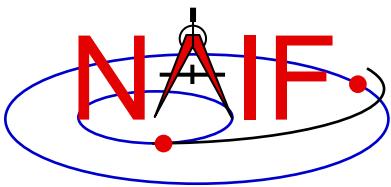
### Frame definition for the DAWN spacecraft

```
FRAME_DAWN_SPACECRAFT = -203000
FRAME_-203000_NAME    = 'DAWN_SPACECRAFT'
FRAME_-203000_CLASS   = 3
FRAME_-203000_CLASS_ID= -203000
FRAME_-203000_CENTER = -203
CK_-203000_SCLK       = -203
CK_-203000_SPK        = -203
```

### Frame definition for the DAWN Framing Camera #1

```
FRAME_DAWN_FC1          = -203110
FRAME_-203110_NAME      = 'DAWN_FC1'
FRAME_-203110_CLASS     = 4
FRAME_-203110_CLASS_ID  = -203110
FRAME_-203110_CENTER = -203
TKFRAME_-203110_RELATIVE= 'DAWN_SPACECRAFT'
TKFRAME_-203110_SPEC    = 'ANGLES'
TKFRAME_-203110_UNITS   = 'DEGREES'
TKFRAME_-203110_ANGLES  = ( 0.0, 0.0, 0.0 )
TKFRAME_-203110_AXES    = ( 1, 2, 3 )
```

- The FRAME...CENTER specifies the ephemeris object at which the frame origin is located
  - It is used ONLY to compute the light-time corrected orientation of the frame



# Frame Definition Details - 6

## Navigation and Ancillary Information Facility

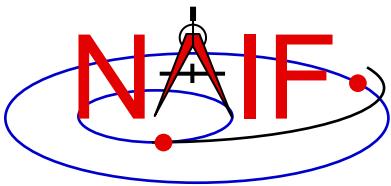
### Frame definition for the DAWN spacecraft

```
FRAME_DAWN_SPACECRAFT = -203000
FRAME_-203000_NAME    = 'DAWN_SPACECRAFT'
FRAME_-203000_CLASS   = 3
FRAME_-203000_CLASS_ID= -203000
FRAME_-203000_CENTER  = -203
CK_-203000_SCLK       = -203
CK_-203000_SPK        = -203
```

### Frame definition for the DAWN Framing Camera #1

```
FRAME_DAWN_FC1          = -203110
FRAME_-203110_NAME      = 'DAWN_FC1'
FRAME_-203110_CLASS     = 4
FRAME_-203110_CLASS_ID = -203110
FRAME_-203110_CENTER   = -203
TKFRAME_-203110_RELATIVE= 'DAWN_SPACECRAFT'
TKFRAME_-203110_SPEC    = 'ANGLES'
TKFRAME_-203110_UNITS   = 'DEGREES'
TKFRAME_-203110_ANGLES  = ( 0.0, 0.0, 0.0 )
TKFRAME_-203110_AXES    = ( 1, 2, 3 )
```

- Additional frame definition keywords may be required depending on the frame class
  - For CK frames, CK...SCLK and CK...SPK keywords identify the spacecraft clock ID and physical object ID associated with the CK structure ID
  - For fixed-offset frames, TKFRAME\_\* keywords specify the base frame and the fixed orientation with respect to this frame
  - For dynamic frames, additional keywords depend on the dynamic frame family



# APIs Using the Frames Subsystem

Navigation and Ancillary Information Facility

## SXFORM, PXFORM, and PXFRM2

return 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 )
CALL PXFRM2( 'FROM_FRAME_NAME', 'TO_FRAME_NAME', ETFROM, ETTO, MAT3X3 )
```

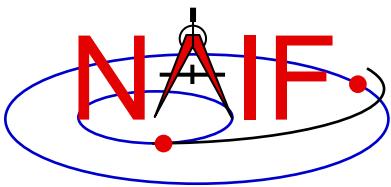
## SPKEZR and SPKPOS

return 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 APIs.

The same interfaces exist for the other supported languages (CSPICE, Icy, Mice).

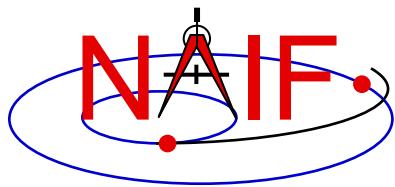


# CK-Based Frames “Must Know”

Navigation and Ancillary Information Facility

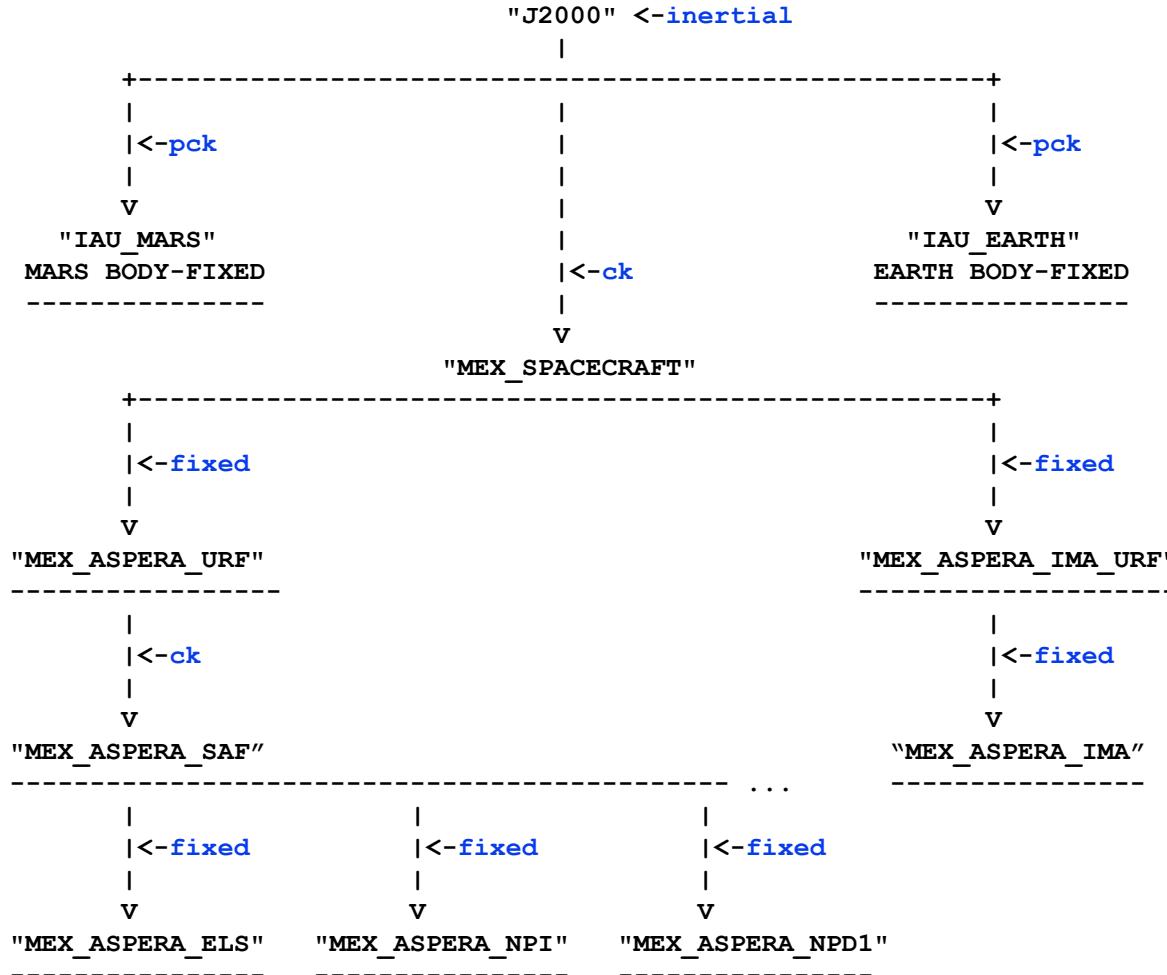
These are **VERY IMPORTANT** points you must understand!

- The frames routines (**SPKEZR**, **SPKPOS**, **SXFORM**, **PXFORM**) all read CK files using tolerance = 0
  - For **discrete** CKs (Type 1) the orientation of a CK-based frame will be computed only if the time provided to a Frames routine exactly matches one of the times stored in the CK file; otherwise an error will be signaled.
  - For **continuous** CKs (Types 2 – 6) the orientation of a CK-based frame will be computed only if the time provided to a Frames routine falls within one of the interpolation intervals defined by the CK file; otherwise an error will be signaled.
- Using **SPKEZR** or **SXFORM** requires CKs that contain angular rate data
  - Since these routines return a state vector (6x1) or state transformation matrix (6x6), angular rate must be present in the CK in order to compute vectors and matrices; if angular rate is not present an error will be signaled.
  - **SPKPOS** and **PXFORM**, which return a position vector (3x1) and a position transformation matrix (3x3) respectively, can be used when angular rate data are NOT present in a CK.
- Ephemeris time input to Frames routines is converted to SCLK to access CKs
  - SCLK and LSK kernels must be loaded to support this conversion.
  - The SCLK ID is specified in one of the CK frame definition keywords; if not, it's assumed to be the Frame ID divided by 1000.



# Frame Tree Example: ASPERA Instrument on Mars Express

## Navigation and Ancillary Information Facility



Blue text indicates frame class



# FK Utility Programs

---

Navigation and Ancillary Information Facility

- **The following FK and frames utility programs are included in the Toolkit**

**FRMDIFF** samples orientation of a frame or compares orientation of two frames

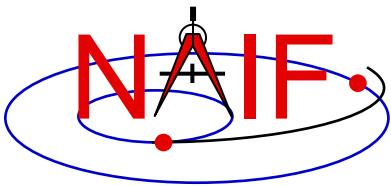
**CKBRIEF** summarizes coverage for one or more CK files

**BRIEF** summarizes coverage for one or more binary PCK files

- **These additional FK and frames utility programs are provided on the NAIF Web site**

**PINPOINT** creates SPK and topocentric frames FK files for fixed locations (ground stations, etc)

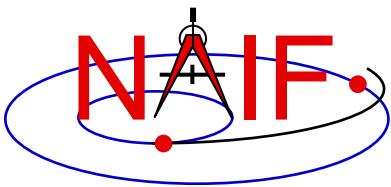
**BINGO** converts FK files between UNIX and DOS text formats



# Additional Information on FK

Navigation and Ancillary Information Facility

- For more information about FK and frames, look at the following documents
  - Frames Required Reading
  - Using Frames Tutorial
  - Dynamic Frames Tutorial
  - NAIF IDs Tutorial
  - Headers for the routines mentioned in this tutorial
  - Most Useful SPICELIB Routines
  - FRMDIFF User's Guide
  - Porting\_kernels tutorial
- Related documents
  - CK Required Reading
  - PCK Required Reading
  - SPK Required Reading
  - Rotations Required Reading



---

Navigation and Ancillary Information Facility

# Using the Frames Subsystem

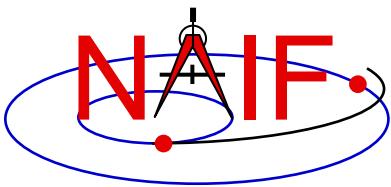
January 2020



# What is the Power of Frames?

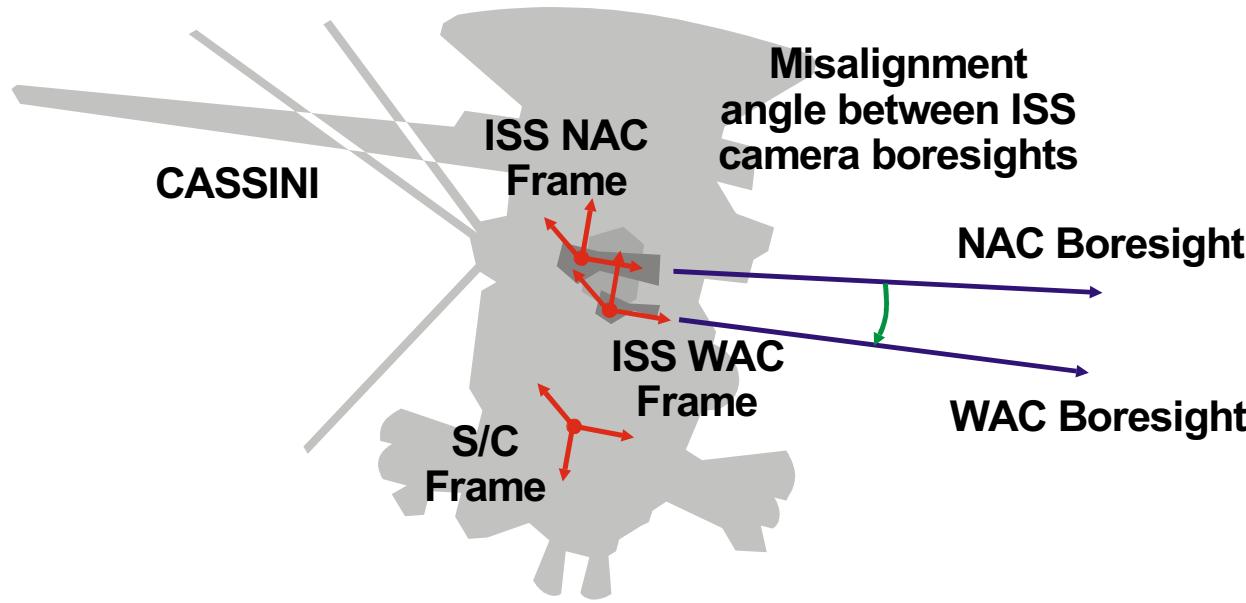
Navigation and Ancillary Information Facility

- The “power” of the Frames capability stems from the SPICE system’s ability to construct complex reference frame transformations with no programming effort required of you - the end user
  - But it’s crucial that you select and load the needed kernels
- The principal benefit from the Frames capability is obtained through the main SPK subsystem interfaces (SPKEZR and SPKPOS) and the Frames subsystem interfaces (SXFORM, PXFORM, PXFRM2)
- The remaining pages illustrate typical use of frames
- Several **VERY IMPORTANT** usage issues are mentioned in the Frames tutorial; be sure to also read that.



# Offset Between Instruments

Navigation and Ancillary Information Facility



Required Kernels:

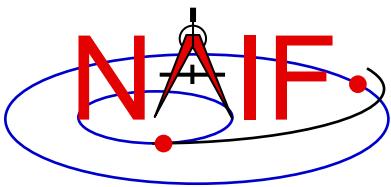
- Generic LSK
- Mission FK
- Camera IK(s)

ISS = Imaging Science System

Compute the angular separation between the Cassini ISS Narrow Angle Camera and Wide Angle Camera boresights:

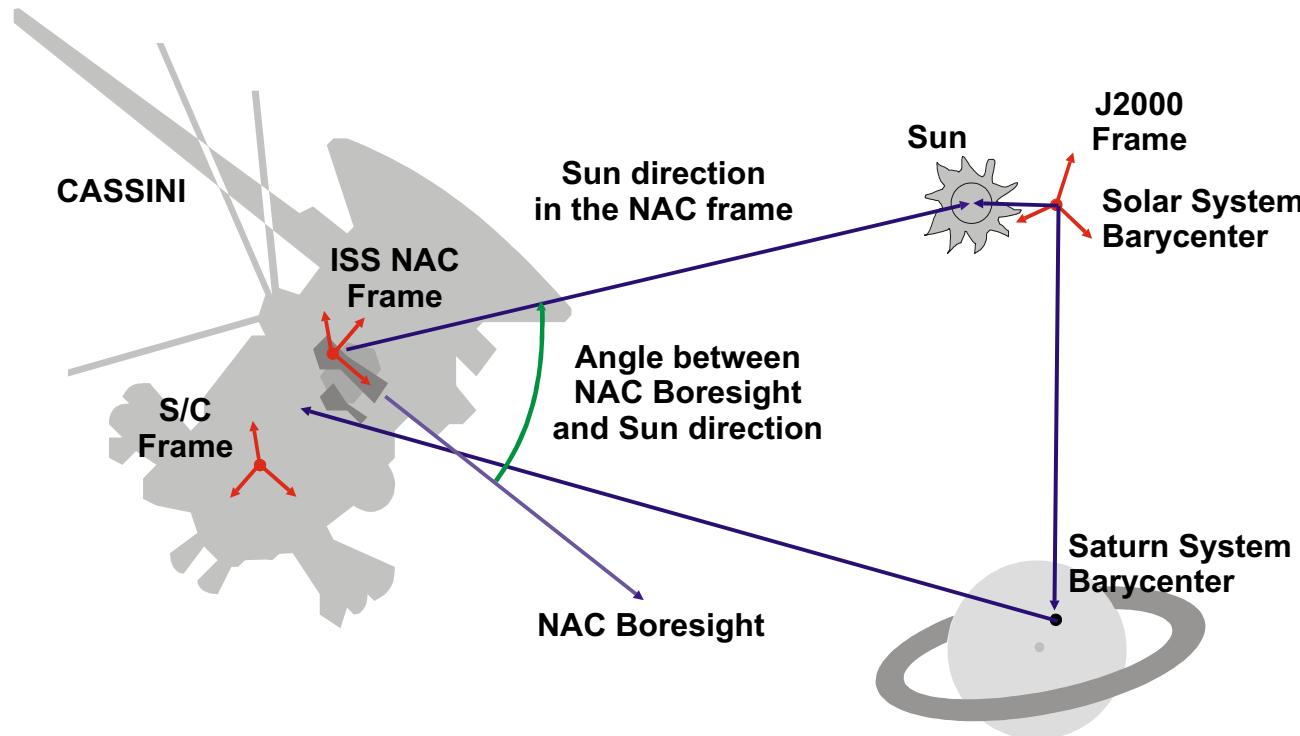
C      Retrieve the matrix that transforms vectors from NAC to WAC frame  
CALL PXFORM( 'CASSINI\_ISS\_NAC' , 'CASSINI\_ISS\_WAC' , ET, MAT )  
C      Transform NAC boresight to WAC frame and find separation angle  
CALL MXV ( MAT, NAC\_BORESIGHT\_nac, NAC\_BORESIGHT\_wac )  
ANGLE = VSEP( NAC\_BORESIGHT\_wac , WAC\_BORESIGHT\_wac )

Fortran example



# Angular Constraints

Navigation and Ancillary Information Facility



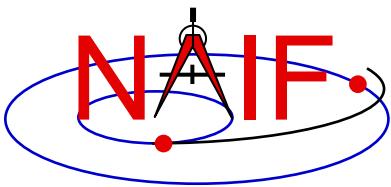
**Required Kernels:**

- Generic LSK
- Mission FK
- Spacecraft SCLK
- Camera IK
- Planetary Ephemeris SPK
- Spacecraft SPK
- Spacecraft CK

**Check whether the angle between the camera boresight and the direction to the Sun is within the allowed range:**

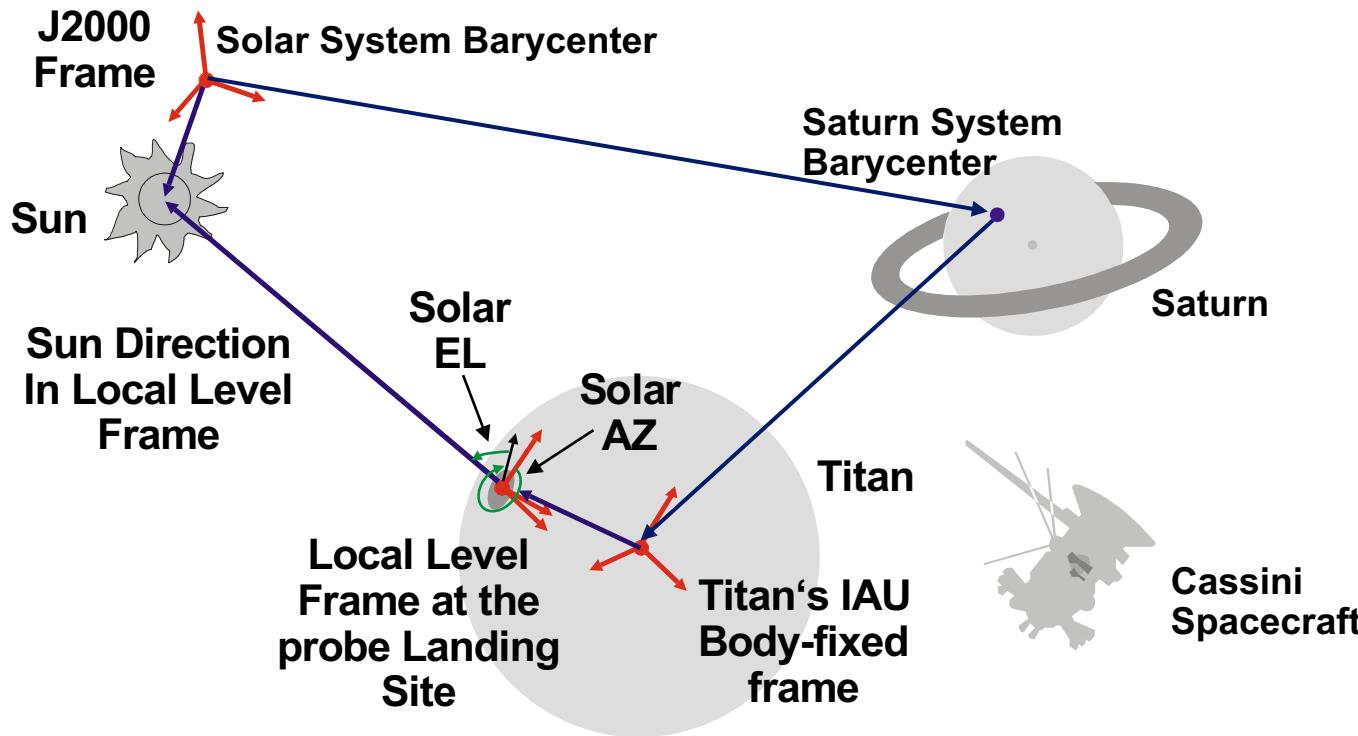
```
CALL SPKPOS( 'SUN', ET, 'CASSINI_ISS_NAC', 'LT+S', 'CASSINI', SUNVEC, LT )
ANGLE = VSEP( NAC_BORESIGHT_nac, SUNVEC )
IF ( ANGLE .LE. CONSTRAINT ) WRITE(*,*) 'WE ARE IN TROUBLE!'
```

Fortran example



# Angles at the Surface

Navigation and Ancillary Information Facility



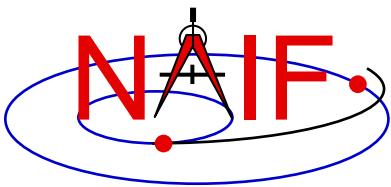
**Required Kernels:**

- Generic LSK
- Generic PCK
- Mission FK
- Planetary Ephemeris SPK
- Satellite Ephemeris SPK
- Landing Site SPK

**Compute solar azimuth and elevation at the Huygens probe landing site**

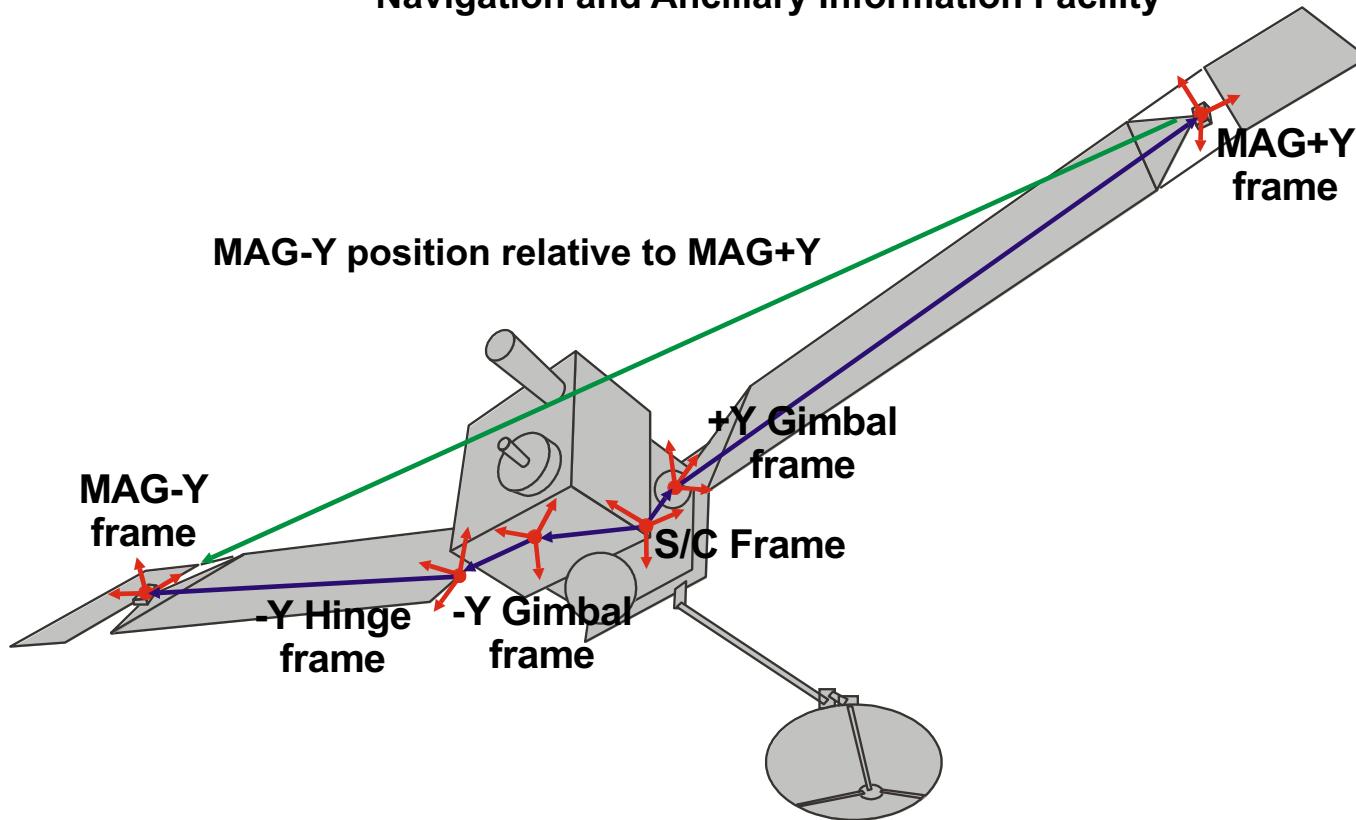
```
CALL SPKPOS('SUN', ET, 'HUYGENS_LOCAL_LEVEL', 'LT+S', 'HUYGENS_PROBE', SUNVEC, LT)
CALL RECLAT(SUNVEC, R, AZIMUTH, ELEVATION)
ELEVATION = -ELEVATION
IF (AZIMUTH .LT. 0.D0) THEN
  AZIMUTH = AZIMUTH + TWOPI()
ENDIF
```

Fortran example



# Relative Position of Sensors

Navigation and Ancillary Information Facility



Required Kernels:

- Generic LSK
- Mission FK
- Structure Locations SPK
- Spacecraft SCLK
- Solar Array CK

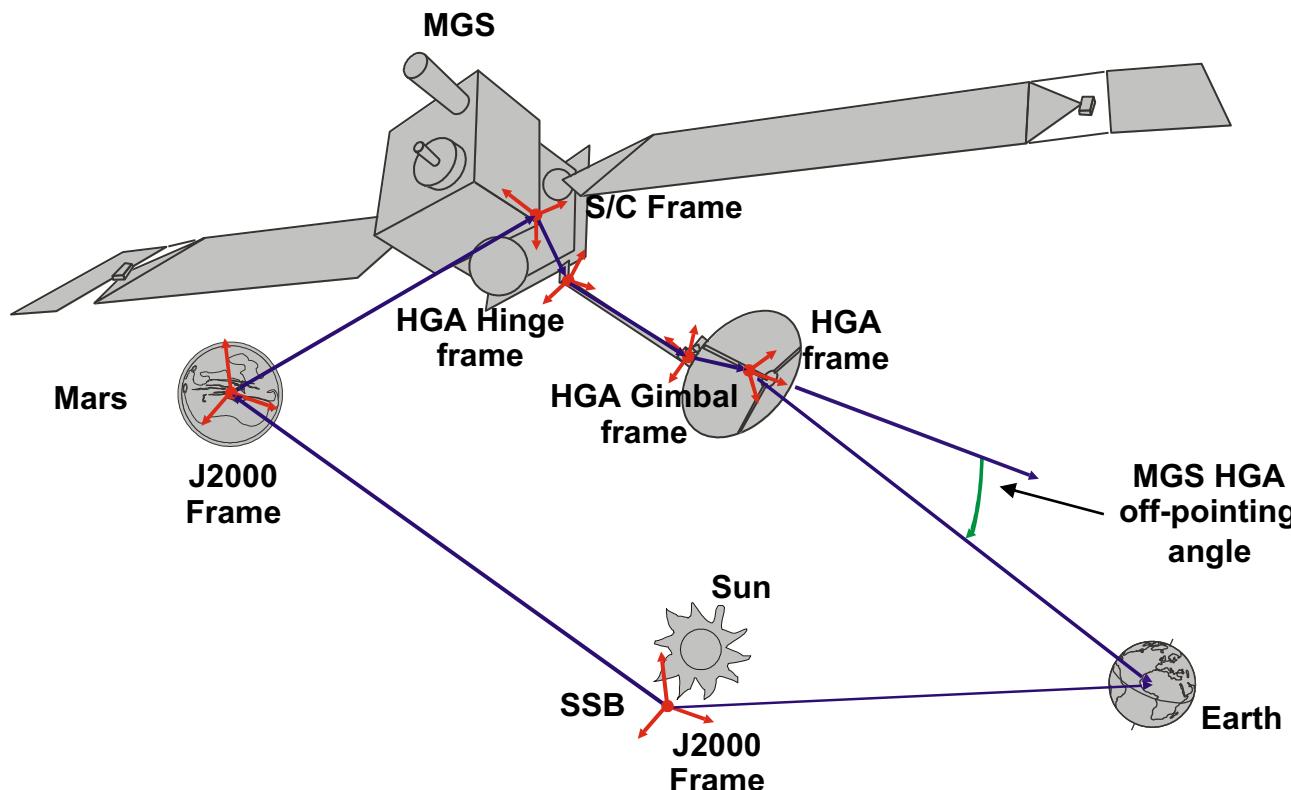
Find the position of one MGS MAG sensor with respect to the other in the MGS s/c frame. Also find the relative orientation of the sensors:

```
CALL SPKEZR('MGS_MAG-Y', ET, 'MGS_SPACECRAFT', 'NONE', 'MGS_MAG+Y', STATE, LT)  
CALL PXFORM('MGS_MAG_+Y_SENSOR', 'MGS_MAG_-Y_SENSOR', ET, MAT)
```

Fortran  
example

# Manipulators - 1

Navigation and Ancillary Information Facility



- Required Kernels:**
- Generic LSK
  - Mission FK
  - Spacecraft SCLK
  - HGA IK
  - Structure Locations SPK
  - Planetary Ephemeris SPK
  - Spacecraft SPK
  - Spacecraft CK
  - HGA CK

HGA = High Gain Antenna

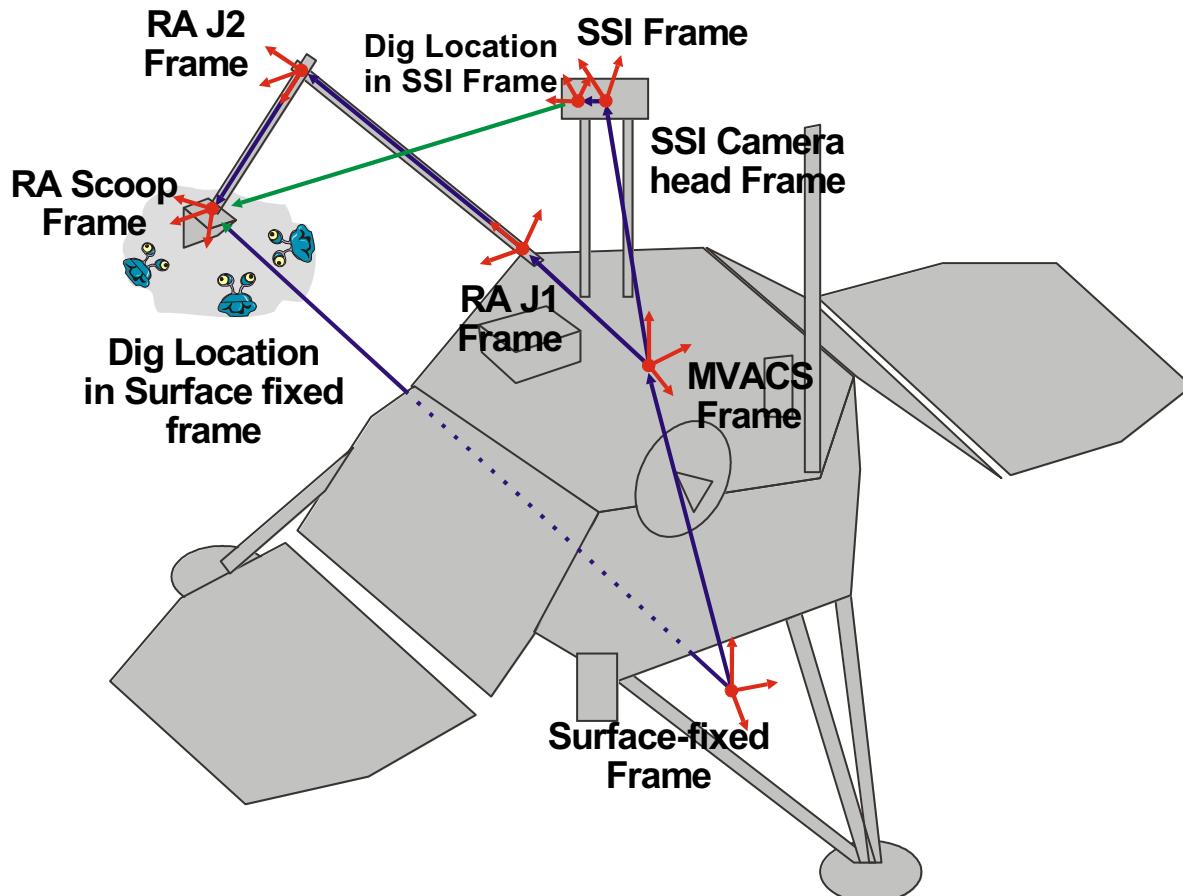
Compute the angle between the direction to Earth and the MGS HGA boresight:

```
CALL SPKEZR( 'EARTH', ET, 'MGS_HGA', 'LT+S', 'MGS', EARTH_STATE, LT )
ANGLE = VSEP( HGA_BORESIGHT, EARTH_STATE )
```

Fortran  
example

# Manipulators - 2

Navigation and Ancillary Information Facility



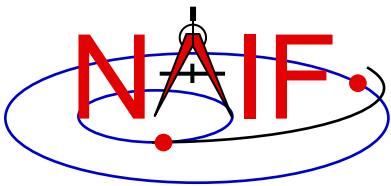
**Required Kernels:**

- Generic LSK
- Mission FK
- Lander SCLK
- Structure Locations SPK
- Lander SPK
- Lander CK
- SSI CK
- RA CK

Compute the soil digging location in the MPL surface-fixed and camera left eye frames:

```
CALL SPKEZR( 'MPL_RA_SCOOP' ,ET, 'MPL_SURFACE_FIXED' , 'NONE' , 'MPL_SURF' ,ST1,LT )
CALL SPKEZR( 'MPL_RA_SCOOP' ,ET, 'MPL_SSI_LEFT' ,           'NONE' , 'MPL_SSI' , ST2,LT )
```

Fortran example

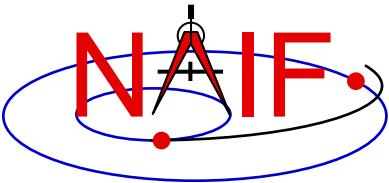


---

Navigation and Ancillary Information Facility

# “High Accuracy” Orientation and Body-fixed Frames for the Moon and Earth

January 2020

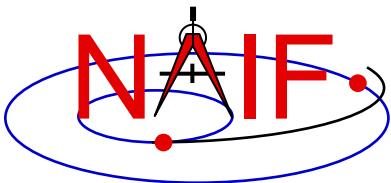


# Topics

---

Navigation and Ancillary Information Facility

- **Introduction**
- **Earth binary PCKs**
- **Lunar binary PCKs**
- **Lunar Frames Kernel**
  - Frame specifications
  - Frame alias names
- **Binary PCK file format**
- **Using Binary PCKs**
  - Precedence rules
  - Utilities
- **Backup**
  - Earth and Moon frame association kernels

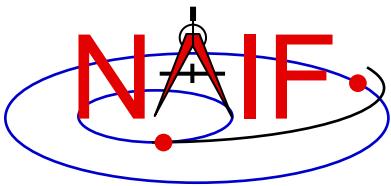


# Introduction-1

---

Navigation and Ancillary Information Facility

- Having read about “standard” PCKs and FKs in other tutorials you may want to learn about several “special” PCKs and FKs dealing with the Earth and the Moon.
- While it is ultimately up to you, in most cases you should use the PCK and FK kernels described here when working with the Earth or the Moon.
  - If you need only “low accuracy” Earth or Moon orientation information, for instance for a visualization tool, then the standard text-style PCK data may be used.

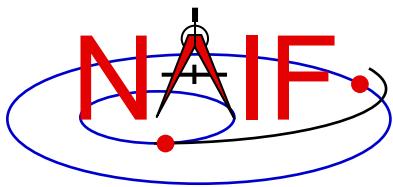


# Introduction-2

---

Navigation and Ancillary Information Facility

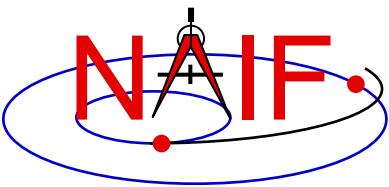
- NAIF provides “high accuracy” orientation data for the Earth and the Moon in binary PCKs.
  - For the Earth, three versions are made:
    - » High accuracy, frequently updated file
      - Contains high accuracy historical data and fairly accurate, short-term predict data
    - » High accuracy, infrequently updated historical file
    - » Lower accuracy long term predict file
  - For the Moon, a single, long-term file is made upon release of an official new JPL “Developmental Ephemeris” (DE).
    - » Contains accurate historical and predictive lunar orientation data
- To use these kernels:
  - Select binary PCK(s) having properties and time coverage that meet your needs
    - » Unlike text PCKs, the time span covered by binary PCKs is limited
  - Load the PCK(s) using FURNSH
  - For the Moon, also load the lunar FK
  - Reference the Earth body-fixed frame using the name ‘ITRF93’
    - » **CAUTION:** ‘IAU\_EARTH’ **cannot** be used to reference high-accuracy Earth orientation data
  - Reference a lunar body-fixed frame using one of these names:
    - » ‘MOON\_ME’ (Moon Mean Earth/Rotation axis frame)
    - » ‘MOON\_PA’ (Moon Principal Axes frame)
    - » **CAUTION:** ‘IAU\_MOON’ **cannot** be used to reference high-accuracy lunar orientation data



---

Navigation and Ancillary Information Facility

# Earth Binary PCKs

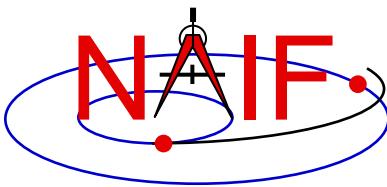


# High Accuracy Earth Rotation Model

---

Navigation and Ancillary Information Facility

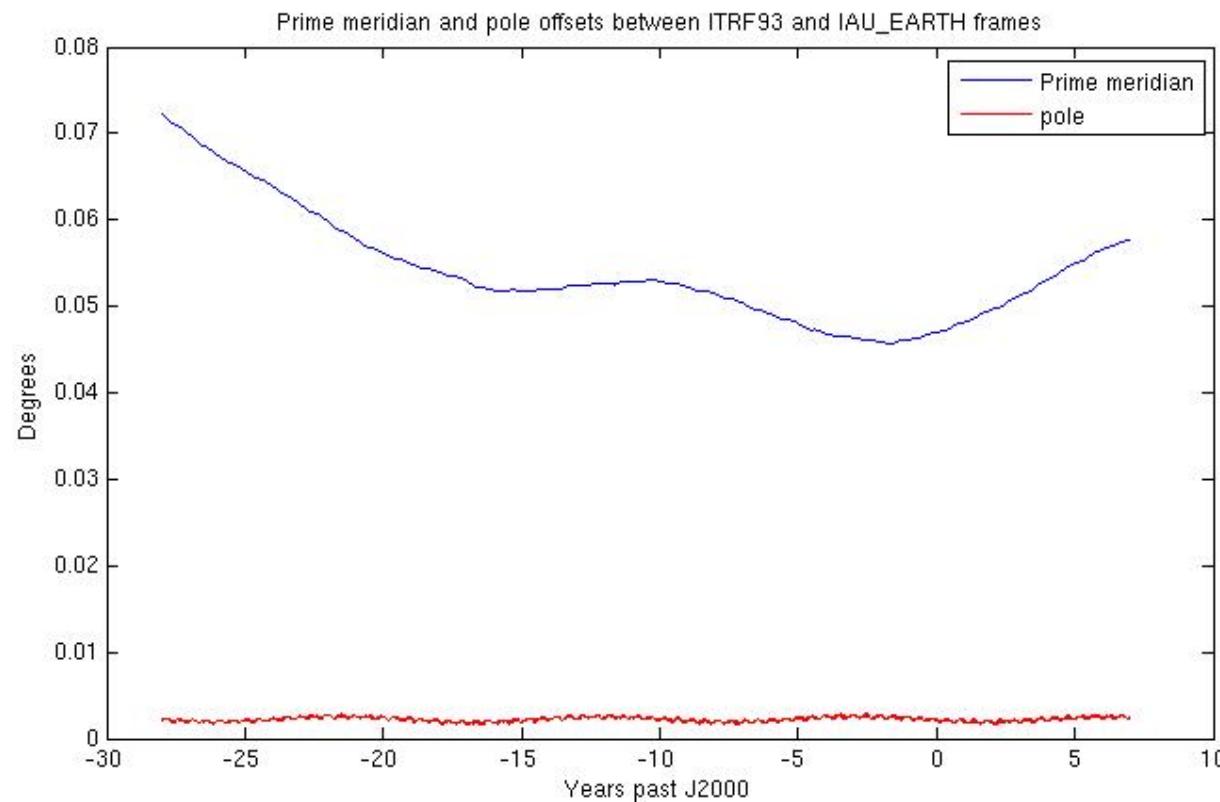
- **The ITRF93 high accuracy Earth rotation model takes into account:**
  - Precession: 1976 IAU model due to Lieske.
  - Nutation: 1980 IAU model, with IERS corrections due to Herring et al.
  - True sidereal time using accurate values of TAI-UT1
  - Polar motion
- **It is more accurate than the IAU rotation models found in text PCKs.**
  - See the plot on the next slide comparing orientation of the ITRF93 frame to that of the IAU\_EARTH frame.
    - » IAU\_EARTH frame orientation error is ~0.06 degrees (~1 milliradian), or ~6km on a great circle!
- **The highest accuracy is obtainable only for past epochs.**
  - Unpredictable variations of UT1-TAI and polar motion limits the accuracy of predicted earth orientation. See plot on page 8.

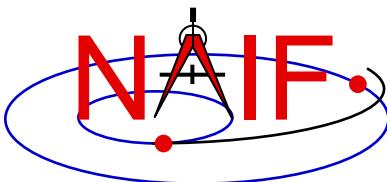


# IAU\_EARTH vs ITRF93 Comparison Plot

Navigation and Ancillary Information Facility

Difference between the 'IAU\_EARTH' frame and the 'ITRF93' frame

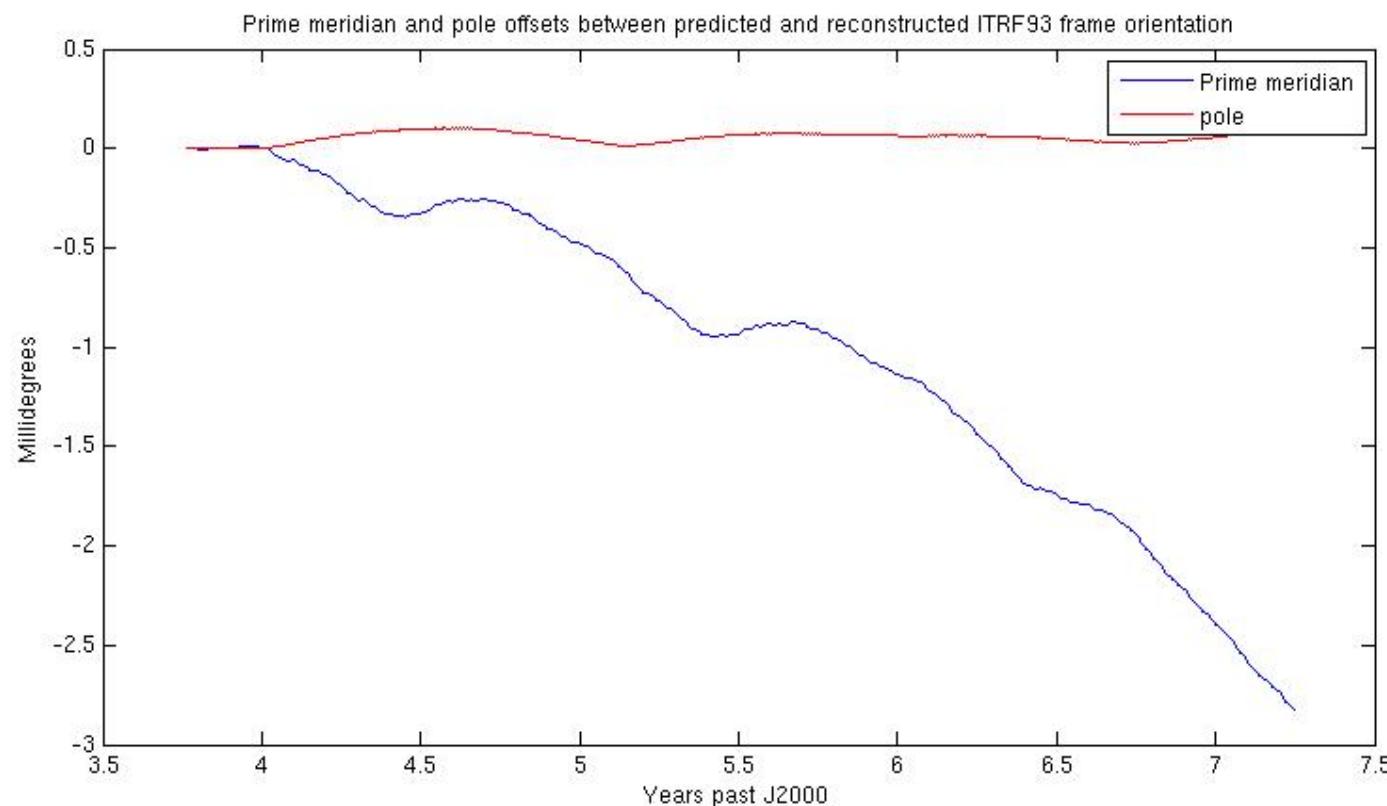


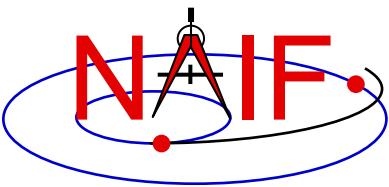


# Earth Predicted vs Reconstructed ITRF93 Plot

Navigation and Ancillary Information Facility

Difference between predicted and reconstructed orientation of ITRF93 frame





# Data Source for Earth High Accuracy Model

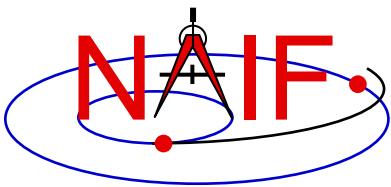
---

Navigation and Ancillary Information Facility

- **Binary Earth PCKs represent the orientation of an Earth ITRFxx body-fixed reference frame relative to the ICRF.**
  - ITRF frames are defined by the International Earth Rotation Service (IERS).
  - Currently only the ITRF93 frame is supported within SPICE.
    - » An update to a more modern version is planned.
  - Source data come from a JPL Earth Orientation Parameters (EOP) file.

**ICRF** = International Celestial Reference Frame, often referred to in SPICE as the “J2000” frame, and also often referred to as the EME 2000 frame. This is an inertial frame.

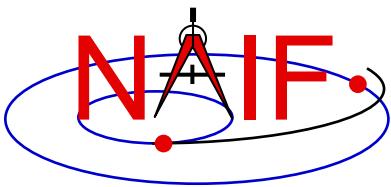
**ITRF** = International Terrestrial Reference Frame



# Earth PCK Production Scheme

Navigation and Ancillary Information Facility

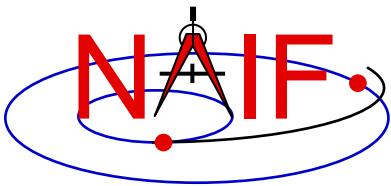
- Three versions of the high accuracy binary Earth PCK are produced
  - “**The latest**,” using each new release by JPL of a reconstructed EOP file
    - » Covers well into the past and approximately two months into the future beyond the production date
    - » Accuracy of the future data degrades rapidly past the production date
    - » Produced several times per week using an automated script
  - **Long term predict**, for uses beyond two months from the production date, and where the highest accuracy is not required
    - » Produced infrequently
    - » Covers several years into the past and approximately 30 years into the future
    - » Accuracy at epochs in the future is low compared to that for past epochs, but any of it is far better than what is obtained from the IAU rotation model for the earth ('IAU\_EARTH') provided in any text PCK
  - **History file**, containing only high accuracy historical data
- All are in the pck directory under generic\_kernels on the NAIF server: [https://naif.jpl.nasa.gov/pub/naif/generic\\_kernels/pck/](https://naif.jpl.nasa.gov/pub/naif/generic_kernels/pck/)
  - Read the “aareadme” file to see the file naming schema and more details



# Accurate Earth Surface Locations

Navigation and Ancillary Information Facility

- **High accuracy determination of surface locations relative to an inertial frame involves motions in addition to Earth rotation, including:**
  - tectonic plate motion
  - tidal effects
  - relativistic effects
- **From the list of effects above...**
  - Tectonic plate motion is accounted for in NAIF's DSN and some non-DSN station SPK files.
    - » This helps ONLY station locations, not other surface objects.
  - The other two non-rotational effects affecting surface locations are not accounted for by any PCK, or by any other SPICE component. But the magnitude of these is under one meter.

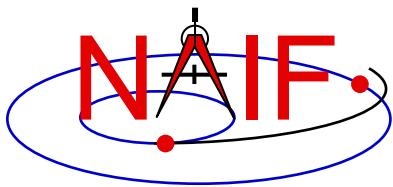


# Kernel Usage Summary: Earth

---

Navigation and Ancillary Information Facility

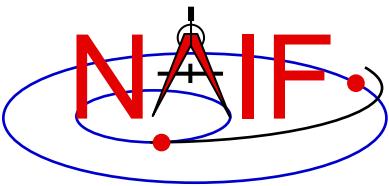
- To use high accuracy Earth orientation data
  - Load one or more binary Earth PCKs
    - » If a long-term predict is used, load that kernel **\*before\*** loading any kernel containing reconstructed data so that the reconstructed data have precedence during the overlap period.
    - If your application uses any of the old, pre-N0062 APIs that make use of the default Earth body-fixed frame (see backup slides), load an Earth frame association kernel making ITRFxx the default Earth body-fixed frame.
      - » **But best to switch to use the “new” APIs that require you to specify which frame to use.**
        - The new APIs are: ILUMIN, SINCPT, SUBPNT, SUBSLR
  - If you’re using SPICE to access Earth size and shape information, you’ll also need to load a text PCK file containing these data.
    - Typically use the latest generic text PCK: pck000xx.tpc



---

**Navigation and Ancillary Information Facility**

# Lunar Binary PCKs

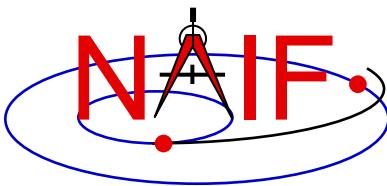


# High Accuracy Lunar Rotation Model

---

Navigation and Ancillary Information Facility

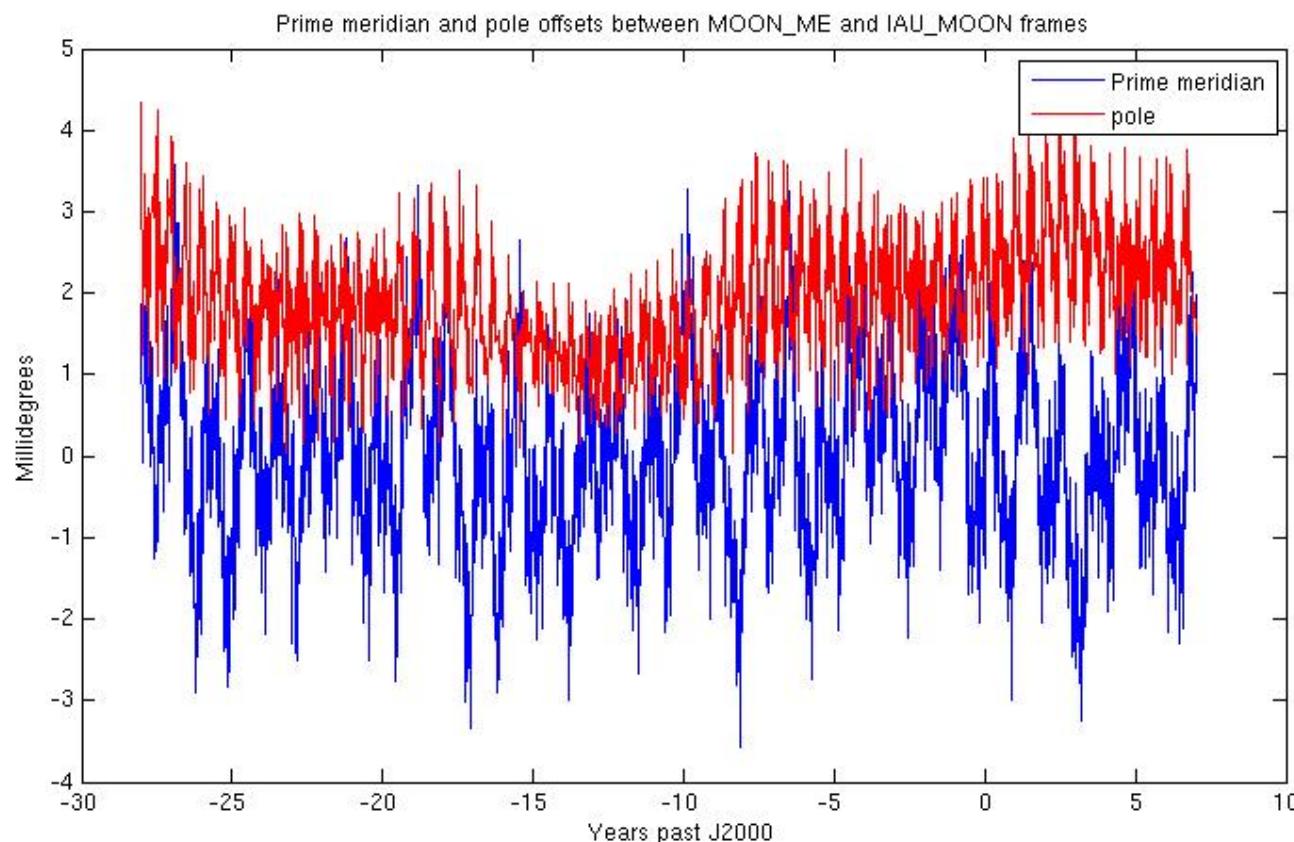
- The high accuracy lunar rotation models available in binary PCKs are more accurate than the IAU rotation model found in a text PCK ('IAU\_MOON').
  - For the time period of 2000-2020 the difference is approximately:
    - » Worst case: ~0.0051 degrees, or ~155m on a great circle
    - » Average: ~0.0025 degrees, or ~76m on a great circle
  - The error is due to truncation of the libration series in the IAU model
  - See the plot in the following chart comparing the IAU lunar rotation model to the integrated DE-421 model.

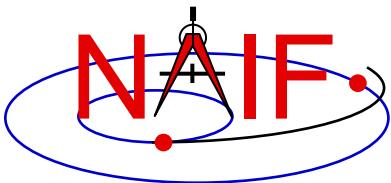


# IAU\_Moon vs MOON\_ME Comparison Plot

Navigation and Ancillary Information Facility

Difference between the IAU\_Moon frame and the Moon\_ME frame (equivalent to the Moon\_ME\_DE421 frame)





# Data Sources for High Accuracy Models

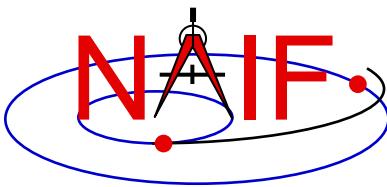
---

Navigation and Ancillary Information Facility

- **Data for lunar orientation come from JPL's DE/LExxx planet/lunar ephemeris files.**
  - Binary lunar PCKs represent the orientation of the Moon's "principal axis" reference frame—referred to as **MOON\_PA\_Dexxx**—relative to the ICRF\*.

ICRF = International Celestial Reference Frame, often referred to in SPICE as the "J2000" frame, and also often referred to as the EME 2000 frame. This is an inertial frame.

JPL-produced planet/lunar ephemeris files are sometimes referred to as "DE/LExxx" but more often are referred to as simply "DExxx."

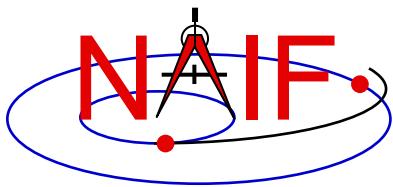


# Lunar Rotation Model Effects

Navigation and Ancillary Information Facility

- The high accuracy lunar orientation model obtained from the DE421 lunar ephemeris represents the result of a simultaneous numerical integration of lunar rotation and orbit, and of orbits of the planets.
  - The DE421 integration model includes\*:
    - » A “solid Moon”
    - » Torques on the Moon from the static gravity field of degree 2-4. Torque is due to Earth, Sun, Venus, and Jupiter.
    - » Torques on the Moon and moments of inertia due to (degree 2) tides raised by Earth.
    - » Dissipation effects on torques and moments due to tides on the Moon.
    - » Torques due to Earth J2 interacting with Moon degree 2 (J2 and C22).
  - Lunar quantities fit for DE421 include:
    - » Initial conditions for lunar orbit and rotation of body
    - » Moment of inertia difference  $(C_{\text{axis}} - A_{\text{axis}})/B_{\text{axis}}$  and  $(B - A)/C$
    - » Third-degree gravity field coefficients
    - » Tidal Love numbers and dissipation
    - » Locations of four laser retroreflector arrays
- It is anticipated that further improvements in the orientation of the Moon will become available in new DExxx-based kernels in the future.

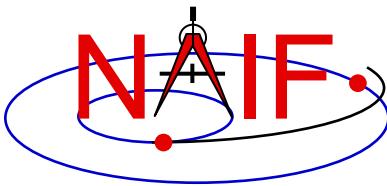
\*Description provided by James G. Williams (JPL)



---

Navigation and Ancillary Information Facility

# Lunar Frames Kernel

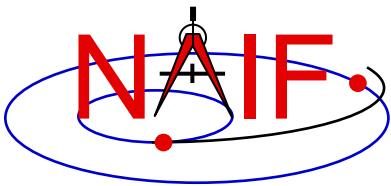


# Lunar Frames Kernel

---

Navigation and Ancillary Information Facility

- A lunar frames kernel is available from NAIF. It has four functions.
  1. Make two lunar frames—Principal Axes (PA) and Mean Earth/rotation axis (ME)—known to the SPICE system.
    - » Within SPICE their names are MOON\_PA\_DExxx and MOON\_ME\_DExxx
    - » These frames are unique to a particular JPL-produced planetary and lunar ephemeris (e.g. DE421).
  2. Connect the MOON\_PA\_DExxx frame name to the high accuracy lunar orientation PCK data that implement the PA orientation.
  3. Provide specifications for implementing the rotation between the PA frame and the ME frame.
    - » Makes the MOON\_ME\_DExxx frame available to SPICE.
  4. Provide generic frame names, aliased to the MOON\_PA\_DExxx and MOON\_ME\_DExxx frame names.
    - » The generic frame names are simply **MOON\_PA** and **MOON\_ME**.
    - » The generic names need not be changed in your programs when the MOON\_PA\_DExxx and MOON\_ME\_DExxx names change due to use of new defining data.
      - » The DE-specific frames to which these aliases “point” will be updated by NAIF whenever a new binary lunar orientation PCK is produced. NAIF will release a new lunar FK at that time.

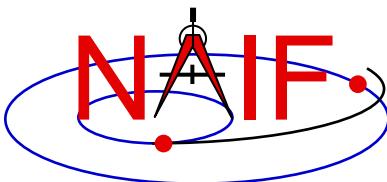


# Kernel Usage Summary: Moon

---

Navigation and Ancillary Information Facility

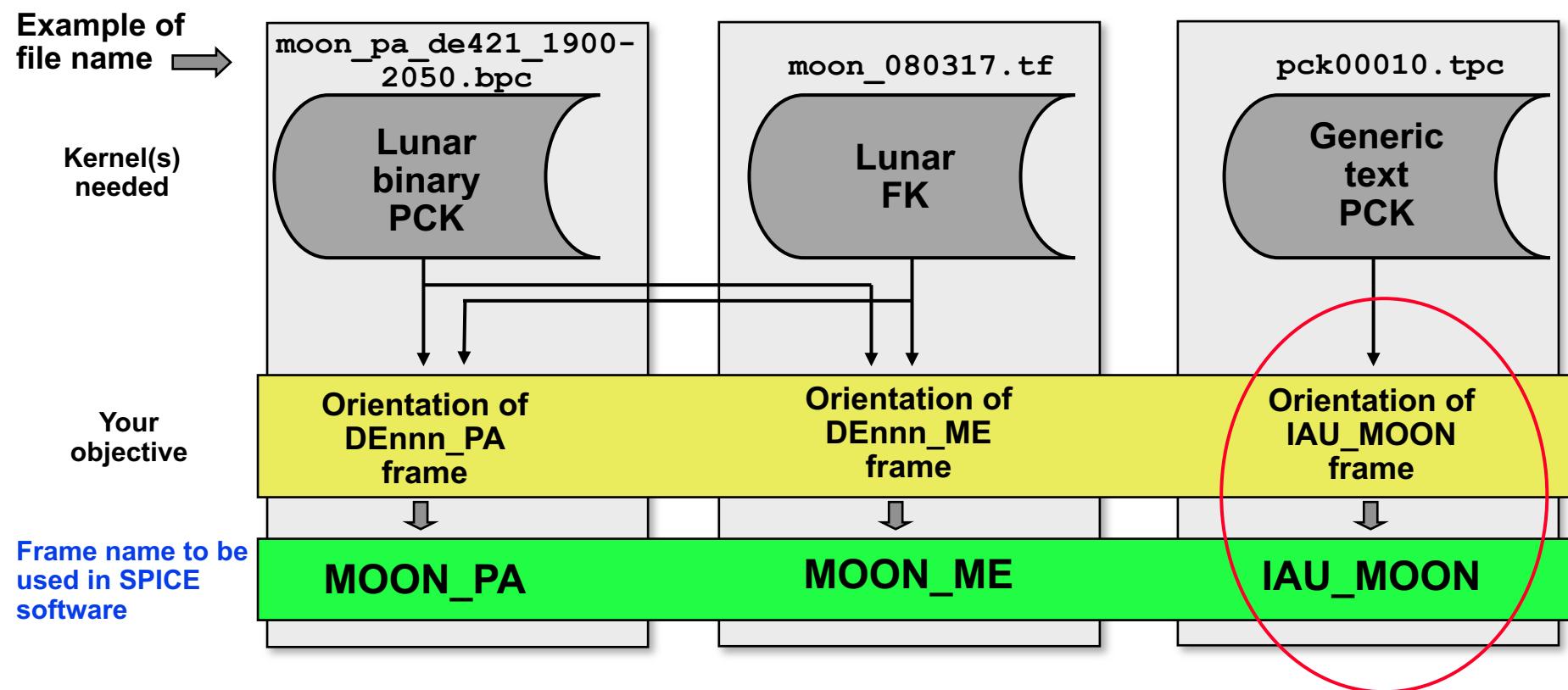
- **To use high accuracy Moon orientation data**
  - [Load the current binary lunar PCK](#)
  - [Load the current lunar FK](#)
  - If your application uses any of the old, pre-N0062 APIs that make use of the default lunar body-fixed frame (see [Backup](#)), load a Moon frame association kernel making either MOON\_ME or MOON\_PA the default lunar body-fixed frame.
    - » But best to switch to use the “new” APIs that require you to specify which frame to use.
      - The new APIs are ILUMIN, SINCPT, SUBPNT, SUBSLR
- **If you’re using SPICE to access Moon size and shape information, you’ll also need to [load a text PCK](#) file containing these data.**
  - Typically use the latest generic text PCK, such as pck00010.tpc



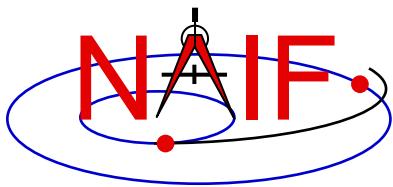
# Lunar PCK/FK Summary

Navigation and Ancillary Information Facility

Which kernels are needed to access each of the three lunar body-fixed reference frames providing lunar orientation?



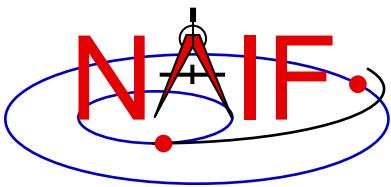
Usually a bad  
choice for the  
moon!



---

Navigation and Ancillary Information Facility

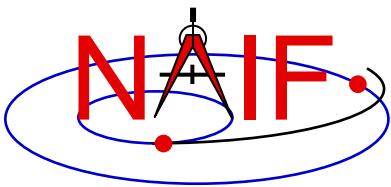
# Binary PCK File Format



# Binary PCK File Format

Navigation and Ancillary Information Facility

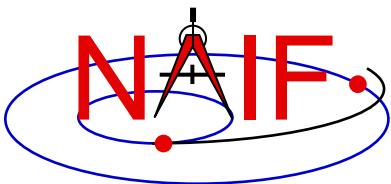
- **SPICE binary PCK files are used to accommodate high accuracy rotation models.**
  - Just as for SPKs and CKs, the data are held in SPICE Double Precision Array files (DAF)
  - Multiple types are supported
    - » Type 2: Chebyshev polynomials are used to represent Euler angles giving orientation as a function of time. Rates are obtained by differentiating polynomials. Coverage intervals have fixed length.
      - Used for the Earth and the Moon
    - » Type 3: Separate sets of Chebyshev polynomials are used to represent Euler angles and their rates. Coverage intervals have variable length.
      - Used only for Eros
    - » Type 20: Chebyshev polynomials representing Euler angle rates
      - Provided to accurately represent “EPM” orientation data produced by the Russian Institute of Applied Astronomy (Ephemeris of the Planets and Moon)
  - Binary PCKs include a “comment area” for storing descriptive metadata
    - » Access the comment area using the Toolkit’s *commnt* utility program
  - Binary PCKs support high-speed direct access
    - » Chebyshev polynomials are fit to source Euler angles; these evaluate very quickly



---

Navigation and Ancillary Information Facility

# Using Binary PCKs

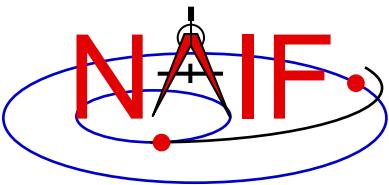


# Precedence Rules for Text and Binary PCKs

---

Navigation and Ancillary Information Facility

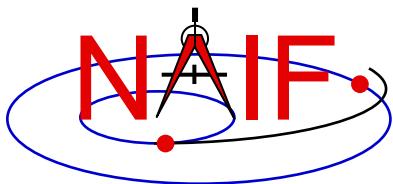
- If two (or more) binary PCKs with functionally equivalent data are loaded, a later loaded file takes precedence.
- Loading one text PCK that supersedes another can lead to errors if data from the “old” PCK remain in the kernel pool.
  - It’s essential to unload the old text PCK before loading the new one.
    - » Use UNLOAD or KCLEAR to unload the old text PCK.
  - This problem doesn’t apply to binary PCKs.
- If both a binary and a text PCK provide orientation for the same frame, data available from the binary PCK **always** take precedence over data available from the text PCK.
  - This is independent of file loading order
  - Note: the binary PCKs discussed in this tutorial define Earth-fixed and Moon-fixed frames different from those defined by a text PCK (e.g. pck00010.tpc), so there will be no conflict.



# Tools for use with Binary PCKs

Navigation and Ancillary Information Facility

- Use the ***commnt*** utility to access a binary PCK comment area
  - Read, extract or insert metadata
- Use the ***brief*** or ***spacit*** utility to summarize a binary PCK
  - *brief* is easier to use; *spacit* provides more information
- Non-native binary PCKs can be read **without** first being converted to the native binary form
  - Converting a non-native binary PCK to native form will also speed up data access somewhat. Use *toxfr/tobin* or *bingo*.
    - » *toxfr* and *tobin* are available in each Toolkit; *bingo* is available only from the NAIF website

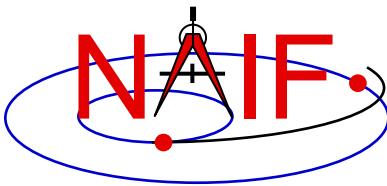


---

Navigation and Ancillary Information Facility

# Backup

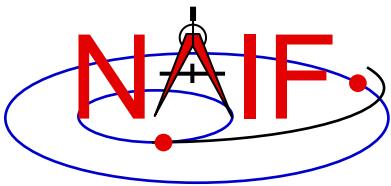
**Association Frames Kernels  
for the Earth and the Moon**



# Association FKs: Introduction

Navigation and Ancillary Information Facility

- In most SPICE modules that deal with one or more reference frames the name(s) of that/those frame(s) must be provided as input argument(s), for example:
  - CALL SPKEZR (target, time, **frame**, observer, correction, state, lighttime)
- Many years ago the SPICE developers assumed there would be only one body-fixed reference frame associated with each natural body during a program run.
  - Thus a specific body-fixed frame name would rarely be needed as an input to modules dealing with body-fixed frames
  - Instead, SPICE could use the body-fixed frame associated with a given body simply by knowing the body name or ID
    - » For most bodies SPICE associates the body with a body-fixed frame named **IAU\_<body name>** (example: **IAU\_MOON**)
    - » This is known as the default body-fixed frame
- This was a bad assumption... at least for the Earth and the Moon!
  - Multiple body-fixed frames exist for the Earth and the Moon
  - The default body-fixed frames for the Earth and the Moon, for which the defining data are provided in a generic text PCK (taken from an IAU report), are inaccurate (for the Moon) and very inaccurate (for the Earth) representations of the actual orientations of these bodies

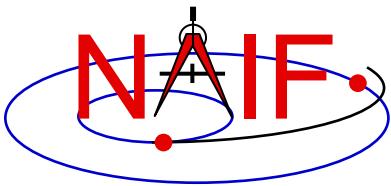


# Better Choice for the Default

Navigation and Ancillary Information Facility

- For the Earth and the Moon there are other choices for body-fixed frame that are almost certainly better than the default body-fixed frame conjured up by SPICE

<u>Body</u>	<u>SPICE Default Body-fixed Frame</u>	<u>Better choice</u>
Earth	IAU_Earth	ITRF93 ITRFxx (in the future)
Moon	IAU_Moon	Moon_PA or Moon_ME



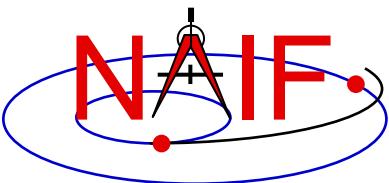
# The Problem

Navigation and Ancillary Information Facility

- The SPICE modules that make use of the default body-fixed reference frame are these
  - LSPCN, ET2LST, **ILLUM, SRFXPT, SUBPT, SUBSOL** (and their C, Icy and Mice equivalents)
  - Your code might overtly call one of these, or it could call one indirectly through use of a parameterized dynamic frame
- NAIF rules regarding stability of our software offerings prevent us from changing the designs of those modules
  - So we have provided you means to change the default body-fixed frame associated with any solar system body of interest to you. See the next several pages.
- However, starting with the version N62 Toolkits, a new set of modules is available for those calculations where precision body orientation is important.
  - These modules require the user to name the desired body-fixed frame, rather than using a default body-fixed frame
  - The new modules are these:
    - » **ILUMIN, SINCPT, SUBPNT, SUBSLR**

*Old: still available, but better to use those noted below*

*New: safer to use, and offer improved accuracy in some cases*



# Changing the Default Body-Fixed Frame Name in the Older APIs

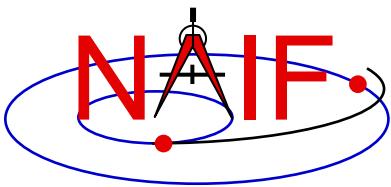
---

Navigation and Ancillary Information Facility

- All bodies for which a body-fixed frame is defined by the IAU, and where the defining data are found in a SPICE text PCK file, have an associated default body-fixed frame name within SPICE:
  - The name pattern is: IAU\_<body name>
  - Examples: IAU\_MARS, IAU\_MOON, IAU\_EARTH
- A different default body-fixed frame name can be assigned within a program by placing the following assignment in any text kernel that is loaded into the program:

```
OBJECT_<body name>_FRAME = '<new default frame name>'
```

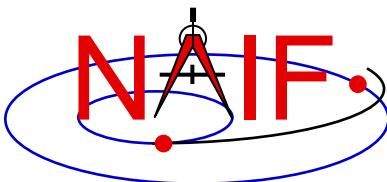
  - Example: OBJECT\_MOON\_FRAME = 'MOON\_ME'
- NAIF offers three “association FKs” to accomplish this.
  - See next page.



# Using Association FKs to Change the Default Frame in the Older APIs

Navigation and Ancillary Information Facility

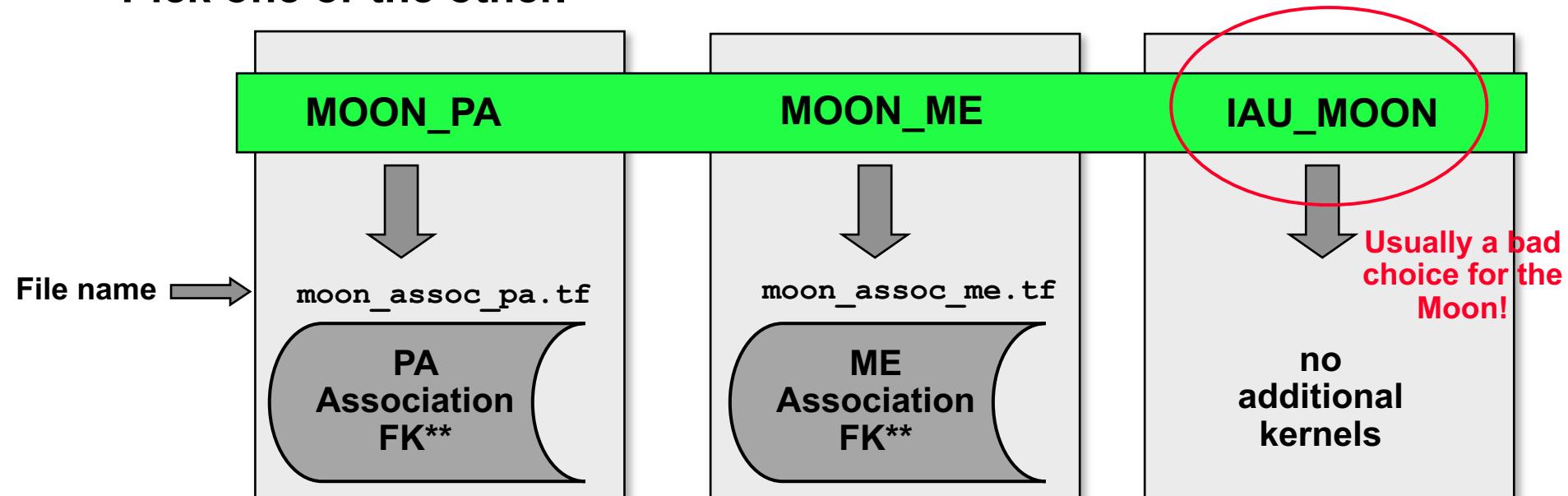
- For the Earth and the Moon, changing the default body-fixed frame name as described on the previous page can be accomplished by loading the appropriate “association” frame kernel provided by NAIF. The association kernels available are shown below
  - For the Earth:
    - » earth\_assoc\_itrf93.tf
  - For the Moon: (pick one or the other—not both)
    - » moon\_assoc\_me.tf
    - » moon\_assoc\_pa.tf
- These kernels are available on the NAIF server
  - For the Earth:
    - » [https://naif.jpl.nasa.gov/pub/naif/generic\\_kernels/fk/planets/](https://naif.jpl.nasa.gov/pub/naif/generic_kernels/fk/planets/)
  - For the Moon:
    - » [https://naif.jpl.nasa.gov/pub/naif/generic\\_kernels/fk/satellites/](https://naif.jpl.nasa.gov/pub/naif/generic_kernels/fk/satellites/)



# Lunar FK/PCK/Association FK Usage

Navigation and Ancillary Information Facility

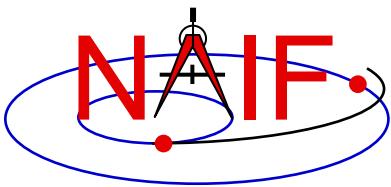
Which **additional** kernel is needed to use the indicated frame in those (older) SPICE APIs\* that use a default (assumed) frame?  
Pick one or the other.



\* **LSPCN, ET2LST, ILLUM, SRFXPT, SUBPT, SUBSOL**  
(and their C, Icy and Mice equivalents)

\*\*Any version of one or the other of these kernels is good indefinitely;  
you do not need to use the latest instance offered on the NAIF server.

But best to use the replacements for these four, which don't use a default body-fixed frame:  
• **ILUMIN**  
• **SINCPT**  
• **SUBPNT**  
• **SUBSLR**

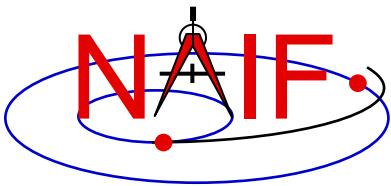


---

Navigation and Ancillary Information Facility

# Dynamic Reference Frames

January 2020



# Topics

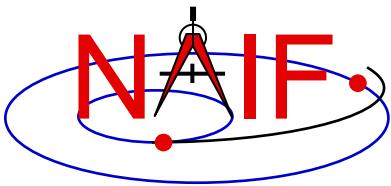
---

Navigation and Ancillary Information Facility

- Overview of Dynamic Reference Frames
- Terminology
- Rationale for Dynamic Frames
- How to Use a Dynamic Frame
- Parameterized Dynamic Reference Frames
- Defining Dynamic Reference Frames
  - Two-Vector Frames
  - "Of-Date" Frames
  - Euler Frames

and two special cases for any of these families...

  - Frozen Dynamic Frames
  - Inertial Dynamic Frames
- Backup

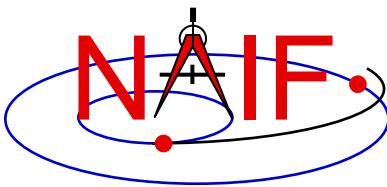


# What are Dynamic Frames

---

Navigation and Ancillary Information Facility

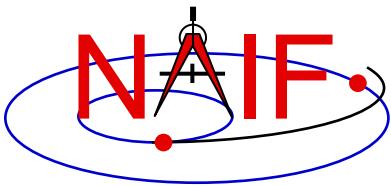
- **Dynamic reference frames ("dynamic frames" for short) have time-dependent orientation.**
  - CK- and PCK-based frames are not considered to be dynamic frames although they are time-varying.
  - Some examples are given on the next page.
- **Dynamic frames are specified using a frames kernel (FK).**
- **The dynamic frames capability is an extension to the original SPICE frames subsystem.**
- **This capability enables SPICE users to conveniently use a wide variety of frames that are not "built in" to SPICE.**



# Examples of Dynamic Frames

Navigation and Ancillary Information Facility

- **Geocentric Solar Ecliptic (GSE)**
- **Solar Magnetic (SM)**
- **Spacecraft-centered roll-celestial frame**
- **Geocentric Solar Magnetospheric (GSM)**
- **Geomagnetic (MAG)**
  - Using constant north centered geomagnetic dipole
  - Using dipole direction defined by time-dependent Euler angles
- **Geocentric Solar Equatorial (GSEQ)**
- **Solar Equatorial frame for any ephemeris object**
- **Orbital frame for any ephemeris object**
- **Earth mean equator and equinox of date**
- **Earth true equator and equinox of date**
- **Earth mean ecliptic and equinox of date**
- **Nadir-oriented frame for planetary orbiter**
- **Radial, tangential, normal (RTN)**

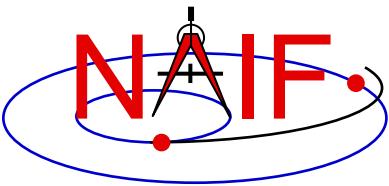


# Rationale

---

Navigation and Ancillary Information Facility

- **Why should SPICE offer dynamic frames implemented using a frames kernel? Instead...**
  - a user could build a C-kernel for any frame.
  - SPICE could provide a limited number of "built-in" dynamic frames which wouldn't require a frames kernel.
  - users can create their own routines to implement dynamic frames.
- **Benefits of using a SPICE FK for this purpose**
  - Convenience: using a formula rather than a C-kernel avoids C-kernel creation, dissemination, storage, and consistency issues.
  - Flexibility: the dynamic frame mechanism enables creation of a vast variety of reference frames.
  - Integration: once defined, and once supporting kernels are loaded, dynamic frames may be referenced in SPICE API calls.
  - Correct implementation: extensive testing done by NAIF, and perhaps further verified by a user, seems less likely to result in an error.

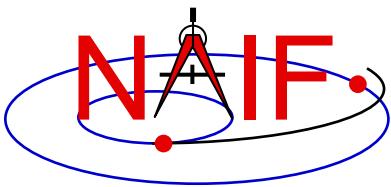


# Using a Dynamic Frame

---

Navigation and Ancillary Information Facility

- **Using a dynamic frame in a SPICE-based program is straightforward.**
  - At program initialization:
    - » Load the needed dynamic frame kernel to make the frame definition known to SPICE.
      - Note: a single dynamic frames kernel likely contains many dynamic frame specifications.
    - » Load any kernels on which the dynamic frame depends.
      - Some dynamic frames require the use of SPK, FK, PCK, CK, or SCLK SPICE kernels.
  - Then, in calls to SPICE routines, refer to the dynamic frame by name just as you would do with other kinds of frame names, such as "J2000."



# Examples of Using Dynamic Frames

Navigation and Ancillary Information Facility

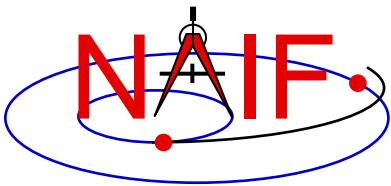
- Find the 6x6 matrix to transform states from the J2000 frame to the Geocentric Solar Ecliptic (GSE) frame at the TDB epoch given by ET1.

```
CALL SXFORM( 'J2000' , 'GSE' , ET1 , XFORM )
```

- Look up the state of Jupiter relative to the Earth in the GSE frame:

```
CALL SPKEZR( 'JUPITER' , ET1 , 'GSE' ,
              'NONE' , 'EARTH' , STATE , LT )
```

- You can use dynamic frames any place in SPICE where a frame name is used.
  - However, some restrictions apply to use of dynamic frames in SPICE kernels (see “Limitations” in the backup slides).

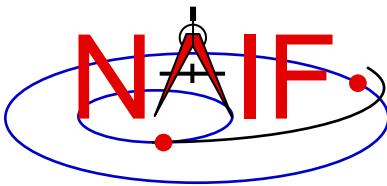


# Terminology - 1

---

Navigation and Ancillary Information Facility

- **"Frame"** is an abbreviation for **"reference frame."**
- A frame can be defined as a set of three mutually orthogonal, unit-length vectors.
  - These vectors are called **"basis vectors."** The lines containing the basis vectors are the **"axes"** of the frame.
  - The basis vectors indicate the **"positive"** axis directions; we label these vectors **+X**, **+Y**, and **+Z**. The negatives of these vectors are labeled **-X**, **-Y**, and **-Z**.
  - We number the axes as follows:  
 $X = \text{axis 1}; Y = \text{axis 2}; Z = \text{axis 3}$
- All of the frames we'll deal with are **"right-handed"**: this means **+Z** is the cross product  **$+X \times +Y$** .
- A reference frame's orientation is defined relative to another specified frame: the **"base frame."**

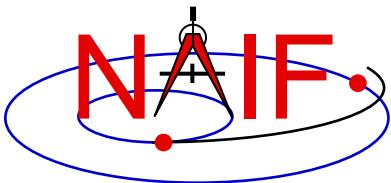


# Terminology - 2

---

Navigation and Ancillary Information Facility

- When we say that a frame is "time-dependent" or "time-varying," we mean:
  - The orientation of the frame is time-dependent.
  - Equivalently, the rotation between the frame and its base frame is time-dependent.
- By "evaluating" a frame or "evaluating the orientation of a frame," we mean computing the rotation between the frame and its base frame.
  - Because of their time-dependent nature, an epoch is required in order to evaluate a dynamic frame.
- In the SPICE system, frames are considered to have "centers."
  - The center of a frame is always an ephemeris object, something whose location can be specified with an SPK file.
  - Frame centers come into play when light time corrections are used: the apparent orientation of a time-dependent frame as seen by an observer is affected by the one-way light time between the frame's center and the observer.

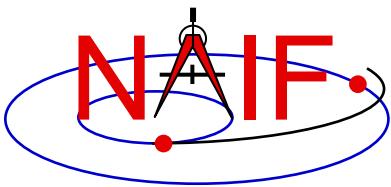


# Terminology - 3

---

Navigation and Ancillary Information Facility

- When we say that a vector is "aligned" with another vector, we mean that the angular separation between the two vectors is zero.
- We use the terms "defining a frame" and "specifying a frame" interchangeably. Both refer to creating a frame definition in a frames kernel (FK).
- The notation  $[\theta]_n$  indicates a frame rotation of theta radians (unless other units are specified) about axis n, where n is one of {1, 2, 3}. This transformation rotates vectors by –theta radians about axis n.



# Creating a Frames Kernel

## Defining Dynamic Frames

---

Navigation and Ancillary Information Facility

- **To define dynamic frames using a frames kernel, a fairly detailed understanding of the SPICE dynamic frames capability is required.**
- **A good understanding of the basic SPICE system (in particular, the SPK and frames subsystems) is also a prerequisite.**
- **See the Frames Required Reading document for the most detailed documentation available.**
- **The rest of this tutorial is concerned with:**
  - explaining the SPICE dynamic frames capability
  - showing how to create dynamic frames kernels
    - » We present many dynamic frame definition examples

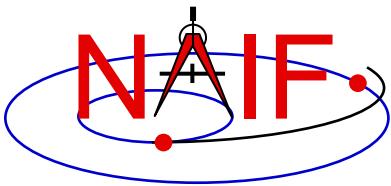


# Types of Dynamic Frames

---

Navigation and Ancillary Information Facility

- NAIF envisages providing two types of dynamic frames:
  - parameterized
  - scripted
- As of now only the parametrized type is implemented within SPICE; the rest of this tutorial focuses on this type.

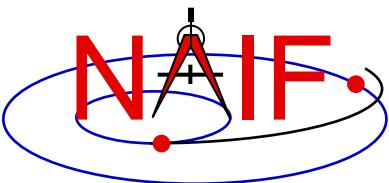


# Parameterized Dynamic Frames

---

Navigation and Ancillary Information Facility

- **Parameterized dynamic frames are defined using formulas.**
  - The code implementing the formulas is built into SPICE.
  - The parameters used in the formulas are specified in a frames kernel.
- **Parameterized dynamic frames are grouped into frame "families". Each family corresponds to a distinct, parameterized geometric formula providing a frame definition. The families are:**
  - Two-Vector frames
  - Of-date frames
  - Euler frames

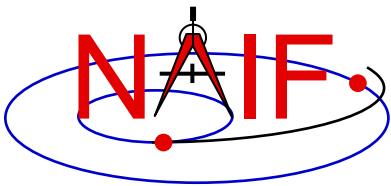


# Defining Parameterized Dynamic Frames

Navigation and Ancillary Information Facility

- **Parameterized dynamic frames are defined using "keyword=value" assignments in a frames kernel (FK).**
- **The following items must be specified in the frame definition:**
  - Frame name
  - Frame ID code
    - » The range 1400000-2000000 is reserved for people outside of the NAIF group
  - Class (= 5 for dynamic frames)
  - Class ID code (= frame ID code for dynamic frames)
  - Frame center (= name or NAIF ID code for central body)
  - Frame definition style (= 'PARAMETERIZED')
  - Base frame name ( called “relative”)
    - » The frame definition specifies a rotation from the dynamic frame to the base (relative) frame.
  - Frame family
  - Family-specific assignments

[continued on next page](#)



# Defining Parameterized Dynamic Frames

---

Navigation and Ancillary Information Facility

[continued from previous page](#)

- Rotation state
  - » Possible states are '**ROTATING**' and '**INERTIAL**'.
    - A frame is treated as rotating or inertial for the purpose of velocity transformations.
  - » The default dynamic frame rotation state is '**ROTATING**'.
    - For rotating two-vector and Euler frames, the rotation state assignment can be omitted from the frame definition.
    - For "of-date" frames, the frame definition must either specify the rotation state or designate the frame as "frozen" at a specified epoch (but not both).
- Freeze epoch
  - » The presence of this optional assignment in a frames kernel indicates that the frame orientation, relative to the base frame, is held constant ("frozen") at the specified epoch.
  - » Most dynamic frames are not frozen.

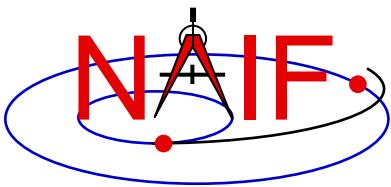


# What's Next?

---

Navigation and Ancillary Information Facility

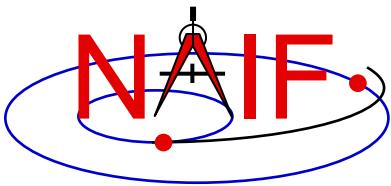
- In the next approximately 35 pages we'll provide implementation details and examples of frames belonging to each of the parameterized dynamic frame families:
  - Two-Vector frames
  - "Of-Date" frames
  - Euler frames
- We'll also explain the optional frame attributes which can be assigned to frames belonging to the above families:
  - “Frozen”
  - “Inertial”



# Two-Vector Frame Concepts - 1

Navigation and Ancillary Information Facility

- **Two-vector frames are defined using two time-dependent vectors: the "primary" and "secondary" vectors.**
  - Each vector may be defined by a variety of geometric means:
    - » Position vector
    - » Target near point vector
    - » Velocity vector
    - » Constant vector
- **The user associates specified positive or negative axes of the two-vector frame with the primary and secondary vectors.**
  - Two-vector frames are always right-handed and have orthogonal axes, so two non-parallel vectors and associations of axes with these vectors suffice to define the orientation of a frame.

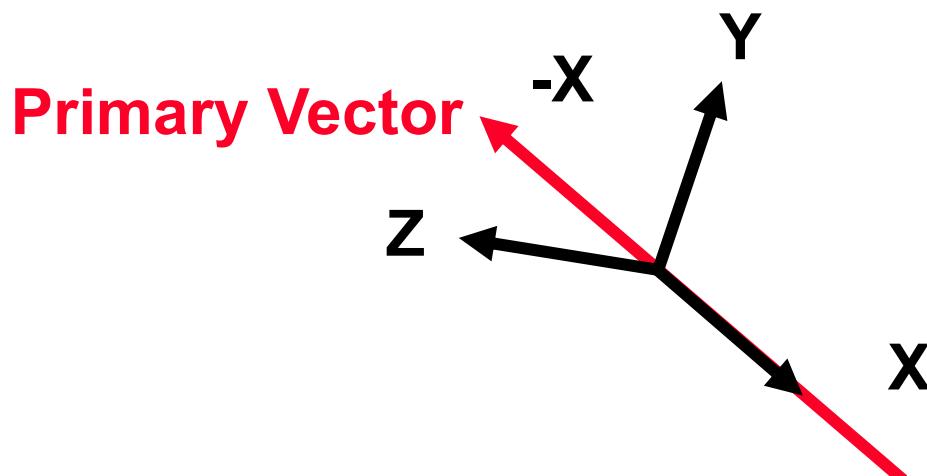


# Two-Vector Frame Concepts - 2

Navigation and Ancillary Information Facility

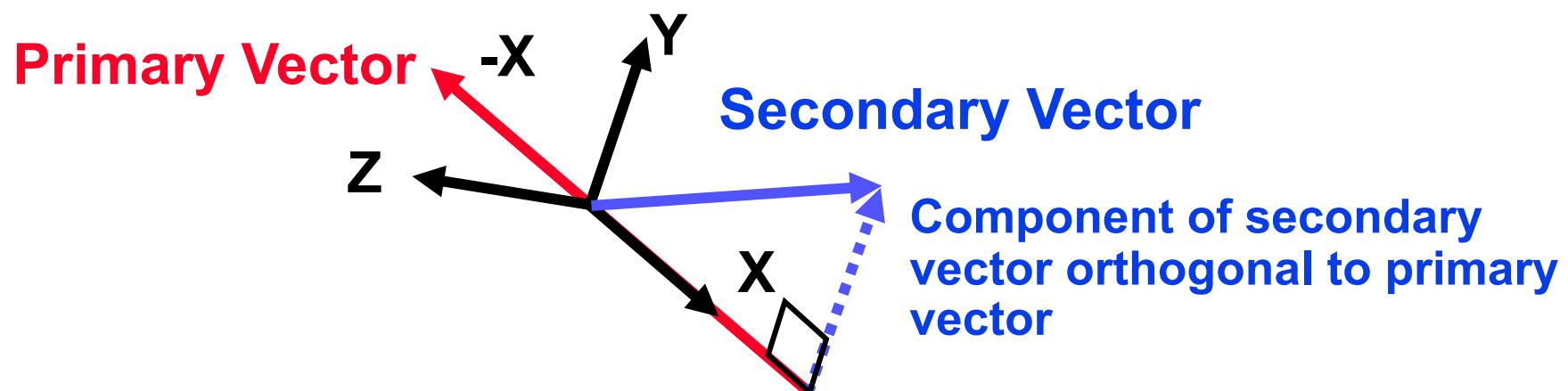
- **Primary Vector**

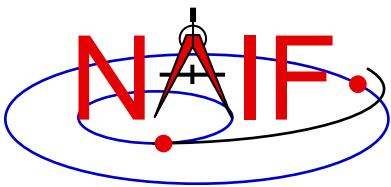
- A specified positive or negative axis of the two-vector frame is aligned with this vector.
  - » The frame kernel creator assigns to this vector one of the axis designations { +X, -X, +Y, -Y, +Z, -Z }.
- Two degrees of freedom of the frame orientation are removed by association of an axis with the primary vector. The third degree of freedom is the frame's rotation about the primary vector.
- Example: a frame's -X axis is aligned with the primary vector:



- **Secondary Vector**

- A specified positive or negative axis of the two-vector frame is aligned with the component of the secondary vector orthogonal to the primary vector.
  - » The frame kernel creator associates with this vector one of the axis designations { +X, -X, +Y, -Y, +Z, -Z }, where the axis is orthogonal to that associated with the primary vector.
- Example, continued: the frame's +Y axis is associated with the secondary vector. The component of the secondary vector orthogonal to the primary vector is aligned with the frame's +Y axis. The secondary vector thus lies in the frame's X-Y plane.





# Two-Vector Frame Concepts - 4

Navigation and Ancillary Information Facility

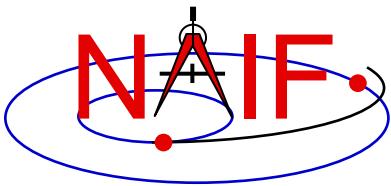
## • Secondary Vector, continued

- Typically the secondary vector itself is not orthogonal to the primary vector.
- However, the secondary vector must be linearly independent of the primary vector.
  - » Near-degenerate geometry can lead to extreme loss of precision.
    - This problem can be difficult to diagnose.
  - » SPICE enforces independence using a default angular separation tolerance of 1 milliradian. The angular separation of the primary and secondary vectors may not differ from 0 or Pi radians by less than this tolerance.
  - » A frame kernel creator can specify a different tolerance value.  
The frame kernel assignment for this is:

```
FRAME <frame_ID> ANGLE_SEP_TOL = <tolerance>
```

where the tolerance is given in radians.

- Designers of two-vector frames should ensure that the primary and secondary vectors can't become nearly parallel for any realistic evaluation epoch.

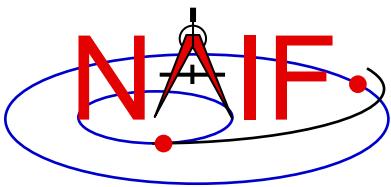


# Two-Vector Frame Concepts - 5

---

Navigation and Ancillary Information Facility

- In the next several pages we'll describe methods for specifying the primary and secondary vectors used in creating a dynamic frame.

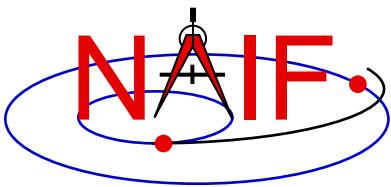


# Two-Vector Frame Concepts - 6

Navigation and Ancillary Information Facility

## • Using a Position Vector

- Defined by the position of one ephemeris object relative to another. The frame kernel creator specifies:
  - » the target
  - » the observer
  - » the aberration correction
    - The vector may optionally be corrected for light time and stellar aberration.
- The epoch at which the position vector is computed is supplied via a call to a SPICE API:
  - » as an input to an SPK routine, e.g. SPKEZR, SPKPOS.
  - » as an input to a frame system routine, e.g. SXFORM, PXFORM.
- The reference frame relative to which the vector is expressed is not specified by the frame kernel creator.
  - » SPICE automatically selects this frame.



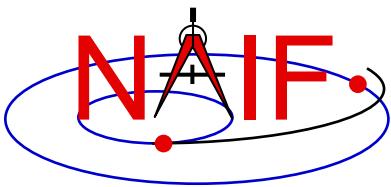
# Two-Vector Frame Concepts - 7

---

Navigation and Ancillary Information Facility

- **Using a Target Near Point Vector**

- Defined as the vector from an observer to the nearest point on a specified extended target body to that observer. The frame kernel creator specifies:
  - » the target
  - » the observer
  - » the aberration correction
    - The vector may optionally be corrected for one-way light time and stellar aberration.
    - When one-way light time correction is used, both the position and orientation of the target body are corrected for light time.
- The extended target body is modeled as a triaxial ellipsoid.
  - » Size and shape data are given by a PCK.
- The epoch is supplied via a SPICE API call.
- The reference frame relative to which the vector is expressed is not specified by the frame kernel creator.
  - » SPICE automatically selects this frame.

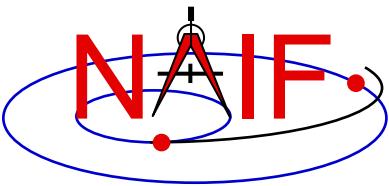


# Two-Vector Frame Concepts - 8

Navigation and Ancillary Information Facility

## • Using a Velocity Vector

- Defined by the velocity of a target ephemeris object relative to an observing ephemeris object. The frame kernel creator specifies:
  - » the target
  - » the observer
  - » the velocity reference frame
    - This frame may be distinct from the base frame.
    - Different velocity frame choices can lead to radically different two-vector frame definitions.
  - » the aberration correction
    - The velocity vector may optionally be corrected for one-way light time and stellar aberration.
    - Use of light time correction also implies evaluation of the velocity vector's frame at a light time corrected epoch: the epoch is corrected for light time between the velocity frame's center and the observer, if the velocity frame is non-inertial.
- The epoch is supplied via a SPICE API call.



# Two-Vector Frame Concepts - 9

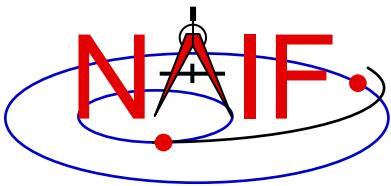
---

Navigation and Ancillary Information Facility

## • Using a Constant Vector

- The vector is constant in a frame specified by the kernel creator.
  - » The constant vector's frame may be time-dependent.
  - » This frame may be distinct from the base frame.
- The vector may be specified in a variety of coordinate systems.
  - » Cartesian
  - » Latitudinal
  - » Right ascension/declination (RA/DEC)
- An observer may optionally be associated with a constant vector for the purpose of defining aberration corrections.
  - » The orientation of the constant vector's frame may optionally be corrected for one-way light time between the frame's center and the observer: if the frame is non-inertial, it is evaluated at a light time corrected epoch.

[continued on next page](#)



# Two-Vector Frame Concepts - 10

---

Navigation and Ancillary Information Facility

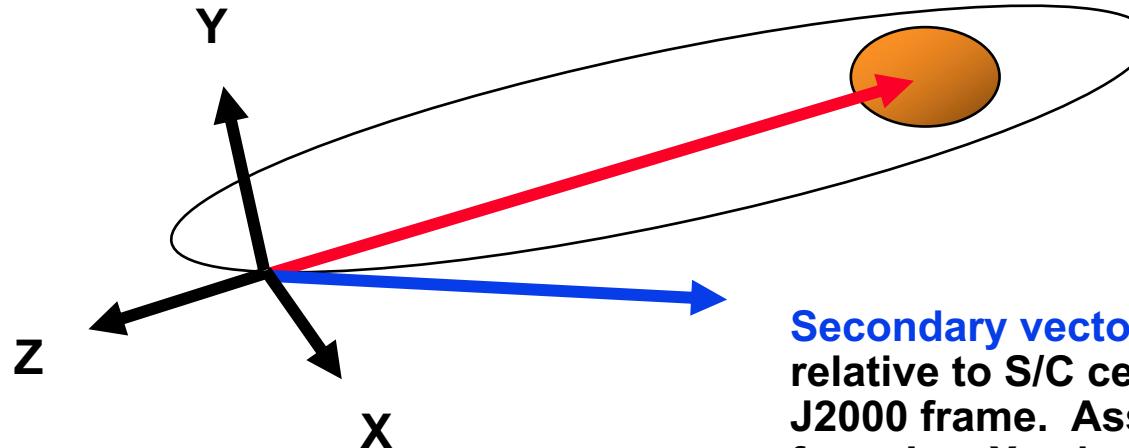
[continued from previous page](#)

- » **A constant vector may optionally be corrected for stellar aberration due to motion of observer relative to solar system barycenter.**
  - Stellar aberration can be specified without light time correction; the string indicating stellar aberration correction alone is 'S'
- The epoch is supplied via a SPICE API call, as for position vectors.
  - » If the constant vector's frame is time-dependent, that frame is evaluated at this epoch, optionally adjusted for light time.

## Nadir-Oriented Spacecraft-Centered Frame

$Y = Z \times X$ , completing the right-handed frame.

**Primary vector:** spacecraft nadir direction vector. Aligned with nadir frame's -Z axis.



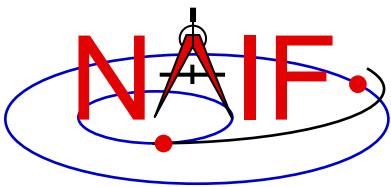
Nadir vector can be defined to point to either:

- closest point to spacecraft on ellipsoid
- center of mass of orbited body

**Secondary vector:** spacecraft velocity relative to S/C center of motion in the J2000 frame. Associated with nadir frame's +X axis.

Normalized component of secondary vector orthogonal to primary vector is aligned with the nadir frame's +X axis.

See the next page for the FK specifications for this frame.



# Two-Vector Frame Examples - 2

## Navigation and Ancillary Information Facility

Nadir-Oriented Spacecraft-Centered Frame: Frame kernel specification.

The -Z axis points from the spacecraft toward the closest point on Mars.

The component of the inertially referenced spacecraft velocity vector orthogonal to Z is aligned with the +X axis.

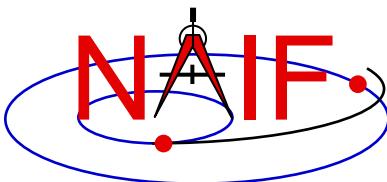
The +Y axis is the cross product of the +Z axis and the +X axis.

```
\begindata
```

```
FRAME_<frame_name> = <frame_ID>
FRAME_<frame_ID>_NAME = <frame_name>
FRAME_<frame_ID>_CLASS = 5
FRAME_<frame_ID>_CLASS_ID = <frame_ID>
FRAME_<frame_ID>_CENTER = <orbiter_ID>
FRAME_<frame_ID>_RELATIVE = 'J2000'
FRAME_<frame_ID>_DEF_STYLE = 'PARAMETERIZED'
FRAME_<frame_ID>_FAMILY = 'TWO-VECTOR'
FRAME_<frame_ID>_PRI_AXIS = '-Z'
FRAME_<frame_ID>_PRI_VECTOR_DEF = 'TARGET_NEAR_POINT'
FRAME_<frame_ID>_PRI_OBSERVER = <orbiter_ID/name>
FRAME_<frame_ID>_PRI_TARGET = 'MARS'
FRAME_<frame_ID>_PRI_ABCORR = 'NONE'
FRAME_<frame_ID>_SEC_AXIS = 'X'
FRAME_<frame_ID>_SEC_VECTOR_DEF = 'OBSERVER_TARGET_VELOCITY'
FRAME_<frame_ID>_SEC_OBSERVER = 'MARS'
FRAME_<frame_ID>_SEC_TARGET = <orbiter_ID/name>
FRAME_<frame_ID>_SEC_ABCORR = 'NONE'
FRAME_<frame_ID>_SEC_FRAME = 'J2000'
```

### Definitions

<frame_ID>	= integer frame ID code
<frame_name>	= user-specified frame name
<orbiter_ID>	= NAIF ID code of spacecraft
<orbiter_ID/name>	= NAIF ID code or name of spacecraft

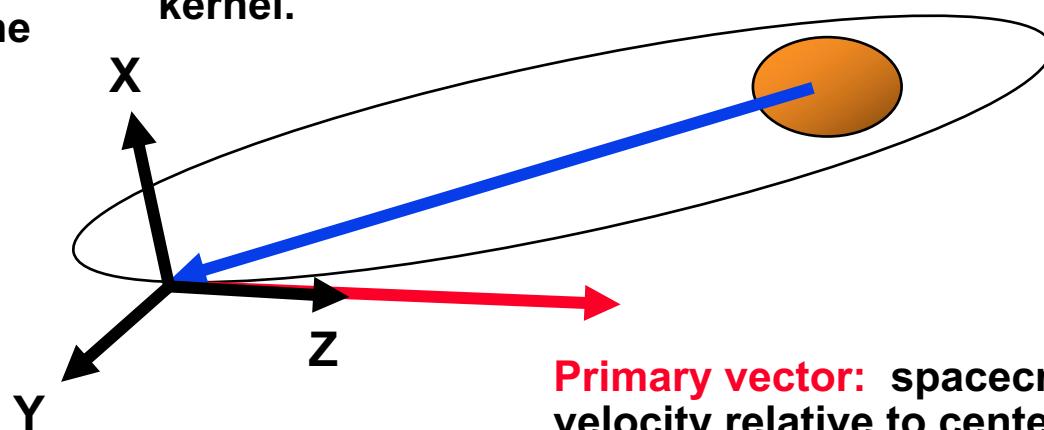


# Two-Vector Frame Examples - 3

Navigation and Ancillary Information Facility

## Spacecraft "View Frame"

$X = Y \times Z$ , completing the right-handed frame.  
("Out of plane" direction)



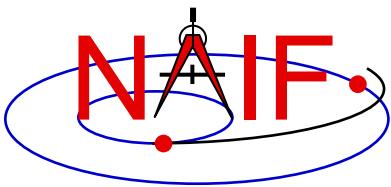
Normalized component of secondary vector orthogonal to primary vector is aligned with the view frame's +Y axis. ("In plane" direction)

### Secondary vector:

spacecraft position relative to center of motion.  
Associated with view frame's +Y axis in frame kernel.

**Primary vector:** spacecraft velocity relative to center of motion in J2000 frame. Aligned with view frame's +Z axis in frame kernel. ("Down track" direction)

See the next page for the FK specifications for this frame.



# Two-Vector Frame Examples - 4

## Navigation and Ancillary Information Facility

Spacecraft "View Frame": Frame kernel specification.

The +Z axis is aligned with the J2000-referenced velocity of the spacecraft relative to Mars.

The component of the spacecraft position orthogonal to +Z is aligned with the +Y axis.

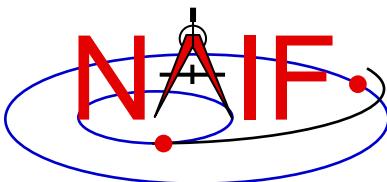
The +X axis is the cross product of the +Y axis and the +Z axis.

```
\begindata
```

```
FRAME_<frame_name> = <frame_ID>
FRAME_<frame_ID>_NAME = <frame_name>
FRAME_<frame_ID>_CLASS = 5
FRAME_<frame_ID>_CLASS_ID = <frame_ID>
FRAME_<frame_ID>_CENTER = <orbiter_ID>
FRAME_<frame_ID>_RELATIVE = 'J2000'
FRAME_<frame_ID>_DEF_STYLE = 'PARAMETERIZED'
FRAME_<frame_ID>_FAMILY = 'TWO-VECTOR'
FRAME_<frame_ID>_PRI_AXIS = 'Z'
FRAME_<frame_ID>_PRI_VECTOR_DEF = 'OBSERVER_TARGET_VELOCITY'
FRAME_<frame_ID>_PRI_OBSERVER = 'MARS'
FRAME_<frame_ID>_PRI_TARGET = <orbiter_ID/name>
FRAME_<frame_ID>_PRI_ABCORR = 'NONE'
FRAME_<frame_ID>_PRI_FRAME = 'J2000'
FRAME_<frame_ID>_SEC_AXIS = 'Y'
FRAME_<frame_ID>_SEC_VECTOR_DEF = 'OBSERVER_TARGET_POSITION'
FRAME_<frame_ID>_SEC_OBSERVER = 'MARS'
FRAME_<frame_ID>_SEC_TARGET = <orbiter_ID/name>
FRAME_<frame_ID>_SEC_ABCORR = 'NONE'
```

### Definitions

<frame_ID>	= integer frame ID code
<frame_name>	= user-specified frame name
<orbiter_ID>	= NAIF ID code of spacecraft
<orbiter_ID/name>	= NAIF ID code or name of spacecraft

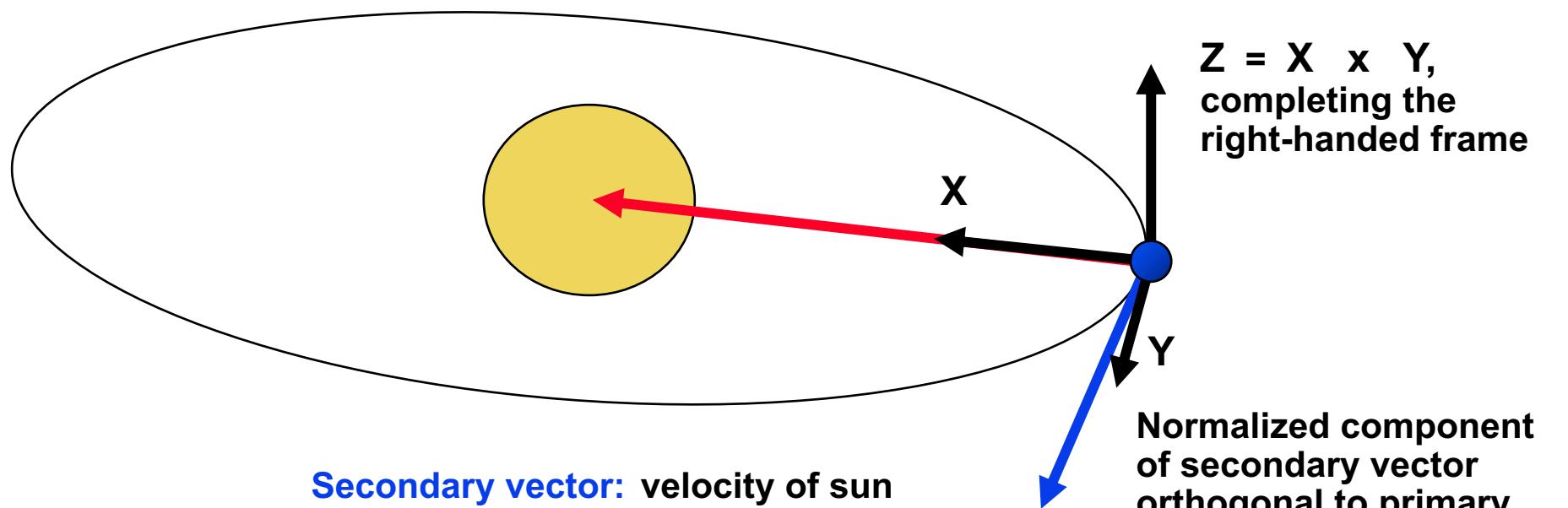


# Two-Vector Frame Examples - 5

Navigation and Ancillary Information Facility

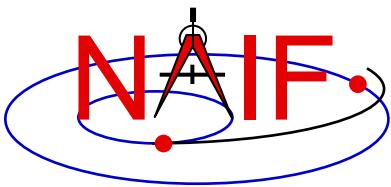
## Geocentric Solar Ecliptic Frame (GSE)

**Primary vector:** position of sun relative to earth.  
Aligned with GSE frame's +X axis.



**Secondary vector:** velocity of sun  
relative to earth in J2000 frame.  
Associated with GSE frame's +Y axis in  
frame kernel.

See the next page for the FK specifications for this frame.



# Two-Vector Frame Examples - 6

## Navigation and Ancillary Information Facility

Geocentric Solar Ecliptic (GSE) frame:

+X is parallel to the geometric earth-sun position vector.

+Y axis is the normalized component of the geometric earth-sun velocity vector orthogonal to the GSE +X axis.

+Z axis is parallel to the cross product of the GSE +X axis and the GSE +Y axis.

\begindata

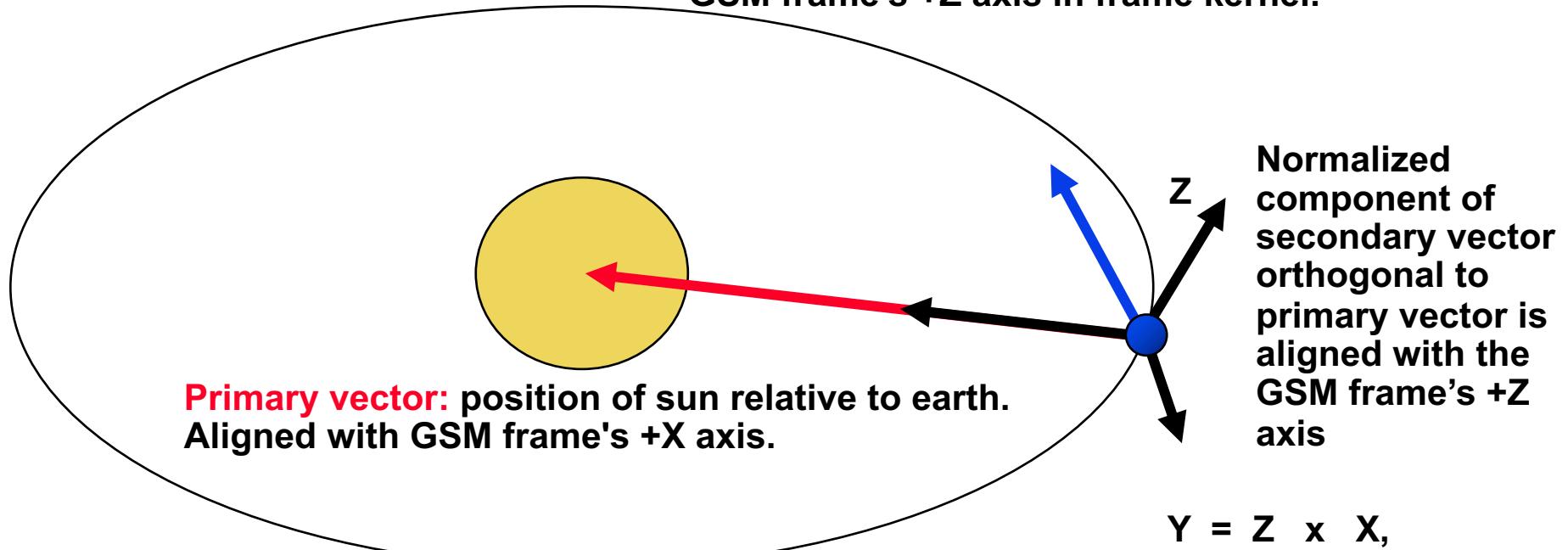
FRAME_GSE	= <frame_ID>
FRAME_<frame_ID>_NAME	= 'GSE'
FRAME_<frame_ID>_CLASS	= 5
FRAME_<frame_ID>_CLASS_ID	= <frame_ID>
FRAME_<frame_ID>_CENTER	= 399
FRAME_<frame_ID>_RELATIVE	= 'J2000'
FRAME_<frame_ID>_DEF_STYLE	= 'PARAMETERIZED'
FRAME_<frame_ID>_FAMILY	= 'TWO-VECTOR'
FRAME_<frame_ID>_PRI_AXIS	= 'X'
FRAME_<frame_ID>_PRI_VECTOR_DEF	= 'OBSERVER_TARGET_POSITION'
FRAME_<frame_ID>_PRI_OBSERVER	= 'EARTH'
FRAME_<frame_ID>_PRI_TARGET	= 'SUN'
FRAME_<frame_ID>_PRI_ABCORR	= 'NONE'
FRAME_<frame_ID>_SEC_AXIS	= 'Y'
FRAME_<frame_ID>_SEC_VECTOR_DEF	= 'OBSERVER_TARGET_VELOCITY'
FRAME_<frame_ID>_SEC_OBSERVER	= 'EARTH'
FRAME_<frame_ID>_SEC_TARGET	= 'SUN'
FRAME_<frame_ID>_SEC_ABCORR	= 'NONE'
FRAME_<frame_ID>_SEC_FRAME	= 'J2000'

### Definition

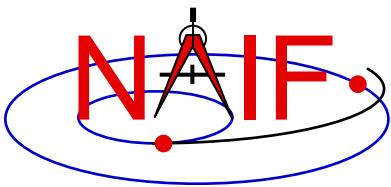
<frame\_ID> = integer frame ID code

## Geocentric Solar Magnetospheric Frame (GSM)

**Secondary vector:** North geomagnetic centered dipole in IAU\_EARTH frame. Associated with GSM frame's +Z axis in frame kernel.



See the next page for the FK specifications for this frame.



# Two-Vector Frame Examples - 8

## Navigation and Ancillary Information Facility

Geocentric Solar Magnetospheric (GSM) frame:

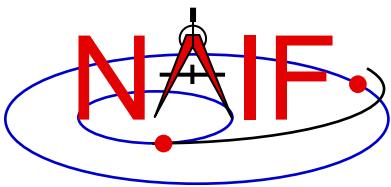
- +X is parallel to the geometric earth-sun position vector.
- +Z axis is normalized component of north centered geomagnetic dipole vector orthogonal to GSM +X axis.
- +Y completes the right-handed frame.

\begindata

```
FRAME_GSM                      = <frame_ID>
FRAME_<frame_ID>_NAME          = 'GSM'
FRAME_<frame_ID>_CLASS          = 5
FRAME_<frame_ID>_CLASS_ID       = <frame_ID>
FRAME_<frame_ID>_CENTER          = 399
FRAME_<frame_ID>_RELATIVE        = 'J2000'
FRAME_<frame_ID>_DEF_STYLE       = 'PARAMETERIZED'
FRAME_<frame_ID>_FAMILY          = 'TWO-VECTOR'
FRAME_<frame_ID>_PRI_AXIS         = 'X'
FRAME_<frame_ID>_PRI_VECTOR_DEF  = 'OBSERVER_TARGET_POSITION'
FRAME_<frame_ID>_PRI_OBSERVER    = 'EARTH'
FRAME_<frame_ID>_PRI_TARGET        = 'SUN'
FRAME_<frame_ID>_PRI_ABCORR      = 'NONE'
FRAME_<frame_ID>_SEC_AXIS          = 'Z'
FRAME_<frame_ID>_SEC_VECTOR_DEF   = 'CONSTANT'
FRAME_<frame_ID>_SEC_FRAME         = 'IAU_EARTH'
FRAME_<frame_ID>_SEC_SPEC          = 'LATITUDINAL'
FRAME_<frame_ID>_SEC_UNITS         = 'DEGREES'
FRAME_<frame_ID>_SEC_LONGITUDE     = 288.43
FRAME_<frame_ID>_SEC_LATITUDE      = 79.54
```

Definition

<frame\_ID> = integer frame ID code

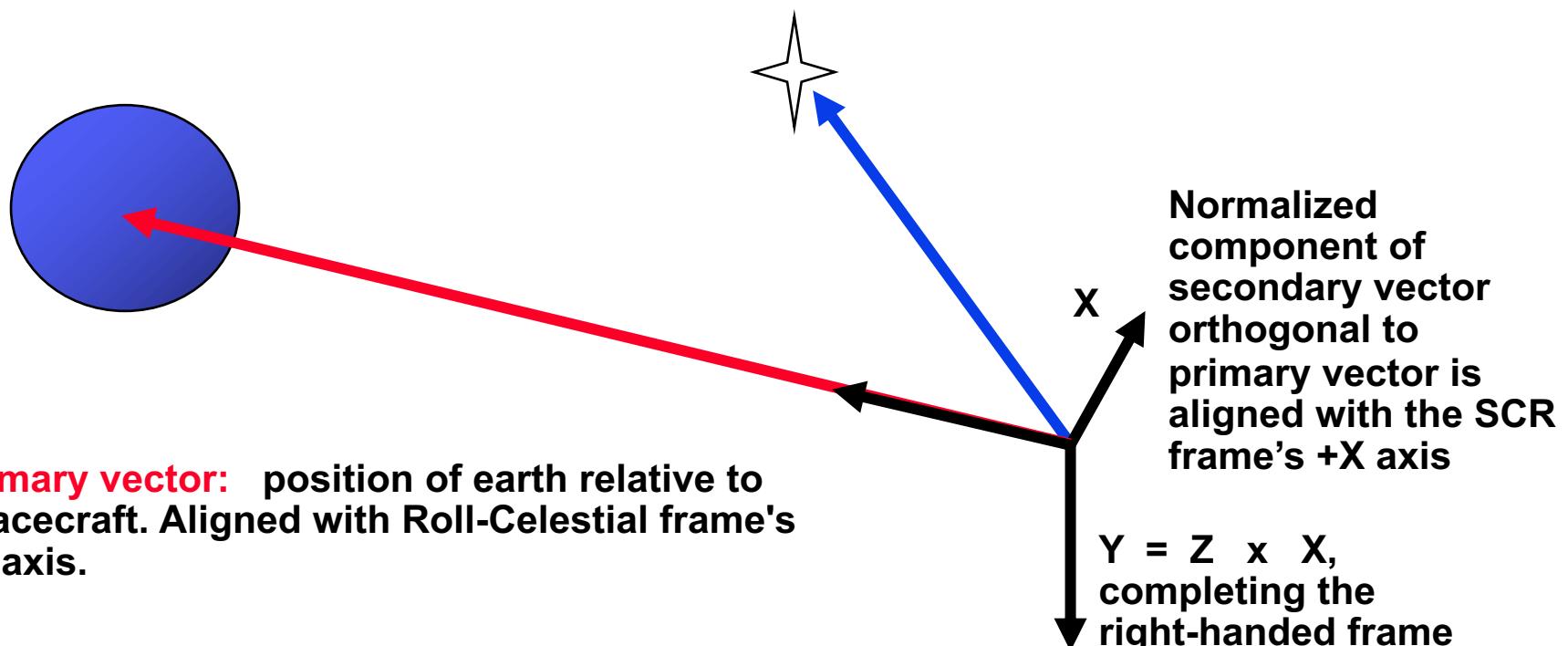


# Two-Vector Frame Examples - 9

Navigation and Ancillary Information Facility

## Spacecraft-Centered Roll-Celestial Frame

**Secondary vector:** Lock star direction in J2000 frame, corrected for stellar aberration due to spacecraft motion. Associated with Roll-Celestial frame's +X axis in frame kernel.

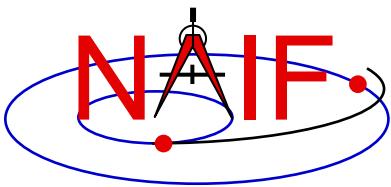


**Primary vector:** position of earth relative to spacecraft. Aligned with Roll-Celestial frame's +Z axis.

Normalized component of secondary vector orthogonal to primary vector is aligned with the SCR frame's +X axis

$Y = Z \times X$ ,  
completing the  
right-handed frame

See the next page for the FK specifications for this frame.



# Two-Vector Frame Examples - 10

## Navigation and Ancillary Information Facility

Spacecraft-centered roll-celestial frame:

+Z is parallel to the geometric earth-sun position vector.

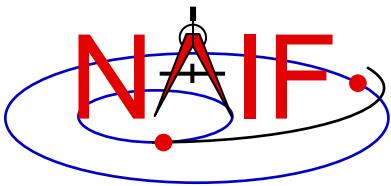
+X axis is normalized component of star direction orthogonal to Z axis. The star direction is corrected for stellar aberration due to motion of the spacecraft.

+Y completes the right-handed frame.

```
\begindata
  FRAME_<frame_name>          = <frame_ID>
  FRAME_<frame_ID>_NAME        = <frame_name>
  FRAME_<frame_ID>_CLASS       = 5
  FRAME_<frame_ID>_CLASS_ID    = <frame_ID>
  FRAME_<frame_ID>_CENTER      = <spacecraft_ID>
  FRAME_<frame_ID>_RELATIVE    = 'J2000'
  FRAME_<frame_ID>_DEF_STYLE   = 'PARAMETERIZED'
  FRAME_<frame_ID>_FAMILY      = 'TWO-VECTOR'
  FRAME_<frame_ID>_PRI_AXIS    = 'Z'
  FRAME_<frame_ID>_PRI_VECTOR_DEF = 'OBSERVER_TARGET_POSITION'
  FRAME_<frame_ID>_PRI_OBSERVER = <spacecraft_ID/name>
  FRAME_<frame_ID>_PRI_TARGET   = 'EARTH'
  FRAME_<frame_ID>_PRI_ABCORR   = 'NONE'
  FRAME_<frame_ID>_SEC_AXIS     = 'X'
  FRAME_<frame_ID>_SEC_VECTOR_DEF = 'CONSTANT'
  FRAME_<frame_ID>_SEC_FRAME    = 'J2000'
  FRAME_<frame_ID>_SEC_SPEC     = 'RA/DEC'
  FRAME_<frame_ID>_SEC_UNITS    = 'DEGREES'
  FRAME_<frame_ID>_SEC_RA        = <star right ascension in degrees>
  FRAME_<frame_ID>_SEC_DEC       = <star declination in degrees>
  FRAME_<frame_ID>_SEC_OBSERVER = <spacecraft_ID/name>
  FRAME_<frame_ID>_SEC_ABCORR   = 'S'
```

### Definitions

<frame_ID>	= integer frame ID code
<frame_name>	= user-specified frame name
<spacecraft_ID>	= NAIF ID code of spacecraft
<spacecraft_ID/name>	= NAIF ID code or name of spacecraft

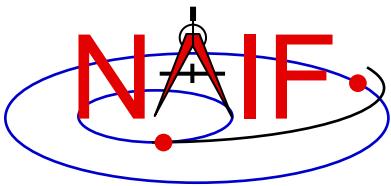


# "Of-Date" Frames - 1

---

Navigation and Ancillary Information Facility

- Of-date frames are associated with user-specified bodies and are based on user-selected dynamical models.
  - Implementations of these models are built into SPICE.
- The currently supported "of-date" frame families are
  - Mean Equator and Equinox of Date
  - True Equator and Equinox of Date
  - Mean Ecliptic and Equinox of Date
- Currently the Earth is the only supported body for of-date dynamic frames.

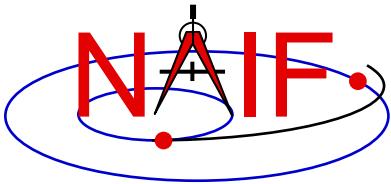


# "Of-Date" Frames - 2

---

Navigation and Ancillary Information Facility

- **The supported types of models are:**
  - Precession
  - Nutation
  - Mean obliquity
- **The of-date frame implementation is intended to be flexible...**
  - The set of supported bodies can grow over time.
  - The set of supported models can grow over time.
    - » SPICE is not forever locked into using a single hard-coded implementation, such as the 1976 IAU precession model.

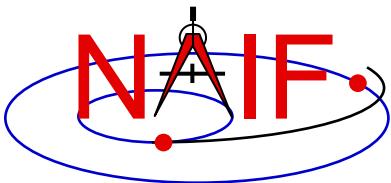


# "Of-Date" Frames - 3

---

Navigation and Ancillary Information Facility

- **Mean Equator and Equinox of Date Family**
  - For all reference frames in this family...
    - » The frame's relationship to the J2000 frame is given by a precession model.
    - » The frame kernel creator selects a precession model from those built into the SPICE software.
      - Currently supported only for the Earth, and only the 1976 IAU precession model (Lieske model)
    - » The frame kernel creator must either specify the frame's rotation state or must designate the frame "frozen" at a specified "freeze epoch" (but not both).



# "Of-Date" Frames - 4

## Navigation and Ancillary Information Facility

Earth mean equator and equinox of date frame:

+Z axis is perpendicular to the mean equator of date **and points north**.

+X axis is parallel to the cross product of the +Z axis and  
the north-pointing vector normal to the mean ecliptic of date.

+Y axis completes the right-handed frame.

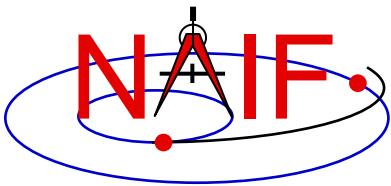
```
\begindata  
  
FRAME_<frame_name>  
FRAME_<frame_ID>_NAME  
FRAME_<frame_ID>_CLASS  
FRAME_<frame_ID>_CLASS_ID  
FRAME_<frame_ID>_CENTER  
FRAME_<frame_ID>_RELATIVE  
FRAME_<frame_ID>_DEF_STYLE  
FRAME_<frame_ID>_FAMILY  
FRAME_<frame_ID>_PREC_MODEL  
FRAME_<frame_ID>_ROTATION_STATE
```

```
= <frame_ID>  
= <frame_name>  
= 5  
= <frame_ID>  
= 399  
= 'J2000'  
= 'PARAMETERIZED'  
= 'MEAN_EQUIATOR_AND_EQUINOX_OF_DATE'  
= 'EARTH_IAU_1976'  
= 'ROTATING'
```

### Definitions

<frame_ID>	= integer frame ID code
<frame_name>	= user-specified frame name

This is currently the  
only allowed value for  
precession model.

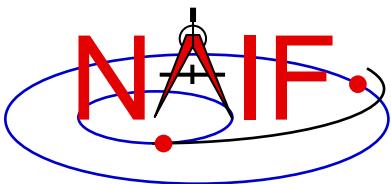


# "Of-Date" Frames - 5

---

Navigation and Ancillary Information Facility

- **True Equator and Equinox of Date Family**
  - For all reference frames in this family...
    - » The frame's relationship to the J2000 frame is given by a precession model and a nutation model.
    - » The frame kernel creator selects models from those built into the SPICE software.
      - Currently supported only for the Earth
      - Currently only the 1976 IAU precession model (Lieske model) is allowed.
      - Currently only the 1980 IAU nutation model is allowed.
    - » The frame kernel creator must either specify the frame's rotation state or must designate the frame "frozen" at a specified "freeze epoch" (but not both).



# "Of-Date" Frames - 6

## Navigation and Ancillary Information Facility

Earth true equator and equinox of date frame:

+Z axis is perpendicular to the true equator of date **and points north**.

+X axis is parallel to the cross product of the +Z axis and  
the north-pointing vector normal to the mean ecliptic of date.

+Y axis completes the right-handed frame.

```
\begindata  
  
FRAME_<frame_name>  
FRAME_<frame_ID>_NAME  
FRAME_<frame_ID>_CLASS  
FRAME_<frame_ID>_CLASS_ID  
FRAME_<frame_ID>_CENTER  
FRAME_<frame_ID>_RELATIVE  
FRAME_<frame_ID>_DEF_STYLE  
FRAME_<frame_ID>_FAMILY  
FRAME_<frame_ID>_PREC_MODEL  
FRAME_<frame_ID>_NUT_MODEL  
FRAME_<frame_ID>_ROTATION_STATE
```

```
= <frame_ID>  
= <frame_name>  
= 5  
= <frame_ID>  
= 399  
= 'J2000'  
= 'PARAMETERIZED'  
= 'TRUE_EQUATOR_AND_EQUINOX_OF_DATE'  
= 'EARTH_IAU_1976'  
= 'EARTH_IAU_1980'  
= 'ROTATING'
```

### Definitions

<frame_ID>	= integer frame ID code
<frame_name>	= user-specified frame name

These are currently the only allowed values for precession model and nutation model.

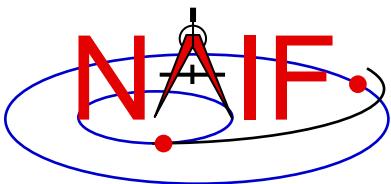


# "Of-Date" Frames - 7

---

Navigation and Ancillary Information Facility

- **Mean Ecliptic and Equinox of Date Family**
  - For all reference frames in this family...
    - » The frame's relationship to the J2000 frame is given by a precession model and an obliquity model.
    - » The frame kernel creator selects models from those built into the SPICE software.
      - Currently supported only for the Earth.
      - Currently only the 1976 IAU precession model (Lieske model) is allowed.
      - Currently only the 1980 IAU mean obliquity model is allowed.
    - » The frame kernel creator must either specify the frame's rotation state or must designate the frame "frozen" at a specified "freeze epoch."



# "Of-Date" Frames - 8

## Navigation and Ancillary Information Facility

Earth mean ecliptic and equinox of date frame:

+Z axis is perpendicular to mean ecliptic of date and points toward ecliptic north.

+X axis is parallel to the cross product of the north-pointing vector normal to mean equator of date and the +Z axis.

+Y axis completes the right-handed frame.

\begin{data}

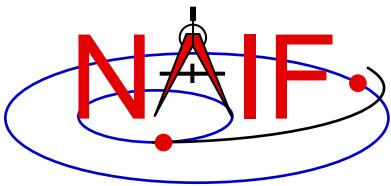
```
FRAME_<frame_name>
FRAME_<frame_ID>_NAME
FRAME_<frame_ID>_CLASS
FRAME_<frame_ID>_CLASS_ID
FRAME_<frame_ID>_CENTER
FRAME_<frame_ID>_RELATIVE
FRAME_<frame_ID>_DEF_STYLE
FRAME_<frame_ID>_FAMILY
FRAME_<frame_ID>_PREC_MODEL
FRAME_<frame_ID>_OBLIQ_MODEL
FRAME_<frame_ID>_ROTATION_STATE
```

```
= <frame_ID>
= <frame_name>
= 5
= <frame_ID>
= 399
= 'J2000'
= 'PARAMETERIZED'
= 'MEAN_ECLIPTIC_AND_EQUINOX_OF_DATE'
= 'EARTH_IAU_1976' ←
= 'EARTH_IAU_1980' ←
= 'ROTATING'
```

### Definitions

<frame_ID>	= integer frame ID code
<frame_name>	= user-specified frame name

These are currently the only allowed values for precession model and obliquity model.

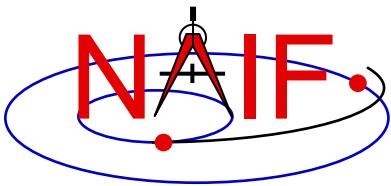


# Euler Frames - 1

---

Navigation and Ancillary Information Facility

- Euler frames are defined by a time-dependent rotation relative to a base frame.
  - The rotation from an Euler frame to its base frame is given by three Euler angles.
  - Each angle is given by a separate polynomial.
    - » The polynomials may have different degrees.
    - » The independent variable is a time offset, in TDB seconds, from an epoch specified by the frame kernel creator.
    - » The units associated with the angles are specified by the frame kernel creator.
    - » The sequence of rotation axes is specified by the frame kernel creator.
      - The central axis must differ from the other two.
      - The rotation from the Euler frame to the base frame is  $[angle\_1]_{axis\_1} [angle\_2]_{axis\_2} [angle\_3]_{axis\_3}$



# Euler Frames - 2

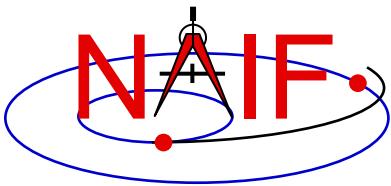
---

Navigation and Ancillary Information Facility

- **Five examples**

- Dynamic version of earth magnetospheric frame (MAG)
  - » Latitude and longitude of the north centered geomagnetic dipole are given by polynomials.
- Spinning spacecraft frame
  - » The base frame could be a:
    - Built-in inertial frame
    - C-kernel frame
    - Roll-celestial frame (using lock star)
    - Nadir frame
- Topocentric frames for tracking stations for which crustal plate motion is modeled
  - » The frame rotation keeps the frame orientation consistent with the changing station location.

[continued on next page](#)

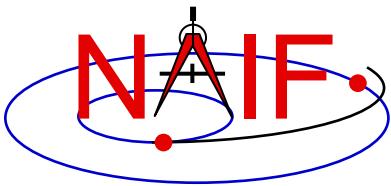


# Euler Frames - 3

---

Navigation and Ancillary Information Facility

- Mean or true body equator and body equinox of date frame, where the body is a planet or satellite other than the earth
  - » The base frame is an IAU\_<body> frame.
  - » The Euler frame "removes" the body's rotation about the spin axis.
- Variation on supported "of date" frame
  - » An existing supported "of date" frame is used as the base frame.
  - » Perturbations to the "of date" frame are expressed using Euler angles.



# Euler Frames - 4

## Navigation and Ancillary Information Facility

As an example, we construct an Euler frame called IAU\_MARS\_EULER. Frame IAU\_MARS\_EULER is mathematically identical to the PCK frame named IAU\_MARS. The PCK data defining the underlying IAU\_MARS frame are:

```
BODY499_POLE_RA = ( 317.68143 -0.1061 0.      )
BODY499_POLE_DEC = ( 52.88650 -0.0609 0.      )
BODY499_PM       = ( 176.630   350.89198226 0. )
```

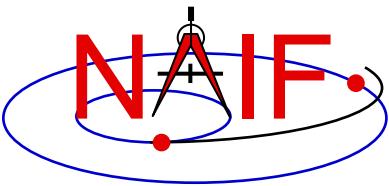
Relative to the angles used to define the IAU\_MARS frame, the angles for our Euler frame definition are reversed and the signs negated. Angular units are degrees. Rate units are degrees/second, unlike the PCK units of degrees/day.

PCK:	angle_3 is 90 + RA	Euler Frame:	angle_1 is -90 - RA
	angle_2 is 90 - Dec		angle_2 is -90 + Dec
	angle_1 is PM		angle_3 is - PM

```
\begindata
  FRAME_IAU_MARS_EULER           = <frame_ID>
  FRAME_<frame_ID>_NAME          = 'IAU_MARS_EULER'
  FRAME_<frame_ID>_CLASS         = 5
  FRAME_<frame_ID>_CLASS_ID      = <frame_ID>
  FRAME_<frame_ID>_CENTER        = 499
  FRAME_<frame_ID>_RELATIVE      = 'J2000'
  FRAME_<frame_ID>_DEF_STYLE     = 'PARAMETERIZED'
  FRAME_<frame_ID>_FAMILY        = 'EULER'
  FRAME_<frame_ID>_EPOCH         = @2000-JAN-1/12:00:00
  FRAME_<frame_ID>_AXES          = ( 3 1 3 )
  FRAME_<frame_ID>_UNITS         = 'DEGREES'
  FRAME_<frame_ID>_ANGLE_1_COEFFS = ( -47.68143 0.33621061170684714E-10 )
  FRAME_<frame_ID>_ANGLE_2_COEFFS = ( -37.1135 -0.19298045478743630E-10 )
  FRAME_<frame_ID>_ANGLE_3_COEFFS = ( -176.630 -0.40612497946759260E-02 )
```

Definition

<frame_ID>	= integer frame ID code
------------	-------------------------

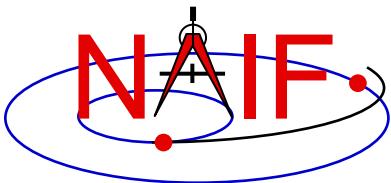


# Frozen Dynamic Frames - 1

---

Navigation and Ancillary Information Facility

- **A frozen dynamic frame is a "Snapshot" of a dynamic frame at a specified epoch.**
  - The frame is frozen relative to the base frame specified by the frame kernel creator in the frame kernel definition.
  - The rotation from the frozen frame to the base frame is constant.
  - The rotation is not frozen with respect to inertial frames unless the base frame is inertial.
  - A frame is designated frozen by the presence of a "freeze epoch" specification in the frame definition, for example:  
`FRAME_<FRAME_ID>_FREEZE_EPOCH = @1949-DEC-31/22:09:46.861901`
  - The freeze epoch is specified using SPICE text kernel rules.
    - » The "@" syntax is used.
    - » The time system is assumed to be TDB.



# Frozen Dynamic Frames - 2

## Navigation and Ancillary Information Facility

Frozen version of Earth mean equator and equinox of date frame:

+Z axis is perpendicular to the mean equator of date **and points north**.

+X axis is parallel to the cross product of the +Z axis and the vector normal to the mean ecliptic of date.

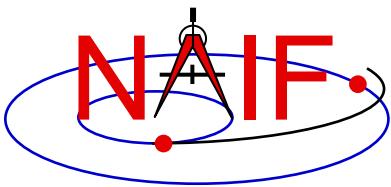
+Y axis completes the right-handed frame.

```
\begindata  
  
FRAME_<frame_name>  
FRAME_<frame_ID>_NAME  
FRAME_<frame_ID>_CLASS  
FRAME_<frame_ID>_CLASS_ID  
FRAME_<frame_ID>_CENTER  
FRAME_<frame_ID>_RELATIVE  
FRAME_<frame_ID>_DEF_STYLE  
FRAME_<frame_ID>_FAMILY  
FRAME_<frame_ID>_PREC_MODEL  
FRAME_<frame_ID>_FREEZE_EPOCH
```

```
= <frame_ID>  
= <frame_name>  
= 5  
= <frame_ID>  
= 399  
= 'J2000'  
= 'PARAMETERIZED'  
= 'MEAN_EQUATOR_AND_EQUINOX_OF_DATE'  
= 'EARTH_IAU_1976'  
= @1949-DEC-31/22:09:46.861901
```

### Definitions

<frame_ID>	= integer frame ID code
<frame_name>	= user-specified frame name



# Inertial Dynamic Frames - 1

Navigation and Ancillary Information Facility

- Inertial dynamic frames are specified by setting the rotation state to 'INERTIAL' in the rotation state assignment:

FRAME\_<FRAME\_ID>\_ROTATION\_STATE = 'INERTIAL'

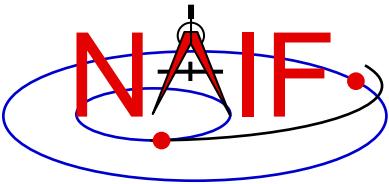
- The 'INERTIAL' state implies the frame is treated as inertial for the purpose of velocity transformations.
- The state transformation between any inertial frame and "inertial dynamic frame" has a zero derivative block: the state transformation matrix has the form

$$\begin{array}{c|c} R(t) & 0 \\ \hline \hline 0 & R(t) \end{array}$$

Derivative block

where  $R(t)$  is a time-dependent rotation.

[continued on next page](#)



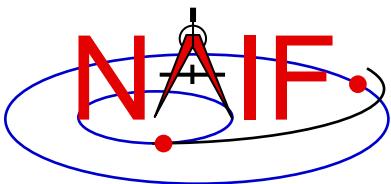
# Inertial Dynamic Frames - 2

Navigation and Ancillary Information Facility

- In contrast, for any rotating frame  $R(t)$ , the state transformation between any inertial frame and  $R(t)$  has a corresponding matrix of the form

$$\begin{array}{c|c} R(t) & | & 0 \\ \hline \hline dR(t)/dt & | & R(t) \end{array}$$

- The inertial rotation state:
  - » simplifies velocity transformations – velocities are transformed by a rotation.
  - » may be useful for maintaining consistency with other dynamic frame implementations.
  - » only makes sense if the "inertial" dynamic frame actually rotates very slowly!



# Inertial Dynamic Frames - 3

## Navigation and Ancillary Information Facility

Inertial version of Earth true equator and equinox of date frame:

+Z axis is perpendicular to the true equator of date **and points north**.

+X axis is parallel to the cross product of the +Z axis and the vector normal to the mean ecliptic of date.

+Y axis completes the right-handed frame.

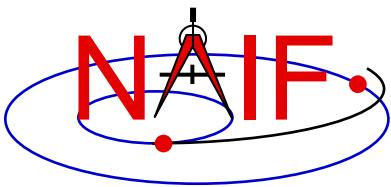
```
\begindata  
  
FRAME_<frame_name>  
FRAME_<frame_ID>_NAME  
FRAME_<frame_ID>_CLASS  
FRAME_<frame_ID>_CLASS_ID  
FRAME_<frame_ID>_CENTER  
FRAME_<frame_ID>_RELATIVE  
FRAME_<frame_ID>_DEF_STYLE  
FRAME_<frame_ID>_FAMILY  
FRAME_<frame_ID>_PREC_MODEL  
FRAME_<frame_ID>_NUT_MODEL  
FRAME_<frame_ID>_ROTATION_STATE
```

```
= <frame_ID>  
= <frame_name>  
= 5  
= <frame_ID>  
= 399  
= 'J2000'  
= 'PARAMETERIZED'  
= 'TRUE_EQUATOR_AND_EQUINOX_OF_DATE'  
= 'EARTH_IAU_1976'  
= 'EARTH_IAU_1980'  
= 'INERTIAL'
```

### Definitions

<frame_ID>	= integer frame ID code
<frame_name>	= user-specified frame name

These are currently the only allowed values for precession model and nutation model.

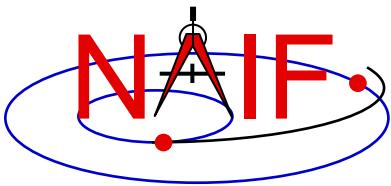


# Backup

---

Navigation and Ancillary Information Facility

- Numerical Issues
- Limitations

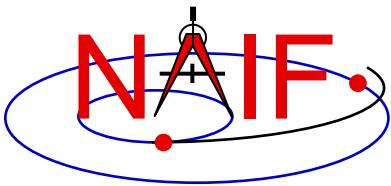


# Numerical Issues - 1

---

Navigation and Ancillary Information Facility

- **Two-vector frame derivatives may be inaccurate. Let  $R(t)$  represent a time-dependent rotation:**
  - If  $R(t)$  depends on CK data,  $dR(t)/dt$  may be inaccurate because CK rates frequently have low accuracy.
  - If  $R(t)$  depends on velocity vectors, then  $dR(t)/dt$  depends on acceleration determined via numerical differentiation. Typically such derivatives suffer loss of accuracy.
    - » However, if velocities are "well-behaved," numerically derived acceleration can be quite good. Example: GSE frame.
  - If  $R(t)$  depends on position vectors, the velocities associated with those vectors by the SPK system may not be mathematically consistent with the positions. This can happen for SPK types with separate polynomials for position and velocity, such as types 3, 8, 9, and 14.
  - If  $R(t)$  depends on aberration-corrected vectors, the associated velocities may be inaccurate due to accuracy limitations of the aberration corrections applied to velocities by the SPK system.



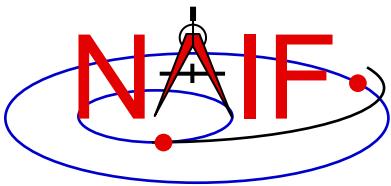
# Numerical Issues - 2

---

Navigation and Ancillary Information Facility

- **Recommendations**

- Avoid using aberration corrections in two-vector frame definitions if accurate velocity transformations are required.
- Be aware of the accuracy of the data on which two-vector frames are based.



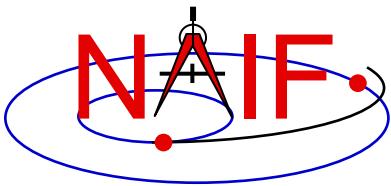
# Limitations - 1

---

Navigation and Ancillary Information Facility

- **Simulated recursion:**

- ANSI Fortran 77 doesn't support recursion, so the SPICE dynamic frame system implements limited, simulated recursion.
  - » Two levels of recursion are supported for selected SPK and Frames System routines.
- Users must avoid requesting "deeper" recursion than the SPICE dynamic frame system can support.
  - » When defining dynamic frames:
    - Choose J2000 as the base frame for two-vector frames.
    - Except for Euler frames, avoid using dynamic frames as base frames.
    - Try to avoid choosing a dynamic frame as the frame associated with a velocity or constant vector.
  - » In SPK, CK, or PCK kernels, don't use two-vector frames as the base frame relative to which ephemeris or attitude data are specified.
    - "Of-date" or Euler frames are OK for this purpose.

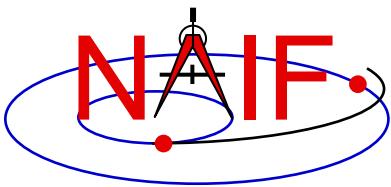


# Limitations - 2

---

Navigation and Ancillary Information Facility

- **Run-time efficiency:**
  - Dynamic frame evaluation typically requires more computation than is needed for CK- or PCK-based frames.
    - » For example, evaluation of a two-vector frame may involve several SPK calls.
    - » Euler frames are an exception: these are fairly efficient as long as they don't have a base frame that requires a lot of computation to evaluate.
  - To minimize the performance penalty:
    - » use J2000 as the base frame for two-vector frames.
    - » use the simplest frames possible for association with velocity or constant vectors in two-vector frame definitions.
      - Prefer non-dynamic frames to dynamic frames and inertial frames to non-inertial frames where there is a choice.

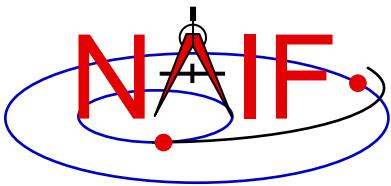


---

Navigation and Ancillary Information Facility

# Instrument Kernel IK

January 2020

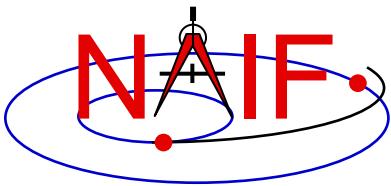


# Purpose

---

Navigation and Ancillary Information Facility

- **The Instrument Kernel serves as a repository for instrument-specific geometry information useful within the SPICE context.**
  - Always included:
    - » If an instrument has a field-of-view (FOV), specifications for an instrument's size, shape, and orientation
  - Other possibilities:
    - » Timing parameters
    - » Optical parameters
    - » Detector geometric parameters
    - » Optical distortion parameters
- **An antenna or solar array or other structure for which pointing is important can also use the IK**
- **Note: instrument mounting alignment data are specified in a mission's Frames Kernel (FK)**



# I-Kernel Structure

---

Navigation and Ancillary Information Facility

- An I-Kernel is a SPICE text kernel. The format and structure of a typical I-Kernel is shown below.

KPL/IK

Comments describing the keywords and values to follow, as well as any other pertinent information.

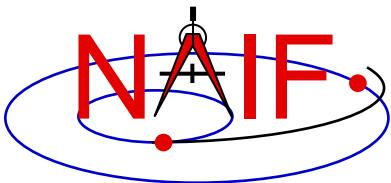
```
\begindata
    Keyword = Value(s) Assignment
    Keyword = Value(s) Assignment
```

```
\begintext
```

More descriptive comments.

```
\begindata
    Keyword = Value(s) Assignment
\begintext
```

More descriptive comments.  
etc ...

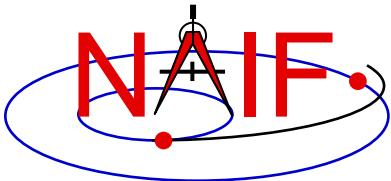


# I-Kernel Contents (1)

---

Navigation and Ancillary Information Facility

- Examples of IK keywords, with descriptions:
  - **INS-94031\_FOCAL\_LENGTH** MGS MOC NA focal length
  - **INS-41220\_INFOV** MEX HRSC SRC pixel angular size
  - **INS-41130\_NUMBER\_OF\_SECTORS** MEX ASPERA NPI number of sectors
- In general SPICE does not require any specific keywords to be present in an IK
  - One exception is a set of keywords defining an instrument's FOV, if the SPICE Toolkit's GETFOV routine is planned to be used to retrieve the FOV attributes
    - » Keywords required by GETFOV will be covered later in this tutorial
- The requirements on keywords in an IK are the following:
  - Keywords must begin with **INS[#]**, where **[#]** is replaced with the NAIF instrument ID code (which is a negative number)
  - The total length of the keyword must be less than or equal to 32 characters
  - Keywords are case-sensitive (Keyword != KEYWORD)

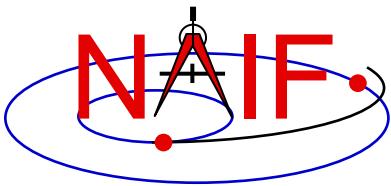


# I-Kernel Contents (2)

---

Navigation and Ancillary Information Facility

- **IKs should contain extensive comments regarding:**
  - Instrument overview
  - Reference source(s) for the data included in the IK
  - Names/IDs assigned to the instrument and its parts
  - Explanation of each keyword included in the file
  - Description of the FOV and detector layout
  - Where appropriate, descriptions of the algorithms in which parameters provided in the IK are used, and even fragments of source code implementing these algorithms
    - » For example optical distortion models or timing algorithms
- **These comments exist primarily to assist users in integrating I-Kernel data into their applications**
  - One needs to know the keyword name to get its value(s) from the IK data
  - One needs to know what each value means in order to use it properly



# I-Kernel Interface Routines

Navigation and Ancillary Information Facility

- As with any SPICE kernel, an IK is loaded using FURNSH

```
CALL FURNSH ( 'ik_file_name.ti' ) { Better yet, use a FURNSH kernel }
```

- By knowing the name and type (DP, integer, or character) of a keyword of interest, the value(s) associated with that keyword can be retrieved using G\*POOL routines

```
CALL GDPOOL ( NAME, START, ROOM, N, VALUES, FOUND ) for DP values
```

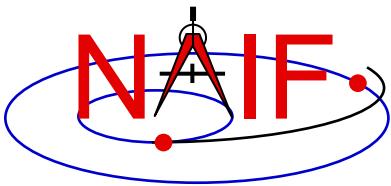
```
CALL GIPOOL ( NAME, START, ROOM, N, VALUES, FOUND ) for integer values
```

```
CALL GCPOOL ( NAME, START, ROOM, N, VALUES, FOUND ) for character string values
```

- When an instrument's FOV is defined in the IK using a special set of keywords discussed later in this tutorial, the FOV shape, reference frame, boresight vector, and boundary vectors can be retrieved by calling the GETFOV routine

```
CALL GETFOV ( INSTID, ROOM, SHAPE, FRAME, BSIGHT, N, BOUNDS )
```

*FORTRAN examples are shown*



# FOV Definition Keywords (1)

Navigation and Ancillary Information Facility

- The following keywords defining FOV attributes for the instrument with NAIF ID (#) must be present in the IK if the SPICE Toolkit's GETFOV module will be used

- Keyword defining shape of the FOV

```
INS#_FOV_SHAPE      = 'CIRCLE' or 'ELLIPSE' or  
                      'RECTANGLE' or 'POLYGON'
```

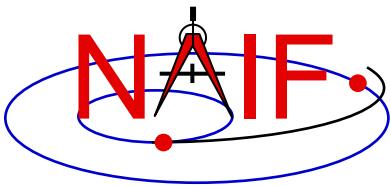
- Keyword specifying the reference frame in which the boresight vector and FOV boundary vectors are specified

```
INS#_FOV_FRAME      = 'frame name'
```

- Keyword defining the boresight vector

```
INS#_BORE SIGHT     = ( X, Y, Z )
```

continued on next page



# FOV Definition Keywords (2)

---

Navigation and Ancillary Information Facility

- **Keyword(s) defining FOV boundary vectors, provided in either of two ways**

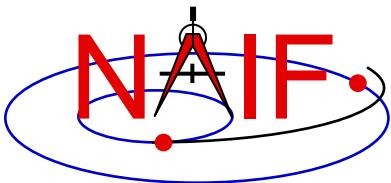
- 1) **By specifying boundary vectors explicitly**

```
INS#_FOV_CLASS_SPEC      = 'CORNERS'  
INS#_FOV_BOUNDARY_CORNERS = ( X(1), Y(1), Z(1),  
                                ...           ...           ...  
                                X(n), Y(n), Z(n) )
```

where the `FOV_BOUNDARY_CORNERS` keyword provides an array of vectors that point to the "corners" of the instrument field of view.

**Note:** Use of the `INS#_FOV_CLASS_SPEC` keyword is optional when explicit boundary vectors are provided.

continued on next page



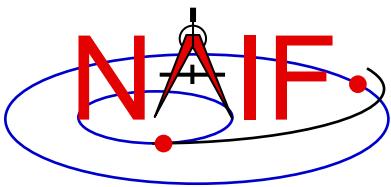
# FOV Definition Keywords (3)

Navigation and Ancillary Information Facility

- 2) By providing half angular extents of the FOV (possible only for circular, elliptical or rectangular FOVs)

<code>INS#_FOV_CLASS_SPEC</code>	= 'ANGLES'
<code>INS#_FOV_REF_VECTOR</code>	= ( X, Y, Z )
<code>INS#_FOV_REF_ANGLE</code>	= halfangle1
<code>INS#_FOV_CROSS_ANGLE</code>	= halfangle2
<code>INS#_FOV_ANGLE_UNITS</code>	= 'DEGREES' or 'RADIANS' or ...

where the `FOV_REF_VECTOR` keyword specifies a reference vector that, together with the boresight vector, define the plane in which the half angle given in the `FOV_REF_ANGLE` keyword is measured. The other half angle given in the `FOV_CROSS_ANGLE` keyword is measured in the plane normal to this plane and containing the boresight vector.

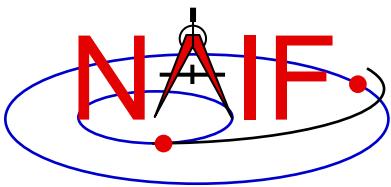


# FOV Definition Keywords (4)

---

Navigation and Ancillary Information Facility

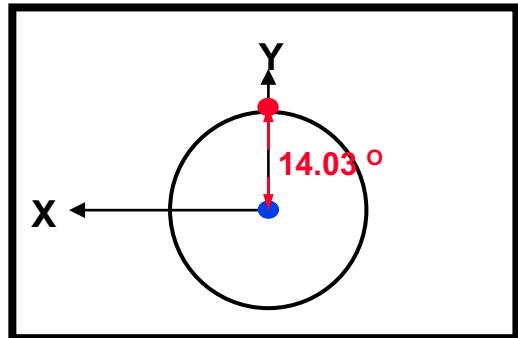
- When explicit boundary vectors are provided, they must be listed in either clockwise or counter-clockwise order, not randomly
- Neither the boresight nor reference vector has to be co-aligned with one of the FOV frame's axes
  - But for convenience, each is frequently defined to be along one of the FOV axes
- None of the boresight, corner or reference vector has to be a unit vector
  - But these frequently are defined as unit vectors
- When a FOV is specified using the half angular extents method, the boresight and reference vectors have to be linearly independent but they don't have to be perpendicular
  - But for convenience the reference vector is usually picked to be normal to the boresight vector
- Half angular extents for a rectangular FOV specify the angles between the boresight and the FOV sides, i.e. they are for the middle of the FOV
- The next several pages show examples of FOV definitions



# Circular Field of View

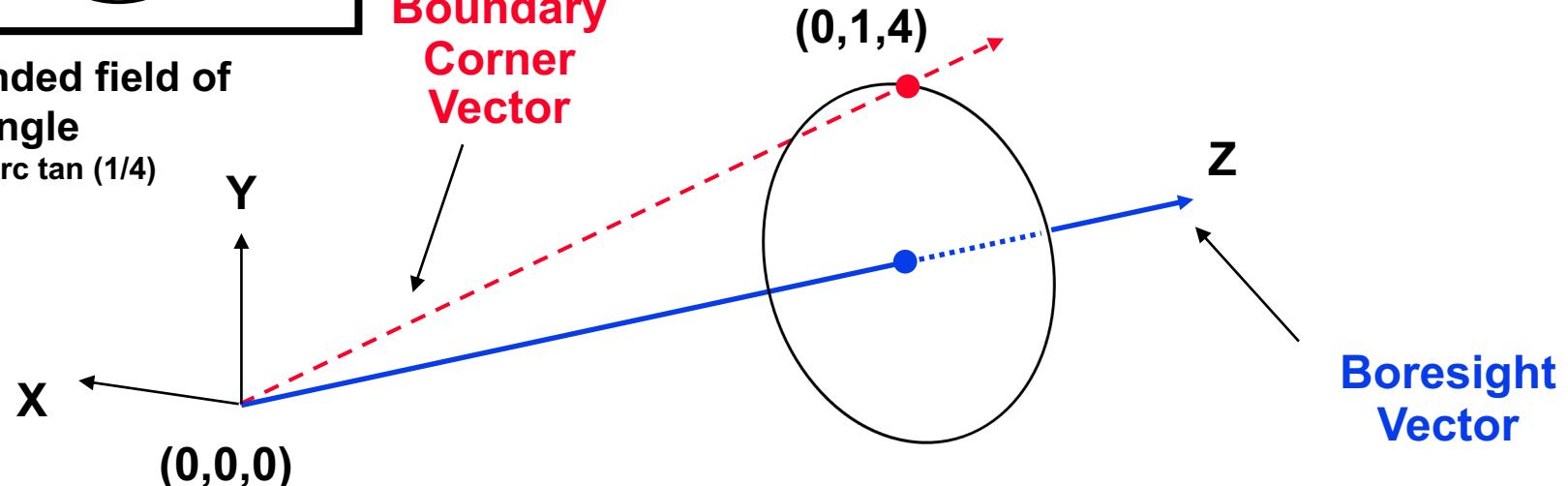
Navigation and Ancillary Information Facility

Consider an instrument with a circular field of view.



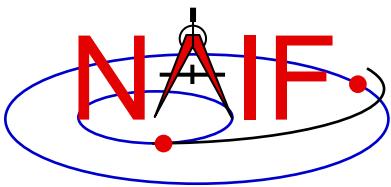
Subtended field of view angle  
 $14.03 = \text{arc tan} (1/4)$

Boundary  
Corner  
Vector



Instrument  
focal point

Boresight  
Vector



# Circular FOV Definition

---

Navigation and Ancillary Information Facility

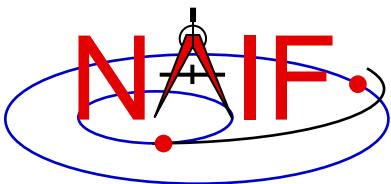
The following sets of keywords and values describe this circular field of view:

Specifying boundary vectors explicitly:

<code>INS-11111_FOV_SHAPE</code>	<code>= 'CIRCLE'</code>
<code>INS-11111_FOV_FRAME</code>	<code>= 'FRAME_FOR_INS-11111'</code>
<code>INS-11111_BORESIGHT</code>	<code>= ( 0.0 0.0 1.0 )</code>
<code>INS-11111_FOV_BOUNDARY_CORNERS</code>	<code>= ( 0.0 1.0 4.0 )</code>

Specifying half angular extents of the FOV:

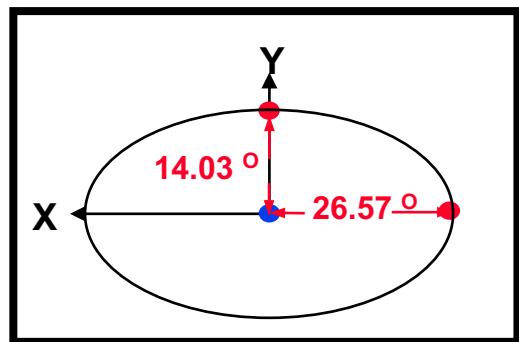
<code>INS-11111_FOV_SHAPE</code>	<code>= 'CIRCLE'</code>
<code>INS-11111_FOV_FRAME</code>	<code>= 'FRAME_FOR_INS-11111'</code>
<code>INS-11111_BORESIGHT</code>	<code>= ( 0.0 0.0 1.0 )</code>
<code>INS-11111_FOV_CLASS_SPEC</code>	<code>= 'ANGLES'</code>
<code>INS-11111_FOV_REF_VECTOR</code>	<code>= ( 0.0 1.0 0.0 )</code>
<code>INS-11111_FOV_REF_ANGLE</code>	<code>= 14.03624347</code>
<code>INS-11111_FOV_ANGLE_UNITS</code>	<code>= 'DEGREES'</code>



# Elliptical Field of View

Navigation and Ancillary Information Facility

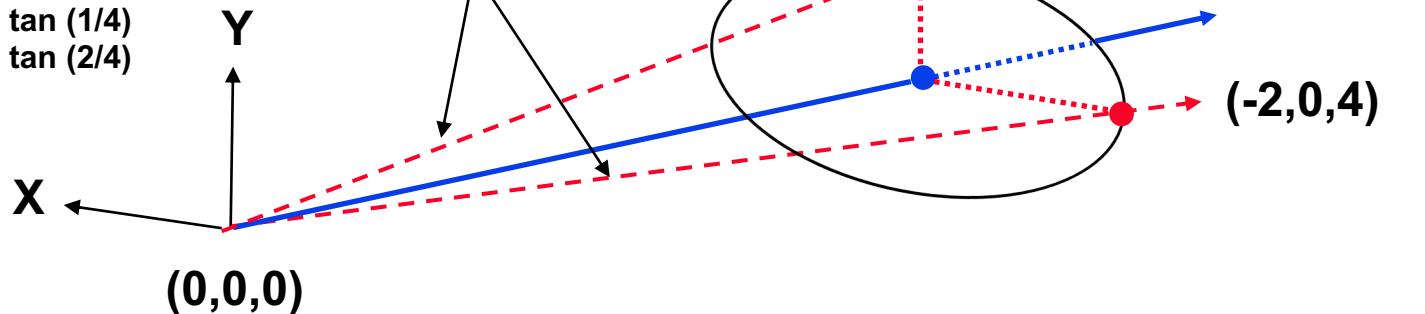
Consider an instrument with an elliptical field of view.



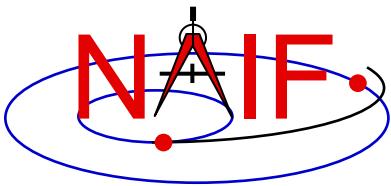
Subtended field of view angle

$$14.03 = \text{arc tan} (1/4)$$
$$26.57 = \text{arc tan} (2/4)$$

Boundary  
Corner  
Vectors



Instrument  
focal point



# Elliptical FOV Definition

---

Navigation and Ancillary Information Facility

**The following sets of keywords and values describe this elliptical field of view:**

**Specifying boundary vectors explicitly:**

<code>INS-22222_FOV_SHAPE</code>	<code>= 'ELLIPSE'</code>
<code>INS-22222_FOV_FRAME</code>	<code>= 'FRAME_FOR_INS-22222'</code>
<code>INS-22222_BORESIGHT</code>	<code>= ( 0.0 0.0 1.0 )</code>
<code>INS-22222_FOV_BOUNDARY_CORNERS</code>	<code>= ( 0.0 1.0 4.0 -2.0 0.0 4.0 )</code>

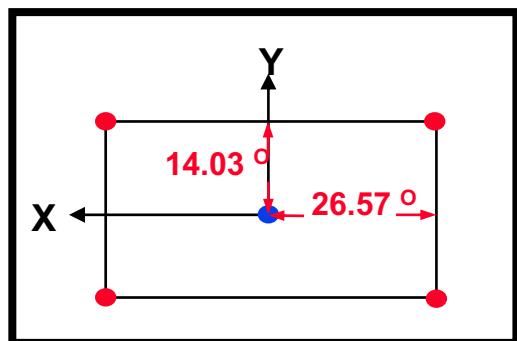
**Specifying half angular extents of the FOV:**

<code>INS-22222_FOV_SHAPE</code>	<code>= 'ELLIPSE'</code>
<code>INS-22222_FOV_FRAME</code>	<code>= 'FRAME_FOR_INS-22222'</code>
<code>INS-22222_BORESIGHT</code>	<code>= ( 0.0 0.0 1.0 )</code>
<code>INS-22222_FOV_CLASS_SPEC</code>	<code>= 'ANGLES'</code>
<code>INS-22222_FOV_REF_VECTOR</code>	<code>= ( 0.0 1.0 0.0 )</code>
<code>INS-22222_FOV_REF_ANGLE</code>	<code>= 14.03624347</code>
<code>INS-22222_FOV_CROSS_ANGLE</code>	<code>= 26.56505118</code>
<code>INS-22222_FOV_ANGLE_UNITS</code>	<code>= 'DEGREES'</code>

# Rectangular Field of View

Navigation and Ancillary Information Facility

Consider an instrument with a rectangular field of view.



**Subtended field of view angle**

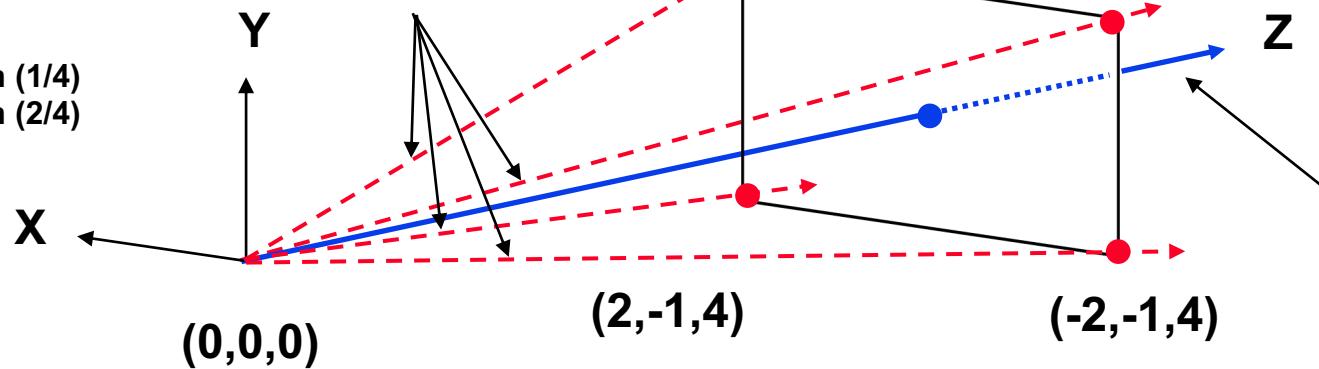
$$14.03 = \text{arc tan} (1/4)$$

$$26.57 = \text{arc tan} (2/4)$$

**Boundary  
Corner  
Vectors**

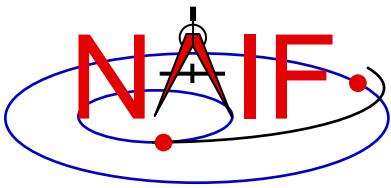
(2,1,4)

(-2,1,4)



**Instrument  
focal point**

**Boresight  
Vector**



# Rectangular FOV Definition

Navigation and Ancillary Information Facility

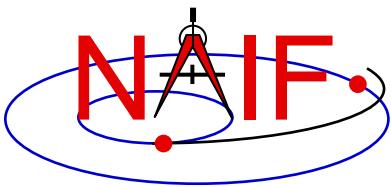
The following sets of keywords and values describe this rectangular field of view:

Specifying boundary vectors explicitly:

INS-33333_FOV_SHAPE	= 'RECTANGLE'
INS-33333_FOV_FRAME	= 'FRAME_FOR_INS-33333'
INS-33333_BORESIGHT	= ( 0.0 0.0 1.0 )
INS-33333_FOV_BOUNDARY_CORNERS	= ( 2.0 1.0 4.0 -2.0 1.0 4.0 -2.0 -1.0 4.0 2.0 -1.0 4.0 )

Specifying half angular extents of the FOV:

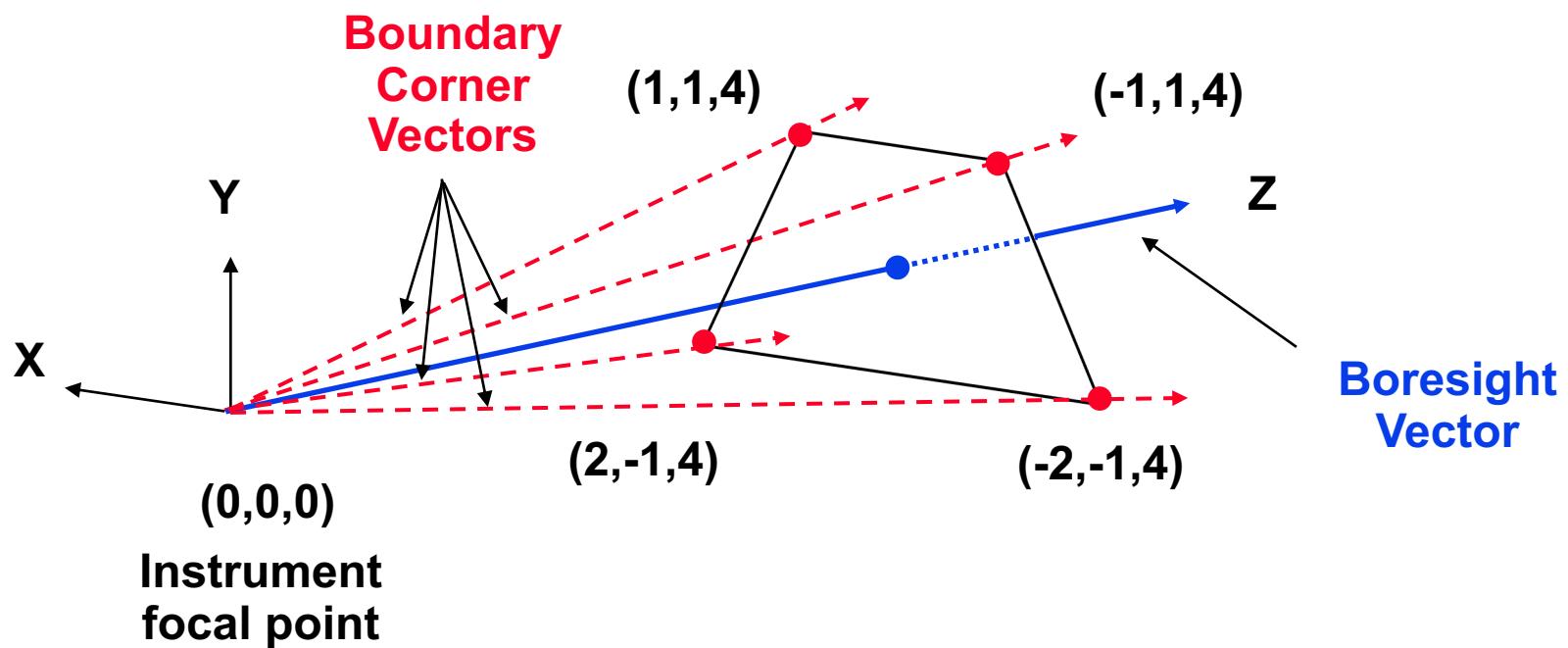
INS-33333_FOV_SHAPE	= 'RECTANGLE'
INS-33333_FOV_FRAME	= 'FRAME_FOR_INS-33333'
INS-33333_BORESIGHT	= ( 0.0 0.0 1.0 )
INS-33333_FOV_CLASS_SPEC	= 'ANGLES'
INS-33333_FOV_REF_VECTOR	= ( 0.0 1.0 0.0 )
INS-33333_FOV_REF_ANGLE	= 14.03624347
INS-33333_FOV_CROSS_ANGLE	= 26.56505118
INS-33333_FOV_ANGLE_UNITS	= 'DEGREES'

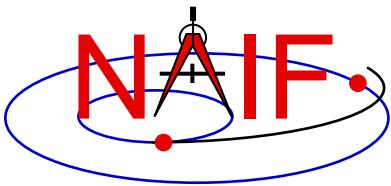


# Polygonal Fields of View

Navigation and Ancillary Information Facility

Consider an instrument with a trapezoidal field of view.





# Polygonal FOV Definition

---

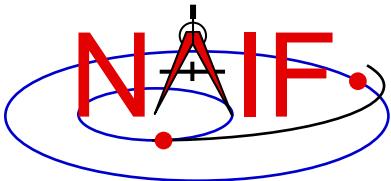
Navigation and Ancillary Information Facility

**The following sets of keywords and values describe this polygonal field of view:**

**Specifying boundary vectors explicitly:**

```
INS-44444_FOV_SHAPE      = 'POLYGON'  
INS-44444_FOV_FRAME      = 'FRAME_FOR_INS-44444'  
INS-44444_BORESIGHT      = ( 0.0  0.0  1.0 )  
INS-44444_FOV_BOUNDARY_CORNERS = ( 1.0  1.0  4.0  
                                     -1.0  1.0  4.0  
                                     -2.0 -1.0  4.0  
                                     2.0 -1.0  4.0 )
```

- A polygonal FOV cannot be specified using half angular extents.

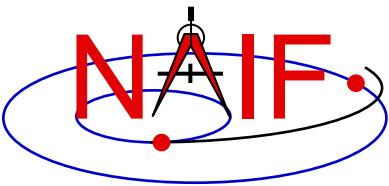


# IK Utility Programs

---

Navigation and Ancillary Information Facility

- **No IK utility programs are included in the Toolkit**
- **Two IK utility programs are provided on the NAIF website (<https://naif.jpl.nasa.gov/naif/utilities.html>)**
  - OPTIKS**    displays field-of-view summary for all FOVs defined in a collection of IK files.
  - BINGO**    converts IK files between UNIX and DOS text formats

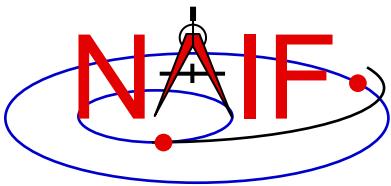


# Additional Information on IK

---

Navigation and Ancillary Information Facility

- **The best way to learn more about IKs is to examine some found in the NAIF Node archives.**
  - Start looking here:  
[https://naif.jpl.nasa.gov/naif/data\\_archived.html](https://naif.jpl.nasa.gov/naif/data_archived.html)
- **NAIF does not yet have an “I-Kernel Required Reading” document**
- **But information about IKs is available in other documents:**
  - header of the GETFOV routine
  - Kernel Required Reading
  - OPTIKS User’s Guide
  - Porting\_kernels tutorial
  - NAIF IDs Tutorial
  - Frames Required Reading

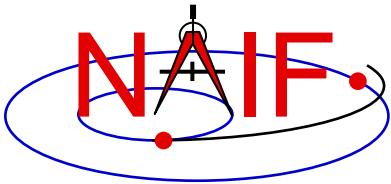


# Backup

---

Navigation and Ancillary Information Facility

- IK file example
- Computing angular extents from corner vectors returned by GETFOV



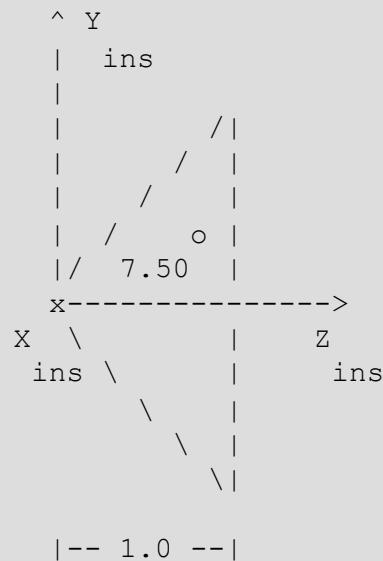
# Sample IK Data

Navigation and Ancillary Information Facility

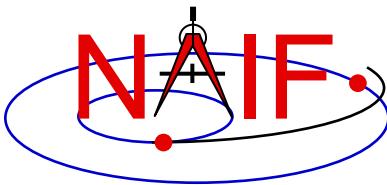
**The following LEMMS1 FOV definition was taken from the Cassini MIMI IK (cas\_mimi\_v11.ik):**

Low Energy Magnetospheric Measurements System 1 (LEMMS1)

Since the MIMI\_LEMMS1 detector's FOV is circular and it's diameter is 15.0 degrees, looking down the X-axis in the CASSINI\_MIMI\_LEMMS1 frame, we have:  
(Note we are arbitrarily choosing a vector that terminates in the Z=1 plane.)



**continues**



# Sample IK Data

Navigation and Ancillary Information Facility

## FOV definition from the Cassini MIMI IK (continued):

```
The Y component of one 'boundary corner' vector is:
```

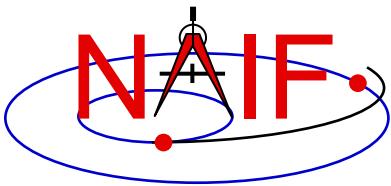
$$\begin{aligned} \text{Y Component} &= 1.0 * \tan(7.50 \text{ degrees}) \\ &= 0.131652498 \end{aligned}$$

```
The boundary corner vector as displayed below is  
normalized to unit length:
```

```
\begindata

INS-82762_FOV_FRAME = 'CASSINI_MIMI_LEMMS1'
INS-82762_FOV_SHAPE = 'CIRCLE'
INS-82762_BORESIGHT =
  0.0000000000000000  0.0000000000000000 +1.0000000000000000
  )
INS-82762_FOV_BOUNDARY_CORNERS =
  0.0000000000000000 +0.1305261922200500  +0.9914448613738100
  )

\begintext
```



# Circular FOV Angular Size

---

Navigation and Ancillary Information Facility

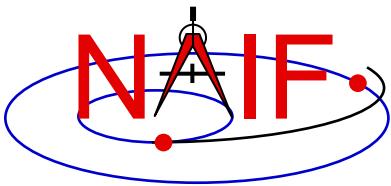
**The angular separation between the boundary corner vector and the boresight is the angular size.**

**FORTRAN EXAMPLE**

```
C      Retrieve FOV parameters.  
      CALL GETFOV(-11111, 1, SHAPE, FRAME, BSGHT, N, BNDS)  
  
C      Compute the angular size.  
      ANGSIZ = VSEP( BSGHT, BNDS(1,1) )
```

**C EXAMPLE**

```
/* Define the string length parameter. */  
#define STRSIZ          80  
  
/* Retrieve the field of view parameters. */  
getfov_c(-11111, 1, STRSIZ, STRSIZ, shape, frame,  
         bsght, &n, bnnds);  
  
/* Compute the angular separation. */  
angsiz = vsep_c( bsght, &(bnnds[0][0]));
```



# Elliptical FOV Angular Size - 1

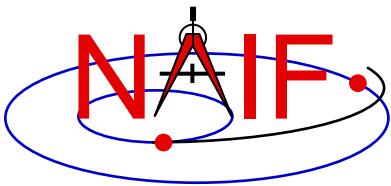
---

Navigation and Ancillary Information Facility

**The angular sizes are the angular separations between the boresight and the boundary vectors.**

**FORTRAN EXAMPLE**

```
C   Retrieve the FOV parameters from the kernel pool.  
CALL GETFOV(-22222, 2, SHAPE, FRAME, BSGHT, N, BNDS)  
  
C   Compute the angular separations.  
ANG1    = VSEP( BSGHT, BNDS(1,1) )  
ANG2    = VSEP( BSGHT, BNDS(1,2) )  
  
C   The angle along the semi-major axis is the larger  
C   of the two separations computed.  
LRGANG = MAX( ANG1, ANG2)  
SMLANG = MIN( ANG1, ANG2)
```



# Elliptical FOV Angular Size - 2

---

Navigation and Ancillary Information Facility

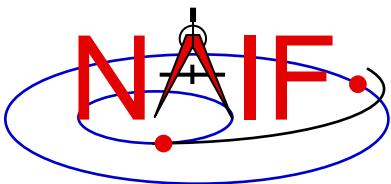
## C EXAMPLE

```
/* Define the string length parameter. */
#define STRSIZ      80

/* Retrieve the FOV parameters from the kernel pool. */
getfov_c(-22222, 2, STRSIZ, STRSIZ, shape, frame,
          bsght, &n, bnds);

/* Compute the angular separations. */
ang1 = vsep_c( bsght, &(bnds[0][0]));
ang2 = vsep_c( bsght, &(bnds[1][0]));

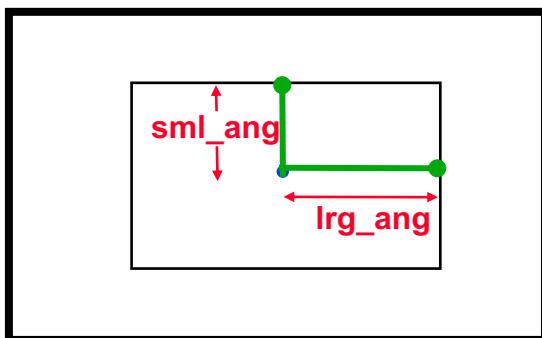
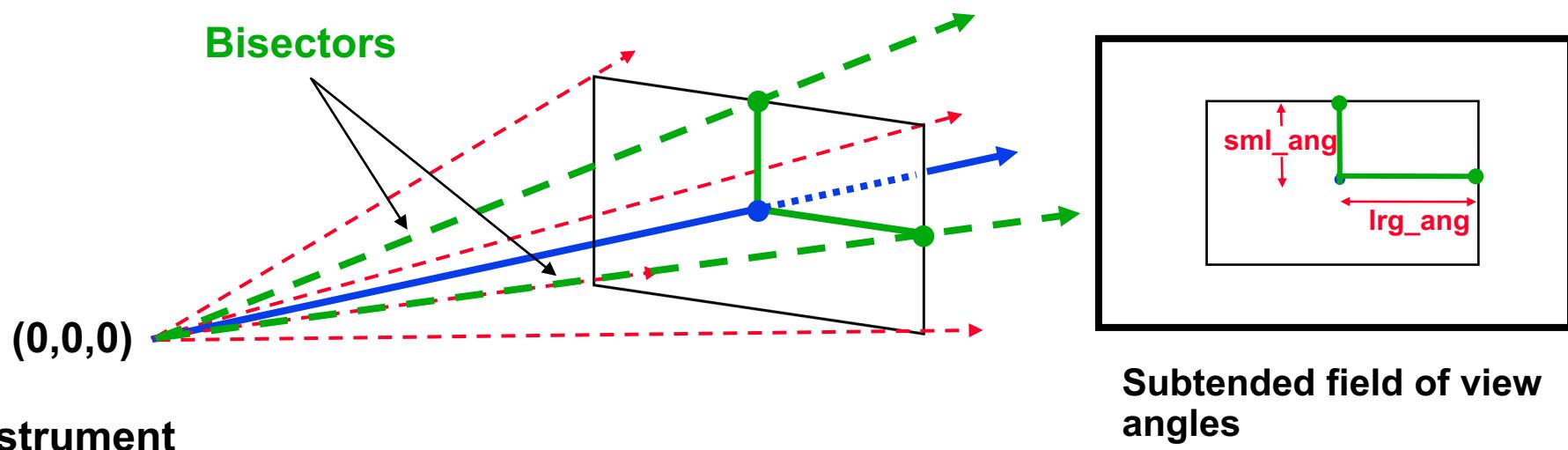
/* The angle along the semi-major axis is the larger of the
two separations computed. */
if ( ang1 > ang2 ) {
    lrgang = ang1; smlang = ang2; }
else {
    lrgang = ang2; smlang = ang1; }
```



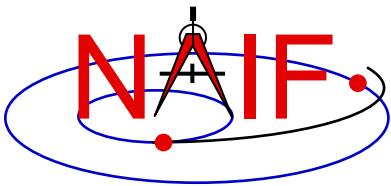
# Rectangular FOV Angular Size - 1

Navigation and Ancillary Information Facility

The angular extents of the FOV are computed by calculating the angle between the bisector of adjacent unit boundary vectors and the boresight.



Subtended field of view angles



# Rectangular FOV Angular Size - 2

---

Navigation and Ancillary Information Facility

## FORTRAN EXAMPLE

C      Retrieve FOV parameters from the kernel pool.

```
CALL GETFOV(-33333, 4, SHAPE, FRAME, BSGHT, N, BNDS)
```

C      Normalize the 3 boundary vectors

```
CALL UNORM(BNDS(1,1), UNTBND(1,1), MAG)
CALL UNORM(BNDS(1,2), UNTBND(1,2), MAG)
CALL UNORM(BNDS(1,3), UNTBND(1,3), MAG)
```

C      Compute the averages.

```
CALL VADD(UNTBND(1,1), UNTBND(1,2), VEC1)
CALL VSCL(0.5, VEC1, VEC1)
```

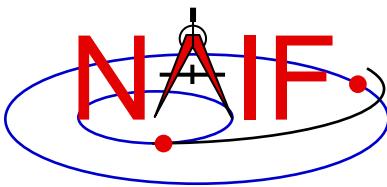
```
CALL VADD(UNTBND(1,2), UNTBND(1,3), VEC2)
CALL VSCL(0.5, VEC2, VEC2)
```

C      Compute the angular separations

```
ANG1    = VSEP( BSGHT, VEC1 )
ANG2    = VSEP( BSGHT, VEC2 )
```

C      Separate the larger and smaller angles.

```
LRGANG = MAX( ANG1, ANG2 )
SMLANG = MIN( ANG1, ANG2 )
```



# Rectangular FOV Angular Size - 3

## Navigation and Ancillary Information Facility

### C EXAMPLE

```
/* Define the string length parameter. */
#define STRSIZ      80

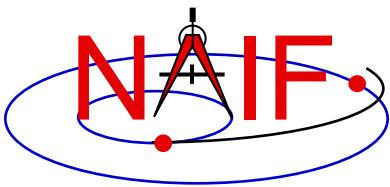
/* Retrieve the FOV parameters from the kernel pool. */
getfov_c(-33333, 4, STRSIZ, STRSIZ, shape, frame,
          bsght, &n, bnds);

/* Normalize the 3 boundary vectors. */
unorm_c(&(bnds[0][0]), &(untbnd[0][0]), &mag);
unorm_c(&(bnds[1][0]), &(untbnd[1][0]), &mag);
unorm_c(&(bnds[2][0]), &(untbnd[2][0]), &mag);

/* Compute the averages */
vadd_c(&(untbnd[0][0]), &(untbnd[1][0]), vec1);
vscl_c(0.5, vec1, vec1);
vadd_c(&(untbnd[1][0]), &(untbnd[2][0]), vec2);
vscl_c(0.5, vec2, vec2);

/* Compute the angular separations. */
ang1 = vsep_c( bsght, vec1);
ang2 = vsep_c( bsght, vec2);

/* Separate the larger and smaller angles. */
if ( ang1 > ang2 ) {
    lrgang = ang1; smlang = ang2; }
else {
    lrgang = ang2; smlang = ang1; }
```

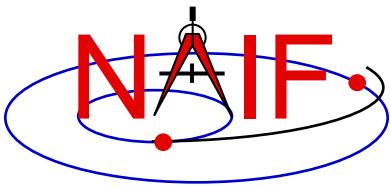


---

Navigation and Ancillary Information Facility

# Reading FKs and IKs

January 2020

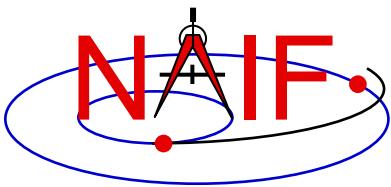


# See The Real Stuff

---

Navigation and Ancillary Information Facility

- It may be useful for the student to examine a few existing Frames Kernels and Instrument Kernels to get a better understanding of the FK and IK tutorial information.
- NAIF suggests you use your browser to examine some of the following “real life” kernels. (You may use `http` in place of `ftp`.)
- DEEP IMPACT:
  - [https://naif.jpl.nasa.gov/pub/naif/pds/data/di-c-spice-6-v1.0/isp\\_1000/data/fk/](https://naif.jpl.nasa.gov/pub/naif/pds/data/di-c-spice-6-v1.0/isp_1000/data/fk/)
  - [https://naif.jpl.nasa.gov/pub/naif/pds/data/di-c-spice-6-v1.0/isp\\_1000/data/ik/](https://naif.jpl.nasa.gov/pub/naif/pds/data/di-c-spice-6-v1.0/isp_1000/data/ik/)
- CASSINI:
  - [https://naif.jpl.nasa.gov/pub/naif/pds/data/co-s\\_j\\_e\\_v-spice-6-v1.0/cosp\\_1000/data/fk/](https://naif.jpl.nasa.gov/pub/naif/pds/data/co-s_j_e_v-spice-6-v1.0/cosp_1000/data/fk/)
  - [https://naif.jpl.nasa.gov/pub/naif/pds/data/co-s\\_j\\_e\\_v-spice-6-v1.0/cosp\\_1000/data/ik/](https://naif.jpl.nasa.gov/pub/naif/pds/data/co-s_j_e_v-spice-6-v1.0/cosp_1000/data/ik/)
- MESSENGER:
  - [https://naif.jpl.nasa.gov/pub/naif/pds/data/mess-e\\_v\\_h-spice-6-v1.0/messsp\\_1000/data/fk/](https://naif.jpl.nasa.gov/pub/naif/pds/data/mess-e_v_h-spice-6-v1.0/messsp_1000/data/fk/)
  - [https://naif.jpl.nasa.gov/pub/naif/pds/data/mess-e\\_v\\_h-spice-6-v1.0/messsp\\_1000/data/ik/](https://naif.jpl.nasa.gov/pub/naif/pds/data/mess-e_v_h-spice-6-v1.0/messsp_1000/data/ik/)
- MARS EXPRESS:
  - <https://naif.jpl.nasa.gov/pub/naif/MEX/kernels/fk/>
  - <https://naif.jpl.nasa.gov/pub/naif/MEX/kernels/ik/>

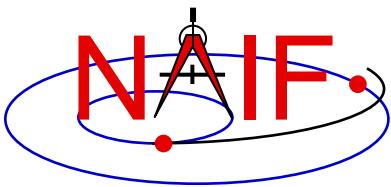


---

Navigation and Ancillary Information Facility

# Derived Quantities

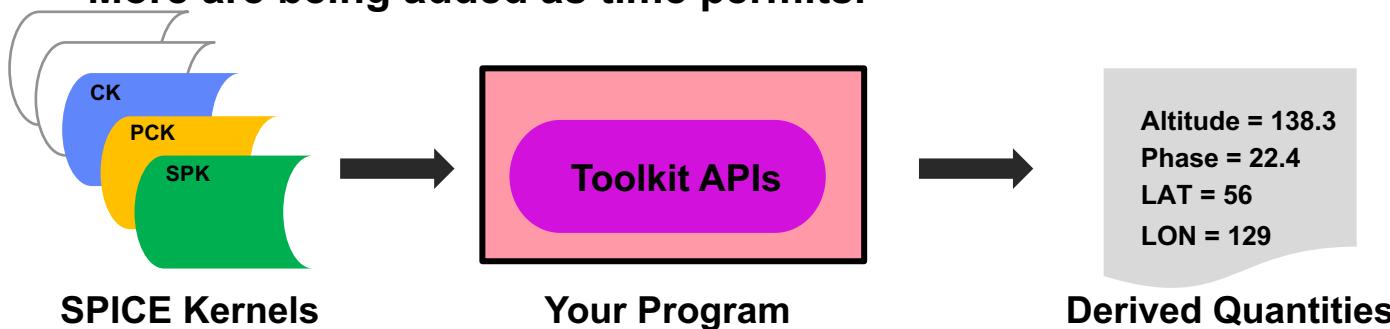
January 2020



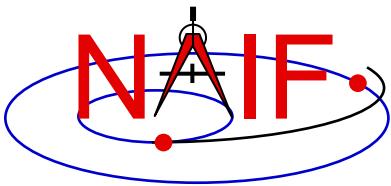
# What are Derived Quantities?

Navigation and Ancillary Information Facility

- Derived quantities, what we often call “observation geometry,” are produced using data from SPICE kernels and Toolkit software.
  - These are the primary reason that SPICE exists!
- The SPICE Toolkit contains many routines that assist with the computations of derived quantities.
  - Some are fairly low level, some are quite high level.
  - More are being added as time permits.



- Examples follow on the next several pages.

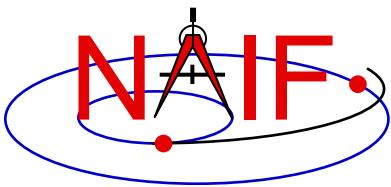


# High-level Geometric Computations

---

Navigation and Ancillary Information Facility

- **Geometric Parameter**
  - Determine a geometric quantity or condition at a specified time.
- **Geometry Finder (GF)**
  - Find times, or time spans, when a specified “geometric event” occurs, or when a specified “geometric condition” exists.
    - » This is such a large topic that a separate tutorial (“geometry\_finder”) has been written for it.

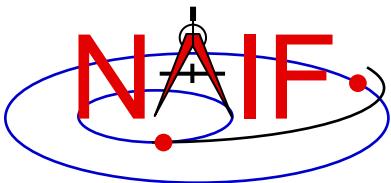


# Examples of Geometric Parameters

Navigation and Ancillary Information Facility

- **Illumination angles (phase, incidence, emission)**
  - ILLUMF, ILLUMG, ILUMIN\*
- **Sub-solar point**
  - SUBSLR\*
- **Sub-observer point**
  - SUBPNT\*
- **Surface intercept point**
  - SINCPT\*, DSKXV, DSKXSI
- **Longitude of the sun (Ls), an indicator of season**
  - LSPCN
- **Phase angle between body centers**
  - PHASEQ
- **Limb and terminator points on an ellipsoid or DSK**
  - LIMBPT, TERMPT
- **Surface points at specified longitude, latitude coordinates**
  - LATSRF
- **Outward surface normal on extended object**
  - SRFNRM

\* These routines supercede the now deprecated routines ILLUM, SUBSOL, SUBPT and SRFXPT

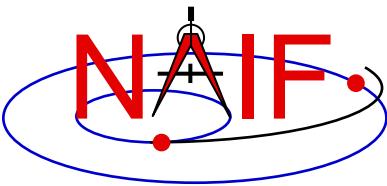


# Examples of Geometric Conditions

---

Navigation and Ancillary Information Facility

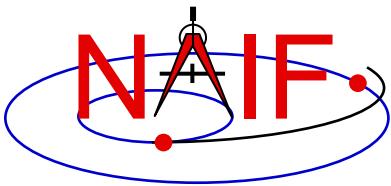
- **Ray in field-of-view?**
  - FOVRAY
- **Ephemeris object within field-of-view?**
  - FOVTRG
- **Determine occultation condition**
  - OCCULT



# Examples of Geometric Searches

Navigation and Ancillary Information Facility

- **Find times when:**
  - ray is in field-of-view
    - » GFRFOV
  - ephemeris object is within field-of-view
    - » GFTFOV
  - object is in occultation or transit
    - » GFOCLT
  - object is at periapse
    - » GFDIST
  - latitude and longitude are in specified ranges
    - » GFPOSC
  - solar incidence angle is below a specified limit
    - » GFILUM
- **Far more GF functionality is available; see the GF tutorial.**



# Geometry Related to Objects

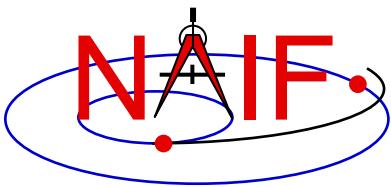
Navigation and Ancillary Information Facility

## Function

- **Ellipsoids**
  - nearest point
  - surface ray intercept
  - surface normal
  - point on surface
  - limb
  - slice with a plane
  - altitude of ray w.r.t. to ellipsoid
- **Planes**
  - intersect ray and plane
- **Ellipses**
  - project onto a plane
  - find semi-axes of an ellipse
- **Lines**
  - nearest point on line or segment

## Routine

- NEARPT , DNEARP
- SURFPT
- SURFNM , EDNMPT
- EDPNT
- EDLIMB
- INELPL
- NPEDLN
- INRYPL
- PJELPL
- SAELEV
- NPLNPT , NPSGPT



# Position and State Coordinate Transformations

Navigation and Ancillary Information Facility

## Coordinate Transformation

- Transform state vector between two coordinate systems

- 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

## Routine

- XFMSTA

General purpose API

- LATREC  
RECLAT

- PGRREC  
RECPGR

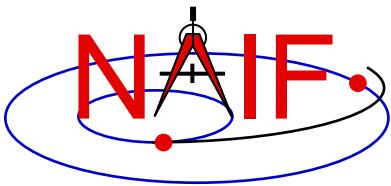
- RADREC  
RECRAD

- GEOREC  
REC GEO

- CYLREC  
RECCYL

- SPHREC  
RECSPH

Single purpose APIs

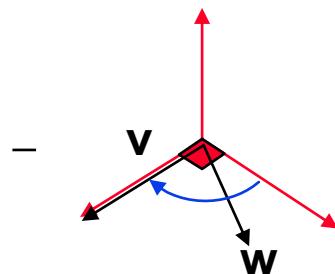
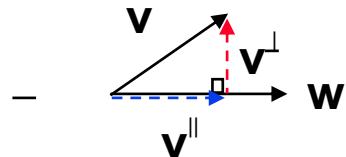


# Vectors

Navigation and Ancillary Information Facility

- **Function**

- $\langle v, w \rangle$
- $v \times w$
- $v / \|v\|$
- $v \times w / \|v \times w\|$
- $v + w$
- $v - w$
- $av [+ bw [+ cu]]$
- angle between  $v$  and  $w$
- $\|v\|$

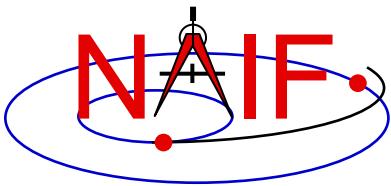


- **Routine**

- **VDOT**, **DVDOT**
- **VCROSS**, **DVCRSS**
- **VHAT**, **DVHAT**
- **UCROSS**, **DUCRSS**
- **VADD**, **VADDG**
- **VSUB**, **VSUBG**
- **VSCL**, **[VLCOM, [VLCOM3] ]**
- **VSEP**
- **VNORM**

- **VPROJ**, **VPERP**

- **TWOVEC**, **FRAME**



# Matrices

Navigation and Ancillary Information Facility

## Selected Matrix-Vector Linear Algebra Routines

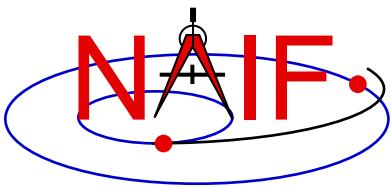
- **Function**
  - $M \times v$
  - $M \times M$
  - $M^t \times v$
  - $M^t \times M$
  - $M \times M^t$
  - $v^t \times M \times v$
  - $M^t$
  - $M^{-1}$
- **Routine**
  - **MXV**
  - **MXM**
  - **MTXV**
  - **MTXM**
  - **MXMT**
  - **VTMV**
  - **XPOSE**
  - **INVERT , INVSTM**

**M** = Matrix

**v** = Vector

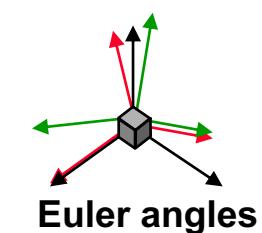
**x** = Multiplication

**T** = Transpose



# Matrix Conversions

Navigation and Ancillary Information Facility



## Function

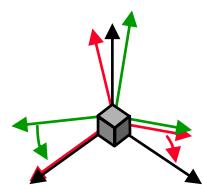
Transform between

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

## Routines

— **EUL2M, M2EUL**

3x3 rotation matrix



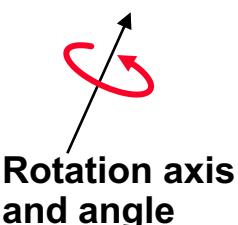
Euler angles and Euler angle rates  
or  
rotation matrix and angular velocity  
vector

Transform between

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

— **EUL2XF, XF2EUL**  
**RAV2XF, XF2RAV**

6x6 state transformation  
matrix



Transform between

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

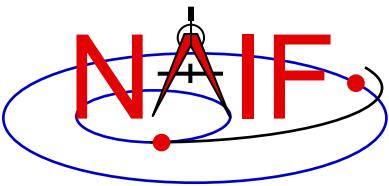
— **RAXISA, AXISAR**  
**ROTATE, ROTMAT**

(Q<sub>0</sub>, Q<sub>1</sub>, Q<sub>2</sub>, Q<sub>3</sub>)  
SPICE Style  
Quaternion

Transform between

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

— **Q2M, M2Q**



# Velocity Coordinate Transformations

Navigation and Ancillary Information Facility

## Coordinate Transformation

- Transform state vector between two coordinate systems

## Routine

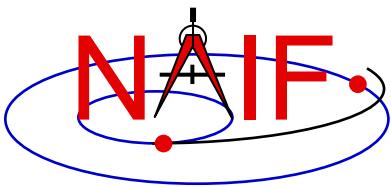
- XFMSTA

General purpose API

- 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

- DRDLAT  
DLATDR
- DRDPGR  
DPGRDR
- DRDLAT  
DLATDR
- DRDGEO  
DGEODR
- DRDCYL  
DCYLDLDR
- DRDSPH  
DSPHDR

Single purpose  
Jacobian matrix APIs



# Examples of Velocity Coordinate Transformations

Navigation and Ancillary Information Facility

This example is for rectangular to spherical

- Using full state vector transformation API

CALL SPKEZR ( TARG, ET, REF, CORR, OBS, STATE, LT )

CALL XFMSTA ( STATE, 'RECTANGULAR', 'SPHERICAL', ' ', OUTSTATE )

- Using velocity-only (Jacobian) APIs

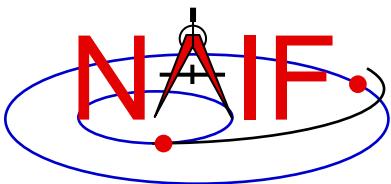
- Transform velocities from rectangular to spherical coordinates using the SPICE Jacobian matrix routines. The SPICE calls that implement this computation are:

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
    - (  $d(r)/dt$ ,  $d(\text{colatitude})/dt$ ,  $d(\text{longitude})/dt$  )
  - 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.

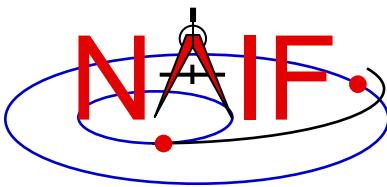


# Examples of Computing Derived Quantities

---

Navigation and Ancillary Information Facility

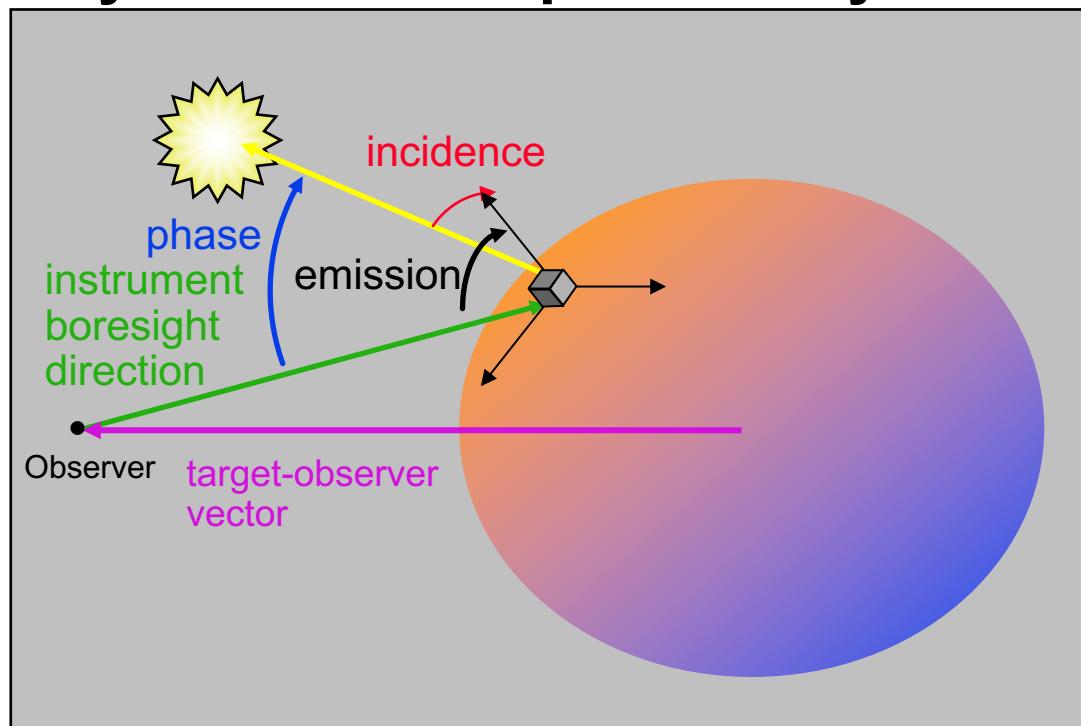
- On the next several pages we present examples of using some of the “derived quantity” APIs.
- Explore the “Most Used SPICE APIs” document to learn more.

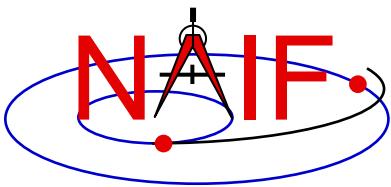


# Computing Illumination Angles

Navigation and Ancillary Information Facility

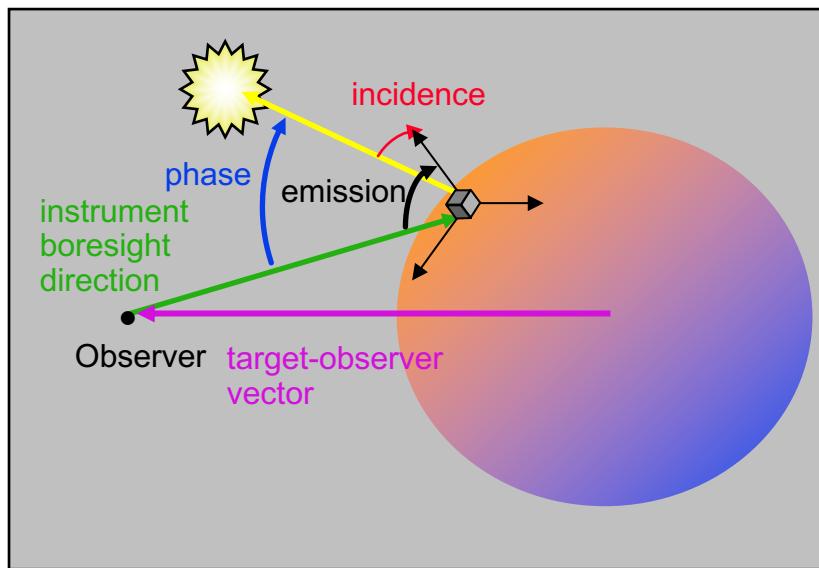
- Given the direction of an instrument boresight in a body-fixed frame, return the illumination angles (incidence, phase, emission) at the boresight's surface intercept on an object, with the object's shape modeled by a tri-axial ellipsoid or by DSK data.



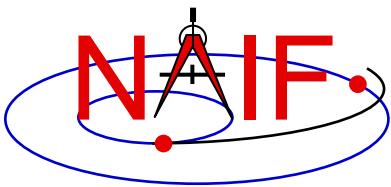


# Computing Illumination Angles

Navigation and Ancillary Information Facility



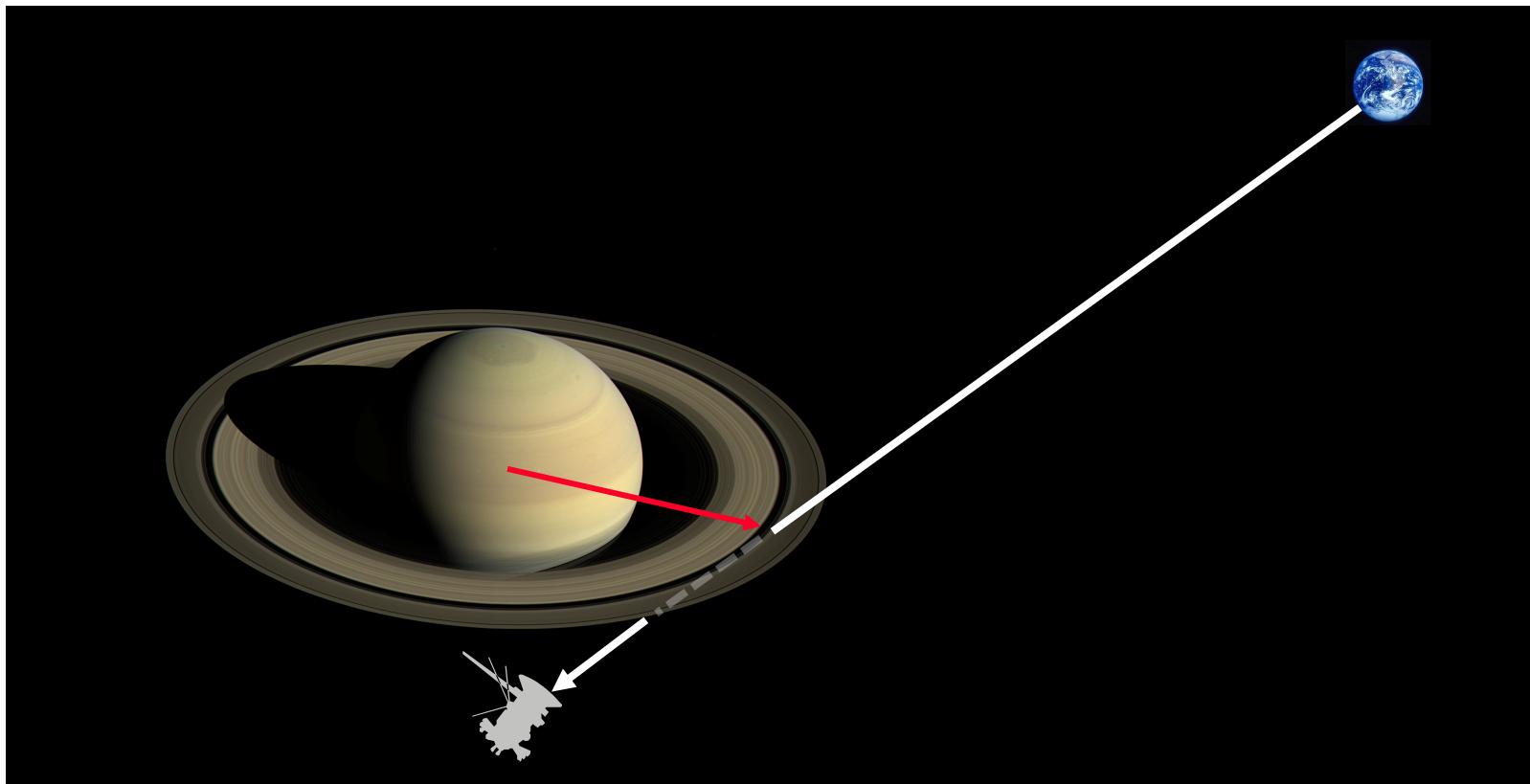
- CALL **GETFOV** to obtain boresight direction vector
- CALL **SINCPT** to find intersection of boresight direction vector with surface
- CALL **ILUMIN** to determine illumination angles

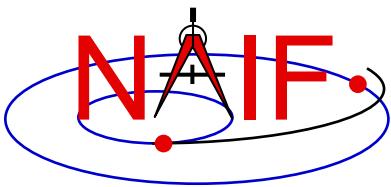


# Computing Ring Plane Intercepts

Navigation and Ancillary Information Facility

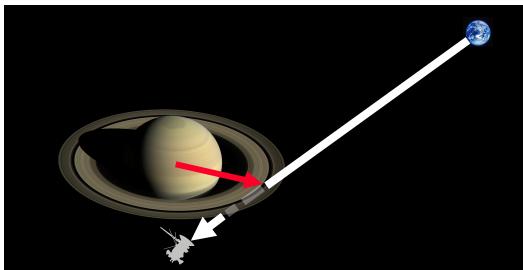
- Determine the intersection of the apparent line of sight vector between Earth and Cassini with Saturn's ring plane and determine the distance of this point from the center of Saturn.





# Computing Ring Plane Intercepts-2

Navigation and Ancillary Information Facility

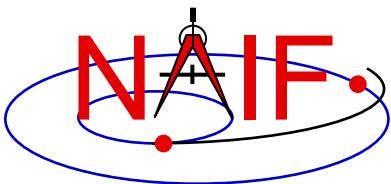


This simplified computation ignores the difference between the light time from Saturn to the Earth and the light time from the ring intercept point to the Earth.

The position and orientation of Saturn could be re-computed using the light time from Earth to the intercept; the intercept could be re-computed until convergence is attained.

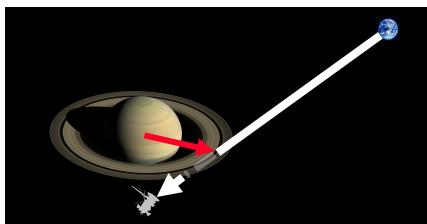
This computation is for the reception case, when radiation is received at the Earth at a given epoch “ET”.

- CALL [\*\*SPKEZR\*\*](#) to get light time corrected position of spacecraft as seen from Earth at time ET in J2000 reference frame SCVEC.
- CALL [\*\*SPKEZR\*\*](#) to get light time corrected position of center of Saturn at time ET as seen from Earth in J2000 frame SATCTR.
- CALL [\*\*PXFORM\*\*](#) to get rotation from Saturn body-fixed coordinates to J2000 at light time corrected epoch. The third column of this matrix gives the pole direction of Saturn in the J2000 frame SATPOL.
- CALL [\*\*NVP2PL\*\*](#) and use SATCTR and SATPOL to construct the ring plane RPLANE.
- CALL [\*\*INRYPL\*\*](#) to intersect the Earth-spacecraft vector SCVEC with the Saturn ring plane RPLANE to produce the intercept point X.
- CALL [\*\*VSUB\*\*](#) to get the position of the intercept with respect to Saturn XSAT (subtract SATCTR from X) and use [\*\*VNORM\*\*](#) to get the distance of XSAT from the center of Saturn.



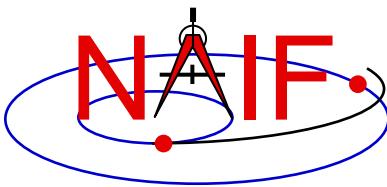
# Computing Ring Plane Intercepts-3

Navigation and Ancillary Information Facility



An  
alternate  
approach

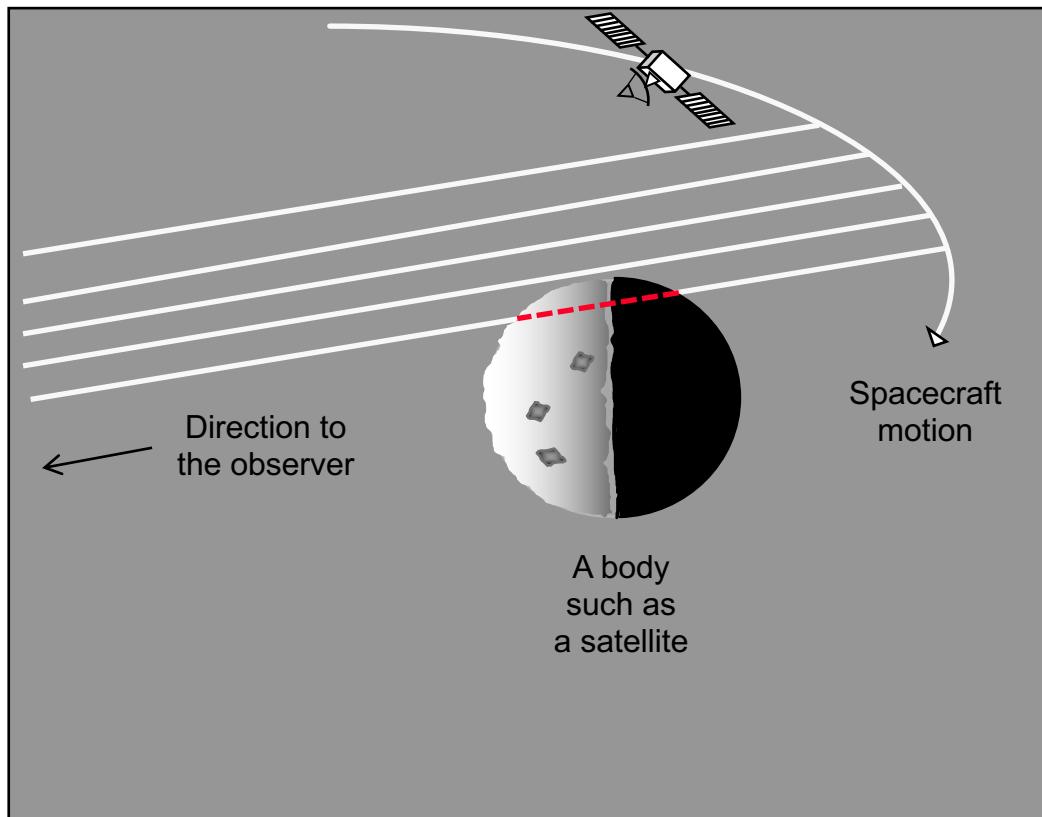
- Create a dynamic frame with one axis pointing from Earth to the light time corrected position of the Cassini orbiter. Use the CN correction for this position vector. (This gives us a frame in which the direction vector of interest is constant.)
- Temporarily change the radii of Saturn to make the polar axis length 1 cm and the equatorial radii 1.e6 km. This can be done either by editing the PCK or by calling [BODVCD](#) to fetch the original radii, then calling [PDPOOL](#) to set the kernel pool variable containing the radii to the new values. This flat ellipsoid will be used to represent the ring plane.
- Use [SINCPT](#) to find the intercept of the Earth-Cassini ray with the flat ellipsoid. Use the CN correction. SINCPT returns both the intercept in the IAU\_SATURN frame and the Earth-intercept vector. Use [VNORM](#) to get the distance of the intercept from Saturn's center.
- Restore the original radii of Saturn. If PDPOOL was used to update the radii in the kernel pool, use [PDPOOL](#) again to restore the radii fetched by BODVCD.

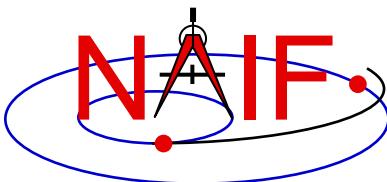


# Computing Occultation Events

Navigation and Ancillary Information Facility

- Determine when the spacecraft will be occulted by an object (such as a natural satellite) as seen from an observer (such as Earth).

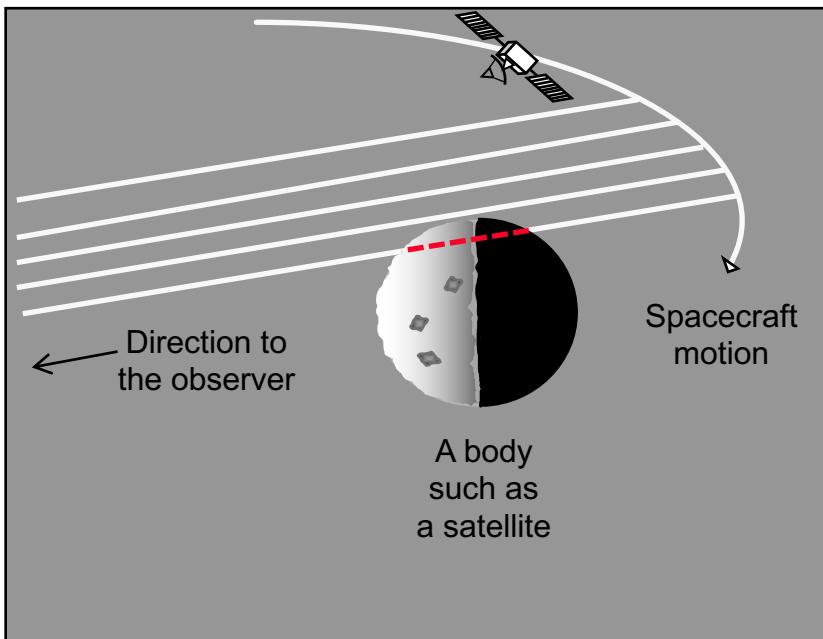


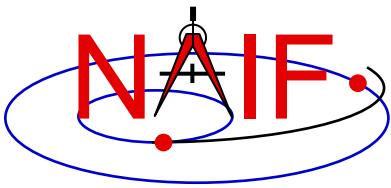


# Find Occultation Ingress/Egress

Navigation and Ancillary Information Facility

- **Select a start epoch, stop epoch and step size.**
  - Start and stop epochs can bracket multiple occultation events
  - Step size should be smaller than the shortest occultation duration of interest, and smaller than the minimum interval between occultation events that are to be distinguished, but large enough to solve the problem with reasonable speed.
- **Insert search interval into a SPICE window. This is the “confinement window.”**
- **CALL [GFOCLT](#) to find occultations, if any. The time intervals, within the confinement window, over which occultations occur will be returned in a SPICE window.**
  - GFOCLT can treat targets as ellipsoids, DSK shapes, or points (but at least one must be an ellipsoid or DSK shape, and DSK shapes can be used only with point targets).
  - GFOCLT can search for different occultation or transit geometries: full, partial, annular, or “any.”



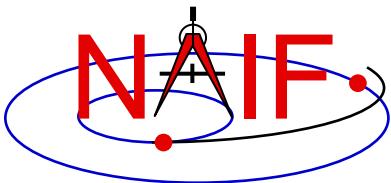


---

Navigation and Ancillary Information Facility

# Other Useful Functions

January 2020

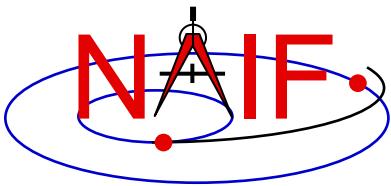


# Topics

---

Navigation and Ancillary Information Facility

- **Overview**
- **Language-specific status**
- **File Operations**
- **String Manipulation**
- **Searching, Sorting and Other Array Manipulations**
- **Windows**
- **Symbol Tables**
- **Sets and Cells**
- **Constants and Unit Conversion**
- **Numerical Functions**



# Overview

---

Navigation and Ancillary Information Facility

- The routines described in this tutorial originated in the Fortran version of the the SPICE Toolkit.
- Many, but not all, of these routines have implementations in the C, IDL, and MATLAB Toolkits.
- The descriptions include a language “identifier” or set of identifiers prefixed to the routine’s name to indicate which Toolkit language(s) include that routine.
  - [F] available in Fortran (SPICELIB)
  - [C] available in C (CSPICE)
  - [I] available in IDL (Icy)
  - [M] available in MATLAB (Mice)
- NAIF adds interfaces to the CSPICE, Icy and Mice Toolkits as needed or when requested by a customer.
- CSPICE, Icy and Mice do not need all of the functionality implemented in the Fortran Toolkit.
- NAIF does not attempt to keep track of which functions are implemented in 3<sup>rd</sup> party toolkits such as SpiceyPy.



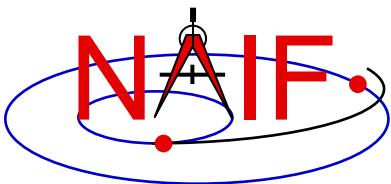
# Text I/O (1)

Navigation and Ancillary Information Facility

- **Text files provide a simple, human readable mechanism for sharing data.**
- **The Toolkit contains several utility routines to assist with the creation and parsing of text, and with the reading and writing of text files.**
  - [F,C] RDTEXT: read a line of text from a text file\*
  - [F] TOSTDO: write a line of text to standard output
  - [F,C] PROMPT: display a prompt, wait for and return user's response
  - [F] TXTOPN: open a new text file returning a logical unit
  - [F] WRITLN: write a line of text to the file attached to a logical unit.



\* The text file must be in native text format for your computer



# Text I/O (2)

## Navigation and Ancillary Information Facility

```
CALL PROMPT ( 'Filename? ', NAME )
CALL TOSTDO ( 'You specified the file: '// NAME )
```

Now that we have the filename, read  
and process its contents

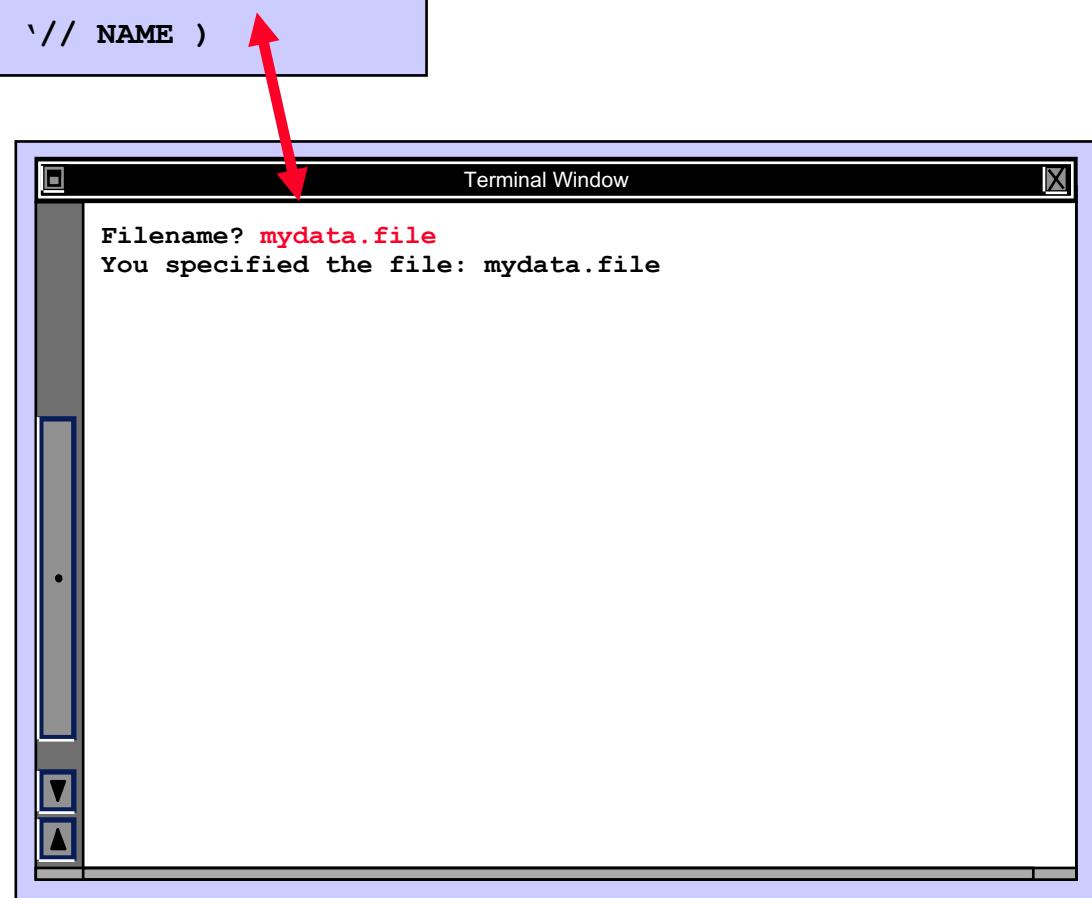
```
CALL RDTEXT ( NAME, LINE, EOF )

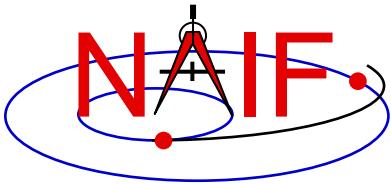
DO WHILE ( .NOT. EOF )

    process the line just read

    CALL RDTEXT ( NAME, LINE, EOF )

END DO
```



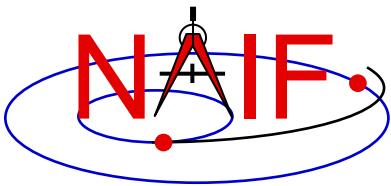


# File Operations

---

Navigation and Ancillary Information Facility

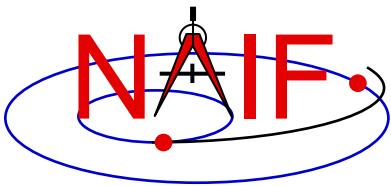
- **Logical unit management - Fortran specific**
  - [F] RESLUN: (reserve logical unit) prohibits SPICE systems from using specified units.
  - [F] FRELUN: (free logical unit) places “reserved” units back into service for SPICE.
  - [F] GETLUN: (get logical unit) locates an unused, unreserved logical unit.
- **Determin whether or not a file exists**
  - [F,C,I] EXISTS
- **Delete an existing file**
  - [F] DELFIL



# String Manipulation - Parsing (1)

Navigation and Ancillary Information Facility

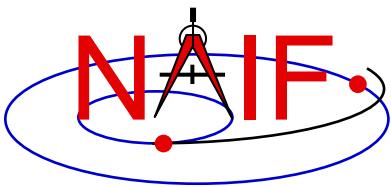
- **Breaking apart a list**
  - [F,C,I] LPARSE: parses a list of items delimited by a single character.
  - [F,C] LPARSM: parses a list of items separated by multiple delimiters.
  - [F] NEXTWD: returns the next word in a given character string.
  - [F] NTHWD: returns the nth word in a string and the location of the word in the string.
  - [F,C] KXTRCT: extracts a substring starting with a keyword.
- **Removing unwanted parts of a string**
  - [F,C,I] CMPRSS: compresses a character string by removing instances of more than N consecutive occurrences of a specified character.
  - [F] ASTRIP: removes a set ASCII characters from a string.
  - [F] REMSUB: removes a substring from a string.



# String Manipulation - Parsing (2)

Navigation and Ancillary Information Facility

- Locating substrings
  - [F] LTRIM, RTRIM: return the location of the leftmost or rightmost non-blank character.
  - [F,C] POS, CPOS, POSR, CPOSR, NCPOS, NCPOSR: locate substring or member of specified character set searching forward or backward.
- Pattern matching
  - [F,C,I] MATCHI: matches a string against a wildcard template, case insensitive.
  - [F,C,I] MATCHW: matches a string against a wildcard template, case sensitive.
- Extracting numeric and time data
  - [F] NPARSD, NPARI, DXTRCT, TPARTV
  - [F,C,I] PRSDP, PRSINT, TPARSE
- Heavy duty parsing
  - [F] SCANIT



# String Manipulation - Parsing (3)

Navigation and Ancillary Information Facility

'a dog, a cat, and a cow'

lparse or  
lparsm

'a dog'  
'a cat'  
'and a cow'

Split on a comma

'Remove extra spaces'

cmprss

'Remove extra spaces'

'Green eggs and ham'

'the cat in the hat'

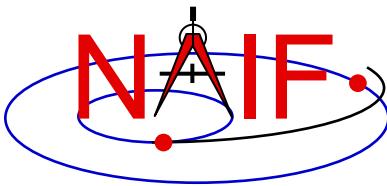
'how the grinch stole Christmas'

matchi( \*g\*)

'green eggs and ham'

'how the grinch stole Christmas'

Match any string containing a 'g'



# String Manipulation - Creating (1)

Navigation and Ancillary Information Facility

- **Fill in the “Blank”**

- [F,C] REPMC: Replace a marker with a character string.

```
CALL REPMC ( 'The file was: #', '#', 'foo.bar', OUT )
```

**OUT has the value “The file was: foo.bar”**

- [F,C] REPML: Replace a marker with an integer.

```
CALL REPML ( 'The value is: #', '#', 7, OUT )
```

**OUT has the value “The value is: 7”**

- [F,C] REPMD: Replace a marker with a double precision number.

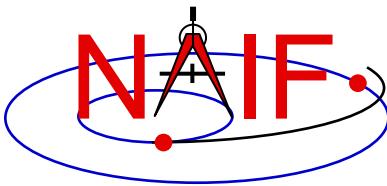
```
CALL REPMD ( 'The value is: #', '#', 3.141592654D0, 10, OUT )
```

**OUT has the value “The value is: 3.141592654E+00”**

- [F,C] REPMOT: Replace a marker with the text representation of an ordinal number.

```
CALL REPMOT ( 'It was the # term.', '#', 'L', 2, OUT )
```

**OUT has the value “It was the second term.”**



# String Manipulation - Creating (2)

Navigation and Ancillary Information Facility

- **Fill in the “Blank” (cont.)**
  - **[F,C] REPMCT:** Replace a marker with the text representation of a cardinal number.

CALL REPMCT ( 'Hit # errors.', '#', 6, 'L', OUT )

OUT becomes ‘Hit six errors.’

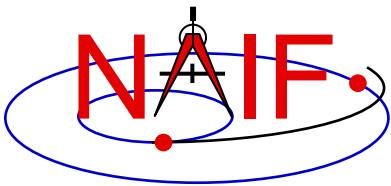
- **Numeric Formatting**

- **[F] DPFMT:** Using a format template, create a formatted string that represents a double precision number

CALL DPFMT ( PI(), 'xxx.yyyy', OUT )

OUT becomes ‘ 3.1416’

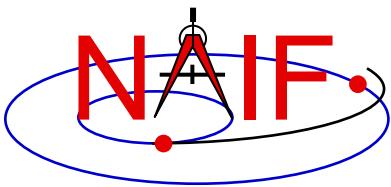
- **[F] DPSTR, INTSTR, INTXT, INTORD**



# String Manipulation - Creating (3)

Navigation and Ancillary Information Facility

- **Time formatting**
  - [F,C,I,M] TPICTR: Given a sample time string, create a time format picture suitable for use by the routine TIMOUT.
  - [F,C,I,M] TIMOUT: Converts an input epoch to a character string formatted to the specifications of a user's format picture.
- **Changing case**
  - [F,C,I] UCASE: Convert all characters in string to uppercase.
  - [F,C,I] LCASE: Convert all characters in string to lowercase.
- **Building strings**
  - [F] SUFFIX: add a suffix to a string
  - [F] PREFIX: add a prefix to a string
  - [F] LJUCRS: left-justify, upper-case and compress

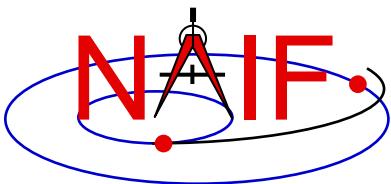


# Searching, Sorting and Other Array Manipulations (1)

---

Navigation and Ancillary Information Facility

- **Sorting arrays**
  - [F,C] SHELLC, SHELLI, SHELLD, ORDERI, ORDERC, ORDERD, REORDC, REORDI, REORDD, REORDL
- **Searching ordered arrays**
  - [F,C] BSRCHC, BSRCHI, BSRCHD, LSTLEC, LSTLEI, LSTLED, LSTLTC, LSTLTI, LSTLTD, BSCHOI
- **Searching unordered arrays**
  - [F,C] ISRCHC, ISRCHI, ISRCHD, ESRCHC
- **Moving portions of arrays**
  - [F] CYCLAC, CYCLAD, CYCLAI
- **Inserting and removing array elements**
  - [F] INSLAC, INSLAD, INSLAI, REMLAC, REMLAD, REMLAI



# Searching, Sorting and Other Array Manipulations (2)

Navigation and Ancillary Information Facility

Body	A.U. <sup>1</sup>
sun	00.0
mercury	00.455
venus	00.720
earth	00.983
mars	01.531
jupiter	05.440
saturn	09.107
uranus	20.74
neptune	30.091
pluto	31.052

**orderc**

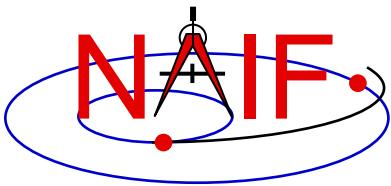
04
06
05
02
09
10
07
01
08
03

**reordc,d**

Sorted Body	A.U. <sup>1</sup>
earth	00.983
jupiter	05.440
mars	01.531
mercury	00.455
neptune	30.091
pluto	31.052
saturn	09.107
sun	00.000
uranus	20.74
venus	00.720

Vector of “Body” indices representing the list sorted in alphabetical order.

<sup>1</sup> Distance in A.U. at Jan 01, 2006.

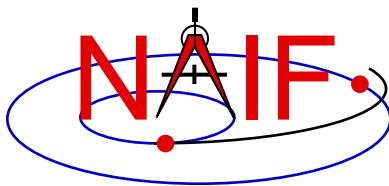


# Windows

---

Navigation and Ancillary Information Facility

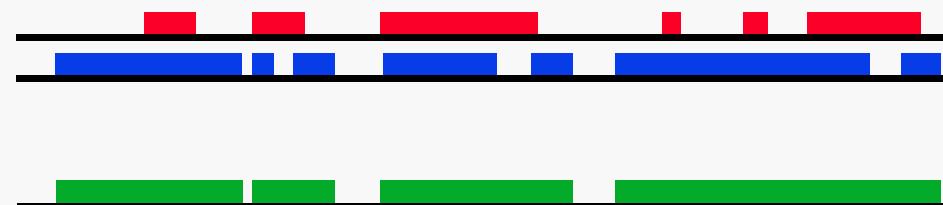
- A SPICE window is a list of disjoint intervals arranged in ascending order.
- An interval is specified by a pair of double precision numbers, with the second greater than or equal to the first.
- The Toolkit contains a family of routines for creating windows and performing “set arithmetic” on them.
- SPICE windows are frequently used to specify intervals of time when some set of user constraints are satisfied.
  - Let window *NotBehind* contain intervals of time when Cassini is not behind Saturn as seen from earth.
  - Let window *Goldstone* contain intervals of time when Cassini is above the Goldstone horizon.
  - Cassini can be tracked from Goldstone during the intersection of these two windows (*Track = NotBehind \* Goldstone*).
- See *windows.req* for more information.



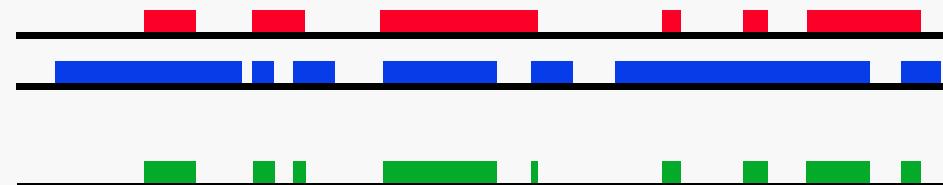
# Windows Math

Navigation and Ancillary Information Facility

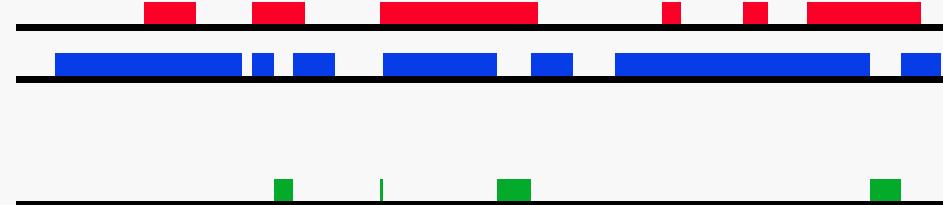
Union

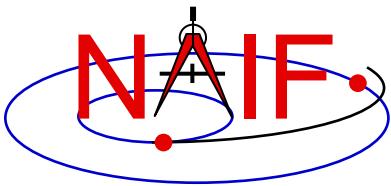


Intersection



Difference



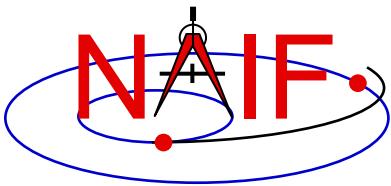


# Symbol Tables

---

Navigation and Ancillary Information Facility

- **SPICELIB (Fortran) supports the use of associative arrays/hashes through the use of an abstract data type called symbol tables.**
  - These are used to associate a set of names with collections of associated values.
  - Values associated with a name are exclusively character, exclusively integer or exclusively double precision.
  - Routines to manipulate a symbol table have the form **SY\*\*\*<T>** where **<T>** is the data type of the values (C, D, or I).
- **Operations include:**
  - Insert a symbol
  - Remove a symbol
  - Push/Pop a value onto/off of the list of values associated with a symbol
  - Fetch/Sort values associated with a symbol
- **See *symbols.req* for more information.**

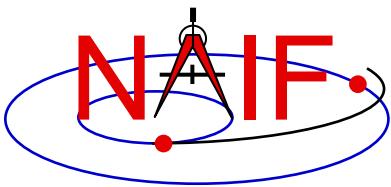


# Sets and Cells (1)

---

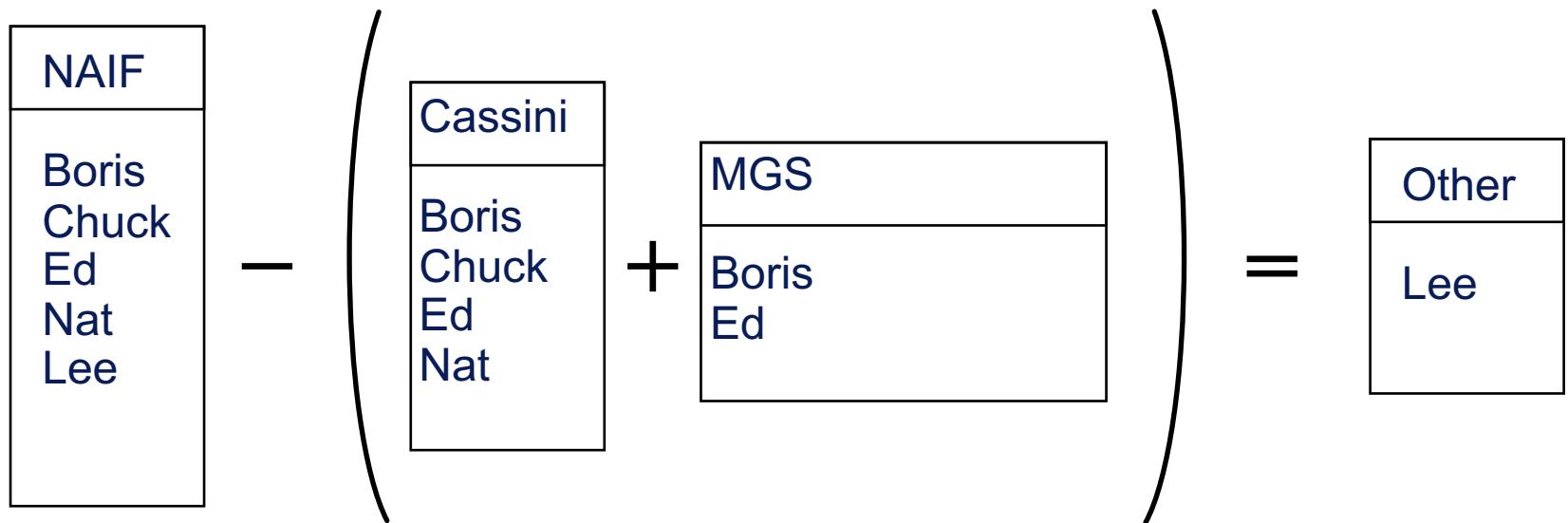
Navigation and Ancillary Information Facility

- **Cells are arrays that “know” how many addresses are available for use and how many are currently used.**
  - Routines that use cells typically have simpler interfaces than routines that use arrays.
  - See *cells.req* for more information.
- **Sets are cells that contain no duplicate elements and whose elements are ordered in ascending order.**
  - Two Sets can be: intersected, unioned, differenced, differenced symmetrically (union - intersection)
  - See *sets.req* for more information.
- **Language support for sets and cells**
  - Double Precision, Integer, and Character string cell types are supported in the Fortran and C Toolkits.
  - Double Precision and Integer cell types are supported in the IDL Toolkits (Icy).
  - Sets and cells aren’t currently needed in the MATLAB Toolkits (Mice) since MATLAB itself supports set math.

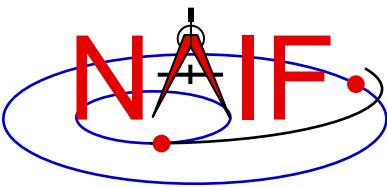


# Sets and Cells (2)

Navigation and Ancillary Information Facility



```
CALL UNIONC ( CASSINI, MGS,           PROJECTS)  
CALL DIFFC  ( NAIF,      PROJECTS, OTHER )
```

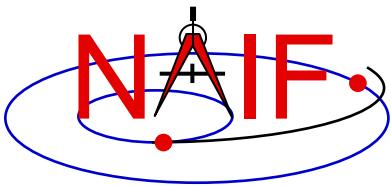


# Constants and Unit Conversion

---

Navigation and Ancillary Information Facility

- **Constants are implemented in the Toolkit as functions.**
  - Thus the changing of a constant by NAIF requires only relinking by the Toolkit user—not recompiling.
    - » Users should NOT change constant functions in the Toolkit.
- **System Constants**
  - [F,C,I,M] DPMIN, DPMAX, INTMIN, INTMAX
- **Numeric Constants**
  - [F,C,I,M] PI, HALFPI, TWOPI, RPD (radians/degree), DPR(degrees/radian)
- **Physical Constants**
  - [F,C,I,M] CLIGHT, SPD, TYEAR, JYEAR
- **Epochs**
  - [F,C,I,M] J2000, J1950, J1900, J2100, B1900, B1950
- **Simple Conversion of Units**
  - [F,C,I,M] CONVRT

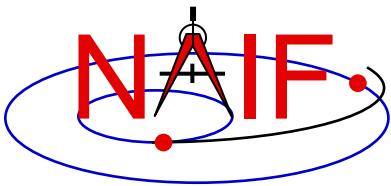


# Numerical Functions (1)

---

Navigation and Ancillary Information Facility

- Several routines are provided to assist with numeric computations and comparisons.
- Functions
  - [F] DCBRT: cube root
  - Hyperbolic Functions:
    - » [F] DACOSH, DATANH
  - Polynomial Interpolation and Evaluation:
    - » [F] LGRESP, LGRINT, LGRIND, POLYDS, HRMESP, HRMINT
  - Chebyshev Polynomial Evaluation:
    - » [F] CHBDER, CHBVAL, CHBINT, CHBIGR

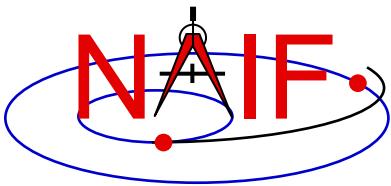


# Numerical Functions (2)

---

Navigation and Ancillary Information Facility

- **Numerical Decisions**
  - Same or opposite sign (Boolean):
    - » [F] **SMSGND**, **SMSGNI**, **OPSGND**, **OPSGNI**
  - Force a value into a range (bracket):
    - » [F,C] **BRCKTD**, **BRCKTI**
  - Determine parity of integers (Boolean):
    - » [F] **ODD**, **EVEN**
  - Truncate conditionally:
    - » [F] **EXACT**
- **Arithmetic**
  - Greatest common divisor:
    - » [F] **GCD**
  - Positive remainder:
    - » [F] **RMAINI**, **RMAIND**



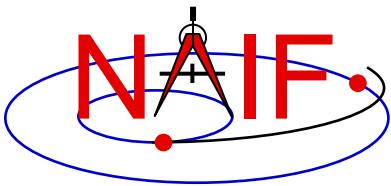
---

Navigation and Ancillary Information Facility

# SPICE Geometry Finder (GF) Subsystem

**Searching for times when specified  
geometric conditions occur**

**January 2020**



# Topics

---

Navigation and Ancillary Information Facility

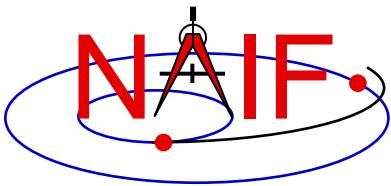
- **GF Subsystem Overview**
- **SPICE Windows**
- **GF Search Examples**
- **Geometric Search Types and Constraints**
- **More Details**
  - Root finding, including step size
  - Workspace
- **An Example**



---

Navigation and Ancillary Information Facility

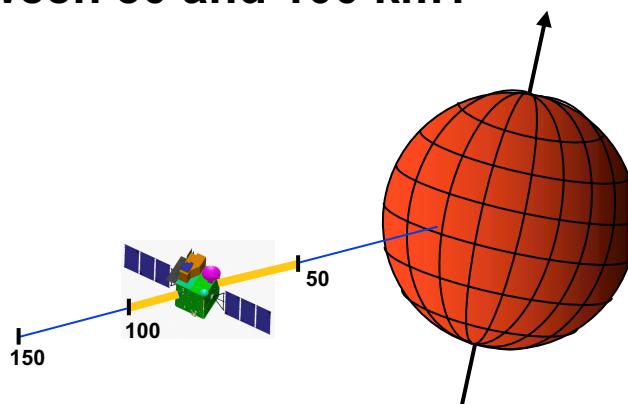
# GF Subsystem Overview

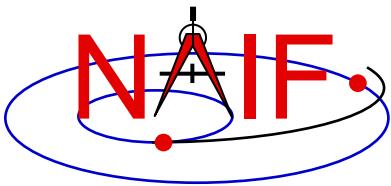


# Purpose

Navigation and Ancillary Information Facility

- Much SPICE software computes a geometry parameter at a given time,  $t$ , i.e.  $x = f(t)$ .
  - Example: on 2011 MAR 30 14:57:08, what is the spacecraft's altitude above Mars?
- The Geometry Finder subsystem does the inverse: it finds times when specified geometric events occur.
  - Example: within some time bounds, when is the spacecraft's altitude between 50 and 100 km?



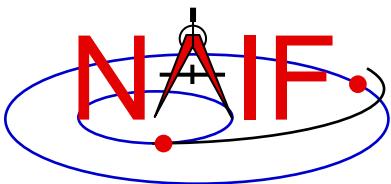


# Some Examples

---

Navigation and Ancillary Information Facility

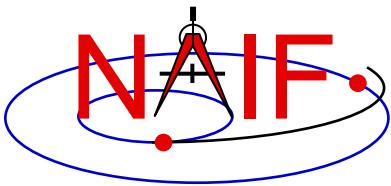
- **The SPICE Geometry Finder (GF) subsystem finds times when specified geometric events occur.**
  - A “geometric event” is an occurrence of a given geometric quantity satisfying a specified condition. For example:
    - » Mars Express distance from Mars is at a local minimum (periapse)
    - » Elevation of the Cassini orbiter is above a given threshold angle as seen from DSS-14
    - » Titan is completely occulted by Saturn
    - » The Saturn phase angle as seen by the Cassini orbiter is 60 degrees
  - Each GF search is conducted over a user-specified time window, called the confinement window.
    - » A “time window” is a union of time intervals.
  - The result of a GF search is the time window over which the specified condition is met.



# Types of GF APIs

Navigation and Ancillary Information Facility

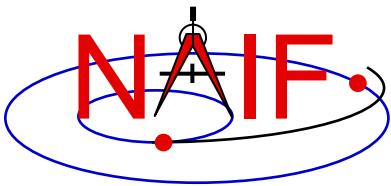
- GF provides two primary types of event-finding APIs
  - Boolean: a geometric condition (an event) is true or false
    - » Example: Phobos is occulted by Mars
    - » Example: Vesta is not in the OSIRIS instrument's field of view
  - We also call these binary conditions
  - Numeric: a geometric quantity has a given value, is within a given range or has achieved a local or global maximum or minimum
    - » Example: spacecraft altitude is between X and Y km above the surface
    - » Example: angular separation of Titan from Saturn has reached the maximum value



# GF High-Level API Routines

Navigation and Ancillary Information Facility

- The GF subsystem provides the following high-level API routines; these search for events involving the respective geometric quantities listed below
  - GFDIST: observer-target distance
  - GFILUM: illumination angles
  - GFOCLT: occultations or transits
  - GFPA: phase angle
  - GFPOSC: position vector coordinates
  - GFRFOV: ray is contained in an instrument's field of view
  - GFRR: observer-target range rate
  - GFSEP: target body angular separation
  - GFSNTC: ray-body surface intercept coordinates
  - GFSUBC: sub-observer point coordinates
  - GFTFOV: target body appears in an instrument's field of view
  - GFUDB: user-defined Boolean quantity (only Fortran, C and JNI)
  - GFUDS: user-defined scalar quantity (only Fortran, C and JNI)



# The SPICE Window

Navigation and Ancillary Information Facility

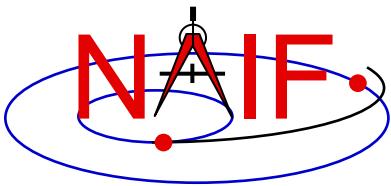
- The high-level GF routines return a search result as a SPICE window. This window specifies intervals of time when the user's constraints are satisfied.
- In simple terms, one can describe a SPICE window as:
  - A span of time defined by a start time and an end time, containing a list of disjoint intervals arranged in ascending order.
  - Within that time span, a time-ordered sequence of zero or more time intervals each having zero or non-zero length
    - » An interval is specified by a pair of double precision numbers, with the second greater than or equal to the first.
    - » A zero-length interval is often called a “singleton”



$T_{\text{start}}$

A SPICE Window

$T_{\text{stop}}$

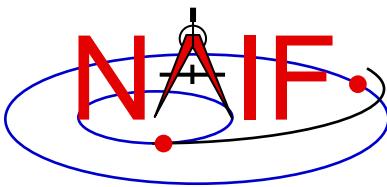


# SPICE Windows Operations

---

Navigation and Ancillary Information Facility

- SPICE provides routines to:
  - compute unions, intersections, and differences of windows
  - contract each interval within a window ...
    - » by increasing the left endpoint and decreasing the right endpoint
- These functions allow one to search for multi-condition events
- See the next page for an example



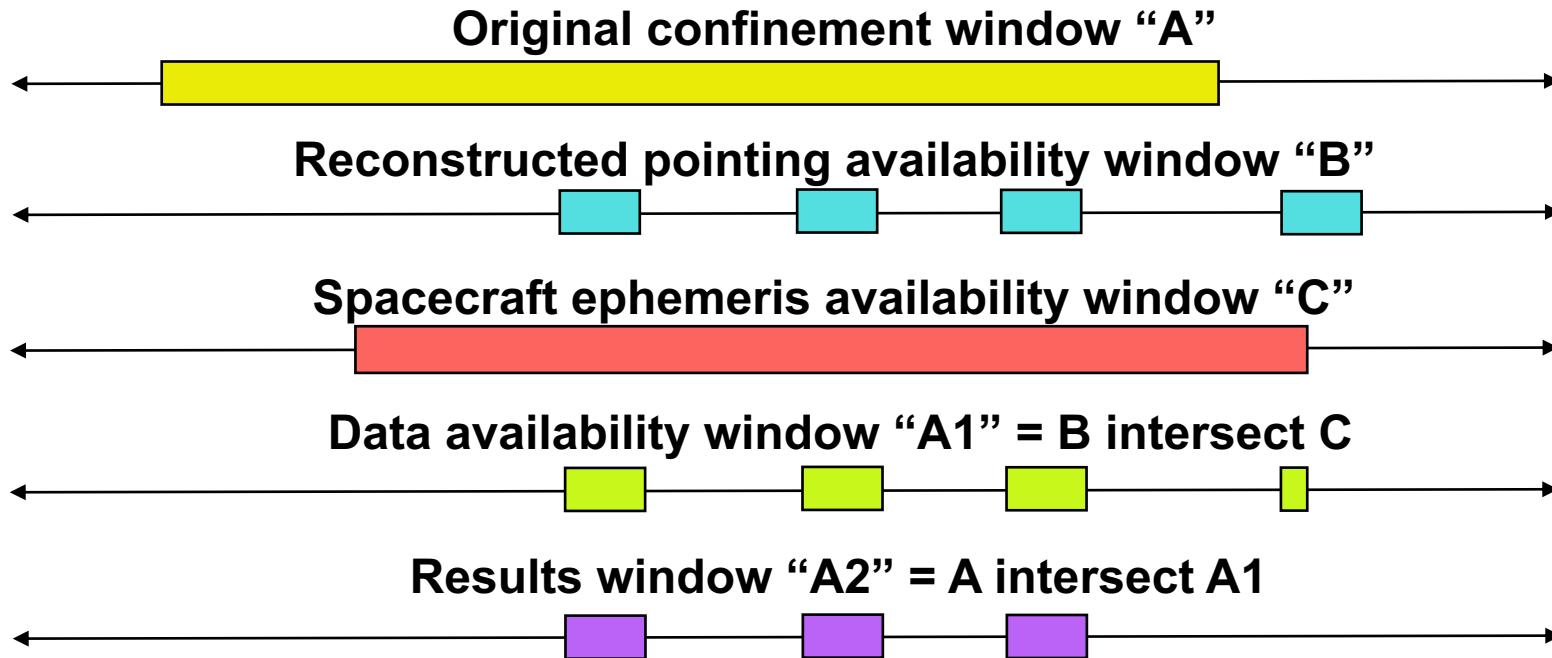
# Example of Window Operations

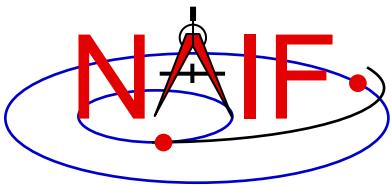
Navigation and Ancillary Information Facility

Given an initial confinement window, A, determine times when CK and SPK data are available within it.

Use CKCOV and SPKCOV to find CK and SPK availability windows, B and C. Use window intersection to obtain the results window.

Contract the CK window slightly to avoid round-off problems. Contract the SPK window by a few seconds if discrete differentiation is used by search algorithms (e.g. for acceleration or for “is function decreasing?” tests).





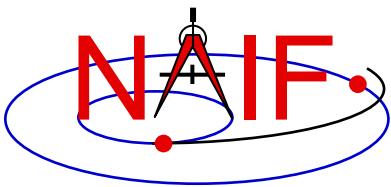
# Using Time Windows in GF

Navigation and Ancillary Information Facility

- GF uses a SPICE window to:
  - confine the time bounds over which your search is to take place

**Search Confinement Window**

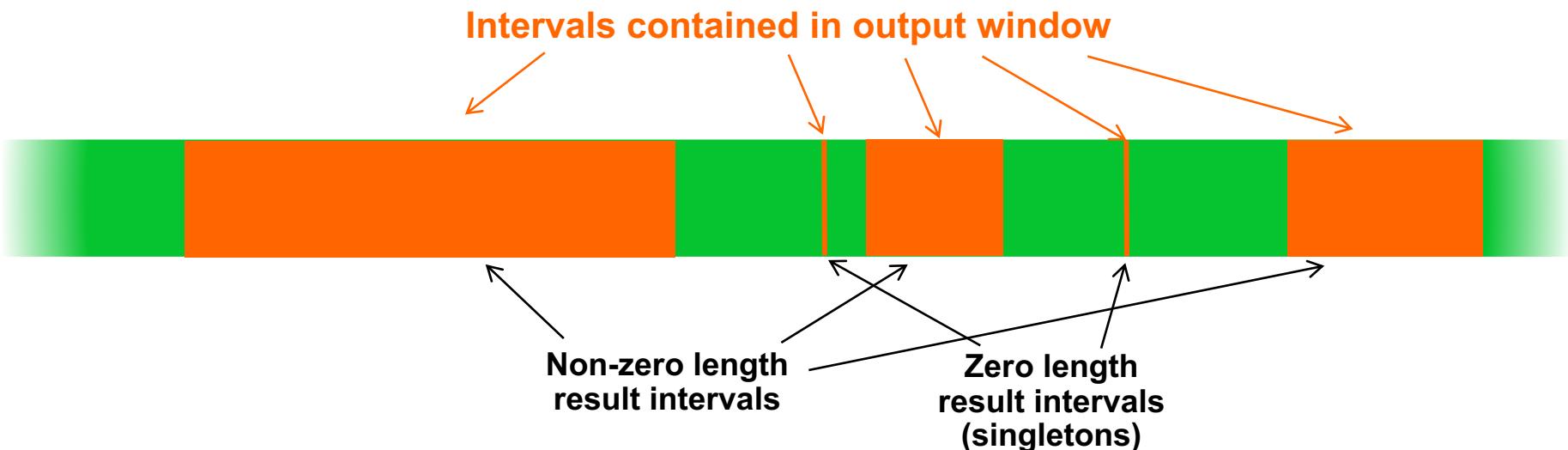


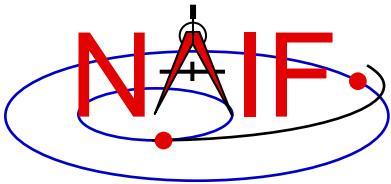


# Using Time Windows in GF

Navigation and Ancillary Information Facility

- GF uses SPICE windows for input and output
  - **Input:** confine the time bounds over which your search is to take place
  - **Output:** contain the time intervals that meet the search criteria
    - » There may be none, one or multiple result intervals
    - » The result intervals can be of non-zero or zero length
      - A zero-length interval is simply an epoch—an instant in time



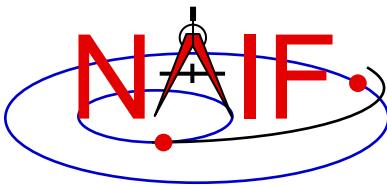


# Cascading Search Using Multiple SPICE Windows

---

Navigation and Ancillary Information Facility

- The result window (the output) from one search can be used as the confinement window (the input) for a subsequent search.
  - This is often a convenient and efficient way of performing searches for times when multiple constraints are met.
  - This technique can be used to accelerate searches in cases where an initial, fast search can be performed to produce a small confinement window for a second, slower search.
    - » See the next chart and the example program “CASCADE” in the Geometry Finder Required Reading document



# Cascading Search Example

Navigation and Ancillary Information Facility

**Example: accelerate a solar occultation search.**

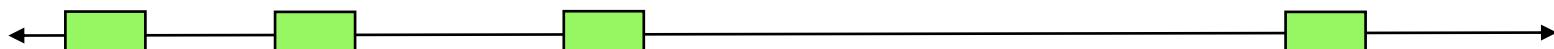
First search for times when the angular separation of the Sun and Moon, as seen from an earth station, is less than 3 degrees.

Use the result window of the angular separation search as the confinement window of an occultation search.

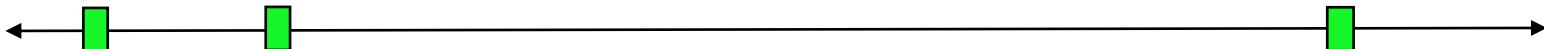
Because the angular separation search is much faster than would be the occultation search on the original confinement window, the total search time is greatly reduced.



**Original confinement window (“A”)**



**Result of angular separation search: second confinement window (“B”)**



**Window “C”: result of occultation search performed on window “B”**

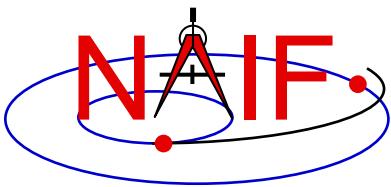


# GF Documentation

---

Navigation and Ancillary Information Facility

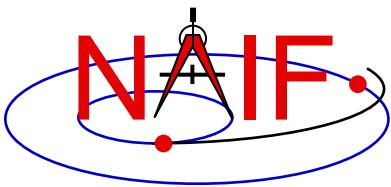
- The GF module headers contain complete example programs for each GF API routine
- The GF Required Reading document (`gf.req`) contains lots of details
- Documentation on SPICE windows:
  - The WINDOWS Required Reading `windows.req`
  - The Other Functions tutorial
  - API documentation for SPICE window routines



---

Navigation and Ancillary Information Facility

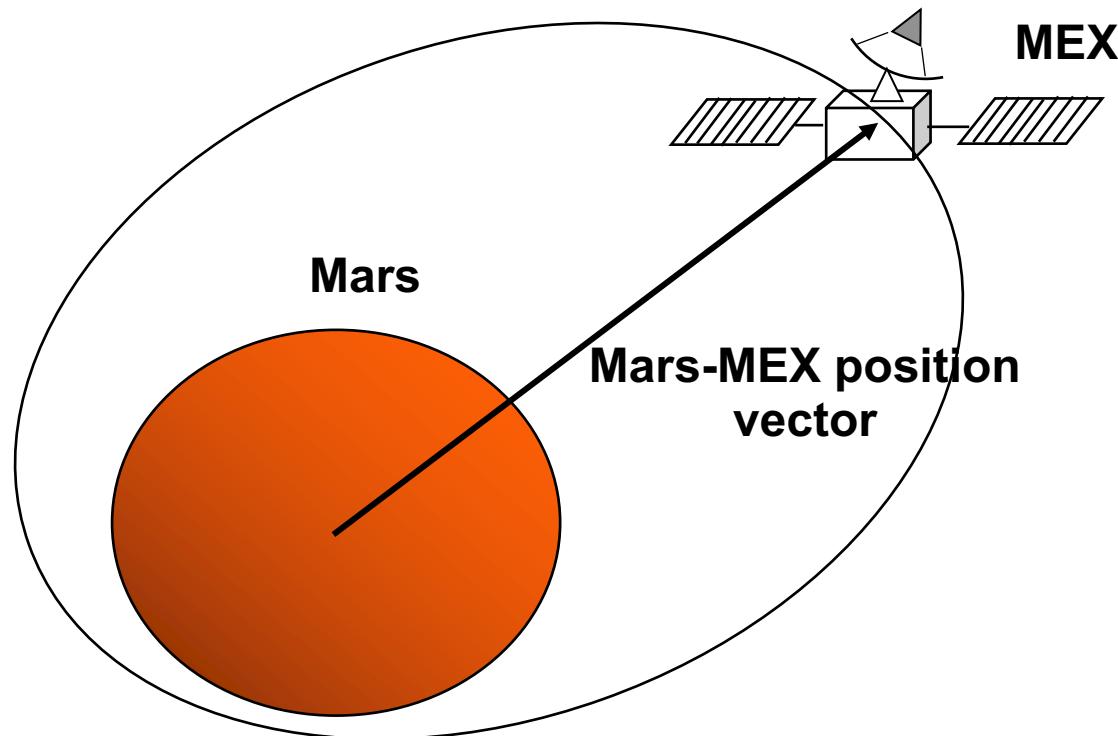
# GF Search Examples



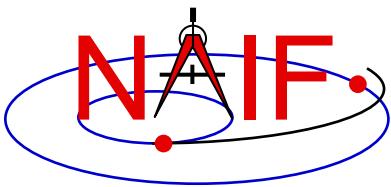
# Distance is Local Maximum (or Minimum)

Navigation and Ancillary Information Facility

Find the times of apoapse of the Mars Express Orbiter (MEX)



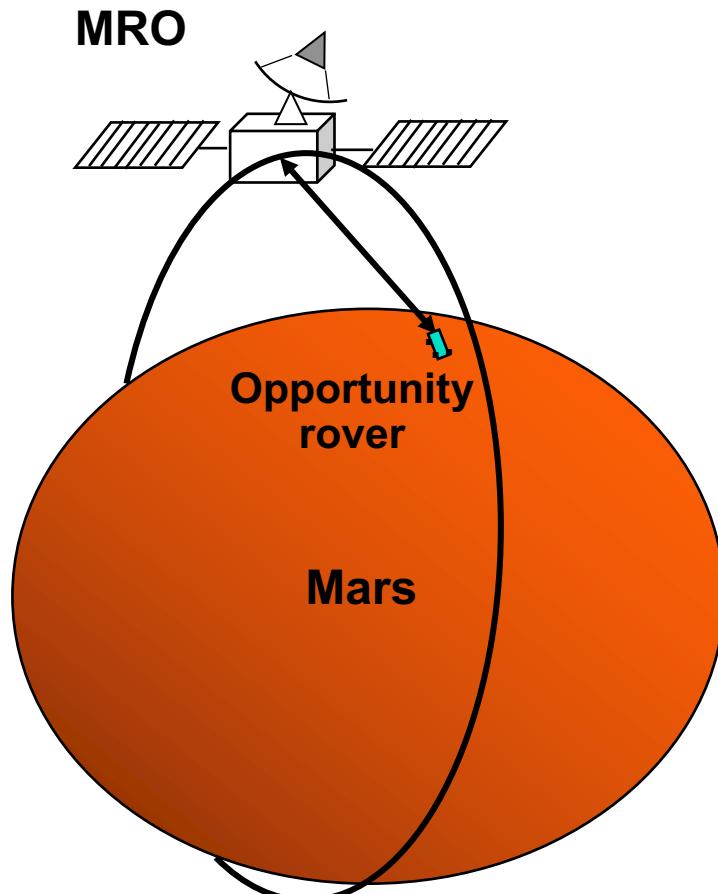
API: GFDIST



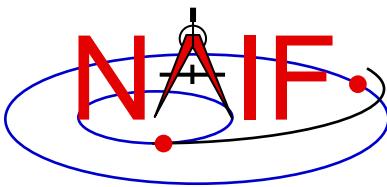
# Distance Within a Range

Navigation and Ancillary Information Facility

Find the time periods when the Mars Reconnaissance Orbiter (MRO) is within 500km of the Opportunity rover.



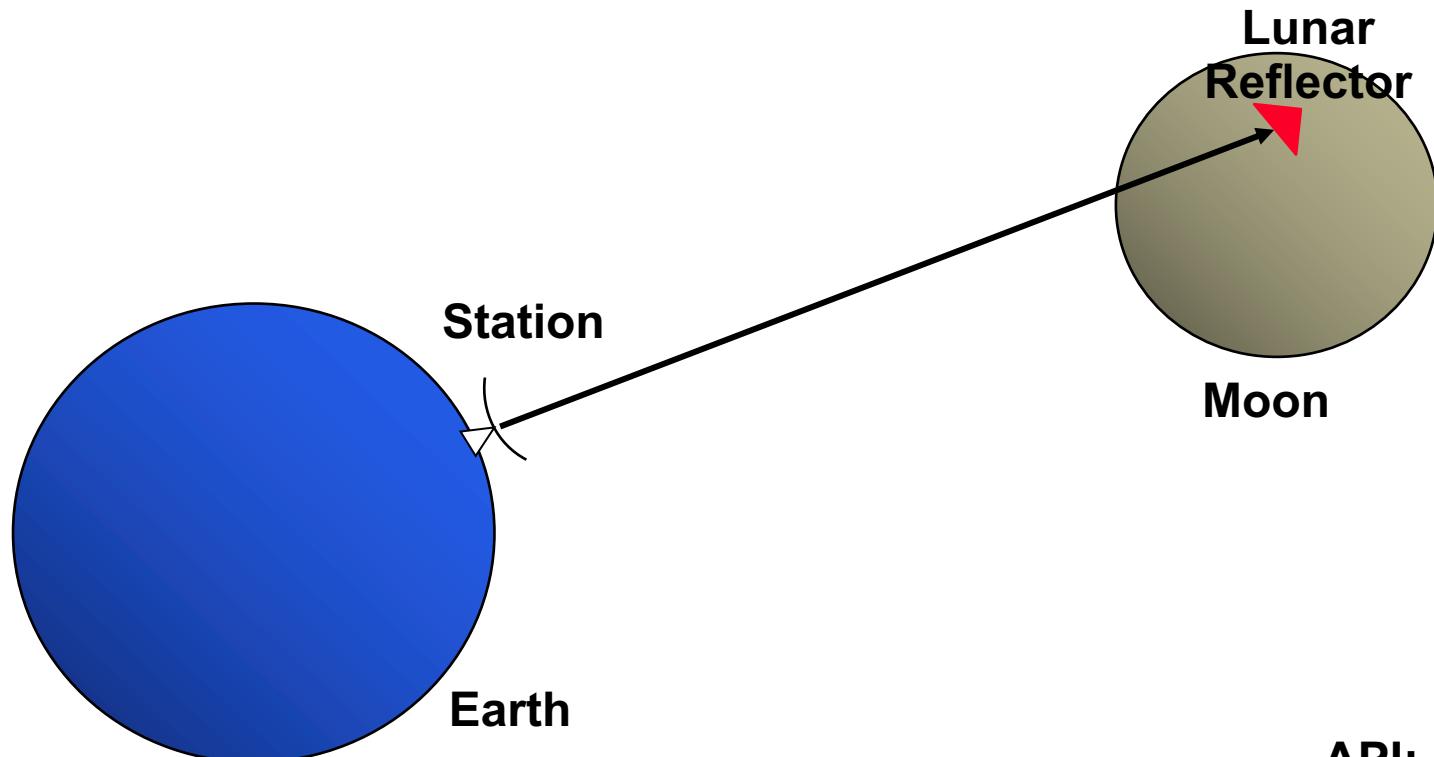
API: GFDIST



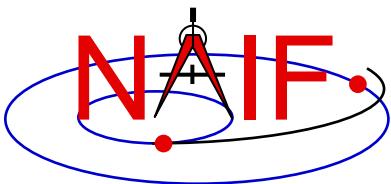
# Range Rate Extremum

Navigation and Ancillary Information Facility

Find the time periods when the range rate of a lunar reflector, as seen by an Earth station, attains an absolute extremum.



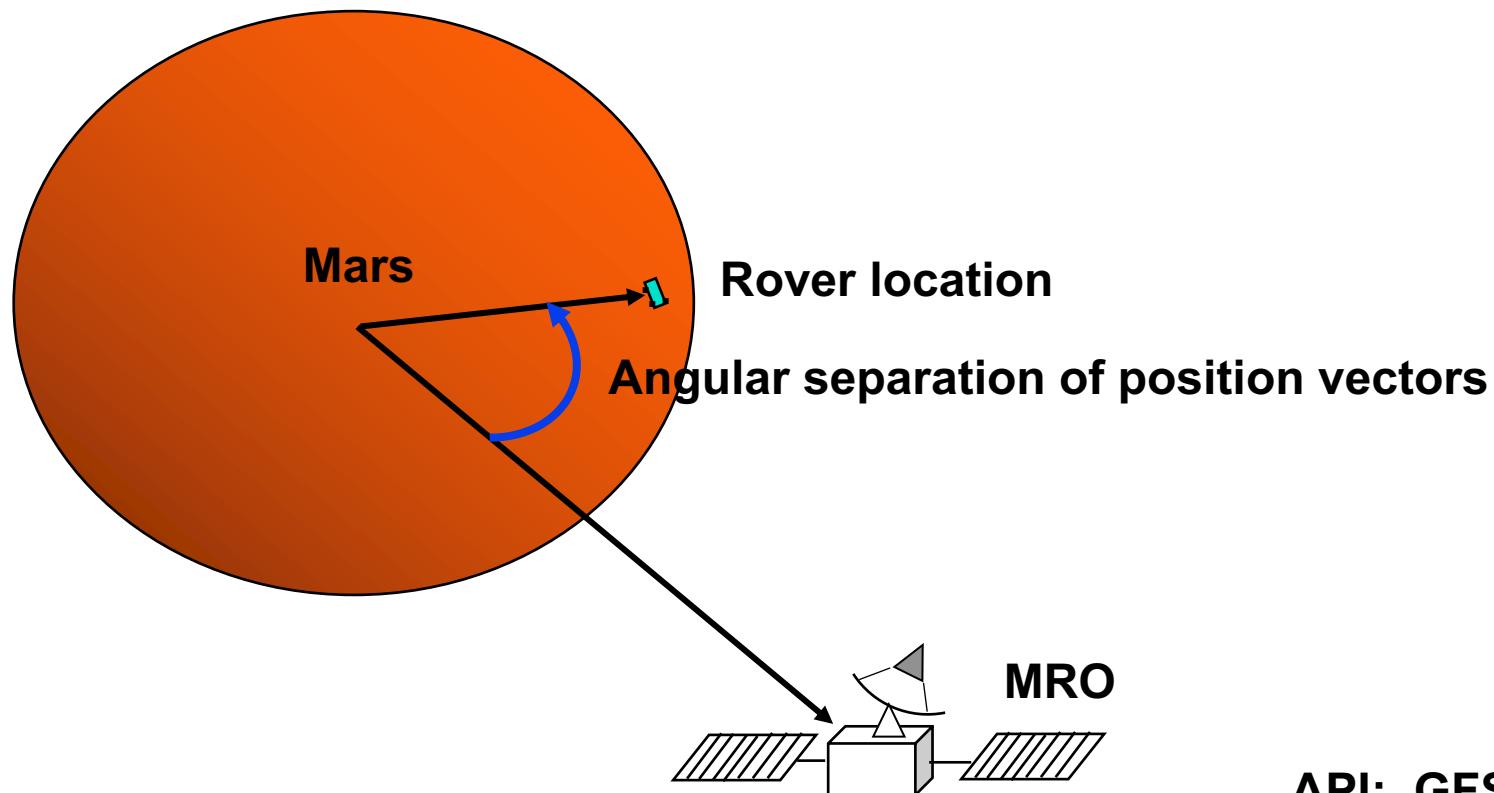
API: GFRR



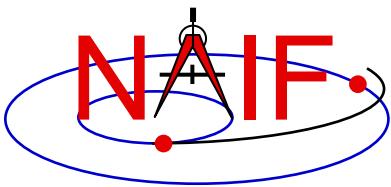
# Angular Separation Inequality Search -1

Navigation and Ancillary Information Facility

Find the time periods when the angular separation of the Mars-to Mars Reconnaissance Orbiter (MRO) and Mars-to-Opportunity Rover position vectors is less than 3 degrees. Both targets are modeled as points.



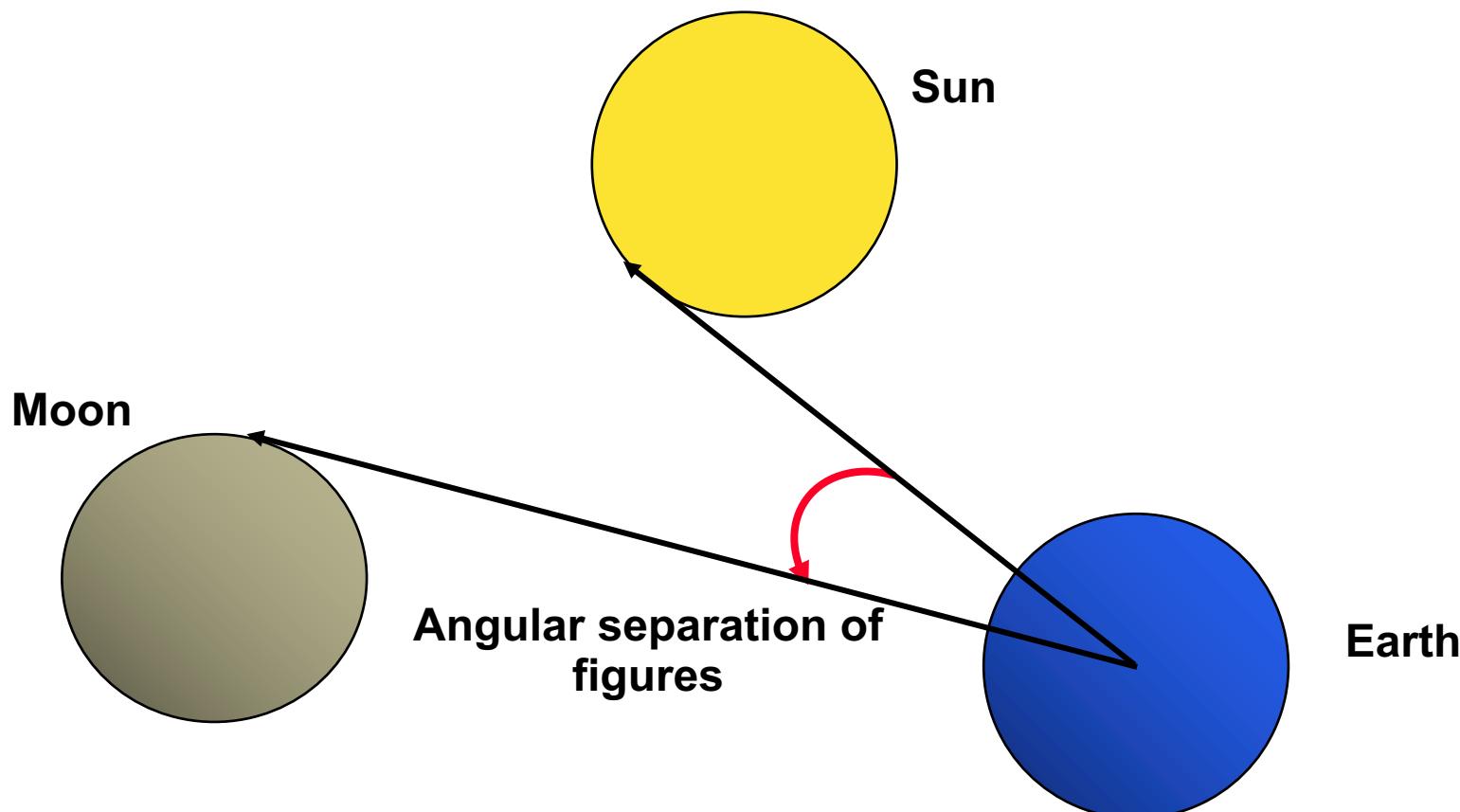
API: GFSEP



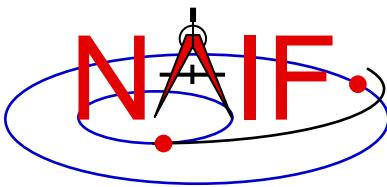
# Angular Separation Inequality Search -2

Navigation and Ancillary Information Facility

Find the time periods when the angular separation of the figures of the Moon and Sun, as seen from the Earth, is less than 1 degree. Both targets are modeled as spheres.



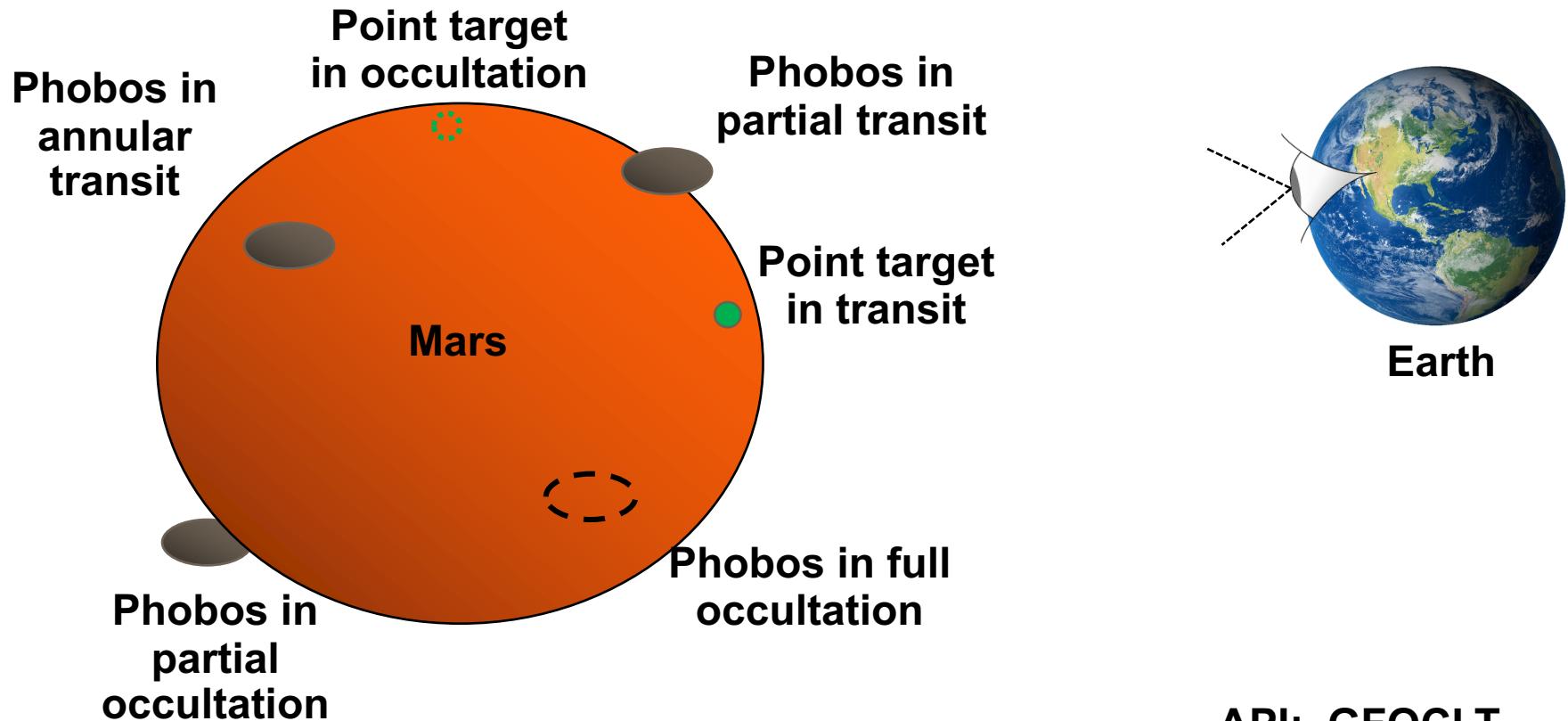
API: GFSEP



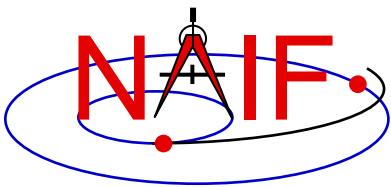
# Occultation/Transit Search

Navigation and Ancillary Information Facility

Find the ingress and egress times of an occultation of Phobos by Mars, as seen from Earth. Phobos and Mars are modeled as triaxial ellipsoids; a spacecraft is modeled as a point target.



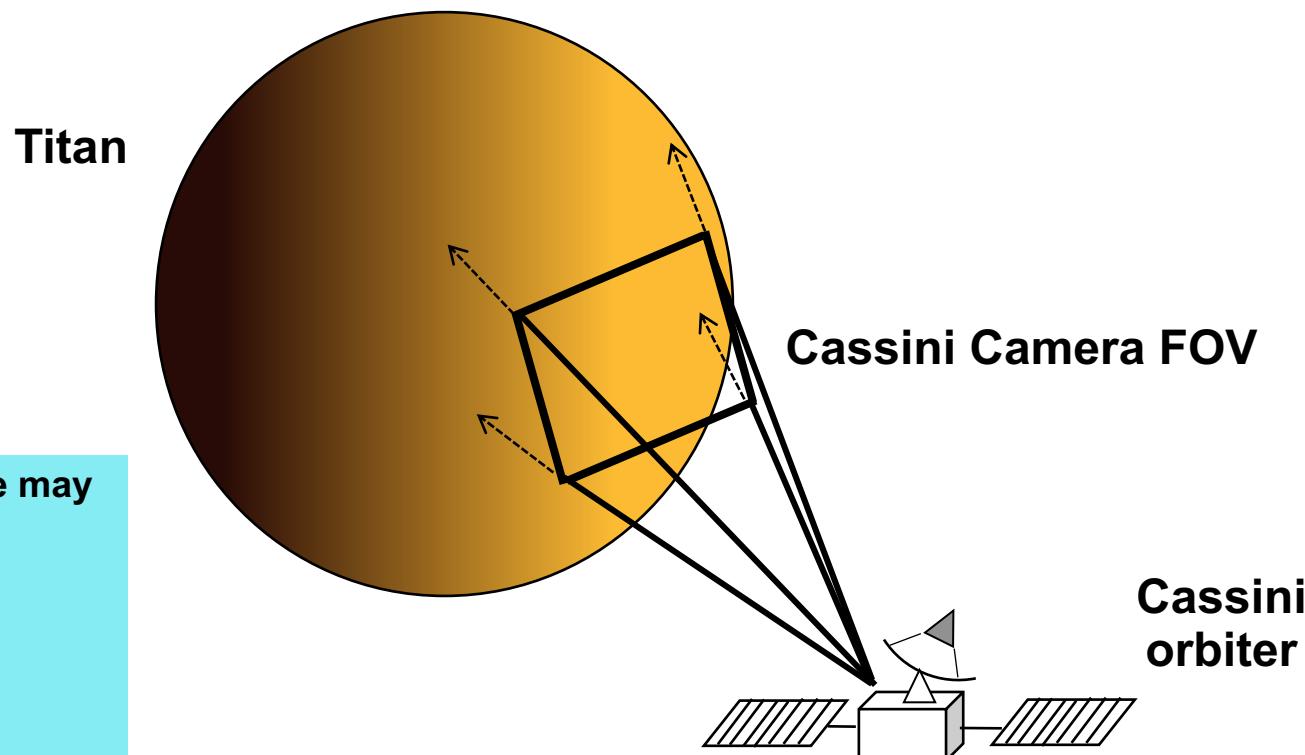
API: GFOCLT



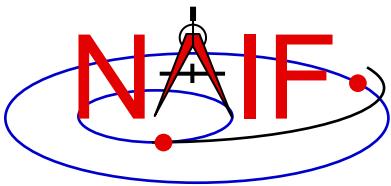
# Target in Field of View

Navigation and Ancillary Information Facility

Find the time periods when Titan appears in the FOV of the Cassini ISS Narrow Angle Camera (NAC). The target shape is modeled as an ellipsoid. (Point targets are also supported.)



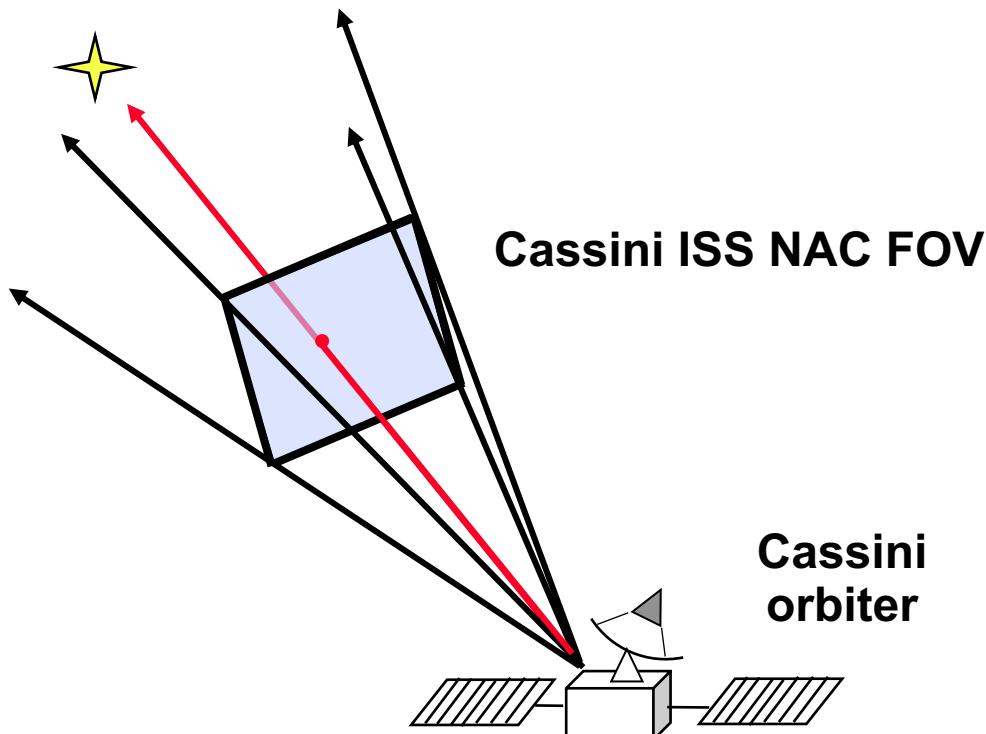
API: GFTFOV



# Ray in FOV Search

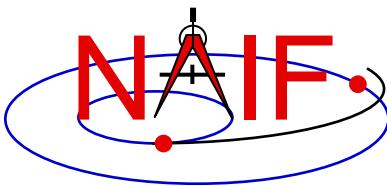
Navigation and Ancillary Information Facility

Find the time periods when a star appears in the FOV of the Cassini ISS Narrow Angle Camera (NAC). The target direction is modeled as a ray, optionally corrected for stellar aberration.



The FOV shape may  
be any of:  
  
Rectangle  
Circle  
Ellipse  
Polygon

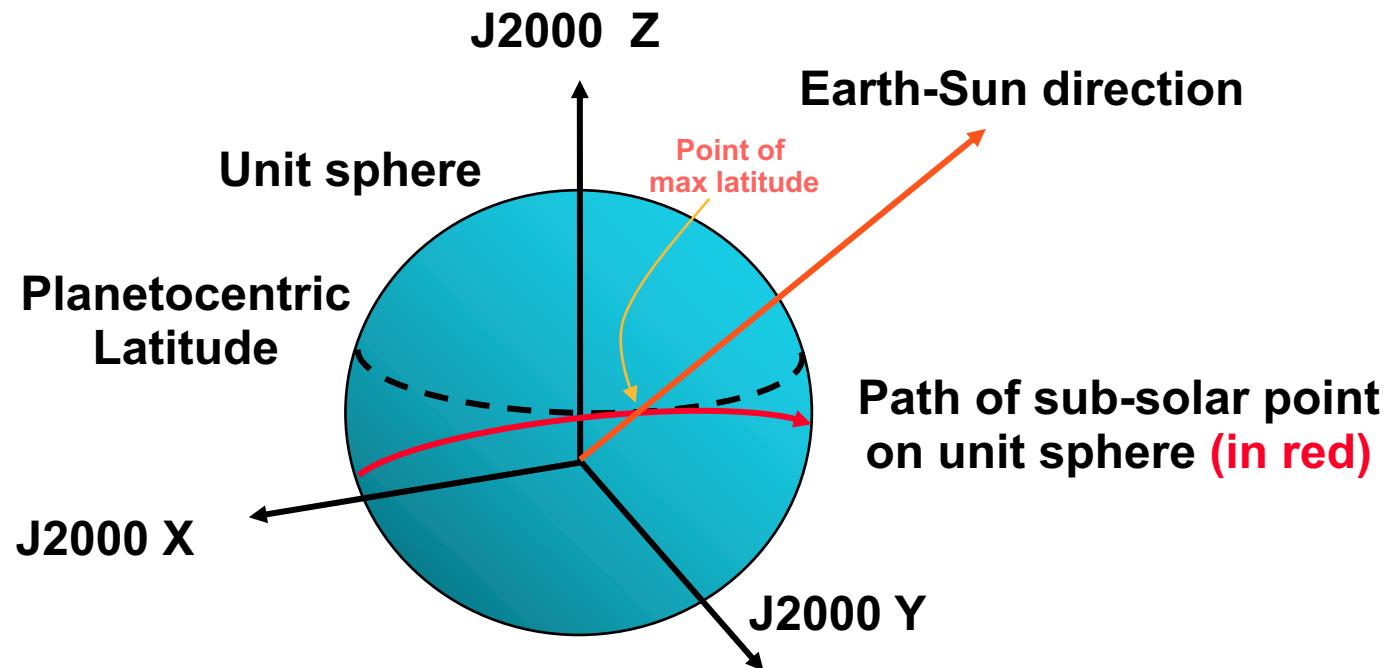
API: GFRFOV



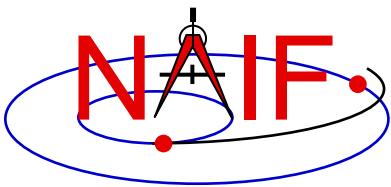
# Position Coordinate Local Extremum Search

Navigation and Ancillary Information Facility

Find the time(s) at which the planetocentric latitude of the Earth-Sun vector, expressed in the J2000 frame, is at a local maximum.



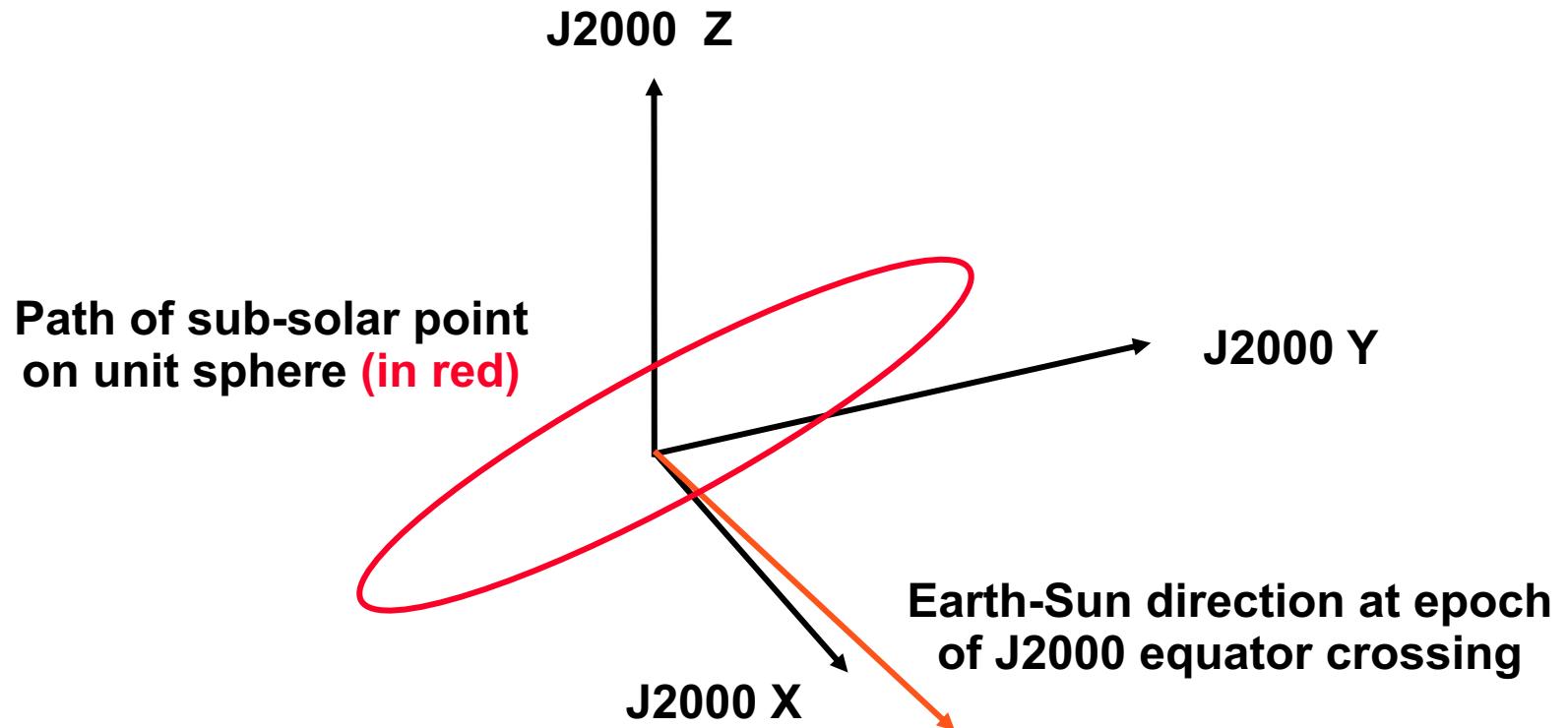
API: GFPOSC



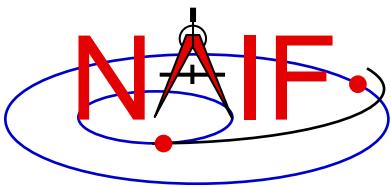
# Position Coordinate Equality Search

Navigation and Ancillary Information Facility

Find the time(s) at which the Z component of the Earth-Sun vector, expressed in the J2000 frame, is 0.



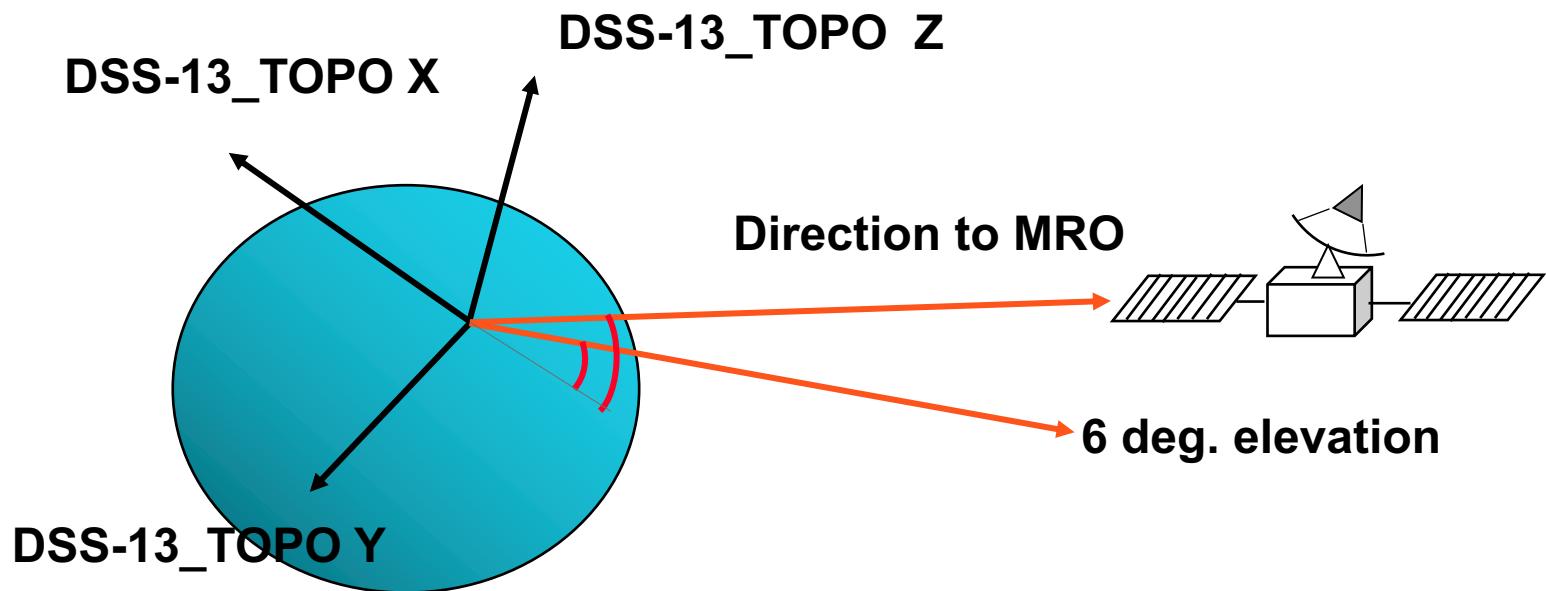
API: GFPOS C



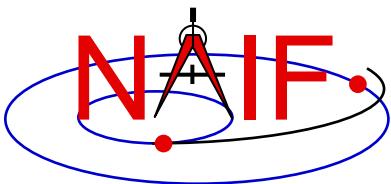
# Position Coordinate Inequality Search -1

Navigation and Ancillary Information Facility

Find the time periods when the elevation of the DSS-13 to Mars Reconnaissance Orbiter (MRO) spacecraft vector, expressed in the DSS-13 topocentric frame, is greater than 6 degrees.



API: GFPOS C

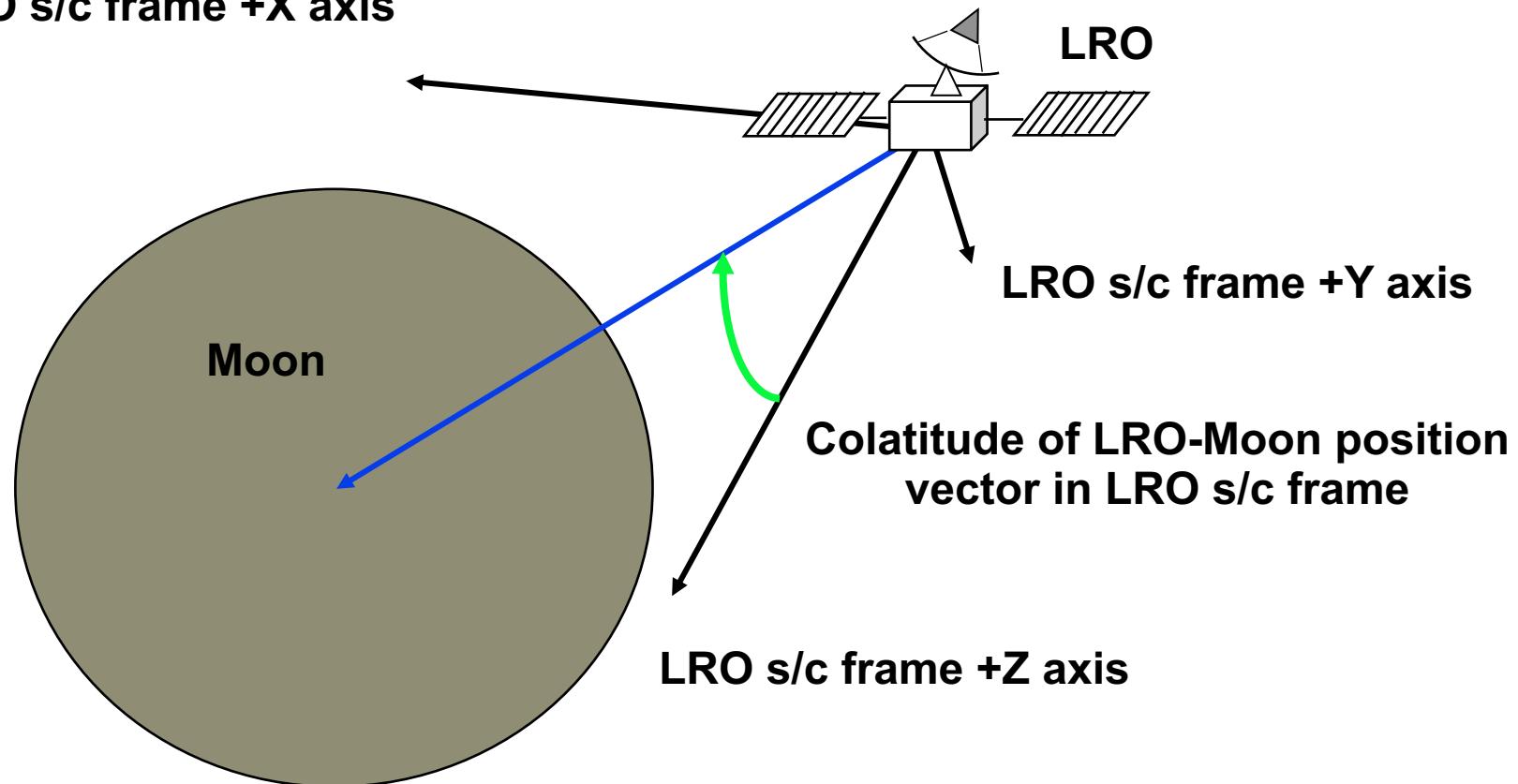


# Position Coordinate Inequality Search -2

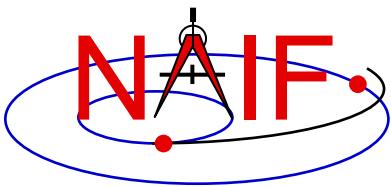
Navigation and Ancillary Information Facility

Find the time periods when the Lunar Reconnaissance Orbiter's (LRO) off-nadir angle of the +Z axis exceeds 5 degrees.

LRO s/c frame +X axis



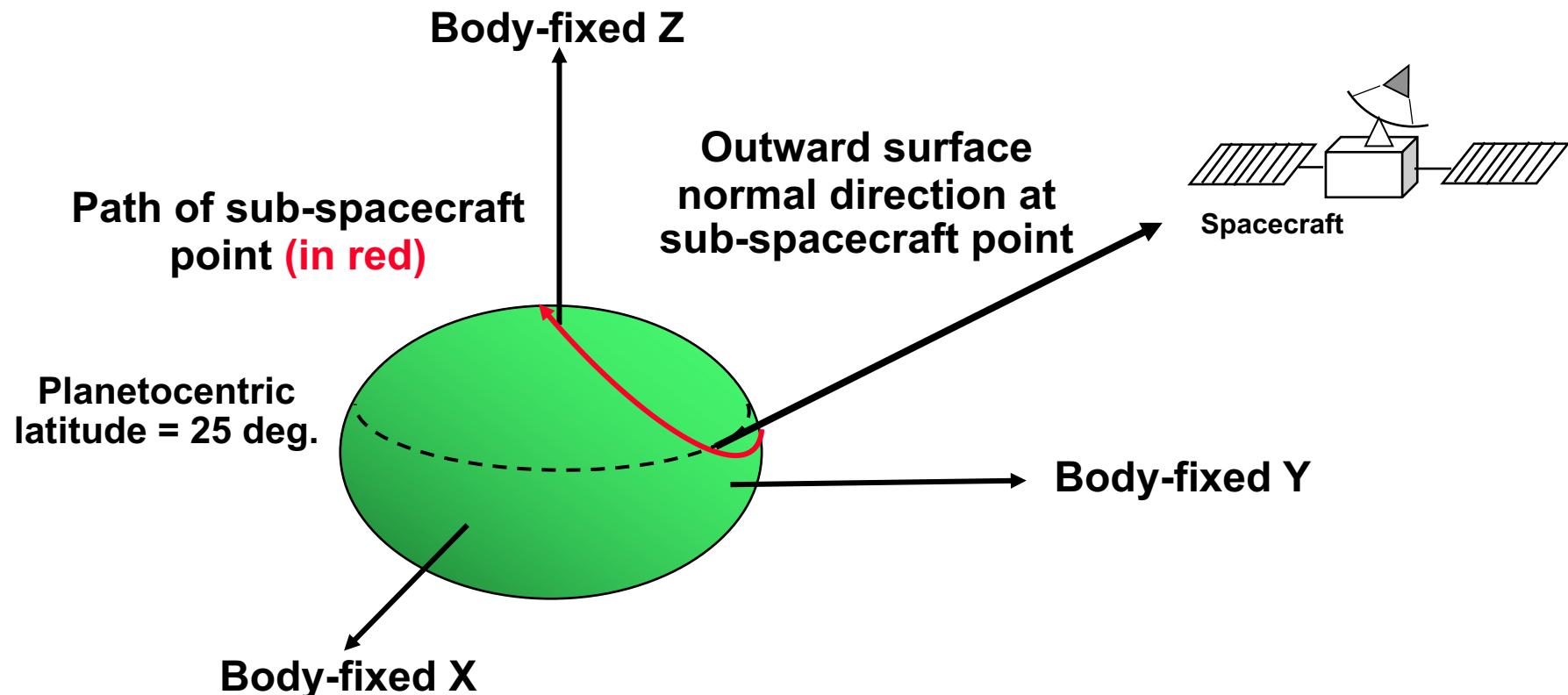
API: GFPOS



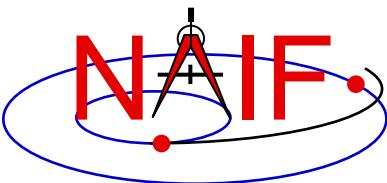
# Sub-Observer Point Coordinate Equality Search

Navigation and Ancillary Information Facility

Find the time(s) at which the planetocentric latitude of the sub-spacecraft point, using the “near point” definition, is 25 degrees.



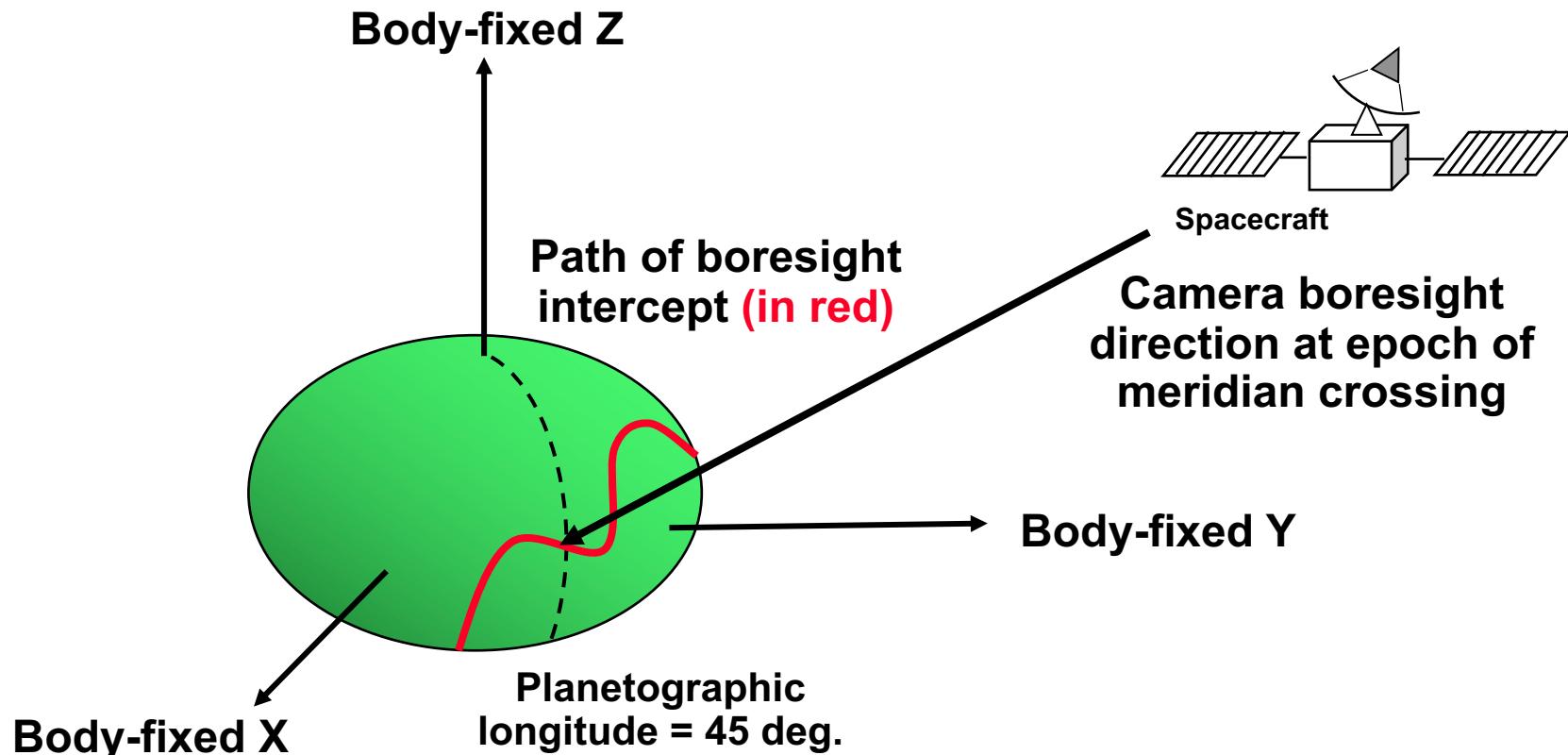
API: GFSUBC



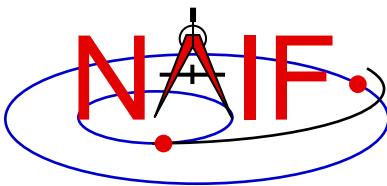
# Surface Intercept Coordinate Equality Search

Navigation and Ancillary Information Facility

Find the time(s) at which the planetographic longitude of a given camera boresight surface intercept is 45 degrees.



API: GFSNTC

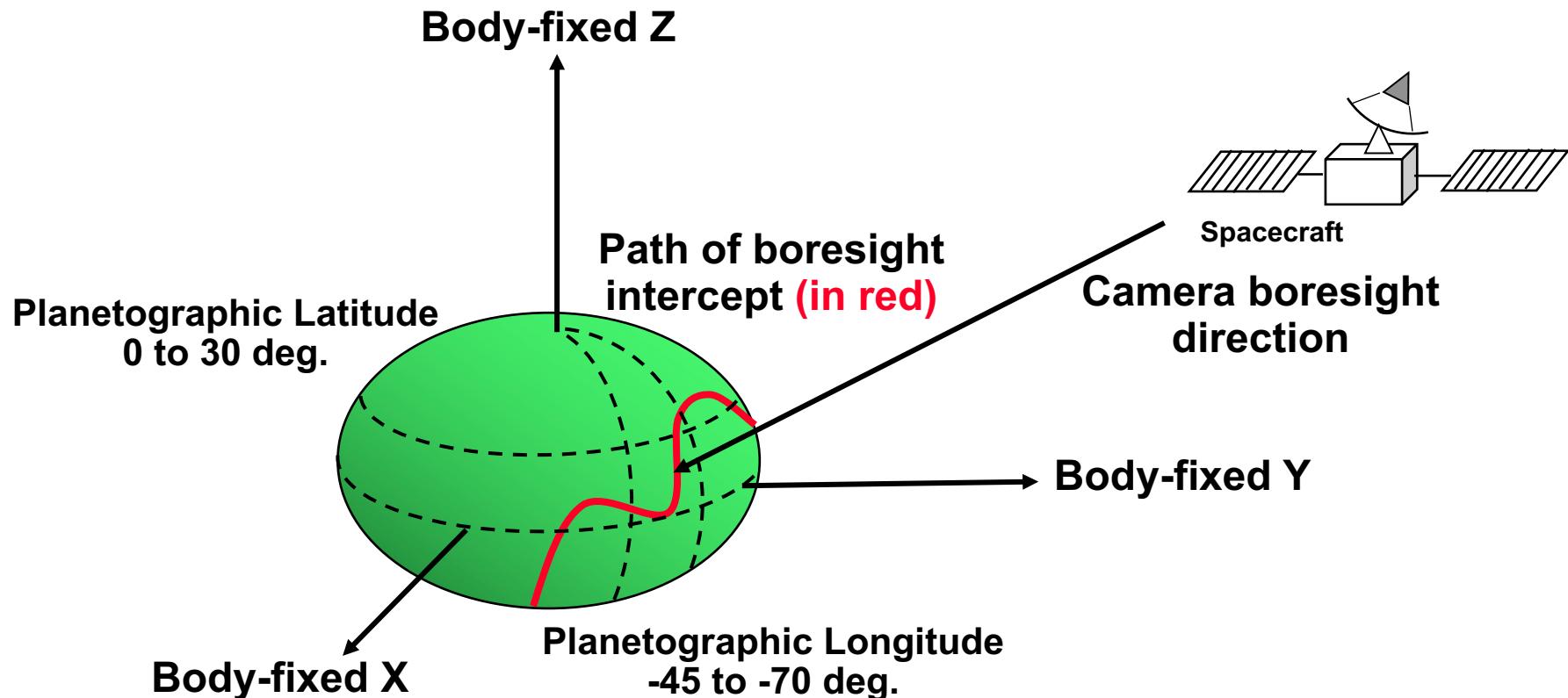


# Surface Intercept “Box” Search

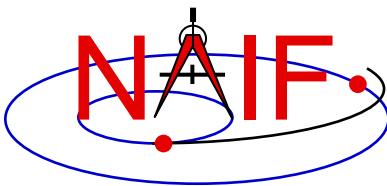
Navigation and Ancillary Information Facility

Find the time periods when the planetographic longitude of a camera boresight surface intercept is between -70 and -45 degrees, and the intercept latitude is between 0 and 30 degrees.

The solution requires four (cascading) inequality searches.



API: GFSNTC

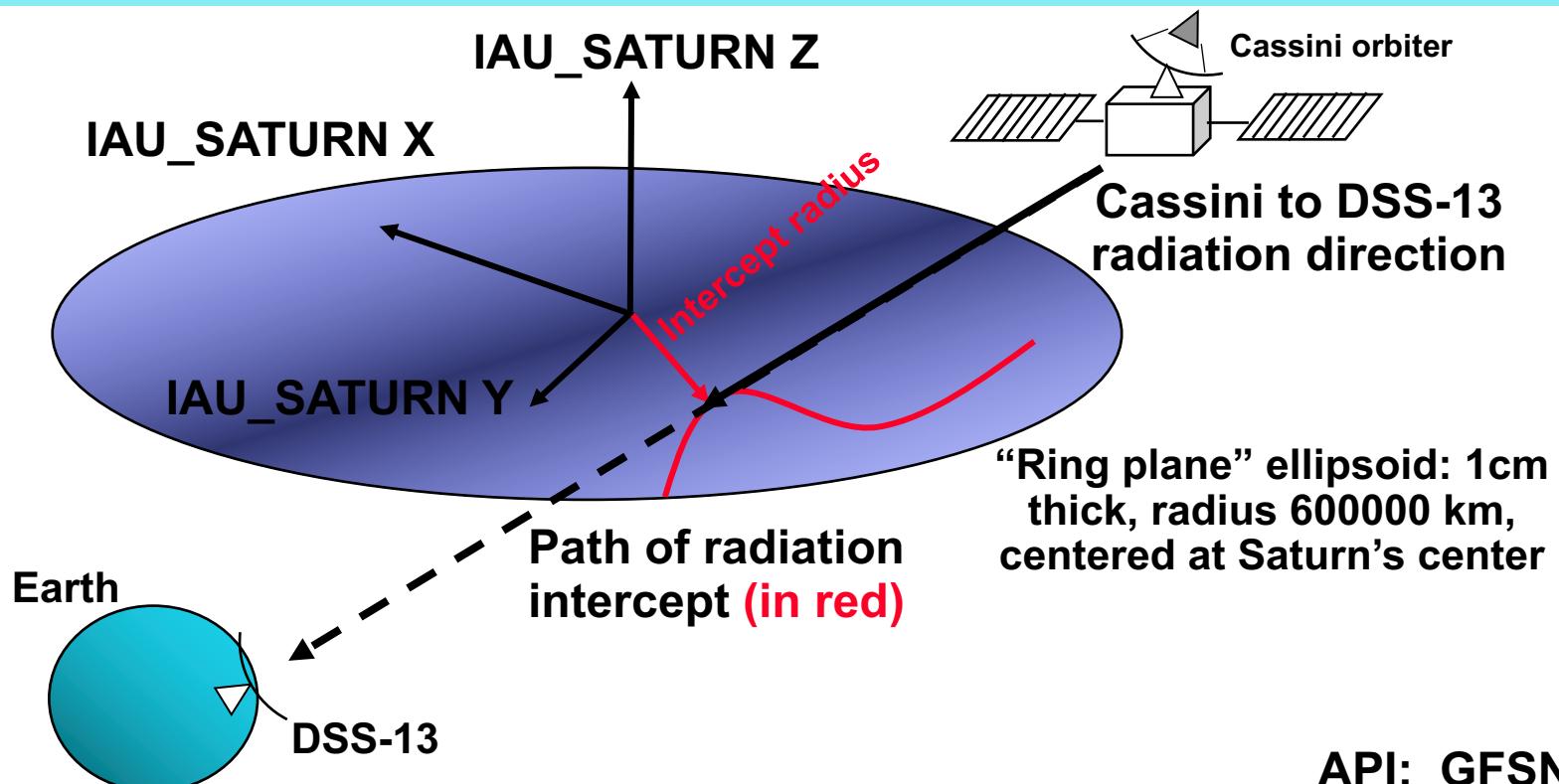


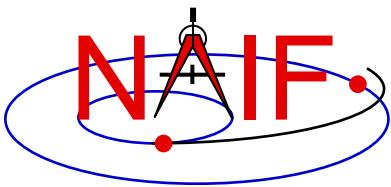
# Surface Intercept Coordinate Equality Search

Navigation and Ancillary Information Facility

Find the time at which the ring plane intercept of the Cassini to DSS-13 vector, corrected for transmission light time (stellar aberration correction is unnecessary), has radius 300000km.

The solution requires a dynamic frame for which one axis points along the radiation path.

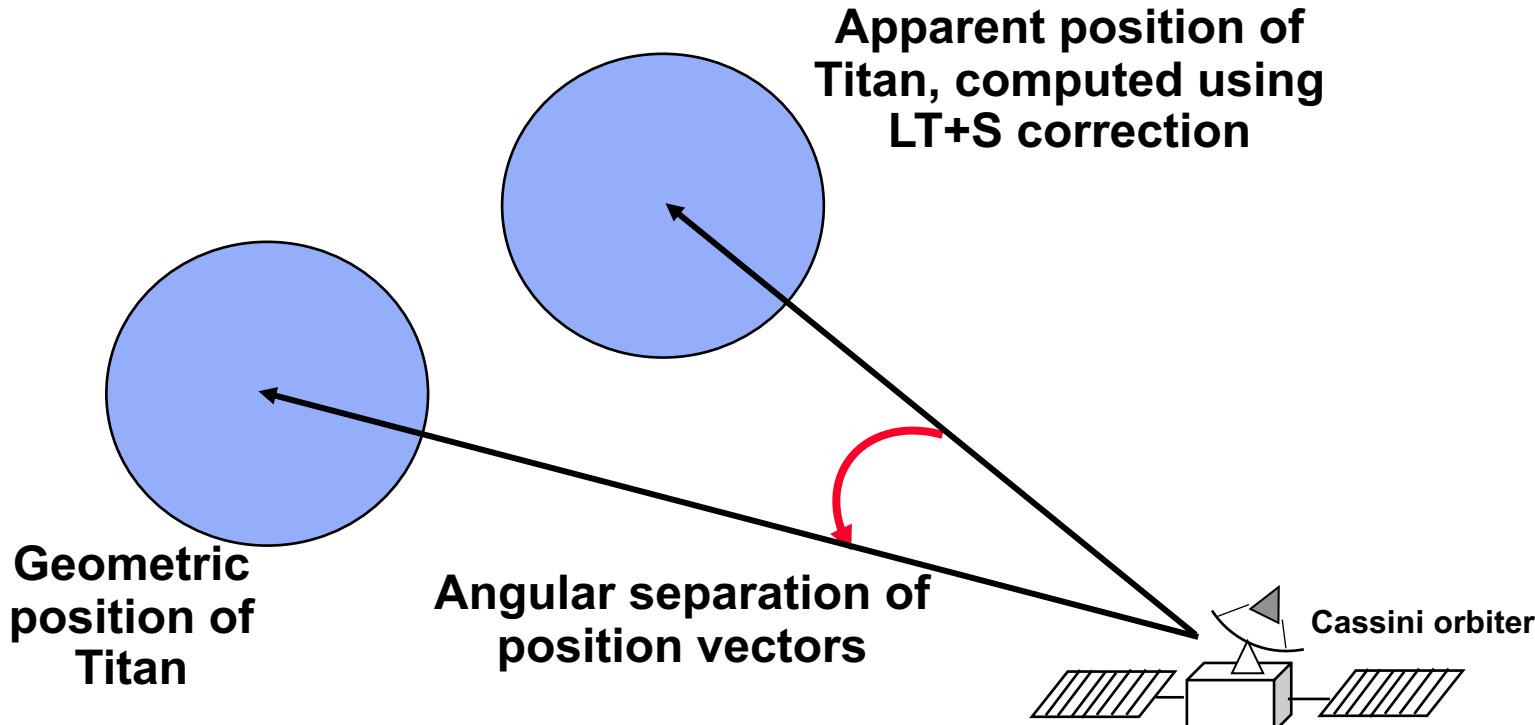




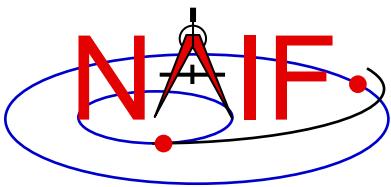
# User-Defined Quantity Extremum Search

Navigation and Ancillary Information Facility

Find the time periods when the angular separation of the geometric and apparent positions of Titan as seen from the Cassini orbiter attains an absolute maximum.



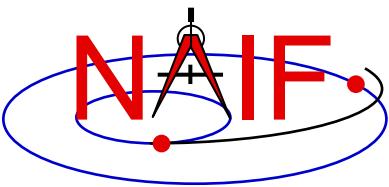
API: GFUDS



---

Navigation and Ancillary Information Facility

# Geometric Search Types and Constraints

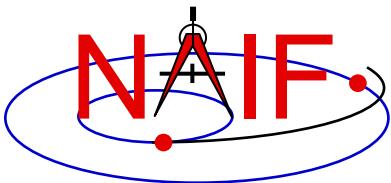


# Types of Geometric Quantities

---

Navigation and Ancillary Information Facility

- The GF subsystem deals with two types of geometric quantities:
  - “Binary state functions”: functions of time that can be “true” or “false.” Examples:
    - » Occultation state, such as: “Titan is fully occulted by Saturn at time t”
    - » Visibility state: A target body or object modeled as a ray (for example, a star) is visible in a specified instrument FOV at time t
  - Scalar numeric functions of time, for example
    - » Observer-target distance
    - » Latitude or longitude of an observer-target vector, sub-spacecraft point, or ray-surface intercept point
    - » Angular separation of two target bodies as seen by an observer

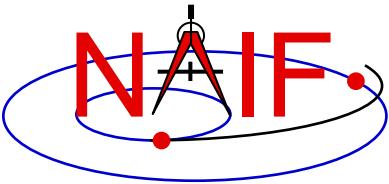


# Binary State Searches

---

Navigation and Ancillary Information Facility

- **Binary state searches find times when a specified binary state function takes the value “true.”**
  - SPICE window arithmetic can be used to find the times when a binary state function is “false.”

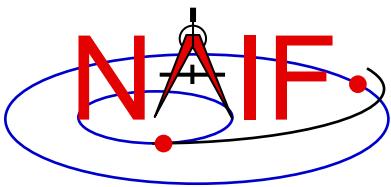


# Numeric Searches

---

Navigation and Ancillary Information Facility

- Numeric searches find times when a scalar numeric quantity satisfies a mathematical constraint. The supported constraints are:
  - The function
    - » equals a specified reference value.
    - » is less than a specified reference value.
    - » is greater than a specified reference value.
    - » is at a local maximum.
    - » is at a local minimum.
    - » is at an absolute maximum.
    - » is at an absolute minimum.
    - » is at an “adjusted” absolute maximum: the function is within a given tolerance of the absolute maximum.
    - » is at an “adjusted” absolute minimum: the function is within a given tolerance of the absolute minimum.
- Examples for a Distance search follow.



# Solve Distance Equality

Navigation and Ancillary Information Facility

Find times at which Distance =  $D_0$

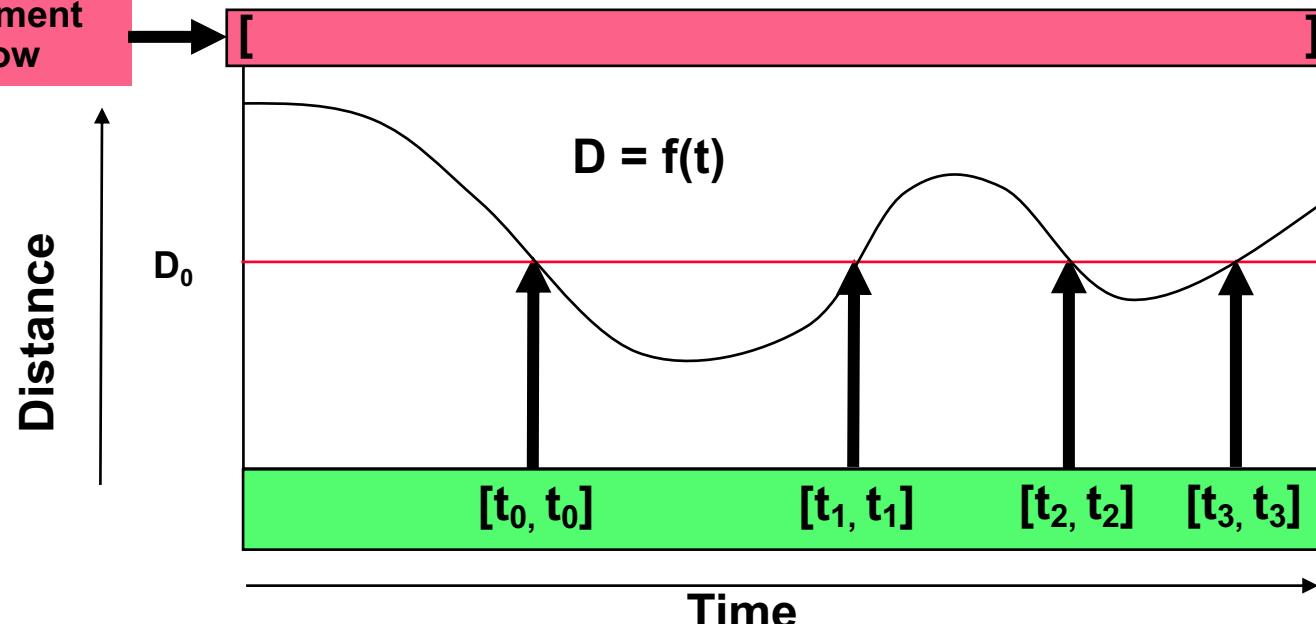
GF API input arguments defining constraint:

RELATE = '='

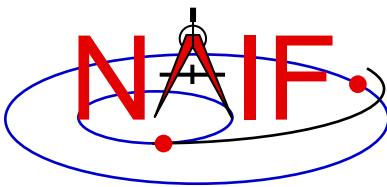
ADJUST = 0.D0

REFVAL = D0

Confinement  
Window



Result  
Window



# Solve Distance < Inequality

Navigation and Ancillary Information Facility

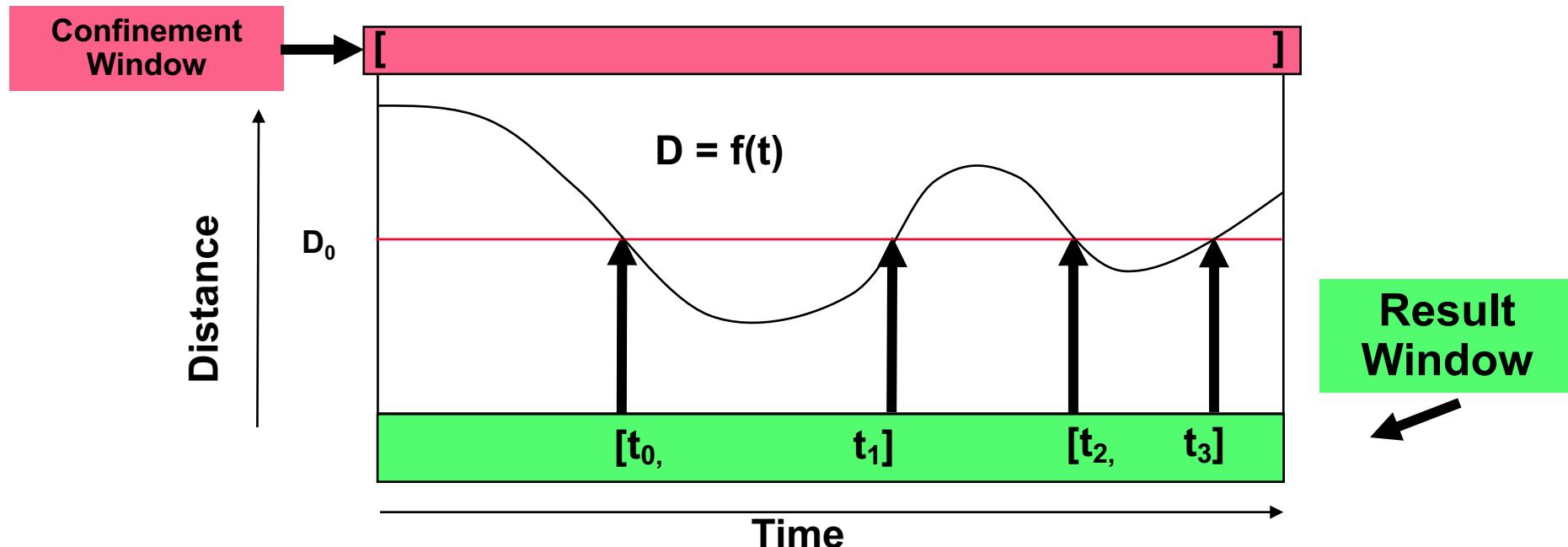
Find times during which Distance  $< D_0$

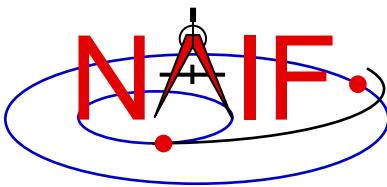
GF API input arguments defining constraint:

RELATE = '<'

ADJUST = 0.D0

REFVAL = D0





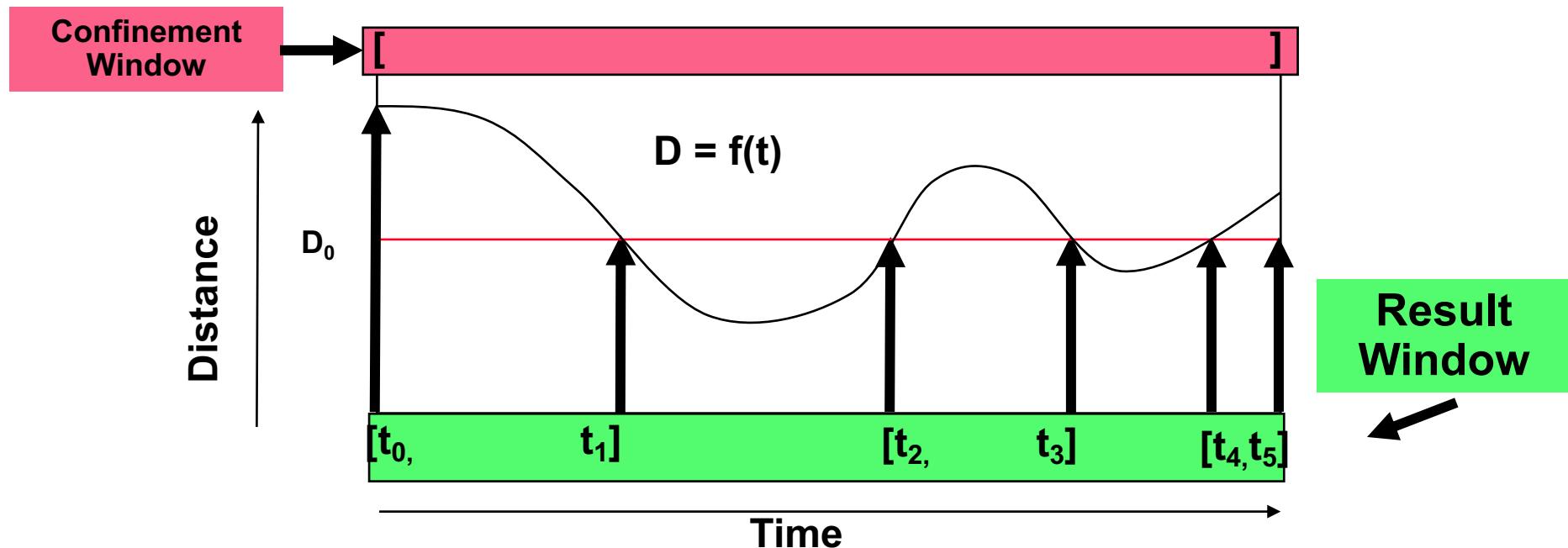
# Solve Distance > Inequality

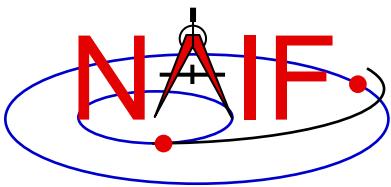
Navigation and Ancillary Information Facility

Find times during which Distance >  $D_0$

GF API input arguments defining constraint:

```
RELATE    =  '>'  
ADJUST    =  0.D0  
REFVAL    =  D0
```





# Find Local Minima

Navigation and Ancillary Information Facility

Find times at which Distance is a local minimum.

GF API input arguments defining constraint:

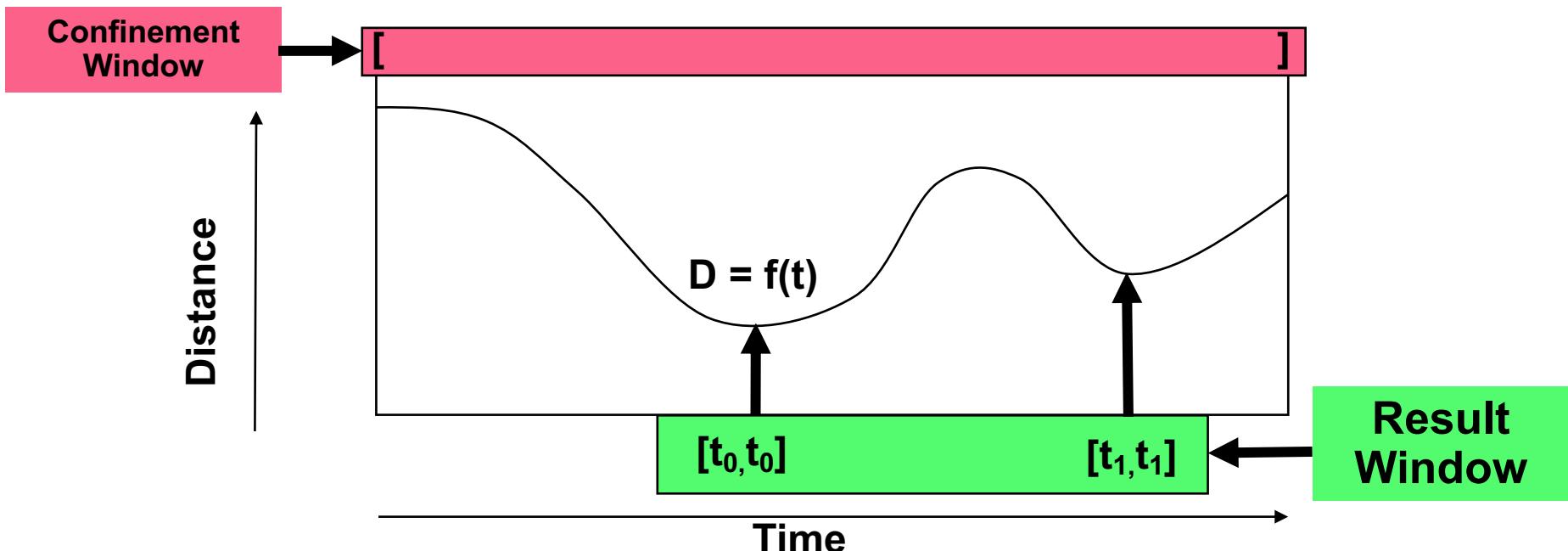
**RELATE** = 'LOCMIN'

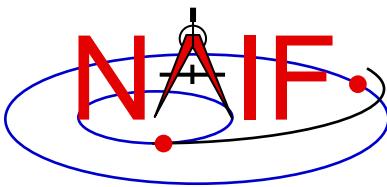
**ADJUST** = 0.0D0

(**REFVAL** is not used in this case

**REFVAL** = 0.0D0

but should be initialized for portability)





# Find Local Maxima

Navigation and Ancillary Information Facility

Find times at which Distance is a local maximum.

GF API input arguments defining constraint:

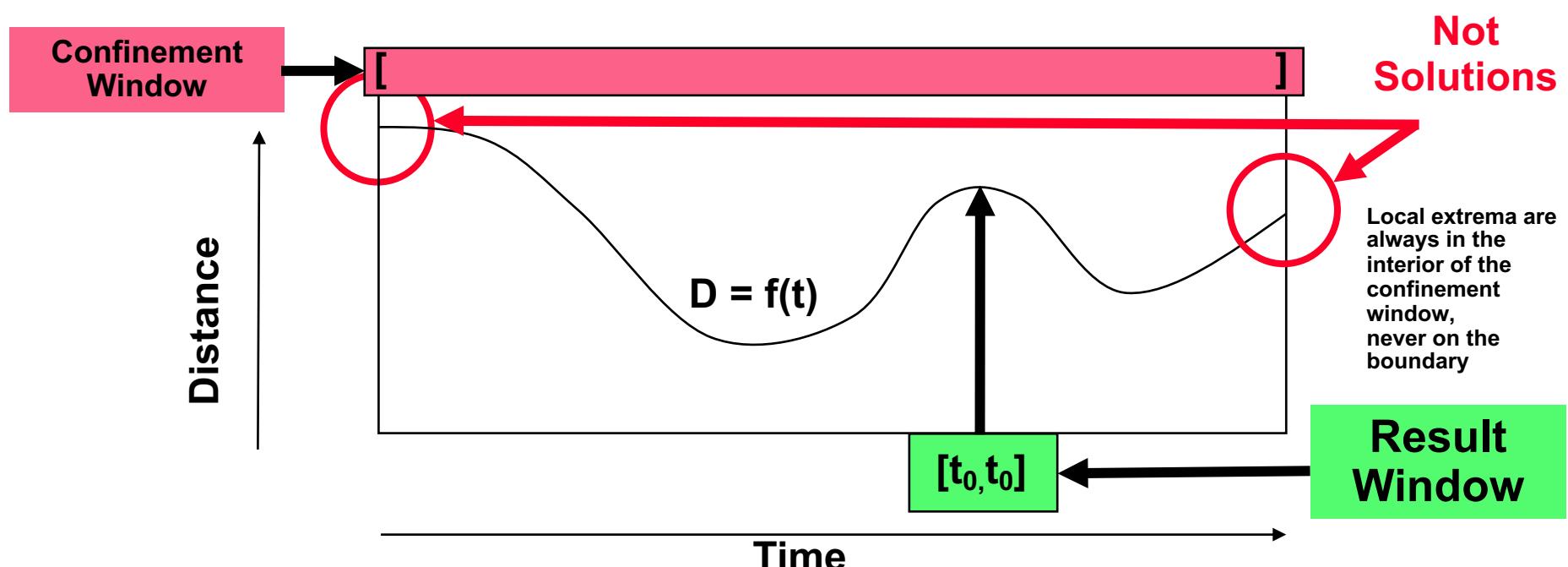
**RELATE** = 'LOCMAX'

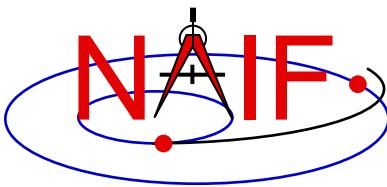
**ADJUST** = 0.D0

(**REFVAL** is not used in this case

**REFVAL** = 0.D0

but should be initialized for portability)





# Find Absolute Minimum

Navigation and Ancillary Information Facility

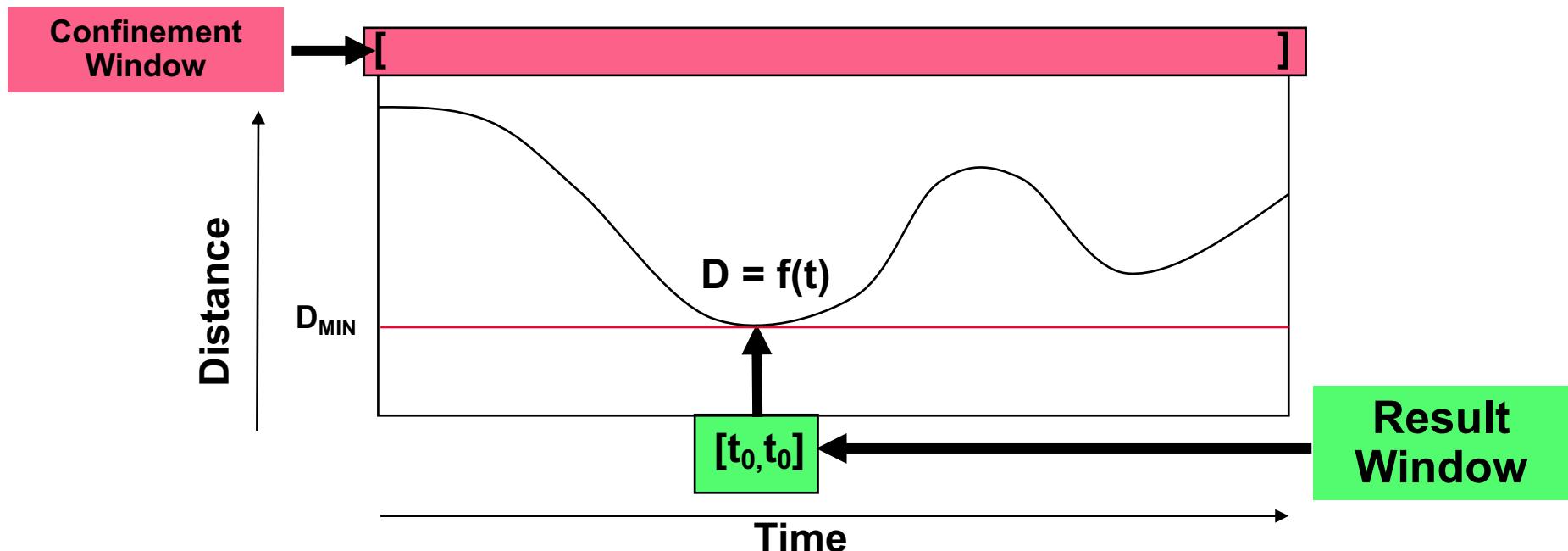
Find the time at which Distance is an absolute minimum.

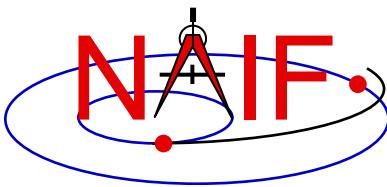
GF API input arguments defining constraint:

`RELATE = 'ABSMIN'`

`ADJUST = 0.D0`      (REFVAL is not used in this case

`REFVAL = 0.D0`      but should be initialized for portability)





# Find Absolute Maximum

Navigation and Ancillary Information Facility

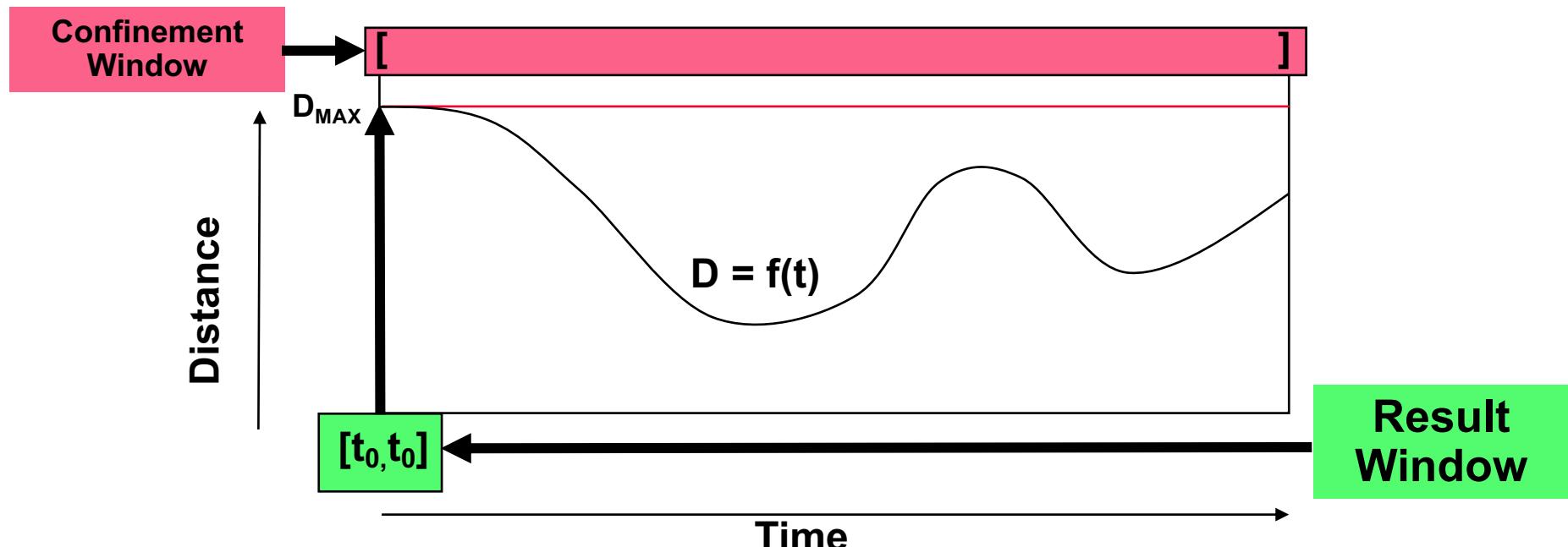
**Find the time at which Distance is an absolute maximum.**

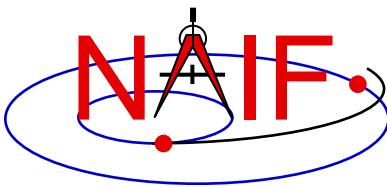
**GF API input arguments defining constraint:**

**RELATE = 'ABSMAX'**

**ADJUST = 0.D0** (REFVAL is not used in this case

**REFVAL = 0.D0** but should be initialized for portability)





# Find Adjusted Absolute Minimum

Navigation and Ancillary Information Facility

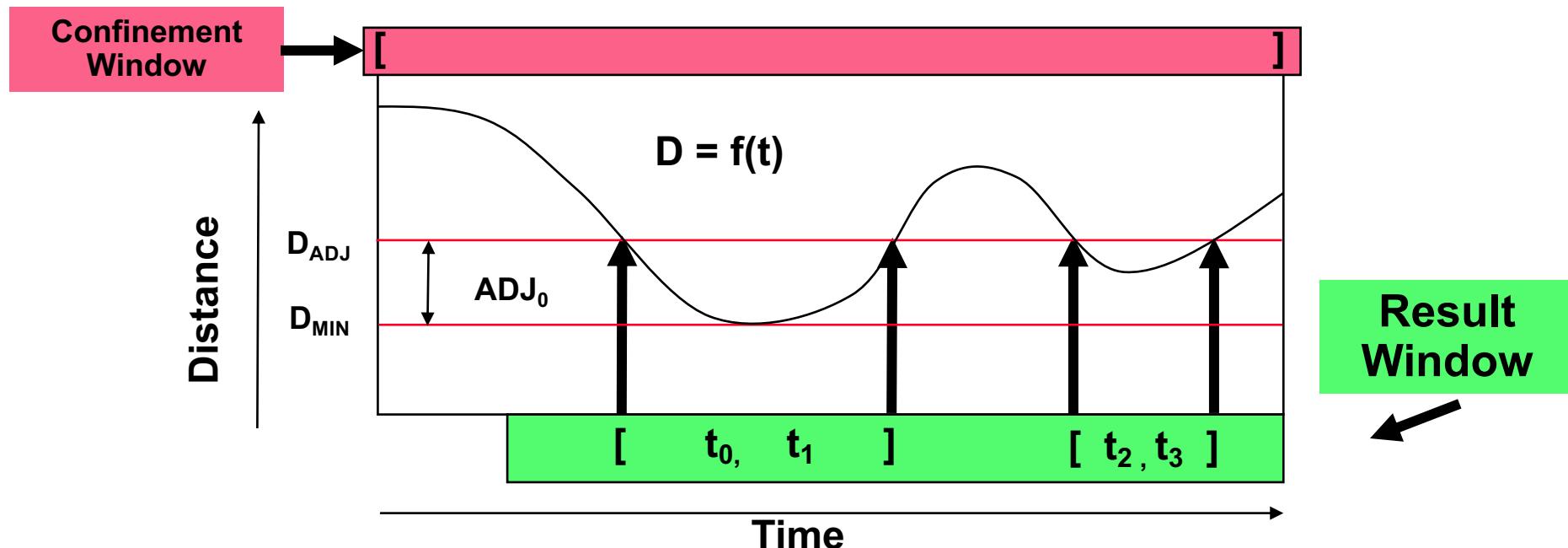
Find times at which Distance  $< D_{ADJ} = D_{MIN} + ADJ_0$

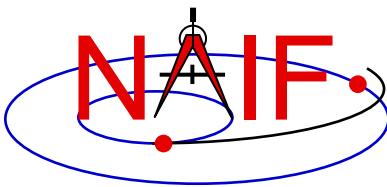
GF API input arguments defining constraint:

`RELATE = 'ABSMIN'`

`ADJUST = ADJ0 (ADJ0 > 0)`

`REFVAL = 0.0D0` (REFVAL is not used in this case but should be initialized for portability)





# Find Adjusted Absolute Maximum

Navigation and Ancillary Information Facility

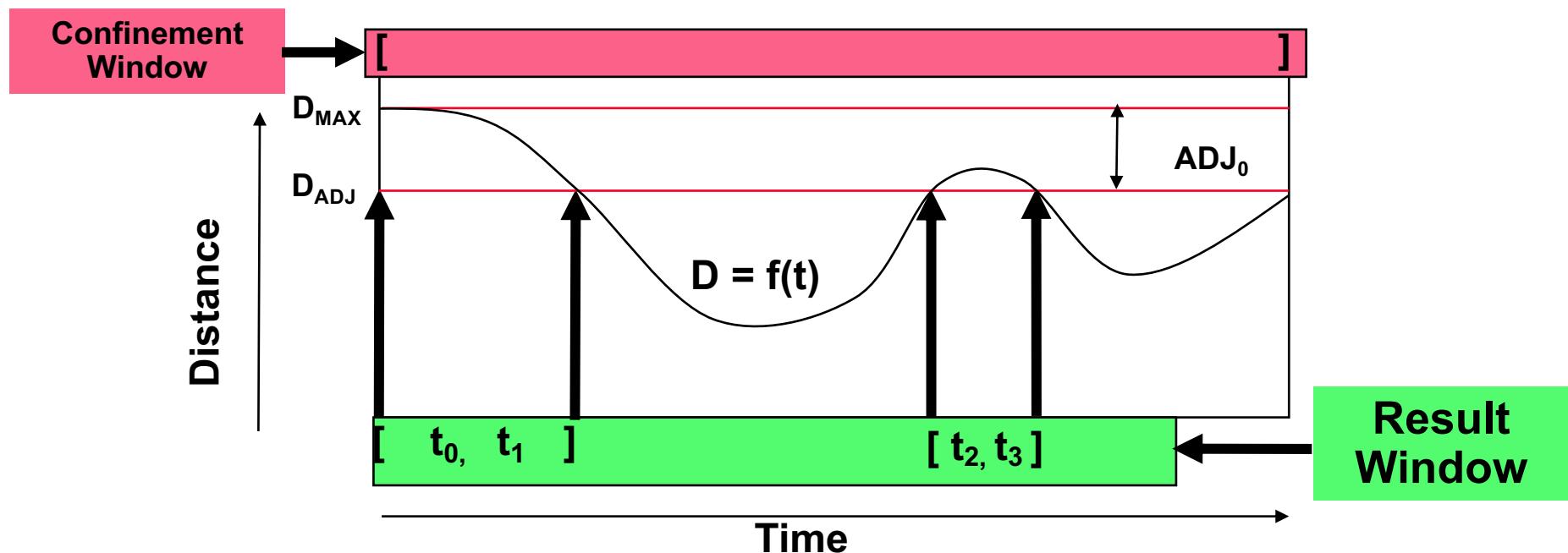
Find times at which Distance >  $D_{ADJ} = D_{MAX} - ADJ_0$

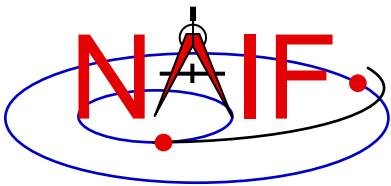
GF API input arguments defining constraint:

`RELATE = 'ABSMAX'`

`ADJUST = ADJ0 (ADJ0 > 0)`

`REFVAL = 0.D0 (REFVAL is not used in this case but should be initialized for portability)`





# More Details

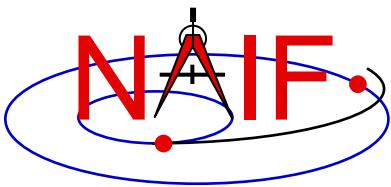
- GF mid-level APIs
- Root Finding, including step size selection
- Workspace
- API Example: GFDIST



# GF Mid-Level API Routines

Navigation and Ancillary Information Facility

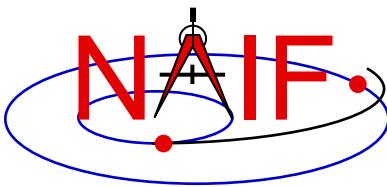
- **The Fortran and C SPICE Toolkits provide some mid-level APIs that provide additional capabilities:**
  - Progress reporting, which can be customized by the user
  - Interrupt handling which can be customized by the user
    - » In Fortran, no default interrupt detection is available
  - User-customizable search step and refinement routines
  - User-adjustable root finding convergence tolerance
- **The GF mid-level search APIs are:**
  - **GFEVNT**: All scalar numeric quantity searches
  - **GFFOVE**: Target or ray in FOV searches
  - **GFOCCE**: Occultation searches



---

**Navigation and Ancillary Information Facility**

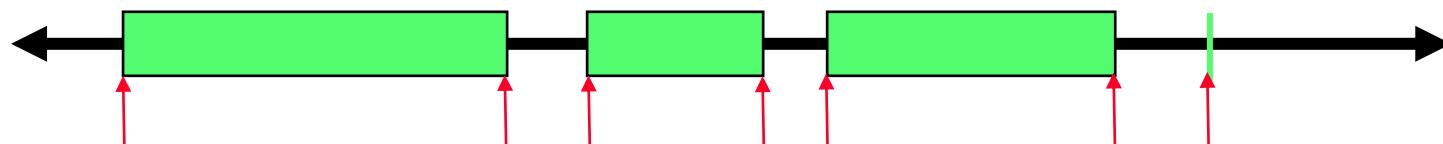
# **Root Finding**



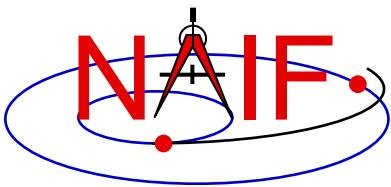
# Root Finding

Navigation and Ancillary Information Facility

- To produce a final or intermediate result window, the GF subsystem must accurately locate the endpoints of the window's intervals. These endpoints are called “roots.”
  - The green regions below (rectangles and vertical line segment) represent intervals of a window.
  - Roots are indicated by the red, vertical arrows.



- Elsewhere, “root finding” often refers to solving  $f(x) = 0$ .
- In the GF setting, roots are boundaries of time intervals over which a specified constraint is met.
  - Roots can be times when a binary state function changes values.
- Most popular root finding methods, e.g. Newton, secant, bisection, require the user to first “bracket” a root: that is, determine two abscissa values such that a **single** root is located between those values.
- The GF subsystem solves a more difficult problem: it performs a **global** search for roots. That is, given correct inputs, it finds **all** roots within a user-specified confinement window.
  - The user is not asked to bracket the roots.

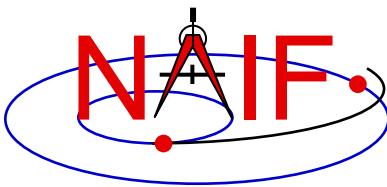


# Step Size Selection

---

Navigation and Ancillary Information Facility

- The GF subsystem asks the user to specify a time step (often called the “step size”) that will be used to bracket roots.
  - For binary state searches, the step size is used to bracket the endpoints of the intervals of the result window.
    - » The step size must be shorter than any event of interest, and shorter than any interval between events that are to be distinguished.
  - For numeric searches, the step size is used to bracket the endpoints of the intervals of the window on which the geometric quantity is monotonically decreasing.
    - » The step size must be shorter than any interval on which the function is monotonically increasing or decreasing.
  - In both cases, the step size must be large enough so the search completes in a reasonable amount of time.



# Monotone Windows -1

Navigation and Ancillary Information Facility

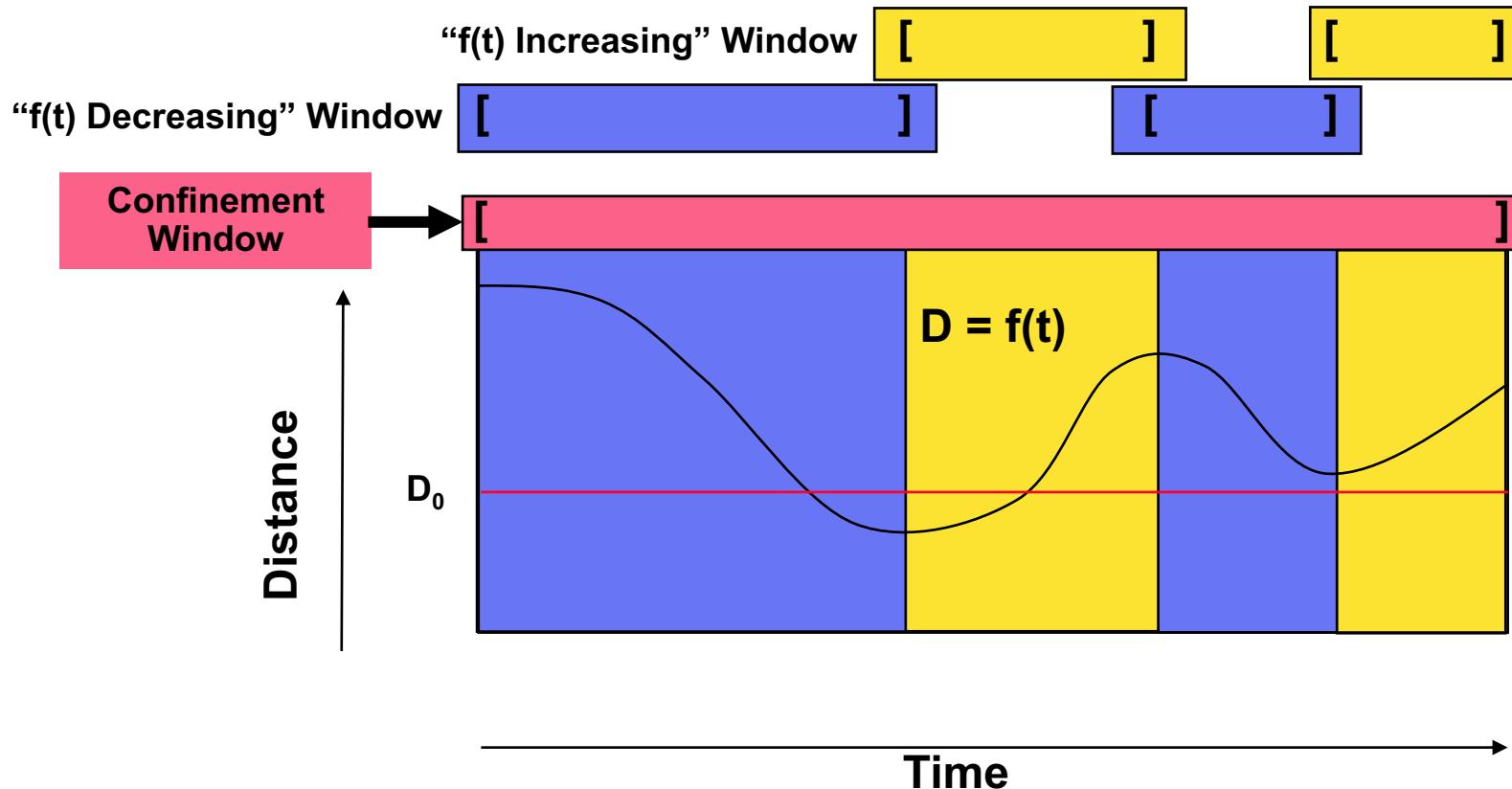
## Example: monotone windows for a distance function

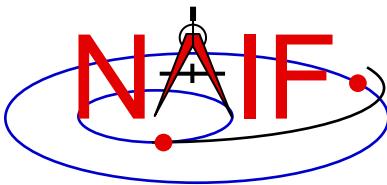
Note that:

Extrema occur only at window interval boundaries.

Each window interval contains at most one root of an equality condition.

Within each window interval, the solution of an inequality is a single (possibly empty) interval.



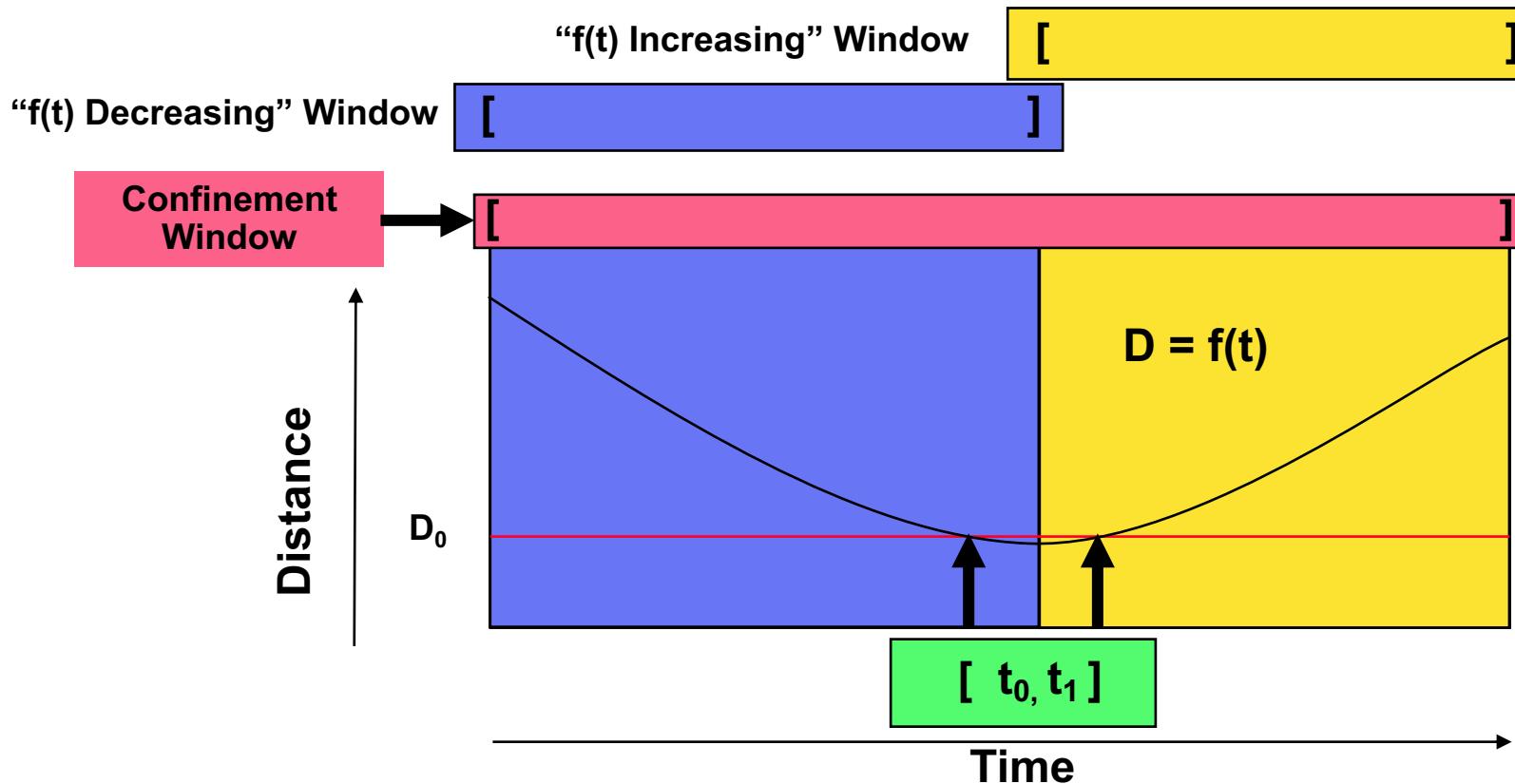


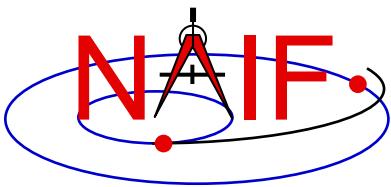
# Monotone Windows-2

Navigation and Ancillary Information Facility

The shortest interval on which the function is monotonically increasing or decreasing may be LONGER than the event of interest.

For example, consider the search for times when  $D < D_0$ . The result window consists of the interval  $[t_0, t_1]$  shown below.

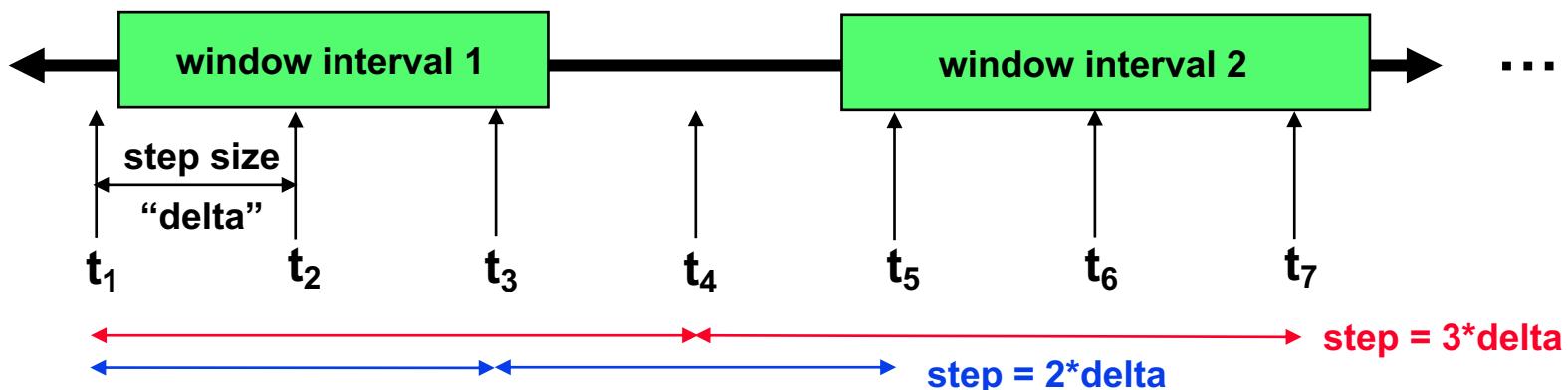




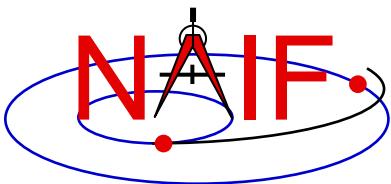
# Bracketing Interval Endpoints

Navigation and Ancillary Information Facility

- In the diagram below, the green boxes denote intervals of a window.
- The start of the first interval is bracketed by the first and second times; the end of the first interval is bracketed by the third and fourth times. The start of the second interval is bracketed by the fourth and fifth times.

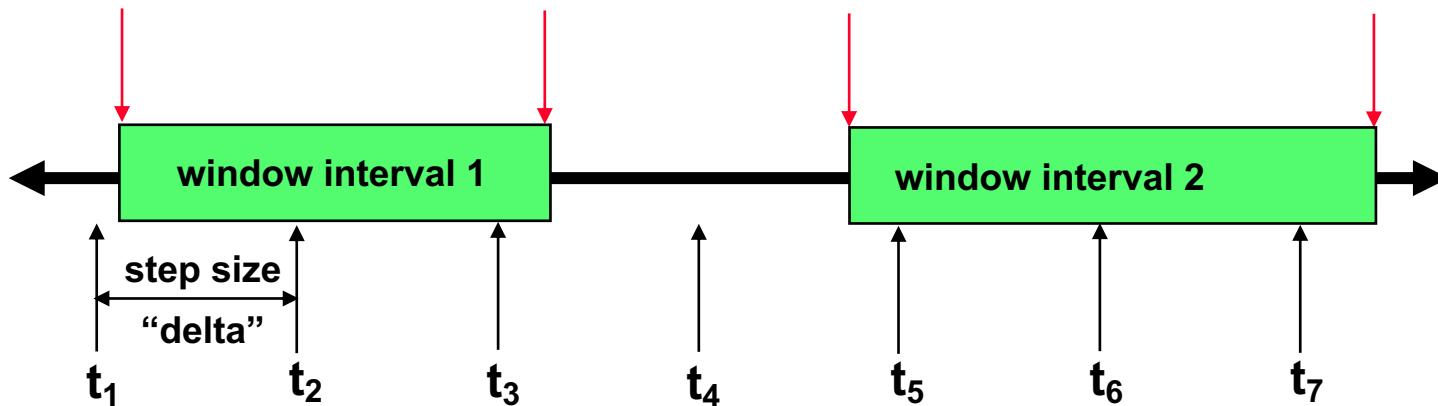


- The step size is a critical determinant of the completeness of the solution:
  - If the step size were equal to  $3*\delta$ , the first interval would not be seen.
  - If the step size were equal to  $2*\delta$ , the first and second intervals would not be seen as distinct.

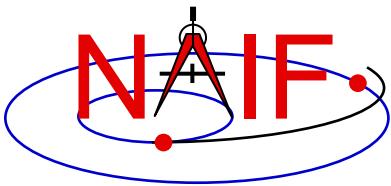


# Solving for Interval Endpoints

Navigation and Ancillary Information Facility



- Once an interval endpoint is bracketed, the GF subsystem performs a refinement search to locate the endpoint precisely (shown by the red arrows).
  - The default tolerance for convergence is 1.e-6 second.
- The refinement search is usually relatively fast compared to the interval bracketing search.

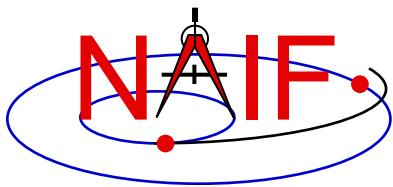


# The Result Window

---

Navigation and Ancillary Information Facility

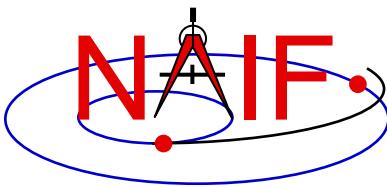
- For binary state searches, the window whose endpoints are found IS the result window.
  - The search is done once the endpoint refinement step has been completed for each interval over which the state is true.
- For numeric searches, once the monotone windows have been found, the result window still must be computed:
  - Local and absolute extrema can be found without further searching.
  - Equalities, inequalities, and adjusted absolute extrema require a second search pass in which each monotone interval is examined.
    - » These searches don't require sequential stepping and are usually relatively fast compared to the interval bracketing search.
- Since the roots are found by a search process, they are subject to approximation errors.
  - NOTE: The geometric condition may not be satisfied at or near the endpoints of the result window's intervals.
- Usually data errors are large enough so that the accuracy of the result is poorer than its precision.



---

Navigation and Ancillary Information Facility

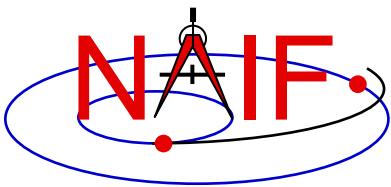
# Workspace



# Specifying Workspace Dimensions

Navigation and Ancillary Information Facility

- **GF numeric scalar searches can require relatively large amounts of memory to store intermediate results.**
  - For Fortran Toolkits, user applications must declare a buffer of workspace windows.
  - For C, IDL, and MATLAB Toolkits, users need only specify the maximum number of workspace window intervals that are needed; these Toolkits will dynamically allocate the amount of memory required by the specified workspace window interval count.
- **In most cases, users need not accurately compute the required amount of workspace; it's usually safe to specify a number much larger than the amount actually needed.**
  - For example, if the result window is anticipated to contain 100 intervals, specifying a workspace window interval count of 10000 will very likely succeed.
  - See the GF Required Reading and the API documentation for details.

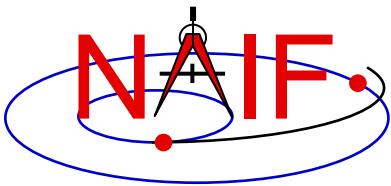


---

Navigation and Ancillary Information Facility

## API Example: GFDIST

### Solve for Distance Constraints

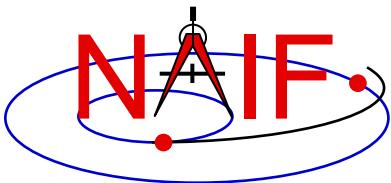


# API Example: GFDIST

---

Navigation and Ancillary Information Facility

- **GFDIST finds times when a specified constraint on the distance between two ephemeris objects is met.**
  - The distance is the norm of a position vector
  - The position vector is defined using a subset of the inputs accepted by SPKPOS:
    - » Target
    - » Observer
    - » Aberration Correction
  - The constraint is a numeric relation: equality or inequality relative to a reference value, or attainment of a local or absolute extremum.
- **The search is conducted within a specified confinement window.**
- **The search produces a result window which indicates the time period, within the confinement window, over which the constraint is met.**

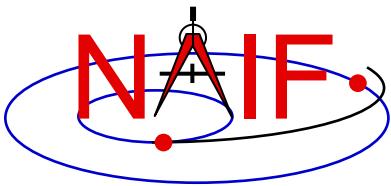


# Language Differences

---

Navigation and Ancillary Information Facility

- Due to use of SPICE windows, some of the GFDIST set-up code differs substantially across languages.
  - We'll show how to perform the set-up unique to each language.
    - » Note: there's no set-up to do in the MATLAB case; hence there's no MATLAB-specific set-up slide.
  - The rest of the code is sufficiently parallel across languages to justify showing only the Fortran code.
  - Note however that the treatment of workspace differs across languages: only in Fortran does the user application have to pass the workspace array to the GF API routine.



# Fortran Set-up

Navigation and Ancillary Information Facility

## Fortran constant and variable declarations

### Declare confinement window, result window, and workspace array

**INCLUDE 'gf.inc'**      Include GF parameters  
such as NWDIST  
...

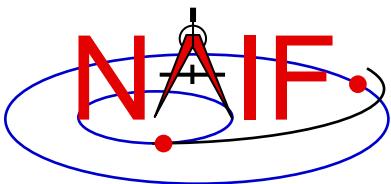
<b>INTEGER</b>	<b>LBCELL</b>
<b>PARAMETER</b>	<b>( LBCELL = -5 )</b>
<b>INTEGER</b>	<b>MAXWIN</b>
<b>PARAMETER</b>	<b>( MAXWIN = 200000 )</b>
<b>DOUBLE PRECISION</b>	<b>CNFINE ( LBCELL : MAXWIN )</b>
<b>DOUBLE PRECISION</b>	<b>RESULT ( LBCELL : MAXWIN )</b>
<b>DOUBLE PRECISION</b>	<b>WORK ( LBCELL : MAXWIN, NWDIST )</b>

} Choose a “large” value for window size (called “MAXWIN” here), if you’re not sure what size is required. The actual requirement depends on underlying geometry, confinement window, and search step size.

Initialization...typically done once per program execution

Initialize confinement and result windows. Workspace need not be initialized here.

**CALL SSIZED ( MAXWIN, CNFINE )**  
**CALL SSIZED ( MAXWIN, RESULT )**



# C Set-up

## Navigation and Ancillary Information Facility

### C constant and variable declarations

**Declare confinement and result windows, as well as size of workspace.**

```
#include "SpiceUsr.h" } Include CSPICE macro, typedef,  
... and prototype declarations  
  
#define NINTVL      100000 } Choose a "large" value for window size (called "MAXWIN"  
#define MAXWIN    ( 2 * NINTVL ) } here), if you're not sure what size is required. Actual  
                                requirement depends on underlying geometry, confinement  
                                window, and search step size. The window size must be twice  
                                the maximum number of intervals the window is meant to hold.  
  
SPICEDOUBLE_CELL ( cnfine, MAXWIN ); } These macro calls declare CSPICE  
SPICEDOUBLE_CELL ( result, MAXWIN ); } window structures and set their  
                                maximum sizes.
```



# IDL Set-up

---

Navigation and Ancillary Information Facility

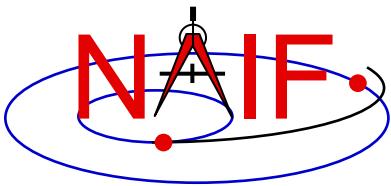
## IDL window creation

Icy windows are created dynamically:

```
cnfine = cspice_celld ( MAXWIN )
```

} Choose a “large” value for window size (called “MAXWIN” here), if you’re not sure what size is required. Actual requirement depends on underlying geometry, confinement window, and search step size. The window size must be twice the maximum number of intervals the window is meant to hold.

The output result window is created by CSPICE\_GFDIST; it does not require a constructor call by the user application.



# All Languages: Additional Set-up

Navigation and Ancillary Information Facility

Initialization...typically done once per program execution

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

**CALL FURNISH ('spk\_file\_name')**

**CALL FURNISH ('leapseconds\_file\_name')**

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

**The next step is to insert times into the confinement window. The simplest confinement window consists of a single time interval.**

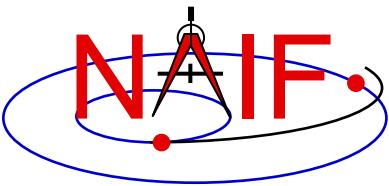
Convert UTC start and stop times to ephemeris time (TDB), if needed:

**CALL STR2ET ( 'utc\_start', *tdb\_0* )**

**CALL STR2ET ( 'utc\_stop', *tdb\_1* )**

**Insert start and stop times into the confinement window:**

**CALL WNINSD ( *tdb\_0*, *tdb\_1*, CNFINE )**



# Execute the Search -1

Navigation and Ancillary Information Facility

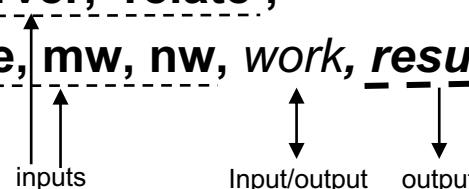
Search execution...done as many times as necessary during program execution

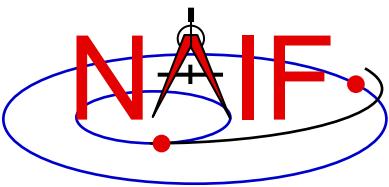
## Choose:

- Geometric input arguments:
  - » Target
  - » Observer
  - » Aberration correction
- Constraint arguments:
  - » Relation
  - » Reference value, if applicable
  - » Adjustment value, if applicable
- Search step size

## Then call GFDIST:

CALL GFDIST (target, ‘correction’, observer, ‘relate’,  
refval, adjust, step, cnfine, mw, nw, work, result)





# Execute the Search -2

---

Navigation and Ancillary Information Facility

Search execution, continued

## Extract intervals from the result window:

```
DO I = 1, WNCARD( RESULT )
```

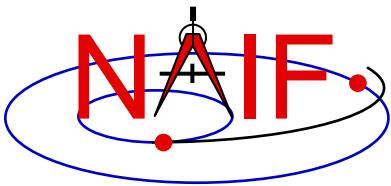
[Fetch the endpoints of the Ith interval of the result window.]

```
CALL WNFETD( RESULT, I, START, FINISH )
```

[ use START, FINISH... ]

```
END DO
```

- Note:
  - The result window may be empty.
  - The constraint might not be satisfied at or near the endpoints of any interval of RESULT.
    - » Consider using the window contraction routine WNCND to shrink the intervals of the result window slightly, so the constraint is met on the entire result window.
    - » Caution: DON'T use WNCND for minimum, maximum, or equality searches---the result window will disappear! (WNCND is ok for adjusted absolute extrema search results, though, since the result intervals are not singletons.)
    - » Caution: using WNCND may not be desirable if the window is an intermediate result: subsequent, derived results might be made less accurate.



# Arguments of GFDIST - 1

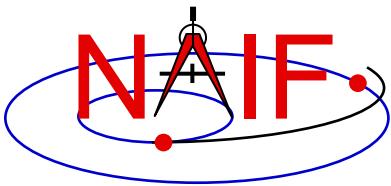
Navigation and Ancillary Information Facility

## INPUTS

- **TARGET\*** and **OBSERVER\***: Character names or NAIF IDs for the end point and origin of the position vector (Cartesian position and velocity vectors) whose length is the subject of the search. **Below, we'll simply call this length “the distance.”**
  - The position vector points from observer to target.
- **CORRECTION**: Specification of what kind of aberration correction(s), if any, to apply in computing the distance.
  - Use ‘LT+S’ to use the apparent position of the target as seen by the observer. ‘LT+S’ invokes light time and stellar aberration corrections.
  - Use ‘NONE’ to use the uncorrected (aka “geometric”) position, as given by the source SPK file or files.

See the headers of the subroutines GFDIST and SPKEZR, the document SPK Required Reading, or the “Fundamental Concepts” tutorial for details. See the SPK tutorial backup charts for examples of aberration correction magnitudes.

\* 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 GFDIST - 2

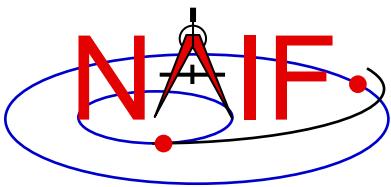
Navigation and Ancillary Information Facility

## INPUTS

- **RELATE, REFVAL, ADJUST:** parameters specifying a constraint to be met by the distance.
  - RELATE may be any of ‘=’, ‘>’, ‘<’, ‘LOCMAX’, ‘LOCMIN’, ‘ABSMAX’, ‘ABSMIN’
  - If RELATE is an equality or inequality operator, REFVAL is the corresponding double precision reference value. Units are km.
    - » For example, if the constraint is “distance = 4.D5 km,” then RELATE is ‘=’ and REFVAL is 4.D5.
  - If RELATE specifies an absolute maximum or minimum, ADJUST is the adjustment value. Units are km.
    - » Set ADJUST to 0.D0 for a simple absolute maximum or minimum.
    - » Set ADJUST to a positive value ADJ for either DISTANCE > absolute max - ADJ or DISTANCE < absolute min + ADJ.
- **STEP:** search step size, expressed as TDB seconds.
- **CNFINE:** the confinement window over which the search will be performed.
- **MW, NW, WORK:** the maximum capacity of each workspace window, the number of workspace windows, and the workspace array.

## OUTPUTS

- **RESULT:** the window of times over which the distance constraint is satisfied.

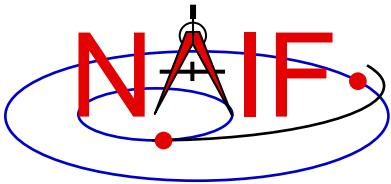


---

Navigation and Ancillary Information Facility

# Toolkit Applications

January 2020



# Toolkit Applications

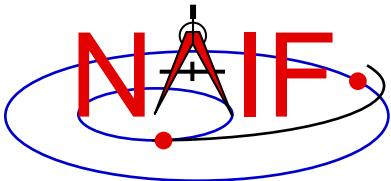
---

Navigation and Ancillary Information Facility

**Toolkit applications create or manipulate kernels, or perform other functions such as time conversion.**

**Each of the following applications is included in the Toolkits.**

- Time conversion tool: *chronos*
- SPK generation tool: *mkspk*
- SPK merge and subset tool: *spkmerge*
- SPK comparison and sample tool: *spkdiff*
- CK generation tool: *msopck*
- Frame comparison tool: *frmdiff*
- DSK generation tool: *mkdsk*
- DSK export tool: *dskexp*
- Kernel summary tools: *brief*, *ckbrief*, *dskbrief*, *spacit*
- Comments manipulation tools: *commnt*, *spacit*
- File format converters: *tobin*, *toxfr*, (and *bingo*, in Fortran toolkits)

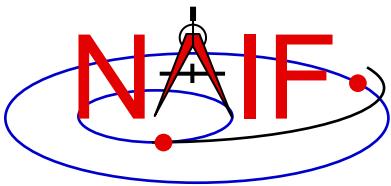


# Using Toolkit Apps

---

Navigation and Ancillary Information Facility

- All of these apps are meant to be used as operating system shell executables
  - One generally cannot run these within IDL or MATLAB
    - » In some cases you can run from within IDL or MATLAB, but this is not recommended:
      - In IDL, use the “spawn” command
      - In MATLAB, use the “system” command



# CHRONOS

---

Navigation and Ancillary Information Facility

***chronos provides a flexible interface to the SPICE Toolkit time conversion capabilities.***

***chronos supports time conversion between the following time systems/types and using the indicated time types for those systems.***

*Supported Time Systems* --> *Supported Time Types*

---

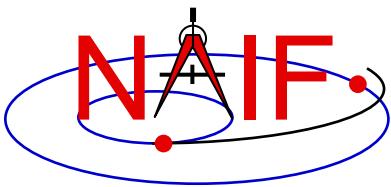
Universal Coord. Time (UTC) --> SCET, ERT, ETT, LT

Ephemeris Time (ET) --> SCET, ERT, ETT, SECONDS, LT

S/C On-board Clock Time (SCLK) --> SCLK, HEX, TICKS

Local Solar Time (LST) --> LST, LSUN

ERT = Earth Received Time  
ETT = Earth Transmit Time



# CHRONOS - Input/Output Matrix

Navigation and Ancillary Information Facility

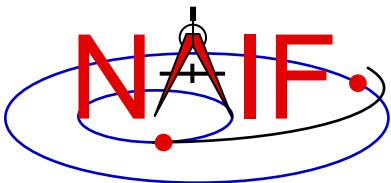
## *Input System/Type*

-----  
UTC / SCET (\*)  
UTC / ERT  
UTC / ETT  
ET / SCET (\*)  
ET / ERT  
ET / ETT  
ET / SECONDS  
SCLK / SCLK (\*)  
SCLK / HEX  
SCLK / TICKS  
LST / LST

## *Output System/Type*

-----  
UTC / SCET (\*)  
UTC / ERT  
UTC / ETT  
UTC / LT  
ET / SCET (\*)  
ET / ERT  
ET / ETT  
ET / SECONDS  
ET / LT  
SCLK / SCLK (\*)  
SCLK / HEX  
SCLK / TICKS  
LST / LST (\*)  
LST / LSUN

(\*) default input/output types

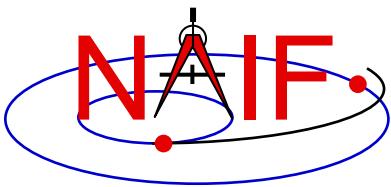


# CHRONOS - Miscellaneous

---

Navigation and Ancillary Information Facility

- ***chronos* normally converts one input time per execution, but can run in batch mode to speed up conversion for multiple input times.**
- **OS shell alias capabilities can be used to define shortcuts for commonly used time conversions.**
- ***chronos* has an extensive user's guide.**



# CHRONOS - Usage

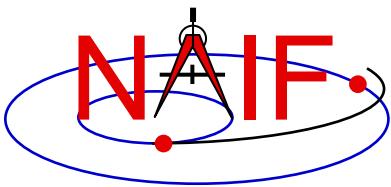
## Navigation and Ancillary Information Facility

Terminal Window

```
$ chronos
...
CHRONOS Usage
-----
To convert time from one supported system/type to another:

% CHRONOS -SETUP <setup file name OR kernel file name(s)>
    -FROM <"from" time system>
    [-FROMTYPE <"from" time type>]
    -TO <"to" time system>
    [-TOTYPE <"to" time type>]
    [-FORMAT <output time format picture>]
    -TIME <input time> | -BATCH
    [-SC <sc ID>]
    [-CENTER <central body ID>]
    [-LANDINGTIME <UTC time of the landing>]
    [-SOL1INDEX <index of the first SOL>]
    [-NOLABEL]
    [-TRACE]
```

Items in square brackets are optional



# CHRONOS - Example

## Navigation and Ancillary Information Facility

### Terminal Window

```
$ cat chronos.cas
Sample CHRONOS setup file for Cassini
\begindata
    KERNELS_TO_LOAD = ( 'naif0007.tls', 'cas00085.tsc' )
    SPACECRAFT_ID = -82
\begintext

$ chronos -setup chronos.cas -from utc -to et -time 1999 JAN 12 12:00
1999-01-12, 12:01:04.184                                (ET/SCET)

$ chronos -setup chronos.cas -from utc -to sclk -time 1999 JAN 12 12:00
1/1294833883.185                                     (SCLK/SCLK)

$ chronos -setup naif0007.tls cas00085.tsc -sc -82 -from sclk -to utc -time
    1/1294833883.185
1999-01-12 11:59:59.998                                (UTC/SCET)

$ chronos -setup naif0007.tls cas00085.tsc -sc -82 -from sclk -to utc -time
    1/1294833883.185 -format 'YYYY-DOYTHR:MN:SC ::RND' -nolabel
1999-012T12:00:00
```

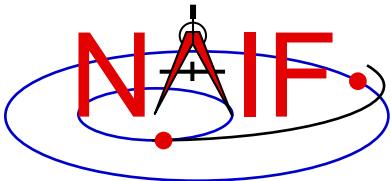


# MKSPK

---

Navigation and Ancillary Information Facility

- *mkspk* may be used to generate an SPK file from any of several types of data, such as discrete state vectors, classical orbital elements, and NORAD two-line elements.
- Use of this program is discussed in a separate tutorial about making SPK files, and in the *mkspk* User's Guide.

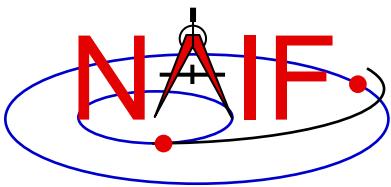


# SPKMERGE

---

Navigation and Ancillary Information Facility

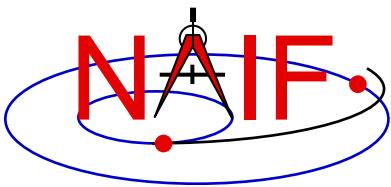
- The contents of an SPK file or set of SPK files may be merged or subsetted using *spkmerge*
  - You can extract an interval of time of interest from a single SPK file or a set of SPK files.
  - You can extract data for one or more objects from a single SPK file or a set of SPK files.
  - You can combine both the time and object selection mechanisms for the greatest flexibility.



# SPKMERGE - Precedence Rule

Navigation and Ancillary Information Facility

- SPK files created with *spkmerge* have no overlapping ephemeris data. The order in which the source files are specified determines precedence when source files have overlapping coverage for a body of interest.
  - **IMPORTANT NOTE:** Data from an **earlier** specified source file take precedence over data from a later specified source file when the new (merged) file is created. **This is different from the usual SPICE precedence rules.**



# SPKMERGE - Example

## Navigation and Ancillary Information Facility

### Terminal Window

```
$ cat spkmerge_cas_example.cmd
;This command file directs spkmerge to take data for
;Cassini, the Sun, the Earth, the Moon, and the Earth-
;Moon barycenter and place them into a single SPK.

leapseconds_kernel = naif0007.tls
spk_kernel        = output.bsp
bodies            = -82, 10, 301, 399, 3
source_spk_kernel = de403s.bsp
source_spk_kernel = 990825A_SCEPH_EM52_JP0.bsp

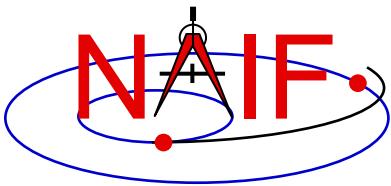
$ spkmerge
SPKMERGE -- SPK Merge Tool, Version 3.2, SPICE Toolkit N0057
```

Enter the name of the command file

```
> spkmerge_cas_example.cmd
```

Creating output.bsp

\$

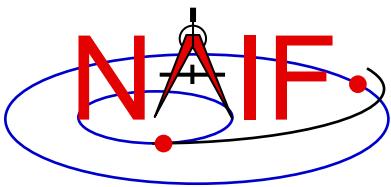


# SPKDIFF

---

Navigation and Ancillary Information Facility

- ***spkdiff* is a command line program for computing the difference between trajectories of two objects obtained from a set of SPK kernels, or for sampling (“dumping”) the trajectory of an object**
- In comparison mode, ***spkdiff* compares trajectories by computing a set of geometric states for a specified body, center and frame over an interval of time with a fixed time step using a set of kernels, then computing another set of geometric states for the same or different body, center, and frame at the same times using the same or another set of kernels, and then subtracting the corresponding states from each other**
  - Depending of the requested output type ***spkdiff* prints to the screen:**
    - » only the maximum differences,
    - » a complete table of differences, or
    - » a statistical analysis of the differences
- In sampling mode, ***spkdiff* computes a set of states for a specified body relative to another body in a specified reference frame over a specified interval with a specified step**



# SPKDIFF - Usage

## Navigation and Ancillary Information Facility

### Terminal Window

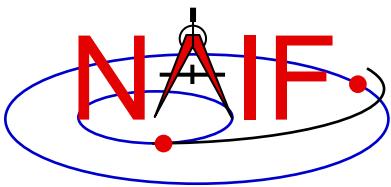
```
% spkdiff
```

... The program usage is:

```
% spkdiff [options] <first SPK name> <second SPK name>
% spkdiff [options] <SPK name>
% spkdiff [options]
```

Options are shown below. ...

```
-k <supporting kernel(s) name(s)>
-b1 <first body name or ID>
-c1 <first center name or ID>
-r1 <first reference frame name>
-k1 <additional supporting kernel(s) for first SPK>
-b2 <second body name or ID>
-c2 <second center name or ID>
-r2 <second reference frame name>
-k2 <additional supporting kernel(s) for second SPK>
-b <interval start time>
-e <interval stop time>
-s <time step in seconds>
-n <number of states: 2 to 1000000 (default: 1000)>
-f <output time format (default: TDB seconds past J2000)>
-d <number of significant digits: 6 to 17 (default: 14)>
-t <report type: basic|stats|dump|dumpvf|dumpc|dumpg (def.: basic|dump)>
```



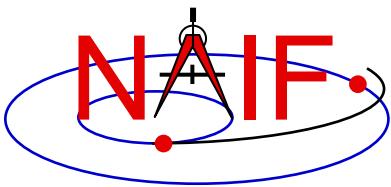
# SPKDIFF – Basic Output Example

## Navigation and Ancillary Information Facility

### Terminal Window

```
$ spkdiff mro_psp.bsp mro_psp_rec.bsp
# Comparison of 1000 'J2000'-referenced geometric states
#
#      of 'MARS RECON ORBITER' (-74) relative to 'MARS BARYCENTER' (4)
#      from SPK 'mro_psp.bsp'
#
# with 1000 'J2000'-referenced geometric states
#
#      of 'MARS RECON ORBITER' (-74) relative to 'MARS BARYCENTER' (4)
#      from SPK 'mro_psp_rec.bsp'
#
# evenly-spaced with 2617.6524668123 second (0d 0h 43m 37.652467s) step size
# within the time interval
#
#      from '2007 APR 01 00:01:05.185 TDB' (228657665.18565 TDB seconds)
#      to   '2007 MAY 01 06:25:00.000 TDB' (231272700.00000 TDB seconds)
#
Relative differences in state vectors:
                                         maximum           average
Position:          8.4872836561757E-05  1.2312974450656E-05
Velocity:         8.5232570159796E-05  1.2314285182022E-05

Absolute differences in state vectors:
                                         maximum           average
Position (km):    3.1341344106404E-01  4.5090516995222E-02
Velocity (km/s):  2.8848827480682E-04  4.2085874877127E-05
```



# SPKDIFF – Dump Output Example

## Navigation and Ancillary Information Facility

### Terminal Window

```
$ spkdiff -t dumpvf mro_psp.bsp mro_psp_rec.bsp | more
# Comparison of 1000 'J2000'-referenced geometric states
#
#      of 'MARS RECON ORBITER' (-74) relative to 'MARS BARYCENTER' (4)
#      from SPK 'mro_psp.bsp'
#
# with 1000 'J2000'-referenced geometric states
#
#      of 'MARS RECON ORBITER' (-74) relative to 'MARS BARYCENTER' (4)
#      from SPK 'mro_psp_rec.bsp'
#
# evenly-spaced with 2617.6524668123 second (0d 0h 43m 37.652467s) step size
# within the time interval
#
#      from '2007 APR 01 00:01:05.185 TDB' (228657665.18565 TDB seconds)
#      to   '2007 MAY 01 06:25:00.000 TDB' (231272700.00000 TDB seconds)
#
# time, down_track_p_diff, normal_to_plane_p_diff, in_plane_p_diff, down_track_v
# _diff, normal_to_plane_v_diff, in_plane_v_diff
2.2865766518565E+08 +4.2593079332056E-02 -9.0540866105197E-05 -3.9705894066565E-04 -8.0803561182349E-08
-1.0394439243989E-07 -3.9614350816493E-05
2.2866028283812E+08 +4.2172435702119E-02 +2.3672255851626E-06 -1.1475679619731E-04 +1.3970238250217E-07
+1.4080506259574E-07 -3.9250157214024E-05
2.2866290049059E+08 +4.4830247467488E-02 +9.1590974014175E-05 -7.3802870365833E-04 +5.7800410436763E-07
-1.1724240528272E-07 -4.2099832045985E-05
2.2866551814305E+08 +4.5968515669515E-02 -1.3529652839857E-04 -7.5686845133612E-05 -4.7565892258325E-07
+3.4127364997784E-08 -4.2529268294482E-05
--More--
```

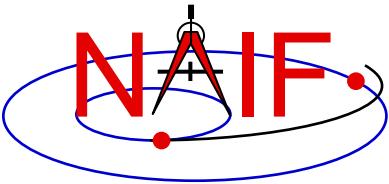


# MSOPCK

---

Navigation and Ancillary Information Facility

- ***msopck* is a program for making CK files from orientation provided in a text file as a time tagged, space-delimited table**
  - It has a simple command line interface
  - It requires all setups to be provided in a setup file that follows the SPICE text kernel syntax
  - It can process quaternions (SPICE and non-SPICE styles), Euler angles, or matrices, tagged with UTC, SCLK, or ET
  - For more details see the “Making a CK File” Tutorial

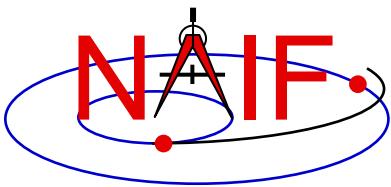


# FRMDIFF

---

Navigation and Ancillary Information Facility

- ***frmdiff* is a command line program for sampling the orientation of a reference frame or for computing the difference between orientations of two reference frames based on provided set(s) of SPICE kernels**
- In sampling mode, *frmdiff* computes a set of transformations from one frame to another frame over a specified interval with a specified time step
- In comparison mode, *frmdiff* computes two sets of transformations for two pairs of “from” and “to” frames and then computes the difference in rotation and angular velocity between these transformations over a specified interval with a specified time step
- Depending on the execution mode and the requested output type, *frmdiff* prints to the screen:
  - only the maximum rotation or the maximum rotation difference,
  - a complete table of rotations or differences (as angle and axis, SPICE- or engineering-style quaternions, matrices, or Euler angles), or
  - a statistical analysis of rotations or differences.

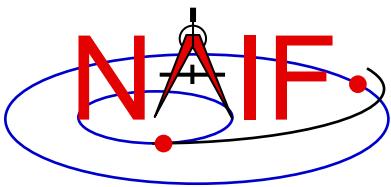


# FRMDIFF - Usage

## Navigation and Ancillary Information Facility

### Terminal Window

```
$ frmdiff
  % frmdiff [options] <first kernel name> <second kernel name>
  % frmdiff [options] <kernel name>
  % frmdiff [options]
where kernel can be a CK, an FK, or a PCK. Options are shown below.
-k <supporting kernel(s) name(s)>
-f1 <first ``from'' frame, name or ID>
-t1 <first ``to'' frame, name or ID>
-c1 <first frame for coverage look up, name or ID>
-k1 <additional supporting kernel(s) for first file>
-f2 <second ``from'' frame, name or ID>
-t2 <second ``to'' frame, name or ID>
-c2 <second frame for coverage look up, name or ID>
-k2 <additional supporting kernel(s) for second file>
-a <compare angular velocities: yes|no>
-m <frame for angular velocities: from|to>
-b <interval start time>
-e <interval stop time>
-n <number of points: 1 to 1000000 (default: 1000)>
-s <time step in seconds>
-f <time format: et|sclk|sclkd|ticks|picture_for_TIMEOUT>
-t <report: basic|stats|dumpaa|dumppm|dumpqs|dumpqo|dumpea|dumpc|dumpg>
-o <rotation axes order (default: z y x)>
-x <units for output angles> (only for -t dumpaa and -t dumpea)
-d <number of significant digits: 6 to 17 (default: 14)>
```



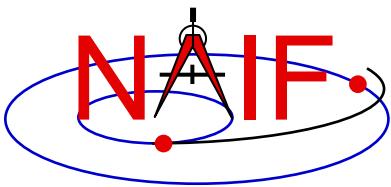
# FRMDIFF – Sampling Example

## Navigation and Ancillary Information Facility

### Terminal Window

```
$ frmdiff -k naif0009.tls DIF_SCLKSCET.00036.tsc di_v17.tf -s 5 -t dumpqo -f sclkd -a yes -m to
dif_sc_2009-01-27.bc > output.txt

$ cat output.txt
#
# Sampling of 16864 rotations
#
#     from 'J2000' (1) to 'DIF_SPACECRAFT' (-140000)
#     computed using
#
#         naif0009.tls DIF_SCLKSCET.00036.tsc di_v17.tf
#         dif_sc_2009-01-27.bc
#
#     with a 5.000000000000 second (0:00:00:05.000000) step size
#     within the non-continuous (with 2 gaps) time period
#
#         from '2009 JAN 27 00:01:06.713' TDB (286286466.71354 TDB seco...
#         to   '2009 JAN 28 00:01:05.346' TDB (286372865.34683 TDB seco...
#
#     including angular velocities relative to 'to' frame.
#
# Times are decimal SCLKs computed using SCLK ID -140.
#
#     time, q_sin1, q_sin2, q_sin3, q_cos, av_x, av_y, av_z
2.8628543276953E+08 +6.9350853049532E-01 +3.7594179111024E-01 -6.1...
2.8628543776953E+08 +6.9350851552324E-01 +3.7594215798843E-01 -6.1...
```



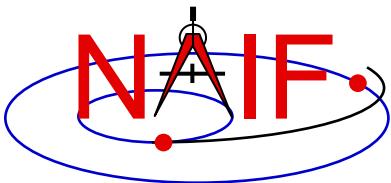
# FRMDIFF – Comparison Example

## Navigation and Ancillary Information Facility

### Terminal Window

```
$ frmdiff -k naif0009.tls cas00130.tsc cas_v40.tf -s 10 -b 2009-JAN-09 00:00 -e 2009-JAN-10 00:00 -t
dumpaa 09009_09025pa_fsiv_lud2.bc 09006_09011ra.bc > output.txt

$ cat output.txt
#
# Comparison of 3143 rotations
#   from 'J2000' (1) to 'CASSINI_SC_COORD' (-82000)
#   computed using
#     naif0009.tls cas00130.tsc cas_v40.tf
#     09009_09025pa_fsiv_lud2.bc
#
#
# with 3143 rotations
#   from 'J2000' (1) to 'CASSINI_SC_COORD' (-82000)
#   computed using
#     naif0009.tls cas00130.tsc cas_v40.tf
#     09006_09011ra.bc
#
#
# with a 10.00000000000 second (0:00:00:10.000000) step size
# within the non-continuous (with 1 gaps) time period
#
#
#   from '2009 JAN 09 15:17:06.359' TDB (284786226.35996 TDB sec...
#   to   '2009 JAN 10 00:01:06.184' TDB (284817666.18419 TDB sec...
#
#
# Times are TDB seconds past J2000.
# angle is shown in radians.
#
#
# time, angle, axis_x, axis_y, axis_z
+2.8478622635996E+08 +5.4958832051797E-05 +8.2101753099566E-01 +4....
+2.8478623635996E+08 +5.4931030131424E-05 +8.2046010733260E-01 +4....
```

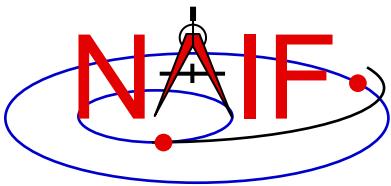


# MKDSK

---

Navigation and Ancillary Information Facility

- ***mkdsk* is a program for making Digital Shape Kernel (DSK) files from digital shape data provided in a text file**
  - It has a simple command line interface
  - It requires all setups to be provided in a setup file that follows the SPICE text kernel syntax
  - It can process shape data in one of the following formats:
    - » plate-vertex table
    - » Gaskell shape model
    - » vertex-facet table
    - » Rosetta/OSIRIS “ver” table
    - » ASCII height grid
  - The N0066 MKDSK can output only Type 2 (plate model) DSKs
  - For more details see the MKDSK User’s Guide

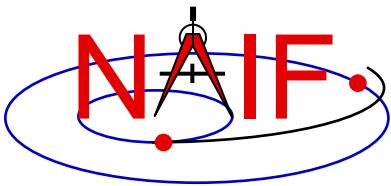


# DSKEXP

---

Navigation and Ancillary Information Facility

- ***dskexp* is a program for exporting digital shape data from a DSK file to a text file**
  - It has a simple command line interface
  - The N0066 DSKEXP can export data only from Type 2 (plate model) DSKs
  - It can output shape data in one of the following formats
    - » plate-vertex table
    - » vertex-facet table
    - » Rosetta/OSIRIS “ver” table
  - It creates a separate output file for each DSK segment
  - For more details see DSKEXP User’s Guide

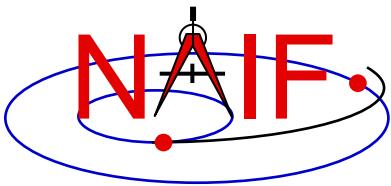


# Kernel Summary Applications

Navigation and Ancillary Information Facility

The contents of binary kernels can be summarized using kernel summary tools.

- ***brief*** displays the bodies and associated time coverage in an SPK file or set of SPK files.
  - *brief* also works on binary PCK files
- ***ckbrief*** displays the structure(s) and associated time coverage in a CK file or set of CK files.
- ***dskbrief*** displays a summary of spatial coverage and attributes for a DSK file or set of DSK files.
- ***spacit*** displays a segment by segment summary of the contents of a CK, SPK, binary PCK, or EK/ESQ file.
  - *spacit* also identifies the SPK or CK data type present in each segment.
  - *spacit* does not work on DSK files.

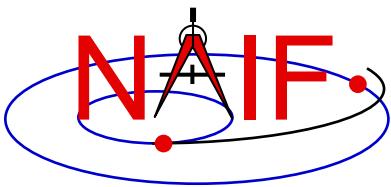


# BRIEF

---

Navigation and Ancillary Information Facility

- ***brief* is a command line program for summarizing the contents of SPK or binary PCK files**
- The files to be summarized can be listed on the command line, given in a meta-kernel provided on the command line, or provided in a list file
- ***brief* provides command line options for**
  - displaying coverage boundaries as date UTC, DOY UTC, or ET seconds past J2000 (default time format is calendar ET)
    - » to display time as UTC an LSK file must be provided on the command line
  - displaying centers of motion along with the bodies
  - treating all input files as if they were a single file
  - displaying a summary only for files covering a specified time or time range or containing data for a specified body
  - displaying a summary in tabular format or grouped by coverage
  - and many others ...



# BRIEF - Usage

## Navigation and Ancillary Information Facility

### Terminal Window

```
$ brief
```

```
...
```

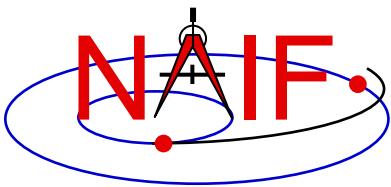
BRIEF is a command-line utility program that displays a summary for one or more binary SPK or binary PCK files. The program usage is:

```
% brief [-options] file [file ...]
```

The most useful options are shown below. For the complete set of options, run BRIEF with the `-h` option. The order of options is not significant. The case of option keys is significant: they must be lowercase as shown below.

- `-c` display centers of motion/relative-to frames
- `-t` display summary in a tabular format
- `-a` treat all files as a single file
- `-utc` display times in UTC calendar date format (needs LSK)
- `-utcdoy` display times in UTC day-of-year format (needs LSK)
- `-etsec` display times as ET seconds past J2000

An LSK file must be provided on the command line to display times in UTC formats. FK file(s) must be provided on the command line to display names of any frames that are not built into the Toolkit.



# BRIEF - Example

## Navigation and Ancillary Information Facility

```
Terminal Window

$ brief de405s.bsp m01_cruise.bsp

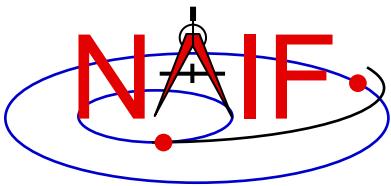
BRIEF -- Version 3.0.0, January 14, 2008 -- Toolkit Version N0063

Summary for: de405s.bsp

Bodies: MERCURY BARYCENTER (1)      SATURN BARYCENTER (6)      MERCURY (199)
        VENUS BARYCENTER (2)       URANUS BARYCENTER (7)      VENUS (299)
        EARTH BARYCENTER (3)      NEPTUNE BARYCENTER (8)     MOON (301)
        MARS BARYCENTER (4)       PLUTO BARYCENTER (9)      EARTH (399)
        JUPITER BARYCENTER (5)    SUN (10)                  MARS (499)
        Start of Interval (ET)           End of Interval (ET)
        -----
        1997 JAN 01 00:01:02.183          2010 JAN 02 00:01:03.183

Summary for: m01_cruise.bsp

Body: MARS SURVEYOR 01 ORBITER (-53)
      Start of Interval (ET)           End of Interval (ET)
      -----
      2001 APR 07 16:25:00.000          2001 OCT 24 05:00:00.000
```

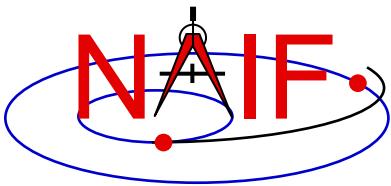


# CKBRIEF

---

Navigation and Ancillary Information Facility

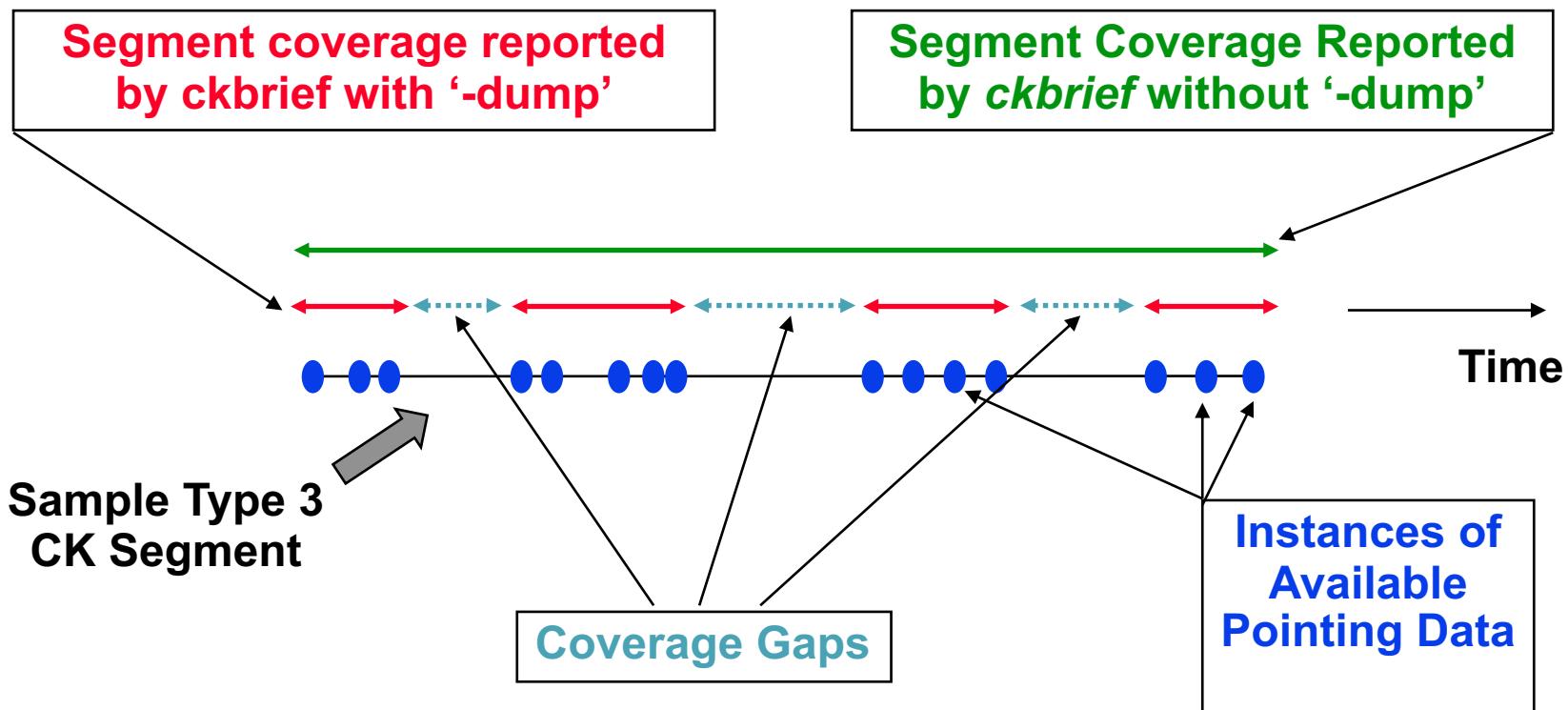
- ***ckbrief* is a command line program for summarizing the contents of CK files**
- The files to be summarized can be listed on the command line, given in a meta-kernel provided on the command line, or provided in a list file
- ***ckbrief* provides command line options for**
  - displaying coverage at interpolation interval level
  - displaying coverage boundaries using UTC, DOY UTC, SCLK, or encoded SCLK (default time format is calendar ET)
    - » To display times as ET, UTC, or SCLK, both an LSK file and a SCLK file(s) must be provided on the command line
  - displaying frames with respect to which orientation is provided
  - displaying the names of the frames associated with CK IDs
    - » An FK file(s) defining these frames must be provided on the command line
  - treating all input CK files as if they were a single file
  - displaying summary only for files with data for a given CK ID
  - and many others ...

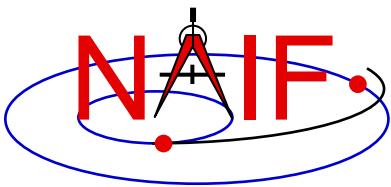


# CKBRIEF – Interval Summary

Navigation and Ancillary Information Facility

- There often are coverage gaps within a CK segment
- Using the ‘-dump’ option allows one to get a complete list of continuous coverage intervals for each segment





# CKBRIEF – Usage

## Navigation and Ancillary Information Facility

### Terminal Window

```
$ ckbrie
```

```
...
```

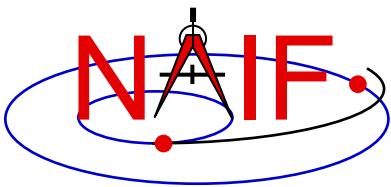
CKBRIEF is a command-line utility program that displays a summary for one or more binary CK files. The program usage is:

```
% ckbrie [-options] file [file ...]
```

The most useful options are shown below. For the complete set of options, run CKBRIEF with the -h option. The order of options is not significant. The option keys must be lowercase as shown below.

-dump	display interpolation intervals
-rel	display relative-to frames (may need FK)
-n	display frames associated with CK IDs (may need FK)
-t	display summary in a tabular format
-a	treat all files as a single file
-utc	display times in UTC calendar date format (needs LSK&SCLK)
-utcdoy	display times in UTC day-of-year format (needs LSK&SCLK)
-sclk	display times as SCLK strings (needs SCLK)

LSK and SCLK files must be provided on the command line to display times in UTC, ET, or SCLK formats. FK file(s) must be provided on the command line to display names of any frames that are not built into the Toolkit.



# CKBRIEF – Example

## Navigation and Ancillary Information Facility

```
Terminal Window

$ ckbrieft -sclk 981116_981228pa.bc sclk.ker

CKBRIEF -- Version 5.0.0, February 11, 2009 -- Toolkit Version N0063

Summary for: 981116_981228pa.bc

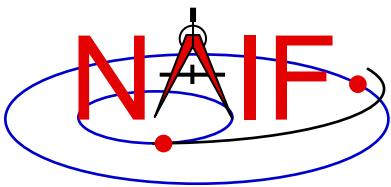
Object: -82000
    Interval Begin SCLK      Interval End SCLK      AV
    -----
    1/1289865849.116        1/1293514473.118      N

$ ckbrieft -utc sclk.ker naif0007.tls 990817_990818ra.bc

CKBRIEF -- Version 5.0.0, February 11, 2009 -- Toolkit Version N0063

Summary for: 990817_990818ra.bc

Object: -82000
    Interval Begin UTC      Interval End UTC      AV
    -----
    1999-AUG-17 17:30:01.418 1999-AUG-17 23:05:42.039 N
    1999-AUG-17 23:05:45.289 1999-AUG-18 06:06:05.874 N
    1999-AUG-18 06:06:09.124 1999-AUG-18 11:52:17.741 N
    1999-AUG-18 11:52:20.991 1999-AUG-18 13:30:00.953 N
```



# CKBRIEF - '-dump' Example

## Navigation and Ancillary Information Facility

Terminal Window

```
$ ckbrief mgs_spice_c_kernel_2004-099.bc MGS_SCLKSCET.00053.tsc naif0007.tls -dump  
-rel -utc

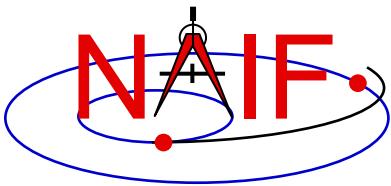
CKBRIEF -- Version 5.0.0, February 11, 2009 -- Toolkit Version N0063

Summary for: mgs_spice_c_kernel_2004-099.bc

Segment No.: 1

Object: -94000

      Interval Begin UTC      Interval End UTC      AV  Relative to FRAME
-----  -----
2004-APR-08 00:00:59.809 2004-APR-08 06:53:47.805 Y   J2000
2004-APR-08 06:54:07.805 2004-APR-08 06:54:07.805 Y   J2000
2004-APR-08 06:54:19.805 2004-APR-08 06:54:35.805 Y   J2000
2004-APR-08 06:54:51.805 2004-APR-08 06:54:55.805 Y   J2000
2004-APR-08 06:55:07.805 2004-APR-08 06:55:07.805 Y   J2000
2004-APR-08 06:55:23.805 2004-APR-08 06:55:23.805 Y   J2000
2004-APR-08 06:55:35.805 2004-APR-08 11:59:55.802 Y   J2000
2004-APR-08 12:00:55.802 2004-APR-08 23:59:55.795 Y   J2000
```

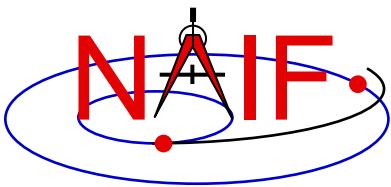


# DSKBRIEF

---

Navigation and Ancillary Information Facility

- ***dskbrief* is a command line program for summarizing the spatial coverage and additional attributes of Digital Shape Kernel (DSK) files**
- DSK files to be summarized can be listed on the command line or given in a meta-kernel provided on the command line
  - Additional text kernels containing body, frame, and surface name-ID associations must also be provided to produce complete summary output
- ***dskbrief* provides command line options for**
  - generating extended, full, and segment-by-segment summaries
  - treating all input files as if they were a single file
  - displaying gaps in spatial coverage
  - controlling the number of significant digits in the output
  - and a few others



# DSKBRIEF - Usage

## Navigation and Ancillary Information Facility

### Terminal Window

```
$ dskbrief -u
```

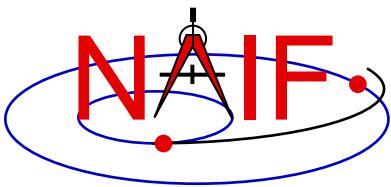
...

DSKBRIEF is a command-line utility program that displays a summary of one or more binary DSK files. The program usage is:

```
% dskbrief [options] file [file...]
```

The available options are shown below. The order of options is not significant. The option keys must be lowercase as shown below.

- a Treat all DSK files as a single file.
- gaps Display coverage gaps. Applies only when -a is used.
- ext Display extended summaries: these include data type, data class, and time bounds. This option applies to summaries of groups of DSK segments.
- tg Require segment time bounds to match when grouping segments.
- seg Display a segment-by-segment summary.
- full Display a detailed summary for each segment, including data-type-specific parameters. This option implies a segment-by-segment summary.
- d <n> Display n significant digits of floating point values.
- v Display the version of this program.
- h Display help text.
- u Display usage text.



# DSKBRIEF - Example

## Navigation and Ancillary Information Facility

### Terminal Window

```
$ dskbrief ROS(CG)_M004_NSPCESA_N_V1.BDS ROS(LU)_M003_OSPCLAM_N_V1.BDS .../FK/ROS_V25.TF
```

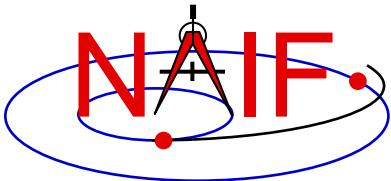
...

Summary for: ROS(CG)\_M004\_NSPCESA\_N\_V1.BDS

Body: 1000012 (CHURYUMOV-GERASIMENKO)  
Surface: 11000 (ROS(CG)\_M004\_NSPCESA\_N\_V1)  
Reference frame: 67P/C-G\_CK  
Coordinate system: Planetocentric Latitudinal  
Min, max longitude (deg): 0.00000 360.000  
Min, max latitude (deg): -90.0000 90.0000  
Min, max radius (km): 4.86351E-01 2.65365

Summary for: ROS(LU)\_M003\_OSPCLAM\_N\_V1.BDS

Body: 2000021 (LUTETIA)  
Surface: 1011 (ROS(LU)\_M003\_OSPCLAM\_N\_V1)  
Reference frame name N/A; ID code: -2260021  
Coordinate system: Planetocentric Latitudinal  
Min, max longitude (deg): 0.00000 360.000  
Min, max latitude (deg): -90.0000 90.0000  
Min, max radius (km): 33.2395 64.6538

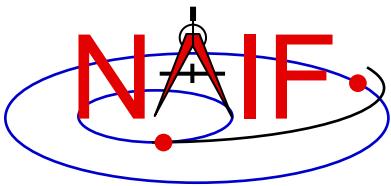


# SPACIT

---

Navigation and Ancillary Information Facility

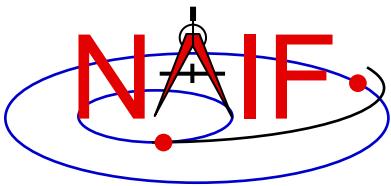
- ***spacit* may be used to obtain a more detailed summary of an SPK or CK file than that offered by *brief* or *ckbrief*, respectively**
  - *spacit* may also be used to summarize a binary PCK or an EK/ESQ.
  - *spacit* is an interactive program
    - » It will prompt you for all needed inputs
- ***spacit* may also be used to manage comments, and to convert between binary and transfer format**



# Comment Manipulation Tools

Navigation and Ancillary Information Facility

- Every kernel should contain metadata – called “comments” – describing the file contents, intended usage, etc.
- In binary kernels – SPKs, CKs, binary PCKs, DSKs and EKs – comments are stored in a special area of the file called the “comment area.”
- *commnt* can read, extract, add, or delete comments stored in the comment area
  - **Caution:** you cannot add or delete comments if the kernel file is not in native format for the machine on which you’re working.
    - » You can convert a non-native binary format file to native binary format by converting the file to “transfer format” using *toxfr* and then converting it back to binary format using *tobin*.
    - » Or use the *bingo* utility (available only from the NAIF website and Fortran toolkits).



# COMMNT

---

Navigation and Ancillary Information Facility

- ***commnt* is both a command line utility and an interactive menu-driven program**
- **In command line mode, *commnt* provides options to**
  - print comments to the screen  
    `$ commnt -r kernel_file`
  - extract comments to a text file  
    `$ commnt -e kernel_file text_file`
  - add comments from a text file  
    `$ commnt -a kernel_file comment_file`
  - delete comments  
    `$ commnt -d kernel_file`
- **Important**
  - When comments are added, they are appended at the end of the existing comments
  - Comments should be deleted ONLY if being replaced with better comments



# COMMNT - Command Line Example

## Navigation and Ancillary Information Facility

### Terminal Window

```
$ commnt -r de405.bsp | more
; de405.bsp LOG FILE
;
; Created 1999-10-03/14:31:58.00.
;
; BEGIN NIOSPK COMMANDS

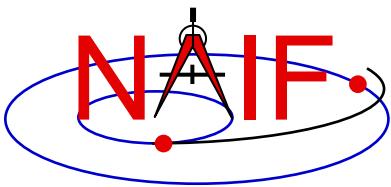
LEAPSECONDS_FILE      = /kernels/gen/lsk/naif0007.tls
SPK_FILE               = de405.bsp
SOURCE_NIO_FILE        = /usr2/nio/gen/de405.nio
BODIES                 = 1 2 3 4 5 6 7 8 9 10 301 399 199 299 499
BEGIN_TIME             = CAL-ET 1950 JAN 01 00:00:41.183
END_TIME               = CAL-ET 2050 JAN 01 00:01:04.183

; END NIOSPK COMMANDS
```

A memo describing the creation of the DE405 generic planet ephemeris is available from NAIF or from the author: Dr. Myles Standish of JPL's Solar System Dynamics Group. Because this memo was produced using the TeX processor and includes numerous equations

>>> Beginning of extract from Standish's DE405 memo <<

...



# COMMNT – Interactive Example

Navigation and Ancillary Information Facility

Terminal Window

```
$ commnt

Welcome to COMMNT Version: 6.0.0
(Spice Toolkit N0050)

COMMNT Options

( Q ) Quit.
( A ) Add comments to a binary file.
( R ) Read the comments in a binary file.
( E ) Extract comments from a binary file.
( D ) Delete the comments in a binary file.

Option: E

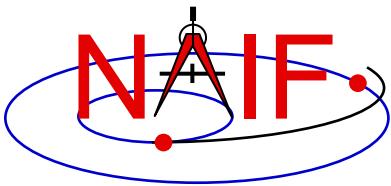
Enter the name of the binary file.

Filename? de405.bsp

Enter the name of the comment file to be created.

Filename? de405_comments.txt

The comments were successfully extracted.
```

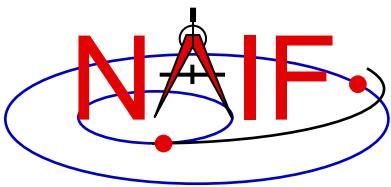


# File Format Conversion Tools

Navigation and Ancillary Information Facility

- With modern Toolkits (N0052 and later) the porting of DAF-based binary kernels\* between computers having dissimilar binary standards is usually not necessary.
  - The advent of binary kernel readers that detect the binary style and do run-time translation if needed generally makes porting unnecessary for DAF-based types.
  - Refer to the “Porting Kernels” tutorial for more on this topic.
- If true porting is needed because you must modify or append to a kernel:
  - use *toxfr* on the source computer and *tobin* on the destination computer
  - or use *bingo* on the destination computer
    - » NOTE: bingo is NOT available in generic Toolkits other than Fortran; it must be downloaded from the NAIF website

\* DAF-based binary kernels are SPK, CK and binary PCK

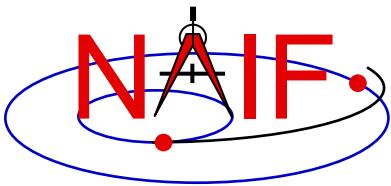


---

Navigation and Ancillary Information Facility

# Non-Toolkit Applications

January 2020



# Summary

---

Navigation and Ancillary Information Facility

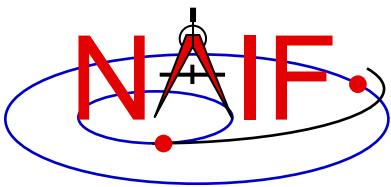
- NAIF makes available a set of applications ***not*** included in the Toolkits. This set includes programs for:
  - making, modifying, validating, inspecting, and converting SPK files:
    - » *pinpoint, dafcat, bspidmod, dafmod, spy, oem2spk, spk2oem*
  - making and modifying CK files
    - » *prediCkt, ckslicer, ckspanit, dafcat, cksmrg, dafmod*
  - making SCLK files
    - » *makclk*
  - merging DSK files
    - » *d lacat*
  - computing derived quantities
    - » *orbnum, optics, spy*
  - determining SPICE kernel type and binary format, and converting between native and non-native formats
    - » *archtype, bff, bingo*
- Executables and User's Guides are on the NAIF server at:
  - <https://naif.jpl.nasa.gov/naif/utilities.html>



# Using Non-Toolkit Apps

Navigation and Ancillary Information Facility

- All of these apps are meant to be used as operating system shell executables
  - One generally cannot run these within IDL or MATLAB—run them from an operating system shell
    - » In some cases you can run from within IDL or MATLAB, but this is not recommended:
      - In IDL, use the “spawn” command
      - In MATLAB, use the “system” command



# PINPOINT

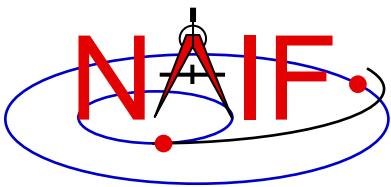
---

Navigation and Ancillary Information Facility

- ***pinpoint* is a program for creating SPK files and topocentric frames FK files for objects for which the position is a constant offset with respect to another object**
  - Ground stations
  - Landing sites, sites along a rover path
  - Relative positions of manipulator joints, etc.
- ***pinpoint* is a command line program with the following usage:**

```
pinpoint -def deffile -spk spkfile [-pck tkfile] [-fk fk] [flags]
```

  - “deffile” is an input definitions file following text kernel file format and containing a set of keywords defining ID, center, reference frame, position (as XYZ or Gaussian Lat/Lon/Alt) and time coverage boundaries, and optionally velocity and topocentric frame axes specifications, for one or more objects
    - » The contents of “deffile” are included in the comment area
  - “spkfile” is an output SPK file containing a type 8 SPK segment for each of the objects specified in the “deffile”
  - “tkfile” is an optional input PCK file (needed if positions in the “deffile” are given as Lat/Lon/Alt) or FK file (needed if one or more of the frames specified in “deffile” is not one of the frames built into the Toolkit)
  - “fk” is an optional output topocentric frames FK file



# PINPOINT Example

## Navigation and Ancillary Information Facility

### Terminal Window

```
$ more mer1_meridiani.def

Sample PINPOINT input for MER-1 landing site coordinates.

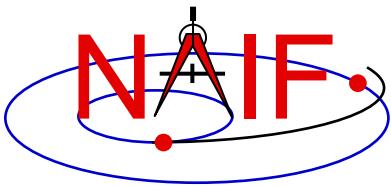
\begindata
    SITES      = ( 'LS' )
    LS_CENTER  = 499
    LS_FRAME   = 'IAU_MARS'
    LS_IDCODE  = -253900
    LS_XYZ     = ( +3.3764222E+03  -3.2664876E+02  -1.1539218E+02  )
    LS_BOUNDS  = ( @2001-01-01-00:00:00.000, @2100-01-01-00:00:00.000 )

\begintext

$ pinpoint -def mer1_meridiani.def -spk mer1_meridiani.bsp

$ brief mer1_meridiani.bsp
Brief. Version: 2.2.0          (SPICE Toolkit N0057)

Summary for: mer1_meridiani.bsp
Body: -253900* w.r.t. MARS (499)
      Start of Interval (ET)           End of Interval (ET)
      -----                    -----
      2001 JAN 01 00:00:00.000        2100 JAN 01 00:00:00.000
```



# DAFCAT

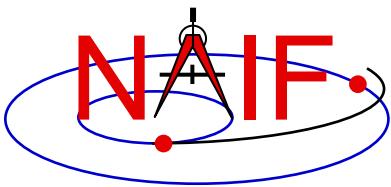
---

Navigation and Ancillary Information Facility

- ***dafcat* is a program for concatenating binary DAF files by simply copying all data segments from all input files, in the order they are provided, into the output file**
  - ***dafcat* works on SPKs, CKs, and binary PCKs**
    - » It will not merge different types of kernels together, i.e. it will not merge SPKs with CKs, CKs with PCKs, etc.
    - » For merging SPKs, in most cases *spkmerge* should be used instead because it provides a much more powerful and sophisticated capability
- ***dafcat* is a command line program with the following usage**

```
dafcat output_file
```

  - “*output\_file*” is the output file name and is the program’s only argument
  - Input file names are provided from standard input
    - » this is very convenient for use with Unix shell pipes
- ***dafcat* does not put any information into the comment area of the output file, leaving this responsibility to the user (use *commnt* to do so)**



# DAFCAT Example: SPK

## Navigation and Ancillary Information Facility

```
Terminal Window

$ dafcat m01_merged.bsp

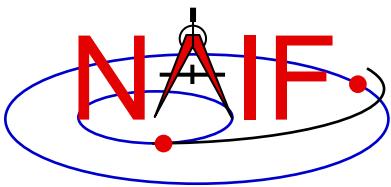
DAF binary files concatenation program version 1.00

spk_m_od33905-33993_rec_v1.bsp
spk_m_od33992-34065_rec_v1.bsp
^D
Concatenating files:
  spk_m_od33905-33993_rec_v1.bsp
  spk_m_od33992-34065_rec_v1.bsp
to:
  m01_merged.bsp

$ ls -1 spk_m_od*_rec_v1.bsp | dafcat m01_merged_2.bsp

DAF binary files concatenation program version 1.00

Concatenating files:
  spk_m_od32371-32458_rec_v1.bsp
  ...
to:
  m01_merged_2.bsp
```



# DAFCAT Example: CK

Navigation and Ancillary Information Facility

Terminal Window

```
$ dafcat m01.bc

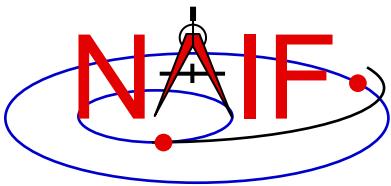
DAF binary files concatenation program version 1.00

m01_sc_2004-04-20.bc
m01_sc_2004-04-21.bc
^D
Concatenating files:
  m01_sc_2004-04-20.bc
  m01_sc_2004-04-21.bc
to:
  m01.bc

$ ls -1 m01_sc_2004-04-2*.bc | dafcat m01.bc

DAF binary files concatenation program version 1.00

Concatenating files:
  m01_sc_2004-04-20.bc
  m01_sc_2004-04-21.bc
to:
  m01.bc
```



# DLACAT

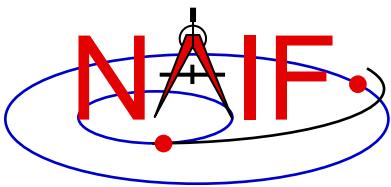
---

Navigation and Ancillary Information Facility

- ***dlacat* is a program for concatenating binary DLA files by simply copying all data segments from all input files, in the order they are provided, into the output file**
  - Works on DSKs
- ***dlacat* is a command line program with the following usage**

```
dlacat output_file
```

  - “*output\_file*” is the output file name and is the program’s only argument
  - Input file names are provided from standard input
    - » this is very convenient for use with Unix shell pipes
- ***dlacat* does not put any information into the comment area of the output file, leaving this responsibility to the user (use *commnt* to do so)**



# DLACAT Example: DSK

## Navigation and Ancillary Information Facility

### Terminal Window

```
$ dlacat phoebe_shape.bds
```

DLA binary files concatenation program version 1.00

phoebe\_shape\_part1.bds

phoebe\_shape\_part2.bds

^D

Concatenating files:

phoebe\_shape\_part1.bds

phoebe\_shape\_part2.bds

to:

phoebe\_shape.bds

```
$ ls -1 phoebe_shape_part?.bds | dlacat phoebe_shape_2.bds
```

DLA binary files concatenation program version 1.00

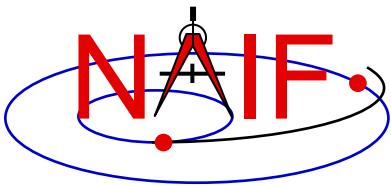
Concatenating files:

phoebe\_shape\_part1.bds

phoebe\_shape\_part2.bds

to:

phoebe\_shape\_2.bds



# BSPIDMOD

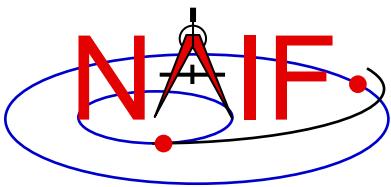
---

Navigation and Ancillary Information Facility

- ***bspidmod* is a program for altering the object IDs in a binary SPK file**
  - It can be used to modify IDs in an SPK file(s) produced with a “bogus” spacecraft ID (or a simulation spacecraft ID)
  - It can be used to replace “official” IDs with “bogus” IDs if two different trajectories for the same object need to be used in the same program at the same time (for example for comparison, such as is done by *spkdiff*)
- ***bspidmod* has the following usage:**

```
bspidmod -spki inpspk -idi inpid -ido outid -mod item -oflg
```

- “**inpspk**” is the input SPK file; “**inpid**” and “**outid**” are the current ID and new ID
- “**item**” indicates which IDs are to be replaced:
  - TARGET -- only target IDs are replaced,
  - CENTER -- only center IDs are replaced, or
  - OBJECT -- both target and center IDs are replaced
- » Replacements are made only when “**inpid**” matches an ID found in the input SPK
- “**-oflg**” flag indicating that changes should be made directly in the input file; if not specified, the program produces an output file with name that has “**\_out**” appended to the name of the input file
  - » In order for changes to be made in the input file it must be in the native binary format; if it is not, *bingo* may be used to convert it to the native binary format
- A note stating which IDs were modified is put in the comment area



# BSPIDMOD Example

## Navigation and Ancillary Information Facility

Terminal Window

```
$ brief mer2_crus_sim_id.bsp
Brief. Version: 2.2.0          (SPICE Toolkit N0057)

Summary for: mer2_crus_sim_id.bsp

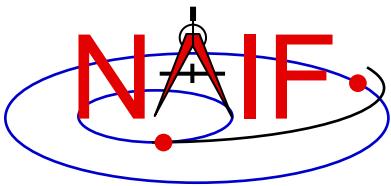
Body: -255
      Start of Interval (ET)           End of Interval (ET)
      -----
      2003 JUL 09 00:15:00.000        2004 JAN 04 04:25:42.557

$ bspidmod -spki mer2_crus_sim_id.bsp -idi -255 -ido -254 -mod target -oflg
The file mer2_crus_sim_id.bsp has been updated.

$ brief mer2_crus_sim_id.bsp
Brief. Version: 2.2.0          (SPICE Toolkit N0057)

Summary for: mer2_crus_sim_id.bsp

Body: MER-2 (-254)
      Start of Interval (ET)           End of Interval (ET)
      -----
      2003 JUL 09 00:15:00.000        2004 JAN 04 04:25:42.557
```

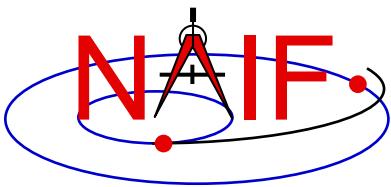


# DAFMOD

---

Navigation and Ancillary Information Facility

- ***dafmod* is a program for altering selected segment attributes in a binary SPK, CK, or PCK file**
  - In an SPK file it can alter the target, center, or reference frame ID
  - In a CK or binary PCK file it can alter the object or reference frame ID
- ***dafmod* is an interactive program. When executed it prompts the user for**
  - name of the file to be modified
  - “item” to be modified
    - » the set of items depends on the kernel type
  - “old” item value
  - “new” item value
- ***dafmod* puts into the comment area a warning note stating which items in which segments of the file were changed**
- ***dafmod* works only on files in native binary format**
  - *bingo* may be used to convert a non-native binary kernel to native binary format



# DAFMOD Example: SPK

## Navigation and Ancillary Information Facility

Terminal Window

```
$ brief mer2_crus_sim_id.bsp

Summary for: mer2_crus_sim_id.bsp

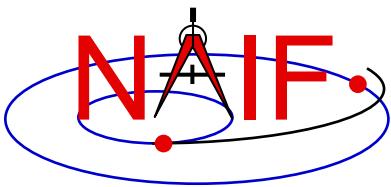
Body: -255
...
$ dafmod

DAFMOD -- Version 2.0.0, January 30, 2008 -- Toolkit Version N0063
(... banner providing usage instructions ...)
1) File      : mer2_crus_sim_id.bsp
2) Item      : target
3) Old Value: -255
4) New Value: -254
The file mer2_crus_sim_id.bsp has been updated.

$ brief mer2_crus_sim_id.bsp

Summary for: mer2_crus_sim_id.bsp

Body: MER-2 (-254)
```



# DAFMOD Example: CK

## Navigation and Ancillary Information Facility

```
Terminal Window

$ ckbrief -rel mro_sc_pred.bc mro.tsc naif0009.tls

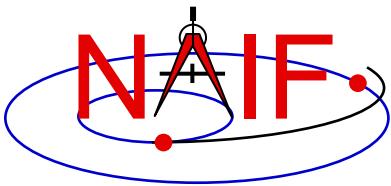
Summary for: mro_sc_pred.bc
...
2009-AUG-15 23:31:02.347 2009-AUG-30 00:00:58.388 Y -74900

$ dafmod

DAFMOD -- Version 2.0.0, January 30, 2008 -- Toolkit Version N0063
(... banner providing usage instructions ...)
1) File      : mro_sc_pred.bc
2) Item      : frame
3) Old Value: -74900
4) New Value: 16
The file mro_sc_pred.bc has been updated.

$ ckbrief -rel mro_sc_pred.bc mro.tsc naif0009.tls

Summary for: mro_sc_pred.bc
...
2009-AUG-15 23:31:02.347 2009-AUG-30 00:00:58.388 Y MARSIAU
```

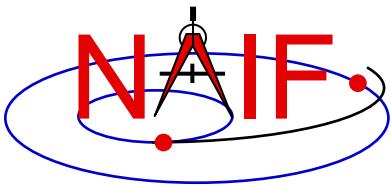


# SPY

---

## Navigation and Ancillary Information Facility

- **Spy is a command line program for validating, inspecting, and analyzing SPK files**
- **Spy can:**
  - check SPK files
    - » Validate SPK structure
    - » Check sampled data for bounds violations
    - » Locate invalid double precision numbers
  - sample data from a set of loaded kernels
    - » Sample position, distance, velocity, derived velocity, speed, acceleration, acceleration magnitude, osculating elements
  - dump SPK file contents
    - » Data
    - » Summary information
    - » Comment area
    - » Bookkeeping information
  - find some geometric events
    - » Distance: find times when specified constraints on observer-target distance are met
    - » Elevation: find times when specified constraints on elevation of target in specified frame are met

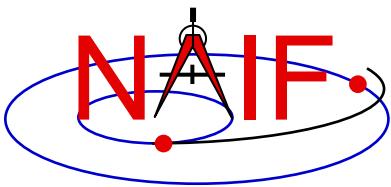


# SPY: Selected Features

---

Navigation and Ancillary Information Facility

- **Operating modes**
  - Interactive, batch, shell command line
- **Auxiliary files**
  - Start-up file, command files, log file, save file
- **Interactive command support**
  - Command history: recall, repetition, and command editing; editor selection; command error detection; (limited) automatic command error correction
- **User default support**
  - Set, show, reset default values
- **Input options**
  - Define user symbols in commands
  - Embed prompts in commands
- **Output options**
  - Dump subsets of SPK data
  - Show epoch and packet deltas in data dumps
  - Set sample count or density
  - Set time and number formats
  - Set angular units
  - Set coordinate system for sampled data
  - Control error diagnostic verbosity
- **Online help: command language summary**



# SPY Example: Dump SPK Data

## Navigation and Ancillary Information Facility

### Terminal Window

```
Spy > dump data spk testspk.bsp segment index 13 stop packet 2;
```

```
Dump of SPK File testspk.bsp
```

```
=====
Segment number 13
```

```
-----
Segment Summary:
```

```
Segment ID      : SPY test segment: type 18 subtype 0
Target Body     : Body 1800
Center Body     : Body 1899
Reference Frame : Frame 17, ECLIPJ2000
SPK Data Type   : Type 18
    Description  : Mex/Rosetta Hermite/Lagrange Interpolation
UTC Start Time  : 2000 JAN 01 11:59:05.816
UTC Stop Time   : 2000 JAN 01 12:32:15.816
ET Start Time   : 2000-JAN-01 12:00:10.000000 (TDB)
ET Stop Time    : 2000-JAN-01 12:33:20.000000 (TDB)
DAF Begin Address: 35287
DAF End Address  : 37890
-----
```

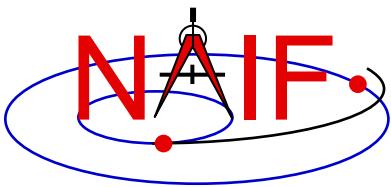
```
Segment Parameters:
```

```
Packet Count     : 200
Directory Count   : 1
Window Size - 1  : 6
Polynomial Degree: 13
Subtype          : 0
    Description  : Hermite interpolation, 12-element packets
-----
```

```
Time Tags and Packets:
```

```
State Components: Position X, Y, Z (km)
                  Velocity X, Y, Z (km/s)
                  Velocity X, Y, Z (km/s)
                  Accel. X, Y, Z (km/s^2)

1      2000-JAN-01 12:00:10.000000 (TDB)  1.00103333E+03  1.00203333E+03  1.00303333E+03  1.00403333E+03  1.00503333E+03
     1.00603333E+03                           1.00703333E+03  1.00803333E+03  1.00903333E+03  1.01003333E+03  1.01103333E+03
     1.01203333E+03
2      2000 JAN 01 12:00:20.000000 (TDB)  2.00103333E+03  2.00203333E+03  2.00303333E+03  2.00403333E+03  2.00503333E+03
     2.00603333E+03                           2.00703333E+03  2.00803333E+03  2.00903333E+03  2.01003333E+03  2.01103333E+03
     2.01203333E+03
```



# SPY Example: Sample State Vectors

## Navigation and Ancillary Information Facility

### Terminal Window

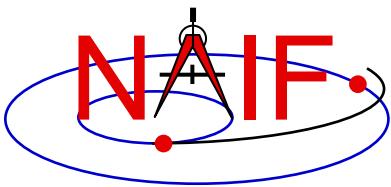
```
Spy > load naif0009.tls;
Spy > load de421.bsp;
Spy > sample states
    observer earth
    target moon
    start time 2008 oct 28 00:00:00.000000 TDB
    stop time 2008 oct 28 00:01:00.000000 TDB
    frame eclipJ2000
    aberration correction none
    coordinates latitudinal
    time format numeric E23.16
    number format F13.6
    step size 10.0;
```

#### Sample STATE Results

```
=====
Target          : moon
Observer        : earth
Frame           : eclipJ2000
Aberration Correction: none
Coordinate System : latitudinal
=====
```

0.2784240000000000E+09	395800.315095	-156.260092	-4.660937	0.035837	0.000145	-0.000005
0.2784240100000000E+09	395800.673459	-156.258644	-4.660983	0.035836	0.000145	-0.000005
0.2784240200000000E+09	395801.031820	-156.257196	-4.661028	0.035836	0.000145	-0.000005
0.2784240300000000E+09	395801.390177	-156.255748	-4.661074	0.035836	0.000145	-0.000005
0.2784240400000000E+09	395801.748532	-156.254300	-4.661120	0.035835	0.000145	-0.000005
0.2784240500000000E+09	395802.106883	-156.252851	-4.661165	0.035835	0.000145	-0.000005
0.2784240600000000E+09	395802.465231	-156.251403	-4.661211	0.035835	0.000145	-0.000005

```
=====
```



# SPY Example: Check SPK Integrity

## Navigation and Ancillary Information Facility

### Terminal Window

```
Spy > check integrity spk testspk.bsp;
```

```
Structure Inspection of SPK File testspk.bsp
```

```
=====
Segment Number 11
-----
```

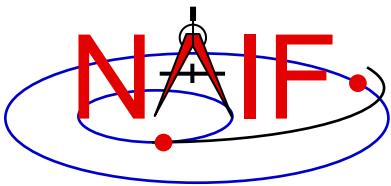
```
Segment Summary:
```

```
Segment ID      : SPY test segment: type 15
Target Body     : Body 1501
Center Body     : Body 1599
Reference Frame : Frame 17, ECLIPJ2000
SPK Data Type   : Type 15
    Description  : Two-Body with J2 Precession
UTC Start Time  : 2000 JAN 01 11:59:05.816
UTC Stop Time   : 2000 JAN 01 12:32:15.816
ET Start Time   : 2000-JAN-01 12:00:10.000000 (TDB)
ET Stop Time    : 2000-JAN-01 12:33:20.000000 (TDB)
DAF Begin Address: 35259
DAF End Address  : 35274
-----
```

```
%% Error: Invalid Unit Periapsis Pole Vector
```

```
The periapsis pole vector should have unit length but in fact has length 4.58257569E+04.
```

```
=====
One error diagnostic and no warnings generated for SPK file testspk.bsp
=====
```



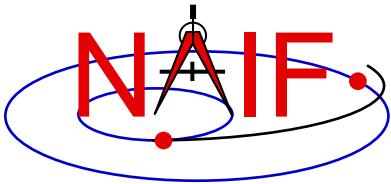
# OEM2SPK and SPK2OEM

---

Navigation and Ancillary Information Facility

- ***oem2spk* is a program for converting a CCSDS\* “Orbit Ephemeris Message” (OEM) text file to a Type 9 or 13 SPICE SPK file**
  - It is a command line program using a setup file to specify conversion parameters
  - It can process OEM versions 1 and 2
  - It is primarily used for exchange of spacecraft trajectories between space agencies
- ***spk2oem* is a program for converting a Type 1, 9 or 13 SPICE SPK to an OEM file**
  - It is a command line program using a setup file to specify conversion parameters
  - It performs conversion in “data-driven” or “uniform sampling” mode
- **For more details see the *oem2spk* and *spk2oem* user guides**

\*CCSDS = Consultative Committee on Space Data Systems

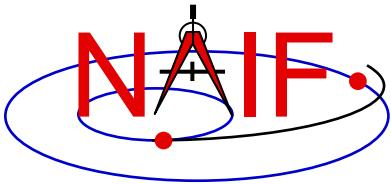


# PREDICKT

---

Navigation and Ancillary Information Facility

- ***predCkt* is a command line program for making CK files from a set of orientation specification rules, using schedules defining when these rules are to be applied**
  - It requires orientation and schedule specification to be provided in a setup file that follows the SPICE text kernel syntax
  - It requires all supporting kernels -- SPK, PCK, etc -- to be loaded using a meta kernel
  - For more details see the “Making a CK Tutorial”



# CKSLICER

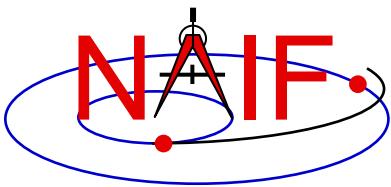
---

Navigation and Ancillary Information Facility

- *ckslicer* is a command line program for subsetting a CK file
- *ckslicer* has the following usage

```
ckslicer -lsk <lsk_file_name>
          -sclk <sclk_file_name(s)>
          -inputck <ck_file_name>
          -outputck <ck_file_name>
          -id <naif_id>
          -timetype <utc|sclk|ticks>
          -start <start_time>
          -stop <stop_time>
```

- *ckslicer* is useful in the situation when only a portion of a CK covering a short interval of time is needed (for example when the whole CK is not needed and it takes up a lot of space) or to cut out parts from a few CKs with the intent to merge them together (if reconstructed CKs from different sources have too much overlap to simply “cat” them together)
- A note stating which subset was extracted is put into the comment area of the output CK file



# CKSLICER Example

## Navigation and Ancillary Information Facility

Terminal Window

```
$ dir mgs_sc_ab1_v2.bc
-rw-rw-r--    1 naifuser 195535872 Jul 17 1999 mgs_sc_ab1_v2.bc

$ ckslicer -lsk naif0007.tls -sclk MGS_SCLKSCET.00054.tsc -inputck
mgs_sc_ab1_v2.bc -outputck mgs_sc_ab1_970915.bc -id -94000 -timetype utc -start
1997-SEP-15 18:00 -stop 1997-SEP-15 21:00

CKSLICER: Version 1.0.1 July 17, 1999; Toolkit Version N0057

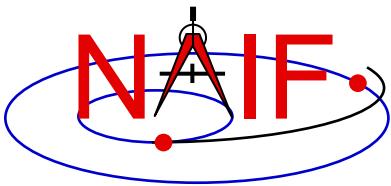
$ dir mgs_sc_ab1_970915.bc
-rw-rw-rw-    1 naifuser 480256 Apr 25 10:23 mgs_sc_ab1_970915.bc

$ ckbrief mgs_sc_ab1_970915.bc naif0007.tls MGS_SCLKSCET.00054.tsc -utc

CKBRIEF Version: 2.0.0, 2001-05-16. SPICE Toolkit Version: N0057.

Summary for: mgs_sc_ab1_970915.bc

Object: -94000
      Interval Begin UTC      Interval End UTC      AV
      -----
      1997-SEP-15 18:00:00.001 1997-SEP-15 21:00:00.000 Y
```



# CKSPANIT

---

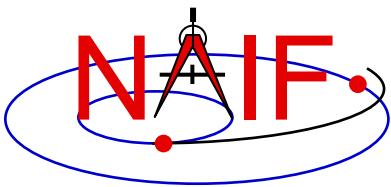
Navigation and Ancillary Information Facility

- ***ckspanit* is a command line program for modifying interpolation interval information in type 3 CK segments**
  - It can also convert a type 1 CK to a type 2 or 3 CK
- ***ckspanit* is used when one is dealing with a type 3 CK containing many small gaps within segments. It allows you to alter the CK in such a way that SPICE will interpolate over those gaps**
- ***ckspanit* has the following usage**

```
ckspanit -in inp_ck -out out_ck -tol threshold [-frm fk]
```

  - “Threshold” is the longest time interval over which interpolation is to be permitted in the output CK file
    - » Must be specified in SCLK ticks
      - For example if 1 tick is 1/256 of a second and interpolation over 30 second intervals is needed, “threshold” must be set to  $256 \times 30 = 7680$
    - “fk” is an optional FK file name, needed only if the base frame in the input CK is not one of the frames built into the Toolkit
- **See also the description of *cksmrg***

**CAUTION:** before running *ckspanit*, make sure that interpolation over larger gaps is appropriate for the vehicle or structure you are dealing with. And don't forget to add appropriate comments to the newly created CK file.



# CKSPANIT Example

## Navigation and Ancillary Information Facility

### Terminal Window

```
$ ckbrief m01_sc_2004-04-22.bc naif0007.tls ORB1_SCLKSCET.00078.tsc -utc -dump
```

CKBRIEF Version: 2.0.0, 2001-05-16. SPICE Toolkit Version: N0057.

Summary for: m01\_sc\_2004-04-22.bc

Segment No.: 1

Object: -53000

Interval Begin UTC	Interval End UTC	AV
2004-APR-22 00:00:05.455	2004-APR-22 18:53:29.054	Y
2004-APR-22 18:55:05.054	2004-APR-22 21:44:22.979	Y
2004-APR-22 21:51:34.974	2004-APR-22 23:59:58.919	Y

```
$ ckspanit -in m01_sc_2004-04-22.bc -out m01_sc_2004-04-22_sp.bc -tol 153600
```

```
$ ckbrief m01_sc_2004-04-22_sp.bc naif0007.tls ORB1_SCLKSCET.00078.tsc -utc -dump
```

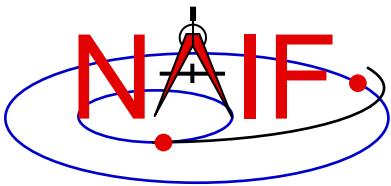
CKBRIEF Version: 2.0.0, 2001-05-16. SPICE Toolkit Version: N0057.

Summary for: m01\_sc\_2004-04-22\_sp.bc

Segment No.: 1

Object: -53000

Interval Begin UTC	Interval End UTC	AV
2004-APR-22 00:00:05.455	2004-APR-22 23:59:58.919	Y



# CKSMRG

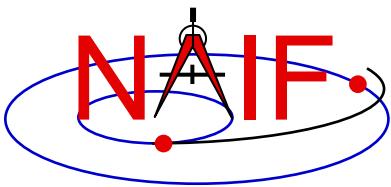
---

Navigation and Ancillary Information Facility

- ***cksmrg* is a command line program that merges data from Type 3 CK segments provided in a single CK file, having the same ID and base frame**
- ***cksmrg* is used for eliminating gaps between segments (that cannot be removed by *ckspanit*) and removing duplicate data points contained in different segments**
- ***cksmrg* has the following usage**

```
cksmrg -k|-kernels <meta kernel name|kernel file names>
        -i|-input <input ck file name>
        -o|-output <output ck file name>
        -s|-segid <output ck segment id string>
        -f|-fileid <output ck file id string>
        -b|-body <body id|name>
        -r|-reference <reference id|name>
        -a|-av <drop|keep|make|makeavrg>
        -t|-tolerance <tolerance (number units)>
        [-c|-correction <time delta|cor. table file>]
```

**CAUTION: *cksmrg* should not be used to merge CK segments from different sources (e.g. predicted and reconstructed), nor should it be used to merge overlapping predict CK segments**



# CKSMRG Example

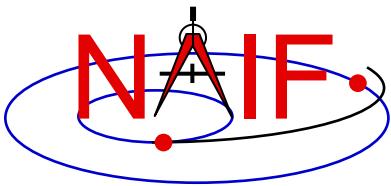
## Navigation and Ancillary Information Facility

```
Terminal Window

$ ckbrief m01.bc naif0007.tls ORB1_SCLKSCET.00078.tsc -utc -rel
. . .
Object: -53000
      Interval Begin UTC      Interval End UTC      AV  Relative to FRAME
-----
2004-APR-20 00:00:03.622 2004-APR-20 23:59:56.288 Y    MARSIAU
2004-APR-21 00:00:02.288 2004-APR-21 23:59:59.455 Y    MARSIAU

$ cksmrg -k naif0007.tls ORB1_SCLKSCET.00078.tsc -i m01.bc -o m01s.bc -s
'CKSMRGed' -f 'CKSMRGed' -b -53000 -r 'MARSIAU' -a keep -t 60 seconds
. . .
(cksmrg displays quite a lot of diagnostics and progress information)
. . .

$ ckbrief m01s.bc naif0007.tls ORB1_SCLKSCET.00078.tsc -utc -rel
. . .
Object: -53000
      Interval Begin UTC      Interval End UTC      AV  Relative to FRAME
-----
2004-APR-20 00:00:03.622 2004-APR-21 23:59:59.455 Y    MARSIAU
```

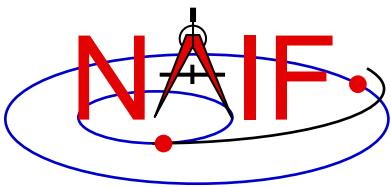


# MAKCLK

---

Navigation and Ancillary Information Facility

- ***makclk* is a program for converting a SCLKSCET file to an SCLK kernel**
  - SCLKSCET (a.k.a. SCLKvSCET) is a time correlation file used by most JPL missions
  - It is an ASCII text file providing piece-wise linear clock correlation function as an array of triplets consisting of the reference on-board time, the reference UTC time, and the clock rate
  - NAIF found that in many cases it is much easier to write an application to first make a SCLKSCET file and then convert it to an SCLK kernel using *makclk* than to write an application to make an SCLK kernel from “scratch”
- ***makclk* is an interactive program prompting for a single input - the name of the setup file**
- **The setup file uses KEYWORD=VALUE assignments to specify input files (SCLKSCET, template SCLK, and LSK), output files (SCLK kernel and log), and control parameters (spacecraft ID, partition tolerance, time filtering flag, and rate adjustment flag)**
- **The *makclk* User’s Guide provides detailed information about the setup file parameters and the SCLKSCET file format and contents.**



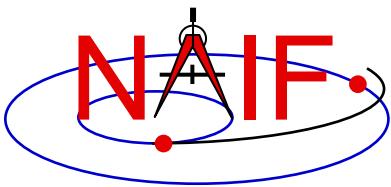
# MAKCLK Example

## Navigation and Ancillary Information Facility

Terminal Window

```
$ more makclk.setup
SCLKSCET_FILE          = flc_sclkscet.00007
OLD_SCLK_KERNEL         = flc_template.tsc
FILE_NAME               = flc_sclkscet.00007.tsc
NAIF_SPACECRAFT_ID     = -77
LEAPSECONDS_FILE       = naif0009.tls
PARTITION_TOLERANCE    = 10
LOG_FILE                = flc_sclkscet.00007.log

$ more flc_sclkscet.00007
(... SCLKSCET SFDU header ...)
CCSD3RE00000$$scet$$NJPL3IS00613$$data$$
* ____SCLK0____ SCETO____ DUT____ SCLKRATE_____
  0.000 2000-001T11:58:55.816 64.184 1.000000000
 189345665.000 2006-001T00:00:00.816 64.184 0.000010000
 189345666.000 2006-001T00:00:00.817 65.184 1.000000000
 268620868.000 2008-188T12:53:23.211 65.184 0.999998631
 276588129.000 2008-280T18:00:53.314 65.184 0.999999788
 281552200.000 2008-338T04:55:23.270 65.184 1.000000029
 284040077.000 2009-001T00:00:00.341 65.184 0.000010000
 284040078.000 2009-001T00:00:00.342 66.184 1.000000029
 287261113.000 2009-038T06:43:55.535 66.184 1.000000131
 291848718.000 2009-091T09:04:01.136 66.184 1.000000166
CCSD3RE00000$$data$$CCSD3RE00000$$sclk$$
```



# MAKCLK Example (continued)

## Navigation and Ancillary Information Facility

Terminal Window

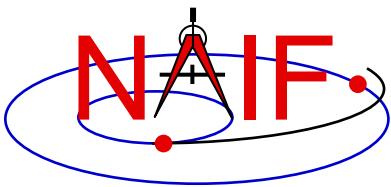
```
$ more flc_template.tsc
KPL/SCLK
\begin{data
    SCLK_KERNEL_ID          = ( @2009-04-07/12:00 )
    SCLK_DATA_TYPE_77        = ( 1 )
    SCLK01_TIME_SYSTEM_77   = ( 2 )
    SCLK01_N_FIELDS_77      = ( 2 )
    SCLK01_MODULI_77         = ( 4294967296 256 )
    SCLK01_OFFSETS_77        = ( 0 0 )
    SCLK01_OUTPUT_DELIM_77  = ( 1 )
    SCLK_PARTITION_START_77 = ( 0.000000000000E+00 )
    SCLK_PARTITION_END_77   = ( 1.0995116277750E+12 )
    SCLK01_COEFFICIENTS_77 = ( 0.E+00 0.E+00 1.E+00 )
\begin{text

$ makclk
.....
Enter the name of the command file

> flc_sclkscet.00007.setup

flc_sclkscet.00007.tsc created.

$
```



# ORBNUM

---

Navigation and Ancillary Information Facility

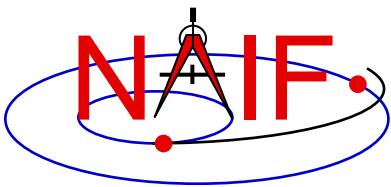
- ***orbnum* is a program for generating a SPICE orbit number file containing orbit numbers and corresponding orbit start/stop times, along with some additional derived quantities (orbital elements and coordinates of sub-spacecraft and sub-solar points)**
  - The orbit number increment can be specified as occurring at one of these events: periapsis or apoapsis, ascending or descending equatorial node crossing, min or max value for the s/c position's Z-coordinate, or min or max value of the s/c's latitude
- ***orbnum* is a command line program with the following usage**

```
orbnum -pref pref_file -num init_orbit -file orbnum_file -d -v -audit -tdb -verbose
```

optional

- “*pref\_file*” is a preferences file using text kernel syntax, specifying setup parameters along with the kernels containing data to be used to search for orbit start and stop events, spacecraft trajectory SPKs, center body PCK, spacecraft SCLK, etc.
- “*init\_orbit*” is the number to be assigned to the first orbit determined using the kernels provided; subsequent orbits are assigned by incrementing “*init\_orbit*” by 1
- “*orbnum\_file*” is the name of the orbit number file to be created

- **An *orbnum* file is not considered a SPICE kernel**
  - It's just a convenient, derived product that NAIF offers to make for orbital missions that wish to have it



# ORBNUM Example

## Navigation and Ancillary Information Facility

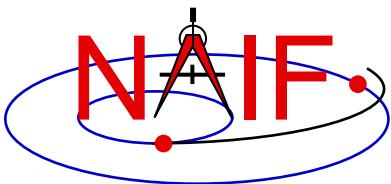
Terminal Window

```
$ more mex_orbnum.setup
\begin{data
TARGET          = -41
OBSERVER        = 499
EVENT_DETECTION_FRAME = 'MARSIAU'
EVENT_DETECTION_KEY    = 'PERI'
ELEMENTS_INERTIAL_FRAME = 'MARSIAU'
ABERRATION_CORRECTION = 'NONE'
ORBIT_PARAMS      = ( 'Sub Sol Lon', 'Sub Sol Lat', ... )
TEXT KERNELS     = ( 'de-245-masses.tpc', 'NAIF0007.TLS', 'mex_030722_step.tsc', ... )
BIN_KERNELS      = ( 'ORMF_PSTPIX_DB_00001.bsp', 'DE405S.BSP' )
SAFETY_MARGIN     = 0.5
STEP_SIZE_TDB    = 'DEFAULT'
\begin{text

$ orbnum -pref mex_orbnum.setup -num 1 -file mex_orbnum.orb
....Loading Kernels

Start UTC (RET for default = 2004 JAN 13 15:54:19.8):<RETURN>
End   UTC (RET for default = 2004 AUG 05 02:10:24.8):<RETURN>

Working, please wait.
Program Finished!
```



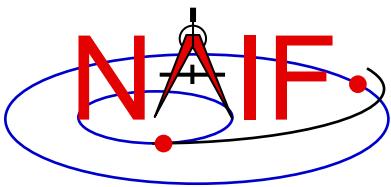
# OPTIKS

---

Navigation and Ancillary Information Facility

- ***optiks* is a command line program that generates information about instrument fields of view (FOV) from parameters present in IK and FK files**
  - FOVs must be defined using the keywords required by the GETFOV routine
- ***optiks* is used in one of two ways**

```
optiks [options]... kernel ...
optiks [options]... meta-kernel ...
```
- ***optiks* uses a set of SPICE kernels specified on the command line; one or more of these kernels may be a meta-kernel**
- **The output data are organized in three tables**
  - The first table lists the angular extents (size) of circular, elliptical, and rectangular FOVs. Using command line options “-units” and “-half” the user can select the unit of measure for the angular measurements, and whether half or full FOV angular extents are listed.
  - The second table contains FOV boresights in a user specified frame at a particular epoch, specified using the “-epoch” option
  - The third table shows FOV boundary vectors and boresights as returned from the GETFOV API, or unitized and rotated into a user-specified frame at a particular epoch



# OPTIKS Example

## Navigation and Ancillary Information Facility

```
Terminal Window

$ optiks -frame CASSINI_SC_COORD cas_iss_v09.ti cas_v37.tf naif0007.tls
cas00084.tsc

. . .

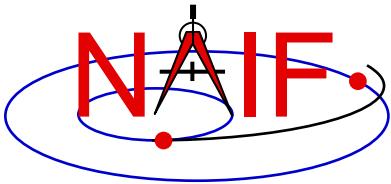
Kernels Loaded:

. . .

FOV full-angular extents computed in RADIANS

Field of View           Shape           Length          Width
-----                 -----           -----          -----
CASSINI_ISS_NAC        RECTANGULAR    +0.006108652382  +0.006108652382
CASSINI_ISS_NAC_RAD    CIRCULAR       +3.141592653590  +3.141592653590
CASSINI_ISS_WAC         RECTANGULAR    +0.060737457969  +0.060737457969
CASSINI_ISS_WAC_RAD    CIRCULAR       +3.141592653590  +3.141592653590

FOV boresights computed at epoch 2001-JAN-01 12:00
FOV boresights computed in frame CASSINI_SC_COORD
Field of View           Boresight Vector
-----                   -----
CASSINI_ISS_NAC        ( +0.000575958621, -0.999999819520, -0.000170972424 )
CASSINI_ISS_NAC_RAD    ( +1.000000000000, -0.000000000000, +0.000000000000 )
CASSINI_ISS_WAC         ( +0.001218344236, -0.999999225446, +0.000254451360 )
CASSINI_ISS_WAC_RAD    ( +1.000000000000, -0.000000000000, +0.000000000000 )
```

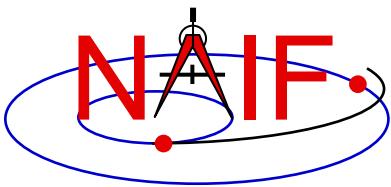


# ARCHTYPE

---

Navigation and Ancillary Information Facility

- ***archtype* is a program that displays the file architecture and type of a SPICE kernel; it is useful for scripting applications**
  - To identify the architecture and type *archtype* uses the same mechanism as the FURNSH routine
- ***archtype* has a simple command line interface and requires only one argument -- the name of a kernel file:**  
`archtype kernel_name`
- ***archtype* prints architecture and type to standard output as two space delimited acronyms**
  - Architecture can be:
    - » ‘DAF’ or ‘DAS’ for binary kernels
    - » ‘KPL’ for text kernels
  - Type can be ‘SPK’, ‘PCK’, ‘IK’, ‘CK’, ‘EK’, ‘LSK’, ‘SCLK’, ‘FK’, ‘MK’, ‘DSK’
- **If architecture and/or type cannot be determined, the program displays ‘UNK’**
- **In order for text kernels to be recognized, the first few characters of the file must contain ‘KPL/<type>’ (i.e. ‘KPL/IK’, ‘KPL/FK’, etc.)**



# ARCHTYPE Examples

Navigation and Ancillary Information Facility

Terminal Window

```
$ archetype 020514_SE_SAT105.bsp
DAF SPK

$ archetype 04135_04171pc_psiv2.bc
DAF CK

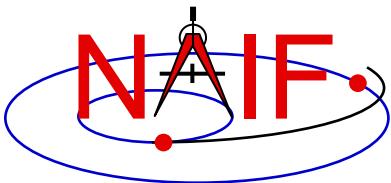
$ archetype cas00084.tsc
KPL SCLK

$ archetype cas_v37.tf
KPL FK

$ archetype cpck05Mar2004.tpc
KPL PCK

$ archetype naif0008.tls
KPL LSK

$ archetype .cshrc
UNK UNK
```



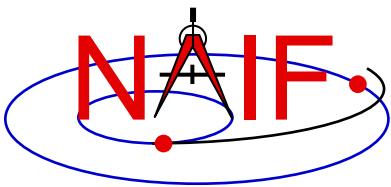
# BFF

---

Navigation and Ancillary Information Facility

- ***bff* is a program that displays the binary file format of one or a few SPICE kernels**
- ***bff* has a simple command line interface requiring kernel names to be listed on the command line:**  

```
bff kernel_name [kernel_name ...]
```
- ***bff* prints the binary file format string ('BIG-IEEE' or 'LTL-IEEE') to standard output**
  - When run on a single kernel, it prints only the format string
  - When run on more than one kernel, it prints the format string followed by the file name on a separate line for each file
- **If an input file is not a binary kernel, the program displays the format string 'N/A'**
- **If the binary file format cannot be determined (for DAS files produced by applications linked to SPICE Toolkit N0051, April 2000 or earlier), the program displays the format string 'UNK'**



# BFF Examples

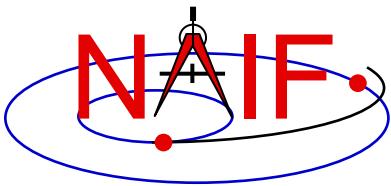
## Navigation and Ancillary Information Facility

### Terminal Window

```
$ bff mer2_surf_rover.bsp
BIG-IEEE

$ bff ./*.bc ./*.bsp ./*.tf ./*.xsp
BIG-IEEE ./MRO_PHX_EDL_07260_PASS1_sc_20070917181502.bc
LTL-IEEE ./070416BP_IRRE_00256_14363.bsp
LTL-IEEE ./mars_north.bsp
BIG-IEEE ./mer2_surf_rover.bsp
LTL-IEEE ./sb406-20pb.bsp
LTL-IEEE ./zero_offset.bsp
N/A      ./vo.tf
N/A      ./mgn06127.xsp

$
```



# BINGO

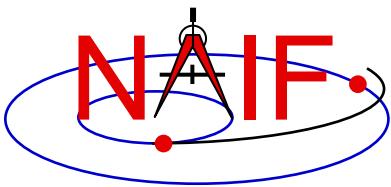
---

Navigation and Ancillary Information Facility

- ***bingo* is a program that converts:**
  - binary SPICE kernels between IEEE big endian and little endian formats
  - text format SPICE kernels between DOS and UNIX text formats
- ***bingo* has a simple command line interface:**

```
bingo [option] input_kernel output_kernel
```

  - “option” is a flag specifying the conversion direction: ‘-ieee2pc’ or ‘-pc2ieee’ for binary kernels and ‘-unix2dos’ or ‘-dos2unix’ for text format kernels
  - “input\_kernel” is the input kernel file name
  - “output\_kernel” is the output kernel file name. If the output file already exists, the program overwrites it.
- **The conversion direction flag does not need to be specified for DAF-based binary file conversions (SPK, CK, binary PCK) and post-N0051 DAS-based binary file conversions (EK, DBK, DSK)**
  - The program automatically determines the input file format and performs conversion to the other format
- **The conversion direction flag must be specified for pre-N0051 DAS-based binary file conversions, and for text file conversions**



# BINGO Examples

## Navigation and Ancillary Information Facility

### Terminal Window

DAF-based binary kernel conversions:

```
$ bingo de405s_ieee.bsp de405s_pc.bsp
```

```
$ bingo de405s_pc.bsp de405s_ieee.bsp
```

Modern DAS-based binary kernel conversions:

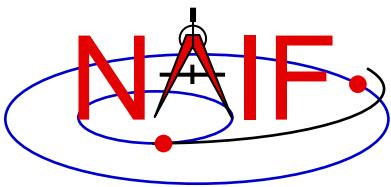
```
$ bingo 10A_ieee.bdb 10A_pc.bdb
```

```
$ bingo 10A_pc.bdb 10A_ieee.bdb
```

Text kernel conversions:

```
$ bingo -unix2dos naif0008_unix.tls naif0008_dos.tls
```

```
$ bingo -dos2unix naif0008_dos.tls naif0008_unix.tls
```

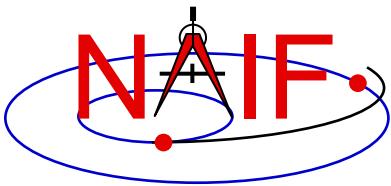


---

Navigation and Ancillary Information Facility

# Exception Handling

January 2020

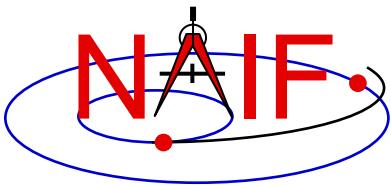


# Topics

---

Navigation and Ancillary Information Facility

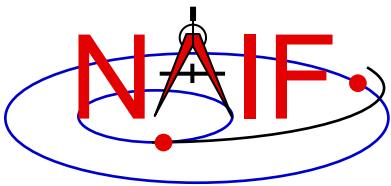
- **What Exceptions Are**
- **Language Dependencies**
- **C and Fortran Error Handling Features**
- **Error Messages**
- **Error Handling Actions**
- **Error Device**
- **Customize Error Handling**
- **Get Error Status**
- **Signal Errors**
- **Icy Error Handling**
- **Mice Error Handling**
- **Recommendations**



# Exceptions Are... - 1

Navigation and Ancillary Information Facility

- Run-time error conditions such as:
  - Files
    - » Required files not loaded
    - » Gaps in data
    - » Corrupted or malformed files (e.g. ftp'd in wrong mode)
  - Invalid subroutine/function arguments
    - » String values unrecognized
    - » Numeric values out of range
    - » Data type/dimension mismatch
  - Arithmetic errors
    - » Divide by zero, taking the square root of a negative number
  - Environment problems
    - » Insufficient disk space for output files
    - » Lack of required read/write permission/privileges

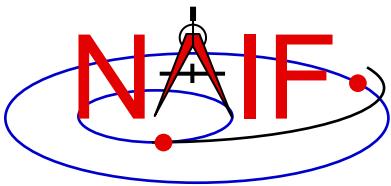


# Exceptions Are... - 2

---

Navigation and Ancillary Information Facility

- **Valid but unusual conditions, such as:**
  - » Normalize the zero vector
  - » Find the rotation axis of the identity matrix
  - » Find the boresight intercept lat/lon for a non-intercept case
  - » Find a substring where the end index precedes the start index
  - Such cases are normally not SPICE “Error Conditions”
  - Typically must be handled by a logical branch
- **Errors found by analysis tools, such as:**
  - » Invalid SQL query
  - » Invalid string representing a number
  - Such cases are normally not SPICE “Error Conditions”
  - However, if a SPICE parsing routine failed because it couldn’t open a scratch file, that would be an “error condition”

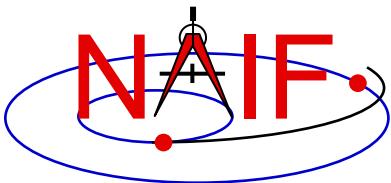


# SPICE “Errors”

---

Navigation and Ancillary Information Facility

- Most “errors” made while using SPICE result from a mistake in how you are trying to use SPICE code, or in how you are trying to use SPICE files
  - It’s rare that a SPICE user finds an error within SPICE Toolkit code
- The SPICE “exception handling subsystem” helps detect user’s errors
- All “errors” detected by SPICE result in a SPICE error message
  - Such errors will not make your program crash
- A program crash indicates an error in your own code, a corrupted SPICE kernel, or (rarely) a SPICE bug

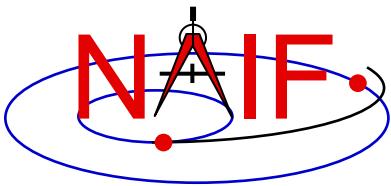


# Language Dependencies

---

Navigation and Ancillary Information Facility

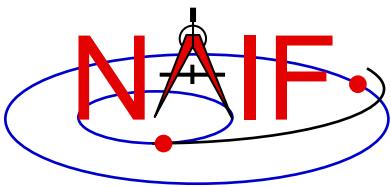
- **SPICELIB and CSPICE provide essentially identical error handling capabilities.**
- **Icy and Mice provide similar error handling functionality; this functionality is quite different from that of CSPICE.**
  - These systems do rely on CSPICE for most error detection.
  - Icy and Mice provide no API for customizing underlying CSPICE error handling behavior.
  - Short, long, and traceback error messages are merged into a single, parsable, message.
  - Use IDL or MATLAB features to customize error handling...
    - » to prevent your program from stopping.
    - » to capture SPICE error messages.
- **Most of this tutorial deals with SPICELIB and CSPICE error handling.**
  - There is a bit on Icy and Mice near the end.



# Fortran and C Error Handling Features - 1

Navigation and Ancillary Information Facility

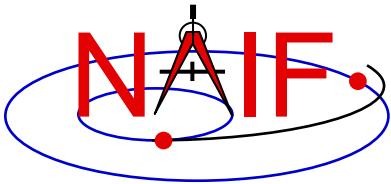
- **Error handling in SPICE: safety first**
  - Trap errors where they occur; don't let them propagate.
    - » Don't let errors “fall through” to the operating system.
  - Supply meaningful diagnostic messages.
    - » Incorporate relevant run-time data.
    - » Supply context in human-readable form.
  - Don't depend on callers to handle errors.
    - » Normally, “error flags” are not returned to callers.
  - Stop unless told not to.
    - » Don't try to continue by making “smart guesses.”
- **Subroutine interface for error handling**
  - Interface routines called within SPICE may be called by users' application programs



# Fortran and C Error Handling Features - 2

Navigation and Ancillary Information Facility

- **Signal errors**
  - Create descriptive messages when and where an error is detected
    - » Short message, long message, (explanation), traceback
  - “Signal” the error: set error status, output messages
    - » By default, CSPICE error output goes to stdout (not stderr)
- **Retrieve error information**
  - Get status and error messages via subroutine calls
- **Customize error response---actions taken when an error occurs.**
  - Set error handling mode (“action”)
  - Set error output device
  - Set message selection
- **Inhibit tracing**
  - To improve run-time performance (only for thoroughly debugged code)

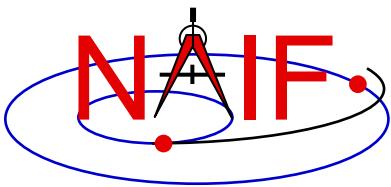


# Error Messages

---

Navigation and Ancillary Information Facility

- **Short message**
  - Up to 25 characters.
  - Can easily be compared with expected value.
    - » Example: SPICE(FILEOPENFAILED).
- **Long message**
  - Up to 1840 characters.
  - Can contain values supplied at run time.
    - » Example: 'The file <sat077.bsp> was not found.'
- **Traceback**
  - Shows call tree above routine where error was signaled.
    - » Not dependent on system tracing capability.
    - » Don't need a “crash” to obtain a traceback.



# Error Handling Actions - 1

---

Navigation and Ancillary Information Facility

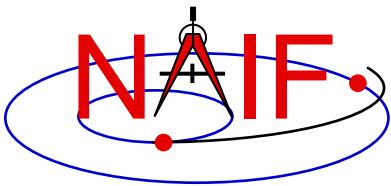
- **ABORT**

- Designed for safety.
  - » Output messages and traceback to your screen or stdout.
  - » Stop program; return status code if possible.

- **RETURN**

- For use in programs that must keep running.
- Attempts to return control to the calling application.
- Preserves error information so calling application can respond.
  - » Output messages to current error device.
  - » Set error status to “true”: FAILED() will return “true.”
  - » Set “return” status to “true”: RETURN() will return “true.”
  - » Most SPICE routines will return on entry. Very simple routines will generally execute anyway.

[continued on next page](#)

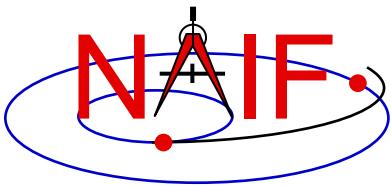


# Error Handling Actions - 2

---

Navigation and Ancillary Information Facility

- » Capture traceback at point where error was signaled.
- » Inhibit error message writing and error signaling.
- » Must call RESET to resume normal error handling.

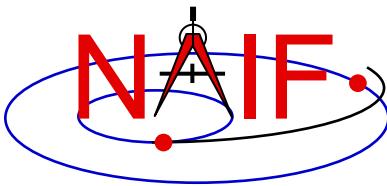


# Error Device

---

Navigation and Ancillary Information Facility

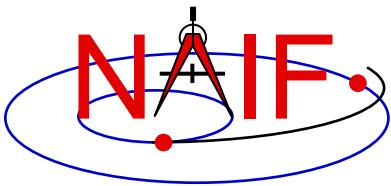
- **Destination of error messages**
  - Screen/stdout (default)
  - Designated file
    - » Error diagnostics are appended to the file as errors are encountered.
  - “NULL” --- suppress output
    - » When the NULL device is specified, error messages can still be retrieved using API calls.
- **Limitations**
  - In C, cannot send messages to stderr.
  - In C, writing to a file opened by means other than calling `errdev_c` is possible only if CSPICE routines were used to open the file.
    - » These limitations may be removed in a later version of CSPICE.



# Customize Error Handling - 1

Navigation and Ancillary Information Facility

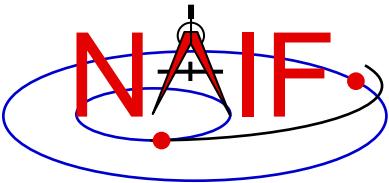
- **Set error action**
  - CALL ERRACT ( ‘SET’, ‘RETURN’ )
  - erract\_c ( “set”, LEN, “return” );
    - » Length argument is ignored when action is “set”; when action is “get”, LEN should be set to the available room in the output string, for example:
    - » erract\_c ( “get”, ACTLEN, action );
- **Set error device**
  - CALL ERRDEV ( ‘SET’, ‘errlog.txt’ )
  - errdev\_c ( “set”, LEN, “errlog.txt” );
- **Select error messages**
  - CALL ERRPRT ( ‘SET’, ‘NONE, SHORT, TRACEBACK’ )
    - » If tracing is disabled (see next page), selecting TRACEBACK has no effect.
  - errprt\_c ( “set”, LEN, “none, short, traceback” );



# Customize Error Handling - 2

Navigation and Ancillary Information Facility

- **Disable tracing**
  - Normally done to speed up execution by a few percent
  - Benefit is highly dependent on application
  - NAIF normally recommends users not turn tracing off
  - Use TRCOFF:
    - » **CALL TRCOFF or trcoff\_c();**
      - Do this at the beginning of your program.
      - Once disabled you cannot re-enable tracing during a program run.

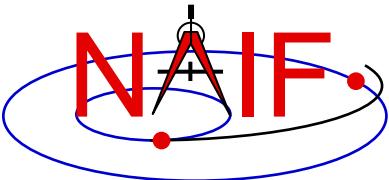


# Get Error Status - 1

---

Navigation and Ancillary Information Facility

- **Use FAILED to determine whether an error has been signaled**
  - `IF ( FAILED() ) THEN ...`
  - `if ( failed_c() ) { ...`
- **Use FAILED after calling one or more SPICE routines in a sequence**
  - Normally, it's safe to call a series of SPICE routines without testing FAILED after each call
- **Use GETMSG to retrieve short or long error messages**
  - `CALL GETMSG ( 'SHORT' , SMSG )`
  - `getmsg_c ( "short", LEN, smsg );`

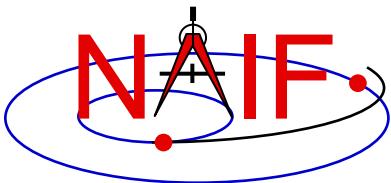


# Get Error Status - 2

---

Navigation and Ancillary Information Facility

- **Use QCKTRC or TRCDEP and TRCNAM to retrieve traceback message**
- **Test value of RETURN() to determine whether routines should return on entry**
  - Only relevant if user code is designed to support RETURN mode
- **Handle error condition, then reset error status:**
  - CALL RESET
  - `reset_c();`
  - In Icy-based applications you only need handle the error condition; a reset is automatically performed by Icy



# Signal Errors - 1

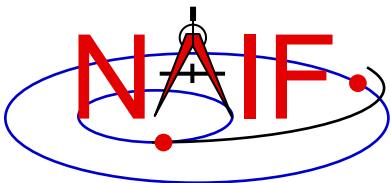
---

Navigation and Ancillary Information Facility

- **Create long error message**
  - Up to 1840 characters
  - Use SETMSG

```
» CALL SETMSG ( 'File <#> was not found.' )
» setmsg_c      ( "File <#> was not found." );
```
- **Substitute string, integer, or d.p. values at run time**
  - Use ERRCH

```
» CALL ERRCH ( '#', 'cassini.bsp' )
» errch_c      ( "#", "cassini.bsp" );
```
  - Also can use ERRINT, ERRDP
  - In Fortran, can refer to files by logical unit numbers: ERRFNM

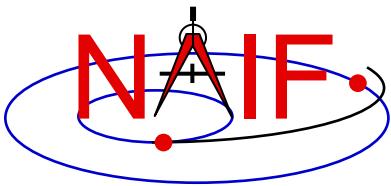


# Signal Errors - 2

---

Navigation and Ancillary Information Facility

- **Signal error**
  - Use SIGERR to signal error. Supply short error message as input to SIGERR.
    - » CALL SIGERR ( 'FILE OPEN FAILED' )
    - » `sigerr_c` ( "FILE OPEN FAILED" );
  - “Signaling” error causes SPICE error response to occur
    - » Output messages, if enabled
    - » Set error status
    - » Set return status, if error action is RETURN
    - » Inhibit further error signaling if in RETURN mode
    - » Stop program if in abort mode
- **Reset error status after handling error**
  - CALL RESET()
  - `reset_c()`



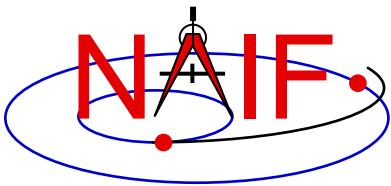
# Icy Error Handling

---

Navigation and Ancillary Information Facility

- **Error action:**
  - By default, a SPICE error signal stops execution of IDL scripts; a SPICE error message is displayed; control returns to the execution level (normally the command prompt).
  - Icy sets the CSPICE shared object library's error handling system to RETURN mode. No other modes are used.
    - » The CSPICE error state is reset after detecting an error.
  - Use the IDL CATCH feature to respond to error condition.
- **Error status**
  - Value of !error\_state.name
    - » ICY\_M\_BAD\_IDL\_ARGS - indicates invalid argument list.
    - » ICY\_M\_SPICE\_ERROR - indicates occurrence of a SPICE error.
- **Error message**
  - CSPICE short, long, and traceback error messages are merged into a single, parseable, message.
    - » The merged error message is contained in the variable !error\_state.msg.
    - » Example:

```
CSPICE_ET2UTC: SPICE(MISSINGTIMEINFO) : [et2utc->ET2UTC->UNITIM]
The following, needed to convert between the
uniform time scales, could not be found in the
kernel pool: DELTET/DELTA_T_A, DELTET/K,
DELTET/EB, DELTET/M. Your program may have failed to load...
```



# Mice Error Handling

Navigation and Ancillary Information Facility

- **Error action**

- By default, a SPICE error signal stops execution of MATLAB scripts; a SPICE error message is displayed; control returns to the execution level.
- Mice sets the CSPICE shared object library's error handling system to RETURN mode. No other modes are used.
  - » The CSPICE error state is reset after detecting an error.
- Use the MATLAB try/catch construct to respond to error condition.

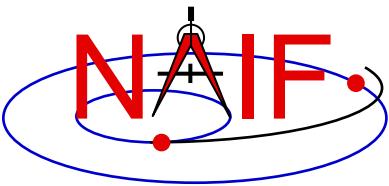
- **Error message**

- CSPICE short, long, and traceback error messages are merged into a single, parsable, message.
  - » Example:

```
??? SPICE(MISSINGTIMEINFO) : [et2utc->ET2UTC->UNITIM]
The following, needed to convert between the
uniform time scales, could not be found in the
kernel pool: DELTET/DELTA_T_A, DELTET/K,
DELTET/EB, DELTET/M. Your program may have failed to load...
```

- **Use the MATLAB function lasterror to retrieve SPICE error diagnostics. When a SPICE error occurs:**

- the “message” field of the structure returned by lasterror contains the SPICE error message.
- the “stack” field of this structure refers to the location in the m-file from which the Mice wrapper was called (and so is generally not useful).
- the “identifier” field of this structure currently is not set.

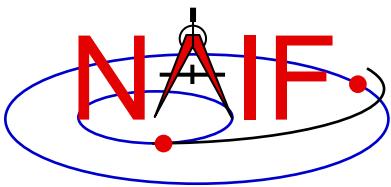


# Recommendations

---

Navigation and Ancillary Information Facility

- **For easier problem solving**
  - Leave tracing enabled when debugging.
  - Always test FAILED after a sequence of one or more consecutive calls to SPICE routines.
  - Don't throw away error output. It may be the only useful clue as to what's going wrong.
    - » Programs that must suppress SPICE error output should trap it and provide a means for retrieving it.
      - Test FAILED to see whether an error occurred.
      - Use GETMSG to retrieve error messages
      - Use RESET to clear the error condition
    - Use SPICE error handling in your own code where appropriate.
    - When reporting errors to NAIF, have SPICE error message output available
      - » Note whether error output is actually from SPICE routines, from non-SPICE code, or was generated at the system level.

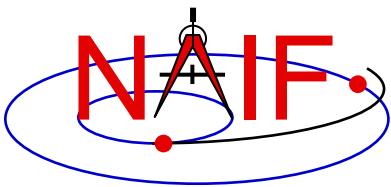


---

Navigation and Ancillary Information Facility

# SPICE Toolkit Common Problems

January 2020

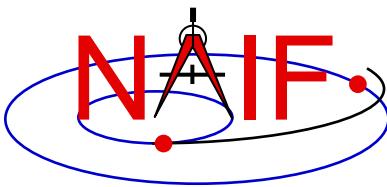


# Topics

---

Navigation and Ancillary Information Facility

- Prevention
- Useful Documentation
- Reporting a Problem to NAIF

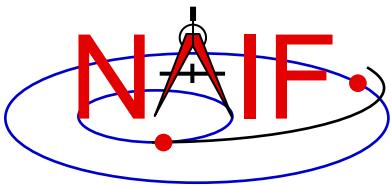


# Prevention - 1

---

Navigation and Ancillary Information Facility

- **Use a Toolkit obtained directly from NAIF and intended for your specific environment (platform/OS/compiler/compiler options)**
  - Be extra careful about **32-bit** versus **64-bit** hardware
- **Use a current Toolkit**
  - Newer Toolkits may have bug fixes and new features you need
    - » Toolkits are always backwards compatible, so you should have no problem re-linking your “old” application to the latest Toolkit
- **Read the pertinent documentation**
  - Tutorials, module headers, Required Reading technical reference documents, comments inside kernels
- **Use the correct kernels**
  - Often, but not always, this means the latest version
  - Verify that contents, time coverage (if applicable) and intended use are suitable for your work
- **If you are using a Fortran Toolkit, be sure your text kernels all use the line termination appropriate for your platform.**
  - Unix/Linux/OSX use <LF>; PC/Windows uses <CR><LF>
  - Using the BINGO utility from the NAIF website to make the change is one solution
  - Be sure the last line in your text kernel ends with an end of line termination



# Prevention - 2

---

Navigation and Ancillary Information Facility

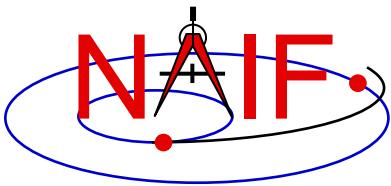
- **Avoid common implementation problems**
  - Verify use of the correct time system for your need
    - » e.g., TDB (also called ET), UTC, or SCLK?
  - When using SCLK time tags, be sure to form your SCLK string to match the specification within the SCLK kernel
    - » Make sure the fractional part is in the form that is expected
  - Verify that correct reference frames are used
    - » e.g., MOON\_PA versus MOON\_ME versus IAU\_MOON?
    - » e.g. IAU\_Mars versus MARSIAU? (these are VERY different frames)
  - Check definitions of geometric quantities
    - » e.g. Planetocentric vs. planetographic vs planetodetic coordinates
    - » Oblate, spherical or DSK body shape
  - Check aberration corrections
    - » Converged Newtonian light time + stellar aberration, light time + stellar aberration, light time only, or none?
    - » Target orientation corrected for light time?
  - Don't confuse an instrument reference frame ID with the ID of the instrument itself (the object ID)



# Useful Documentation

Navigation and Ancillary Information Facility

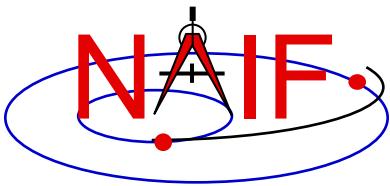
- NAIF has compiled a list of common problems, probable causes, and solutions:
  - Refer to .../doc/html/req/problems.html or ...doc/req/PROBLEMS.REQ, both of which are provided in each Toolkit package. Or, access the HTML document corresponding to the supported language at:
    - » [https://naif.jpl.nasa.gov/pub/naif/toolkit\\_docs/FORTRAN/req/problems.html](https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/FORTRAN/req/problems.html)
    - » [https://naif.jpl.nasa.gov/pub/naif/toolkit\\_docs/C/req/problems.html](https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/problems.html)
    - » [https://naif.jpl.nasa.gov/pub/naif/toolkit\\_docs/IDL/req/problems.html](https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/IDL/req/problems.html)
    - » [https://naif.jpl.nasa.gov/pub/naif/toolkit\\_docs/MATLAB/req/problems.html](https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/MATLAB/req/problems.html)
- Some on-line tutorials (e.g. SPK and CK) include a common problems section near the end of the tutorial
- It may be useful to read these documents BEFORE embarking on extensive SPICE-based programming projects, since some problems are best solved early in the software development cycle



# Reporting a Problem to NAIF

Navigation and Ancillary Information Facility

- If you need help troubleshooting a programming or usage problem, you can send email to NAIF. Try to include these items in your email message:
  - The SPICE or operating system diagnostic messages written to the screen or to log files
  - The name and version of the operating system you're using
  - The name and version of the compiler or programming environment (gcc, gfortran, ifort, clang, IDL, Matlab, etc.)
  - The Toolkit version you're using, e.g. N0066 (also called N66)
  - Names of the kernel files being used
    - » Include any meta-kernel you're using
    - » You may need to provide the kernels themselves if these are not available to NAIF
  - Your inputs to the SPICE modules that signaled the error
  - If possible, a code fragment from where the error seems to occur
- Send the email to only one person on the NAIF team
  - It will get routed to the best person to provide an answer
  - Contact information: <https://naif.jpl.nasa.gov/naif/contactinfo.html>



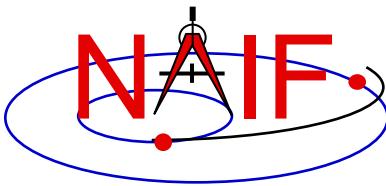
---

Navigation and Ancillary Information Facility

# Obtaining SPICE Products Available from the NAIF and Horizons Servers

## Emphasis on Kernels

January 2020

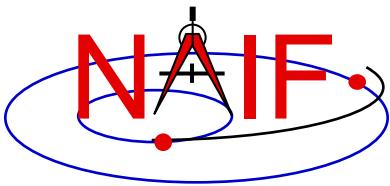


# Overview

---

Navigation and Ancillary Information Facility

- **Many SPICE products are available from the NAIF server**
  - These are mostly SPICE products produced at JPL
  - Access is available using the https protocol
  - See the next page for URLs
- **SPICE products made by other organizations are controlled by those organizations**
  - Some may be available from the NAIF server
  - Some may be available at other public servers, or on restricted servers, or not at all
    - » The International Planetary Data Alliance (IPDA) is working towards making large amounts of archived planetary data, including SPICE, universally available through “all” agency archives
  - Unfortunately there is no simple rule set to describe what may be found where
  - As a general rule, NAIF has no cognizance of these products



# NAIF Server URLs

## Navigation and Ancillary Information Facility

- NAIF home page

<https://naif.jpl.nasa.gov>

- Here you may access all official SPICE products produced by NAIF
  - kernels (generic, mission operations, and PDS archived ancillary data)
  - software (Toolkits and individual utility programs)
  - documents
  - tutorials
  - programming lessons
  - problem solving tips
  - rules about using SPICE
  - links to useful resources
  - access to the WebGeocalc tool
  - access to the Cosmographia visualization program

- SPICE announcements (by NAIF)

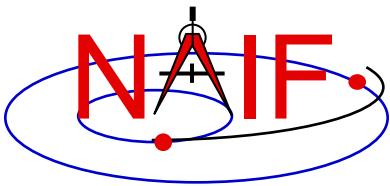
[https://naif.jpl.nasa.gov/mailman/listinfo/spice\\_announce](https://naif.jpl.nasa.gov/mailman/listinfo/spice_announce)

For use by NAIF staff in making assorted announcements.

- SPICE discussion (by anyone)

[https://naif.jpl.nasa.gov/mailman/listinfo/spice\\_discussion](https://naif.jpl.nasa.gov/mailman/listinfo/spice_discussion)

For use by SPICE users who wish to communicate with other SPICE users  
(Don't use this if you have questions for NAIF staff)

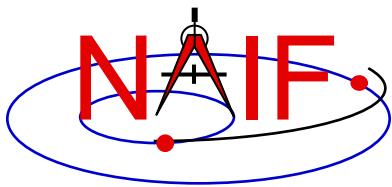


# Getting SPICE Kernels

---

Navigation and Ancillary Information Facility

- The remaining charts discuss where to find the various categories of SPICE kernel files
  - Operational flight project kernels
    - » For active flight projects, mostly those at JPL
  - PDS archived kernels
    - » Those that have been delivered to and reviewed and accepted by the NAIF Node of NASA's Planetary Data System
    - » These are the most easily used, due to the existence of furnsh kernels (meta-kernels)!
    - » These cover from launch to typically 6-to-9 months behind current time
  - Generic kernels
    - » Not tied to any one specific mission
    - » Used by many flight projects and other endeavors
    - » Some of these are also available in the other two categories
  - How to generate SPKs for comets and asteroids



# Obtaining Operational Flight Project Kernels - 1

## Navigation and Ancillary Information Facility

Data\_Operational  
naif.jpl.nasa.gov/naif/data\_operational.html

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION + View the NASA Portal

# NAIF

## The Navigation and Ancillary Information Facility

**Operational Flight Project Kernels and Assorted Other Project Kernels**

Kernels for currently active missions where NAIF produces the kernels or otherwise has access to them, kernels from some pre-SPICE era missions, and miscellaneous other mission kernels may be obtained from the NAIF server using the links below. Included under this category are some kernels from past missions that have not yet been archived in the PDS.

- o Heliophysics Missions
- o Mercury Missions
- o Venus Missions
- o Earth Missions
- o Lunar Missions
- o Mars Missions
- o Outer Planet Missions
- o Comet and Asteroid Missions
- o Astrophysics Missions

Please note that kernels produced by agencies other than JPL are usually available only at those agencies, and may not be available to other than the flight project's team members. (By agreement between ESA and NASA, kernels for a few ESA-sponsored missions are mirrored at NAIF for the convenience of U.S. participating scientists.)

PDS Nodes: Atmospheres Geosciences Imaging NAIF PPI Rings Small Bodies

**FIRST GOV** Your First Click to the U.S. Government + NASA Privacy Statement, Disclaimer

Open "http://naif.jpl.nasa.gov/naif/pds.html" in a new tab

Clearance: CL#05-2438 Site Manager: Charles Acton NASA Official: William Knopf Webmaster: Ron Baalke Last Updated: 26 Feb 2014

1 - Select the mission class of interest

2a - Select the project name to get access to the kernels folder for that project.  
(see next page)

- or -

NAIF Data  
naif.jpl.nasa.gov/naif/data\_outer.html

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION + View the NASA Portal

# NAIF

## The Navigation and Ancillary Information Facility

**Data**

The number of files for each SPICE kernel type is shown in the table below for the missions specified. An asterisk (\*) indicates that one or more non-kernel files are also present; usually this is an 'areadme' file that explains the kernel file naming convention. The count of the number of kernels is made ONLY in the primary directory; in some cases there are additional kernels in a subdirectory (for instance, older versions of kernels that have been replaced with newer versions).

**Outer Planet Missions**

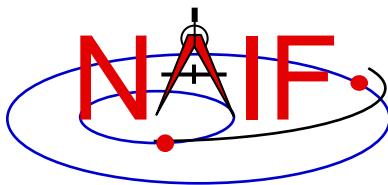
Mission	ck	ek	fk	ik	lsk	pck	sclk	spk
CASSINI*	5058*	473*	23*	13*	4*	352*	114*	4146*
GALILEO*	26	0		7	2	3	1	12*
JUNO*	222*			3*	11*	2*	3*	38*
NEW HORIZONS*								
PIONEER 10*								1
PIONEER 11								2*
VOYAGER 1, 2*	10		2	4	0*	0*	2	9*

PDS Nodes: Atmospheres Geosciences Imaging NAIF PPI Rings Small Bodies

**FIRST GOV** Your First Click to the U.S. Government + NASA Privacy Statement, Disclaimer

Clearance: CL#05-2438 Site Manager: Charles Acton NASA Official: William Knopf Webmaster: Ron Baalke Last Updated: 10 Oct 2014

2b - Select the kernel type to get access to all kernels of that type for that project. The number tells how many kernels of that type are available.  
(see next page)



# Obtaining Operational Flight Project Kernels - 2

## Navigation and Ancillary Information Facility

Access to all kernels for the named project

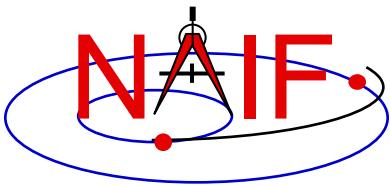
Access to kernels of the selected type for the named project

Name	Last modified	Size
<a href="#">Parent Directory</a>		-
<a href="#">aareadme.txt</a>	11-Mar-2004 14:39	400
<a href="#">ck/</a>	06-Aug-2019 11:26	-
<a href="#">ek/</a>	18-Oct-2017 19:04	-
<a href="#">fk/</a>	25-Jul-2019 12:01	-
<a href="#">ik/</a>	25-Jul-2019 12:06	-
<a href="#">lsk/</a>	03-Aug-2016 03:12	-
<a href="#">pck/</a>	22-Sep-2018 06:41	-
<a href="#">sclk/</a>	16-May-2018 09:04	-
<a href="#">spk/</a>	27-Sep-2018 16:55	-

Then change to the folder containing the kind of kernels of interest to you, such as SPK.

Name	Last modified	Size
<a href="#">Parent Directory</a>		-
<a href="#">000202R SK LP0 V1P32.bsp</a>	21-Feb-2002 13:02	238K
<a href="#">000202R SK LP0 V1P32.bsp.lbl</a>	21-Feb-2002 13:02	2.3K
<a href="#">000202R SK V1P32 V2P12.bsp</a>		502K
<a href="#">000202R SK V1P32 V2P12.bsp.lbl</a>	3:02	2.6K
<a href="#">000202R SK V2P12 EP15.bsp</a>	21-Feb-2002 13:02	198K
<a href="#">000202R SK V2P12 EP15.bsp.lbl</a>	21-Feb-2002 13:02	2.5K
<a href="#">000203 SE JUP156.bsp</a>	21-Feb-2002 13:03	4.9M
<a href="#">000203 SE JUP156.bsp.lbl</a>	21-Feb-2002 13:03	3.2K
<a href="#">000331RB SK V1P32 V2P12.bsp</a>	16-Jun-2005 13:02	623K
<a href="#">000331RB SK V1P32 V2P12.bsp.lbl</a>	16-Jun-2005 13:02	5.2K
<a href="#">000331R SK LP0 V1P32.bsp</a>	21-Feb-2002 13:03	307K
...		
<a href="#">aareadme.txt</a>		
...		

Description of file naming scheme

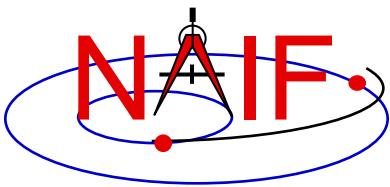


# Obtaining PDS Archived Kernels

---

Navigation and Ancillary Information Facility

- **The best method for obtaining PDS archived kernels is directly from the NAIF server.**
  - Complete SPICE data sets exist on the NAIF server fully expanded—not bundled in a Zip or tar file
  - Unless you have reason to do otherwise, download the entire archival data set using the "Archive Link"
    - » That way you'll get all the latest data, the associated "furnsh kernels", and the best documentation.
  - If the data set is large and you need only a portion of it based on start/stop time, use the "subsetter" link to obtain the smaller amount of data needed.
- **Pictorial examples are shown on the next two pages.**



# Obtaining Archived Kernels from the NAIF Server - 1

## Navigation and Ancillary Information Facility

Mission Name	Archive Readme	Archive Link	PDS3 or PDS4	Data Size (GB)	Start Time	Stop Time	Subset Link
Cassini Orbiter	<a href="#">readme</a>	<a href="#">link</a>	3	17.1	1997-10-15	2014-09-30	<a href="#">subset</a>
Clementine	<a href="#">readme</a>	<a href="#">link</a>	3	0.8	1994-01-26	1994-05-07	<a href="#">subset</a>
DAWN	<a href="#">readme</a>	<a href="#">link</a>	3	13.5	2007-09-27	2012-09-13	<a href="#">subset</a>
Deep Impact	<a href="#">readme</a>	<a href="#">link</a>	3	0.7	2005-01-12	2005-08-09	<a href="#">subset</a>
Deep Space 1	<a href="#">readme</a>	<a href="#">link</a>	3	0.9	1998-10-24	2001-12-18	<a href="#">subset</a>
EPOXI	<a href="#">readme</a>	<a href="#">link</a>	3	1.0	2005-08-23	2011-03-01	<a href="#">subset</a>
GRAIL	<a href="#">readme</a>	<a href="#">link</a>	3	4.3	2011-09-10	2012-12-17	<a href="#">subset</a>
Hayabusa	<a href="#">readme</a>	<a href="#">link</a>	3	0.3	2005-09-11	2005-11-19	<a href="#">subset</a>

[http://naif.jpl.nasa.gov/pub/naif/pds/data/co-s\\_j\\_e\\_v-spice-6-v1.0/cosp\\_1000](http://naif.jpl.nasa.gov/pub/naif/pds/data/co-s_j_e_v-spice-6-v1.0/cosp_1000)

If you select “PDS SPICE Archives” on the NAIF web page you can do any of the following.

- You can copy and paste the "link" URL into the Unix "wget" or the FileZilla tool, or some equivalent tool, to download the entire data set—recommended if not too large! See the next page if data set size is an issue.
- Or you can click the "link" to display the mission's archive folder and then select specific kernels from the kernel data folders, and/or “furnsh” meta-kernels (mk) and other items from the extras folder.

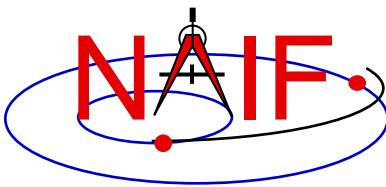
Name	Last modified	Size
Parent Directory		-
<a href="#">aareadme.htm</a>	28-Mar-2013 13:15	13K
<a href="#">aareadme.lbl</a>	28-Mar-2013 13:15	797
<a href="#">aareadme.txt</a>	28-Mar-2013 13:15	10K
<a href="#">catalog/</a>	22-Oct-2018 14:23	-
<a href="#">data/</a>	30-Jun-2005 14:25	-
<a href="#">document/</a>	13-Mar-2006 13:19	-
<a href="#">errata.txt</a>	23-Sep-2015 11:58	11K
<a href="#">extras/</a>	28-Mar-2013 13:15	-
<a href="#">index/</a>	22-Oct-2018 14:24	-
<a href="#">software/</a>	24-Nov-2007 09:02	-
<a href="#">voldesc.cat</a>	28-Jul-2008 11:30	2.0K

Up to higher level directory

Name
<a href="#">ck</a>
<a href="#">ek</a>
<a href="#">fk</a>
<a href="#">ik</a>
<a href="#">lsk</a>
<a href="#">pck</a>
<a href="#">sclk</a>
<a href="#">spk</a>

Up to higher level directory

Name
<a href="#">ckxtra</a>
<a href="#">extrinfo.txt</a>
<a href="#">mk</a>
<a href="#">orbnrm</a>



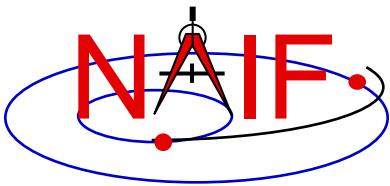
# Obtaining Archived Kernels from the NAIF Server - 2

## Navigation and Ancillary Information Facility

Mission Name	Archive Readme	Archive Link	PDS3 or PDS4	Data Size (GB)	Start Time	Stop Time	Subset Link
Cassini Orbiter	<a href="#">readme</a>	<a href="#">link</a>	3	47.4	1997-10-15	2014-09-30	<a href="#">subset</a>
Clementine	<a href="#">readme</a>	<a href="#">link</a>	3	0.8	1994-01-26	1994-05-07	<a href="#">subset</a>
DAWN	<a href="#">readme</a>	<a href="#">link</a>	3	13.5	2007-09-27	2012-09-13	<a href="#">subset</a>
Deep Impact	<a href="#">readme</a>	<a href="#">link</a>	3	0.7	2005-01-12	2005-08-09	<a href="#">subset</a>
Deep Space 1	<a href="#">readme</a>	<a href="#">link</a>	3	0.9	1998-10-24	2001-12-18	<a href="#">subset</a>
EPOXI	<a href="#">readme</a>	<a href="#">link</a>	3	1.0	2005-08-23	2011-03-01	<a href="#">subset</a>
GRAIL	<a href="#">readme</a>	<a href="#">link</a>	3	4.3	2011-09-10	2012-12-17	<a href="#">subset</a>
Hayabusa	<a href="#">readme</a>	<a href="#">link</a>	3	0.3	2005-09-11	2005-11-19	<a href="#">subset</a>
Lunar Reconnaissance Orbiter	<a href="#">readme</a>	<a href="#">link</a>	3	201.2	2009-06-18	2015-03-15	<a href="#">subset</a>

For “large” data sets that might take a long time to download, if you really need just a subset of the data covering a limited amount of time you should use the “Subset Link” for the data set of interest.

This process will automatically select just the kernels that fall within or overlap the time bounds you specify, construct a new “FURNSH” kernel(s) containing the names of this subset of kernels (thus making it easy for you to load the subset into your program), and create a custom wget script you may use to download these files to your computer.

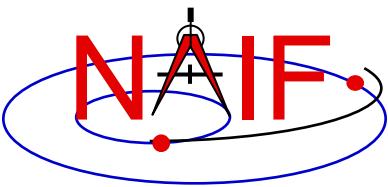


# Downloading Archived Kernels from the NAIF Server

---

Navigation and Ancillary Information Facility

- **Use GNU's wget, or FileZilla, or a similar utility to download the complete SPICE data set**
  - Example using wget on the Deep Impact mission:
    - » Open a terminal window
    - » wget -m -nH --cut-dirs=5 -nv (insert the URL of the "Archive Link" for the SPICE data set here). For example:
      - wget -m -nH --cut-dirs=5 -nv [http://naif.jpl.nasa.gov/pub/naif/pds/data/di-c-spice-6-v1.0/isp\\_1000/](http://naif.jpl.nasa.gov/pub/naif/pds/data/di-c-spice-6-v1.0/isp_1000/)
  - FileZilla info
    - » [http://filezilla-project.org/client\\_features.php](http://filezilla-project.org/client_features.php)



# Obtaining Generic Kernels

## Navigation and Ancillary Information Facility

NASA Jet Propulsion Laboratory California Institute of Technology + View the NASA Portal

# NAIF

The Navigation and Ancillary Information Facility

[Home](#)  
[Announcements](#)  
[About SPICE](#)  
[About NAIF](#)  
[For Projects](#)  
[For the Public](#)  
[Data](#) (circled)  
[Utilities](#)

SPICE Data (SPICE Kernels)

- [PDS Archived SPICE Data Sets](#)
- [Operational Flight Projects Kernels and Other Non-archived Project Kernels](#)
- [Generic Kernels](#) (circled)

As shown above, three categories of SPICE data, often referred to as kernels, are available from this website. You should carefully read about all three of these categories using the links below in order to find the best data for your needs.



NASA Jet Propulsion Laboratory California Institute of Technology + View the NASA Portal

# NAIF

The Navigation and Ancillary Information Facility

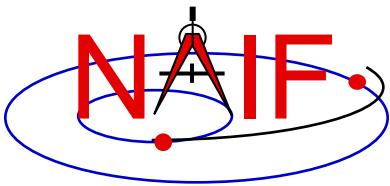
[Home](#)  
[Announcements](#)  
[About SPICE](#)  
[About NAIF](#)  
[For Projects](#)  
[For the Public](#)  
[Data](#) (circled)

Generic Kernels

Generic Kernels (circled) → SPICE kernels that exist independent of any particular flight project are called generic kernels. These may be obtained from the Generic Kernels link of the NAIF server appearing above.

**Generic kernels are just a few clicks away...**

Name	Last modified	Size
<a href="#">Parent Directory</a>		-
<a href="#">AACLEARANCE STATEMENT.pdf</a>	28-Sep-2017 16:47	49K
<a href="#">aareadme.txt</a>	07-Nov-2018 11:32	3.6K
<a href="#">dsk/</a>	08-Jul-2017 04:03	-
<a href="#">fk/</a>	02-Apr-2007 17:57	-
<a href="#">lsk/</a>	29-May-2018 16:36	-
<a href="#">pck/</a>	31-Oct-2019 17:07	-
<a href="#">spk/</a>	29-Aug-2013 14:25	-
<a href="#">stars/</a>	15-Feb-2007 17:36	-

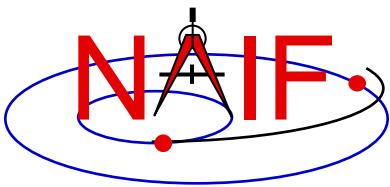


# Horizons

---

Navigation and Ancillary Information Facility

- **The Horizons server is an on-line ephemeris generator for natural bodies (and more)**
  - It is operated by JPL's Solar System Dynamics Group, not by NAIF
- **Of primary interest to SPICE users is its ability to generate up-to-date SPK files for comets and asteroids**
  - Horizons home page:
    - » <http://ssd.jpl.nasa.gov/?horizons>
  - Horizons web interface for manual generation of small body SPKs:
    - » <http://ssd.jpl.nasa.gov/x/spk.html>
  - Horizons telnet interface for automated (programmatic) generation of small body SPKs:
    - » **telnet ssd.jpl.nasa.gov 6775**
    - » **For an example script, use anonymous ftp to go to:**
      - **ssd.jpl.nasa.gov**
    - » **and once there, look at:**
      - **/pub/ssd/SCRIPTS/smb\_spk**

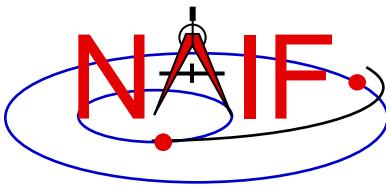


---

Navigation and Ancillary Information Facility

# Summary of Key Points

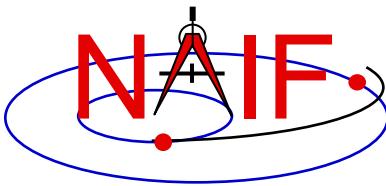
January 2020



# Which Pieces of SPICE Must I Use?

Navigation and Ancillary Information Facility

- **There's not a simple answer**
  - Depends on what activity or mission you are working on
  - Depends on what computation(s) you wish to make
- **Don't feel overwhelmed**
  - Many seemingly complex computations can be made using just a few SPICE APIs
- **The next several charts highlight some key points**
  - We assume you have already looked at the major SPICE tutorials, or already have some familiarity with SPICE
  - We assume you have successfully downloaded and installed the SPICE Toolkit
- **Consider printing this tutorial and keeping it near your workstation**



# Reminder of Key Subsystems

Navigation and Ancillary Information Facility

**SPK:** Position (and velocity) of things (“ephemeris objects”)

**PCK:** Size/shape/orientation of solar system bodies

For binary PCKs, only orientation is provided; use a text PCK to obtain size/shape

See also DSK below

**IK:** Instrument field-of-view geometry (see also FK below)

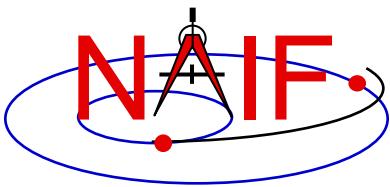
**CK:** Orientation of spacecraft or spacecraft structures that rotate

**FK:** Definition and specification details for many reference frames; also includes instrument mounting alignments

**DSK:** High fidelity shape data, better than what's in a text PCK  
(But limited availability)

**LSK:** Time conversion: UTC (SCET)  $\longleftrightarrow$  ET (TDB)

**SCLK and LSK:** Time conversion: SCLK  $\longleftrightarrow$  ET (TDB)



# Primary Kernel Interfaces - 1

Navigation and Ancillary Information Facility

Which SPICE APIs are most commonly used with a given kernel type?

SPK	SPKEZR, SPKPOS, SPKCOV, SPKOBJ
PCK	SXFORM, PXFORM, SPKEZR, SPKPOS, BODVRD
IK	GETFOV, G*POOL
CK	SXFORM, PXFORM SPKEZR, SPKPOS, CKCOV, CKOBJ (CKGPAV, CKGP)

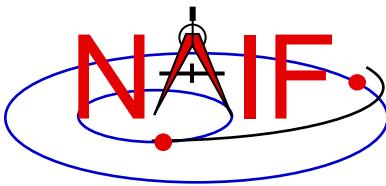
FK	SXFORM, PXFORM, SPKEZR, SPKPOS
LSK	STR2ET, TIMOUT, SCE2C, SCT2E, SCE2S, SCS2E
SCLK	SCS2E, SCE2S SXFORM, PXFORM, SPKEZR, SPKPOS
DSK*	SINCPT, LATSRF, ILLUMF, SRFNRM, LIMBPT, TERMPT, ...

\* Partial implementation starting with N66 Toolkits

Notes: FURNSH is used to load (provide access to ) all SPICE kernels.

API names shown are for FORTRAN versions:

- use lower case and add an “\_c” when using C
- use lower case and prepend “cspice\_” when using Icy (IDL) and Mice (MATLAB)



# Primary Kernel Interfaces - 2

Navigation and Ancillary Information Facility

For a given high-level Toolkit API, which kinds of kernels will or may be needed?

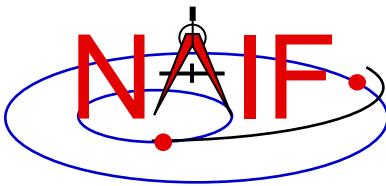
*Kernel Type(s) Needed*

API Name	SPK	PCK	IK	CK	FK	LSK	SCLK	DSK
SPKEZR, SPKPOS	Y	M		M	M	L	M	
SXFORM, PXFORM	M	M		M	L	M	M	
CKGP, CKGPAV	M	M		Y	M	L	L	
GETFOV			Y					
G*POOL		M	M					
STR2ET, TIMOUT						Y		
SCS2E, SCE2S						Y	Y	
CHRONOS (time conversion app.)	M	M		M	M	Y	M	

Yes = is needed

Likely = very likely needed

Maybe = may be needed



# Primary Kernel Interfaces - 3

Navigation and Ancillary Information Facility

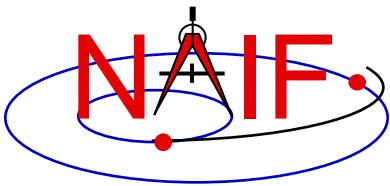
More: for a given high-level API, which kinds of kernels will or may be needed?

*Kernel Type(s) Needed*

API Name	SPK	PCK	IK	CK	FK	LSK	SCLK	DSK*
SINCPT	Y	L	M	M	L	L	M	M
DSKXV, DSKXSI	M	M	M	M	M	M	M	Y
LATSRF		M		M	M	M	M	M
ILUMIN, ILLUMG, ILLUMF	Y	L		M	M	L	M	M
SUBPNT, SUBSLR	Y	L		M	M	L	M	M
GFOCLT, OCCULT	Y	L		M	M	L	M	M
SRFNRM		M		M	M	M	M	M
LIMBPT	Y	L		M	M	L	M	M
TERMPT	Y	Y		M	M	L	M	M

\* Partial implementation starting with N66 Toolkits

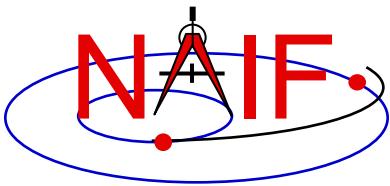
Yes = is needed  
Likely = likely needed  
Maybe = may be needed



# Kernel “Coverage” Cautions

Navigation and Ancillary Information Facility

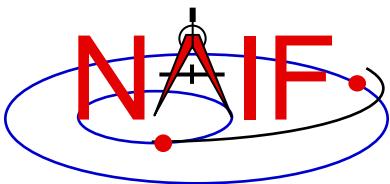
- Your set of kernels must:
  - contain data for all “objects” of interest
    - » Sometimes you must include intermediary objects that provide a connection (recall the chaining discussion in the SPK tutorial)
  - contain data covering the time span(s) of interest to you
    - » Watch out for data gaps within that time span
    - » Watch out for the difference between ET and UTC
      - The difference as of 2017 January 01 is ~69.182 seconds (ET > UTC)
  - contain all the kernel types needed by SPICE to answer your question
    - » As the previous charts show, you may need one or more kernels that are not obvious
  - be managed (loaded) properly if there are overlapping (competing) data within the set of files you are using



# What Kernels are Available?

Navigation and Ancillary Information Facility

- It depends on the mission or task you are working on.
- There are typically three categories of kernel data available.
  - **Mission operations** kernels – those used by the flight teams to fly the mission and prepare the archival science products
    - » These are the most up to date, but it could be a challenge to select the ones you need
  - **PDS Archived** kernels – those that have been selected from (or made from) the mission ops kernels, and then are well organized and documented for the PDS archive.
    - » These data sets are well organized, well documented, and contain helpful “furnsh” kernels (meta-kernels).
  - **Generic** kernels – those that are used by many missions and are not tied to any one mission
    - » Relevant generic kernels are usually included in the **PDS Archived** and the **Mission Operations** kernels data sets mentioned above
  - All three types can be found here: <https://naif.jpl.nasa.gov/naif/data.html>
- The situation might be similar for non-JPL missions, but this is up to whatever institution is producing the kernels.

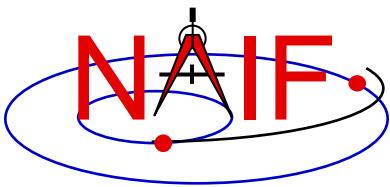


# How Can I Find Possibly Useful Toolkit APIs?

---

Navigation and Ancillary Information Facility

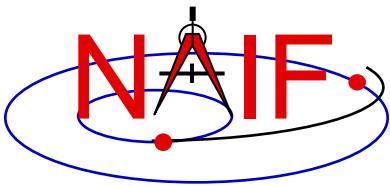
- **Review the previous charts**
- **Look at the appropriate SPICE tutorial(s)**
- **Look at the “Most Used xxx APIs” document**  
  .../doc/html/info/mostused.html
- **Search the permuted index:**
  - **spicelib\_idx** for the FORTRAN toolkits .../doc/html/info/spicelib\_idx.html
    - » This index also correlates entry point names with source code files.
  - **cspice\_idx** for the C toolkits .../doc/html/info/cspice\_idx.html
  - **icy\_idx** for the IDL toolkits .../doc/html/info/icy\_idx.html
  - **mice\_idx** for the MATLAB toolkits .../doc/html/info/mice\_idx.html
- **Read relevant portions of a SPICE “required reading” technical reference document (e.g. “spk.req”)**
  - .../doc/html/req/spk.html for the hyperlinked html version (best)
  - .../doc/spk.req for the plain text version



# How Can I Understand How To Use Those APIs?

Navigation and Ancillary Information Facility

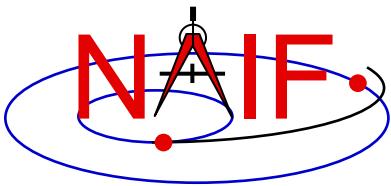
- The primary user-oriented documentation about each API is found in the “header” located at the top of each source code file and also in the API’s HTML page in the API reference guide.
  - You can “Google” an API name to see its header
    - » For example: spkezr, spkezr\_c, or cspice\_spkezr (for Icy or Mice)
  - (More documentation is found at the additional entry points for those FORTRAN APIs that have multiple entry points.)
- Reference documentation for major subsystems is found in like-named “required reading” documents (e.g. spk.req, ck.req, etc.)
- The SPICE tutorials contain much helpful information.
- NAIF’s self-training materials provide an orderly approach to learning about SPICE:
  - [https://naif.jpl.nasa.gov/naif/self\\_training.html](https://naif.jpl.nasa.gov/naif/self_training.html)



# Does NAIF Provide Any Examples?

Navigation and Ancillary Information Facility

- **Nearly all API headers contain one or more working examples**
- **“Most Useful SPICELIB Subroutines” has code fragments**  
.../doc/html/info/mostused.html
- **The “required reading” reference documents often contain examples** .../doc/html/req/index.html
- **The “Program\_<language>” tutorial contains a substantial working example**
- **Some simple “cookbook” programs are found in the Toolkit**  
.../src/cookbook/...
- **Make use of the SPICE Programming Lessons available from the NAIF server**
  - [ftp://naif.jpl.nasa.gov/pub/naif/toolkit\\_docs/Lessons/](ftp://naif.jpl.nasa.gov/pub/naif/toolkit_docs/Lessons/)



---

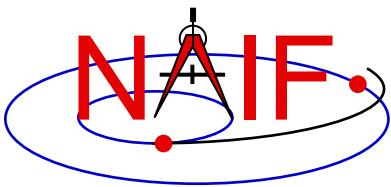
Navigation and Ancillary Information Facility

# WebGeocalc (WGC)

<https://wgc.jpl.nasa.gov:8443/webgeocalc> (GUI only)

<https://wgc2.jpl.nasa.gov:8443/webgeocalc> (GUI and API)

January 2020

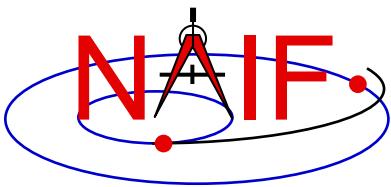


# Overview

---

Navigation and Ancillary Information Facility

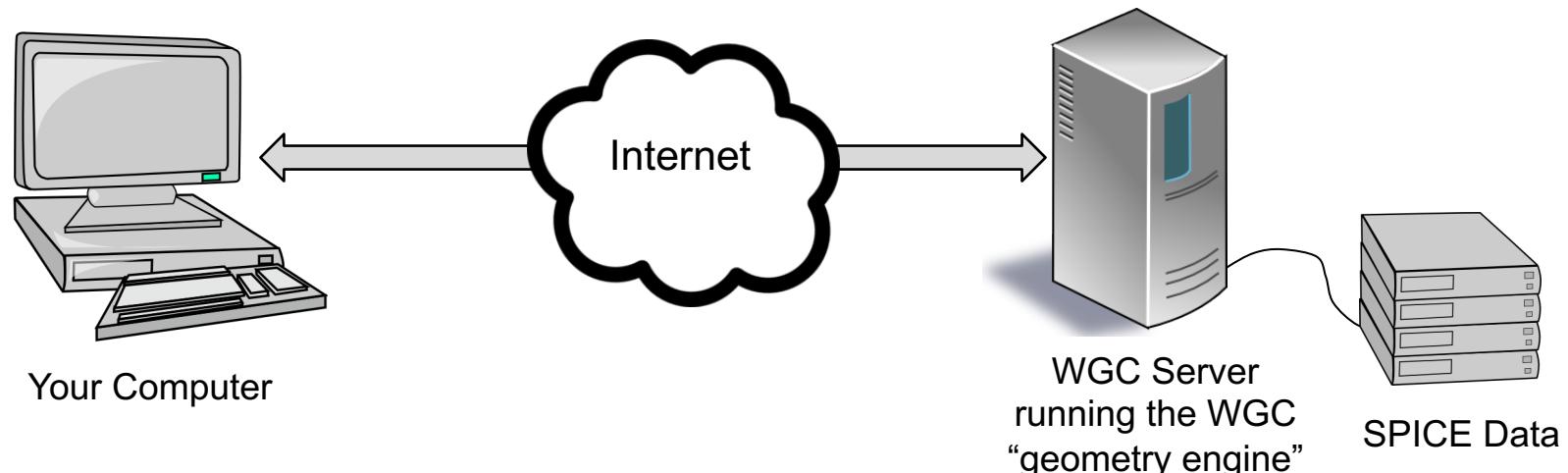
- **WebGeocalc (WGC) provides a graphical user interface (GUI) to a SPICE server running a geometry computation engine**
  - The server has access to SPICE kernels containing ancillary data
- **Some WGC installations also offer a programmatic(API) interface to the WGC SPICE geometry engine**
- **This tutorial covers mostly the GUI interface. The API interface is described in a link provided in the API-enabled version of WGC**

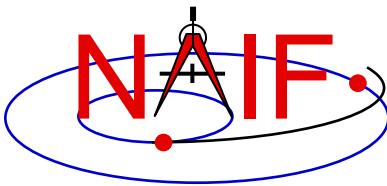


# Architecture

Navigation and Ancillary Information Facility

- **WGC uses a client-server architecture**
  - The user needs only a computer running a web browser
  - The browser connects via Internet to a WGC “computation engine” running on a server
    - » The WGC server has access to a variety of SPICE kernel files

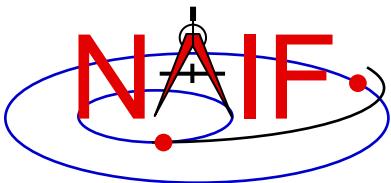




# Using WebGeocalc

Navigation and Ancillary Information Facility

- **WGC makes it “easy” to do many kinds of SPICE computations**
  - You need not write a program using SPICE Toolkit software
  - Instead, open a web browser and use standard GUI widgets to:
    - » select the computation desired
    - » select the data to be used in your computation
    - » specify the computation details
    - » press the “CALCULATE” button
  - Your results, possibly including some plots, appear in your browser window
  - WGC has a good deal of built-in HELP
- **WGC computations are limited in scope: the tool cannot do nearly as much as an own-built program that uses SPICE Toolkit APIs**
  - But WGC can probably do a good deal more than you first realize!

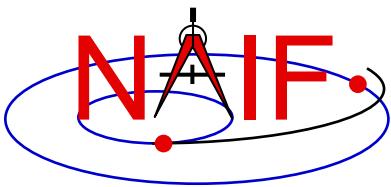


# Purpose

---

Navigation and Ancillary Information Facility

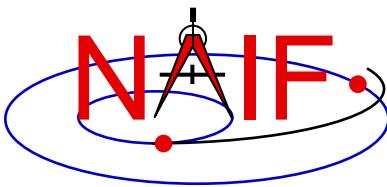
- **WGC can support planetary science in several ways**
  - Help a user check his/her own SPICE-based program under development (the “gold bar” check)
  - Help a user quickly solve a one-time space geometry problem
  - Allow those unable to write a SPICE-based program to nevertheless make some kinds of space geometry computations
  - Help a science data peer reviewer do spot checks of geometry parameters contained in an archive about to be submitted to an archive center



# Computations

Navigation and Ancillary Information Facility

- Three categories of SPICE computations are possible
  - 1. Geometry Calculator
    - » Compute a parameter value at a given time, or over a time range
      - Example: Compute the angular size of Phobos as seen from the SPIRIT Mars rover from 2009 March 10 12:00:00 to 2009 March 10 14:00:00
  - 2. Geometric Event Finder
    - » Within a specified time bounds (the confinement window)...
      - Find time intervals when a particular geometric condition exists
        - Example: Find time intervals when Phobos is occulted by Mars as seen from Mars Odyssey within the period 2010 June 01 to 2010 June 02
      - Find time intervals when a geometry parameter is within a given range
        - Example: Find time intervals when the spacecraft altitude is between 300 and 400 km
      - Find time intervals when a geometry parameter has reached a local or global maximum or minimum
        - Example: Find time intervals when the angular separation of a satellite from a planet, as seen from a spacecraft, has reached its minimum value
  - 3. Time conversion calculator
    - » Convert between various time systems and time formats
- See the WGC “menu” on the next page for some details



# Computation Menu\*

## Navigation and Ancillary Information Facility Geometry Calculator

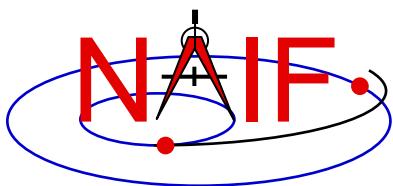
<a href="#">State Vector</a>	Calculate the position and velocity of a target with respect to an observer.
<a href="#">Angular Separation</a>	Calculate the angular separation between two targets as seen from an observer.
<a href="#">Angular Size</a>	Calculate the angular size of a target as seen from an observer.
<a href="#">Frame Transformation</a>	Calculate the transformation between two reference frames.
<a href="#">Illumination Angles</a>	Calculate the emission, phase and solar incidence angles at a point on a target as seen from an observer.
<a href="#">Sub-solar Point</a>	Calculate the sub-solar point on a target as seen from an observer.
<a href="#">Sub-observer Point</a>	Calculate the sub-observer point on a target as seen from an observer.
<a href="#">Surface Intercept Point</a>	Calculate the intercept point of a vector or vectors on a target as seen from an observer.
<a href="#">Orbital Elements</a>	Calculate the osculating elements of the orbit of a target body around a central body.

## Geometric Event Finder

<a href="#">Position Finder</a>	Find time intervals when a coordinate of an observer-target position vector satisfies a condition.
<a href="#">Angular Separation Finder</a>	Find time intervals when the angle between two bodies, as seen by an observer, satisfies a condition.
<a href="#">Distance Finder</a>	Find time intervals when the distance between a target and observer satisfies a condition.
<a href="#">Sub-Point Finder</a>	Find time intervals when a coordinate of the sub-observer point on a target satisfies a condition.
<a href="#">Occultation Finder</a>	Find time intervals when an observer sees one target occulted by, or in transit across, another.
<a href="#">Surface Intercept Finder</a>	Find time intervals when a coordinate of a surface intercept vector satisfies a condition.
<a href="#">Target in Field of View</a>	Find time intervals when a target intersects the space bounded by the field-of-view of an instrument.
<a href="#">Ray in Field of View</a>	Find time intervals when a specified ray is contained in the space bounded by an instrument's field-of-view.
<a href="#">Range Rate Finder</a>	Find time intervals when the range rate between a target and observer satisfies a condition.
<a href="#">Phase Angle Finder</a>	Find time intervals when the phase angle between a target and illuminator satisfies a condition.

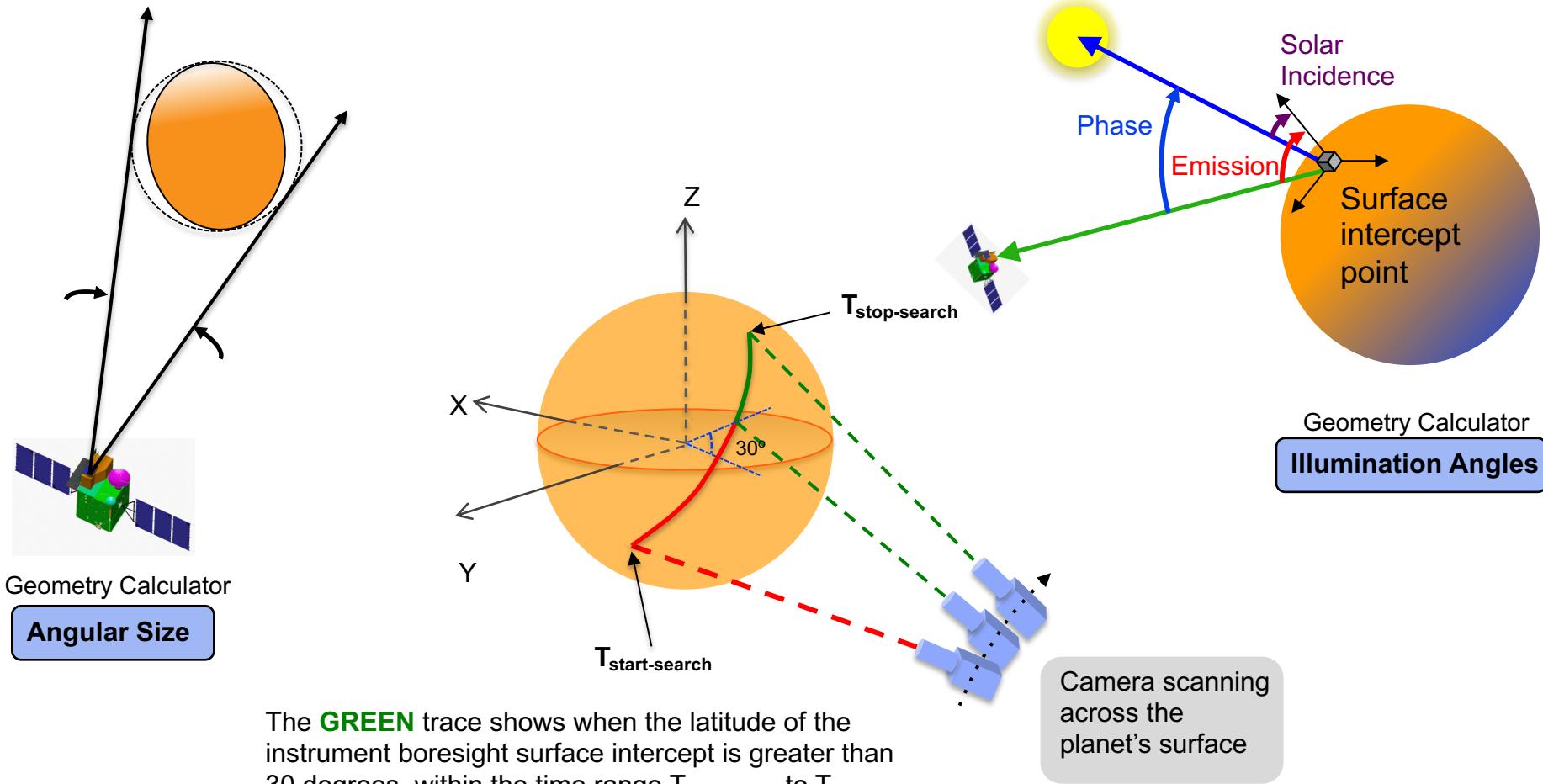
## Time Calculator

<a href="#">Time Conversion</a>	Convert times from one time system or format to another.
---------------------------------	--



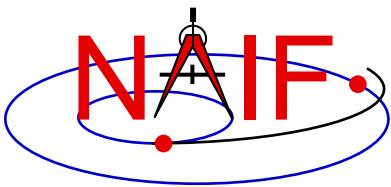
# Illustrations of Three Available Computations

Navigation and Ancillary Information Facility



Geometric Event Finder

Surface Intercept Event Finder



# Typical Geometry Calculator Input

## Navigation and Ancillary Information Facility

### Angular Size

Calculate the angular size of a target as seen from an observer. [?](#)

Kernel selection: MER2 Rover (Spirit) [?](#)

Target: PHOBOS [?](#)

Observer: SPIRIT [?](#)

#### Aberration Correction

Light propagation:  None  To observer  From observer [?](#)

Light-time algorithm: Converged Newtonian [?](#)

Stellar aberration:  Include stellar aberration correction [?](#)

#### Input Time

Time system: UTC [?](#)

Time format: Calendar date and time [?](#)

Input times:  Single time  Single interval  List of times  List of intervals

Start time: 2009 MAR 10 12:00:00 [?](#)

Stop time: 2009 MAR 10 14:00:00 [?](#)

Time step: 1 minutes [?](#)

#### Plots

Time series plots:  Angular Size [?](#)

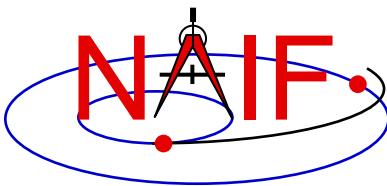
X-Y plots: X: Angular Size vs. Y: Angular Size [Add Plot](#)

Error handling: Stop on error [?](#)

[Calculate](#)

- Compute the angular size of Phobos as seen from the Mars rover “SPIRIT” over a two hour period on 2009 March 10.

- Use typical GUI drop-down menus, fill-in boxes, radio buttons and check boxes to specify the details of the computation you wish to make.



# Typical Geometry Calculator Output

Navigation and Ancillary Information Facility

## Input Values

Calculation type	Angular Size
Target	PHOBOS
Observer	SPIRIT
Light propagation	No correction
Time system	UTC
Time format	Calendar date and time
Time range	2009 MAR 10 12:00:00 to 2009 MAR 10 14:00:00, step 1 minutes

Summary of your input

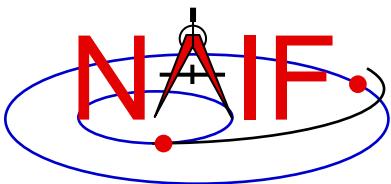
## Tabular Results

Click a value to save it for a subsequent calculation.

	UTC calendar date	Angular Size (deg)
1	2009-03-10 12:00:00.000000 UTC	0.20212256
2	2009-03-10 12:01:00.000000 UTC	0.20294481
3	2009-03-10 12:02:00.000000 UTC	0.20377024
4	2009-03-10 12:03:00.000000 UTC	0.20459871
5	2009-03-10 12:04:00.000000 UTC	0.20543007
6	2009-03-10 12:05:00.000000 UTC	0.20626418
7	2009-03-10 12:06:00.000000 UTC	0.20710088
8	2009-03-10 12:07:00.000000 UTC	0.20794000
9	2009-03-10 12:08:00.000000 UTC	0.20878138
10	2009-03-10 12:09:00.000000 UTC	0.20962484
11	2009-03-10 12:10:00.000000 UTC	0.21047019
12	2009-03-10 12:11:00.000000 UTC	0.21131725
13	2009-03-10 12:12:00.000000 UTC	0.21216581
14	2009-03-10 12:13:00.000000 UTC	0.21301567

*Angular size of Phobos as seen from the Mars rover “SPIRIT”*

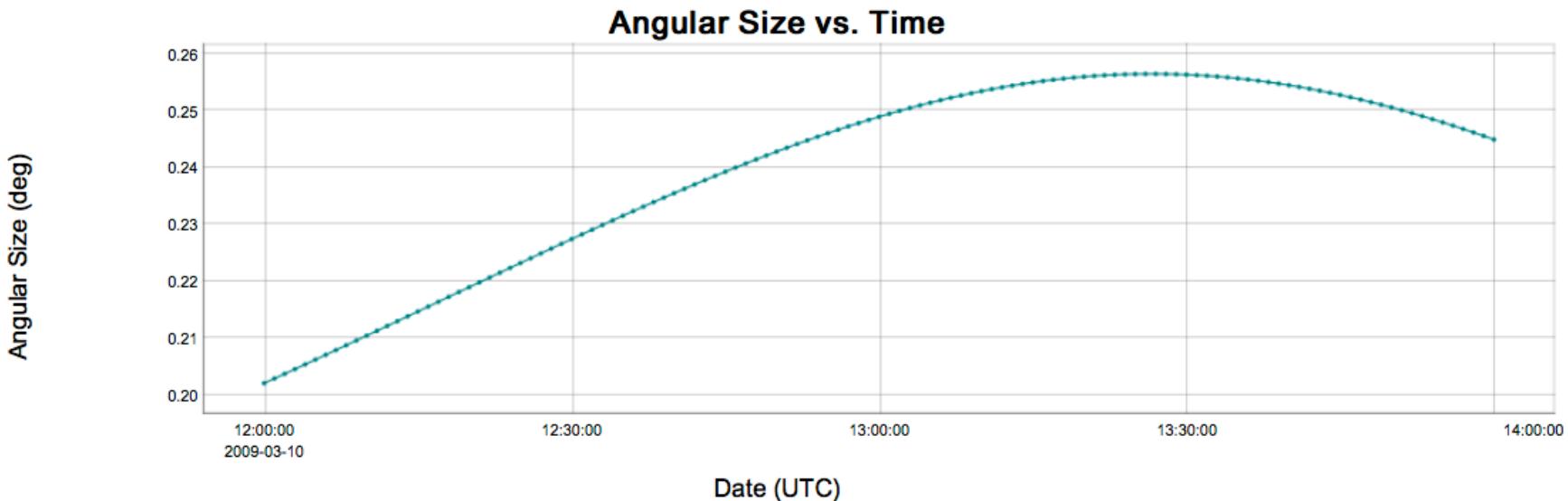
Tabular results



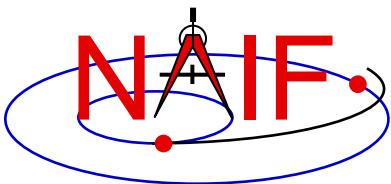
# Typical Geometry Calculator Plot

Navigation and Ancillary Information Facility

- Some Geometry Calculator computations offer optional plots



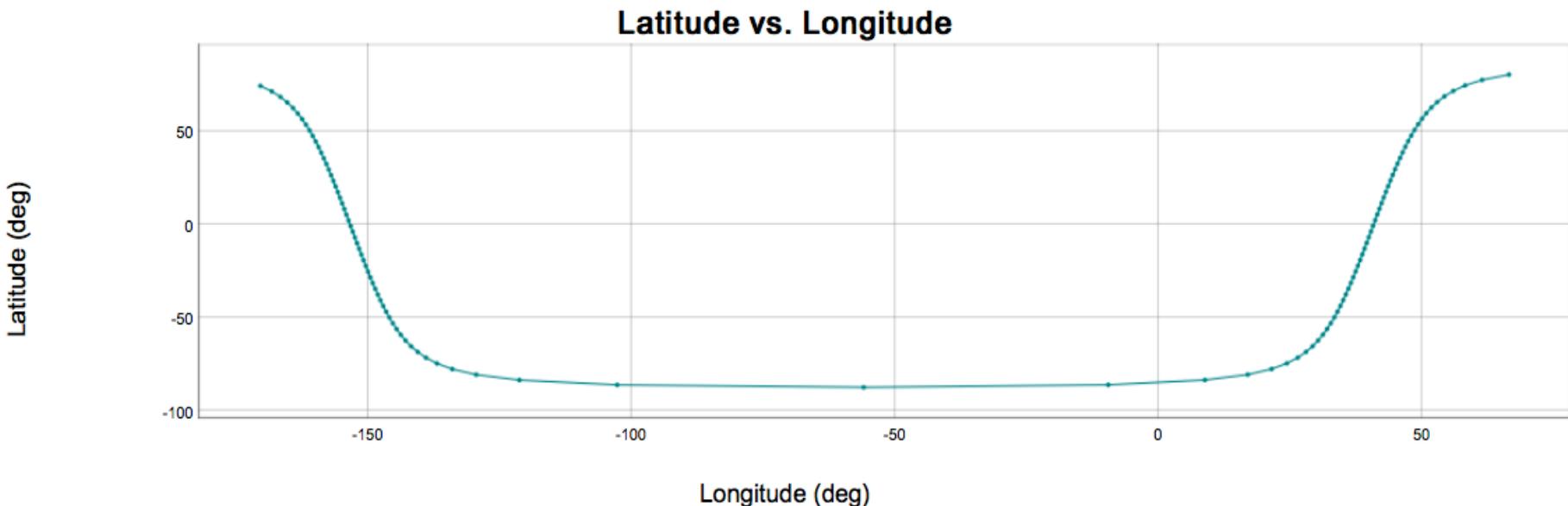
*Angular size of Phobos as seen from the Mars rover "SPIRIT"*



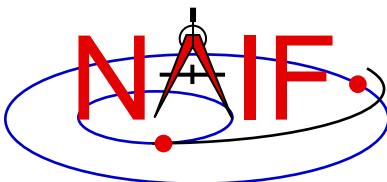
# Another Geometry Calculator Plot

Navigation and Ancillary Information Facility

- Some Geometry Calculator computations offer plots using other than time on the X axis



***Mars Global Surveyor sub-point on Mars  
from 2008 JAN 1 00:10:00 to 2008 JAN 1 02:00:00***



# Typical Geometric Event Finder Input

## Navigation and Ancillary Information Facility

### Occultation Event Finder

Find time intervals when an observer sees one target occulted by, or in transit across, another. [?](#)

Kernel selection: Mars Odyssey [?](#)

Occultation type:  Any  Full  Annular  Partial [?](#)

Front body: MARS [?](#)

Front body shape:  Point  Ellipsoid  DSK model [?](#)

Front body frame: IAU\_MARS [?](#)

Back body: PHOBOS [?](#)

Back body shape:  Point  Ellipsoid  DSK model [?](#)

Back body frame: IAU\_PHOBOS [?](#)

Observer: MARS ODYSSEY [?](#)

Aberration Correction

Light propagation:  None  To observer  From observer [?](#)

Input Time

Time system: UTC [?](#)

Time format: Calendar date and time [?](#)

Input times:  Single interval  List of intervals

Start time: 2010 JUN 01 [?](#)

Stop time: 2010 JUN 02 [?](#)

Time step: 1 [?](#) minutes [?](#)

Result Window

Output time units:  seconds  minutes  hours  days [?](#)

Complement:  Complement result window [?](#)

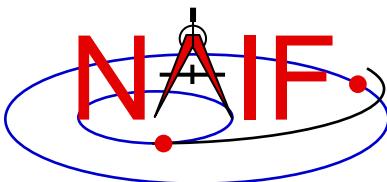
Adjust endpoints: No adjustment [?](#)

Adjust amount:  seconds [?](#)

Filter intervals: No filtering [?](#)

Filter threshold:  seconds [?](#)

- Find the times when Phobos is occulted by Mars as viewed from the Mars Odyssey spacecraft, during the period 2010 JUN 01 to 2010 JUN 02.
- Use typical GUI drop-down menus, fill-in boxes, radio buttons and check boxes to specify the details of the computation you wish to make.



# Typical Geometric Event Finder Output

## Navigation and Ancillary Information Facility

### Input Values

Calculation type	Occultation Event Finder
Occultation type	Any
Front body	MARS
Front body shape	Ellipsoid
Front body frame	IAU_MARS
Back body	PHOBOS
Back body shape	Ellipsoid
Back body frame	IAU_PHOBOS
Observer	MARS ODYSSEY
Light propagation	No correction
Time system	UTC
Time format	Calendar date and time
Time range	2010 JUN 01 to 2010 JUN 02
Step	1 minutes
Output time unit	minutes
Complement result window	no
Result interval adjustment	No adjustment
Result interval filtering	No filtering

Summary of your input

### Tabular Results

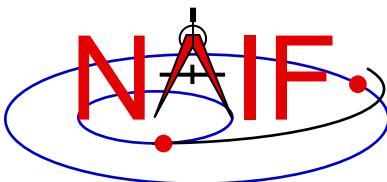
Click a value to save it for a subsequent calculation.

	Start Time	Stop Time	Duration (mins)
1	2010-06-01 00:04:26.044991 UTC	2010-06-01 00:51:10.243911 UTC	46.73664866
2	2010-06-01 01:24:29.583277 UTC	2010-06-01 02:00:24.436145 UTC	35.91421447
3	2010-06-01 03:03:10.426707 UTC	2010-06-01 03:57:18.102560 UTC	54.12793088
4	2010-06-01 06:01:49.759691 UTC	2010-06-01 06:55:34.702564 UTC	53.74904789
5	2010-06-01 07:58:43.124284 UTC	2010-06-01 08:39:21.185492 UTC	40.63435347
6	2010-06-01 09:10:48.850884 UTC	2010-06-01 09:54:44.464484 UTC	43.92689334
7	2010-06-01 10:57:18.650296 UTC	2010-06-01 11:50:49.315679 UTC	53.51108973
8	2010-06-01 13:55:36.207413 UTC	2010-06-01 14:49:37.807259 UTC	54.02666411
9	2010-06-01 15:53:04.681014 UTC	2010-06-01 16:24:27.102771 UTC	31.37369595
10	2010-06-01 17:00:06.171340 UTC	2010-06-01 17:48:55.450641 UTC	48.82132168
11	2010-06-01 18:51:22.483546 UTC	2010-06-01 19:43:35.606641 UTC	52.21871826
12	2010-06-01 20:25:04.776643 UTC	2010-06-01 20:44:18.007484 UTC	19.22051402
13	2010-06-01 21:49:30.118805 UTC	2010-06-01 22:43:33.989673 UTC	54.06451446

*When is Phobos occulted by Mars as seen from Mars Odyssey?*

Tabular results





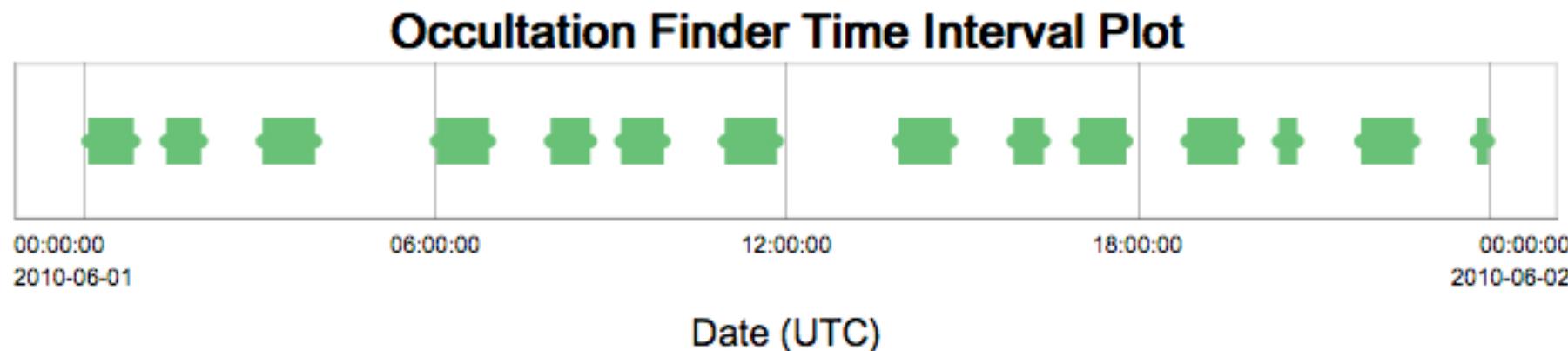
# Typical Geometric Event Finder Plot

Navigation and Ancillary Information Facility

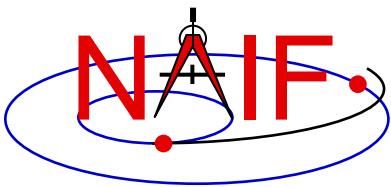
- **Geometric Event Finder computations all produce “plots” of the time intervals that satisfy your search computations**

Click and drag to zoom, shift-click and drag to pan. Double-click or use button to reset zoom level.

[Download Plot](#)   [Prior Zoom](#)   [Reset Zoom](#)



*Between June 1, 2010 and June 2, 2010, find times when Phobos is occulted by Mars, as viewed from the Mars Odyssey spacecraft*



# Example of Time Conversion

Navigation and Ancillary Information Facility

Convert a spacecraft clock string to UTC

## Time Conversion

Convert times from one time system or format to another.

Kernel selection:

Lunar Reconnaissance Orbiter

### Input Time

Time system:

Spacecraft clock

Spacecraft clock ID:

-85

Time format:

Spacecraft clock string

Input times:

Single time

Single interval

List of times

List of intervals

Spacecraft clock string  
Spacecraft clock ticks

Time:

1/0330220800.000

### Output Time

Time system:

UTC

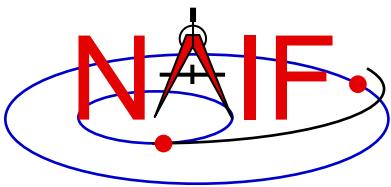
Calendar (year/month/day)  
Calendar (year/day-of-year)  
Julian date  
Seconds past J2000  
Custom format

Time format:

Calendar (year-month-day)

Custom format:

The output is:  
2011-06-20 00:00:00.044032 UTC



# Second Example of Time Conversion

Navigation and Ancillary Information Facility

## Time Conversion

Compute a series of UTC times with a specified time step

Convert times from one time system or format to another.

Kernel selection:

Mars Reconnaissance Orbiter



### Input Time

Time system:

UTC



Time format:

Calendar date and time



Input times:

Single time

Single interval

List of times

List of intervals

Start time:

2011 MAR 10



Stop time:

2011 MAR 11



Time step:

20

minutes



UTC  
TDB  
TDT  
Spacecraft clock

Calendar (year/month/day)  
Calendar (year/day-of-year)  
Julian date  
Seconds past J2000

### Output Time

Time system:

UTC



Time format:

Julian date

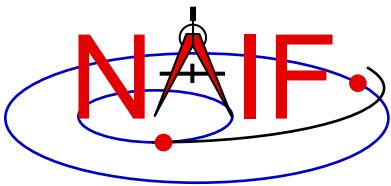


Custom format:



The output is:

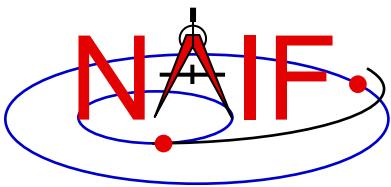
2011-03-10 00:00:00.000000 UTC 2455630.500000000 JD UTC  
2011-03-10 00:20:00.000000 UTC 2455630.513888900 JD UTC  
2011-03-10 00:40:00.000000 UTC 2455630.527777800 JD UTC  
2011-03-10 01:00:00.000000 UTC 2455630.541666700 JD UTC  
etc.



# Categories of Available Data

Navigation and Ancillary Information Facility

- The JPL/NAIF Group is operating two WGC servers
  - These servers provide access to three categories of SPICE kernels
    - » Generic SPICE kernels, not specifically tied to a single planetary mission
    - » Archived SPICE kernels, from planetary missions that have been formally ingested into NASA's Planetary Data System
      - This includes a few non-NASA missions for which NAIF provides a shadow archive
    - » Operations SPICE kernels, for JPL-operated planetary missions, for three ESA planetary missions, and for a few past missions for which an archive does not exist
      - This category often includes some predictive data
      - This category is the most difficult to use because...
        - there are no meta-kernels for these collections
        - there is sometimes a large number of kernels from which you must choose the ones needed
        - there is little readily available information to help you make your kernel choices
  - **VERY IMPORTANT:** Read the “*About the data*” webpage provided within the tool for details



# Kernel Selection

Navigation and Ancillary Information Facility

## Angular Size

Calculate the angular size of a target as seen from an observer.

Kernel selection:

Target:

Solar System Kernels  
Latest Leapseconds Kernel  
Latest Planetary Constants Kernel  
Ground Stations Kernels

Aberration Correction

Cassini Huygens

Light propagation:

Clementine  
Dawn

Light-time algorithm:

Deep Impact (Primary mission)  
Deep Impact (EPOXI mission)

Stellar aberration:

Deep Space 1  
GRAIL

Input Time

Hayabusa  
JUNO

Time system:

Lunar Reconnaissance Orbiter  
MAVEN

Time format:

MER1 Rover (Opportunity)  
MER2 Rover (Spirit)

Input times:

MESSENGER  
Mars Express

Start time:

.

Stop time:

.

Time step:

Manual

1 minutes

Plots:

Angular Size

Error handling:

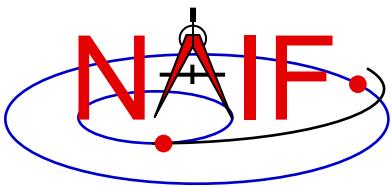
Stop on error

Calculate

**A scrollable drop-down menu is used to select the kernel set(s) to be used in your calculation.**

**Use the menu to select:**

- generic kernel sets
- archived mission kernel sets  
(includes relevant generic kernels)
- manual selection of individual kernels from operations collections



# “Tooltip” Feature

Navigation and Ancillary Information Facility

## Angular Size

Calculate the angular size of a target as seen from an observer.

Kernel selection:

MER2 Rover (Spirit)

Target:

- Solar System Kernels
- Latest Leapseconds Kernel
- Latest Planetary Constants Kernel
- Ground Stations Kernels
- Cassini Huygens
- Clementine
- Dawn
- Deep Impact (Primary mission)
- Deep Impact (EPOXI mission)
- Deep Space 1
- GRAIL
- Hayabusa

Aberration Correction

Light propagation:

Light-time algorithm:

Stellar aberration:

Input Time

Time system:

Time format:

Input times:

Start time:

Stop time:

Time step:

MER2 Rover (Spirit)



If you hover your **cursor** over a kernel set name, some information about the kernel set will appear—for example, dates covered by the data.



Archived MESSENGER kernels covering from 2004-08-03 to 2015-04-30



times  List of intervals

Plots:

Angular Size

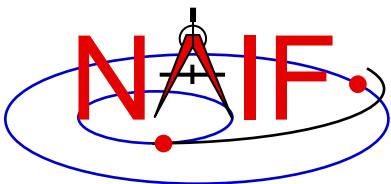
Error handling:

Stop on error

Calculate

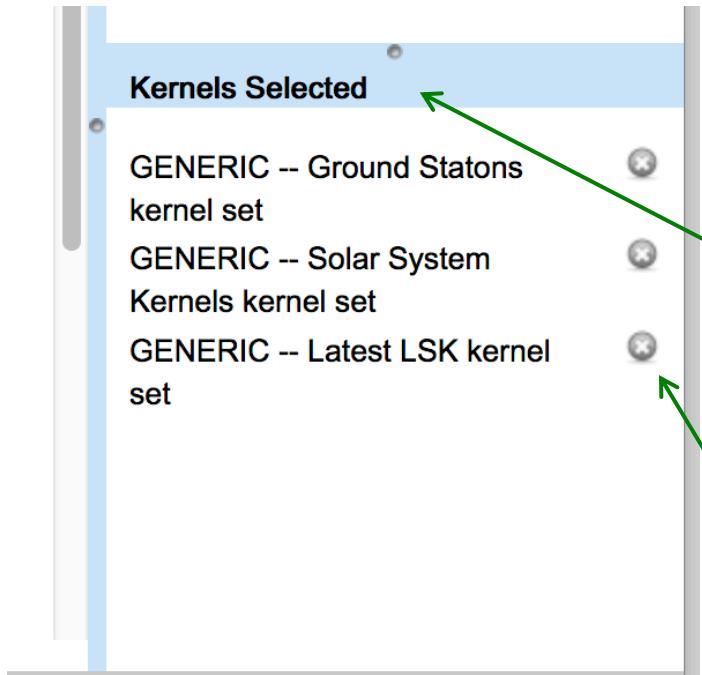
You can hover over the kernel set name in the “Kernel selection” menu, or in the “Kernels Selected” panel.

This feature is **not** available for “Manual” kernel selection.



# Pre-load Feature

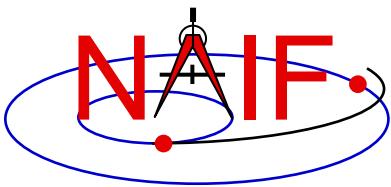
Navigation and Ancillary Information Facility



**WGC automatically loads at startup (“pre-loads”) generic kernel sets that are often needed in WGC calculations.**

**Pre-loaded kernel sets appear in the “Kernels Selected” area at the bottom-right of any WGC page.**

**Any of these pre-loaded kernel sets can be unloaded if desired using the (x) symbol appearing next to it**



# Auto-complete Feature

Navigation and Ancillary Information Facility

- If you select any kernel set(s) other than “Manual”, many of the input widgets will be supplied with the names of all available selections
  - Just start typing the name you want and all items matching what you typed will appear in a drop down menu
  - Alternatively, simply type a “blank” and all items available within the kernel set(s) you selected will appear
- In the example below, using the Cassini Huygens archive, the user has typed “mi” in the “Target” selection box. The names of the three objects containing those letters are displayed for the user’s selection. (All three are satellites of Saturn.)

Kernel selection:  

Target:    

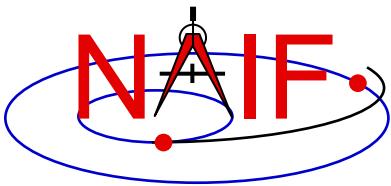
BERGELMIR  
MIMAS  
YMIR



# Downloading Results

Navigation and Ancillary Information Facility

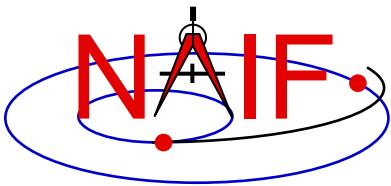
- You can download tabular results to your computer by clicking the “Download Results” button, then selecting the format desired:
  - Excel
  - Comma separated values
  - Plain text
  
- You can download any plots you've created by clicking on the “Download Plot” button
  - Plots are saved in PNG format with a transparent background
    - » Easily pasted into a document or presentation



# Saving Results for Use as New Inputs

Navigation and Ancillary Information Facility

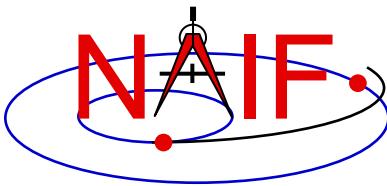
- You can save a numeric output, or an event finder interval start or stop time, by clicking on the value
  - The saved value will appear in a “Saved Values” panel on the right side of your browser window
  - This value can then be dragged to an input widget in a subsequent calculation
- You can save a complete set of event finder output interval start and stop times by clicking the “Save All Intervals” button
  - These can then be used as part of the input for a subsequent geometric event finder computation if you select “List of intervals” for the “Input times” selection. Simply drag the list of saved intervals to the “List of intervals” box.



# WGC Programmatic Interface

Navigation and Ancillary Information Facility

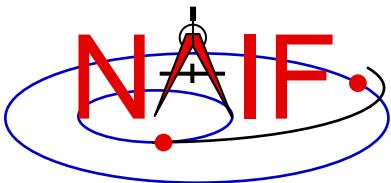
- NAIF also offers a programmatic(API) interface to the WGC SPICE geometry engine
  - To use this interface, one writes a program that constructs JSON payloads identifying the computation of interest, the kernel set(s) to be used, and the computation inputs, and submits them to the WGC server using RESTful URLs
  - The WGC SPICE geometry engine executes the computation and returns the results in JSON format
- The “[API Docs](#)” link found near the top-right of a WGC installation having the API interface points to a page with complete details
  - For the NAIF instance, that link is also provided here:  
<https://wgc2.jpl.nasa.gov:8443/webgeocalc/documents/api-info.html>



# Getting Help

Navigation and Ancillary Information Facility

- WGC users **must read** the “*About the Data*” web page to understand the kinds of SPICE kernels (data) available to the WGC tool
- Each calculation and most GUI controls have associated HELP available by clicking the ? icon
- Most computation descriptions have an associated graphic depicting one or more examples of what may be computed
- Some GUI controls have a second-level, more extensive help description, available by clicking the “Read more...” text displayed in the first level help

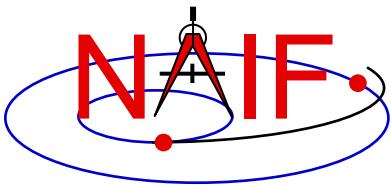


# Usage Rules

---

Navigation and Ancillary Information Facility

- The WGC program has a link entitled “*Rules of Use*”
  - WGC users must read and abide by these rules
- While easier than writing a SPICE-aware program, using WGC nevertheless requires some knowledge of space geometry and of NASA’s SPICE system
  - The NAIF website provides much SPICE information:
    - » <https://naif.jpl.nasa.gov>

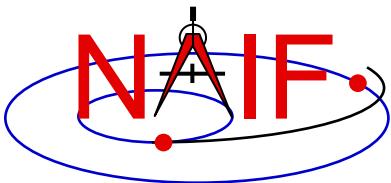


# Limited Capability

---

Navigation and Ancillary Information Facility

- **WGC does not provide all of the space geometry computational capability offered by the SPICE Toolkits**
  - But WGC nevertheless provides substantial capability—likely more than is obvious at first glance
- **More capability might be added if the user community finds this tool useful**



# Feedback

---

Navigation and Ancillary Information Facility

- **WGC includes a “*Feedback*” button, making it rather easy to provide the NAIF team with any sort of useful feedback...**
  - What you like or don’t like about WGC
  - What seems incorrect, incomplete or unclear
  - What features you would like to see added
    - » Caution: NAIF already has a long list of improvements we’d like to make, so we make no promises to act on any specific feedback

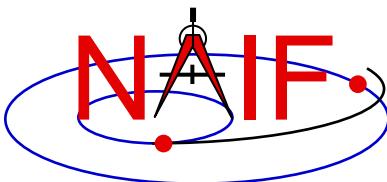


# Problems Using WGC

---

Navigation and Ancillary Information Facility

- **There are several limitations and errors you might encounter in using WebGeocalc**
  - See the next several pages for examples
  - Some of these conditions could be unique to the WGC installation at NAIF, and not exist with some other installation



# Missing Input

Navigation and Ancillary Information Facility

- WGC will alert you to missing inputs.

## Angular Separation

Calculate the angular separation between two targets as seen from an observer. 

A body name or code is required.

Kernel selection:  

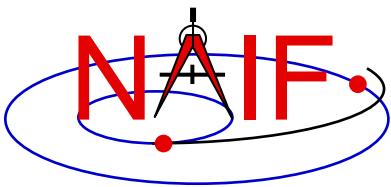
Target 1:  

Target 1 shape:  Point  Sphere 

Target 2:  

Target 2 shape:  Point  Sphere 

Observer:   \* Required



# Request is Too Large

Navigation and Ancillary Information Facility

- **WGC has limits set on computational resources.**
  - No more than 50,000 “Geometry Calculator” computations
  - No more than 10 million “Geometric Event Finder” time steps
  - No more than 3 minutes of wall clock time
- **If any of these limits will be (or have been) exceeded, you’ll see a message saying so and your computation request will be terminated.**
- **Some examples:**

Too many data points. This version of WebGeocalc can only calculate 50000 data points in a single calculation, and the requested time inputs specify 631152010 data points. Either use a smaller time range or a larger time step. NAI福 plans to remove this restriction in a future version of WebGeocalc.

Time step is too small. This version of WebGeocalc requires the time step to be at least 1.0E-7 times the size of the time window. The requested time step is only 9.506426208650559E-8 times the size of the window. Either specify a smaller time range or a larger time step.

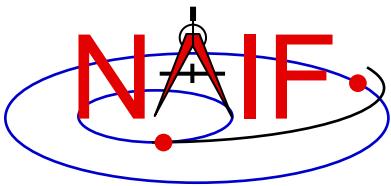


# SPICE Error Messages

---

Navigation and Ancillary Information Facility

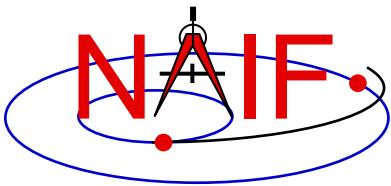
- **WGC will display a SPICE Toolkit error message when an underlying SPICE API is able to detect a problem.**
- **These SPICE error messages all begin with:**  
“CSPICE\_N00xx:” followed by some text
- **Refer to the remaining charts for some examples.**



# Unusable Time Tag(s) - 1

Navigation and Ancillary Information Facility

- Time tag processing within WGC makes use of SPICE Toolkit modules.
- These Toolkit modules offer a great deal of capability. But not every conceivable style of time tag, and not every time system, are acceptable.
- What to do?
  - Use the help icons  next to any Time system or Time format input.
  - See Appendix 2 of the [Chronos User's Guide](#) for some details and many examples.
    - » [https://naif.jpl.nasa.gov/misc/chronos\\_ug.html](https://naif.jpl.nasa.gov/misc/chronos_ug.html)
  - See the [Time Required Reading](#) document for a full explanation of time treatment within SPICE.
    - » [https://naif.jpl.nasa.gov/pub/naif/toolkit\\_docs/FORTRAN/req/time.html](https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/FORTRAN/req/time.html)



# Unusable Time Tag(s) - 2

Navigation and Ancillary Information Facility

- **A few examples of bad time tags and resultant error messages.**
  - Using UTC and Calendar format, and inputting: 2010:10:03 05:44:12
    - » Would have worked using: 2010-10-03 05:44:12 or 2010/10/03 05:44:12
  - Using UTC and Calendar format, and inputting: 2455160.098304
    - » Needed to specify “Julian date” format
  - Using UTC and Calendar format, and inputting: 2010:10:03T05:44:12
    - » Instead, use an ISO-defined format such as: 2010-10-03T05:44:12
      - (For separators, use dashes before the “T” and colons after the “T”)

CSPICE\_N0065: CSPICE.str2et: SPICE(UNPARSEDTIME): [str2et\_c --> STR2ET] The meaning of the integer <03> could not be determined: ' 2010:10:<03> 05:44:12'

» Would have worked using: 2010-10-03 05:44:12 or 2010/10/03 05:44:12

– Using UTC and Calendar format, and inputting: 2455160.098304

CSPICE\_N0065: CSPICE.str2et: SPICE(UNPARSEDTIME): [str2et\_c --> STR2ET] The meaning of the decimal number <2455160.098304> could not be determined: <2455160.098304>

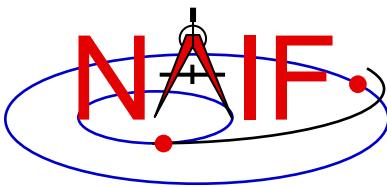
» Needed to specify “Julian date” format

– Using UTC and Calendar format, and inputting: 2010:10:03T05:44:12

CSPICE\_N0065: CSPICE.str2et: SPICE(UNPARSEDTIME): [str2et\_c --> STR2ET] The input string uses the ISO “T” date/time delimiter but does not match any of the accepted ISO formats.

» Instead, use an ISO-defined format such as: 2010-10-03T05:44:12

• (For separators, use dashes before the “T” and colons after the “T”)



# Unusable Time Tag(s) - 3

Navigation and Ancillary Information Facility

- A few more examples of bad time tags and resultant error messages.

- Using UTC and Calendar format, and inputting: 2010-10-03 05:44:12 JD

```
CSPICE_N0065: CSPICE.str2et: SPICE(UNPARSEDTIME): [str2et_c --> STR2ET] The meaning of the integer <12>
could not be determined: '2010-10-03 05:44:<12> JD'
```

» 2010-10-03 05:44:12 is not a Julian date format: remove the trailing JD

- Using TDB and Julian date format, and inputting: 2.4404005000000000 D+06

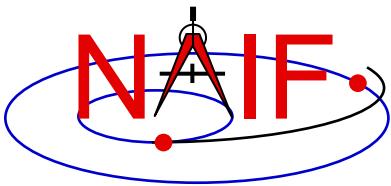
```
CSPICE_N0065: CSPICE.str2et: SPICE(UNPARSEDTIME): [str2et_c --> STR2ET] The meaning of the integer <06>
could not be determined: '2.4404005000000000 D+<06> JD TDB'
```

» “E” and “D” exponent notations are not allowed

- Using Spacecraft clock and Spacecraft clock string format, and inputting: 81138762:563

```
CSPICE_N0065: CSPICE.scencd: SPICE(KERNELVARNOTFOUND): [scencd_c --> SCENCD --> SCTIKS -->
SCTYPE --> SCLI01] SCLK_DATA_TYPE_0 not found. Did you load the SCLK kernel?
```

» The Spacecraft clock ID value is 0, which is not a valid clock ID. Either you did not load an archival kernel set (containing the mission SCLK kernel) or you did not manually load a SCLK kernel.



# Incorrect Use of or Interpretation of Time Tags

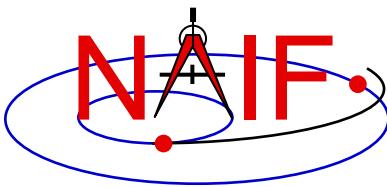
Navigation and Ancillary Information Facility

- Be aware that a time tag of this general form:

2019 DEC 18 13:21:53.261

that you've taken from elsewhere, or that you have produced using SPICE—what we call calendar format—could represent a time in either the UTC or the TDB (a.k.a. ET) time system. Be sure you know which time system you are dealing with!

- Refer to the "Time" tutorial for further information
- Spacecraft clock times occur as either a SCLK string or a double precision number of tics.
  - SCLK strings often appear like this: 53321876.214
  - The decimal character within this string is NOT a decimal point; rather it is simply a separator between the two parts of the clock string. See the "LSK and SCLK" tutorial for details.



# Time out of Bounds - 1

Navigation and Ancillary Information Facility

- WGC will display a SPICE error message when SPICE is able to detect a problem.

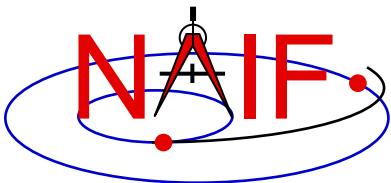
## Angular Size

Calculate the angular size of a target as seen from an observer.

CSPICE\_N0064: CSPICE.spkezr: SPICE(SPKINSUFFDATA): [spkezr\_c --> SPKEZR --> SPKEZ --> SPKGEO] Insufficient ephemeris data has been loaded to compute the state of 199 (MERCURY) relative to -236 (MESSENGER) at the ephemeris epoch 1995 MAY 03 00:01:01.185.

Kernel selection:	<input type="text" value="MESSENGER"/>
Target:	<input type="text" value="MERCURY"/>
Observer:	<input type="text" value="MESSENGER"/>

- In this example the user requested the position of Mercury relative to the MESSENGER spacecraft at a time outside the bounds of loaded SPK kernels. (In this case it was before MESSENGER was launched.)

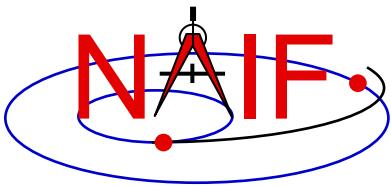


# Time out of Bounds - 2

Navigation and Ancillary Information Facility

- **Users involved with mission planning often ask to make a computation for a future time that is not within the bounds of loaded time-dependent kernels (most usually SPKs and CKs).**
  - Kernels available in SPICE archives cover only up to a date in the past. This can be seen using the Tool Tip displayed when you hover your cursor over the name of the archived kernel set.
    - » Example for the Cassini archive as of October 2017:

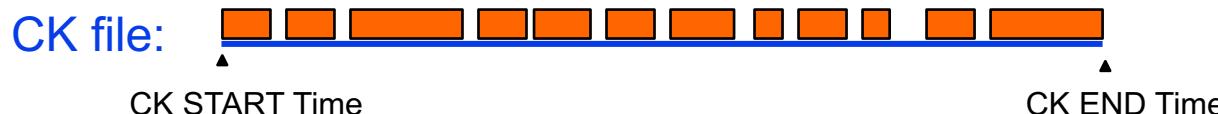
Archived Cassini kernels covering from 1997-10-15 to 2016-12-31
  - Some kernels available via MANUAL loading from NAIF's operations collections might contain “predict” data, but there is no guarantee, and it can be difficult for a person not familiar with the mission of interest to determine which kernel to load.



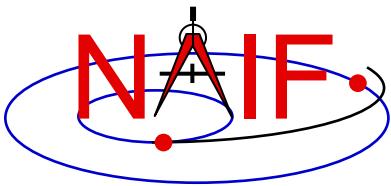
# Missing Data - 1

Navigation and Ancillary Information Facility

- If you try to make a series of calculations (e.g. over a time interval, or at each of a set of times) it could be that some of the calculations can be made while others cannot due to data gaps or otherwise missing data.
  - A gap in a CK is the most likely culprit. This is usually caused by sparse or missing downlinked spacecraft attitude telemetry.
  - The graphic below represents a CK file.
    - » The orange bars are called “**interpolation intervals**” during which orientation can be determined.
    - » The white spaces between the orange bars are gaps during which orientation cannot be determined.



- » See the next page for some options to deal with this.

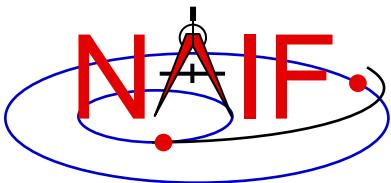


# Missing Data - 2

Navigation and Ancillary Information Facility

- When making “**Geometry Calculator**” computations you can control the action taken by WGC using the “Error handling” control found at the bottom of each “**Geometry Calculator**” web page.
  - The options are:
    - » Stop on error (the default setting)
    - » Report errors and continue (a more friendly but still safe option)
    - » Silently omit errors (dangerous; not recommended)
- However, when making “**Geometric Event Finder**” calculations you haven’t any options: WGC will issue an error message and discontinue the computation, showing no interval results at all. ☹

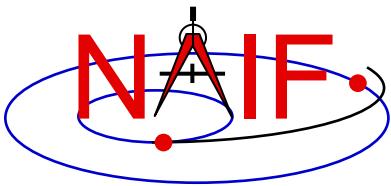
CSPICE\_N0064: CSPICE.gftfov: SPICE(NOFRAMECONNECT): [gftfov\_c --> GFTFOV --> GFMOVE --> ZZGFSOLV --> ZZGFFVST --> PXFORM --> REFCHG] At epoch 3.1753086618393E+08 TDB (2010 JAN 23 15:01:06.183 TDB), there is insufficient information available to transform from reference frame 1 (J2000) to reference frame -82360 (CASSINI\_ISS\_NAC). Frame CASSINI\_ISS\_NAC could be transformed to frame -82000 (CASSINI\_SC\_COORD). The latter is a CK frame; a CK file containing data for instrument or structure -82000 at the epoch shown above, as well as a corresponding SCLK kernel, must be loaded in order to use this frame. Failure to find required CK data could be due to one or more CK files not having been loaded, or to the epoch shown above lying within a coverage gap or beyond the coverage bounds of the loaded CK files. You can use CKBRIEF with the -dump option to display coverage intervals of a CK file.



# Incomplete Data

Navigation and Ancillary Information Facility

- **Computations involving ephemerides (trajectories) or orientations often require the chaining—the stringing together—of multiple “chunks” of data to obtain needed positional information.**
- **Kernels containing ALL the needed “chunks” of data to complete the chain must be loaded into WGC.**
  - This is rarely a problem when using a mission’s archive since much care is taken to make the archive complete.
  - But incomplete data is often a problem when **MANUALLY** loading individual kernels from a mission operations collection. The user may not understand what are the needed components of an ephemeris chain (SPK, FK) or an orientation chain (some of CK, FK, PCK).



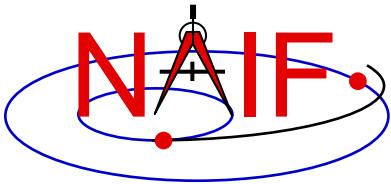
# Mixed Up Target and Observer

Navigation and Ancillary Information Facility

- A user wants to find the sub-observer point on the moon using the Lunar Reconnaissance Orbiter (LRO) spacecraft as the observer.
- S/he selects the “Sub-Observer Point” calculation, loads the LRO kernel set, and specifies the Target as “LRO,” the reference frame as “IAU\_MOON” and the observer as the “MOON.”

CSPICE\_N0065: CSPICE.subpnt: SPICE(INVALIDFRAME): [subpnt\_c --> SUBPNT] Reference frame IAU\_MOON is not centered at the target body LUNAR RECONNAISSANCE ORBITER. The ID code of the frame center is 301.

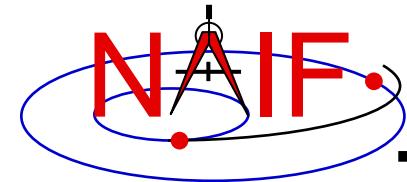
- S/he mixed up what should be the observer and the target. Interchange these two items and the calculation will work.



# Limitation – One at a Time

Navigation and Ancillary Information Facility

- **WebGeocalc executes only one computation at a time.**
  - Any computation requests received while one computation is in progress will be queued in the order received.
    - » A “queued” message will be displayed in your browser’s window.
    - » Each request will automatically execute once having reached the top of the queue.



---

Navigation and Ancillary Information Facility

# Digital Shape Kernel Subsystem (DSK)

January 2020

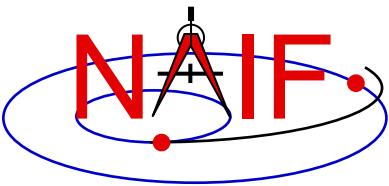


# Topics

---

Navigation and Ancillary Information Facility

- **DSK subsystem overview**
- **DSK shape representations**
- **N66 version of DSK subsystem**
- **DSK APIs and graphical depictions**
- **DSK API example**
- **DSK utility programs**
- **DSK concepts**
- **Writing and using DSK files**

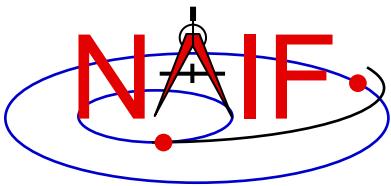


# DSK Subsystem Overview

---

Navigation and Ancillary Information Facility

- **The DSK subsystem**
  - enables SPICE-based applications to conveniently make use of high fidelity surface shape (topographic) data in geometry computations
  - serves as a format for transmission and archival of surface shape data
  - consists of SPICE software, DSK file format specifications, and documentation



# DSK Shape Representations

---

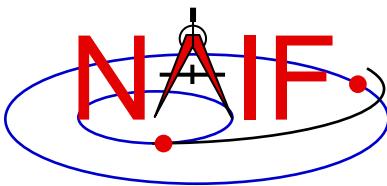
Navigation and Ancillary Information Facility

- The DSK subsystem handles two representations of shape data
  - Tessellated plate model (Type 2)



- Digital elevation model (development not yet finished) (Type 4)

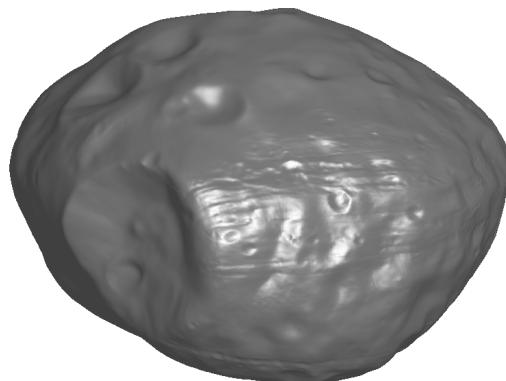




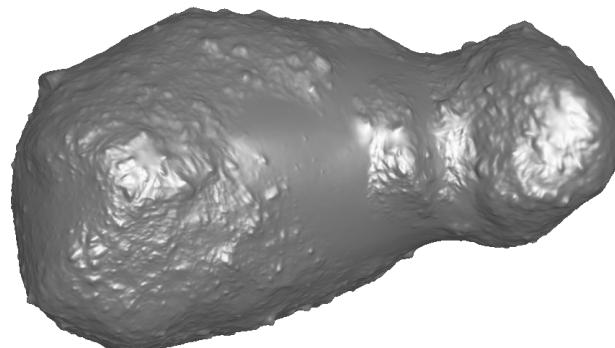
# Tessellated Plate Model – Type 2

Navigation and Ancillary Information Facility

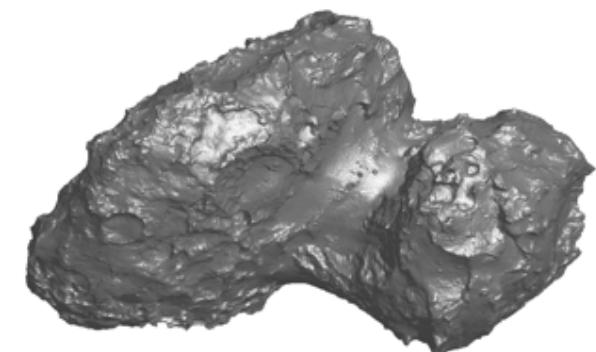
- The surface of the object is represented as a collection of triangular plates
- More flexible than digital elevation model: any arbitrary 3-D surface can be modeled
  - Surface could be a complicated shape with multiple surface points having the same latitude and longitude
    - » Examples: “dumbbell”-shaped asteroid, caves, arches
- Less efficient than digital elevation model (DSK Type 4) of similar resolution in terms of storage and computational speed



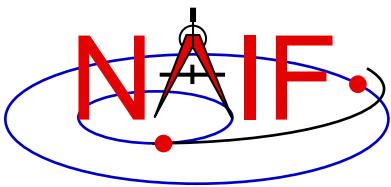
Phobos



Itokawa



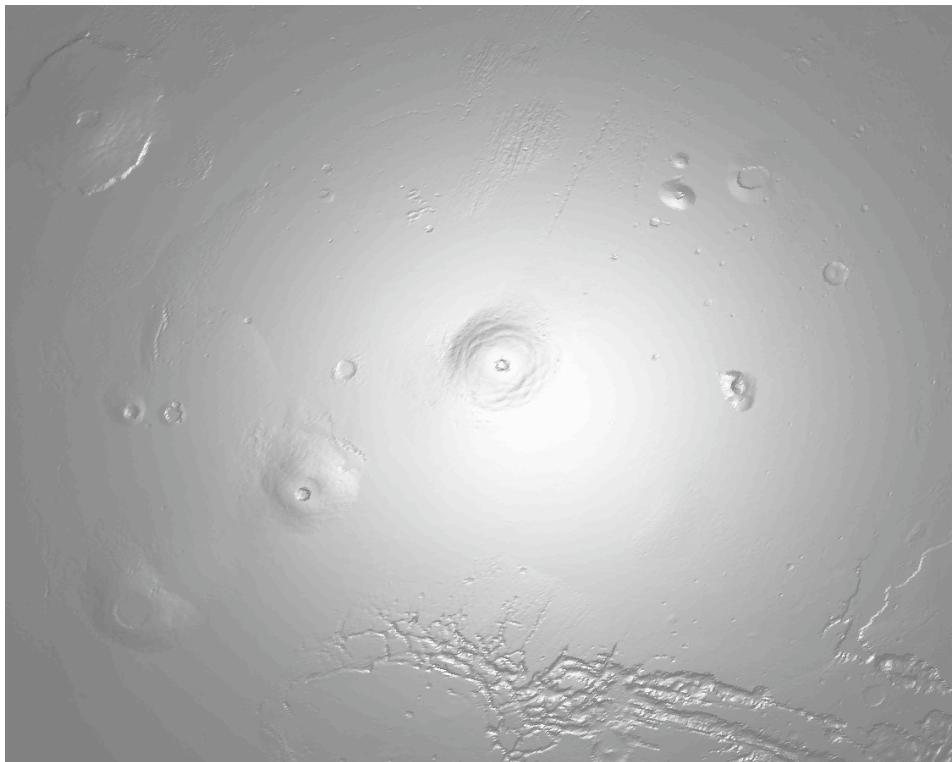
Churyumov-Gerasimenko

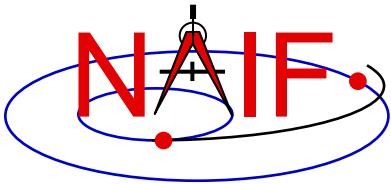


# Digital Elevation Model – Type 4

Navigation and Ancillary Information Facility

- **Maps longitude/latitude to “elevation”**
  - Elevation of a surface point can be defined as distance from the origin of a body-fixed reference frame or height above a reference ellipsoid
- **Example: rendering of a piece of DSK data created from MGS laser altimeter (MOLA) Mars DEM**



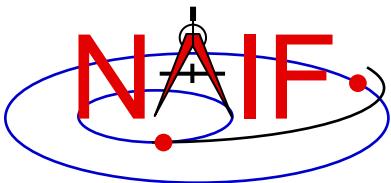


# N66 Toolkit with DSK

---

Navigation and Ancillary Information Facility

- **Supports only the tessellated plate model data type (Type 2 DSK)**
- **Support for Digital Elevation Model (DEM) (Type 4 DSK) will be added in a future Toolkit version**

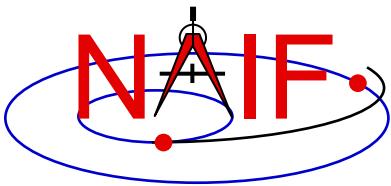


# Some DSK Features

---

Navigation and Ancillary Information Facility

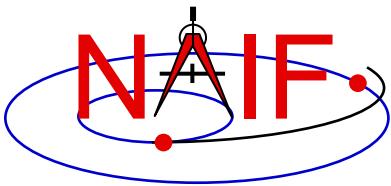
- **Supports multi-segment, multi-file DSK data sets**
  - Up to 5000 DSK files can be loaded simultaneously
  - Up to 10,000 DSK segments can be loaded simultaneously
- **Supports run-time data translation: big-endian DSK files can be read on little-endian platforms, and vice versa**
- **Pre-DSK era SPICE Toolkit geometry APIs will support DSK shape data, where applicable**



# APIs Available in N66 Toolkits -1

Navigation and Ancillary Information Facility

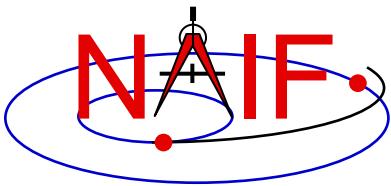
- **Kernel load/unload/info:**
  - FURNSH, UNLOAD, KCLEAR, KTOTAL, KINFO, KDATA
- **Geometry:**
  - Ray-surface intercept: SINCPT, DSKXV, DSKXSI
  - Sub-observer point: SUBPNT
  - Sub-solar point: SUBSLR
  - Illumination angles at surface point: ILLUMF, ILLUMG, ILUMIN
  - Longitude-latitude pairs to surface points: LATSRF
  - Find occultation state at a given time: OCCULT
  - Find occultation or transit of point target behind/across DSK shape: GFOCLT
  - Generate limb points: LIMBPT
  - Generate terminator points: TERMPT
  - Compute outward normal vector at surface point: SRFNRM



# APIs Available in N66 Toolkits -2

Navigation and Ancillary Information Facility

- **Low-level access:**
  - DLA segment traversal: DLABFS, DLABBS, DLAFNA, DLAFPA
  - Fetch type 2 counts/plates/vertices/normals: DSKZ02, DSKP02, DSKV02, DSKN02
  - Fetch all type 2 data structure contents: DSKI02, DSKD02
  - Fetch DSK segment descriptor: DSKGD
- **Plate utilities:**
  - PLTVOL, PLTAR, PLTEXP, PLTNP, PLTNRM
- **Create DSK files:**
  - DSKOPN, DSKW02, DSKCLS, DSKMI2, DSKRB2
- **Summary routines:**
  - DSKOBJ, DSKSRF
- **Surface name-code translation:**
  - SRFS2C, SRFSCC, SRFC2S, SRFCSS

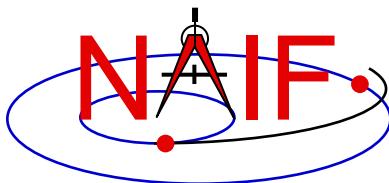


# Graphic Depictions

---

Navigation and Ancillary Information Facility

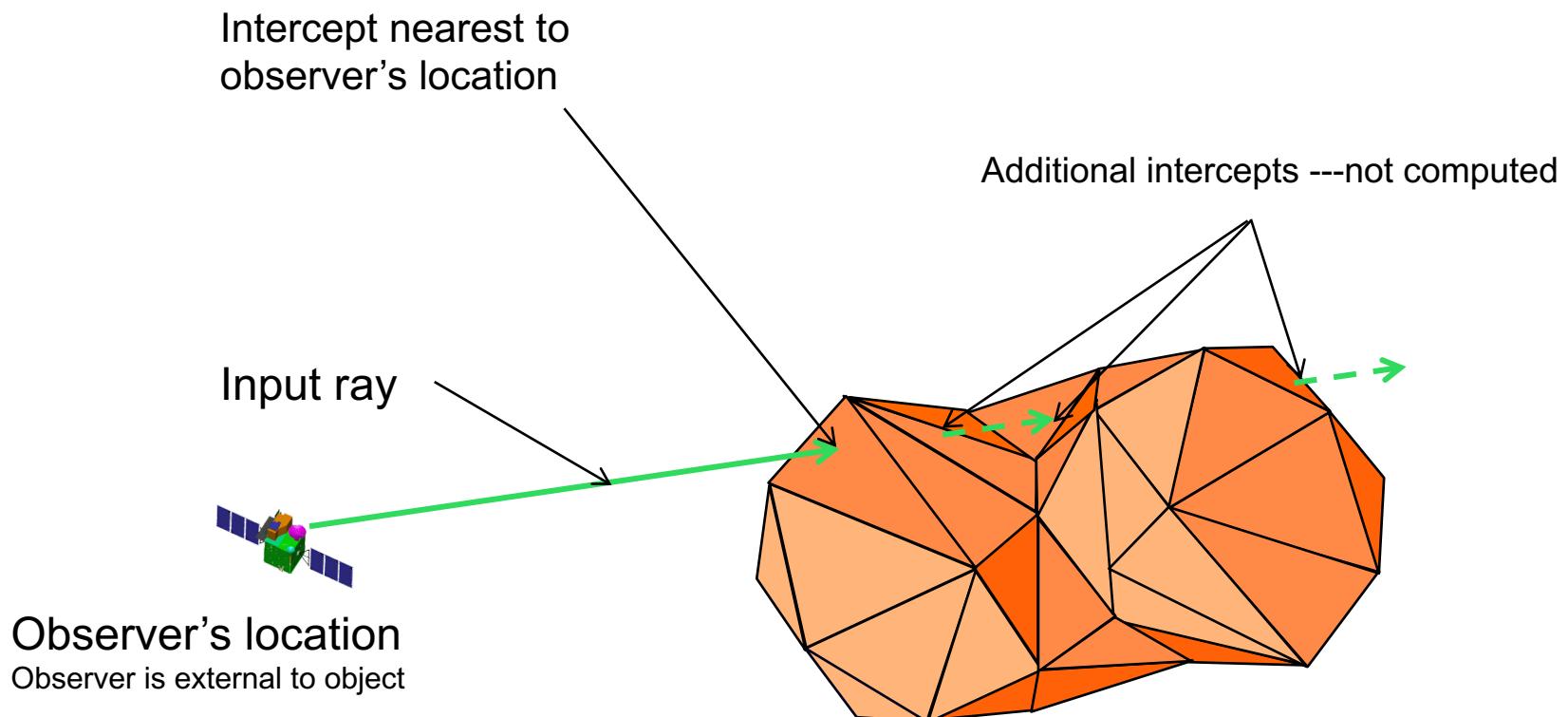
- In the next several charts we provide graphic depictions of the high-level APIs that should be of interest to many users

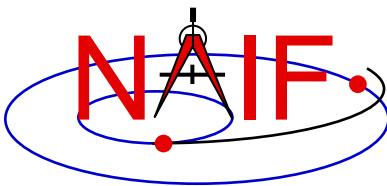


# Plate Model Surface Intercept

Navigation and Ancillary Information Facility

API: SINCPT

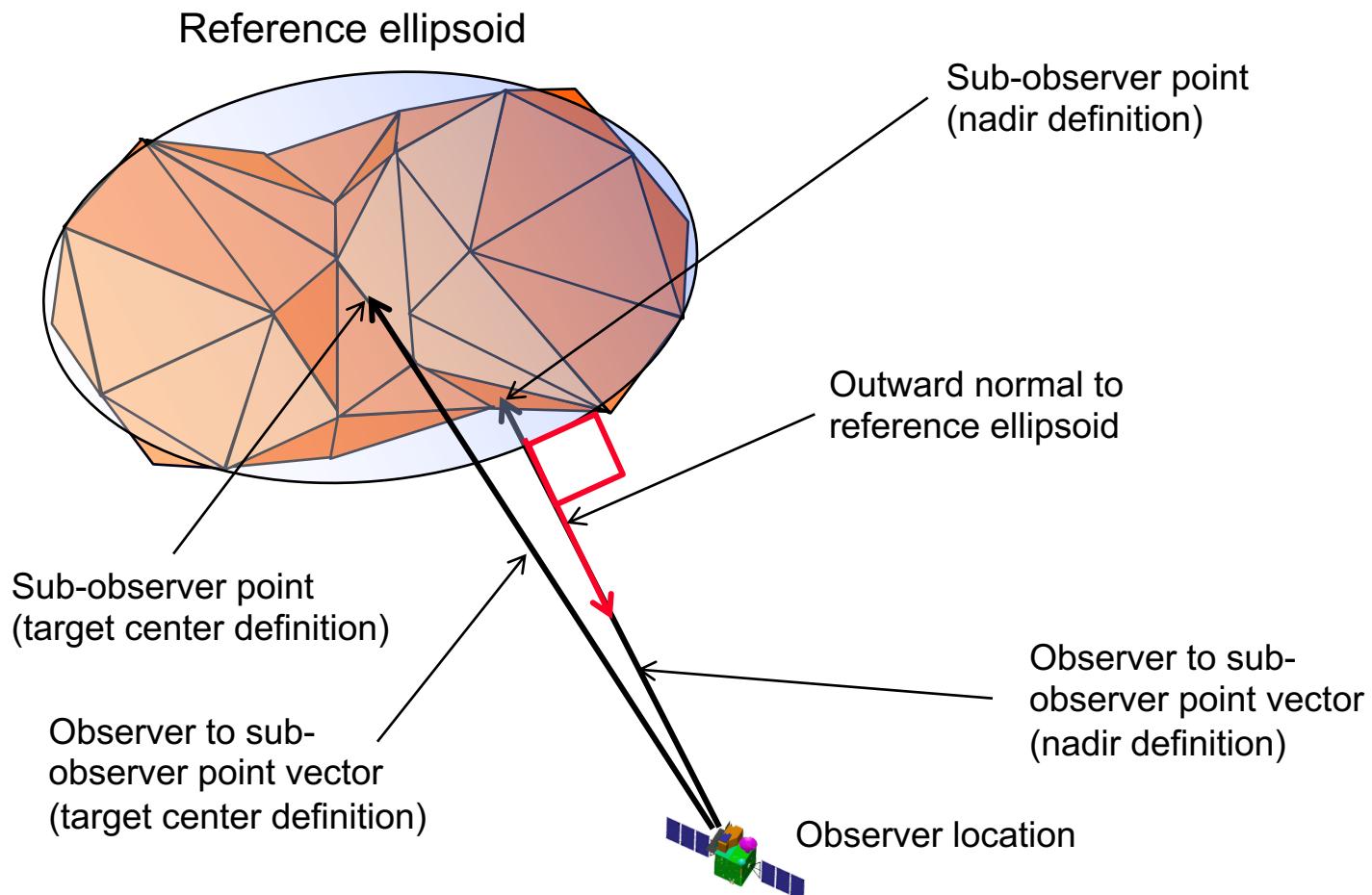


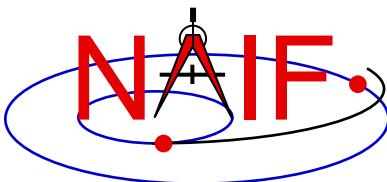


# Plate Model Sub-observer Point

Navigation and Ancillary Information Facility

API: SUBPNT

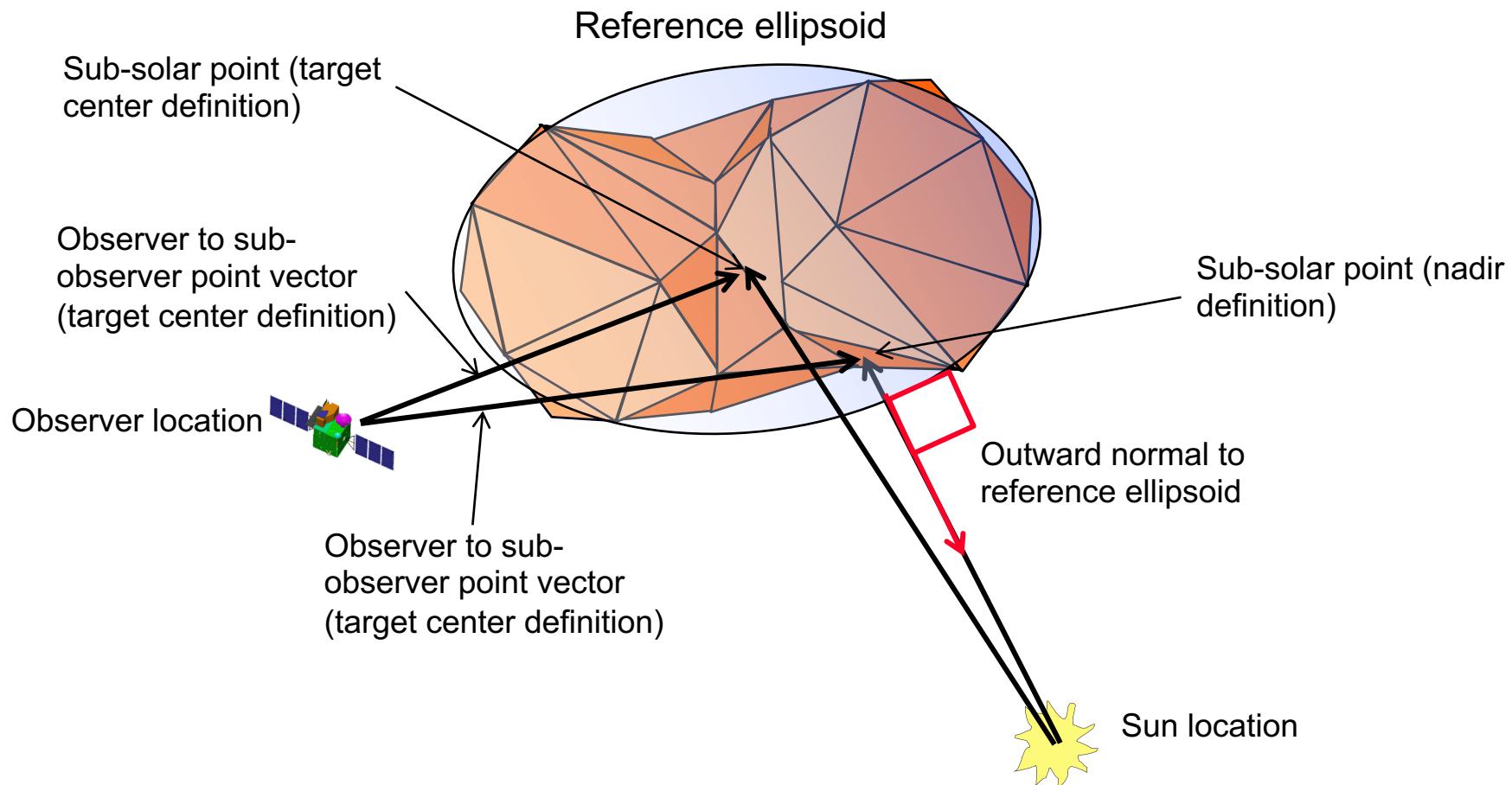


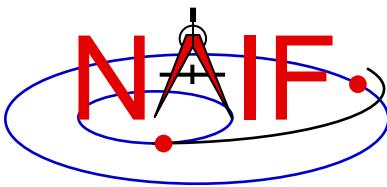


# Plate Model Sub-solar Point

Navigation and Ancillary Information Facility

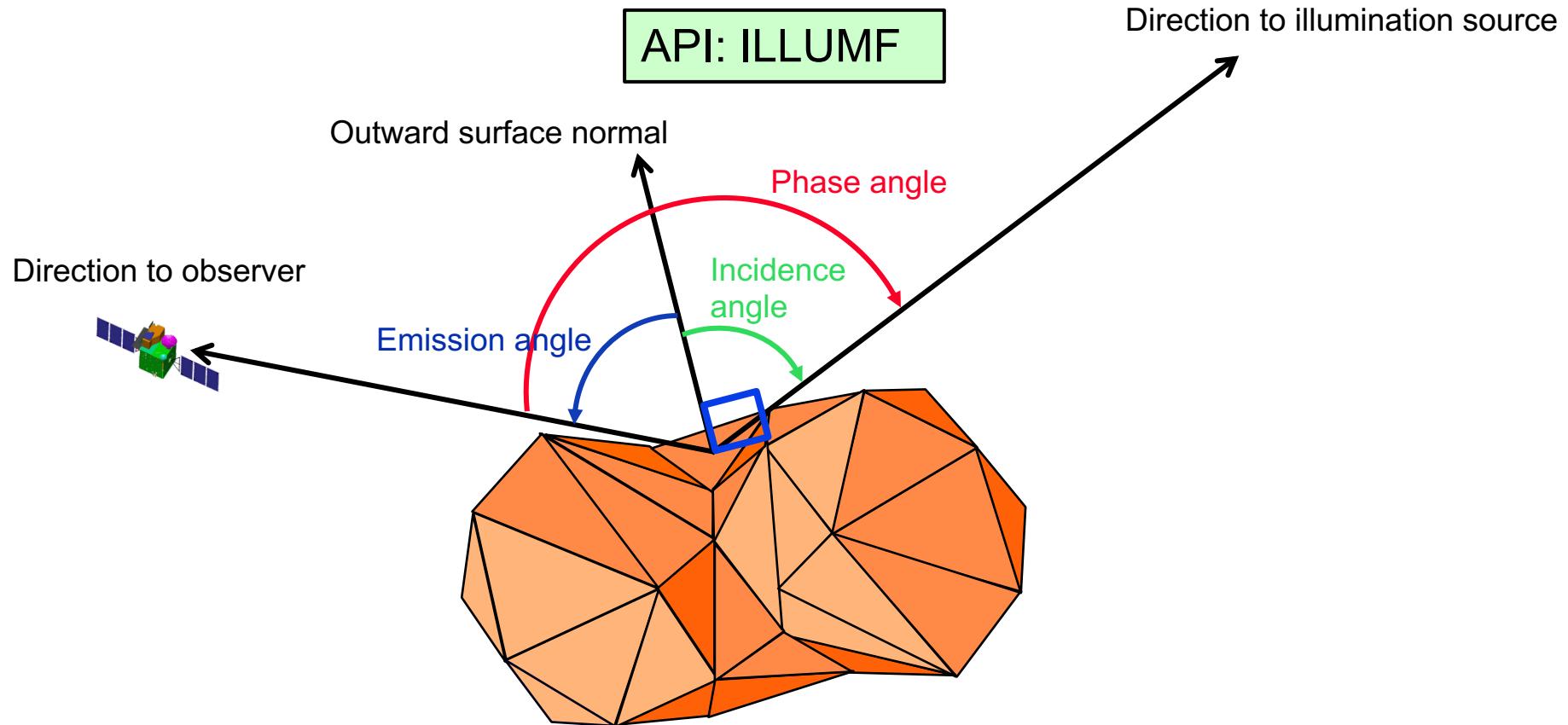
API: SUBSLR





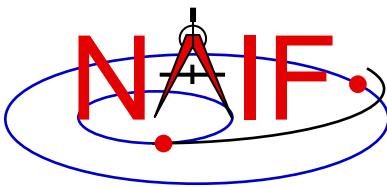
# Plate model illumination Angles

Navigation and Ancillary Information Facility



Also returned:

- target epoch (corrected for light time),
- observer visibility flag,
- illumination source visibility flag



# Plate Model Surface Point Grid

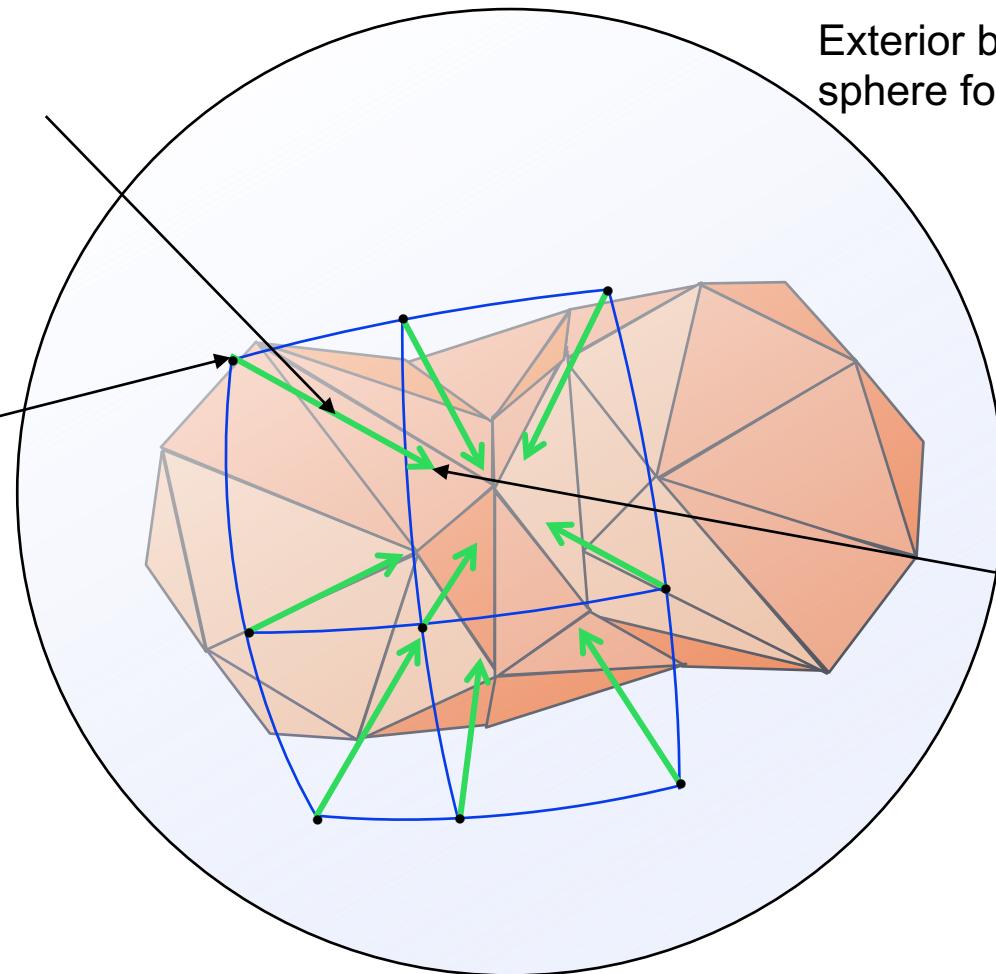
Navigation and Ancillary Information Facility

API: LATSRF

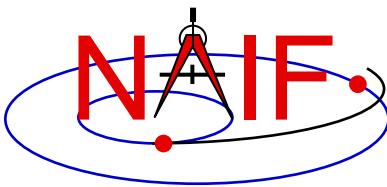
Ray emanating from sphere point, pointing toward center of body-fixed, body-centered reference frame

Point on bounding sphere, specified by planetocentric longitude and latitude, and by radius of exterior bounding sphere. This grid contains 9 such points.

Exterior bounding sphere for target object



Surface intercept point corresponding to point on bounding sphere: planetocentric longitude and latitude of intercept match those of the sphere point. An intercept is computed for each input sphere point.



# Plate Model Limb-1

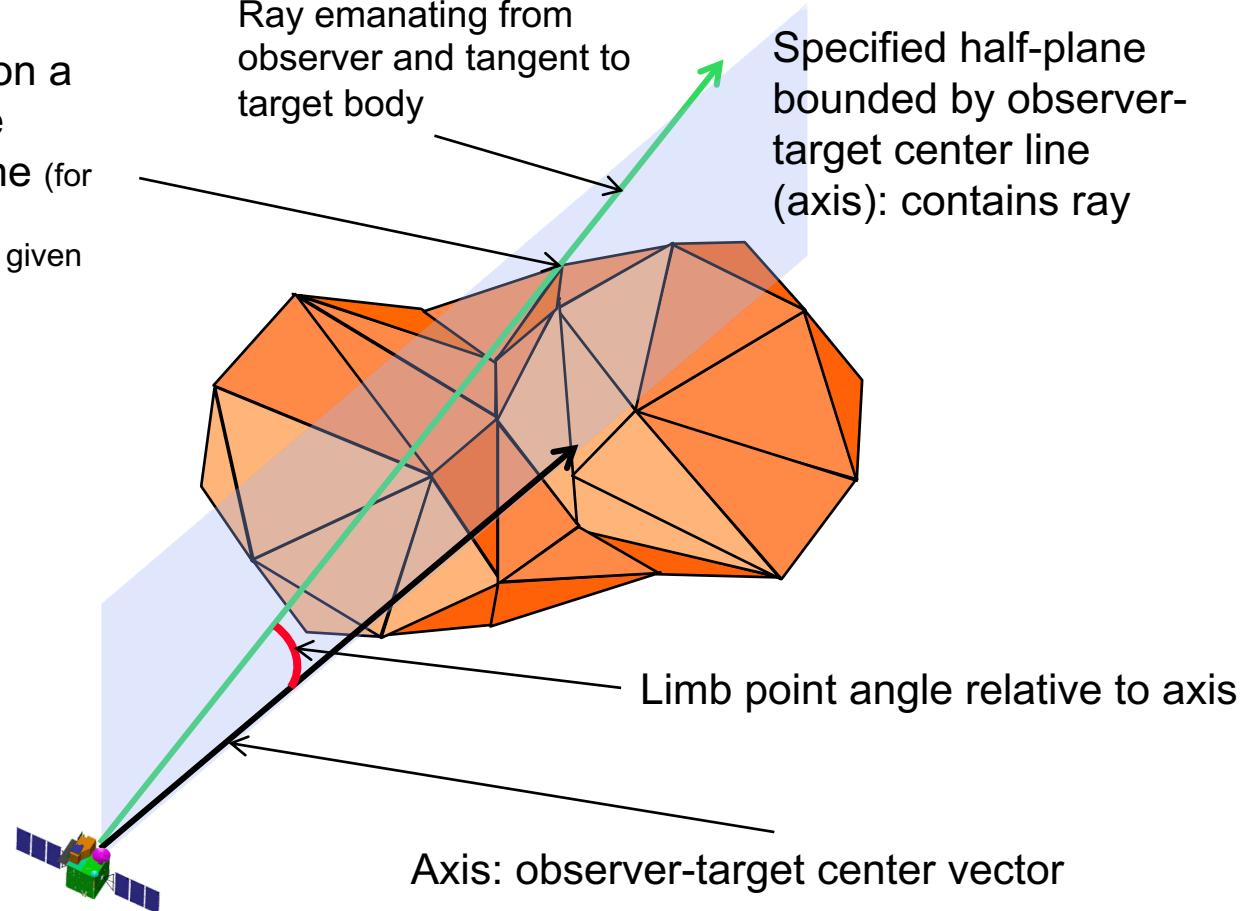
Navigation and Ancillary Information Facility

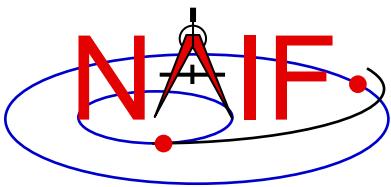
API: LIMBPT

Limb point---lies on a tangent ray in the selected half-plane (for some shapes, multiple tangents will exist for a given axis and half-plane)

Ray emanating from observer and tangent to target body

Specified half-plane bounded by observer-target center line (axis): contains ray

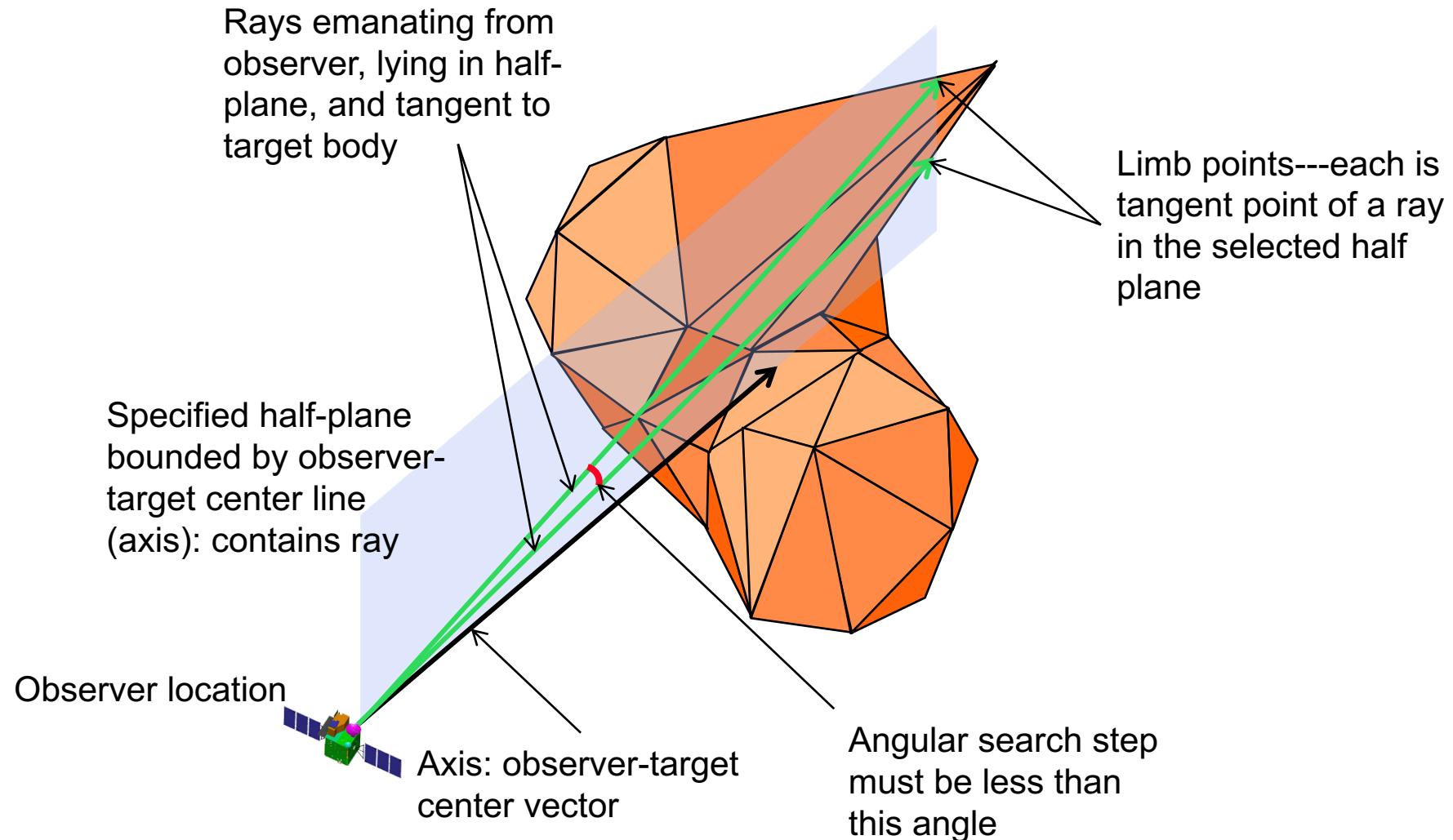


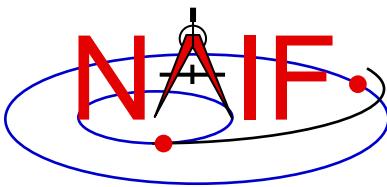


# Plate Model Limb-2

Navigation and Ancillary Information Facility

API: LIMBPT





# Plate Model Terminator-Umbral

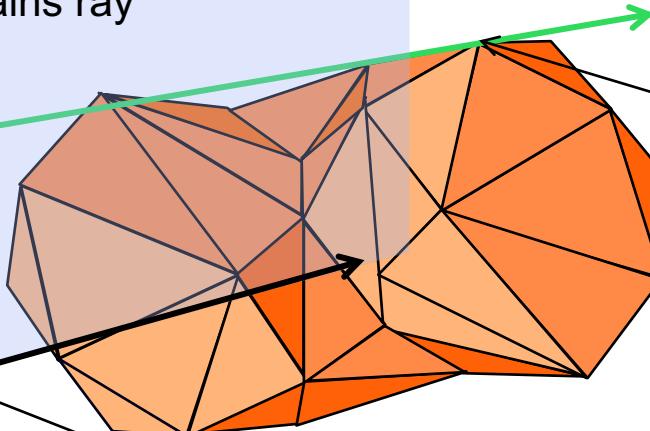
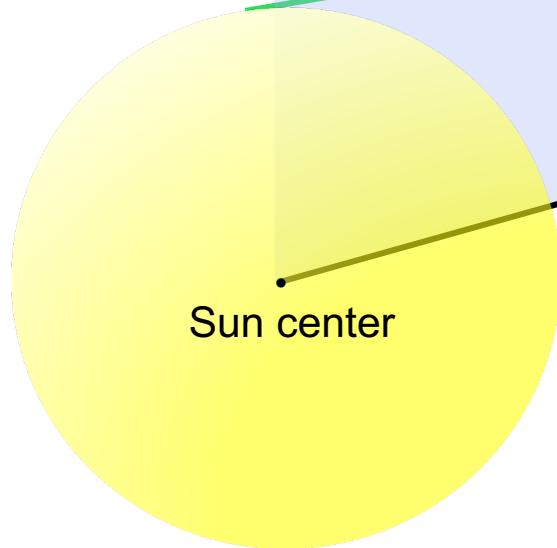
Navigation and Ancillary Information Facility

API: TERMPT

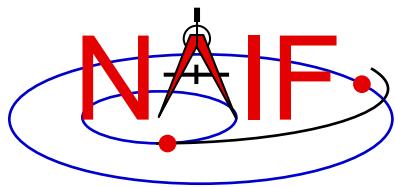
Ray tangent to sun  
and target body

Specified half-plane  
bounded by sun center-  
target center line (axis):  
contains ray

Umbral terminator  
point: lies on a tangent  
ray in the specified  
**half-plane** (for some  
shapes, multiple tangents will  
exist for a given axis and half-  
plane). Terminator point  
and ray vertex are on  
same side of axis.



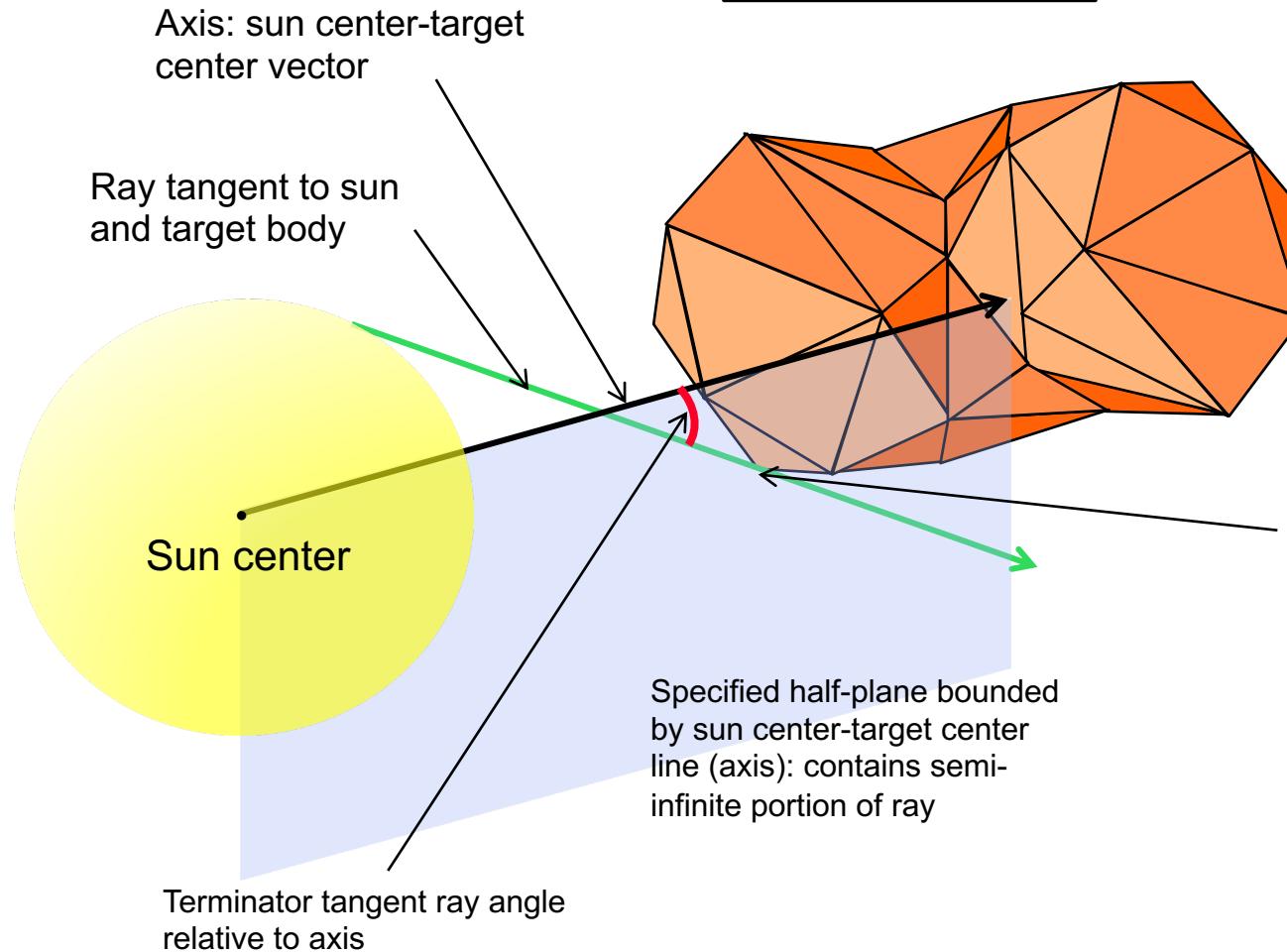
Axis: sun center-target center vector

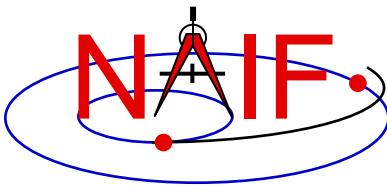


# Plate Model Terminator-Penumbral

Navigation and Ancillary Information Facility

API: TERMPT





# Example of API Using DSK - 1

Navigation and Ancillary Information Facility

- Find ray intercept point on target surface:
  - CALL SINCPT ( **METHOD**, TARGET, ET, FIXREF, ABCORR, OBSRVR, DREF, DVEC, SPOINT, TRGEPC, SRFVEC, FOUND)
  - SINCPT is a high-level SPICE API.
  - The input string argument **METHOD** indicates the surface model to use.
    - » To model the target body shape using an ellipsoid, set METHOD to ‘ellipsoid’
    - » To model the target body shape using DSK data, set METHOD to one of the forms
      - ‘DSK/UNPRIORITYZED’
        - If all DSK segments for the body designated by TARGET are applicable
      - ‘DSK/UNPRIORITYZED/SURFACES = <surface name or ID 1>, ...’
        - If only DSK segments for the specified surfaces associated with the body designated by TARGET are applicable
    - » For the DSK case, the keyword UNPRIORITYZED is currently required. This keyword indicates that no applicable segment can mask another.

Fortran code example

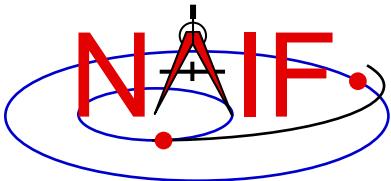


# Example of API Using DSK - 2

---

Navigation and Ancillary Information Facility

- » Other inputs: target body name, epoch, body-fixed reference frame, aberration correction, observer name, reference frame for direction vector, direction vector.
- » Outputs: ray-surface intercept in Cartesian coordinates, expressed in the body-fixed frame associated with the target---evaluated at the optionally light-time corrected epoch TRGEPC, TRGEPC itself, observer-to-intercept vector expressed in body-fixed frame, and found flag indicating whether intercept exists.

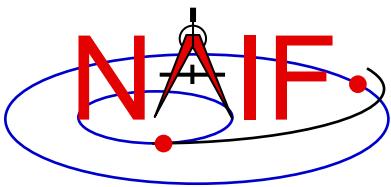


# DSK Utility Programs

---

Navigation and Ancillary Information Facility

- **Create DSK files: MKDSK**
  - Creates a DSK file containing a single type 2 segment
- **Export DSK data to text format files: DSKEXP**
  - Writes data from type 2 DSK segments to one or more text files
  - Supports simple output formats such as obj
- **Summarize DSK files: DSKBRIEF**
- **Merge DSK files: DLACAT**
  - Concatenates segments from multiple DSK files into a single DSK file
- **Transform binary architecture of DSK file: TOXFR, TOBIN, BINGO (BINGO not part of standard SPICE Toolkit)**
- **Read/write comment area: COMMNT**



# DSK Concepts-1

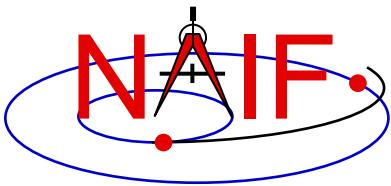
Navigation and Ancillary Information Facility

- **Surface**

- "Surface" is a second identifier, in addition to the central body
  - » A "surface" has a name and an integer ID code
    - Surfaces occupy a name space distinct from that of bodies
    - APIs are provided for surface name/ID conversion
  - Used to distinguish different versions of data for a given body
    - » Allows use of different versions without loading and unloading kernels
      - High-frequency kernel loading and unloading is too inefficient for DSK applications

- **Data class**

- Data class is a "hook" to differentiate kinds of data for different applications
  - » Distinct from concept of "data type"
- Existing classes indicate geometric characteristics of surface data
  - » Class 1: shape is single-valued function of domain coordinates.  
Example, for latitudinal coordinates:
    - Every ray emanating from the origin of the body-fixed reference frame associated with the body passes through the surface once
    - Such surfaces cannot have features such as cliffs or caves
    - DEMs can represent class 1 surfaces
  - » Class 2: arbitrary shape
    - Not required to be convex, closed, or connected
    - Plate models are the only DSK data type that can be used for class 2 surfaces

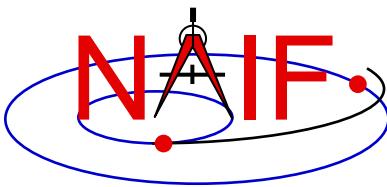


# DSK Concepts-2

---

Navigation and Ancillary Information Facility

- **Kernel priority**
  - Unlike SPK, CK, and binary PCK files, the concept of segment “priority” does not apply to all DSK applications
    - » Not applicable to data sets including segments of class 2
      - Concept simply doesn’t make sense when multiple heights can correspond to a single longitude/latitude coordinate pair
    - » Can apply to data sets containing only class 1 segments
- **Coordinate systems**
  - Associated with segments
    - » Segment coverage is described in terms of a coordinate system associated with that segment
  - Can be any of
    - » Planetocentric (latitudinal)
    - » Planetodetic
    - » Cartesian
- **Segment coverage**
  - The spatial “coverage” of a segment is a region of space within which the segment provides valid surface data
    - » Characterized by three coordinate ranges
      - For example: min, max longitude; min, max latitude; min, max radius
    - » “Padding” data may be provided outside of a segment’s coverage region



# Writing Shape and Orientation Kernels

Navigation and Ancillary Information Facility

LAT/LON and height  
above ellipsoid or  
distance from center  
of frame (post N66)

Lists of plate  
model vertices  
and associated  
plates

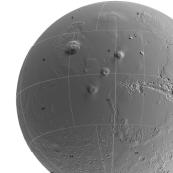
Axes dimensions  
for tri-axial  
ellipsoid

Some source of  
rotation state  
information (pole  
RA/DEC and  
prime meridian  
location)

MKDSK  
Program  
or SPICE  
Routines  
(SPICE Toolkit)

Text editor  
(Usually done by  
NAIF)

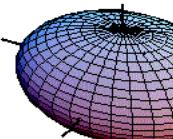
Text editor  
(Usually done by  
NAIF)



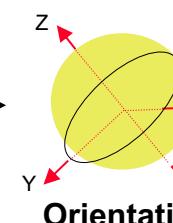
Digital Terrain  
Shape Model



Tessellated Plates  
Shape Model



Triaxial Ellipsoid  
Shape Model



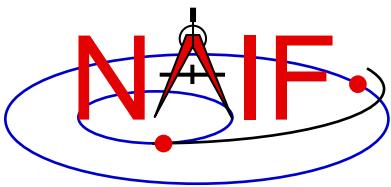
Orientation

DSK

Digital  
shape kernel

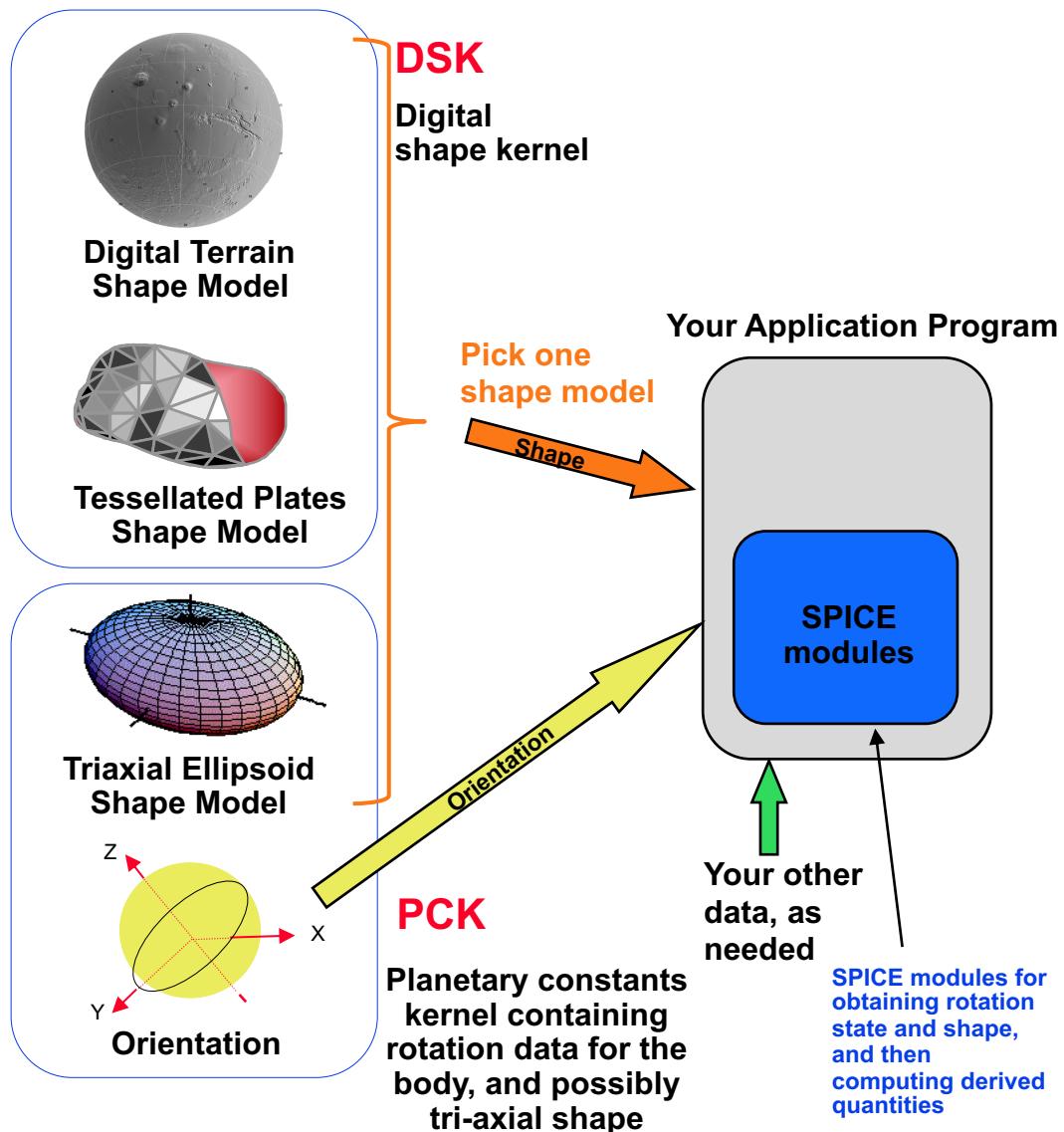
PCK

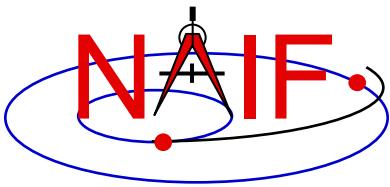
Planetary constants  
kernel containing  
rotation data for the  
body, and possibly  
tri-axial shape



# Using Shape and Orientation Kernels

Navigation and Ancillary Information Facility



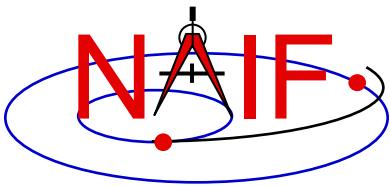


---

Navigation and Ancillary Information Facility

# Writing an Icy (IDL) Based Program

January 2020

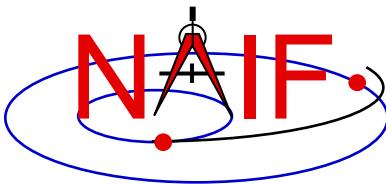


# Viewing This Tutorial

---

Navigation and Ancillary Information Facility

**Undefined variables are displayed in red**  
**Results are displayed in blue**



# Introduction

---

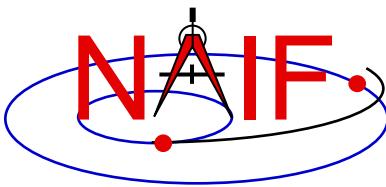
## Navigation and Ancillary Information Facility

First, let's go over the important steps in the process of writing an Icy-based program and putting it to work:

- Understand the geometry problem.
- Identify the set of SPICE kernels that contain the data needed to perform the computation.
- Select the SPICE APIs needed to compute the quantities of interest.
- Write and compile the program.
- Get actual kernel files and verify that they contain the data needed to support the computation for the time(s) of interest.
- Run the program.

To illustrate these steps, let's write a program that computes the apparent intersection of the boresight ray of a given CASSINI science instrument with the surface of a given Saturnian satellite. The program will compute:

- Planetocentric and planetodetic (geodetic) latitudes and longitudes of the intercept point.
- Range from spacecraft to intercept point.
- Illumination angles (phase, solar incidence, and emission) at the intercept point.



# Observation geometry

Navigation and Ancillary Information Facility

We want the boresight intercept on the surface, range from s/c to intercept, and illumination angles at the intercept point.

When? **TIME (UTC, TDB or TT)**

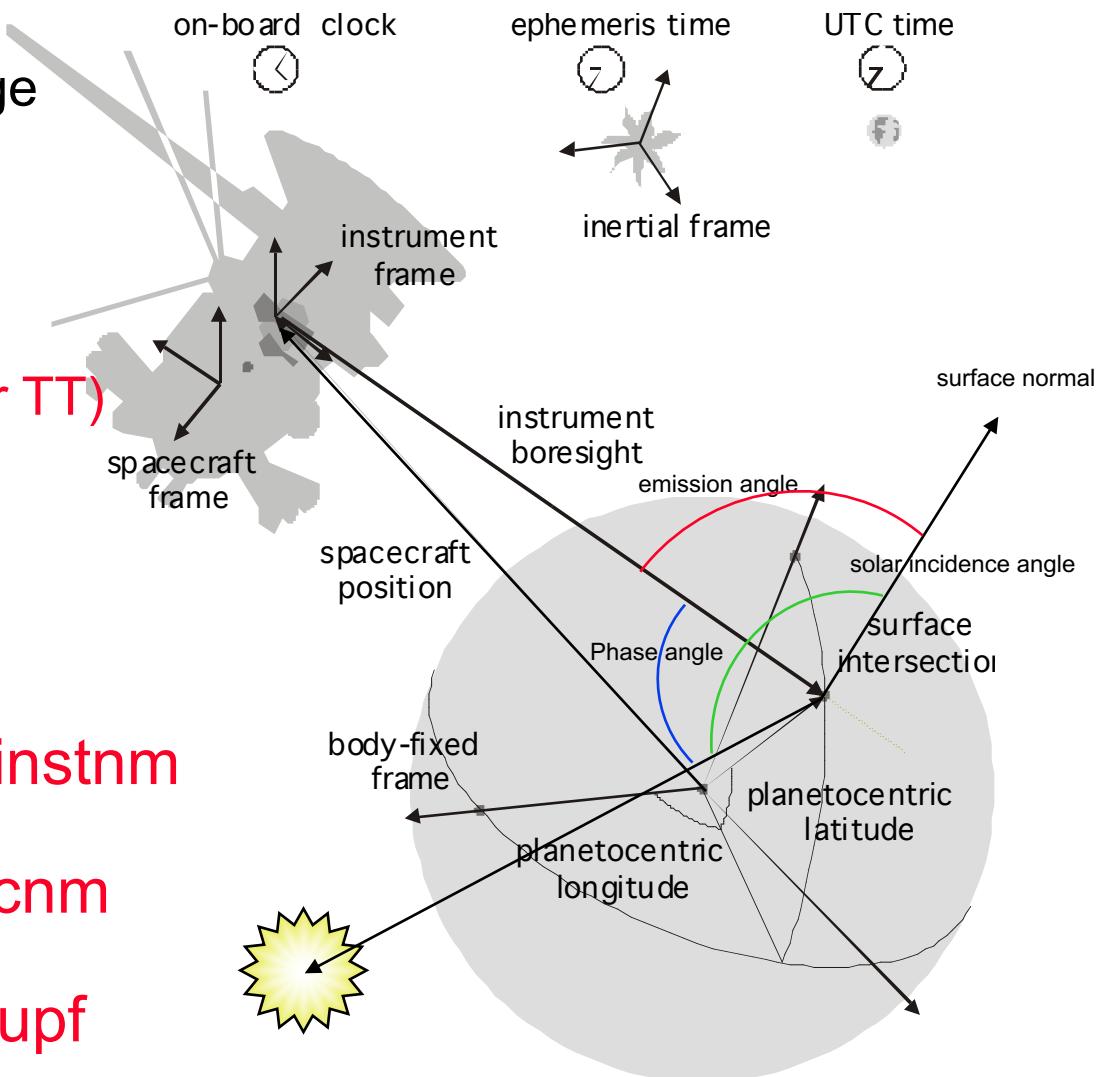
On what object? **satnm**

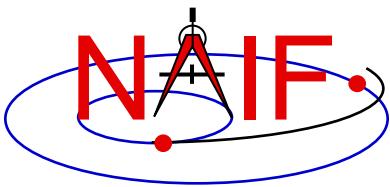
In what frame? **fixref**

For which instrument? **instnm**

For what spacecraft? **scnm**

Using what model? **setupf**





# Needed Data

Navigation and Ancillary Information Facility

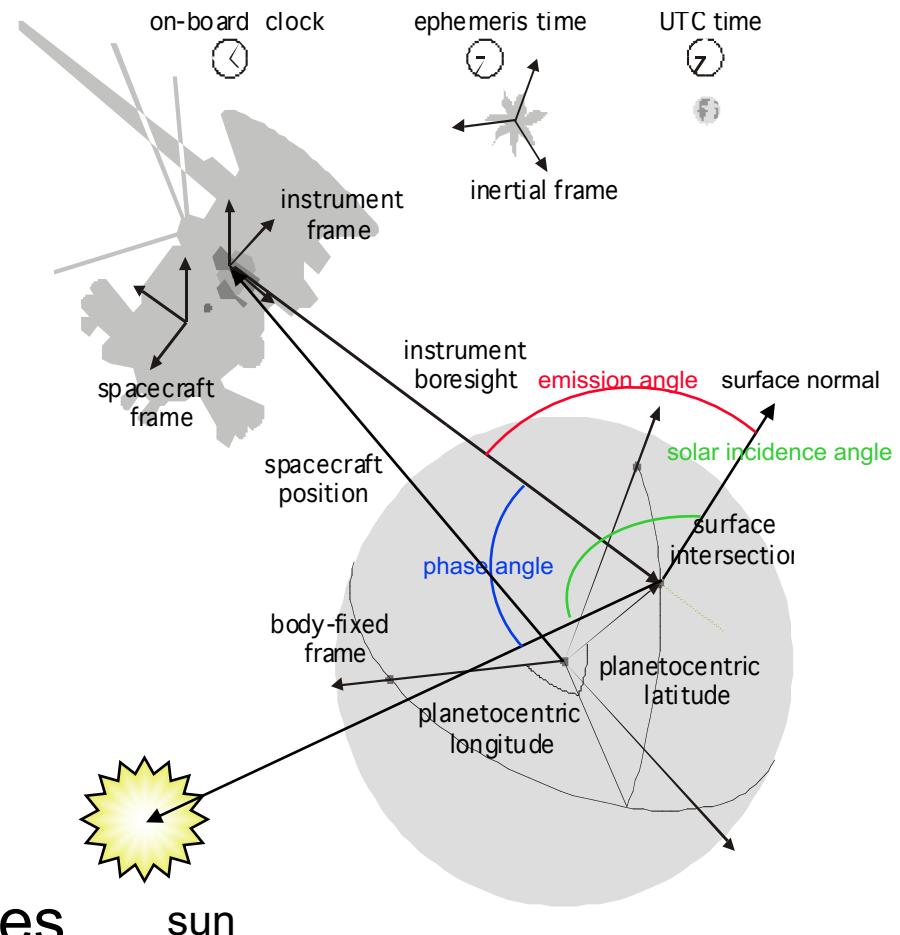
Time transformation kernels

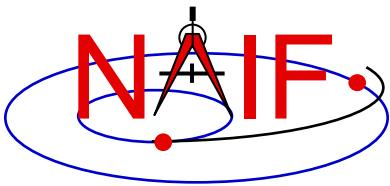
Orientation models

Instrument descriptions

Shapes of satellites, planets

Ephemerides for spacecraft,  
Saturn barycenter and satellites.





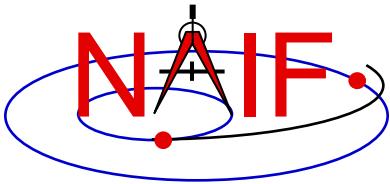
# Which Kernels are Needed?

Navigation and Ancillary Information Facility

Data required to compute vectors, rotations and other parameters shown in the picture are stored in the SPICE kernels listed below.

Note: these kernels have been selected to support this presentation; they should not be assumed to be appropriate for user applications.

Parameter	Kernel Type	File name
time conversions	generic LSK	naif0009.tls
satellite orientation	CASSINI SCLK	cas00084.tsc
satellite shape	CASSINI PCK	cpck05Mar2004.tpc
satellite position	CASSINI PCK	cpck05Mar2004.tpc
	planet/sat	
planet barycenter position	ephemeris SPK	020514_SE_SAT105.bsp
spacecraft position	planet SPK	981005_PLTEPH-DE405S.bsp
spacecraft orientation	spacecraft SPK	030201AP_SK_SM546_T45.bsp
instrument alignment	spacecraft CK	04135_04171pc_psiv2.bc
instrument boresight	CASSINI FK	cas_v37.tf
	Instrument IK	cas_iss_v09.ti



# Load kernels

---

Navigation and Ancillary Information Facility

The easiest and most flexible way to make these kernels available to the program is via `cspice_furnsh`. For this example we make a setup file (also called a “metakernel” or “furnsh kernel”) containing a list of kernels to be loaded:

Note: these kernels have been selected to support this presentation; they should not be assumed to be appropriate for user applications.

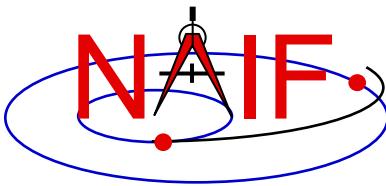
```
\begindata
```

```
KERNELS_TO_LOAD = ('naif0009.tls', 'cas00084.tsc', 'cpck05Mar2004.tpc',
                    '020514_SE_SAT105.bsp', '981005_PLTEPH-DE405S.bsp',
                    '030201AP_SK_SM546_T45.bsp', '04135_04171pc_psiv2.bc',
                    'cas_v37.tf', 'cas_iss_v09.ti')
```

```
\begintext
```

and we make the program prompt for the name of this setup file:

```
read, setupf, PROMPT='Enter setup file name > '
cspice_furnsh, setupf
```



# Programming Solution

---

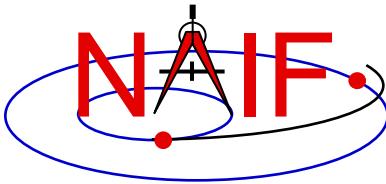
Navigation and Ancillary Information Facility

- Prompt for setup file (“metakernel”) name; load kernels specified via setup file. (Done on previous chart.)
- Prompt for user inputs required to completely specify problem. Obtain further inputs required by geometry routines via Icy calls.
- Compute the intersection of the boresight direction ray with the surface of the satellite, presented as a triaxial ellipsoid.

If there is an intersection,

- Convert Cartesian coordinates of the intersection point to planetocentric latitudinal and planetodetic coordinates
- Compute spacecraft-to-intercept point range
- Find the illumination angles (phase, solar incidence, and emission) at the intercept point
- Display the results.

We discuss the geometric portion of the problem first.



# Compute surface intercept

Navigation and Ancillary Information Facility

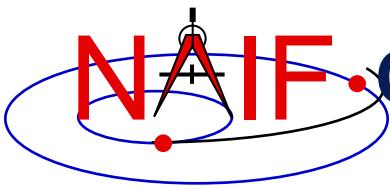
Compute the intercept point (`point`) of the boresight vector (`insite`) specified in the instrument frame (`iframe`) of the instrument mounted on the spacecraft (`scnm`) with the surface of the satellite (`satnm`) at the TDB time of interest (`et`) in the satellite's body-fixed frame (`fixref`). This call also returns the light-time corrected epoch at the intercept point (`trgepc`), the spacecraft-to-intercept point vector (`srfvec`), and a flag indicating whether the intercept was found (`found`). We use "converged Newtonian" light time plus stellar aberration corrections to produce the most accurate surface intercept solution possible. We model the surface of the satellite as an ellipsoid.

```
cspice_sincpt, 'Ellipsoid', satnm, et, fixref, 'CN+S', scnm, iframe, $  
    insite, point, trgepc, srfvec, found
```

The range we want is obtained from the outputs of `cspice_sincpt`. These outputs are defined only if a surface intercept is found. If `found` is true, the spacecraft-to-surface intercept range is the norm of the output argument `srfvec`. Units are km. We use the Icy function `cspice_vnorm` to obtain the norm:

```
cspice_vnorm( srfvec )
```

We'll write out the range data along with the other program results.



# Compute Lat/Lon and Illumination Angles

Navigation and Ancillary Information Facility

Compute the planetocentric latitude (`pclat`) and longitude (`pclon`), as well as the planetodetic latitude (`pdlat`) and longitude (`pdlon`) of the intersection point.

```
if ( found ) then begin
    cspice_reclat, point, r, pclon, pclat

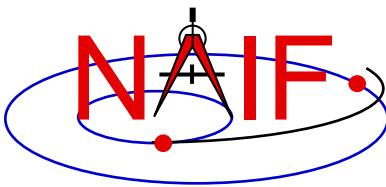
    ; Let re, rp, and f be the satellite's longer equatorial
    ; radius, polar radius, and flattening factor.

    re   =  radii[0]
    rp   =  radii[2]
    f    =  ( re - rp ) / re;

    cspice_recgeo, point, re, f, pdlon, pdlat, alt
```

The illumination angles we want are the outputs of `cspice_ilumin`. Units are radians.

```
cspice_ilumin, 'Ellipsoid', satnm, et, fixref, 'CN+S', scnm, $
    point, trgepc, srfvec, phase, solar, emissn
```



# Geometry Calculations: Summary

Navigation and Ancillary Information Facility

```
; Compute the boresight ray intersection with the surface of the
; target body.

cspice_sincpt, 'Ellipsoid', satnm, et, fixref, 'CN+S', scnm, $
    iframe, insite, point, trgepc, srfvec, found

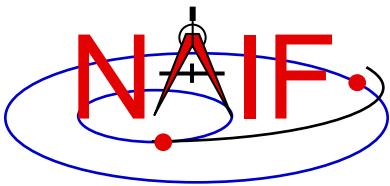
; If an intercept is found, compute planetocentric and planetodetic
; latitude and longitude of the point.

if ( found ) then begin
    cspice_reclat, point, r, pclon, pclat
    ; Let re, rp, and f be the satellite's longer equatorial
    ; radius, polar radius, and flattening factor.
    re = radii[0]
    rp = radii[2]
    f = ( re - rp ) / re;
    cspice_recgeo, point, re, f, pdlon, pdlat, alt

    ; Compute illumination angles at the surface point.

    cspice_ilumin, 'Ellipsoid', satnm, et, fixref, 'CN+S', scnm, $
        point, trgepc, srfvec, phase, solar, emissn
    ...
endif else begin
    ...

```



# Get inputs - 1

---

Navigation and Ancillary Information Facility

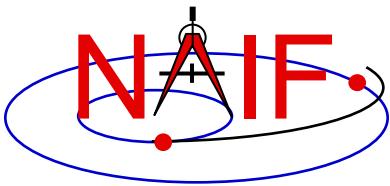
The code above used quite a few inputs that we don't have yet:

- TDB epoch of interest (`et`);
- satellite and s/c names (`satnm`, `scnm`);
- satellite body-fixed frame name (`fixref`);
- satellite ellipsoid radii (`radii`);
- instrument fixed frame name (`iframe`);
- instrument boresight vector in the instrument frame (`insite`);

Some of these values are user inputs; others can be obtained via Icy calls once the required kernels have been loaded.

Let's prompt for the satellite name (`satnm`), satellite frame name (`fixref`), spacecraft name (`scnm`), instrument name (`instnm`) and time of interest (`time`):

```
read, satnm , PROMPT='Enter satellite name > '
read, fixref , PROMPT='Enter satellite frame > '
read, scnm , PROMPT='Enter spacecraft name > '
read, instnm , PROMPT='Enter instrument name > '
read, time , PROMPT='Enter time > '
```



# Get Inputs - 2

---

Navigation and Ancillary Information Facility

Then we can get the rest of the inputs from Icy calls:

To get the TDB epoch (`et`) from the user-supplied time string (which may refer to the UTC, TDB or TT time systems):

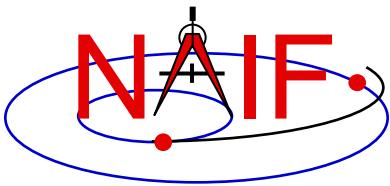
```
cspice_str2et, time, et
```

To get the satellite's ellipsoid radii (`radii`):

```
cspice_bodvrd, satnm, "RADII", 3, radii
```

To get the instrument boresight direction (`insite`) and the name of the instrument frame (`iframe`) in which it is defined:

```
cspice_bodn2c, instnm, instid, found
if ( NOT found ) then begin
    print, "Unable to determine ID for instrument: ", instnm
    return
endif
cspice_getfov, instid, ROOM, shape, iframe, insite, bndry
```



# Getting inputs: summary

Navigation and Ancillary Information Facility

```
; ; Prompt for the user-supplied inputs for our program
read, setupf, PROMPT='Enter setup file name > '
cspice_furnsh, setupf

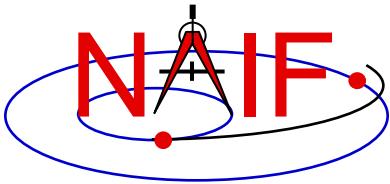
read, satnm , PROMPT='Enter satellite name > '
read, fixref, PROMPT='Enter satellite frame > '
read, scnm , PROMPT='Enter spacecraft name > '
read, instnm, PROMPT='Enter instrument name > '
read, time , PROMPT='Enter time > '

; ; Get the epoch corresponding to the input time:
cspice_str2et, time, et

; ; Get the radii of the satellite.
cspice_bodvrd, satnm, "RADII", 3, radii

; ; Get the instrument boresight and frame name.

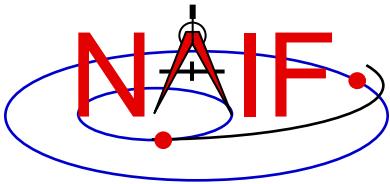
cspice_bodn2c, instnm, instid, found
if ( NOT found ) then begin
    print, "Unable to determine ID for instrument: ", instnm
return endif
cspice_getfov, instid, ROOM, shape, iframe, insite, bndry
```



# Display results

Navigation and Ancillary Information Facility

```
; ; Display results. Convert angles from radians to degrees for output.  
print  
print, 'Intercept planetocentric longitude      (deg): ', $  
      cspice_dpr()*pclon  
print, 'Intercept planetocentric latitude       (deg): ', $  
      cspice_dpr()*pclat  
print, 'Intercept planetodetic longitude        (deg): ', $  
      cspice_dpr()*pdlon  
print, 'Intercept planetodetic latitude         (deg): ', $  
      cspice_dpr()*pdlat  
print, 'Range from spacecraft to intercept point (km): ', $  
      cspice_vnorm(srfvec)  
print, 'Intercept phase angle                  (deg): ', $  
      cspice_dpr()*phase  
print, 'Intercept solar incidence angle        (deg): ', $  
      cspice_dpr()*solar  
print, 'Intercept emission angle               (deg): ', $  
      cspice_dpr()*emissn  
  
endif else begin  
  print, 'No intercept point found at ' + time  
endelse  
END
```



# Complete the program

---

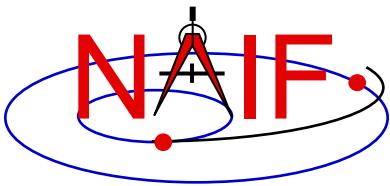
Navigation and Ancillary Information Facility

To finish up the program we need to declare the variables we've used.

- We'll highlight techniques used by NAIF programmers
- Add remaining IDL code required to make a syntactically valid program

```
PRO PROG_GEOMETRY

  ROOM      = 10L
  setupf   = ''
  satnm    = ''
  fixref   = ''
  scnm     = ''
  instrm   = ''
  time     = ''
  R2D      = cspice_dpr()
```

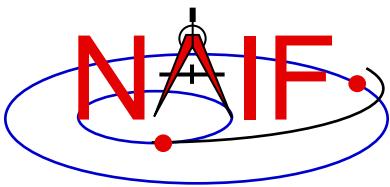


# Complete source code - 1

---

## Navigation and Ancillary Information Facility

```
; ; Prompt for the user-supplied inputs for our program.  
read, setupf, PROMPT='Enter setup file name > '  
cspice_furnsh, setupf  
read, satnm , PROMPT='Enter satellite name > '  
read, fixref, PROMPT='Enter satellite frame > '  
read, scnm , PROMPT='Enter spacecraft name > '  
read, instnm, PROMPT='Enter instrument name > '  
read, time , PROMPT='Enter time > '  
  
; ; Get the epoch corresponding to the input time:  
cspice_str2et, time, et  
  
; ; Get the radii of the satellite.  
cspice_bodvrd, satnm, 'RADII', 3, radii  
  
; ; Get the instrument boresight and frame name.  
  
cspice_bodn2c, instnm, instid, found  
if ( NOT found ) then begin  
    print, "Unable to determine ID for instrument: ", instnm  
    return  
endif  
cspice_getfov, instid, ROOM, shape, iframe, insite, bundry
```



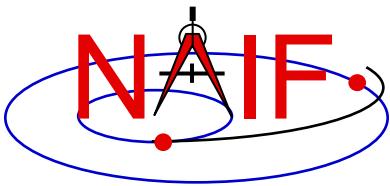
# Complete source code - 2

## Navigation and Ancillary Information Facility

```
; Compute the boresight ray intersection with the surface of the
; target body.
cspice_sincpt, 'Ellipsoid', satnm, et, fixref, 'CN+S', scnm, $
    iframe, insite, point, trgepc, srfvec, found

; If an intercept is found, compute planetocentric and planetodetic
; latitude and longitude of the point.
if ( found ) then begin
    cspice_reclat, point, r, pclon, pclat
    ; Let re, rp, and f be the satellite's longer equatorial
    ; radius, polar radius, and flattening factor.
    re = radii[0]
    rp = radii[2]
    f = ( re - rp ) / re
    cspice_recgeo, point, re, f, pdlon, pdlat, alt

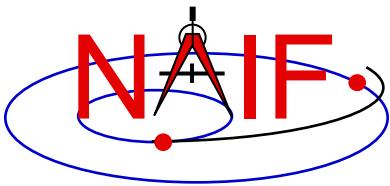
    ; Compute illumination angles at the surface point.
    cspice_ilumin, 'Ellipsoid', satnm, et, fixref, 'CN+S', scnm, $
        point, trgepc, srfvec, phase, solar, emissn
    ; Display results. Convert angles from radians to degrees
    ; for output.
    print
    print, 'Intercept planetocentric longitude      (deg) : ', $
        R2D*pclon
```



# Complete source code - 3

## Navigation and Ancillary Information Facility

```
print, 'Intercept planetocentric latitude      (deg): ', $  
       R2D*pclat  
print, 'Intercept planetodetic longitude      (deg): ', $  
       R2D*pdlon  
print, 'Intercept planetodetic latitude      (deg): ', $  
       R2D*pdlat  
print, 'Range from spacecraft to intercept point (km): ', $  
       cspice_vnorm(srfvec)  
print, 'Intercept phase angle                (deg): ', $  
       R2D*phase  
print, 'Intercept solar incidence angle      (deg): ', $  
       R2D*solar  
print, 'Intercept emission angle            (deg): ', $  
       R2D*emissn  
  
endif else begin  
    print, 'No intercept point found at ' + time  
endelse  
  
;; Unload the kernels and clear the kernel pool  
cspice_kclear  
END
```



# Compile the program - 1

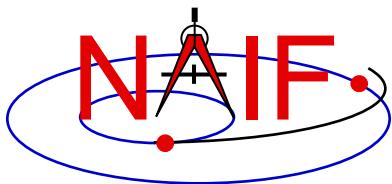
---

Navigation and Ancillary Information Facility

**Though IDL functions in a manner similar to interpreted languages, it does compile source files to a binary form.**

**Ensure that both the Icy Toolkit, and an IDL installation are properly installed. IDL must load the Icy DLM, icy.dlm/icy.so(dll) to compile those scripts containing Icy calls. IDL loads DLMs from default locations and from the current directory when the user ran IDL. The user may also explicitly load a DLM with the `dlm_register` command.**

**Now compile the code.**

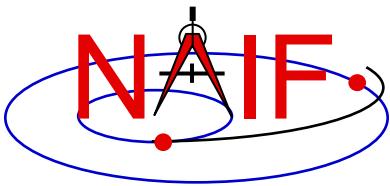


# Compile the program - 2

Navigation and Ancillary Information Facility

Terminal Window

```
IDL> .compile prog_geometry.pro
% Compiled module: PROG_GEOMETRY.
```



# Running the program

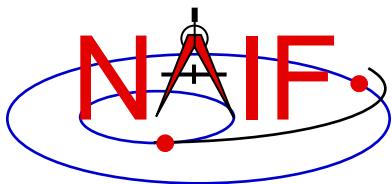
---

Navigation and Ancillary Information Facility

It looks like we have everything taken care of:

- We have all necessary kernels
- We made a setup file (metakernel) pointing to them
- We wrote the program
- We compiled the program

Let's run it.



# Running the program

Navigation and Ancillary Information Facility

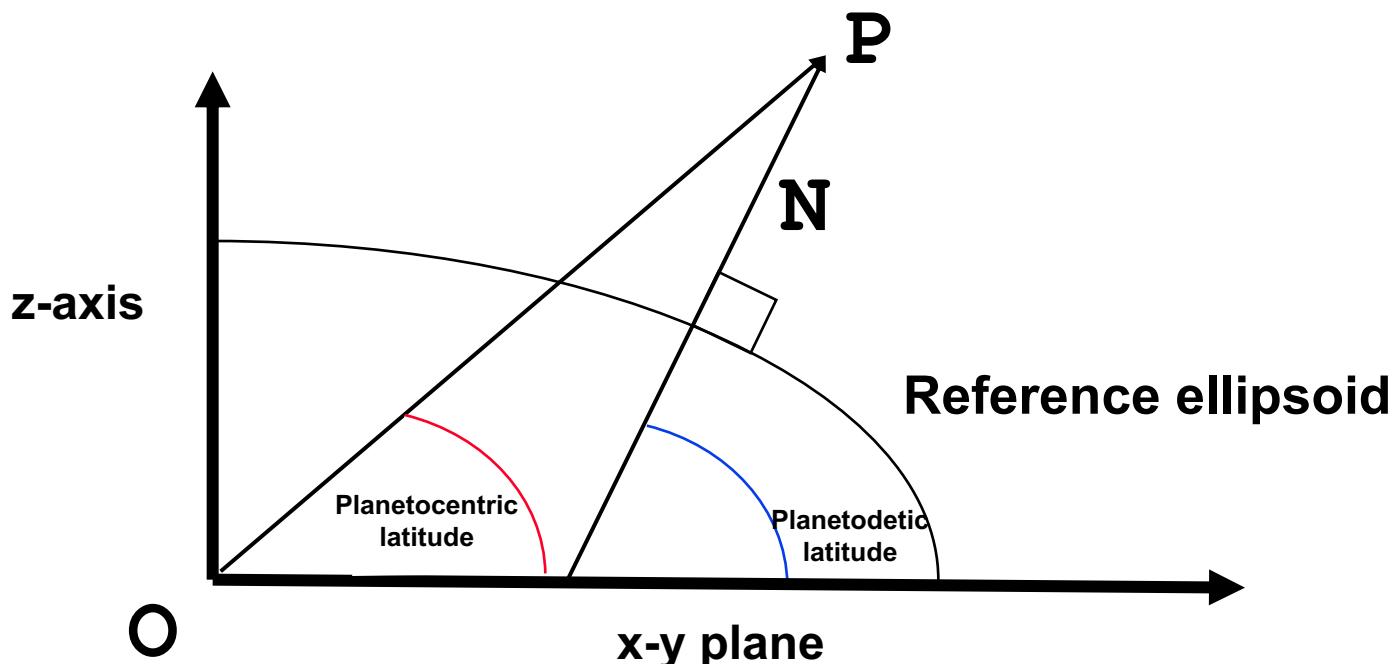
Terminal Window

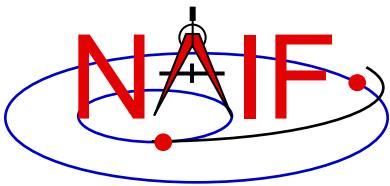
```
IDL> prog_geometry
Enter setup file name > setup.ker
Enter satellite name > PHOEBE
Enter satellite frame > IAU_PHOEBE
Enter spacecraft name > CASSINI
Enter instrument name > CASSINI_ISS_NAC
Enter time > 2004 jun 11 19:32:00

Intercept planetocentric longitude (deg) : 39.843719
Intercept planetocentric latitude (deg) : 4.1958778
Intercept planetodetic longitude (deg) : 39.843719
Intercept planetodetic latitude (deg) : 5.0480106
Range from spacecraft to intercept point (km) : 2089.1697
Intercept phase angle (deg) : 28.139479
Intercept solar incidence angle (deg) : 18.247220
Intercept emission angle (deg) : 17.858309
```

- **Latitude definitions:**

- Planetocentric latitude of a point P: angle between segment from origin to point and x-y plane (red arc in diagram).
- Planetodetic latitude of a point P: angle between x-y plane and extension of ellipsoid normal vector N that connects x-y plane and P (blue arc in diagram).



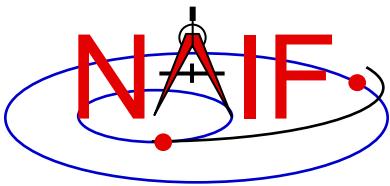


---

Navigation and Ancillary Information Facility

# Writing a Mice (MATLAB) Based Program

January 2020

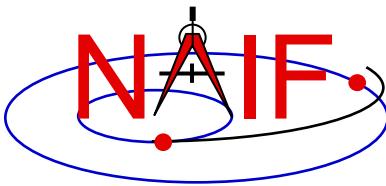


# Viewing This Tutorial

---

Navigation and Ancillary Information Facility

**Undefined variables are displayed in red**  
**Results are displayed in blue**



# Introduction

---

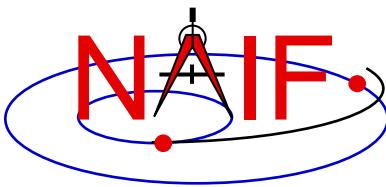
Navigation and Ancillary Information Facility

First, let's go over the important steps in the process of writing a Mice-based program and putting it to work:

- Understand the geometry problem.
- Identify the set of SPICE kernels that contain the data needed to perform the computation.
- Select the SPICE APIs needed to compute the quantities of interest.
- Write and execute the program.
- Get actual kernel files and verify that they contain the data needed to support the computation for the time(s) of interest.
- Run the program.

To illustrate these steps, let's write a program that computes the apparent intersection of the boresight ray of a given CASSINI science instrument with the surface of a given Saturnian satellite. The program will compute:

- Planetocentric and planetodetic (geodetic) latitudes and longitudes of the intercept point.
- Range from spacecraft to intercept point.
- Illumination angles (phase, solar incidence, and emission) at the intercept point.



# Observation geometry

Navigation and Ancillary Information Facility

We want the boresight intercept on the surface, range from s/c to intercept, and illumination angles at the intercept point.

When? **TIME (UTC, TDB or TT)**

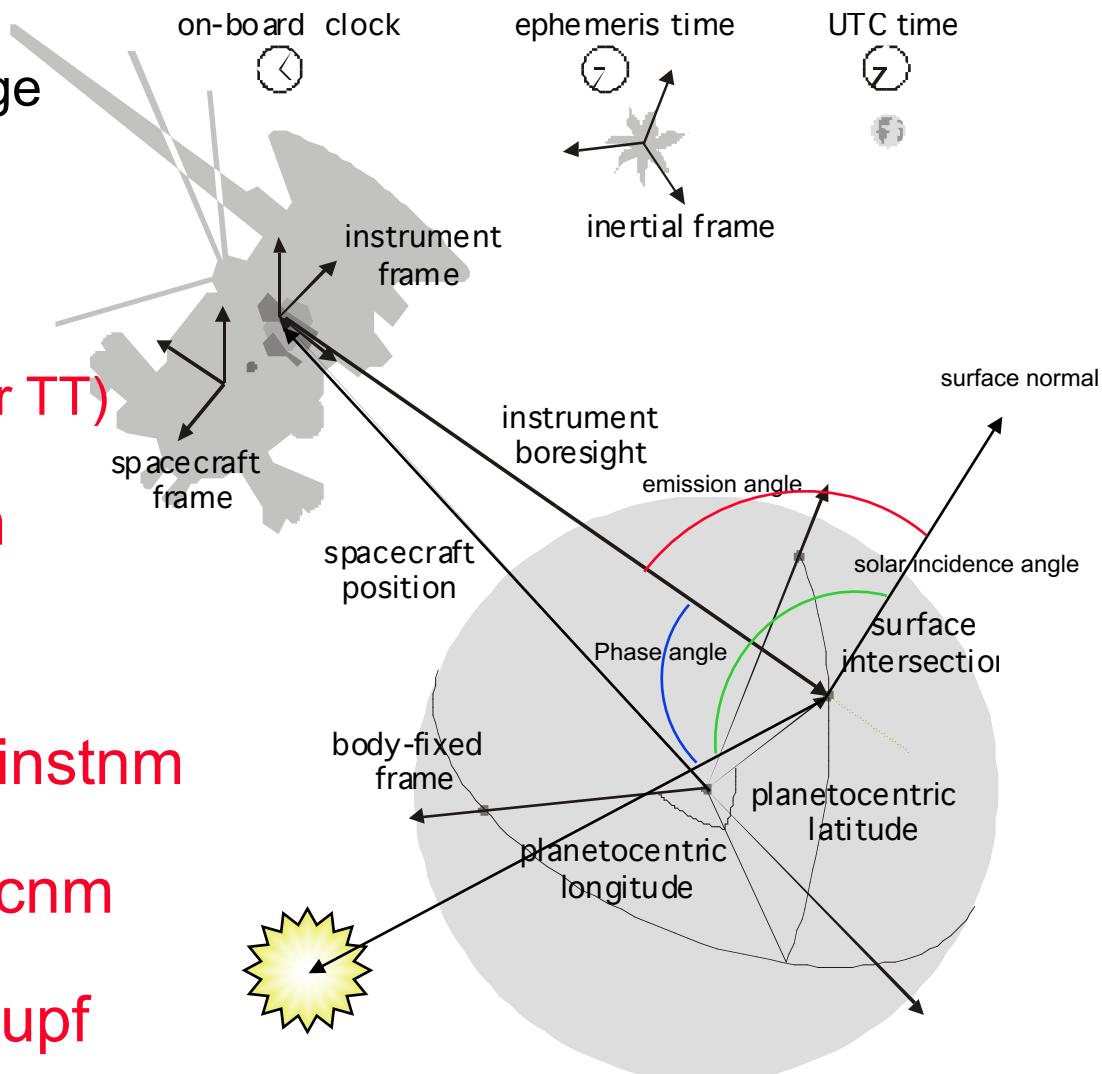
On what object? **satnm**

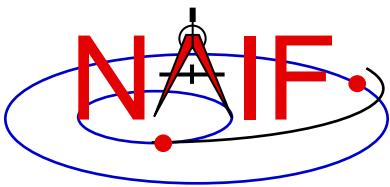
In what frame? **fixref**

For which instrument? **instnm**

For what spacecraft? **scnm**

Using what model? **setupf**





# Needed Data

Navigation and Ancillary Information Facility

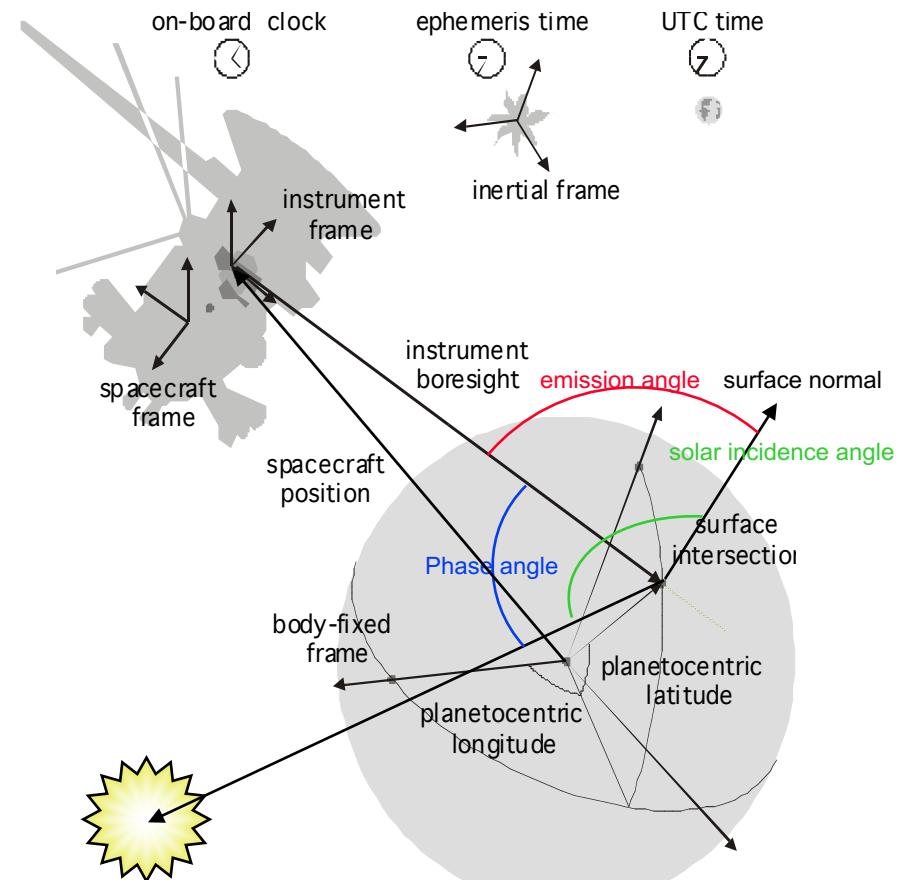
Time transformation kernels

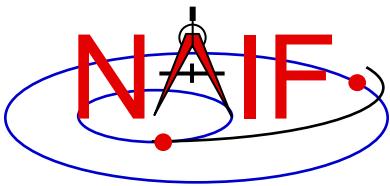
Orientation models

Instrument descriptions

Shapes of satellites, planets

Ephemerides for spacecraft,  
Saturn barycenter and satellites.





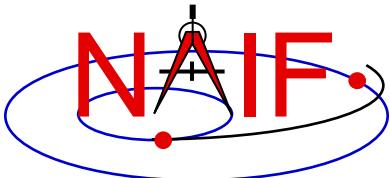
# Which Kernels are Needed?

Navigation and Ancillary Information Facility

Data required to compute vectors, rotations and other parameters shown in the picture are stored in the SPICE kernels listed below.

Note: these kernels have been selected to support this presentation; they should not be assumed to be appropriate for user applications.

Parameter	Kernel Type	File name
time conversions	generic LSK	naif0009.tls
satellite orientation	CASSINI SCLK	cas00084.tsc
satellite shape	CASSINI PCK	cpck05Mar2004.tpc
satellite position	CASSINI PCK	cpck05Mar2004.tpc
	planet/sat	
planet barycenter position	ephemeris SPK	020514_SE_SAT105.bsp
spacecraft position	planet SPK	981005_PLTEPH-DE405S.bsp
spacecraft orientation	spacecraft SPK	030201AP_SK_SM546_T45.bsp
instrument alignment	spacecraft CK	04135_04171pc_psiv2.bc
instrument boresight	CASSINI FK	cas_v37.tf
	Instrument IK	cas_iss_v09.ti



# Load kernels

---

Navigation and Ancillary Information Facility

The easiest and most flexible way to make these kernels available to the program is via `cspice_furnsh`. For this example we make a setup file (also called a “metakernel” or “furnsh kernel”) containing a list of kernels to be loaded:

Note: these kernels have been selected to support this presentation; they should not be assumed to be appropriate for user applications.

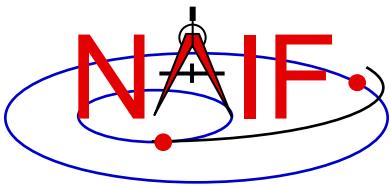
```
\begindata
```

```
KERNELS_TO_LOAD = ('naif0009.tls', 'cas00084.tsc', 'cpck05Mar2004.tpc',
                    '020514_SE_SAT105.bsp', '981005_PLTEPH-DE405S.bsp',
                    '030201AP_SK_SM546_T45.bsp', '04135_04171pc_psiv2.bc',
                    'cas_v37.tf', 'cas_iss_v09.ti')
```

```
\begintext
```

and we make the program prompt for the name of this setup file:

```
setupf = input('Enter setup file name > ', 's');
cspice_furnsh( setupf )
```



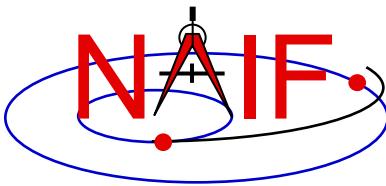
# Programming Solution

---

Navigation and Ancillary Information Facility

- Prompt for setup file (“metakernel”) name; load kernels specified via setup file. (Done on previous chart.)
- Prompt for user inputs required to completely specify problem. Obtain further inputs required by geometry routines via Mice calls.
- Compute the intersection of the boresight direction ray with the surface of the satellite, presented as a triaxial ellipsoid.
- If there is an intersection:
  - Convert Cartesian coordinates of the intersection point to planetocentric latitudinal and planetodetic coordinates
  - Compute spacecraft-to-intercept point range
  - Find the illumination angles (phase, solar incidence, and emission) at the intercept point
- Display the results.

We discuss the geometric portion of the problem first.



# Compute surface intercept

Navigation and Ancillary Information Facility

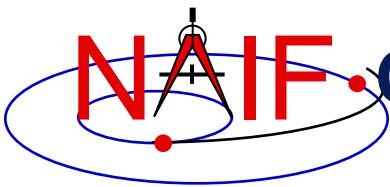
Compute the intercept point (`point`) of the boresight vector (`insite`) specified in the instrument frame (`iframe`) of the instrument mounted on the spacecraft (`scnm`) with the surface of the satellite (`satnm`) at the TDB time of interest (`et`) in the satellite's body-fixed frame (`fixref`). This call also returns the light-time corrected epoch at the intercept point (`trgepc`), the spacecraft-to-intercept point vector (`srfvec`), and a flag indicating whether the intercept was found (`found`). We use "converged Newtonian" light time plus stellar aberration corrections to produce the most accurate surface intercept solution possible. We model the surface of the satellite as an ellipsoid.

```
[point, trgepc, srfvec, found] = cspice_sincpt( ...
    'Ellipsoid', satnm, et, fixref, 'CN+S', scnm, iframe, insite );
```

The range we want is obtained from the outputs of `cspice_sincpt`. These outputs are defined only if a surface intercept is found. If `found` is true, the spacecraft-to-surface intercept range is the norm of the output argument `srfvec`. Units are km. We use the MATLAB function `norm` to obtain the norm:

```
norm( srfvec )
```

We'll write out the range data along with the other program results.



# Compute Lat/Lon and Illumination Angles

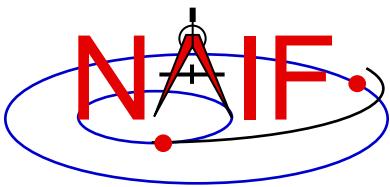
Navigation and Ancillary Information Facility

Compute the planetocentric latitude (`pclat`) and longitude (`pclon`), as well as the planetodetic latitude (`pdlat`) and longitude (`pdlon`) of the intersection point.

```
if ( found )  
    [r, pclon, pclat] = cspice_reclat( point );  
  
    % Let re, rp, and f be the satellite's longer equatorial  
    % radius, polar radius, and flattening factor.  
    re = radii(1);  
    rp = radii(3);  
    f = ( re - rp ) / re;  
  
    [pdlat, pdlon, alt] = cspice_recgeo( point, re, f );
```

The illumination angles we want are the outputs of `cspice_ilumin`. Units are radians.

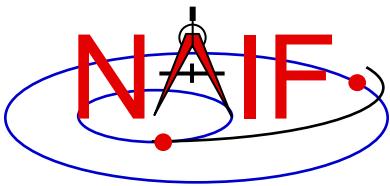
```
[trgepc, srfvec, phase, solar, emissn] = cspice_ilumin( ...  
    'Ellipsoid', satnm, et, fixref, 'CN+S', scnm, point );
```



# Geometry Calculations: Summary

Navigation and Ancillary Information Facility

```
% Compute the boresight ray intersection with the surface of the
% target body.
[point, trgepc, srfvec, found] = cspice_sincpt( ...
    'Ellipsoid', satnm, et, fixref, 'CN+S', scnm, iframe, insite );
% If an intercept is found, compute planetocentric and planetodetic
% latitude and longitude of the point.
if ( found )
    [r, pclon, pclat] = cspice_reclat( point );
    % Let re, rp, and f be the satellite's longer equatorial
    % radius, polar radius, and flattening factor.
    re = radii(1);
    rp = radii(3);
    f = ( re - rp ) / re;
    [pdlon, pdlat, alt] = cspice_recgeo( point, re, f );
    % Compute illumination angles at the surface point.
    [trgepc, srfvec, phase, solar, emissn] = cspice_ilumin( ...
        'Ellipsoid', satnm, et, fixref, 'CN+S', scnm, point );
    ...
else
    ...
end
```



# Get inputs - 1

---

Navigation and Ancillary Information Facility

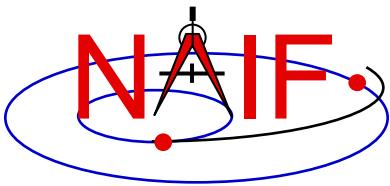
The code above used quite a few inputs that we don't have yet:

- TDB epoch of interest (`et`);
- satellite and s/c names (`satnm`, `scnm`);
- satellite body-fixed frame name (`fixref`);
- satellite ellipsoid radii (`radii`);
- instrument fixed frame name (`iframe`);
- instrument boresight vector in the instrument frame (`insite`);

Some of these values are user inputs; others can be obtained via Mice calls once the required kernels have been loaded.

Let's prompt for the satellite name (`satnm`), satellite frame name (`fixref`), spacecraft name (`scnm`), instrument name (`instnm`) and time of interest (`time`):

```
satnm  = input( 'Enter satellite name > ', 's' );
fixref = input( 'Enter satellite frame > ', 's' );
scnm   = input( 'Enter spacecraft name > ', 's' );
instnm = input( 'Enter instrument name > ', 's' );
time   = input( 'Enter time                 > ', 's' );
```



# Get Inputs - 2

---

Navigation and Ancillary Information Facility

Then we can get the rest of the inputs from Mice calls:

To get the TDB epoch (`et`) from the user-supplied time string (which may refer to the UTC, TDB or TT time systems):

```
et = cspice_str2et( time );
```

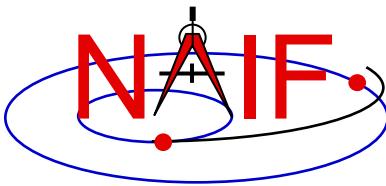
To get the satellite's ellipsoid radii (`radii`):

```
radii = cspice_bodvrd( satnm, 'RADII', 3 );
```

To get the instrument boresight direction (`insite`) and the name of the instrument frame (`iframe`) in which it is defined:

```
[instid, found] = cspice_bodn2c( instnm );
if (~found)
    txt = sprintf( 'Unable to determine ID for instrument: %d', ...
                  instnm );
    error(txt)
end

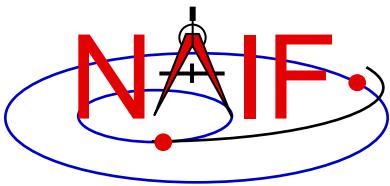
[shape, iframe, insite, bndry] = cspice_getfov( instid, ROOM );
```



# Getting inputs: summary

Navigation and Ancillary Information Facility

```
% Prompt for the user-supplied inputs for our program.  
setupf = input( 'Enter setup file name > ', 's' );  
cspice_furnsh( setupf )  
satnm = input( 'Enter satellite name > ', 's' );  
fixref = input( 'Enter satellite frame > ', 's' );  
scnm = input( 'Enter spacecraft name > ', 's' );  
instnm = input( 'Enter instrument name > ', 's' );  
time = input( 'Enter time > ', 's' );  
  
% Get the epoch corresponding to the input time:  
et = cspice_str2et( time );  
  
% Get the radii of the satellite.  
radii = cspice_bodvrd( satnm, 'RADII', 3 );  
  
% Get the instrument boresight and frame name.  
[instid, found] = cspice_bodn2c( instnm );  
if ( ~found )  
    txt = sprintf( 'Unable to determine ID for instrument: %d', ...  
                  instnm );  
    error(txt)  
end  
[shape, iframe, insite, bundry] = cspice_getfov( instid, ROOM );
```



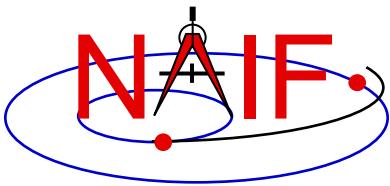
# Display results

Navigation and Ancillary Information Facility

```
...
% Display results. Convert angles from radians to degrees
% for output.

fprintf( 'Intercept planetocentric longitude
          (deg): %11.6f\n', ...
          cspice_dpr()*pclon )
fprintf( 'Intercept planetocentric latitude
          (deg): %11.6f\n', ...
          cspice_dpr()*pclat )
fprintf( 'Intercept planetodetic longitude
          (deg): %11.6f\n', ...
          cspice_dpr()*pdlon )
fprintf( 'Intercept planetodetic latitude
          (deg): %11.6f\n', ...
          cspice_dpr()*pdlat )
fprintf( 'Range from spacecraft to intercept point (km): %11.6f\n', ...
          norm(srfvec) )
fprintf( 'Intercept phase angle
          (deg): %11.6f\n', ...
          cspice_dpr()*phase )
fprintf( 'Intercept solar incidence angle
          (deg): %11.6f\n', ...
          cspice_dpr()*solar )
fprintf( 'Intercept emission angle
          (deg): %11.6f\n', ...
          cspice_dpr()*emissn )

else
    disp( ['No intercept point found at ' time ] )
end
```



# Complete the program

---

Navigation and Ancillary Information Facility

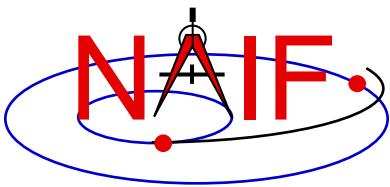
To finish up the program we need to declare the variables we've used.

- We'll highlight techniques used by NAIF programmers
- Add remaining MATLAB code required to make a syntactically valid program

```
ROOM      = 10;
R2D       = cspice_dpr;

% Prompt for the user-supplied inputs for our program.
setupf = input( 'Enter setup file name > ', 's' );
cspice_furnsh( setupf )

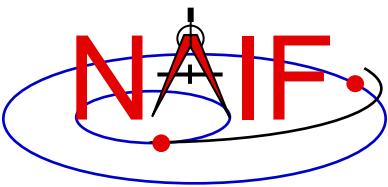
satnm   = input( 'Enter satellite name > ', 's' );
fixref  = input( 'Enter satellite frame > ', 's' );
scnm    = input( 'Enter spacecraft name > ', 's' );
instnm  = input( 'Enter instrument name > ', 's' );
time    = input( 'Enter time                  > ', 's' );
```



# Complete source code - 1

Navigation and Ancillary Information Facility

```
% Get the epoch corresponding to the input time:  
et = cspice_str2et( time );  
  
% Get the radii of the satellite.  
radii = cspice_bodvrd( satnm, 'RADII', 3 );  
  
% Get the instrument boresight and frame name.  
[instid, found] = cspice_bodn2c( instnm );  
  
if ( ~found )  
    txt = sprintf( 'Unable to determine ID for instrument: %d', ...  
                  instnm );  
    error(txt)  
end  
  
[shape, iframe, insite, bndry] = cspice_getfov( instid, ROOM );
```



# Complete source code - 2

## Navigation and Ancillary Information Facility

```
% Compute the boresight ray intersection with the surface of the
% target body.

[point, trgepc, srfvec, found] = cspice_sincpt( ...

    'Ellipsoid', satnm, et, fixref, 'CN+S', scnm, iframe, insite );

% If an intercept is found, compute planetocentric and planetodetic
% latitude and longitude of the point.

if ( found )
    [r, pclon, pclat] = cspice_reclat( point );

    % Let re, rp, and f be the satellite's longer equatorial
    % radius, polar radius, and flattening factor.

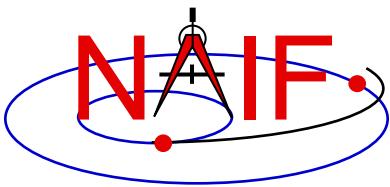
    re = radii(1);
    rp = radii(3);
    f = ( re - rp ) / re;

    [pdlon, pdlat, alt] = cspice_recgeo( point, re, f );

    % Compute illumination angles at the surface point.

    [trgepc, srfvec, phase, solar, emissn] = cspice_ilumin( ...

        'Ellipsoid', satnm, et, fixref, 'CN+S', scnm, point );
```



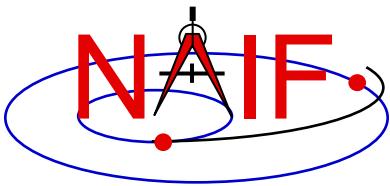
# Complete source code - 3

## Navigation and Ancillary Information Facility

```
% Display results. Convert angles from radians to degrees
% for output.
fprintf( 'Intercept planetocentric longitude
          (deg): %11.6f\n', ...
R2D*pclon )
fprintf( 'Intercept planetocentric latitude
          (deg): %11.6f\n', ...
R2D*pclat )
fprintf( 'Intercept planetodetic longitude
          (deg): %11.6f\n', ...
R2D*pdlon )
fprintf( 'Intercept planetodetic latitude
          (deg): %11.6f\n', ...
R2D*pdlat )
fprintf( 'Range from spacecraft to intercept point (km):
          (deg): %11.6f\n', ...
norm(srfvec) )
fprintf( 'Intercept phase angle
          (deg): %11.6f\n', ...
R2D*phase )
fprintf( 'Intercept solar incidence angle
          (deg): %11.6f\n', ...
R2D*solar )
fprintf( 'Intercept emission angle
          (deg): %11.6f\n', ...
R2D*emissn )

else
    disp( ['No intercept point found at ' time ]
end

% Unload the kernels and clear the kernel pool
cspice_kclear
```



# Running the program

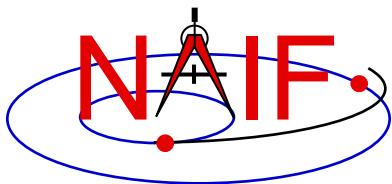
---

Navigation and Ancillary Information Facility

It looks like we have everything taken care of:

- We have all necessary kernels
- We made a setup file (metakernel) pointing to them
- We wrote the program

Let's run it.



# Running the program

Navigation and Ancillary Information Facility

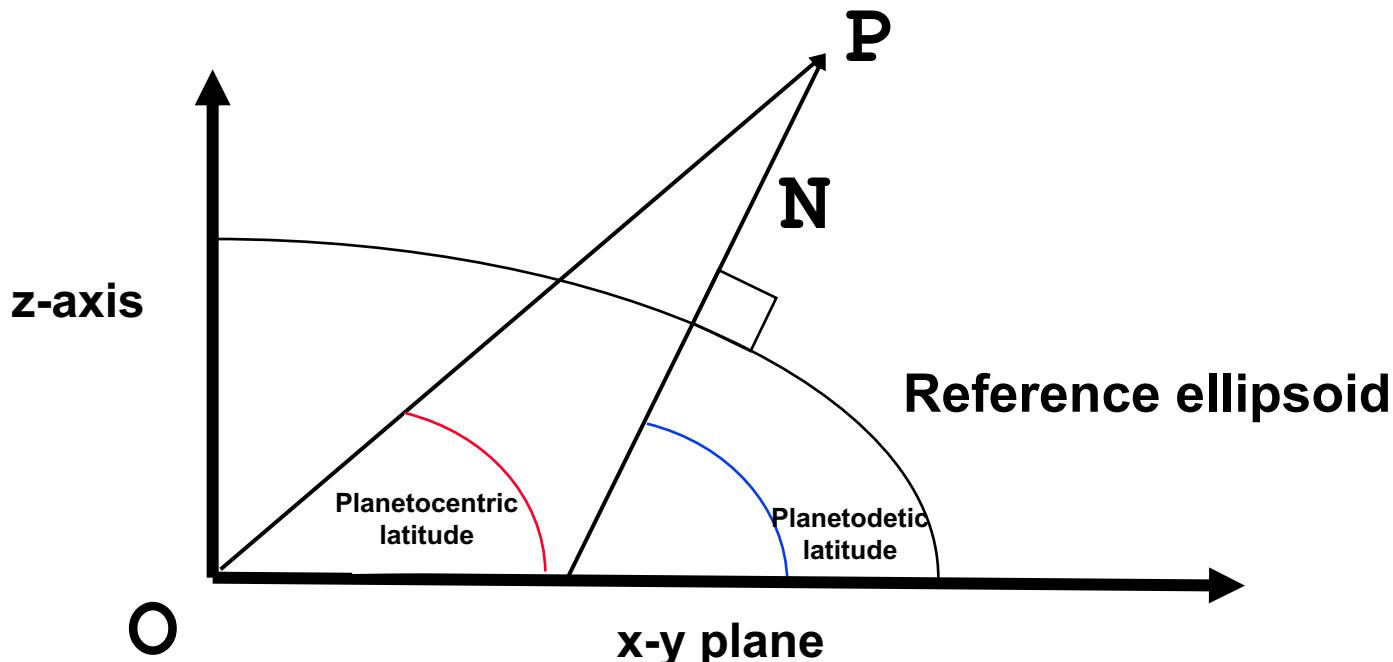
Terminal Window

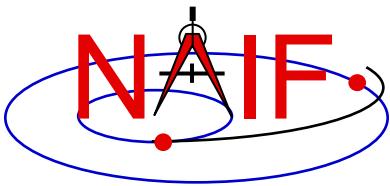
```
>> prog_geometry
Enter setup file name > setup.ker
Enter satellite name > PHOEBE
Enter satellite frame > IAU_PHOEBE
Enter spacecraft name > CASSINI
Enter instrument name > CASSINI_ISS_NAC
Enter time > 2004 jun 11 19:32:00

Intercept planetocentric longitude (deg) : 39.843719
Intercept planetocentric latitude (deg) : 4.195878
Intercept planetodetic longitude (deg) : 39.843719
Intercept planetodetic latitude (deg) : 5.048011
Range from spacecraft to intercept point (km) : 2089.169724
Intercept phase angle (deg) : 28.139479
Intercept solar incidence angle (deg) : 18.247220
Intercept emission angle (deg) : 17.858309
```

- **Latitude definitions:**

- Planetocentric latitude of a point P: angle between segment from origin to point and x-y plane (red arc in diagram).
- Planetodetic latitude of a point P: angle between x-y plane and extension of ellipsoid normal vector N that connects x-y plane and P (blue arc in diagram).



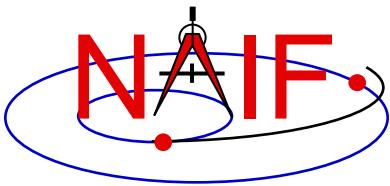


---

Navigation and Ancillary Information Facility

# Writing a CSPICE (C) Based Program

January 2020

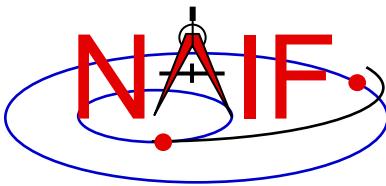


# Viewing This Tutorial

---

Navigation and Ancillary Information Facility

**Undefined variables are displayed in red**  
**Results are displayed in blue**



# Introduction

---

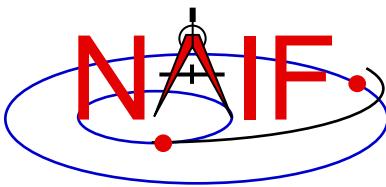
Navigation and Ancillary Information Facility

First, let's go over the important steps in the process of writing a CSPICE-based program and putting it to work:

- Understand the geometry problem.
- Identify the set of SPICE kernels that contain the data needed to perform the computation.
- Select the SPICE APIs needed to compute the quantities of interest.
- Write and compile the program.
- Get actual kernel files and verify that they contain the data needed to support the computation for the time(s) of interest.
- Run the program.

To illustrate these steps, let's write a program that computes the apparent intersection of the boresight ray of a given CASSINI science instrument with the surface of a given Saturnian satellite. The program will compute

- Planetocentric and planetodetic (geodetic) latitudes and longitudes of the intercept point.
- Range from spacecraft to intercept.
- Illumination angles (phase, solar incidence, and emission) at the intercept point.



# Observation geometry

Navigation and Ancillary Information Facility

We want the boresight intercept on the surface, range from s/c to intercept, and illumination angles at the intercept point.

When? **TIME (UTC, TDB or TT)**

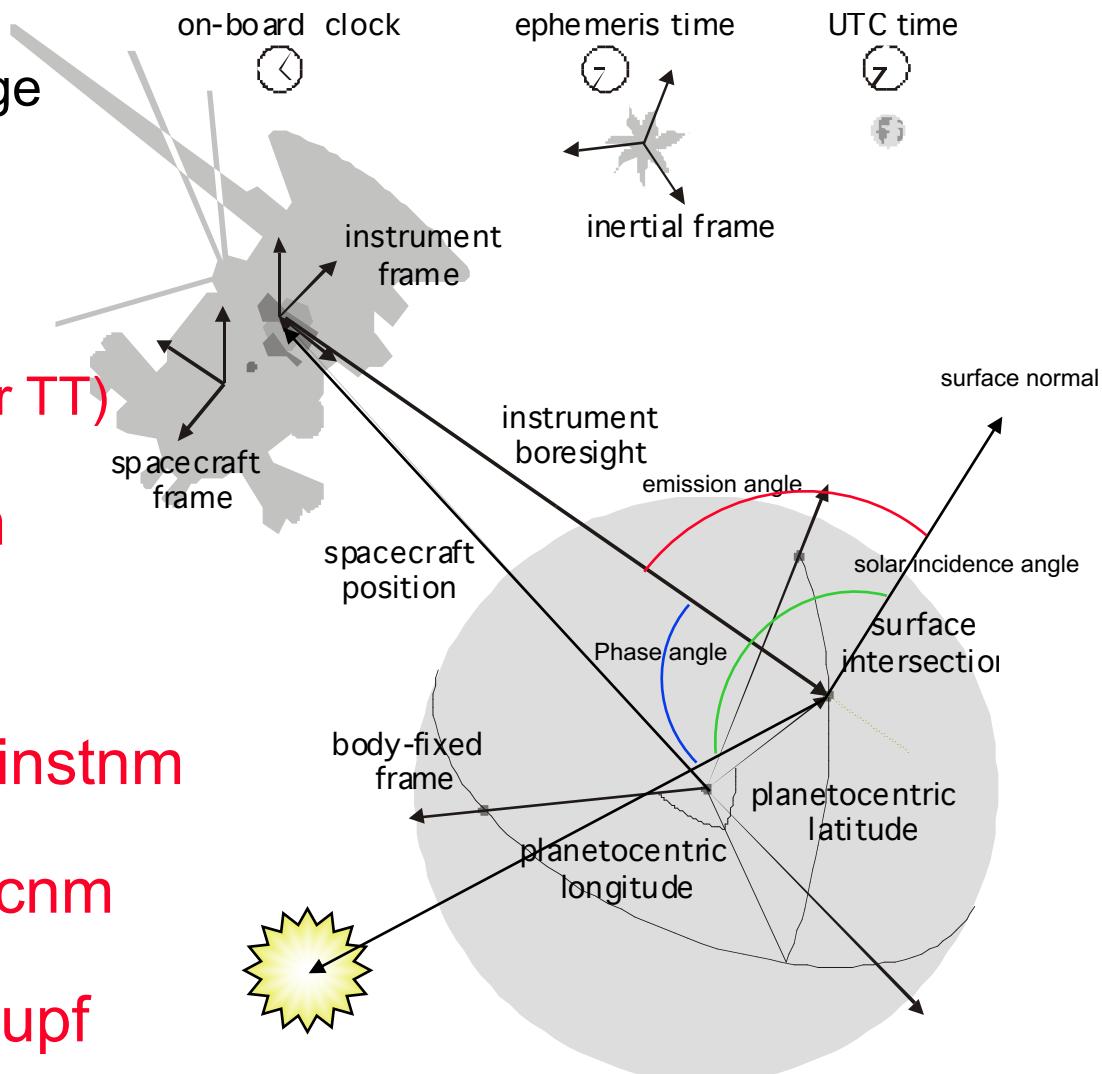
On what object? **satnm**

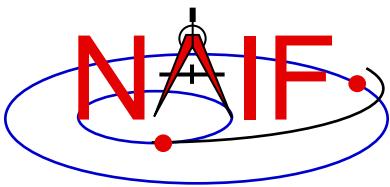
In what frame? **fixref**

For which instrument? **instnm**

For what spacecraft? **scnm**

Using what model? **setupf**





# Needed Data

Navigation and Ancillary Information Facility

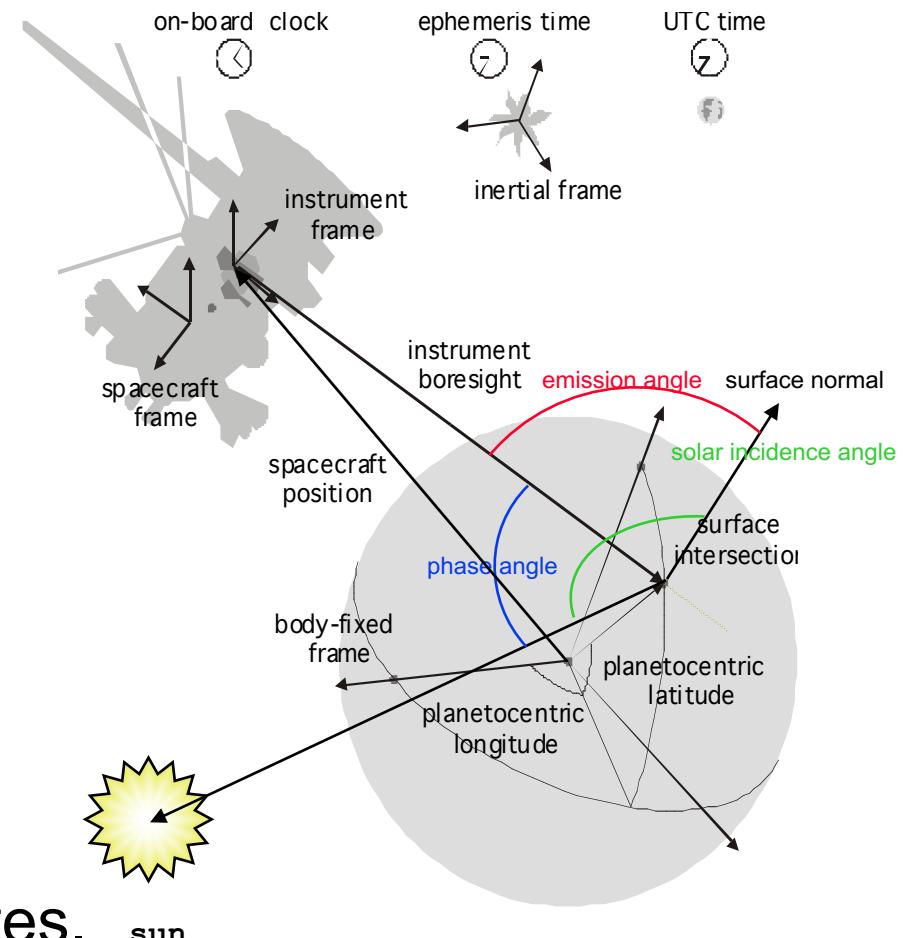
Time transformation kernels

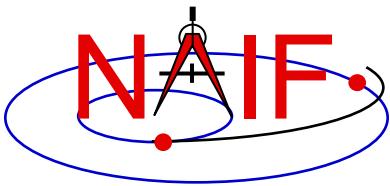
Orientation models

Instrument descriptions

Shapes of satellites, planets

Ephemerides for spacecraft,  
Saturn barycenter and satellites.





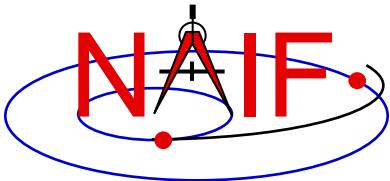
# Which Kernels are Needed?

Navigation and Ancillary Information Facility

Data required to compute vectors, rotations and other parameters shown in the picture are stored in the SPICE kernels listed below.

Note: these kernels have been selected to support this presentation; they should not be assumed to be appropriate for user applications.

Parameter	Kernel Type	File name
time conversions	generic LSK	naif0009.tls
satellite orientation	CASSINI SCLK	cas00084.tsc
satellite shape	CASSINI PCK	cpck05Mar2004.tpc
satellite position	CASSINI PCK	cpck05Mar2004.tpc
	planet/sat	
planet barycenter position	ephemeris SPK	020514_SE_SAT105.bsp
spacecraft position	planet SPK	981005_PLTEPH-DE405S.bsp
spacecraft orientation	spacecraft SPK	030201AP_SK_SM546_T45.bsp
instrument alignment	spacecraft CK	04135_04171pc_psiv2.bc
instrument boresight	CASSINI FK	cas_v37.tf
	Instrument IK	cas_iss_v09.ti



# Load Kernels

---

Navigation and Ancillary Information Facility

The easiest and most flexible way to make required kernels available to the program is via `furnsh_c`. For this example we make a setup file (also called a “metakernel” or “furnsh kernel”) containing a list of kernels to be loaded:

Note: these kernels have been selected to support this presentation; they should not be assumed to be appropriate for user applications.

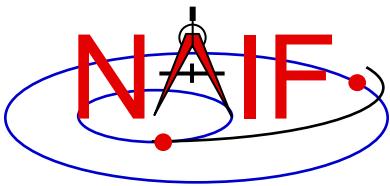
```
\begindata
```

```
KERNELS_TO_LOAD = ('naif0009.tls', 'cas00084.tsc', 'cpck05Mar2004.tpc',
                    '020514_SE_SAT105.bsp', '981005_PLTEPH-DE405S.bsp',
                    '030201AP_SK_SM546_T45.bsp', '04135_04171pc_psiv2.bc',
                    'cas_v37.tf', 'cas_iss_v09.ti')
```

```
\begintext
```

and we make the program prompt for the name of this setup file:

```
prompt_c ( "Enter setup file name > ", FILESZ, setupf );
furnsh_c ( setupf );
```



# Programming Solution

---

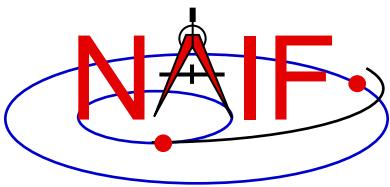
Navigation and Ancillary Information Facility

- Prompt for setup file (“metakernel”) name; load kernels specified via setup file. (Done on previous chart.)
- Prompt for user inputs required to completely specify problem. Obtain further inputs required by geometry routines via CSPICE calls.
- Compute the intersection of the boresight direction ray with the surface of the satellite, presented as a triaxial ellipsoid.

If there is an intersection,

- Convert Cartesian coordinates of the intercept point to planetocentric latitudinal and planetodetic coordinates
- Compute spacecraft-to-intercept point range
- Find the illumination angles (phase, solar incidence, and emission) at the intercept point
- Display the results.

We discuss the geometric portion of the problem next.



# Compute Surface Intercept and Ranges

Navigation and Ancillary Information Facility

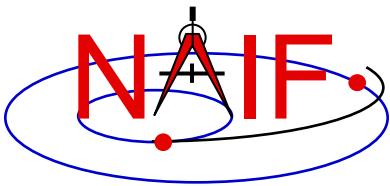
Compute the intercept point (`point`) of the boresight vector (`insite`) specified in the instrument frame (`iframe`) of the instrument mounted on the spacecraft (`scnm`) with the surface of the satellite (`satnm`) at the TDB time of interest (`et`) in the satellite's body-fixed frame (`fixref`). This call also returns the light-time corrected epoch at the intercept point (`trgepc`), the spacecraft-to-intercept point vector (`srfvec`), and a flag indicating whether the intercept was found (`found`). We use "converged Newtonian" light time plus stellar aberration corrections to produce the most accurate surface intercept solution possible. We model the surface of the satellite as an ellipsoid.

```
sincpt_c ( "Ellipsoid", satnm, et, fixref, "CN+S", scnm, iframe, insite,  
           point, &trgepc, srfvec, &found );
```

The range we want is obtained from the outputs of `sincpt_c`. These outputs are defined only if a surface intercept is found. If `found` is true, the spacecraft-to-surface intercept range is the norm of the output argument `srfvec`. Units are km. We use the CSPICE function `vnorm_c` to obtain the norm:

```
vnorm_c ( srfvec )
```

We'll write out the range data along with the other program results.



# Compute Lat/Lon and Illumination Angles

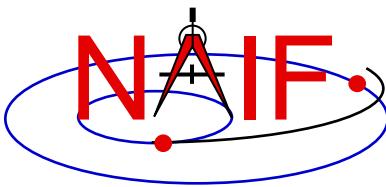
Navigation and Ancillary Information Facility

Compute the planetocentric latitude (`pclat`) and longitude (`pclon`), as well as the planetodetic latitude (`pdlat`) and longitude (`pdlon`) of the intersection point.

```
if ( found )  
{  
    reclat_c ( point,  &r,  &pclon, &pclat );  
  
    /* Let re, rp, and f be the satellite's longer equatorial  
       radius, polar radius, and flattening factor. */  
    re = radii[0];  
    rp = radii[2];  
    f = ( re - rp ) / re;  
  
    recgeo_c ( point, re, f, &pdlon, &pdlat, &alt );
```

The illumination angles we want are the outputs of `ilumin_c`. Units are radians.

```
ilumin_c ( "Ellipsoid", satnm, et, fixref, "CN+S", scnm, point,  
           &trgepc, srfvec, &phase, &solar, &emissn );
```



# Geometry Calculations: Summary

Navigation and Ancillary Information Facility

```
/* Compute the boresight ray intersection with the surface of the
   target body. */

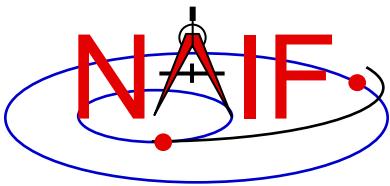
sincpt_c ( "Ellipsoid", satnm, et, fixref, "CN+S", scnm,
           iframe, insite, point, &trgepc, srfvec, &found );

/* If an intercept is found, compute planetocentric and planetodetic
   latitude and longitude of the point. */

if ( found )
{
    reclat_c ( point, &r, &pclon, &pclat );
    /* Let re, rp, and f be the satellite's longer equatorial
       radius, polar radius, and flattening factor. */
    re    =  radii[0];
    rp    =  radii[2];
    f     =  ( re - rp ) / re;
    recgeo_c ( point, re, f, &pdlon, &pdlat, &alt );

    /* Compute illumination angles at the surface point. */
    ilumin_c ( "Ellipsoid", satnm, et, fixref, "CN+S", scnm, point,
               &trgepc, srfvec, &phase, &solar, &emissn );
    ...
}

else
{ ... }
```



# Get Inputs - 1

---

Navigation and Ancillary Information Facility

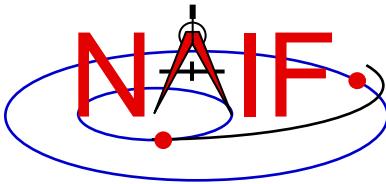
The code above used quite a few inputs that we don't have yet:

- TDB epoch of interest (`et`);
- satellite and s/c names (`satnm`, `scnm`);
- satellite body-fixed frame name (`fixref`);
- satellite ellipsoid radii (`radii`);
- instrument fixed frame name (`iframe`);
- instrument boresight vector in the instrument frame (`insite`);

Some of these values are user inputs; others can be obtained via CSPICE calls once the required kernels have been loaded.

Let's prompt for the satellite name (`satnm`), satellite frame name (`fixref`), spacecraft name (`scnm`), instrument name (`instnm`) and time of interest (`time`):

```
prompt_c ( "Enter satellite name > ", WORDSZ, satnm ) ;
prompt_c ( "Enter satellite frame > ", WORDSZ, fixref ) ;
prompt_c ( "Enter spacecraft name > ", WORDSZ, scnm ) ;
prompt_c ( "Enter instrument name > ", WORDSZ, instnm ) ;
prompt_c ( "Enter time > ", WORDSZ, time ) ;
```



# Get Inputs - 2

---

Navigation and Ancillary Information Facility

Then we can get the rest of the inputs from CSPICE calls:

To get the TDB epoch (`et`) from the user-supplied time string (which may refer to the UTC, TDB or TT time systems):

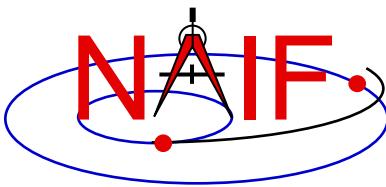
```
str2et_c ( time, &et );
```

To get the satellite's ellipsoid radii (`radii`):

```
bodvrd_c ( satnm, "RADII", 3, &i, radii );
```

To get the instrument boresight direction (`insite`) and the name of the instrument frame (`iframe`) in which it is defined:

```
bodn2c_c ( instnm, &instid, &found );  
  
if ( !found )  
{  
    setmsg_c ( "Instrument name # could not be "  
                "translated to an ID code." );  
    errch_c ( "#", instnm );  
    sigerr_c ( "NAMENOTFOUND" );  
}  
getfov_c ( instid, ROOM, WORDSZ, WORDSZ,  
            shape, iframe, insite, &n, bndry );
```



# Getting Inputs: Summary

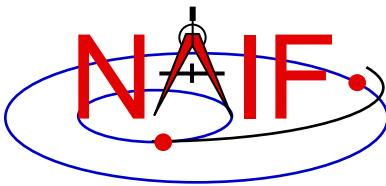
## Navigation and Ancillary Information Facility

```
/* Prompt for the user-supplied inputs for our program      */
prompt_c ( "Enter setup file name > ", FILESZ, setupf );
furnsh_c ( setupf );
prompt_c ( "Enter satellite name > ", WORDSZ, satnm );
prompt_c ( "Enter satellite frame > ", WORDSZ, fixref );
prompt_c ( "Enter spacecraft name > ", WORDSZ, scnm );
prompt_c ( "Enter instrument name > ", WORDSZ, instnm );
prompt_c ( "Enter time             > ", WORDSZ, time );

/* Get the epoch corresponding to the input time: */
str2et_c ( time, &et );

/* Get the radii of the satellite. */
bodvrd_c ( satnm, "RADII", 3, &i, radii );

/* Get the instrument boresight and frame name. */
bodn2c_c ( instnm, &instid, &found );
if ( !found )
{
    setmsg_c ( "Instrument name # could not be "
               "translated to an ID code." );
    errch_c ( "#", instnm );
    sigerr_c ( "NAMENOTFOUND" );
}
getfov_c ( instid, ROOM,     WORDSZ, WORDSZ,
           shape,  iframe,  insite, &n,       bndry );
```

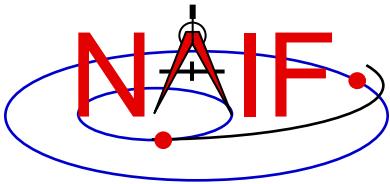


# Display Results

## Navigation and Ancillary Information Facility

```
/* Display results. Convert angles from radians to degrees for output. */
printf ( "\n"
        "Intercept planetocentric longitude          (deg) : %11.6f\n"
        "Intercept planetocentric latitude           (deg) : %11.6f\n"
        "Intercept planetodetic longitude          (deg) : %11.6f\n"
        "Intercept planetodetic latitude           (deg) : %11.6f\n"
        "Range from spacecraft to intercept point   (km) : %11.6f\n"
        "Intercept phase angle                      (deg) : %11.6f\n"
        "Intercept solar incidence angle          (deg) : %11.6f\n"
        "Intercept emission angle                  (deg) : %11.6f",
        dpr_c() * pclon,
        dpr_c() * pclat,
        dpr_c() * pdlon,
        dpr_c() * pdlat,
        vnorm_c( srfvec ),
        dpr_c() * phase,
        dpr_c() * solar,
        dpr_c() * emissn
    );
}

else
{
    printf ( "No intercept point found at %s\n", time );
}
```



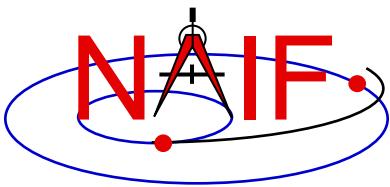
# Complete the Program

---

Navigation and Ancillary Information Facility

To finish up the program we need to declare the variables we've used.

- We'll highlight techniques used by NAIF programmers
- Add remaining C code required to make a syntactically valid program



# Complete Source Code - 1

## Navigation and Ancillary Information Facility

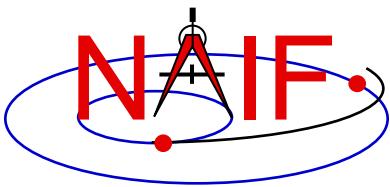
```
#include <stdio.h>
#include "SpiceUsr.h"
```

```
int main ()
{
#define FILESZ          256
#define WORDSZ          41
#define ROOM            10
```

```
SpiceBoolean   found;
```

```
SpiceChar      iframe [WORDSZ];
SpiceChar      instnm [WORDSZ];
SpiceChar      satnm [WORDSZ];
SpiceChar      fixref [WORDSZ];
SpiceChar      scnm  [WORDSZ];
SpiceChar      setupf [FILESZ];
SpiceChar      shape [WORDSZ];
SpiceChar      time  [WORDSZ];
```

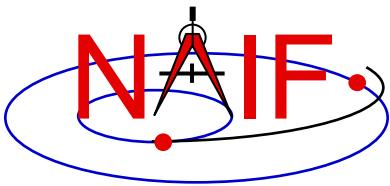
```
SpiceDouble    alt;
SpiceDouble    bndry [ROOM] [3];
SpiceDouble    emissn;
SpiceDouble    et;
SpiceDouble    f;
SpiceDouble    insite [3];
SpiceDouble    srfvec [3];
SpiceDouble    pclat;
SpiceDouble    pclon;
SpiceDouble    pdlat;
SpiceDouble    pdlon;
SpiceDouble    phase;
SpiceDouble    point [3];
SpiceDouble    r;
SpiceDouble    radii [3];
SpiceDouble    re;
SpiceDouble    rp;
SpiceDouble    solar;
SpiceDouble    trgepc;
SpiceInt       i;
SpiceInt       instid;
SpiceInt       n;
```



# Complete Source Code - 2

## Navigation and Ancillary Information Facility

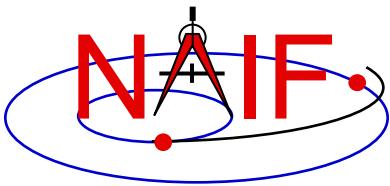
```
/* Prompt for the user-supplied inputs for our program      */
prompt_c ( "Enter setup file name > ", FILESZ, setupf );
furnsh_c ( setupf );
prompt_c ( "Enter satellite name > ", WORDSZ, satnm );
prompt_c ( "Enter satellite frame > ", WORDSZ, fixref );
prompt_c ( "Enter spacecraft name > ", WORDSZ, schnm );
prompt_c ( "Enter instrument name > ", WORDSZ, instnm );
prompt_c ( "Enter time           > ", WORDSZ, time );
/* Get the epoch corresponding to the input time: */
str2et_c ( time, &et );
/* Get the radii of the satellite. */
bodvrd_c ( satnm, "RADII", 3, &i, radii );
/* Get the instrument boresight and frame name. */
bodn2c_c ( instnm, &instid, &found );
if ( !found )
{
    setmsg_c ( "Instrument name # could not be "
                "translated to an ID code." );
    errch_c ( "#", instnm );
    sigerr_c ( "NAMENOTFOUND" );
}
getfov_c ( instid, ROOM,     WORDSZ,   WORDSZ,
            shape,  iframe,   insite,  &n,       bundry );
```



# Complete Source Code - 3

## Navigation and Ancillary Information Facility

```
/* Compute the boresight ray intersection with the surface of the
   target body. */  
  
sincpt_c ( "Ellipsoid", satnm, et, fixref, "CN+S", scnm,
            iframe, insite, point, &trgepc, srfvec, &found );  
  
/* If an intercept is found, compute planetocentric and planetodetic
   latitude and longitude of the point. */  
  
if ( found )  
{  
    reclat_c ( point, &r, &pclon, &pclat );  
    /* Let re, rp, and f be the satellite's longer equatorial
       radius, polar radius, and flattening factor. */  
    re = radii[0];  
    rp = radii[2];  
    f = ( re - rp ) / re;  
    recgeo_c ( point, re, f, &pdlon, &pdlat, &alt );  
    /* Compute illumination angles at the surface point. */  
    ilumin_c ( "Ellipsoid", satnm, et, fixref, "CN+S", scnm, point,
               &trgepc, srfvec, &phase, &solar, &emissn );  
    /* Display results. Convert angles to degrees for output. */  
    printf ( "\n"  
            "Intercept planetocentric longitude      (deg) : %11.6f\n"  
            "Intercept planetocentric latitude       (deg) : %11.6f\n"
```

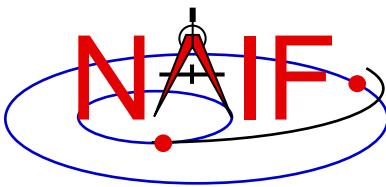


# Complete Source Code - 4

## Navigation and Ancillary Information Facility

```
"Intercept planetodetic longitude          (deg) : %11.6f\n"
"Intercept planetodetic latitude           (deg) : %11.6f\n"
"Range from spacecraft to intercept point (km) : %11.6f\n"
"Intercept phase angle                   (deg) : %11.6f\n"
"Intercept solar incidence angle         (deg) : %11.6f\n"
"Intercept emission angle               (deg) : %11.6f\n",
dpr_c() * pclon,
dpr_c() * pclat,
dpr_c() * pdlon,
dpr_c() * pdlat,
vnorm_c( srfvec ),
dpr_c() * phase,
dpr_c() * solar,
dpr_c() * emissn
);
}

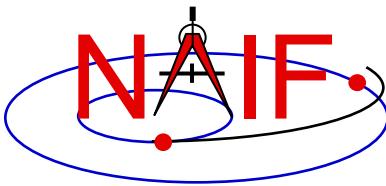
else {
    printf ( "No intercept point found at %s\n", time );
}
return(0);
}
```



# Compile and Link the Program - 1

Navigation and Ancillary Information Facility

- **First be sure that both the CSPICE Toolkit and a C compiler are properly installed.**
  - A "hello world" C program must be able to compile, link, and run successfully in your environment.
  - Any of the mkprodct.\* scripts in the cspice/src/\* paths of the CSPICE installation should execute properly.
- **Ways to compile and link the program:**
  - If you're familiar with the "make" utility, create a makefile. Use compiler and linker options from the mkprodct.\* script found in the cspice/src/cook\_c path of your CSPICE installation.
  - Or, modify the cookbook mkprodct.\* build script.
    - » Your program name must be \*.pgm, for example demo.pgm, to be recognized by the script.
    - » Change the library references in the script to use absolute pathnames.
    - » Change the path for the executable to the current working directory.
    - » If your compiler supports it, add a –I option to reference the cspice/include path to make CSPICE \*.h files available. Otherwise, copy those files from the include path to your current working directory.
    - » On some platforms, you must modify the script to refer to your program by name.



# Compile and Link the Program - 2

Navigation and Ancillary Information Facility

- Or, compile the program on the command line. The program must be linked against the CSPICE object library `cspice.a` (`cspice.lib` under MS Visual C++/C) and the C math library. On a PC running Linux and `gcc`, if

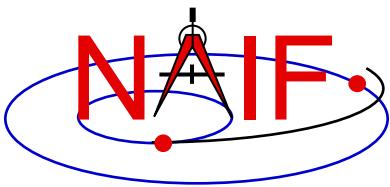
- » The `gcc` compiler is in your path
    - As indicated by the response to the command "which `gcc`"
  - » the Toolkit is installed in the path (for the purpose of this example) `/myhome/cspice`
  - » You've named the program `demo.c`

then you can compile and link your program using the command

```
» gcc -I/myhome/cspice/include \
      -o demo \
      demo.c /myhome/cspice/lib/cspice.a -lm
```

- Note: the preprocessor flag  
`-DNON_UNIX_STDIO`

used in the `mkprodct.csh` script is needed for code generated by `f2c`, but is usually unnecessary for compiling user code.

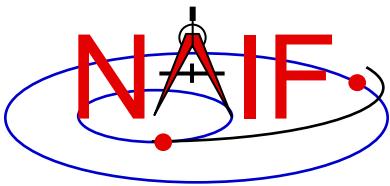


# Compile and Link the Program - 3

## Navigation and Ancillary Information Facility

Terminal Window

```
Prompt> mkprodct.csh
        Setting default compiler:
        gcc
        Setting default compile options:
        -c -ansi -O2 -DNON_UNIX_STDIO
        Setting default link options:
        -lm
        Compiling and linking: demo.pgm
        Compiling and linking: demo.pgm
Prompt>
```



# Running the Program - 1

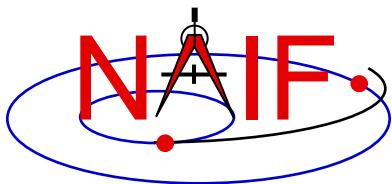
---

Navigation and Ancillary Information Facility

It looks like we have everything taken care of:

- We have all necessary kernels
- We made a setup file (metakernel) pointing to them
- We wrote the program
- We compiled and linked it

Let's run it.



# Running the Program - 2

## Navigation and Ancillary Information Facility

```
Terminal Window

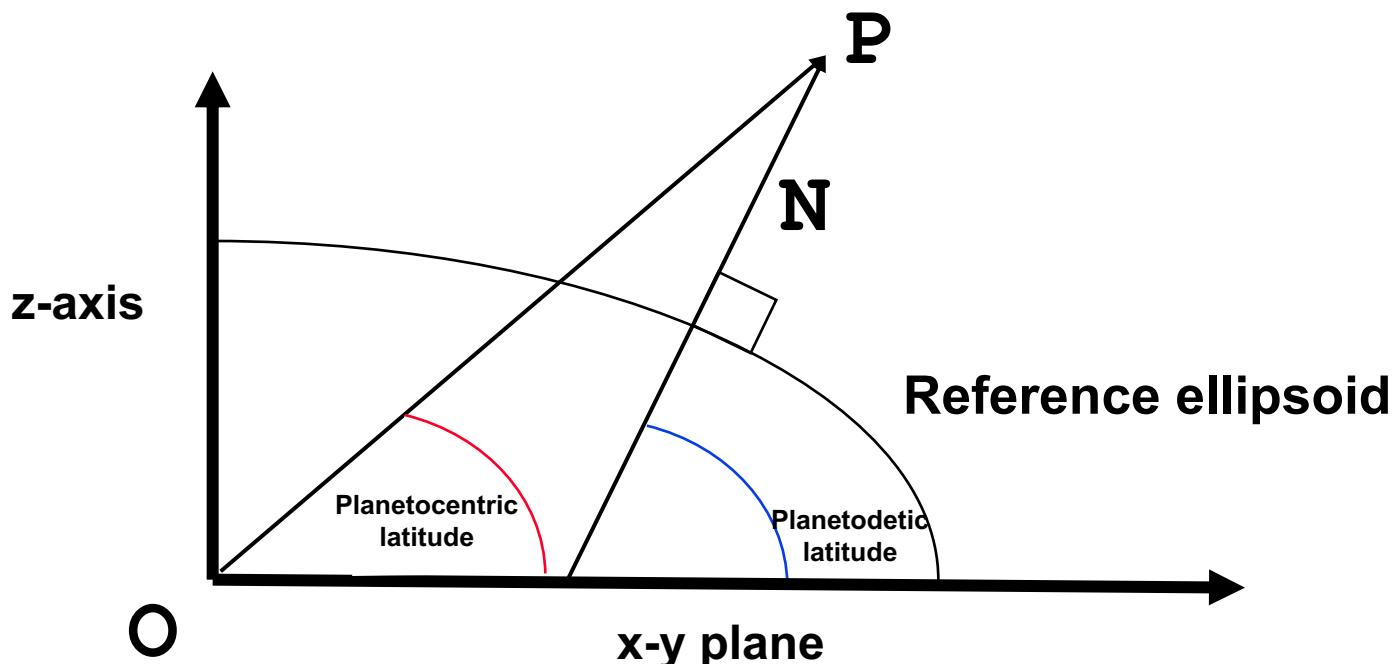
Prompt> demo

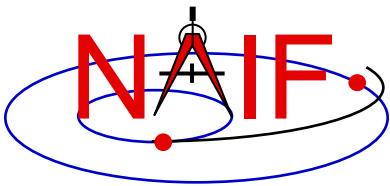
Enter setup file name > setup.ker
Enter satellite name > PHOEBE
Enter satellite frame > IAU_PHOEBE
Enter spacecraft name > CASSINI
Enter instrument name > CASSINI_ISS_NAC
Enter time > 2004 jun 11 19:32:00

Intercept planetocentric longitude (deg) : 39.843719
Intercept planetocentric latitude (deg) : 4.195878
Intercept planetodetic longitude (deg) : 39.843719
Intercept planetodetic latitude (deg) : 5.048011
Range from spacecraft to intercept point (km) : 2089.169724
Intercept phase angle (deg) : 28.139479
Intercept solar incidence angle (deg) : 18.247220
Intercept emission angle (deg) : 17.858309
Prompt>
```

- **Latitude definitions:**

- Planetocentric latitude of a point P: angle between segment from origin to point and x-y plane (red arc in diagram).
- Planetodetic latitude of a point P: angle between x-y plane and extension of ellipsoid normal vector N that connects x-y plane and P (blue arc in diagram).



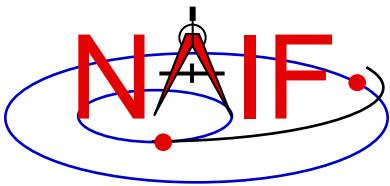


---

Navigation and Ancillary Information Facility

# Writing a SPICE (FORTRAN) Based Program

January 2020

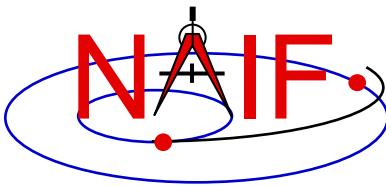


# Viewing This Tutorial

---

Navigation and Ancillary Information Facility

**Undefined variables are displayed in red**  
**Results are displayed in blue**



# Introduction

---

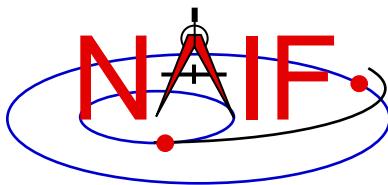
Navigation and Ancillary Information Facility

First, let's go over the important steps in the process of writing a SPICE-based Fortran program and putting it to work:

- Understand the geometry problem.
- Identify the set of SPICE kernels that contain the data needed to perform the computation.
- Select the SPICE APIs needed to compute the quantities of interest.
- Write and compile the program.
- Get actual kernel files and verify that they contain the data needed to support the computation for the time(s) of interest.
- Run the program.

To illustrate these steps, let's write a program that computes the apparent intersection of the boresight ray of a given CASSINI science instrument with the surface of a given Saturnian satellite. The program will compute

- Planetocentric and planetodetic (geodetic) latitudes and longitudes of the intercept point.
- Range from spacecraft to intercept point.
- Illumination angles (phase, solar incidence, and emission) at the intercept point.



# Observation geometry

Navigation and Ancillary Information Facility

We want the boresight intercept on the surface, range from s/c to intercept, and illumination angles at the intercept point.

When? **TIME (UTC, TDB or TT)**

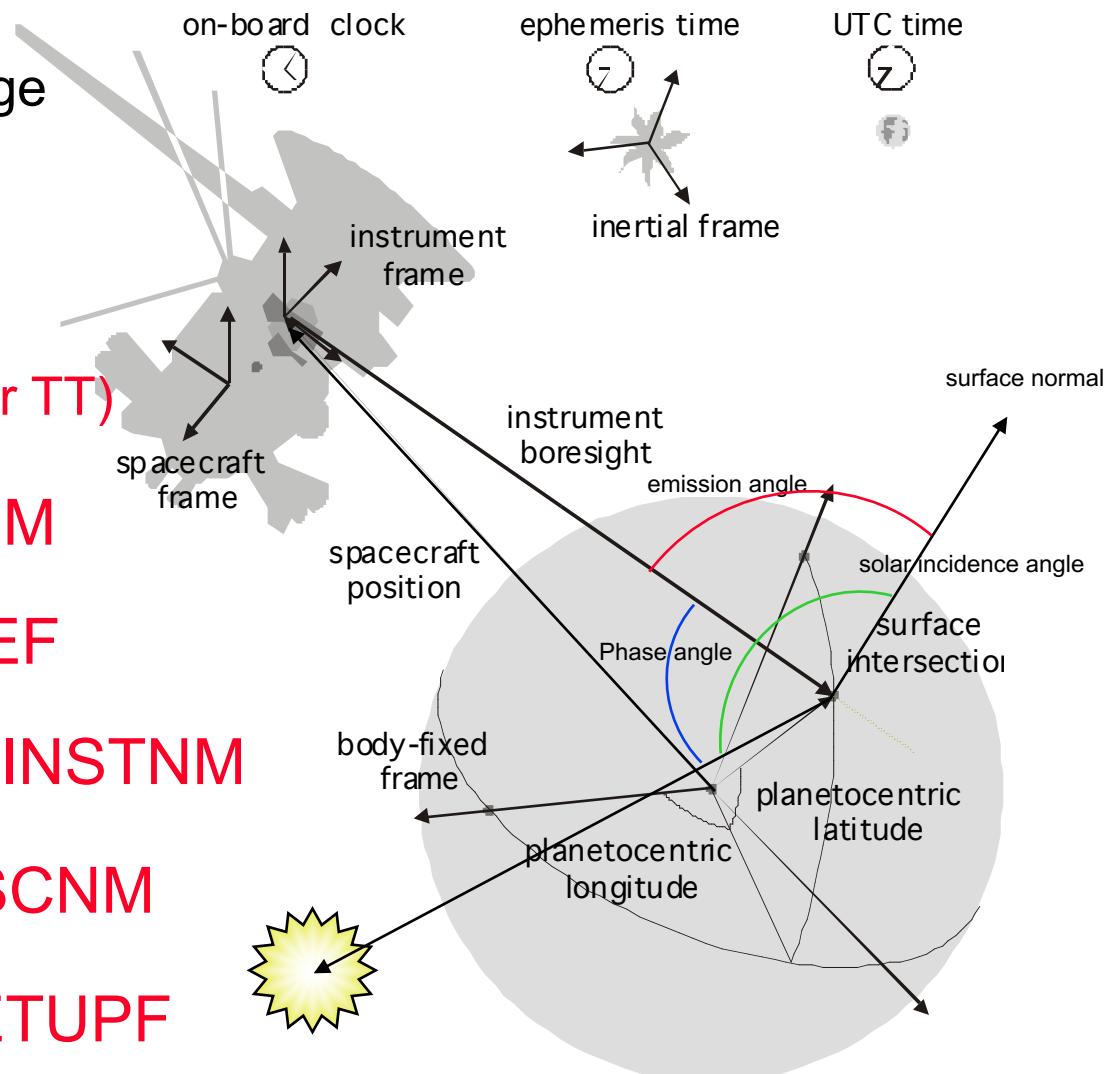
On what object? **SATNM**

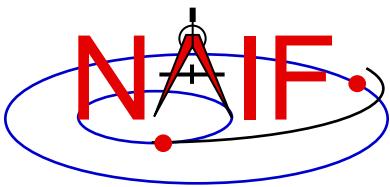
In what frame? **FIXREF**

For which instrument? **INSTNM**

For what spacecraft? **SCNM**

Using what model? **SETUPF**





# Needed Data

Navigation and Ancillary Information Facility

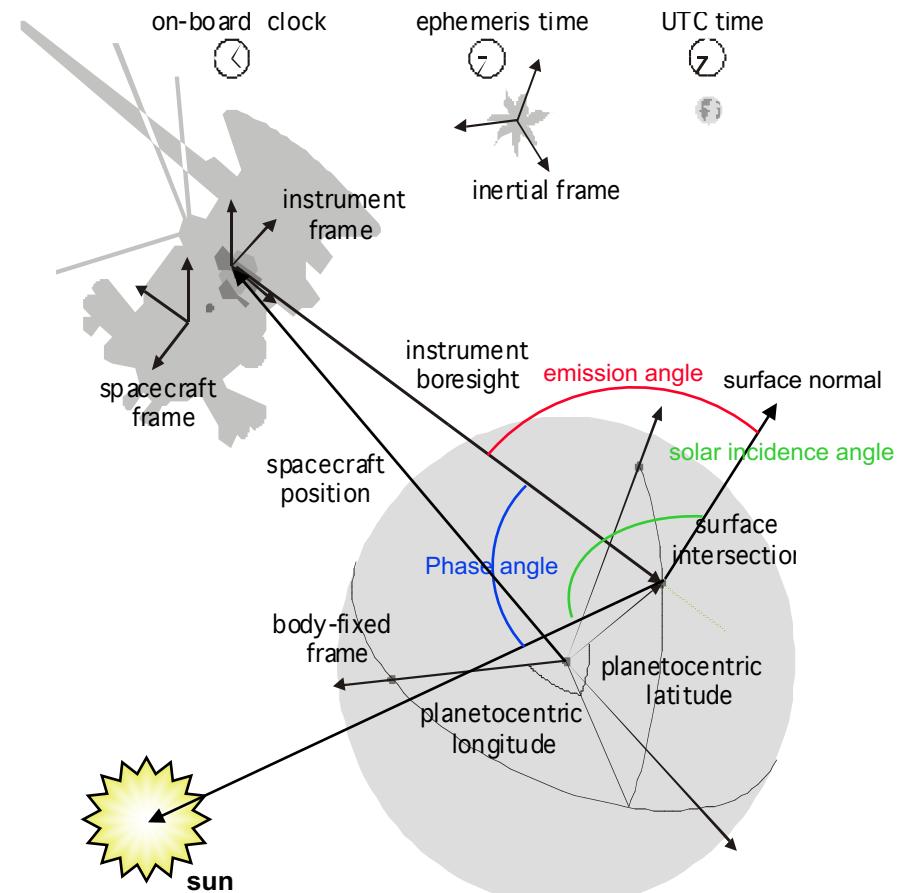
Time transformation kernels

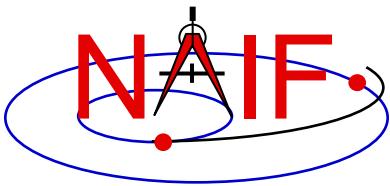
Orientation models

Instrument descriptions

Shapes of satellites, planets

Ephemerides for spacecraft,  
Saturn barycenter and satellites.





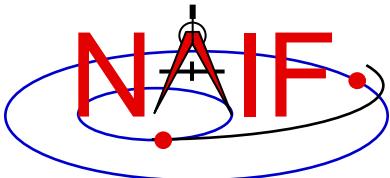
# Which Kernels are Needed?

Navigation and Ancillary Information Facility

Data required to compute vectors, rotations and other parameters shown in the picture are stored in the SPICE kernels listed below.

Note: these kernels have been selected to support this presentation; they should not be assumed to be appropriate for user applications.

Parameter	Kernel Type	File name
time conversions	generic LSK	naif0009.tls
satellite orientation	CASSINI SCLK	cas00084.tsc
satellite shape	CASSINI PCK	cpck05Mar2004.tpc
satellite position	CASSINI PCK	cpck05Mar2004.tpc
	planet/sat	
planet barycenter position	ephemeris SPK	020514_SE_SAT105.bsp
spacecraft position	planet SPK	981005_PLTEPH-DE405S.bsp
spacecraft orientation	spacecraft SPK	030201AP_SK_SM546_T45.bsp
instrument alignment	spacecraft CK	04135_04171pc_psiv2.bc
instrument boresight	CASSINI FK	cas_v37.tf
	Instrument IK	cas_iss_v09.ti



# Load Kernels

---

Navigation and Ancillary Information Facility

The easiest and most flexible way to make required kernels available to the program is via FURNSH. For this example we make a setup file (also called a “metakernel” or “furnsh kernel”) containing a list of kernels to be loaded:

Note: these kernels have been selected to support this presentation they should not be assumed to be appropriate for user applications.

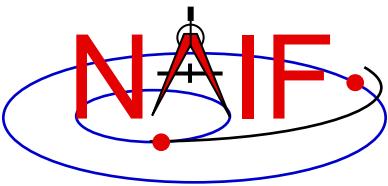
```
\begindata
```

```
KERNELS_TO_LOAD = ('naif0009.tls', 'cas00084.tsc', 'cpck05Mar2004.tpc',
                    '020514_SE_SAT105.bsp', '981005_PLTEPH-DE405S.bsp',
                    '030201AP_SK_SM546_T45.bsp', '04135_04171pc_psiv2.bc',
                    'cas_v37.tf', 'cas_iss_v09.ti')
```

```
\begintext
```

and we make the program prompt for the name of this setup file:

```
CALL PROMPT ( 'Enter setup file name > ', SETUPF )
CALL FURNSH ( SETUPF )
```



# Programming Solution

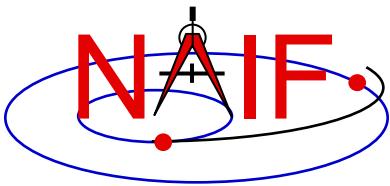
Navigation and Ancillary Information Facility

- Prompt for setup file (“metakernel”) name load kernels specified via setup file. (Done on previous chart.)
- Prompt for user inputs required to completely specify problem. Obtain further inputs required by geometry routines via SPICELIB calls.
- Compute the intersection of the boresight direction ray with the surface of the satellite, presented as a triaxial ellipsoid.

If there is an intersection,

- Convert Cartesian coordinates of the intercept point to planetocentric latitudinal and planetodetic coordinates
- Compute spacecraft-to-intercept point range
- Find the illumination angles (phase, solar incidence, and emission) at the intercept point
- Display the results.

We discuss the geometric portion of the problem next.



# Compute Surface Intercept and Ranges

Navigation and Ancillary Information Facility

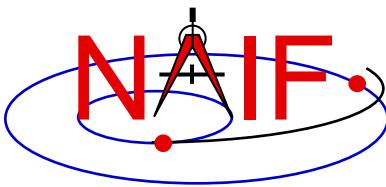
Compute the intercept point (`POINT`) of the boresight vector (`INSITE`) specified in the instrument frame (`IFRAME`) of the instrument mounted on the spacecraft (`SCNM`) with the surface of the satellite (`SATNM`) at the TDB time of interest (`ET`) in the satellite's body-fixed frame (`FIXREF`). This call also returns the light-time corrected epoch at the intercept point (`TRGEPC`), the spacecraft-to-intercept point vector (`SRFVEC`), and a flag indicating whether the intercept was found (`FOUND`). We use "converged Newtonian" light time plus stellar aberration corrections to produce the most accurate surface intercept solution possible. We model the surface of the satellite as an ellipsoid.

```
CALL SINCPT ( 'Ellipsoid', SATNM, ET, FIXREF, 'CN+S', SCNM, IFRAME,  
             INSITE, POINT, TRGEPC, SRFVEC, FOUND )  
. . .
```

The range we want is obtained from the outputs of `SINCPT`. These outputs are defined only if a surface intercept is found. If `FOUND` is true, the spacecraft-to-surface intercept range is the norm of the output argument `SRFVEC`. Units are km. We use the SPICELIB function `VNORM` to obtain the norm:

```
VNORM ( SRFVEC )
```

We'll write out the range data along with the other program results.



# Compute Lat/Lon and Illumination Angles

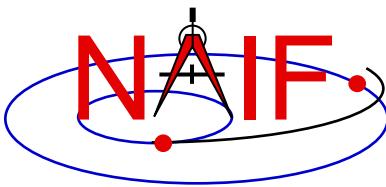
Navigation and Ancillary Information Facility

Compute the planetocentric latitude ([PCLAT](#)) and longitude ([PCLON](#)), as well as the planetodetic latitude ([PDLAT](#)) and longitude ([PDLON](#)) of the intersection point.

```
IF ( FOUND ) THEN
    CALL RECLAT ( POINT, R, PCLON, PCLAT )
C      Let RE, RP, and F be the satellite's longer equatorial
C      radius, polar radius, and flattening factor.
RE   =  RADII(1)
RP   =  RADII(3)
F    =  ( RE - RP ) / RE
CALL RECgeo ( POINT, RE, F, PDLON, PDLAT, ALT )
```

The illumination angles we want are the outputs of [ILUMIN](#). Units are radians.

```
CALL ILUMIN ( 'Ellipsoid', SATNM, ET, FIXREF,
.           'CN+S', SCNM, POINT, TRGEPC, SRFVEC,
.           PHASE, SOLAR, EMISSN )
```



# Geometry Calculations: Summary

## Navigation and Ancillary Information Facility

C Compute the boresight ray intersection with the surface of the  
C target body.

```
CALL SINCPT ( 'Ellipsoid', SATNM, ET, FIXREF, 'CN+S', SCNM, IFRA  
M,  
. INSITE, POINT, TRGEPC, SRFVEC, FOUND )
```

C If an intercept is found, compute planetocentric and planetodetic  
C latitude and longitude of the point.

```
IF( FOUND ) THEN
```

```
    CALL RECLAT ( POINT, R, PCLON, PCLAT )
```

C Let RE, RP, and F be the satellite's longer equatorial  
C radius, polar radius, and flattening factor.

```
    RE = RADII(1)
```

```
    RP = RADII(3)
```

```
    F = ( RE - RP ) / RE
```

```
    CALL RECgeo ( POINT, RE, F, PDLON, PDLAT, ALT )
```

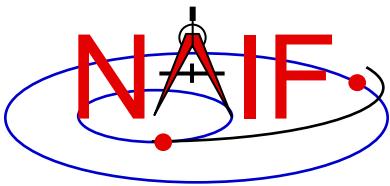
C Compute illumination angles at the surface point.

```
    CALL ILUMIN ( 'Ellipsoid', SATNM, ET, FIXREF, 'CN+S', SCNM,  
M,  
. POINT, TRGEPC, SRFVEC, PHASE, SOLAR, EMISSN )
```

```
    ...
```

```
ELSE
```

```
    ...
```



# Get Inputs - 1

---

Navigation and Ancillary Information Facility

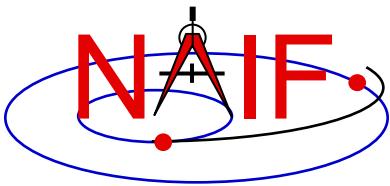
The code above used quite a few inputs that we don't have yet:

- TDB epoch of interest ( `ET` )
- satellite and s/c names ( `SATNM`, `SCNM` )
- satellite body-fixed frame name ( `FIXREF` )
- satellite ellipsoid radii ( `RADI` )
- instrument fixed frame name ( `IFRAME` )
- instrument boresight vector in the instrument frame ( `INSITE` )

Some of these values are user inputs others can be obtained via SPICELIB calls once the required kernels have been loaded.

Let's prompt for the satellite name ( `SATNM` ), satellite frame name ( `FIXREF` ), spacecraft name ( `SCNM` ), instrument name ( `INSTNM` ) and time of interest ( `TIME` ):

```
CALL PROMPT ( 'Enter satellite name > ', SATNM )
CALL PROMPT ( 'Enter satellite frame > ', FIXREF )
CALL PROMPT ( 'Enter spacecraft name > ', SCNM )
CALL PROMPT ( 'Enter instrument name > ', INSTNM )
CALL PROMPT ( 'Enter time > ', TIME )
```



# Get Inputs - 2

---

Navigation and Ancillary Information Facility

Then we can get the rest of the inputs from SPICELIB calls:

To get the TDB epoch (**ET**) from the user-supplied time string (which may refer to the UTC, TDB or TT time systems):

```
CALL STR2ET ( TIME, ET )
```

To get the satellite's ellipsoid radii (**RADII**):

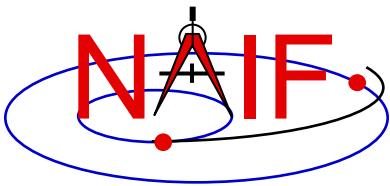
```
CALL BODVRD ( SATNM, 'RADII', 3, I, RADII )
```

To get the instrument boresight direction (**INSITE**) and the name of the instrument frame (**IFRAME**) in which it is defined:

```
CALL BODN2C ( INSTNM, INSTID, FOUND )

IF ( .NOT. FOUND ) THEN
    CALL SETMSG ( 'Instrument name # could not be ' //
                  'translated to an ID code.' )
    .
    CALL ERRCH ( '#', INSTNM )
    CALL SIGERR ( 'NAMENOTFOUND' )
END IF

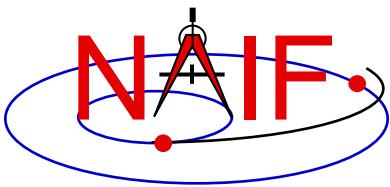
CALL GETFOV ( INSTID, ROOM, SHAPE, IFRAME,
              INSITE, N, BUNDRY )
```



# Getting Inputs: Summary

## Navigation and Ancillary Information Facility

```
C      Prompt for the user-supplied inputs for our program.  
CALL PROMPT ( 'Enter setup file name > ', SETUPF )  
CALL FURNSH ( SETUPF )  
CALL PROMPT( 'Enter satellite name > ', SATNM )  
CALL PROMPT( 'Enter satellite frame > ', FIXREF )  
CALL PROMPT( 'Enter spacecraft name > ', SCNM )  
CALL PROMPT( 'Enter instrument name > ', INSTNM )  
CALL PROMPT( 'Enter time           > ', TIME )  
  
C      Get the epoch corresponding to the input time:  
CALL STR2ET ( TIME, ET )  
  
C      Get the radii of the satellite.  
CALL BODVRD ( SATNM, 'RADII', 3, I, RADII )  
  
C      Get the instrument boresight and frame name.  
CALL BODN2C ( INSTNM, INSTID, FOUND )  
  
IF ( .NOT. FOUND ) THEN  
    CALL SETMSG ( 'Instrument name # could not be ' //  
                 .                           'translated to an ID code.'      )  
    CALL ERRCH  ( '#', INSTNM          )  
    CALL SIGERR ( 'NAMENOTFOUND'      )  
END IF  
  
CALL GETFOV ( INSTID, ROOM, SHAPE, IFRAME,  
             .               INSITE, N, BUNDY )
```

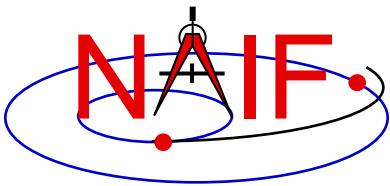


# Display Results

## Navigation and Ancillary Information Facility

```
C      Display results. Convert angles from radians to degrees
C      for output.
      WRITE ( *, '(1X,A,F12.6)' )
      .  'Intercept planetocentric longitude          (deg) : ', DPR() *PCLON
      WRITE ( *, '(1X,A,F12.6)' )
      .  'Intercept planetocentric latitude           (deg) : ', DPR() *PCLAT
      WRITE ( *, '(1X,A,F12.6)' )
      .  'Intercept planetodetic longitude          (deg) : ', DPR() *PDLON
      WRITE ( *, '(1X,A,F12.6)' )
      .  'Intercept planetodetic latitude           (deg) : ', DPR() *PDLAT
      WRITE ( *, '(1X,A,F12.6)' )
      .  'Range from spacecraft to intercept point (km) : ',
      .  VNORM(SRFVEC)
      WRITE ( *, '(1X,A,F12.6)' )
      .  'Intercept phase angle                      (deg) : ', DPR() *PHASE
      WRITE ( *, '(1X,A,F12.6)' )
      .  'Intercept solar incidence angle          (deg) : ', DPR() *SOLAR
      WRITE ( *, '(1X,A,F12.6)' )
      .  'Intercept emission angle                 (deg) : ',
      .  DPR() *EMISSN

      ELSE
        WRITE (*,*) 'No intercept point found at '// TIME
      END IF
```



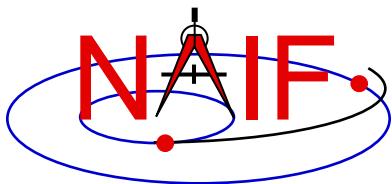
# Complete the Program

---

Navigation and Ancillary Information Facility

To finish up the program we need to declare the variables we've used.

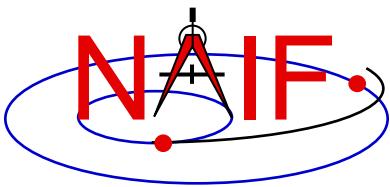
- We'll highlight techniques used by NAIF programmers
- Add remaining Fortran code required to make a syntactically valid program



# Complete Source Code - 1

## Navigation and Ancillary Information Facility

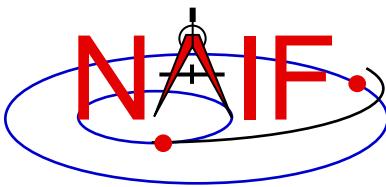
PROGRAM PROG26			
IMPLICIT NONE			
DOUBLE PRECISION	DPR	DOUBLE PRECISION	EMISSN
DOUBLE PRECISION	VNORM	DOUBLE PRECISION	ET
		DOUBLE PRECISION	F
		DOUBLE PRECISION	INSITE (3)
		DOUBLE PRECISION	SRFVEC (3)
		DOUBLE PRECISION	PCLAT
INTEGER	FILESZ	DOUBLE PRECISION	PCLON
PARAMETER	( FILESZ = 255 )	DOUBLE PRECISION	PDLAT
INTEGER	WORDSZ	DOUBLE PRECISION	PDLON
PARAMETER	( WORDSZ = 40 )	DOUBLE PRECISION	PHASE
INTEGER	ROOM	DOUBLE PRECISION	POINT (3)
PARAMETER	( ROOM = 10 )	DOUBLE PRECISION	R
		DOUBLE PRECISION	RADII (3)
CHARACTER* (WORDSZ)	IFRAME	DOUBLE PRECISION	RE
CHARACTER* (WORDSZ)	INSTNM	DOUBLE PRECISION	RP
CHARACTER* (WORDSZ)	SATNM	DOUBLE PRECISION	SOLAR
CHARACTER* (WORDSZ)	FIXREF	DOUBLE PRECISION	TRGEPC
CHARACTER* (WORDSZ)	SCNM		
CHARACTER* (FILESZ)	SETUPF		
CHARACTER* (WORDSZ)	SHAPE	INTEGER	I
CHARACTER* (WORDSZ)	TIME	INTEGER	INSTID
		INTEGER	N
DOUBLE PRECISION	ALT		
DOUBLE PRECISION	BUNDRY (3 , ROOM)	LOGICAL	FOUND



# Complete Source Code - 2

## Navigation and Ancillary Information Facility

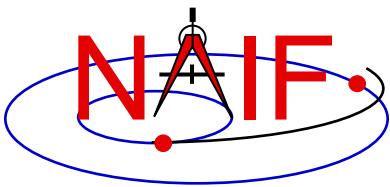
```
C   Prompt for the user-supplied inputs for our program.  
CALL PROMPT ( 'Enter setup file name > ', SETUPF )  
CALL FURNSH ( SETUPF )  
CALL PROMPT ( 'Enter satellite name > ', SATNM )  
CALL PROMPT ( 'Enter satellite frame > ', FIXREF )  
CALL PROMPT ( 'Enter spacecraft name > ', SCNM )  
CALL PROMPT ( 'Enter instrument name > ', INSTNM )  
CALL PROMPT ( 'Enter time           > ', TIME )  
  
C   Get the epoch corresponding to the input time:  
CALL STR2ET ( TIME, ET )  
  
C   Get the radii of the satellite.  
CALL BODVRD ( SATNM, 'RADII', 3, I, RADII )  
  
C   Get the instrument boresight and frame name.  
CALL BODN2C ( INSTNM, INSTID, FOUND )  
IF ( .NOT. FOUND ) THEN  
    CALL SETMSG ( 'Instrument name # could not be ' //  
                 'translated to an ID code.' )  
    CALL ERRCH ( '#', INSTNM )  
    CALL SIGERR ( 'NAMENOTFOUND' )  
END IF  
CALL GETFOV ( INSTID, ROOM, SHAPE, IFRAME,  
              INSITE, N,      BUNDARY )
```



# Complete Source Code - 3

## Navigation and Ancillary Information Facility

```
C      Compute the boresight ray intersection with the surface of the
C      target body.
CALL SINCPT ( 'Ellipsoid', SATNM, ET, FIXREF, 'CN+S', SCNM, IFRAME,
.           INSITE, POINT, TRGEPC, SRFVEC, FOUND )
C      If an intercept is found, compute planetocentric and planetodetic
C      latitude and longitude of the point.
IF( FOUND ) THEN
    CALL RECLAT ( POINT, R, PCLON, PCLAT )
C      Let RE, RP, and F be the satellite's longer equatorial
C      radius, polar radius, and flattening factor.
    RE  =  RADII(1)
    RP  =  RADII(3)
    F   =  ( RE - RP ) / RE
    CALL RECgeo ( POINT, RE, F, PDLON, PDLAT, ALT )
C      Compute illumination angles at the surface point.
    CALL ILUMIN ( 'Ellipsoid', SATNM, ET, FIXREF, 'CN+S', SCNM,
.                 POINT, TRGEPC, SRFVEC, PHASE, SOLAR, EMISSN )
C      Display results. Convert angles from radians to degrees
C      for output.
    WRITE ( *, * )
    WRITE ( *, '(1X,A,F12.6)' )
.     'Intercept planetocentric longitude          (deg) : ', DPR() *PCLON
```

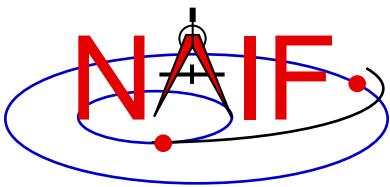


# Complete Source Code - 4

Navigation and Ancillary Information Facility

```
      WRITE ( *, '(1X,A,F12.6)' )
.     'Intercept planetocentric latitude           (deg) :  ', DPR() *PCLAT
      WRITE ( *, '(1X,A,F12.6)' )
.     'Intercept planetodetic longitude           (deg) :  ', DPR() *PDLON
      WRITE ( *, '(1X,A,F12.6)' )
.     'Intercept planetodetic latitude            (deg) :  ', DPR() *PDLAT
      WRITE ( *, '(1X,A,F12.6)' )
.     'Range from spacecraft to intercept point (km) :  ',
.     VNORM(SRFVEC)
      WRITE ( *, '(1X,A,F12.6)' )
.     'Intercept phase angle                      (deg) :  ', DPR() *PHASE
      WRITE ( *, '(1X,A,F12.6)' )
.     'Intercept solar incidence angle          (deg) :  ', DPR() *SOLAR
      WRITE ( *, '(1X,A,F12.6)' )
.     'Intercept emission angle                  (deg) :  ',
.     DPR() *EMISSN

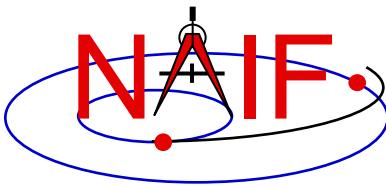
ELSE
      WRITE (*,*) 'No intercept point found at '// TIME
END IF
END
```



# Compile and Link the Program - 1

Navigation and Ancillary Information Facility

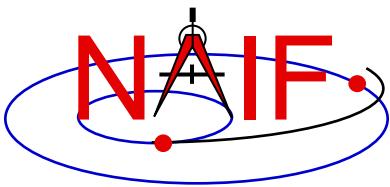
- **First be sure that both the SPICE Toolkit and a Fortran compiler are properly installed.**
  - A "hello world" Fortran program must be able to compile, link, and run successfully in your environment.
  - Any of the mkprodct.\* scripts in the toolkit/src/\* paths of the SPICE Toolkit installation should execute properly.
- **Ways to compile and link the program:**
  - If you're familiar with the "make" utility, create a makefile. Use compiler and linker options from the mkprodct.\* script found in the toolkit/src/cookbook path of your SPICE Toolkit installation.
  - Or, modify the cookbook mkprodct.\* build script.
    - » Your program name must be \*.pgm, for example demo.pgm, to be recognized by the script.
    - » Change the library references in the script to use absolute pathnames.
    - » Change the path for the executable to the current working directory.
    - » On some platforms, you must modify the script to refer to your program by name.



# Compile and Link the Program - 2

Navigation and Ancillary Information Facility

- Or, compile the program on the command line. The program must be linked against the SPICELIB object library spicelib.a (spicelib.lib under MS Windows systems). On a PC running Linux and g77, if
    - » The g77 compiler is in your path
      - As indicated by the response to the command "which g77"
    - » the Toolkit is installed in the path (for the purpose of this example) /myhome/toolkit
    - » You've named the program demo.f
- then you can compile and link your program using the command
- ```
» g77 -o demo demo.f \
      /myhome/toolkit/lib/spicelib.a
```



# Compile and Link the Program - 3

## Navigation and Ancillary Information Facility

```
Terminal Window

Prompt> mkprodct.csh
Using the g77 compiler.

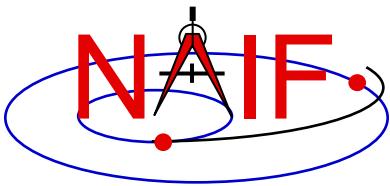
Setting default Fortran compile options:
-c -C

Setting default C compile options:
-c

Setting default link options:

Compiling and linking: demo.pgm
Compiling and linking: demo.pgm

Prompt>
```



# Running the Program - 1

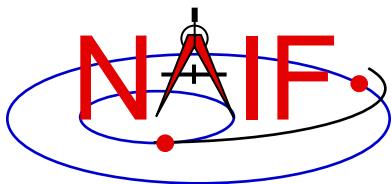
---

Navigation and Ancillary Information Facility

It looks like we have everything taken care of:

- We have all necessary kernels
- We made a setup file (metakernel) pointing to them
- We wrote the program
- We compiled and linked it

Let's run it.



# Running the Program - 2

## Navigation and Ancillary Information Facility

```
Terminal Window

Prompt> demo

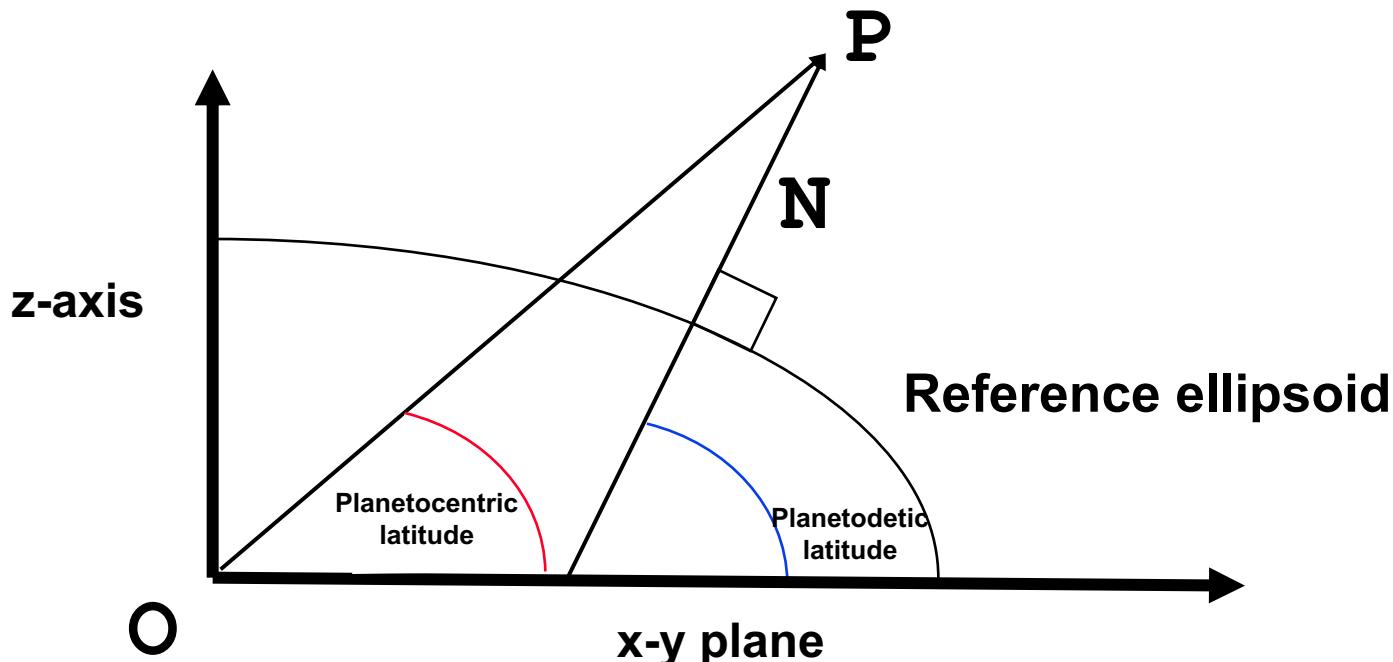
Enter setup file name > setup.ker
Enter satellite name > PHOEBE
Enter satellite frame > IAU_PHOEBE
Enter spacecraft name > CASSINI
Enter instrument name > CASSINI_ISS_NAC
Enter time > 2004 jun 11 19:32:00

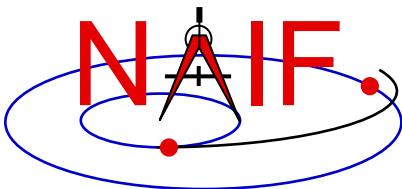
Intercept planetocentric longitude (deg) : 39.843719
Intercept planetocentric latitude (deg) : 4.195878
Intercept planetodetic longitude (deg) : 39.843719
Intercept planetodetic latitude (deg) : 5.048011
Range from spacecraft to intercept point (km) : 2089.169724
Intercept phase angle (deg) : 28.139479
Intercept solar incidence angle (deg) : 18.247220
Intercept emission angle (deg) : 17.858309

Prompt>
```

- **Latitude definitions:**

- Planetocentric latitude of a point P: angle between segment from origin to point and x-y plane (red arc in diagram).
- Planetodetic latitude of a point P: angle between x-y plane and extension of ellipsoid normal vector N that connects x-y plane and P (blue arc in diagram).



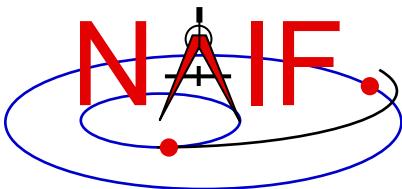


---

Navigation and Ancillary Information Facility

# Making an SPK File

January 2020

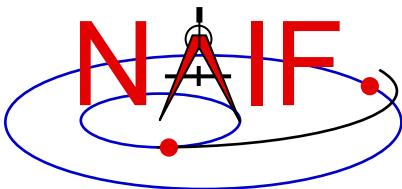


# Table of Contents

---

Navigation and Ancillary Information Facility

- **Purpose**
- **Scope**
- **Assumptions about user's knowledge**
- **SPK overview**
- **Summary of SPK architecture**
- **Discussion applicable to all production methods**
  - Recommended SPK types
- **Selecting the polynomial degree (for polynomial SPK types)**
- **SPK production methods**
  - Using the “Make SPK” (MKSPK) program
  - Using SPICELIB, CSPICE or IDL writer modules (APIs)
- **Finishing up, applicable to all methods**
  - Adding comments
  - Validation
  - Merging
- **Special requirements for making SPKs to be used in DSN/SPS software for view period generation, scheduling, metric predicts generation, and related functions.**
  - Applies only to those entities not making JPL NAV-style “p-files”
- **Issues affecting performance (reading efficiency) and usability**

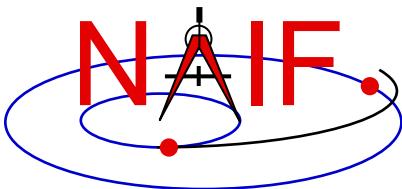


# Purpose

---

Navigation and Ancillary Information Facility

- This tutorial provides guidance for writing an SPK file using software provided by NAIF:
  - the MKSPK application program
  - or
  - SPK writer modules from SPICELIB (FORTRAN), CSPICE (C-language), Icy (IDL) or Mice (MATLAB) toolkit
    - » Only partial implementation in Icy
    - » Only Type 8 writer implemented in Mice



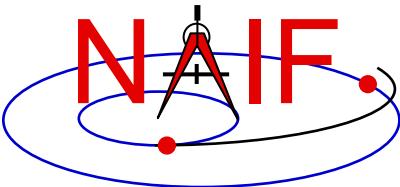
# Scope

---

Navigation and Ancillary Information Facility

- **This tutorial addresses production of SPKs:**
  - for general purposes
  - for use with NASA's Deep Space Network (DSN)
- **This tutorial does *not* address production of SPKs:**
  - by JPL navigation teams using the NIOSPK utility
    - » Those NAV teams may simply learn how to use the NIOSPK program and any useful SPK-related utilities.
  - by ESA/ESAC for its missions that use SPICE
    - » Those use custom "MEX2KER" software that was provided by NAIF
  - having TCB time tags, in support of certain IAU needs
    - » These are SPK Types 101, 102, etc.
  - use of the OEM2SPK utility, used to process CCSDS\* Orbit Ephemeris Message files
    - » See the OEM2SPK User's Guide for this information

\*CCSDS = Consultative Committee for Space Data Systems

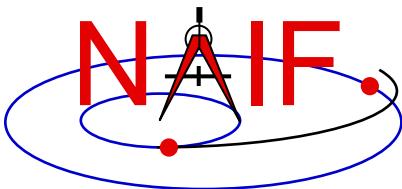


# Background Assumptions

---

Navigation and Ancillary Information Facility

- **It is assumed the reader has some familiarity with the SPICE system, and with basic ideas of orbital mechanics.**
  - The SPICE Overview tutorial is available at:  
[https://naif.jpl.nasa.gov/pub/naif/toolkit\\_docs/Tutorials/pdf/individual\\_docs/03\\_spice\\_overview.pdf](https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/Tutorials/pdf/individual_docs/03_spice_overview.pdf)
- **It is assumed the reader has read the “SPK Tutorial” that characterizes much of the SPK subsystem, with emphasis on reading SPK files.**
  - The SPK “reading” tutorial is available at:  
[https://naif.jpl.nasa.gov/pub/naif/toolkit\\_docs/Tutorials/pdf/individual\\_docs/18\\_spk.pdf](https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/Tutorials/pdf/individual_docs/18_spk.pdf)
- **It is assumed the reader has available the SPK reference document entitled “SPK Required Reading,” supplied with each copy of the SPICE Toolkit (.../doc/spk.req)**
  - SPK Required Reading is also available at:  
<https://naif.jpl.nasa.gov/naif/documentation.html>

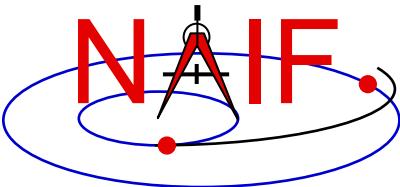


# SPK References

---

Navigation and Ancillary Information Facility

- **References for SPK production**
  - “**Making an SPK**” tutorial (this document)
  - “**SPK (Ephemeris System)**” tutorial (focused on reading an SPK)
  - “**SPK Required Reading**” technical reference (`spk.req`)
  - “**MKSPK Users Guide**” (`mkspk.ug`)
  - The source code “headers” provided as part of the SPK writer modules (subroutines)
  - “**SPKMERGE User’s Guide**” (`spkmerge.ug`)
  - “**SPY User’s Guide**” (`spy.ug`)

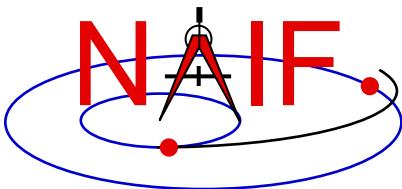


# Brief Overview - 1

---

Navigation and Ancillary Information Facility

- You should understand the physics of your data and how that relates to SPK type. For instance:
  - Type 5 works best for an orbit well approximated by a sequence of one or more conic orbits.
  - Types 9 and 13 fit data regardless of the expected physics.
    - **Caution:** a good fit in the mathematical realm may not respect the physics of the trajectory. For example, fitting polynomials to an excessively sparse set of states for a planetary orbiter could result in an interpolated path that intersects the planet.

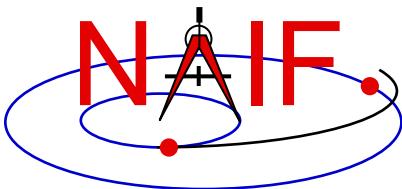


# Brief Overview - 2

---

Navigation and Ancillary Information Facility

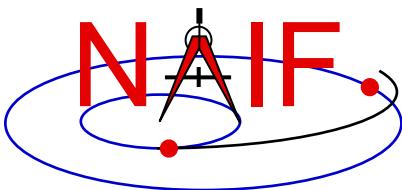
- **Ordinarily, use the NAIF MKSPK application to create SPKs from Cartesian state data or conic elements.**
  - Depending on your source data, SPK types 5, 9, 10, and 13 will satisfy the requirements for most users.
    - » Type 5, yields compact SPK files when the trajectory is well approximated by piecewise two-body motion. Type 5 may be the best choice for planetary or solar orbiters when available state data are sparse.
    - » Type 9, a good, general choice
    - » Type 13, when you have very accurate velocity data
    - » Type 10 applies ONLY to Two Line Element Sets (TLEs).
- **Alternatively, use the Toolkit's SPK writing subroutines in your own production program.**
- **Caution:** an SPK made for use by the NASA DSN has additional requirements, discussed later on.



---

Navigation and Ancillary Information Facility

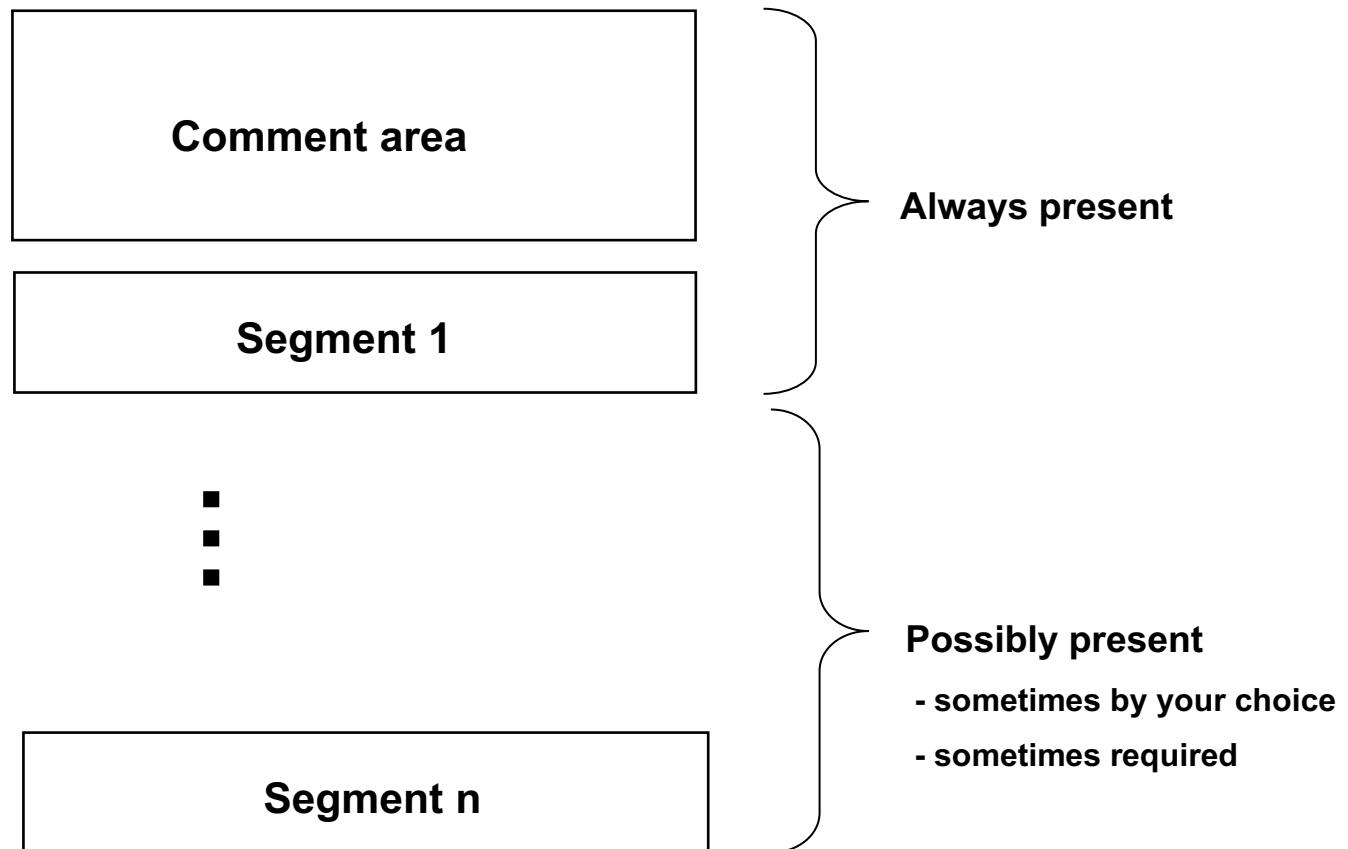
# Summary of SPK Architecture

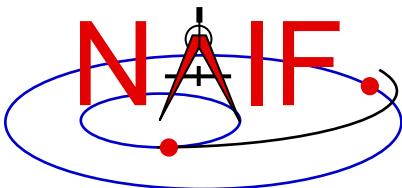


# SPK File Structure: The User's View

Navigation and Ancillary Information Facility

## Logical Organization of an SPK File





# SPK File Structure - 1

Navigation and Ancillary Information Facility

## A minimal SPK file, containing only one segment

Records are fixed-length: 1024 bytes

| ID WORD           | ND | NI | IFNAME | FWD | BWD | FREE | BFF | 0 PAD | FTP | 0 PAD |
|-------------------|----|----|--------|-----|-----|------|-----|-------|-----|-------|
| Comment area text |    |    |        |     |     |      |     |       |     |       |
| N/P/C             | D1 | U  |        |     |     |      |     | U*    |     |       |
| I1                | U  |    |        |     |     |      |     |       |     |       |
| Segment 1         |    |    |        |     |     |      |     |       |     |       |

- ▶ **File record:** One record.
- ▶ **Comment area:** Present only if used. If used, one or more records.
- ▶ **Descriptor record:** Contains 1 to 25 segment descriptors. One record.
- ▶ **Segment ID record:** Contains 1 to 25 segment IDs. One record.
- ▶ **Data segment:** One or more records.

**ID WORD:** Indicates file architecture and type

**ND, NI:** Number of d.p. and integer descriptor components

**IFNAME:** Internal file name

**FWD, BWD:** Forward and backward linked list pointers

**FREE:** First free DAF address

**BFF:** Binary file format ID

**FTP:** FTP corruption test string

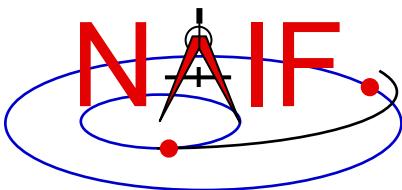
**N/P/C:** Next, previous record pointers and descriptor count

**Dn:** Descriptor for segment n

**In:** Segment ID for segment n

**U:** Unused space

**U\*:** Possibly unused space



# SPK File Structure - 2

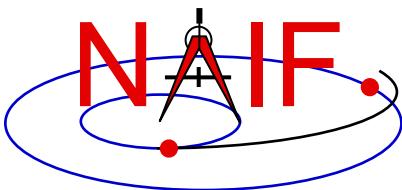
Navigation and Ancillary Information Facility

An SPK file containing multiple segments—in this example, 27

Records are fixed-length: 1024 bytes

| ID WORD           | ND  | NI  | IFNAME | FWD | BWD | FREE | BFF | 0 PAD | FTP | 0 PAD |
|-------------------|-----|-----|--------|-----|-----|------|-----|-------|-----|-------|
| Comment area text |     |     |        |     |     |      |     |       |     |       |
| N/P/C             | D1  | D2  | ...    |     |     | ...  |     |       | D25 |       |
| I1                | I2  | ... |        |     |     | ...  |     | I25   | U   |       |
| Segment 1         |     |     |        |     |     |      |     |       |     |       |
| Segment 2         |     |     |        |     |     |      |     |       |     |       |
| :                 |     |     |        |     |     |      |     |       |     |       |
| Segment 25        |     |     |        |     |     |      |     |       |     |       |
| N/P/C             | D26 | D27 | U      |     |     |      |     |       |     |       |
| I26               | I27 | U   |        |     |     |      |     |       |     |       |
| Segment 26        |     |     |        |     |     |      |     |       |     |       |
| Segment 27        |     |     |        |     |     |      |     |       |     |       |
| U*                |     |     |        |     |     |      |     |       |     |       |

- ▶ **File record:** One record
- ▶ **Comment area:** Present only if used.
- ▶ **Descriptor record:** Contains 1 to 25 segment descriptors. One record.
- ▶ **Segment ID record:** Contains 1 to 25 segment IDs. One or more records.
- ▶ **Data segments:** One or more records per segment. (Up to 25 segments.)
- ▶ **Descriptor record:** Contains 1 to 25 segment descriptors. One record.
- ▶ **Segment ID record:** Contains 1 to 25 segment IDs. One or more records.
- ▶ **Data segments:** One or more records per segment. (Up to 25 segments.)

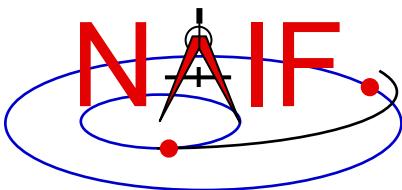


# SPK File Structure - Description

---

Navigation and Ancillary Information Facility

- **File record**
  - **Contents**
    - » Internal file name (set by file creator)
    - » Architecture and binary file format identifiers
    - » File structure parameters
    - » FTP transmission corruption detection string
  - Used by SPK reader and writer subroutines
- **Comment Area**
  - A place where “metadata” (data about data) may be placed to help a user of the SPK file understand the circumstances of its production and any recommendations about for what uses it was intended
- **Descriptor record and Segment ID record**
  - One of each of these is needed for every collection of 1-to-25 segments
- **Segment[s]**
  - **Collection[s] of ephemeris data**
    - » Minimum of one segment
    - » Maximum:
      - The practical maximum is a few thousand segments
      - Serious performance degradation occurs above 100000 segments for a single body
      - Absolute limits are imposed by the range of the INTEGER data type for your computer
  - Numerous SPK types may be used within an SPK file, but only one SPK type may appear within a given segment
  - Segments of different types may be intermixed within a given SPK file

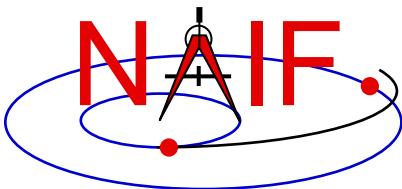


# What is an SPK Segment?

---

Navigation and Ancillary Information Facility

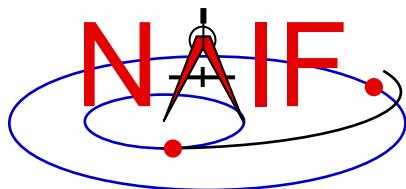
- A segment is a collection of information:
  - » providing ephemeris (position and velocity) of a single object ...
  - » given relative to a single center of motion ...
  - » specified in a single reference frame known to SPICE ...
    - Either built-in (“hard coded”) or defined in a loaded frames kernel (FK)
  - » covering a specified, continuous period of time, and
  - » using a single SPK data type
- Example: ephemeris for the Voyager 2 spacecraft, relative to the center of the Neptunian system (Neptune’s barycenter), given in the J2000 inertial reference frame, covering a specific period of time, and using Hermite interpolation with variable length intervals, produced using SPK Type 13
- An SPK segment must contain enough data to yield an object’s state at any epoch within the time bounds associated with the segment
  - This has implications that depend on the SPK type being produced



---

Navigation and Ancillary Information Facility

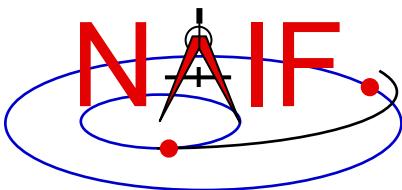
# Discussion applicable to all SPK production methods



# The SPK Family

## Navigation and Ancillary Information Facility

| Type | Description                                                                               | Notes                                                                                                                           |
|------|-------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| 1    | Modified divided difference arrays                                                        | Unique form produced at JPL; not likely to be useful to others.                                                                 |
| 2    | Chebyshev polynomials for position; fixed length time intervals.                          | Velocity is obtained by differentiation. Used at JPL for planets. Evaluates quickly.                                            |
| 3    | Chebyshev polynomials for position and velocity; fixed length time intervals              | Separate polynomial sets for position and velocity. Used at JPL for natural satellites.                                         |
| 4    | Special form used only by Hubble Space Telescope                                          | Not available for general use.                                                                                                  |
| 5    | Discrete states using weighted two-body propagation                                       | Ok if motion very closely approximates two-body motion.                                                                         |
| 6    | Special form of trigonometric expansion of elements for Phobos and Deimos                 | Not available for general use.                                                                                                  |
| 7    | Precessing elements                                                                       | Not available for general use.                                                                                                  |
| 8    | Lagrange interpolation of position and velocity; fixed length intervals between states    | Use Type 9 unless state spacing is truly uniform when measured in the TDB system.                                               |
| 9    | Lagrange interpolation of position and velocity; variable length intervals between states | Versatile type; easy to use with MKSPK.                                                                                         |
| 10   | Weighted two-line element sets (Space Command)                                            | Handles both “near-earth” and “deep space” versions.                                                                            |
| 11   | Not used                                                                                  |                                                                                                                                 |
| 12   | Hermite interpolation of position and velocity; fixed length intervals between states     | Use Type 13 unless state spacing is truly uniform when measured in the TDB system.                                              |
| 13   | Hermite interpolation of position and velocity; variable length intervals between states  | Versatile type; easy to use with MKSPK. Use for DSN support.                                                                    |
| 14   | Chebyshev polynomials for position and velocity, variable length time intervals           | The most flexible of the Chebyshev types.                                                                                       |
| 15   | Precessing conic elements propagator                                                      | ---                                                                                                                             |
| 16   | Special form used by ESA’s Infrared Space Observatory                                     | Not available for general use.                                                                                                  |
| 17   | Equinoctial elements                                                                      | Used for some satellites.                                                                                                       |
| 18   | Emulation of ESOC’s “DDID” format                                                         | Used for SMART-1, MEX, VEX, and Rosetta                                                                                         |
| 19   | Revised emulation of ESOC’s “DDID” format                                                 | Similar to Type 18, but uses “mini-segments” to reduce the number of true segments needed. (New in N65 toolkits.)               |
| 20   | Chebyshev polynomials for velocity; fixed length time intervals.                          | Provided to handle ephemerides produced by the Institute of Applied Astronomy in St. Petersburg, Russia. (New in N65 toolkits.) |
| 21   | Modified divided difference arrays                                                        | Same as Type 1 except allows greater number of coefficients.                                                                    |

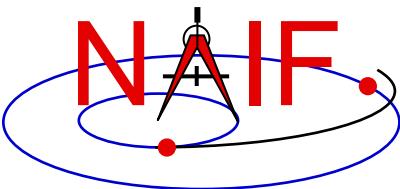


# Recommended SPK Data Types - 1

---

Navigation and Ancillary Information Facility

- **SPK type 2 (Chebyshev polynomials for position, velocity given by differentiation)** Used at JPL for planetary ephemerides.
- **SPK type 3 (Separate Chebyshev polynomials for position and velocity)** Used at JPL for satellite ephemerides.
- **SPK type 5 (Weighted two-body extrapolation)** Often used for comets and asteroids, as well as for sparse data sets where a two-body approximation is acceptable.
- **SPK types 9 and 13 (Sliding-window Lagrange and Hermite interpolation of unequally-spaced states)** Often used by users of NAIF's "Make SPK" (MKSPK) application.
- **SPK type 10 (weighted Space Command two-line element extrapolation)** Often used for earth orbiters.
- **SPK type 14 (Separate Chebyshev polynomials for position and velocity, with variable time steps)** This is the most flexible Chebyshev data type.
- **SPK type 15 (Precessing conic elements)** Provides a very compact ephemeris representation; limited to orbits where this type of approximation is valid.
- **SPK type 17 (Equinoctial elements)** Most suited for representation of ephemerides of natural satellites in equatorial or near-equatorial orbits.

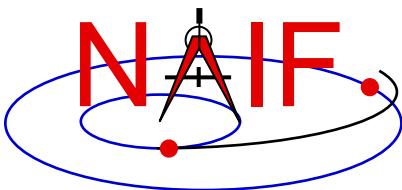


# Recommended SPK Data Types - 2

---

Navigation and Ancillary Information Facility

- **Each type has certain properties that may promote or limit its usefulness in a particular application. These properties include but are not limited to the following.**
  - » Ability to model the actual ephemeris to be represented with the accuracy required for your application.
  - » Consistency between velocity and derivative of position.
  - » Evaluation speed (performance).
  - » Compactness (file size).
  - » Availability of SPICE software needed to write files of that type.
- **Users are encouraged to consult with NAIF about the suitability of an SPK type for a particular purpose.**

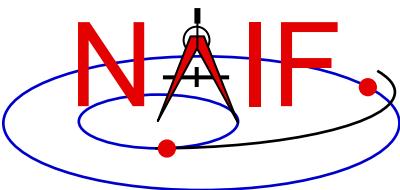


# Creating Multiple SPK Segments

---

Navigation and Ancillary Information Facility

- Each SPK segment must have a single object, center of motion, reference frame and SPK data type.
- Limiting segment size to 10,000 states or “packets of ephemeris data” can improve performance when searching within a segment.
  - Absolute limits on segment size depend on the size of the INTEGER data type.
- The total number of segments for any one body in a single SPK file must be kept under the dimension of the SPKBSR segment buffer, currently set to 100,000.
  - The total number of segments for any one body, contributed by all loaded SPK files, must be kept under the 100K limit.
  - For best efficiency, the total number of concurrently loaded segments for all bodies should be less than this 100K limit. Adherence to this criterion ensures that loaded SPK files need not be searched repeatedly for segment descriptors (which contain summary data).
  - More details about reading efficiency are provided at the end of this tutorial.
- One may elect to initiate a new segment (or more) as the means for modeling a propulsive maneuver.
  - This is because the SPK reader modules will NOT allow interpolation over a segment boundary.
- When starting a new segment you may use a new segment identifier, for instance to indicate a new trajectory leg after a maneuver.
  - Can only be done if using SPK writer modules—not if using the MKSPK application.

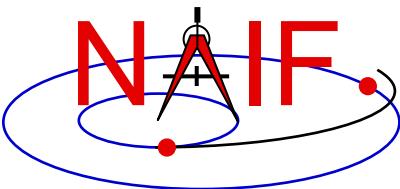


# Choosing Polynomial Degree

---

Navigation and Ancillary Information Facility

- If you make a Type 8 or 9 (Lagrange interpolation) or a Type 12 or 13 (Hermite interpolation) SPK file you must specify the degree of the interpolating polynomial that the SPK reader subroutine will use.
  - This choice needs some consideration about desired accuracy, file size and evaluation speed (performance).
  - This choice is also affected by the “smoothness” of the orbit data you wish to represent with an SPK file.
  - The allowed range of degree is 1-to-27. In addition, to ensure position and velocity continuity over the time span covered by the orbit data:
    - » for Types 8 and 9, the polynomial degree must be odd.
    - » for Types 12 and 13, the polynomial degree must be equivalent to 3-mod-4, meaning degree 3, 7, 11, 15, 19, 23 or 27.

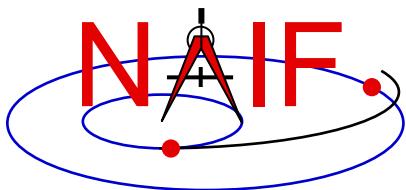


# Reference Frame

---

Navigation and Ancillary Information Facility

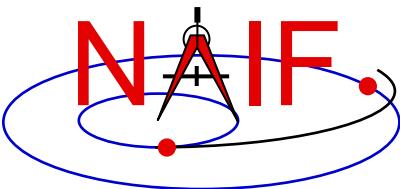
- Any reference frame known to the SPICE system, whether built-in (hard coded) or defined at run time through a Frames Kernel (FK), may be used for ephemeris data placed in an SPK file.
  - Exception: don't use a “two-vector” dynamic frame type
  - If the frame is provided via an FK, that same FK must be used when reading the SPK
- Some examples of typical reference frames used:
  - Inertial (non-rotating):
    - » J2000: (Earth Mean Equator and Equinox of J2000, a.k.a. EME2000)
    - » ECLIPJ2000: (Mean ecliptic and equinox of J2000)
  - Body-fixed (non-inertial)
    - » ITRF93: (International Terrestrial Reference Frame, for earth)
    - » MOON\_PA: (MOON\_Principal Axes)



---

Navigation and Ancillary Information Facility

# SPK Production Methods



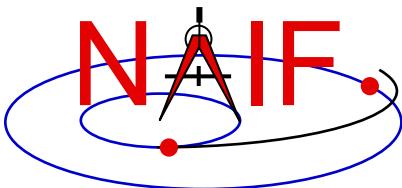
# Choices for Making an SPK File

---

Navigation and Ancillary Information Facility

- **There are two\* methods available for making an SPK file.**
  1. Take an ephemeris data file produced by your own trajectory propagator program and input this into the conversion utility (**MKSPK**) provided by NAIF that outputs an SPK file.
  2. Incorporate the appropriate SPK writer modules (subroutines) into your own code:
    - » Add these routines to your trajectory estimator/propagator.  
or...
    - » Write your own “post-processor” conversion utility, similar to **MKSPK** mentioned above.
- **Both methods are described in the next few pages.**

\*The OEM2SPK and NIOSPK specialized utilities are also methods, but are not discussed in this tutorial.

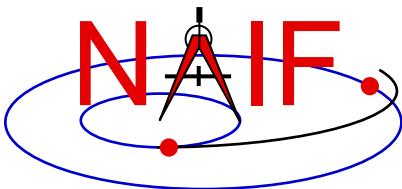


# Making Your Choice - 1

---

Navigation and Ancillary Information Facility

- **Using the MKSPK program provided in the Toolkit could be the easiest and safest for many situations.**
  - Provides considerable flexibility for accepting a wide assortment of input data formats.
  - Allows one to make multi-segment SPK files when the target, center of motion, reference frame, or SPK type changes.
    - » But not as straight forward as it could/should be.
      - Best done through multiple program executions (although one could be tricky and accomplish this in a single execution).
      - A future version of MKSPK may better accommodate this.
      - Note: production of multiple segments in type 5, 8, 9, 12 and 13 SPK files, when the amount of input data requires this, is automatically handled.

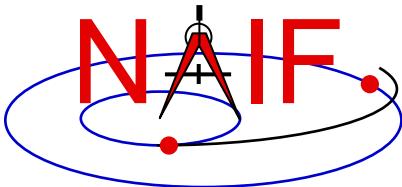


# Making Your Choice - 2

---

Navigation and Ancillary Information Facility

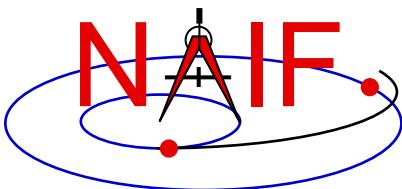
- **Using the SPK “writer” modules found in SPICELIB, CSPICE, Icy and Mice offers the greatest flexibility and user control.**
  - Using these requires that you write your own program.
  - You’ll likely need to use some additional SPICE modules as well.



---

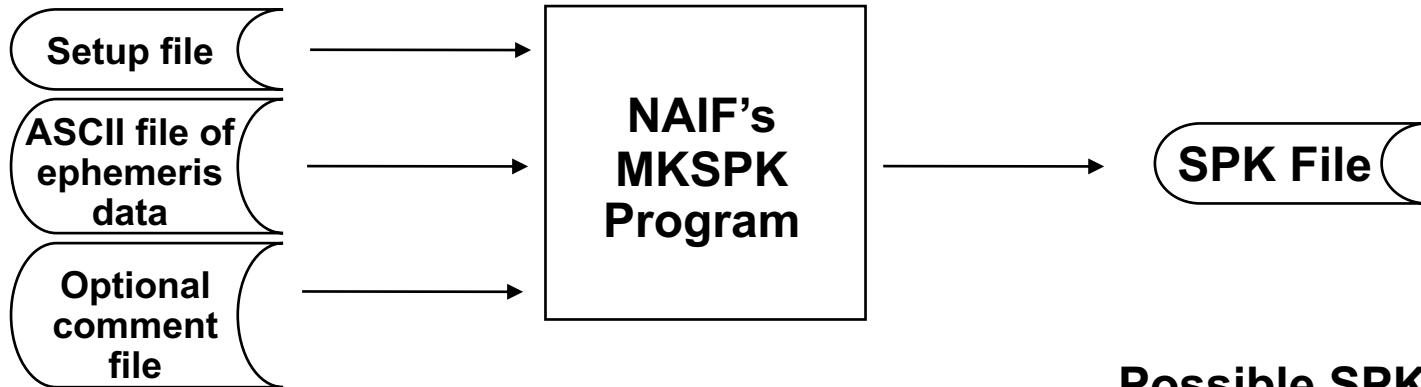
Navigation and Ancillary Information Facility

# Using NAIF's MKSPK Application Program



# Using the MKSPK Utility - 1

Navigation and Ancillary Information Facility

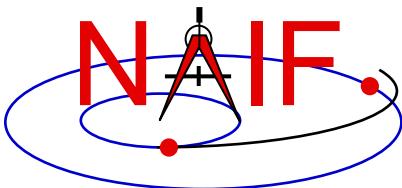


**Suitable kinds of input ephemeris data are:**

- Table of Cartesian state vectors
- Table of conic elements
- One or more sets of equinoctial elements
- One or more sets of Space Command two-line elements

**Possible SPK data types produced are:**

- Type 05
- Type 08
- Type 09
- Type 10
- Type 12
- Type 13
- Type 15
- Type 17



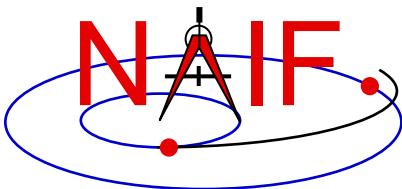
# Using the MKSPK Utility - 2

Navigation and Ancillary Information Facility

This table indicates which SPK types  
can be made from the four kinds  
of input data accepted by MKSPK

| SPK Type Produced by MKSPK -->  | 5 | 8 | 9 | 10 | 12 | 13 | 15 | 17 |
|---------------------------------|---|---|---|----|----|----|----|----|
| Input Data Type                 |   |   |   |    |    |    |    |    |
| Cartesian state vectors         | Y | Y | Y | N  | Y  | Y  | Y  | Y  |
| Conic elements                  | Y | Y | Y | N  | Y  | Y  | Y  | Y  |
| Equinoctial elements            | N | N | N | N  | N  | N  | N  | Y  |
| Space Command Two-line elements | N | N | N | Y  | N  | N  | N  | N  |

Y = Yes    N = No

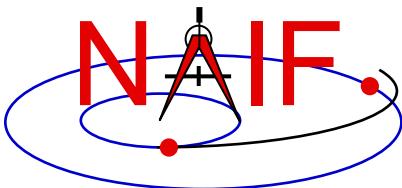


# Using the MKSPK Utility - 3

---

Navigation and Ancillary Information Facility

- **MKSPK will produce a file consisting of one or more segments as needed.**
  - It can write up to 10,000 data points in one segment.
  - For multi-segment files based on types 5, 8, 9, 12 and 13, the program will repeat sufficient data points at both sides of each interior segment boundary to ensure the SPK file will provide a continuous ephemeris through the segment boundary epoch.
- **You can use MKSPK to add a new segment to an existing SPK file.**
- **You can use SPKMERGE to merge two or more SPK files made from separate executions of MKSPK.**
  - It's important to fully understand how SPKMERGE works if you do this; see the User's Guide.

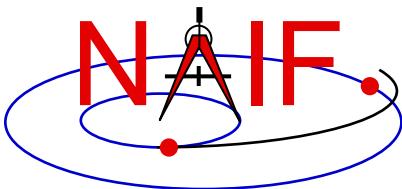


# Using the MKSPK Utility - 4

---

Navigation and Ancillary Information Facility

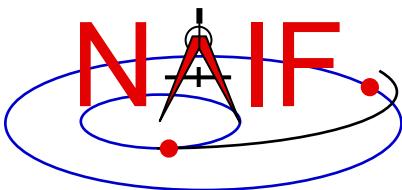
- **MKSPK does not provide direct/specific means for including propulsive maneuvers within an SPK file.**
  - Instead, use either of these two methods.
    - » Append a new SPK segment to an existing SPK file, using MKSPK.
    - » Merge a collection of SPK files, using SPKMERGE.



---

Navigation and Ancillary Information Facility

# Using SPK “Writer” Modules



# Using SPK Writer Routines

Navigation and Ancillary Information Facility

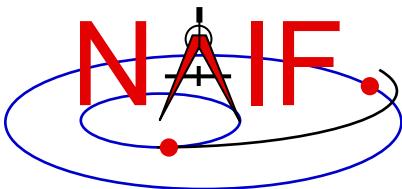
- The next several charts outline how to use the “SPK writer” modules available in the Toolkit libraries. The types available in the supported Toolkits are summarized below.\*

| SPK Types Supported      | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |
|--------------------------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|
| SPICELIB<br>(Fortran 77) | ✓ | ✓ | ✓ |   | ✓ |   |   | ✓ | ✓ | ✓  |    | ✓  | ✓  | ✓  | ✓  |    | ✓  | ✓  | ✓  | ✓  | ✓  |
| CSPICE<br>(C)            |   | ✓ | ✓ |   | ✓ |   |   | ✓ | ✓ | ✓  |    | ✓  | ✓  |    | ✓  |    | ✓  | ✓  |    | ✓  |    |
| ICY<br>(IDL)             |   | ✓ | ✓ |   | ✓ |   |   | ✓ | ✓ | ✓  |    | ✓  | ✓  |    |    |    | ✓  |    |    |    |    |
| MICE<br>(MATLAB)         |   |   |   |   |   |   |   | ✓ | ✓ | ✓  |    | ✓  | ✓  |    |    |    |    |    |    |    |    |

= 4, 6, 7, 16: made for special circumstances; not available for general use

= 11: ID was not used

\* As of Toolkit version N66



# What Routines To Use - 1

Navigation and Ancillary Information Facility

For all except SPK type 14

**SPKOPN**

Open a new SPK file. (Use  
SPKOPA to append to existing file.)

**SPKWxx**

Write a segment of SPK type xx

.

.

**[SPKWxx]**

[ Write more segments ]

.

[ Repeat as needed ]

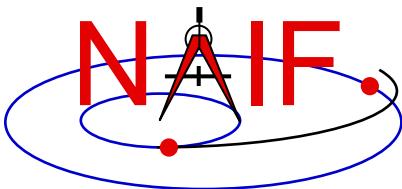
.

**SPKCLS**

Close the file

**[ ... ] indicates possible multiple occurrences**

These routine names are for the FORTRAN (SPICELIB) Toolkit. For CSPICE the names are the same but are in lower case and have an "\_c" appended. For Icy and Mice, module names are case-insensitive and have "cspice\_" prepended.



# What Routines To Use - 2

Navigation and Ancillary Information Facility

## For SPK type 14

**SPKOPN, SPKOPA**

Open file to add data

**SPK14B**

Begin a new segment

**SPK14A**

Add data to segment

**[SPK14A]**

Add more data

**SPK14E**

End the segment

**SPK14B**

Begin a new segment

**SPK14A**

Add data to segment

**[SPK14A]**

Add more data

**SPK14E**

End the segment

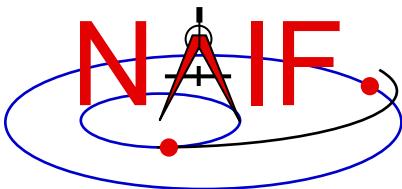
**etc.**

**SPKCLS**

Close the file

Repeat  
as needed

**[SPK14A]** indicates possible multiple occurrences

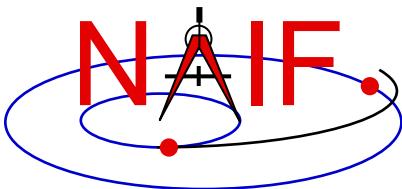


# Close the SPK File

---

Navigation and Ancillary Information Facility

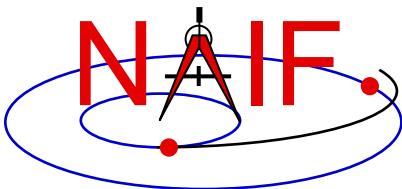
- Once you have completed the addition of all data to your SPK file, be sure to call the SPKCLS routine to close the file.
  - Failure to properly close an SPK file will result in a problem file having been produced.
- This point is emphasized here because it has been a frequent problem.



---

**Navigation and Ancillary Information Facility**

# Finishing Up

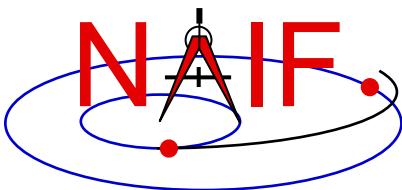


# Not Quite Done Yet

---

Navigation and Ancillary Information Facility

- You've now used either MKSPK or the appropriate SPK writer routines to produce an SPK file. To complete the job you should do the following.
  - Add comments (metadata) to the comment area of the SPK file.
    - » This could have been done during execution of MKSPK.
    - » It can be done after the SPK has been created by using the Toolkit's "commnt" utility program.
  - Validate the file before sending it off to your customer.
  - Consider if there is a need to merge this newly made SPK file with others.
- See the next several charts for more information on these subjects.

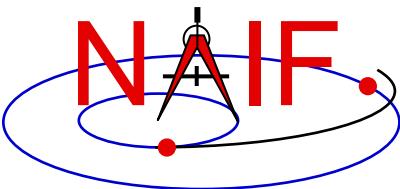


# Add Comments (metadata)

---

Navigation and Ancillary Information Facility

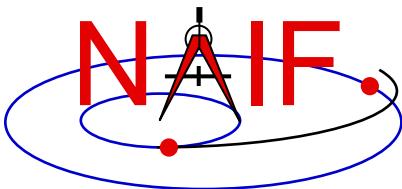
- It is strongly recommended that the producer of an SPK file add to the file, in the “comment area,” appropriate descriptive information. Topics should include at least:
  - when, how and by whom the file was created,
  - intended use for the file,
  - cautions or restrictions on how the file is to be used.
- The comments might also include some of these topics.
  - Time coverage
  - Ephemeris objects included
  - Type(s) of data used (in the sense of reconstructed versus predicted)
  - Any available estimates of accuracy
  - Sources of the data used to produce this SPK file
  - Name(s) of previously generated SPK file(s) being replaced by this file
  - Any knowledge of plans for future updates to (replacements for) this file
  - Name and version number of your SPK production program
  - Type of platform (hardware/OS/compiler) on which the SPK file was generated
- If you are modifying an existing SPK file, be sure to check the existing comments to see if updates are needed in addition to adding appropriate new ones.



# How to Add Comments to an SPK

Navigation and Ancillary Information Facility

- **Several means are available for adding comments (metadata) to an SPK file.**
  - An option in the MKSPK program allows comments supplied in a separate text file to be added to the comment area during MKSPK execution.
  - Use the “COMMNT” utility program from the SPICE Toolkit.
    - » This may be run as an interactive program or in command line mode within a script.
  - If using FORTRAN, C or IDL you can use APIs.
    - » Not currently supported in MATLAB.

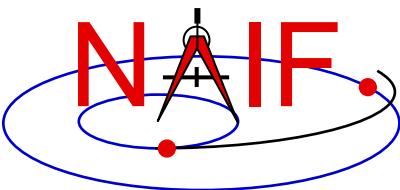


# Validate the SPK File

---

Navigation and Ancillary Information Facility

- **Validation of SPK files is critical**
  - Caution is needed more for one-of-a-kind files than for those generated in a previously tested, unchanging process.
  - Some SPICE utility programs might help with your validation.
    - » **SPY:** can do a variety of structure and semantic checks.
      - SPY is available from the Utilities link of the NAIF website.
    - » **SPKDIFF:** used to statistically compare two supposedly similar SPK files.
      - SPKDIFF is available in each Toolkit package and also from the Utilities link of the NAIF website.
  - Consider writing your own validation program.
  - Caution: successfully running an SPK summary program (e.g. BRIEF or SPACIT) or successfully running the format conversion program (TOXFR or SPACIT) is a positive sign, but is not a sufficient test.

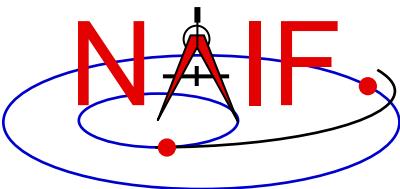


# Validate the Overall Process

---

Navigation and Ancillary Information Facility

- When you first start producing SPK files, or when changing the SPK “type” used or the kind of mission (trajectory) to be represented, validation (or revalidation) of the overall process is advised.
  - Validation of not only SPKs, but of end products derived from SPKs, is advised.
- Consider writing a program that compares states from your source data with states extracted from your new SPK file.
  - Do this using **interpolated states** from your source data—not just the states placed in the SPK file.
  - Verify a uniformly good fit over the whole time interval covered by the file.

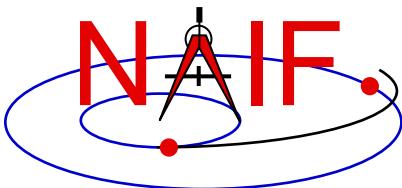


# Common SPK Production Errors

---

Navigation and Ancillary Information Facility

- **Some of the most common SPK production errors are these**
  - Picking an SPK type that can not well represent the ephemeris data you wish to represent
  - For Types 8, 9, 12 and 13, picking an invalid polynomial degree
  - For Types 8, 9, 12 and 13, picking a polynomial degree that does not allow high fidelity representation of your source ephemeris
  - For Types 8, 9, 12 and 13, having a HUGE change in time step size in two time-adjacent SPK segments
  - For Types 8, 9, 12 and 13, allowing position and/or velocity data to have unduly large discontinuities at segment boundaries
  - Leaving time gaps between segments (even very small, sub-second gaps can cause problems for users)
- **It takes careful design to avoid, and careful validation to detect some of these!**

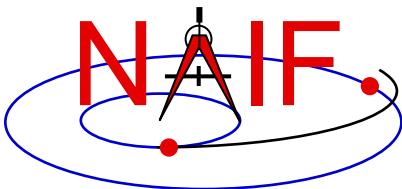


# Make a Merged SPK File ?

---

Navigation and Ancillary Information Facility

- **Sometimes it is helpful to customers if portions of two or more SPK files are merged into just one.**
  - (Sometimes the opposite is true, so be careful!)
- **If making a merged product is appropriate, use the SPICE utility SPKMERGE.**
  - Read the SPKMERGE User's Guide.
    - » Be especially aware of how SPKMERGE directives affect the **precedence order** of the data being merged. (This is different from the precedence order that applies when one reads an SPK file or files.)
  - Carefully examine your results (probably using either BRIEF or SPACIT) to help verify you got what you expected.
- **If you've made a merged SPK file, check to see that the included comments are appropriate.**

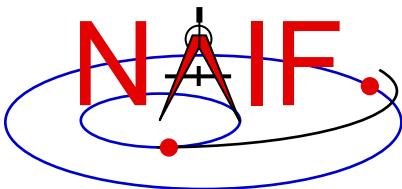


# Get Help

---

Navigation and Ancillary Information Facility

- **Considering consulting with JPL's NAIF team for assistance with:**
  - picking the SPK type to be used
  - picking the method for producing SPK files
  - designing tests to validate the process
- **Examine SPK files from other missions that could help you check your process.**
  - Such files can be found under the "Data" link on the NAIF website.

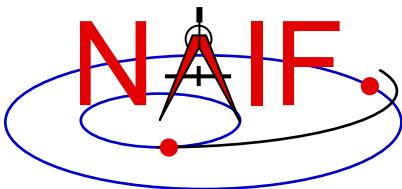


---

Navigation and Ancillary Information Facility

# Backup

- **Making SPKs for use by the DSN**
- **SPK reading efficiency**

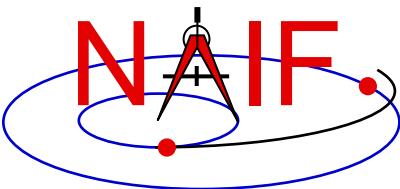


# DSN Interface Overview

---

Navigation and Ancillary Information Facility

- SPKs prepared for use in the DSN/SPS may be used in one or more of four DSN software sets:
  - Metric Predicts Generator (MPG)
    - » Used for view period generation, DSN scheduling and DSN metric predicts (antenna pointing and tuning of the transmitters and receivers)
  - Telecomm Predicts (UTP/TFP)
    - » Subsystem for prediction and analysis of telecommunications signal levels
  - Radiometric Modeling and Calibration Subsystem (RMC)
    - » Used to calibrate atmospheric effects on radio waves
  - Delta Differenced One-way Range (Delta-DOR) subsystem
    - » A special tracking data type providing additional precision to spacecraft navigation
- All SPKs delivered to the SPS must pass through a front-end gateway that has some restrictions that go beyond SPICE requirements; see the next page.

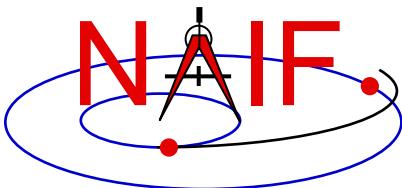


# SPS Validation Gateway

---

Navigation and Ancillary Information Facility

- **The SPS' front-end gateway requires:**
  - an SPK contain data for only one spacecraft
    - » The presence of non-spacecraft ephemeris data is ok
  - an SPK have no data gaps for the spacecraft
- **Additionally, for historical reasons, the spacecraft SPK must be of Type 1 or Type 13**
  - This restriction could be diminished or removed sometime in the future if DSN personnel find time to test the Metric Predicts Generator code using additional SPK data types.

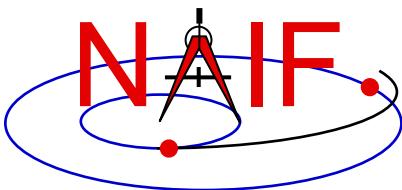


# SPK Reading: Efficiency Issues - 1

---

Navigation and Ancillary Information Facility

- **SPK file creators should design files to support efficient read access by those who will use the SPKs.**
  - This requires knowledge of how SPK file attributes impact efficiency.
- **When you store "large" amounts ( $>10^7$  states or data packets) of ephemeris data in one or more SPK files, SPK reading efficiency may be affected by:**
  - SPK segment size
  - the number of segments for a body in one SPK file
  - the number of segments for a body contributed by multiple SPK files
  - the number of loaded segments for all bodies
  - the number of loaded files



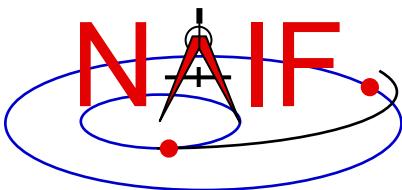
# SPK Reading: Efficiency Issues - 2

---

Navigation and Ancillary Information Facility

- **Segment size**

- When a segment contains more than 10,000 states or data packets, the SPK readers will generally take longer to search the segment for requested data.
  - » When the segment is larger than this size, more records are read to look up segment directory information. If these records are not buffered, more physical records are read from the SPK file.
- There is a trade-off between segment size and numbers of segments and files.
  - » It can be preferable to have large segments rather than have "too many" segments or files. (Read on)

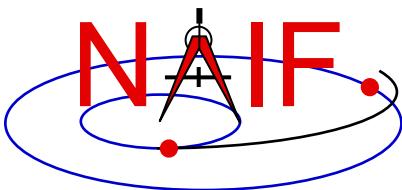


# SPK Reading: Efficiency Issues - 3

---

Navigation and Ancillary Information Facility

- **Number of segments for a body in one SPK file**
  - An SPK file **MUST** not contain more segments for one body than can be "buffered" at one time.
    - » The SPK reading system buffers coverage descriptions ("segment descriptors") for segments it has examined to satisfy previous requests for state data.
      - Don't confuse descriptor buffering with data buffering.
        - The SPK reading system also buffers segment DATA, as opposed to segment descriptors, but this is not relevant to this discussion.
      - » One fixed-size buffer is used for all SPK segment descriptors.
        - The size of this buffer is given by the parameter "STSIZE," declared in the SPKBSR suite of routines.
        - STSIZE is currently set by NAIF to 100,000.
          - NAIF recommends that users NOT change this parameter since maintenance problems may result.
      - » Unsurprisingly, the system works best when all needed segment descriptors are buffered simultaneously.

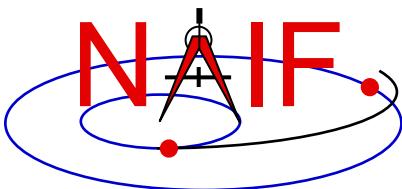


# SPK Reading: Efficiency Issues - 4

---

Navigation and Ancillary Information Facility

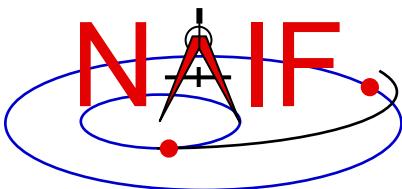
- **Number of segments for a body in one SPK file, continued.**
  - » The buffering scheme is "lazy": no descriptors are buffered for segments that haven't been examined.
    - But when an SPK file is searched for data for a specified body, descriptor data for ALL segments in the file for that body are buffered.
  - » The buffering algorithm can "make room" in the buffer by discarding unneeded, buffered descriptor data.
    - A "least cost" algorithm decides which buffered data to discard.
  - » When more buffer room is needed than can be found:
    - The SPK reading system reads data directly from SPK files without buffering descriptor information.
    - This is NOT an error case: the SPK system will continue to provide correct answers. **But the system will run VERY SLOWLY.**
      - This situation is analogous to "thrashing" in a virtual-memory operating system.
      - If buffer overflow occurs frequently, the SPK reading system may be too slow to be of practical use.



# SPK Reading: Efficiency Issues - 5

Navigation and Ancillary Information Facility

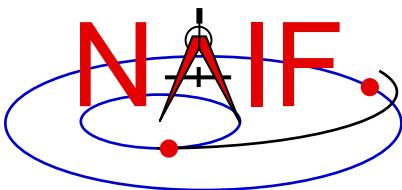
- **Number of segments for a body contributed by multiple SPK files:**
  - Buffer overflow can occur if too many segments for one body are contributed by multiple loaded SPK files.
    - » Overflow can take longer to occur than in the single-SPK case, due to lazy buffering: files that haven't been searched don't consume buffer space.
      - Thus an impending overflow problem may not be detected early in a program run.
  - User applications can avoid buffer overflow if data are appropriately spread across multiple SPK files.
    - » Applications can avoid buffer overflow by:
      - loading only those files of immediate interest
      - unloading files once they're no longer needed



# SPK Reading: Efficiency Issues - 6

Navigation and Ancillary Information Facility

- **Number of segments for all bodies, contributed by all loaded SPK files:**
  - Buffer overflow does not result from loading SPK files contributing more than STSIZE segments for different bodies.
  - However, if the total number of loaded segments for bodies of interest exceeds STSIZE, thrashing can occur as descriptor data are repeatedly discarded from the buffer and then re-read.
    - » Loaded segments for bodies for which data are not requested do not contribute to the problem.
  - For best efficiency, load only files contributing fewer than a total of STSIZE segments for all bodies of interest.
    - » When more than STSIZE segments are needed, applications should process data in batches: unload files containing unneeded data in order to make room for new files.

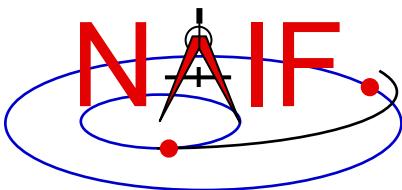


# SPK Reading: Efficiency Issues - 7

Navigation and Ancillary Information Facility

- **Number of loaded SPK files:**

- Up to 5000 SPK files may be loaded at one time by an application.
  - » The 5000 limit applies to DAF-based files, so loaded C-kernels and binary PCK kernels count against this limit.
- But loading large numbers of SPK files hurts efficiency:
  - » Since operating systems usually allow a process to open much fewer than 5000 files, the SPK system must open and close files via the host system's file I/O API in order to provide a "virtual" view of 5000 open files.
    - The more such file I/O, the slower an application runs.
  - » Loading a large number of SPK files could result in a buffering problem if too many segments are loaded for the bodies of interest.

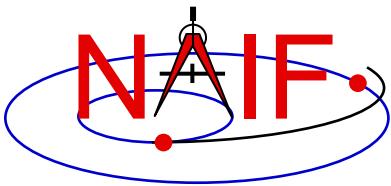


# SPK Reading: Efficiency Issues - 8

Navigation and Ancillary Information Facility

- **Recommendations**

- Limit segment counts to avoid buffer overflow and thrashing
  - » Never have more than STSIZE segments for one body in an SPK file and never have more than STSIZE segments for one body loaded simultaneously.
  - » Don't require users to have more than STSIZE segments loaded at one time.
  - » If necessary, use larger segments to enable smaller segment counts.
- Consider distributing SPK data across multiple files ...
  - » so as to make selective SPK loading convenient
    - facilitate processing data in batches
  - » so that loading very large numbers of SPK files at once is unnecessary

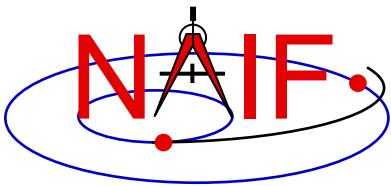


---

Navigation and Ancillary Information Facility

# Making a CK file

January 2020

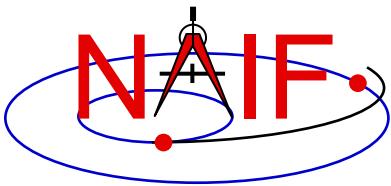


# Summary

---

Navigation and Ancillary Information Facility

- SPICE provides means to create CK files, either by packaging orientation computed by others, or by first computing orientation within SPICE and then packaging it in a CK file
  - Packaging of already existing orientation data can be done in two ways:
    - » Use SPICE CK writer routines by calling them from within your own SPICE-based application
    - » Convert a text file containing orientation data to a CK using the Toolkit's *msopck* program
  - Computing as well as packaging orientation can be done in two ways:
    - » Use SPICE geometry routines and CK writer routines by calling them from within your own SPICE-based application
      - Constructing orientation using SPICE routines is not discussed here
    - » Convert orientation rules and schedules to a CK using the *prediCkt* program available from the NAIF website

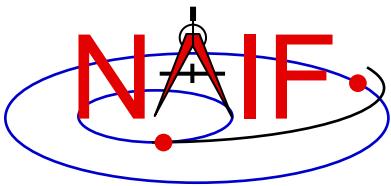


# CK Writer Routines

---

Navigation and Ancillary Information Facility

- The SPICE toolkit provides the following CK writer routines for the FORTRAN, C, IDL and MATLAB toolkits, respectively:
  - For Type 1 CK
    - » CKW01 / ckw01\_c / cspice\_ckw01
  - For Type 2 CK
    - » CKW02 / ckw02\_c / cspice\_ckw02
  - For Type 3 CK
    - » CKW03 / ckw03\_c / cspice\_ckw03
  - For Type 4 CK
    - » CKW04B, CKW04A, CKW04E (no CSPICE, Icy, or Mice wrappers)
  - For Type 5 CK
    - » CKW05 / ckw05\_c (no Icy or Mice wrapper)
  - For Type 6 CK
    - » CKW06 (no CSPICE, Icy or Mice wrappers)
- Only the Type 3 writer is discussed in this tutorial
  - Writers for Types 1 and 2 have very similar interfaces
  - Types 4, 5 and 6 are not commonly used



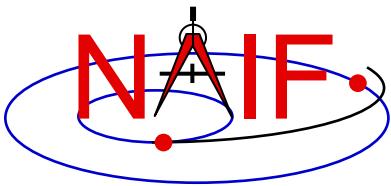
# Type 3 Writer Example - 1

Navigation and Ancillary Information Facility

- The following C-language code fragment illustrates the creation of a Type 3 C-kernel having a single segment.

```
ckopn_c ( filename, "my-ckernel", 0, &handle );
/*
   Insert code that properly constructs the
   sclkdp, quats, avvs, and starts arrays.
*/
ckw03_c ( handle, begtim, endtim, inst,
           "ref", avflag, "segid", nrec,
           sclkdp, quats, avvs, nints, starts );

ckcls_c ( handle );
```

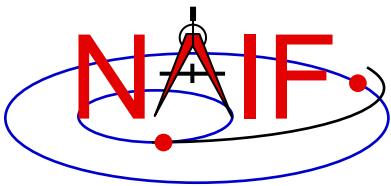


## Type 3 Writer Example - 2

---

Navigation and Ancillary Information Facility

- **handle** - file handle for the newly created C-kernel.
- **begtim**, **endtim** - start and stop times in SCLK ticks for the segment.
- **inst** - ID code for the instrument for which the C-kernel is being made.
- **ref** - name of the base reference frame. Must be one known to SPICE during your program execution.
- **avflag** - a SpiceBoolean indicating whether or not to include angular velocity in the segment.
- **segid** - a string identifying the segment. It must be no more than 40 characters in length.

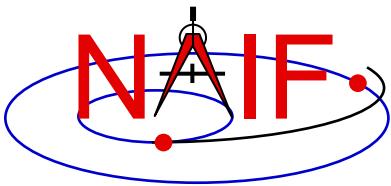


# Type 3 Writer Example - 3

---

Navigation and Ancillary Information Facility

- **nrec** - number of records in **sclkdp**, **quats**, and **avvs**.
- **sclkdp** - monotonically increasing list of times, given in SCLK ticks, that identify when **quats** and **avvs** were sampled or calculated.
- **quats** - a list of SPICE quaternions that rotate vectors from the base frame specified by the **ref** argument to the **inst** frame.
- **avvs** - angular rate vectors given in the base frame specified by the **ref** argument.
- **nints** - number of entries in **starts**.
- **starts** - a list of SCLK ticks indicating the start of interpolation intervals. They must correspond to entries in **sclkdp**.



# Type 3 writer - Making Up Rates

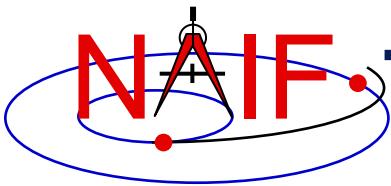
Navigation and Ancillary Information Facility

- One of the easiest ways to make up angular rates is to assume a constant rotation rate between subsequent quaternions:

```
for(k=0; k<(nrec-1); k++ ) {  
    q2m_c ( quats[k][0], init_rot );  
    q2m_c ( quats[k+1][0], final_rot );  
    mtxm_c ( final_rot, init_rot, rotmat );  
    raxisa_c ( rotmat, axis, &angle );  
    sct2e_c ( scid, sclkdp[k], &init_et );  
    sct2e_c ( scid, sclkdp[k+1], &final_et );  
    vscl_c ( angle/(final_et-init_et), axis,  
              &avvs[k][0] );  
}
```

- Then copy the (nrec-1) value of avvs into the last element of avvs.

continued on next page



# Type 3 Writer - Making Up Rates (2)

---

Navigation and Ancillary Information Facility

- **Constructing angular rates in this fashion assumes that no more than one 180-degree rotation has occurred between adjacent quaternions.**
  - `raxisa_c` chooses the smallest angle that performs the rotation encapsulated in the input matrix.
- Other techniques exist, including differentiating quaternions. Care must be exercised when taking that approach.



# MSOPCK

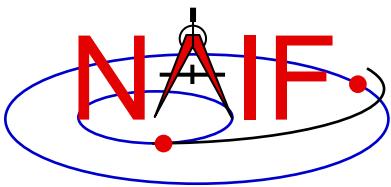
---

Navigation and Ancillary Information Facility

- ***msopck* makes CK files from orientation data provided in a time-tagged, space-delimited table in text format**
- ***msopck* can process quaternions (SPICE and non-SPICE styles), Euler angles, or rotation matrices, tagged with UTC, SCLK, or ET time tags**
- ***msopck* requires all program directives to be provided in a setup file that follows the SPICE text kernel syntax**
- ***msopck* has a simple command line interface with the following usage**

```
msopck setup_file input_data_file output_ck_file
```

- **If the specified output CK already exists, new segment(s) are appended to it**



Supporting  
Kernels/Files

Output CK  
Specifications

Input data  
Specifications

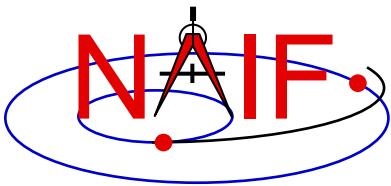
Optional and  
conditional  
keywords are  
shown in green

# MSOPCK

## List of Setup File Keywords

### Navigation and Ancillary Information Facility

|                        |                                                                          |
|------------------------|--------------------------------------------------------------------------|
| LSK_FILE_NAME          | = 'LSK file'                                                             |
| SCLK_FILE_NAME         | = 'SCLK file' (or MAKE_FAKE_SCLK='new SCLK file')                        |
| FRAMES_FILE_NAME       | = 'FRAMES file'                                                          |
| COMMENTS_FILE_NAME     | = 'file containing comments'                                             |
| PRODUCER_ID            | = 'producer group/person name'                                           |
| INTERNAL_FILE_NAME     | = 'internal file name string'                                            |
| CK_SEGMENT_ID          | = 'segment ID string'                                                    |
| CK_TYPE                | = 1, 2, or 3                                                             |
| INSTRUMENT_ID          | = CK ID                                                                  |
| REFERENCE_FRAME_NAME   | = 'reference frame name'                                                 |
| MAXIMUM_VALID_INTERVAL | = interval length, seconds                                               |
| INPUT_TIME_TYPE        | = 'SCLK', 'UTC', 'TICKS', 'DPSCLK', or 'ET'                              |
| TIME_CORRECTION        | = bias to be applied to input times, seconds                             |
| INPUT_DATA_TYPE        | = 'MSOP QUATERNIONS', 'SPICE QUATERNIONS', 'EULER ANGLES', or 'MATRICES' |
| QUATERNION_NORM_ERROR  | = maximum normalization error                                            |
| EULER_ANGLE_UNITS      | = 'DEGREES' or 'RADIAN'S'                                                |
| EULER_ROTATIONS_ORDER  | = ('axis3', 'axis2', 'axis1')                                            |
| EULER_ROTATIONS_TYPE   | = 'BODY' or 'SPACE'                                                      |
| ANGULAR_RATE_PRESENT   | = 'YES', 'NO', 'MAKE UP', 'MAKE UP/NO AVERAGING'                         |
| ANGULAR_RATE_FRAME     | = 'REFERENCE' or 'INSTRUMENT'                                            |
| ANGULAR_RATE_THRESHOLD | = ( max X rate, max Y rate, max Z rate )                                 |
| OFFSET_ROTATIONANGLES  | = ( angle3, angle2, angle1 )                                             |
| OFFSET_ROTATION_AXES   | = ('axis3', 'axis2', 'axis1')                                            |
| OFFSET_ROTATION_UNITS  | = 'DEGREES' or 'RADIAN'S'                                                |
| DOWN_SAMPLE_TOLERANCE  | = down sampling tolerance, radians                                       |
| INCLUDE_INTERVAL_TABLE | = 'YES' or 'NO' (default 'YES')                                          |
| CHECK_TIME_ORDER       | = 'YES' or 'NO' (default 'NO')                                           |



# MSOPCK - Input Details (1)

Navigation and Ancillary Information Facility

## Four Examples

```
INPUT_DATA_TYPE = 'SPICE QUATERNIONS'
```

Input file:

```
TIME1 [TIME2] QCOS QSIN1 QSIN2 QSIN3 [ARX ARY ARZ ]  
..... .... .... .... .... .... .... .... ....  
TIME1 [TIME2] QCOS QSIN1 QSIN2 QSIN3 [ARX ARY ARZ ]
```

```
INPUT_DATA_TYPE = 'MSOP QUATERNIONS'
```

Input file:

```
TIME1 [TIME2] -QSIN1 -QSIN2 -QSIN3 QCOS [ARX ARY ARZ ]  
..... .... .... .... .... .... .... .... ....  
TIME1 [TIME2] -QSIN1 -QSIN2 -QSIN3 QCOS [ARX ARY ARZ ]
```

```
INPUT_DATA_TYPE = 'EULER ANGLES'
```

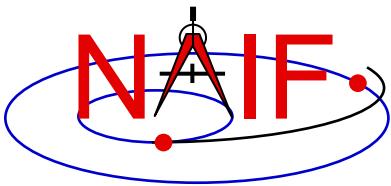
Input file:

```
TIME1 [TIME2] ANG3 ANG2 ANG1 [ARX ARY ARZ ]  
..... .... .... .... .... .... .... ....  
TIME1 [TIME2] ANG3 ANG2 ANG1 [ARX ARY ARZ ]
```

```
INPUT_DATA_TYPE = 'ROTATION MATRICES'
```

Input file:

```
TIME1 [TIME2] M11 M12 M13 M21 ... M33 [ARX ARY ARZ ]  
..... .... .... .... .... .... .... .... ....  
TIME1 [TIME2] M11 M12 M13 M21 ... M33 [ARX ARY ARZ ]
```



# MSOPCK - Input Details (2)

Navigation and Ancillary Information Facility

- **Quaternions**

- INPUT\_DATA\_TYPE = ‘SPICE QUATERNIONS’ indicates the quaternions being used follow the SPICE formation rules(\*)
- INPUT\_DATA\_TYPE = ‘MSOP QUATERNIONS’ indicates the quaternions being used follow the traditional AACS formation rules()
  - » Normally quaternions that come in telemetry are of this type
- QUATERNION\_NORM\_ERROR keyword may be used to identify and filter out input records with quaternions that are not unit vectors
  - » It is set to a tolerance for comparing the norm of the input quaternion with 1

- **Euler angles**

- All three angles must be provided
- For the angles provided on the input as

TIME1 [TIME2] ANG3 ANG2 ANG1 [ ARX ARY ARZ ]

and rotation axes specified in the setup as

EULER\_ROTATIONS\_ORDER = ( ‘axis3’, ‘axis2’, ‘axis1’ )

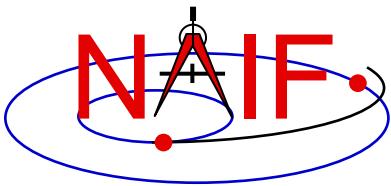
the matrix rotating vectors from base to the structure frame is computed as

Vinst = [ANG3]axis3 \* [ANG2]axis2 \* [ANG1]axis1 \* Vref

- Angles can be provided in degrees or radians

(\*) NAIF prepared a “white paper” explaining differences between various quaternion styles:

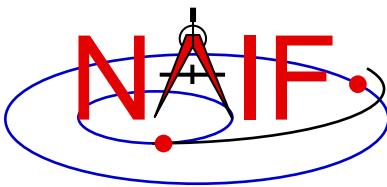
[https://naif.jpl.nasa.gov/pub/naif/misc/Quaternion\\_White\\_Paper/Quaternions\\_White\\_Paper.pdf](https://naif.jpl.nasa.gov/pub/naif/misc/Quaternion_White_Paper/Quaternions_White_Paper.pdf)



# MSOPCK - Input Details (3)

Navigation and Ancillary Information Facility

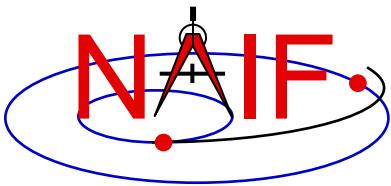
- Angular rates are an optional input. Their presence or absence must be indicated using the **ANGULAR\_RATE\_PRESENT** keyword
  - If angular rates are provided (**ANGULAR\_RATE\_PRESENT** = 'YES'), they must be in the form of a 3D vector expressed either in the base frame (less common) or instrument frame (more common)
    - » The **ANGULAR\_RATE\_FRAME** keyword must be set to indicate which of the two is used
  - If angular rates are not provided, the program can either make a CK without rates (**ANGULAR\_RATE\_PRESENT** = 'NO'), or try to compute rates from the orientation data by using a uniform rotation algorithm implemented in Type 3 CKs, either with averaging (**ANGULAR\_RATE\_PRESENT** = 'MAKE UP') or without averaging (**ANGULAR\_RATE\_PRESENT** = 'MAKE UP/NO AVERAGING') of the rates computed for adjacent orientation data points
  - **ANGULAR\_RATE\_THRESHOLD** may be used to identify and filter out input records with angular rate components that are too large to be real
- Input data can be tagged with UTC, SCLK string, SCLK ticks or ET, as specified using the **INPUT\_TIME\_TYPE** keyword
  - Time tags must not have embedded spaces



# MSOPCK - Output Details (1)

Navigation and Ancillary Information Facility

- ***msopck* can generate Type 1, 2, or 3 CKs**
  - Type 1 is rarely used - only in cases when the input contains very few data points that are far apart so that interpolation between them makes no sense
  - Type 2 is also rarely used, primarily to package orientation for spinning spacecraft
    - » Normally the input for making Type 2 CKs should contain two times and the angular rate in each record
  - Type 3 is the most commonly used type because it provides interpolation between the orientation data points stored in the CK
- **Interpolation intervals are determined based on the threshold value specified in the MAXIMUM\_VALID\_INTERVAL keyword**
  - The threshold interval is specified in units of seconds
  - A Type 3 CK will allow interpolation between all input points for which the duration between points is less than or equal to the threshold
- **An additional transformation to be combined with the input attitude may be specified using OFFSET\_ROTATION\_\* keywords**
  - The convention for specification of the offset rotation angles is the same as for the input Euler angles
  - A vector defined in the base frame is first multiplied by the offset rotation

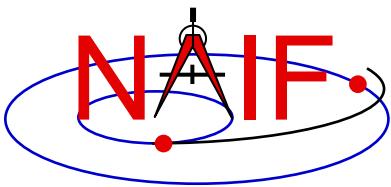


# MSOPCK - Output Details (2)

---

Navigation and Ancillary Information Facility

- The time tags may be adjusted by a constant value, specified in seconds, using the **TIME\_CORRECTION** keyword
- The order of input time tags can be checked using the **CHECK\_TIME\_ORDER** keyword.
- The output CK file contains one or more CK segments
  - Multiple segments are generated if the input data volume is large and does not fit into the program's internal buffer (100,000 pointing records)
  - When the output file has many segments, each segment's start time is equal to the stop time of the previous segment, i.e. there are no gaps at the segment boundaries
- The Comment area of the output CK contains the following information:
  - Contents of a comment file, if it was specified using the **COMMENT\_FILE\_NAME** keyword
  - Contents of the setup file
  - Summary of coverage for each segment written to the file, including a table listing interpolation intervals for segments of Type 2 or 3



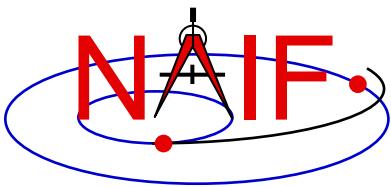
# MSOPCK - Example (1)

## Navigation and Ancillary Information Facility

### Terminal Window

```
$ more msopck_setup.example
MSOPCK setup for predict M'01 CK generation.
=====
\begindata
    PRODUCER_ID          = 'NAIF/JPL'
    LSK_FILE_NAME        = 'naif0007.tls'
    SCLK_FILE_NAME       = 'ORB1_SCLKSCET.00001.tsc'
    COMMENTS_FILE_NAME   = 'msopck_comments.example'
    INTERNAL_FILE_NAME   = 'sample M01 SC Orientation CK File'
    CK_SEGMENT_ID        = 'SAMPLE M01 SC BUS ATTITUDE'
    INSTRUMENT_ID         = -53000
    REFERENCE_FRAME_NAME = 'MARSIAU'
    CK_TYPE               = 3
    MAXIMUM_VALID_INTERVAL = 60
    INPUT_TIME_TYPE       = 'SCLK'
    INPUT_DATA_TYPE        = 'MSOP QUATERNIONS'
    QUATERNION_NORM_ERROR = 1.0E-3
    ANGULAR_RATE_PRESENT  = 'MAKE UP'

\begintext
$
```



# MSOPCK - Example (2)

## Navigation and Ancillary Information Facility

### Terminal Window

```
$ more msopck_comments.example
```

Sample Mars Surveyor '01 Orbiter Spacecraft Orientation CK File

Orientation Data in the File

This file contains sample orientation for the Mars Surveyor '01 Orbiter (M01) spacecraft frame, 'M01\_SPACECRAFT', relative to the Mars Mean Equator and IAU vector of J2000, 'MARSIAU', inertial frame. The NAIF ID code for the 'M01\_SPACECRAFT' frame is -53000.

Status

This file is a special sample C-Kernel file created by NAIF to illustrate MSOPCK program. This file should not be used for any other purposes.

...

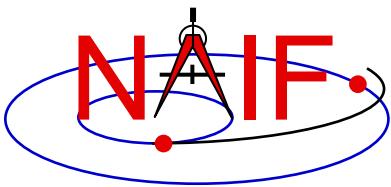


# MSOPCK - Example (3)

## Navigation and Ancillary Information Facility

Terminal Window

```
$ more msopck_input.example
0767491368.064  -0.24376335  0.68291384  0.28475901  0.62699316
0767491372.114  -0.24249471  0.68338563  0.28591829  0.62644323
0767491373.242  -0.24204185  0.68355329  0.28633291  0.62624605
0767491374.064  -0.24194814  0.68358228  0.28641744  0.62621196
0767491380.064  -0.24012676  0.68424169  0.28807922  0.62543010
0767491386.064  -0.23830473  0.68489895  0.28973563  0.62464193
0767491392.064  -0.23648008  0.68555126  0.29139303  0.62384833
0767491398.064  -0.23465389  0.68620253  0.29304524  0.62304745
0767491404.064  -0.23282999  0.68684150  0.29470173  0.62224580
0767491404.114  -0.23277293  0.68686688  0.29475362  0.62221455
0767491405.242  -0.23231585  0.68702790  0.29516507  0.62201253
0767491410.064  -0.23100059  0.68748174  0.29634561  0.62143935
0767491416.064  -0.22917353  0.68811325  0.29799308  0.62062853
0767491422.064  -0.22734161  0.68874177  0.29963482  0.61981412
0767491428.064  -0.22551078  0.68936246  0.30128030  0.61899473
0767491434.064  -0.22367453  0.68998299  0.30291779  0.61816987
0767491436.114  -0.22300583  0.69021050  0.30351804  0.61786298
0767491438.011  -0.22251770  0.69037871  0.30395477  0.61763631
...
```



# MSOPCK - Example (4)

## Navigation and Ancillary Information Facility

### Terminal Window

```
$ msopck msopck_setup.example msopck_input.example msopck_example_ck.bc
```

```
MSOPCK Utility Program, Version 3.0.0, 2003-05-05; SPICE Toolkit Ver. N0057
```

```
...
```

```
<comment file contents>
```

```
...
```

```
<setup file contents>
```

```
...
```

```
*****
```

```
RUN-TIME OBTAINED META INFORMATION:
```

```
*****
```

```
PRODUCT_CREATION_TIME = 2004-04-29T12:17:55
```

```
START_TIME = 2004-04-27T00:00:05.516
```

```
STOP_TIME = 2004-04-27T23:59:56.275
```

```
*****
```

```
INTERPOLATION INTERVALS IN THE FILE SEGMENTS:
```

```
*****
```

```
SEG.SUMMARY: ID -53000, COVERG: 2004-04-27T00:00:05.516 2004-04-27T23:59:56.275
```

```
-----  
2004-04-27T00:00:05.516 2004-04-27T20:05:26.282
```

```
2004-04-27T20:11:20.278 2004-04-27T23:59:56.273
```

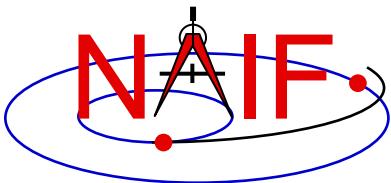


# PREDICKT

---

Navigation and Ancillary Information Facility

- *predCkt* makes CK files from a set of orientation specification rules, and schedules defining when these rules are to be followed
- *predCkt* has a simple command line interface
- *predCkt* requires orientation and schedule specifications to be provided in a setup file that follows the SPICE text kernel syntax
- *predCkt* requires the names of all supporting kernels -- SPK, PCK, etc -- be provided in a meta-kernel (a “furnsh kernel”)
- *predCkt* and its User Guide are available only from the Utilities link of the NAIF webpages

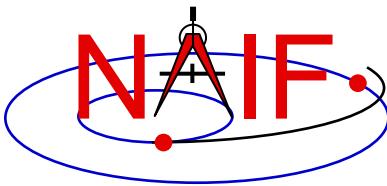


# PREDICKT - Usage

---

Navigation and Ancillary Information Facility

- *predCkt* has the following command line arguments
  - furnish support\_data
  - spec ck\_specs
  - ck outfile
  - tol fit\_tolerance [units]
  - <sclk|newsclk> sclk\_kernel
- ‘-furnish’, ‘-spec’ and ‘-ck’ are used to specify the input meta-kernel, input attitude specification file and output CK file
- ‘-tol’ is used to specify the tolerance to which the orientation stored in the CK should match the specified attitude profile
- ‘-sclk’ or ‘-newsclk’ specify the name of an existing SCLK or the new “fake” SCLK to be created for use with the output CK

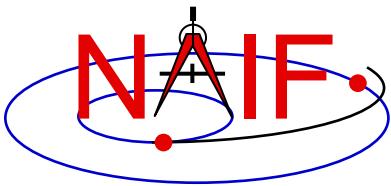


# PREDICKT - Furnsh and Spec Files

---

Navigation and Ancillary Information Facility

- A “FURNSH” kernel lists SPICE kernels that are to be used by prediCkt to determine geometry needed to compute orientations
- A prediCkt attitude specification (spec) file, using the text kernel syntax, is used to provide three types of information:
  - specification of dynamic directions
  - specification of orientations based on these directions
  - specification of the schedules defining when those orientations should be followed
- The contents of the FURNSH kernel and the spec file are included in the comment area of the output CK file



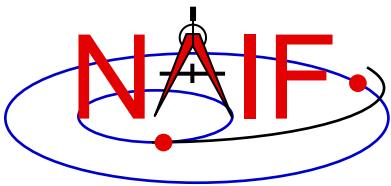
# PREDICKT - Directions

---

Navigation and Ancillary Information Facility

- **Dynamic directions can be of the following types**
  - Based on ephemeris (position vectors, velocity vectors)
  - Fixed with respect to a reference frame (expressed as a Cartesian vector or specified by RA and DEC)
  - Towards sub-observer point
  - Based on the surface normal and lines of constant latitude or longitude
  - Based on other, already defined directions (rotated from them, computed as cross products using them, etc)
- **Example: these two sets of keyword assignments specify nadir and spacecraft velocity directions for the M01 spacecraft**

```
DIRECTION_SPECS      += ( 'ToMars        = POSITION OF MARS -' )
DIRECTION_SPECS      += (                   'FROM M01          -' )
DIRECTION_SPECS      += (                   'CORRECTION NONE'   )
DIRECTION_SPECS      += ( 'scVelocity = VELOCITY OF M01 -' )
DIRECTION_SPECS      += (                   'FROM MARS          -' )
DIRECTION_SPECS      += (                   'CORRECTION NONE'   )
```



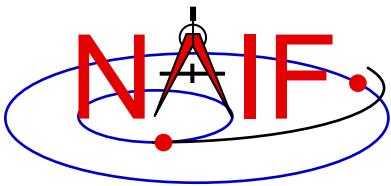
# PREDICKT - Orientations

---

Navigation and Ancillary Information Facility

- An orientation is specified by:
  - defining that one of the frame's axes (+X,+Y,+Z,-X,-Y,-Z) points exactly along one of the defined directions
  - defining that another of the frame's axes points as closely as possible to another defined direction
    - » The third axis is the cross product of the first two
  - specifying the base frame with respect to which the orientation of this “constructed” frame is to be computed
- Example: these keyword assignments specify the nominal nadir orientation for the THEMIS instrument, flown on the M01 spacecraft

```
ORIENTATION_NAME      += 'CameratoMars'  
PRIMARY                += '+Z = ToMars'  
SECONDARY              += '+Y = scVelocity'  
BASE_FRAME              += 'J2000'
```

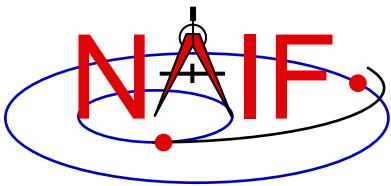


# PREDICKT - Schedules (1)

Navigation and Ancillary Information Facility

- A schedule is defined by specifying a series of time intervals during which a given orientation is to be followed
  - For each interval for a given CK ID, the spec file defines the orientation name, start time, and stop time (as Ephemeris Times)
- Example: these spec file keyword assignments specify a schedule with a single window during which M01 (Mars Odyssey) will yield nadir-pointed orientation for the THEMIS instrument

CK-SCLK	= 53
CK-SPK	= -53
CK-FRAMES	+= -53000
CK-53000ORIENTATION	+= 'SOLUTION TO M01_THEMIS_IR = CameratoMars'
CK-53000START	+= @2004-FEB-10-00:00
CK-53000STOP	+= @2004-FEB-15-00:00

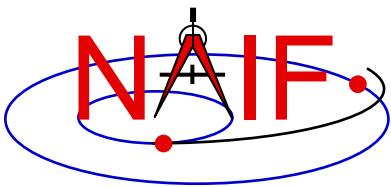


# PREDICKT - Schedules (2)

---

Navigation and Ancillary Information Facility

- In the example on the previous slide:
  - the CK-FRAMES keyword specifies the CK ID to be used in the output CK
    - » This ID is incorporated into the keywords defining the schedule intervals
  - the CK-SCLK keyword specifies the ID of the SCLK kernel to be used in creating the CK
  - the CK-SPK keyword specifies the ID of the object, the position of which is used in applying light time correction when orientation is computed
  - the “SOLUTION TO” construct specifies that although the orientation is sought for the M01 spacecraft frame (ID -53000), it is computed for the camera frame (M01\_THEMIS\_IR) and then transformed to the spacecraft frame



# PREDICKT - Example (1)

## Navigation and Ancillary Information Facility

Terminal Window

```
$ cat m01_map_nadir.predickt
\begin{data

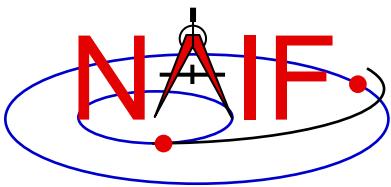
  DIRECTION_SPECS      += ( 'ToMars      = POSITION OF MARS -' )
  DIRECTION_SPECS      += (                   'FROM M01           -' )
  DIRECTION_SPECS      += (                   'CORRECTION NONE'   )

  DIRECTION_SPECS      += ( 'scVelocity = VELOCITY OF M01    -' )
  DIRECTION_SPECS      += (                   'FROM MARS          -' )
  DIRECTION_SPECS      += (                   'CORRECTION NONE'   )

  ORIENTATION_NAME     += 'CameratoMars'
  PRIMARY              += '+Z = ToMars'
  SECONDARY             += '+Y = scVelocity'
  BASE_FRAME            += 'J2000'

  CK-SCLK                = 53
  CK-SPK                 = -53
  CK-FRAMES               += -53000
  CK-53000ORIENTATION   += 'SOLUTION TO M01_THEMEIS_IR = CameratoMars'
  CK-53000START           += @2004-FEB-10-00:00
  CK-53000STOP            += @2004-FEB-15-00:00

\begin{text}
```



# PREDICKT - Example (2)

## Navigation and Ancillary Information Facility

### Terminal Window

```
$ cat m01_map_nadir.furnsh
\begin{data
    KERNELS_TO_LOAD = ( 'naif0007.tls'
                        'm01_v26.tf'
                        'mar033-5.bsp'
                        'm01_map_rec.bsp'
                        'm01.tsc' )

\begin{text
$ predickt -furnish m01_map_nadir.furnsh -spec m01_map_nadir.predickt -ck m01_map_nadir.bc -tol
0.01 degrees -sclk m01.tsc
```

Begin Segment: 1 --- SOLUTION TO M01\_THEMIS\_IR = CameratoMars

Constructing Segment

From: 2004 FEB 10 00:00:00.000

To : 2004 FEB 15 00:00:00.000

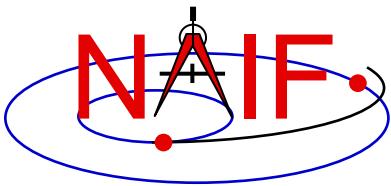
Percentage finished: 0.0%

Percentage finished: 5.0 % (50 quaternions)

...

Percentage finished: 95.0 % (925 quaternions)

\$



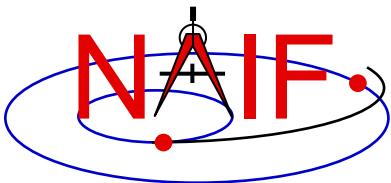
---

Navigation and Ancillary Information Facility

# Overview of the Events Kernel EK

January 2020

Note: the EK is infrequently used by NASA flight projects.  
Only a brief overview of the EK subsystem is provided.

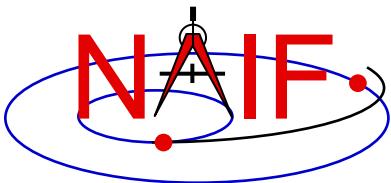


# Scope

---

Navigation and Ancillary Information Facility

- **This tutorial provides an overview of the entire Events Kernel subsystem, comprised of three logical components:**
  - Science Plan            ESP
  - Sequence              ESQ
  - Notebook              ENB
- **Depending on specific circumstances:**
  - the three logical components might exist as three distinct and different mechanisms
  - two or all three logical components might be implemented with a single mechanism
  - one or more logical components may not be used

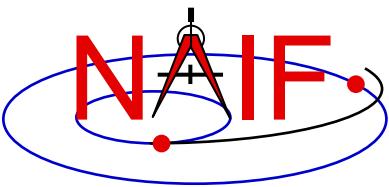


# E-Kernel Subsystem Objectives

---

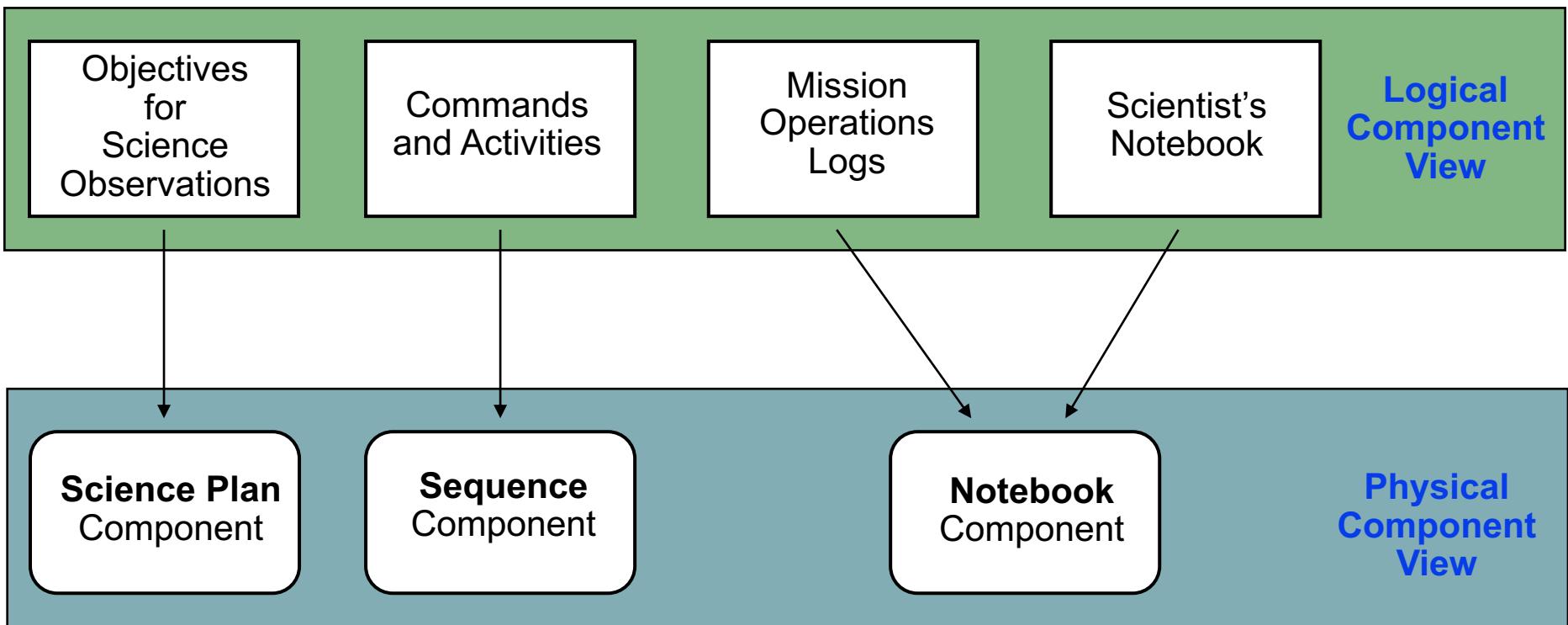
Navigation and Ancillary Information Facility

- **Assemble, archive and provide convenient and useful access to plans, commands and notes about the acquisition of space science observations**
  - For use by on-going project science and engineering team members
  - For use by post-mission researchers
- **Accomplish the above with minimal impact on science and mission operations team members**

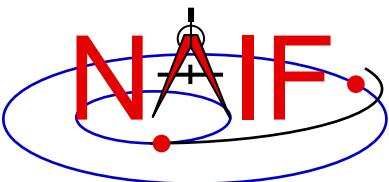


# Nominal E-kernel Composition\*

Navigation and Ancillary Information Facility

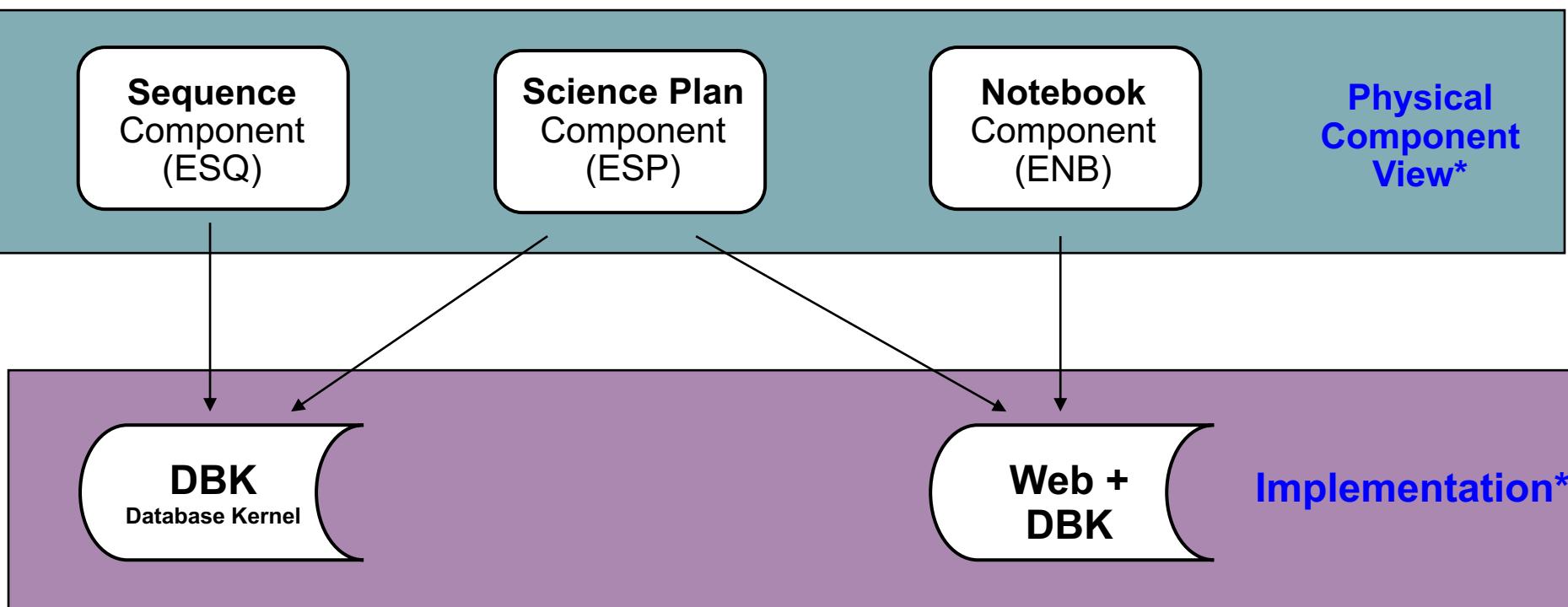


\* As originally envisioned by NAIF



# Nominal E-kernel Implementation\*

Navigation and Ancillary Information Facility



\* As originally implemented by NAIF

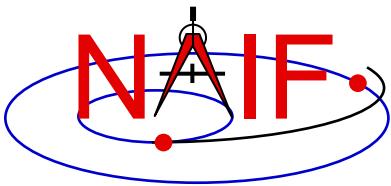


# Science Plan - ESP

---

Navigation and Ancillary Information Facility

- **Each entry is a statement of science objectives for a series of coordinated observations to be made over a stated period of time**
  - Might include some information about the planned mechanics (observation design) for obtaining the data
- **The Science Plan (ESP) could be implemented as a part of the SEQUENCE component (ESQ), or as a part of the NOTEBOOK component (ENB), or as a separate product using some other mechanism**

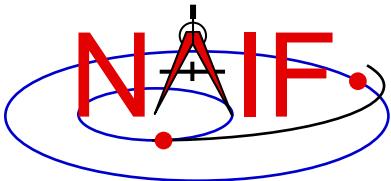


# Sequence - ESQ

---

Navigation and Ancillary Information Facility

- Principal entries are instrument and spacecraft “commands” or “macro calls” that carry out the objectives of the Science Plan. These contain the lowest level of detail that could be helpful while also being practical for inclusion in the E-kernel product
  - Could include ground system events, such as tracking station status
  - Could include “announcements” of the occurrence of geometric conditions of wide interest, such as equator crossing, occultation entry, etc.
  - Could include “state records” that summarize the status of an instrument or subsystem or spacecraft at a given epoch. (If to be included, state records might be derived rather than actually stored as physical objects.)
- CAUTION: within NASA this kind of information might be restricted under ITAR

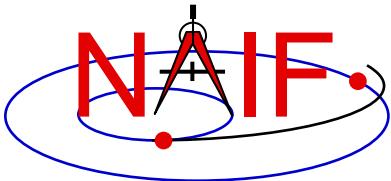


# Notebook - ENB

---

Navigation and Ancillary Information Facility

- **Entries are notes provided by scientists and flight team engineers about what happened as mission operations are conducted, including unplanned, unanticipated or unexplained occurrences**
- **Entries could also be general notes thought to be of interest to scientists**
- **Two methods for providing entries are available**
  - Entries submitted using e-mail can include MIME attachments, such as GIF, JPEG, EXCEL, WORD, etc., in addition to plain ASCII text
  - Entries submitted using WWW are limited to plain ASCII text

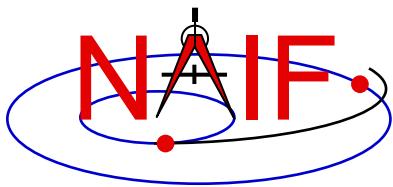


# E-Kernel Status

---

Navigation and Ancillary Information Facility

- **The E-kernel is the least well developed and least used component of the SPICE system**
  - It's of less interest to flight project instrument and engineering teams as compared to the other SPICE components
    - » Their perception is that EK information could be useful to future users of a mission's data, but not so much to an active flight team, and since they are already very busy they have not enough time to contribute inputs to an EK
- **Unfortunately NAIF and other kernel producers seem unlikely to produce EK components in the future**

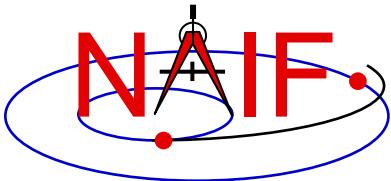


---

Navigation and Ancillary Information Facility

# SPICE Development Plans and Possibilities

January 2020

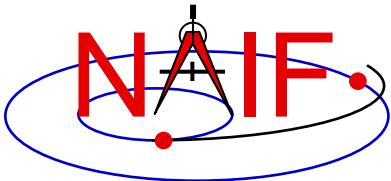


# DSK Shape Models

---

Navigation and Ancillary Information Facility

- **Extension of the DSK shape model subsystem**
  - Complete the Type 4 DSK code for working with digital elevation models
  - Add more functionality to the tessellated plate model (Type 2 DSK)
    - » The first official release of the Type 2 subsystem, for small, irregularly shaped bodies, was released in the N66 Toolkits
  - Unfortunately NAIF has no real target date in mind for this work

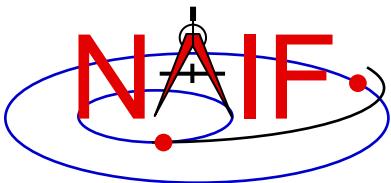


# SPICE 2.0

---

Navigation and Ancillary Information Facility

- **Develop SPICE 2.0: a re-implementation of the SPICE Toolkit from the ground, up, providing thread-safe and object oriented features**
  - This is the major NAIF undertaking, started in May 2017
  - It is being implemented in C++11
  - It is expected to take several years
- **No worries: none of the current Toolkits will be dropped.**



# Program Development

---

Navigation and Ancillary Information Facility

- **Continue adding capabilities to the WebGeocalc tool**
  - More kinds of calculations
  - More ease-of-use features
  - This work is on-going
  
- **Continue adding capabilities to the Cosmographia 3D mission visualization program**
  - This work is on-going



# Model Development

---

Navigation and Ancillary Information Facility

- **Complete and release a large set of dynamic frames in a generic frames kernel (or kernel set)**
  - Much work was done on this, but in the end it appears impossible to achieve community consensus on key aspects
- **Add some aspects of ring models**
  - At least ring reference frames
  - Maybe also shapes?
  - No active work on this

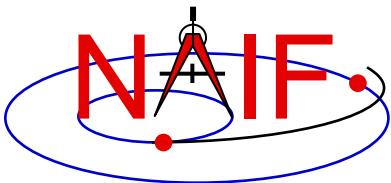


# More API Interfaces - 1

---

Navigation and Ancillary Information Facility

- **Complete the Java Native Interface (JNISpice) Toolkit family**
  - Reliability is felt to be very good
    - » (NAIF used JNISpice to implement the WebGeocalc tool)
  - Additional documentation needs be written
- **Python interface**
  - Several SPICE users have implemented and are offering their own, partial Python interfaces to SPICE
    - » Check here for links to two of them
      - <http://naif.jpl.nasa.gov/naif/links.html>
  - NAIF's use has been limited to preparing a few SpiceyPy lessons
  - Others report these offerings appear to be good quality products
  - Thus NAIF seems unlikely to do any of its own Python work

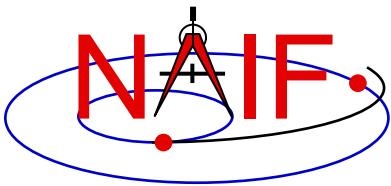


# More API Interfaces - 2

---

Navigation and Ancillary Information Facility

- **3<sup>rd</sup> parties have also implemented Ruby, Swift and Julia interfaces to CSPICE.**
  - NAIF hasn't tried testing any of these
  - NAIF does not know how complete these are
  - Give them a try, but use due caution as you do so
    - » You might be able to do some one-off tests using the WebGeocalc tool as a “gold bar”
    - » You could try using the “spice\_discussion” bulletin board to see what other people have to say about these interfaces

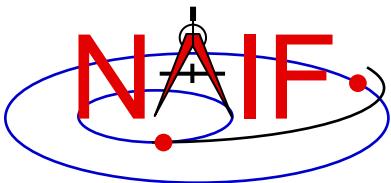


# Some Other Possibilities?

---

Navigation and Ancillary Information Facility

- More high-level SPICE 1.0 (current SPICE) computations, such as instrument footprint coverage
- More “geometry finder” computations
- Develop a more flexible and extensible instrument modeling mechanism

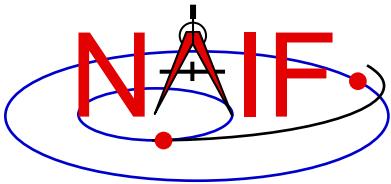


# Programmatic Expansion

---

Navigation and Ancillary Information Facility

- NAIF is helping the Republic of South Korea learn to use SPICE in support of their upcoming Korean Pathfinder Lunar Orbiter (KPLO) mission
- Colleagues at LASP are helping the United Arab Emirates deploy SPICE in support of their upcoming Hope mission to Mars
- We hope to find the means to support upcoming science-focused SmallSat/CubeSat missions
  - Example: Lunar IceCube

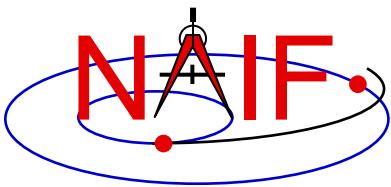


# What do You Suggest?

---

Navigation and Ancillary Information Facility

- **NAIF solicits suggestions from you!**
  - How might we improve SPICE?
  - How might we improve SPICE training?
  - How might we improve NAIF's operations?
  - How might we improve SPICE operability across the large and still growing international community?
- **We're interested in programmatic ideas as well as technical ones.**

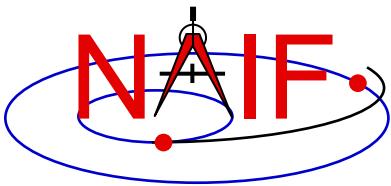


---

Navigation and Ancillary Information Facility

# Porting Kernels

January 2020



# Porting Issues - 1

---

Navigation and Ancillary Information Facility

- Data formats vary across platforms, so data files created on platform “X” may not be usable on platform “Y.”
  - **Binary formats:** different platforms use different bit patterns to represent numbers (and possibly characters).
  - **Text formats:** different platforms use different mechanisms to represent “lines” in text files.
    - Usually a “line terminator character sequence” indicates end-of-line.
- We say two platforms have “compatible” binary or text formats if they use the same binary or text data representations.
- We say that a file is “native” if its format is the same as that of the computer you are using.

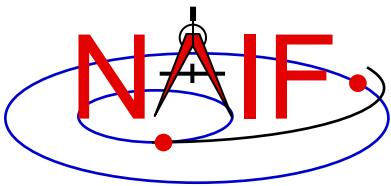


# Porting Issues - 2

---

Navigation and Ancillary Information Facility

- Toolkit software can **usually** read kernels obtained from an incompatible platform
  - Binary SPK, CK, PCK and DSK kernels from one system can always be read on an incompatible system
  - Text kernels from one system can be read on an incompatible system only when using a C, IDL, MATLAB or JNI: not when using Fortran
- See later charts for compatibility matrix

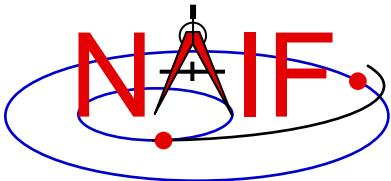


# Porting Issues - 3

---

Navigation and Ancillary Information Facility

- When conversion to native format is required to make the kernel usable (see a later chart), several options are available.
  - Use *bingo* for both binary and text kernels
    - Available only from the NAIF website; not provided in Toolkit packages
    - For text kernels, doing your file download using ftp in ASCII mode will perform the required format conversion on the fly
    - Web browsers often do text format conversion
      - However ASCII mode may not be available – sftp clients usually don't provide it. In such cases other tools such as freeware dos2unix and unix2dos, or bingo from the SPICE utilities page, must be used.
    - For binary kernels, the SPICE *toxfr* and *tobin* tools may be used to convert files to and from SPICE transfer format
      - This is an ASCII-based format that may be transferred in the same way as other ASCII files.



# Compatible Environments for Text Kernels

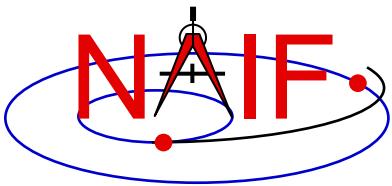
Navigation and Ancillary Information Facility

Since text kernels are only text files...

	<u>Groupings of Text Compatible Environments</u>	<u>End of line indicator</u>
1	PC using Windows or N T	<CR><LF>
2	Unix  PC with LINUX  Macintosh OSX (Motorola or Intel chip)	<LF>

On a Unix/Linux/OSX box you can easily see what kind of line terminator is being used in a text file using the Unix “cat -et” command on your text file.

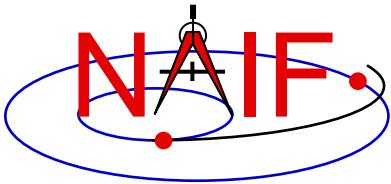
<CR> tokens will appear as ”^M”  
<LF> tokens will appear as “\$”



# Compatible Environments for Binary Kernels

Navigation and Ancillary Information Facility

	<u>Groupings of Binary Compatible Environments</u>	<u>Binary Representation</u>
1	<b>PC/ Windows</b> <b>PC/Linux</b> <b>Mac Pro (Intel chip)</b>	<b>IEEE - Little endian</b>
2	<b>Sun</b> <b>Mac Power PC (Motorola chip, discontinued after 2005)</b>	<b>IEEE - Big endian</b>

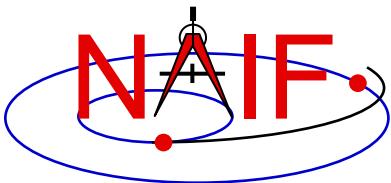


# Caution Using Email

---

Navigation and Ancillary Information Facility

- **NAIF recommends against the use of email to transfer kernels unless previous tests have already proven successful using the same conditions/computers intended for current use. Possible causes of problems are:**
  - incompatible binary or text representations (as already discussed).
  - an attachment size limit somewhere in the e-mail chain.
  - the sender's or recipient's mail client modifies the kernel based on file name or presumed content.
- **When you must email kernels, compress them either with zip, or gzip (or stuffit), then send the compressed file as an email attachment.**



# Binary Kernels - Caveats

---

Navigation and Ancillary Information Facility

- If the kernel you are using is a non-native binary kernel you can read this file but you may not write data to this file.
  - You **can read** most non-native binary kernels using the automatic run-time conversion capability found in the APIs of modern Toolkits.
    - » Exception: non-native DAS-based files (ESQ) created before 2001 cannot be read. They must first be converted to native format.
  - You **cannot write** information into the comment area, or delete information from the comment area.
  - You **cannot append** additional data to the kernel.

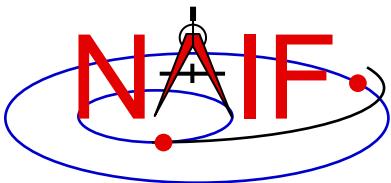


# Binary Kernels Allowed Operations

---

Navigation and Ancillary Information Facility

- You may “load” and read both non-native and native binary kernels in the same runtime instance
- You may merge similar native and non-native files—the resultant, merged file will be in native format.
  - SPKs: using SPKMERGE or DAFCAT
  - CKs: using DAFCAT
  - DSKs: using DLACAT



# Text Kernels - Caveats

---

Navigation and Ancillary Information Facility

- **Cutting/pasting complete, or pieces of, data assignments or \begindata or \begin{text} markers into a text kernel can cause a problem**
  - It may result in insertion of non-printing characters or incorrect end-of-line terminations
  - This is not a problem for comments, but it is probably best to treat all portions of a text kernel the same
- **If creating a text kernel by editing an existing one:**
  - first save a backup copy
  - be sure you are starting with a file in native format for the computer you are using: either Unix/Linux/Mac or Windows
  - be sure to insert a final end-of-line marker at the end of your last line of data or text
    - » Press “return”