Navigation and Ancillary Information Facility

Welcome
to the
SPICE Tutorials

January 2012

Objectives

Navigation and Ancillary Information Facility

- **For you**
  - Provide an overview of the entire SPICE system
  - Provide a sense of the purpose and uses of SPICE
  - Provide an introduction to the use of primary SPICE components
  - Provide examples of how to use SPICE software and data files
  - Provide some insight into conventions and common problems
  - Provide a variety of "hands-on" programming exercises
  - Provide a peek at new capabilities being worked on or considered
  - Familiarize you with available SPICE resources

- **For NAIF (if these tutorials are being seen in a class setting)**
  - Get student's feedback on today's SPICE system
    - » Especially what's hard to understand, hard to use, missing
  - Get student's suggestions for further development of SPICE
  - Get student's suggestions for improvements to NAIF support of the space science community

Scope

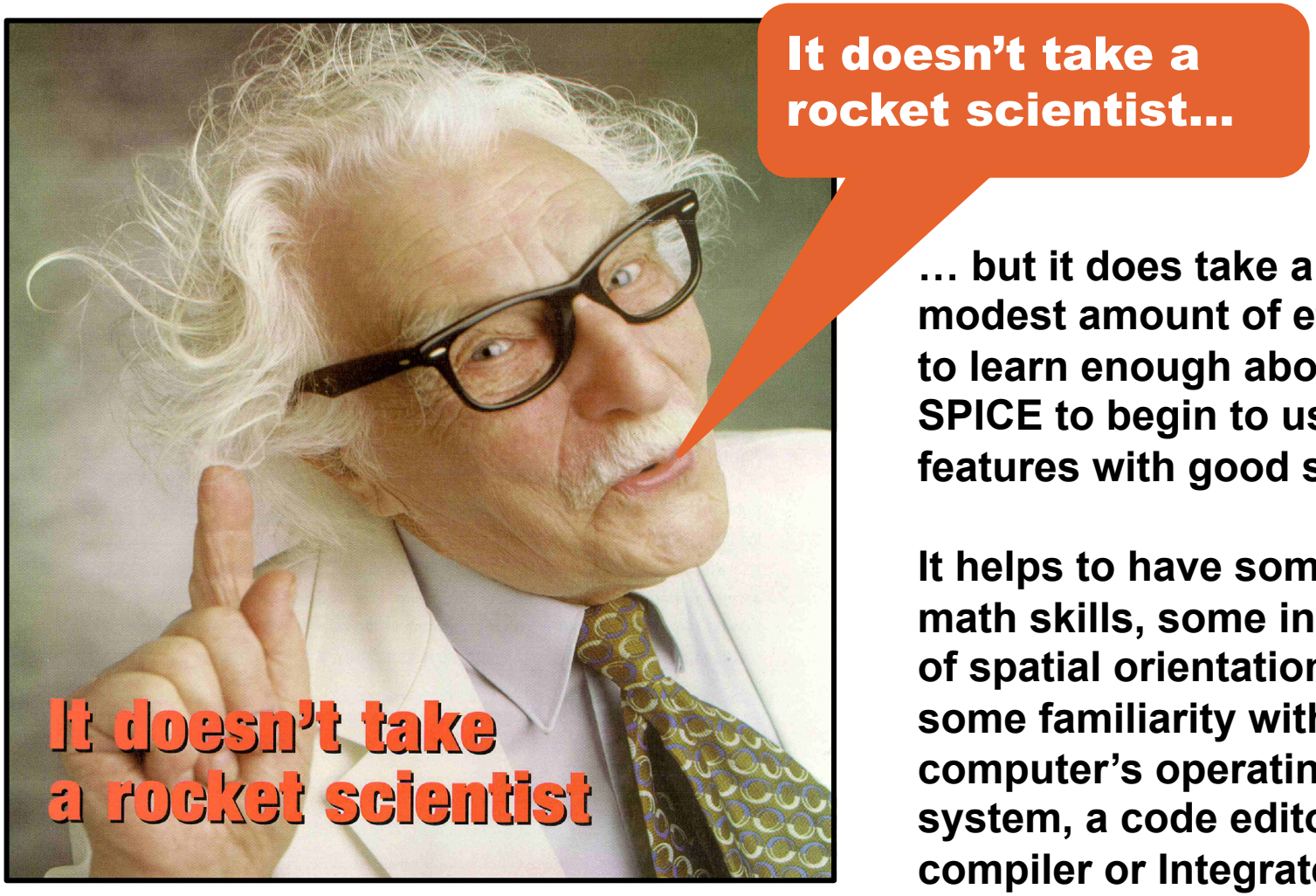
- **Broad coverage**
  - Begins at a high level, but quickly drills down to details
  - Touches on many SPICE-related topics that could be of interest to science and engineering teams
    - » Depth of discussion varies somewhat amongst topics
- **Provide information for FORTRAN, C, IDL and MATLAB programmers**
- **Some tutorials have important material provided in a "Backup" section: you should read these**
- **Some topics are addressed rather little or not at all**
  - Kernel production
  - Archiving SPICE data

Repeated Material

- 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 is a Large Product

- **The generic SPICE Toolkit contains:**
  - Around 900 individual modules (subroutines)
    - » most customers use only a handful of these
  - about 13 utility and application executables (with User Guides)
  - about 23 subsystem reference documents
  - 4 “cookbook” tutorial programs (with User Guides)

  and assorted other documents, scripts and libraries.

- **Don’t let this size bother you…**

  … just work your way into it bit by bit.

The NAIF Team at JPL

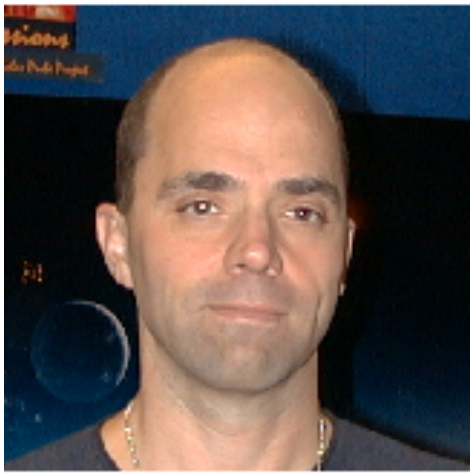
Navigation and Ancillary Information Facility

Chuck Acton

Nat Bachman

Sam Krening

Boris Semenov



Ed Wright

Navigation and Ancillary Information Facility

SPICE Introduction

January 2012

History

Navigation and Ancillary Information Facility

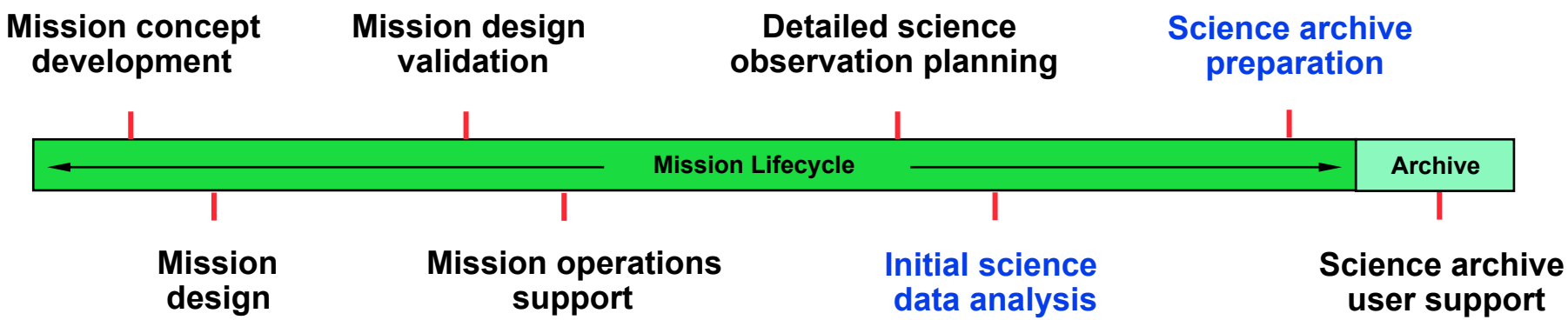
- Implementation of a precursor to SPICE was initiated in 1984 as part of a major initiative to improve archiving and distribution of space science data in all NASA disciplines
- Responsibility for leading SPICE development was assigned to the newly-created Navigation and Ancillary Information Facility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

Breadth of Use

Navigation and Ancillary Information Facility

- The original focus of SPICE was on ancillary data and associated software needed by scientists for:
  - initial science data analysis
  - science archive preparation

- The scope of SPICE usage has grown to cover the full mission lifecycle, as well as post-mission archive uses.



Major SPICE Users*

Navigation and Ancillary Information Facility

- SPICE is used on all NASA planetary exploration projects
  - Examples: All Mars missions, Cassini, Deep Impact, Messenger, Juno
- Limited SPICE data have been (or are being) created for some past missions
  - Examples: Voyager, Viking Orbiter
- SPICE is used to some degree in support of some space physics and astrophysics missions
  - Examples: Hubble Telescope, Spitzer Telescope, IBEX, Wise, Kepler
- SPICE was or is used on many non-NASA missions
  - Russia's Mars 96 and Phobos-Grunt; ESA's Huygens Probe, Smart-1, Mars Express, Rosetta and Venus Express; Japan's Hayabusa and SELENE; India's Chandrayaan-1
- SPICE will be used on at least one NASA earth science mission (SMAP)
- SPICE ephemerides are used at some terrestrial observatories
- SPICE is used by NASA's Deep Space Network for both scheduling and operating the DSN antennas.

* Not all are supported by NAIF; some are using SPICE on their own.

Ancillary Data Archives

- SPICE is the U.S. Planetary Data System's de facto standard for archiving ancillary data
  - (But it's not a formal requirement)
- SPICE data for European planetary missions are archived in ESA's Planetary Science Archive
  - Some of these data will be mirrored on the NAIF server
- SPICE data for some Japanese, Indian and Russian missions may be available in the future from their local archives
  - Already the case for Hayabusa
- SPICE, or some SPICE principles, might play a role in the future International Planetary Data Alliance (IPDA)

Distribution and Use of SPICE

- SPICE system components are freely distributed
  - Projects pay for local deployment and operation, done either by their own personnel, or by NAIF, or a combination
  - There are no U.S. ITAR restrictions on distribution
  - There are no licensing requirements
- Users get complete source code and much documentation
- SPICE user's are encouraged to reference SPICE when it has provided a significant contribution to one's work
- See the "Rules" link on the NAIF Home Page for a complete list of rules and recommendations

Training Materials

- This set of tutorials has been presented and revised numerous times
  - No matter how hard we try, it seems impossible to:
    - » get all the facts absolutely right and up-to-date
    - » get the level of detail “right” for every student
    - » get all of the language clear, complete and concise
    - » present everything in the “correct” order
- These training materials are meant to supplement–not replace–the subroutine headers and the “required reading” reference documents that are the primary sources for user information about SPICE

Navigation and Ancillary Information Facility

Motivation for Developing SPICE

January 2012

Why Did NAIF Build SPICE?

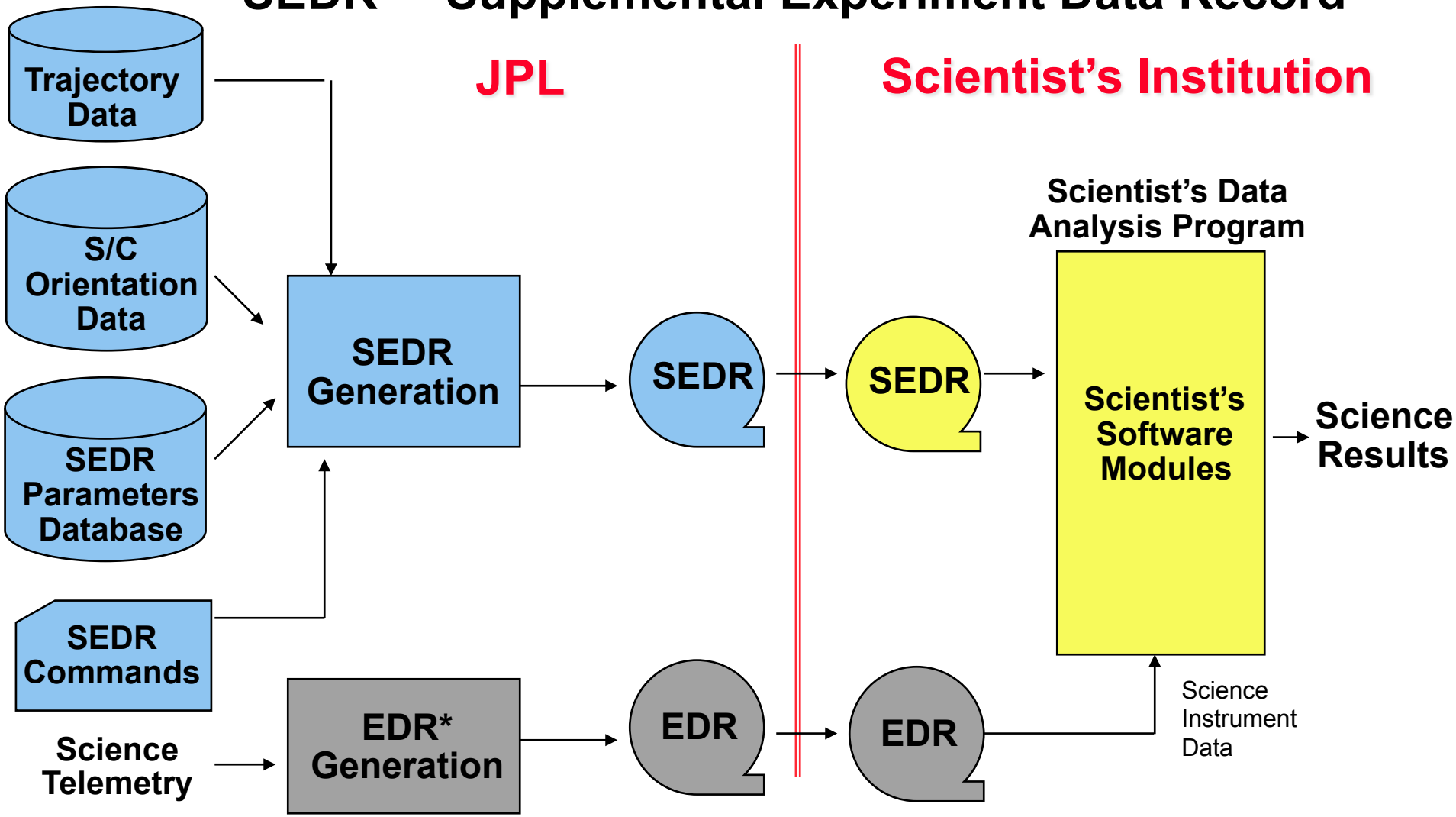
Navigation and Ancillary Information Facility

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

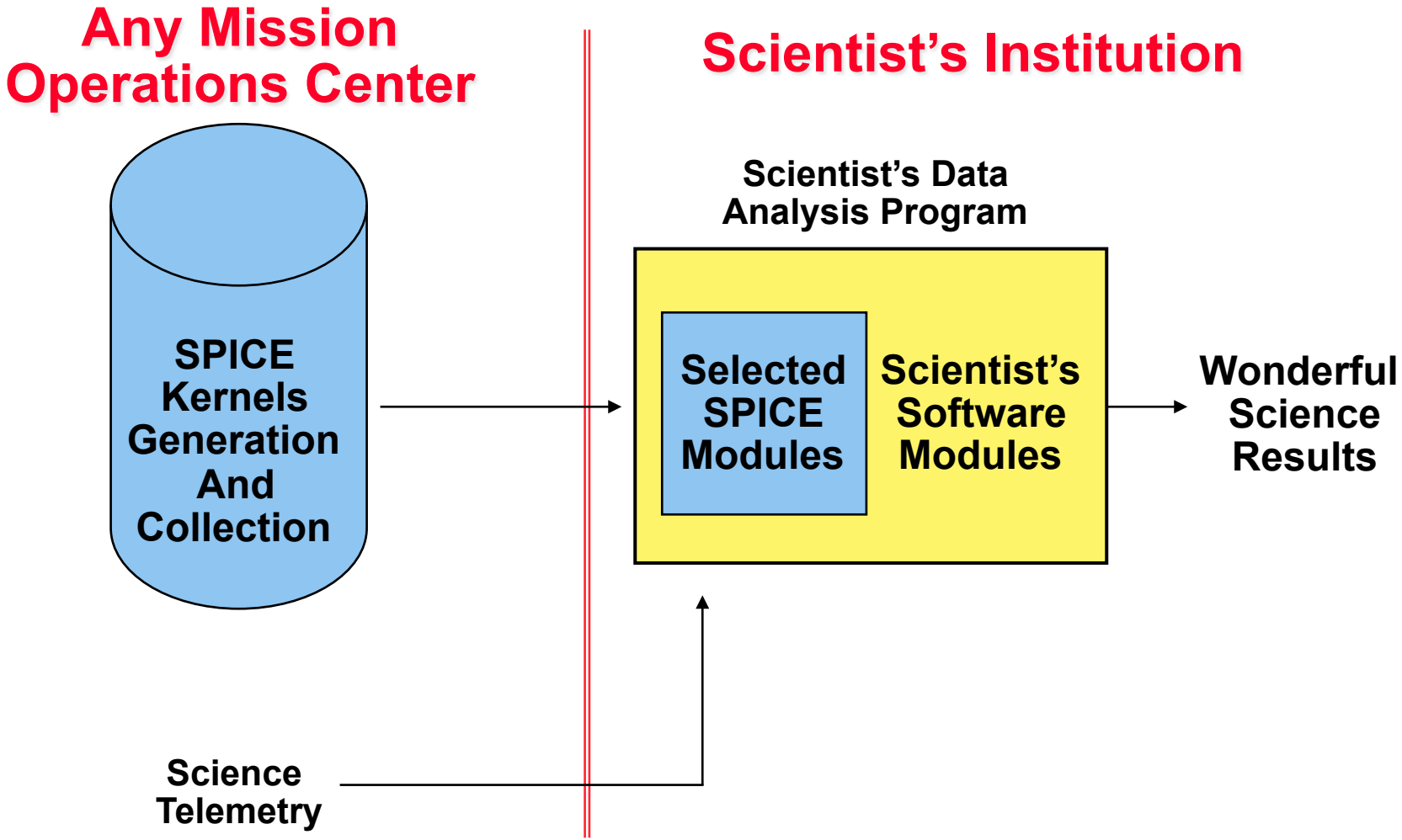
SEDR System Characteristics

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
  - Generally only one SEDR file would be produced for each period of time
    - » Result: the scientist can't get better ancillary data if/when better inputs (e.g. spacecraft trajectory or orientation) are determined
  - 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 may not fully meet science team's expectations
  - The SEDR system was not exportable to other institutions

The SPICE Idea

Navigation and Ancillary Information Facility



SPICE Benefits vs. SEDR

- **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 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:**
  - common tools and methods that can be reused on many tasks
  - good 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**

The Down Side: SPICE Detriments vs. SEDR

- End users ("consumers") 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

January 2012

Space Science Data: Two Kinds

Navigation and Ancillary Information Facility

Ancillary Data

Science Instrument Data

SPICE deals with these data to support the planning for and analysis of these data

What are "Ancillary Data?"

Navigation and Ancillary Information Facility

Reference frames

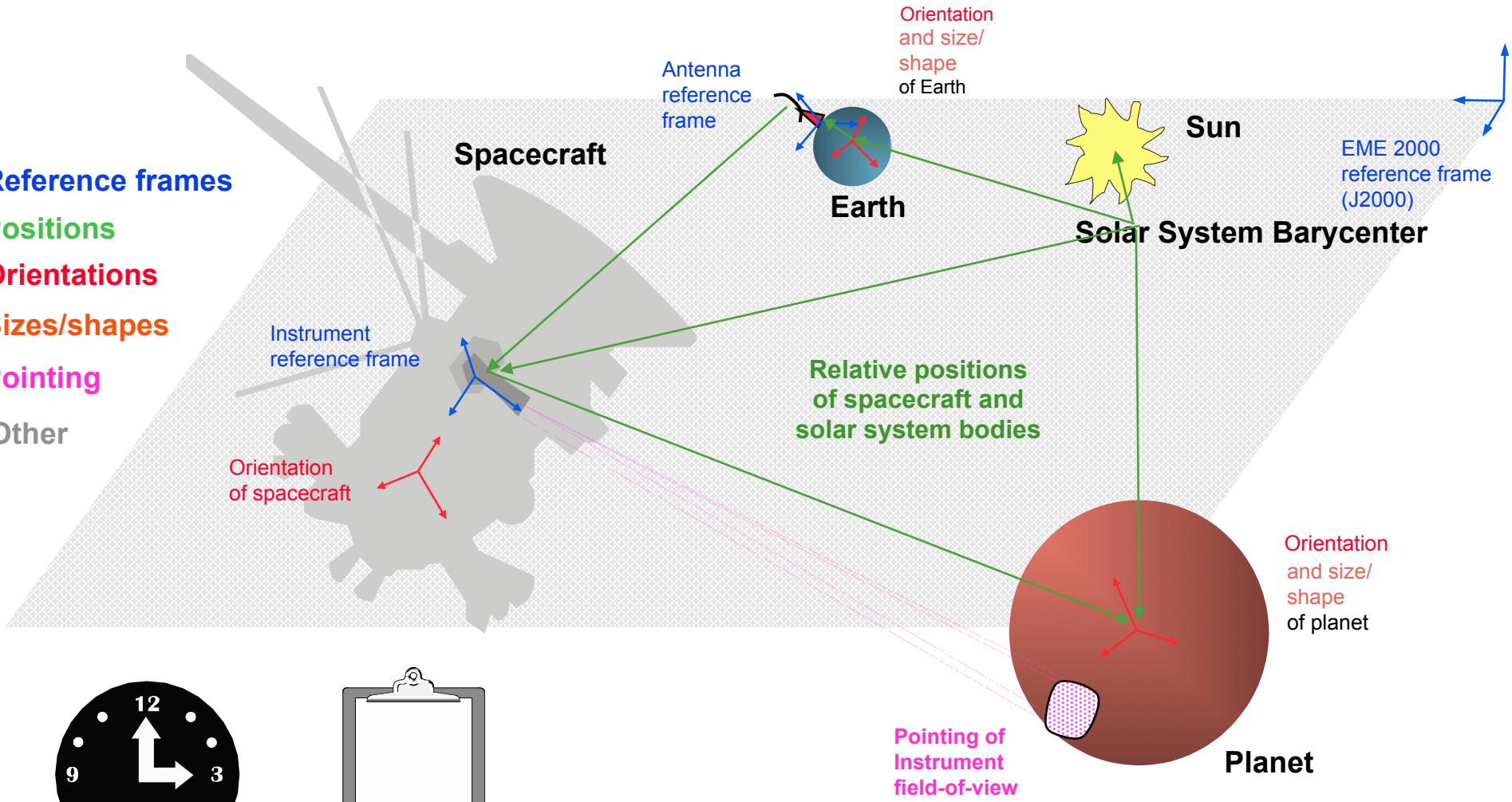
Positions

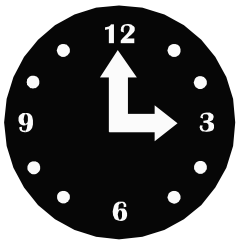
Orientations

Sizes/shapes

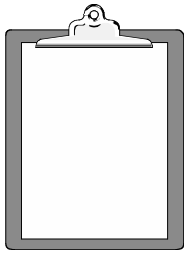
Pointing

Other





Time Conversion Calculations



Logs of Commands and Events

What are "Ancillary Data"?

- "Ancillary data" are those that help scientists and engineers determine:
  - 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
  - what events were occurring on the spacecraft or ground that might affect interpretation of science observations

- In the above we've used past tense, but doing the same functions for future times to support observation planning is equally applicable

From Where do Ancillary Data Come?

Navigation and Ancillary Information Facility

- Some come from the spacecraft
- Some come from the mission control center
- Some come from the spacecraft and instrument builders
- Some come from scientists

- SPICE is used to organize and package these data in a collection of useful, stable file types–called "kernels."
- The kernels are made available, along with SPICE Toolkit software:
  - to help scientists in the planning for and analysis of science observations, and
  - to help engineers in planning for and analysis of spacecraft and ground system operations.

Why SPICE?

- **Knowing observation geometry and events is an important element:**
  - in the design of space missions,
  - in the selection of observations,
  - and in analysis of the science data returned from the instruments.

- **Having 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

Data files (“kernels”)……..……

SPICE Toolkit software……..

Documentation ……..

Tutorials …….

Programming lessons …..

Training classes …….

User consultation ……..

Genesis of the SPICE Acronym*

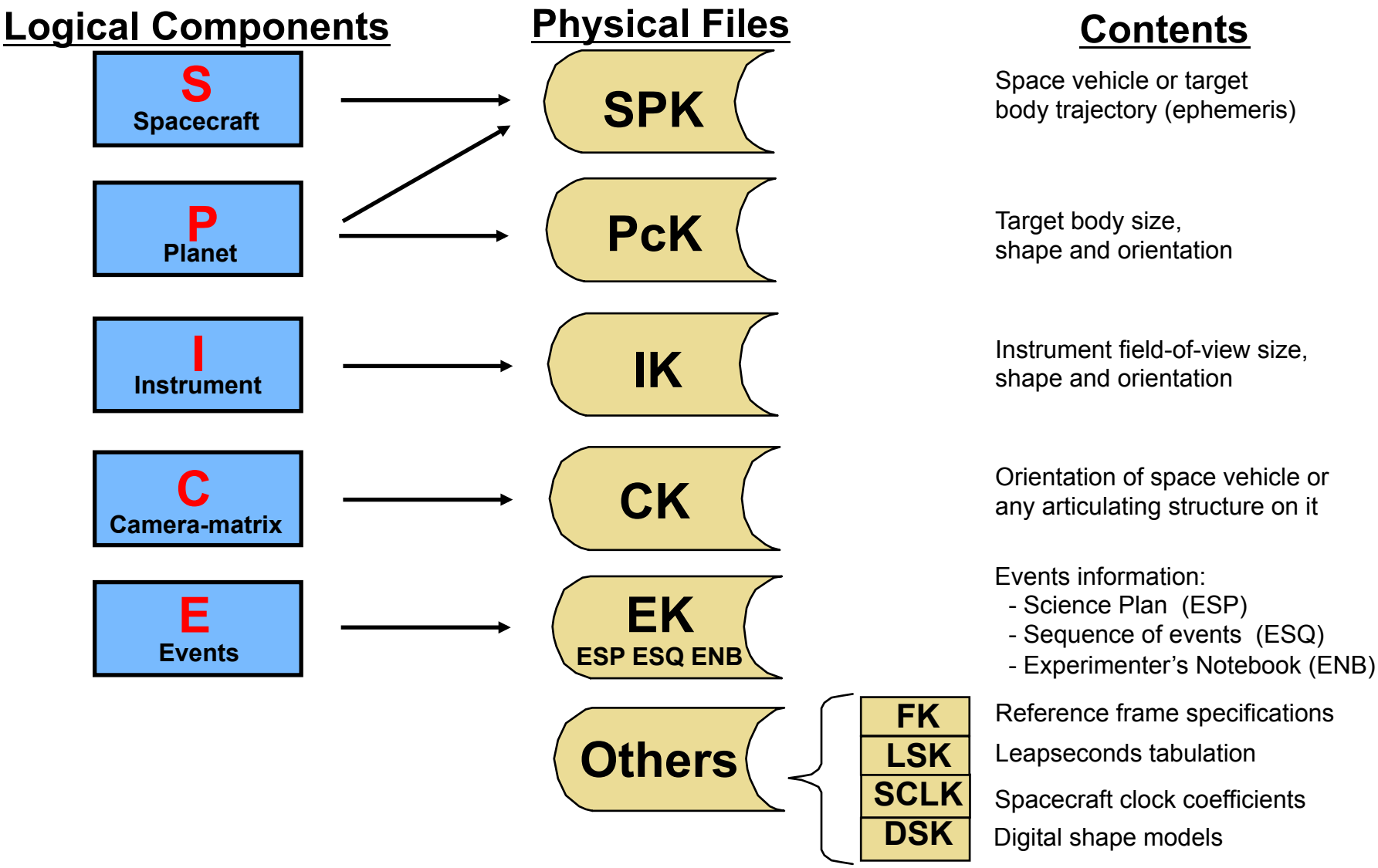
Navigation and Ancillary Information Facility

S	**S**pacecraft
P	**P**lanet
I	**I**nstrument
C	**C**-matrix
E	**E**vents

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

An Overview of SPICE Data

Navigation and Ancillary Information Facility



SPICE System Data - 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
- 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 System Data - 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

Under Development

- Shape models (DEM and tessellated plates) (DSK) [1]
- Star (sky) catalog [2]

[1] under development now
[2] development is stalled

UTC = Coordinated Universal Time ET = Ephemeris Time SCLK = Spacecraft Clock Time

SPICE Toolkit Software

Navigation and Ancillary Information Facility

Contents

- **Library of subroutines (~1500)**
  - Used within a customer's program to compute quantities derived from SPICE data files
- **Programs (~30)**
  - SPICE data production
  - SPICE data management
- **Documentation**
  - Highly annotated source code
  - Technical Reference Manuals (23)
  - User Guides

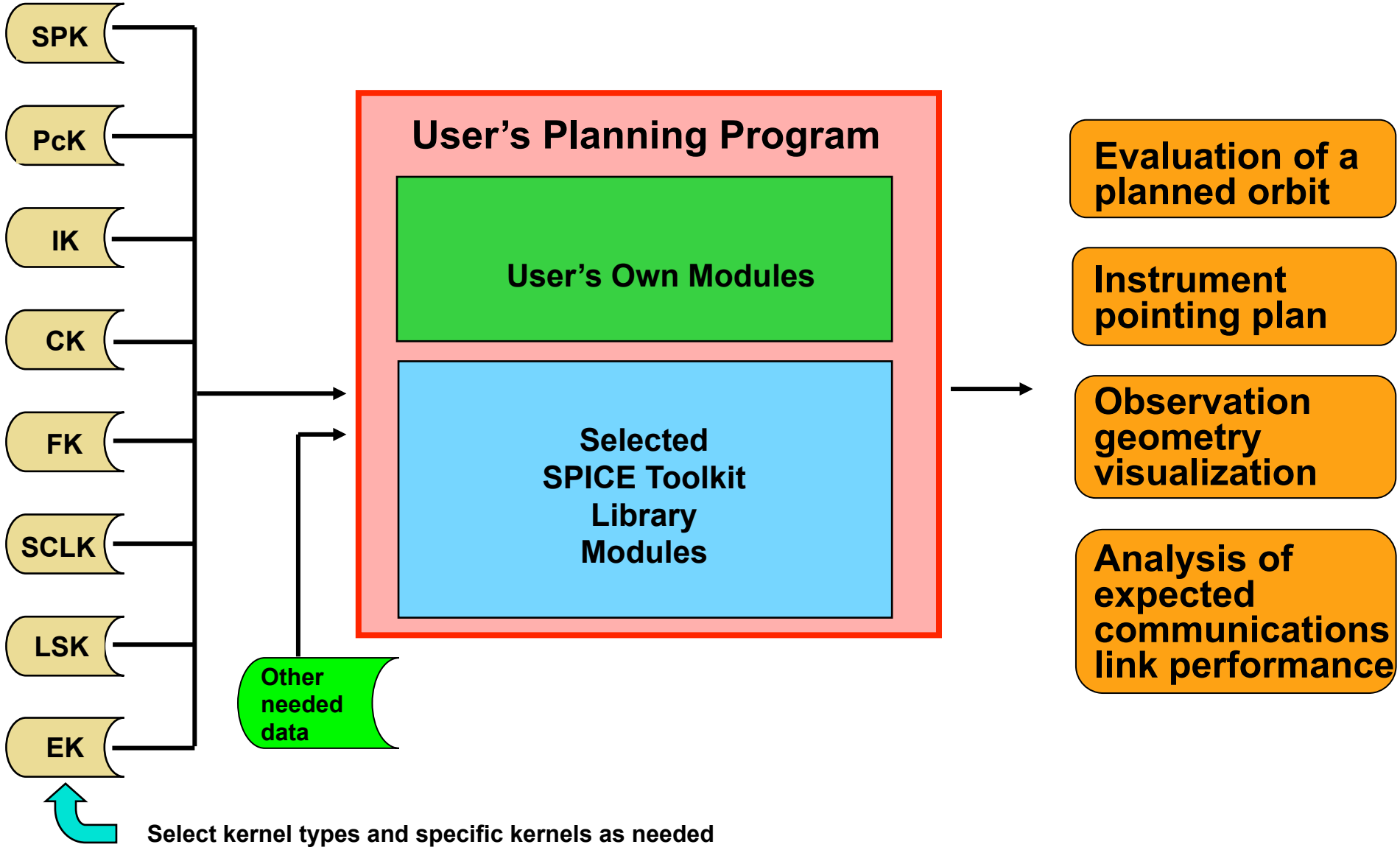
Versions

- **Four languages**
  - Fortran
  - C
  - Interactive Data Language (IDL)
  - MATLAB
  - Coming soon:
    - » Java Native Interface (JNI)
    - » Python
- **Five platforms**
  - PC/Linux
  - PC/Windows
  - Sun/Solaris
  - Mac/OSX (even iPhone)
  - Hewlett Packard/HP-UX
- **43 environments supported now**
  - Language + platform + OS + compiler

≈ 2 million source lines of code

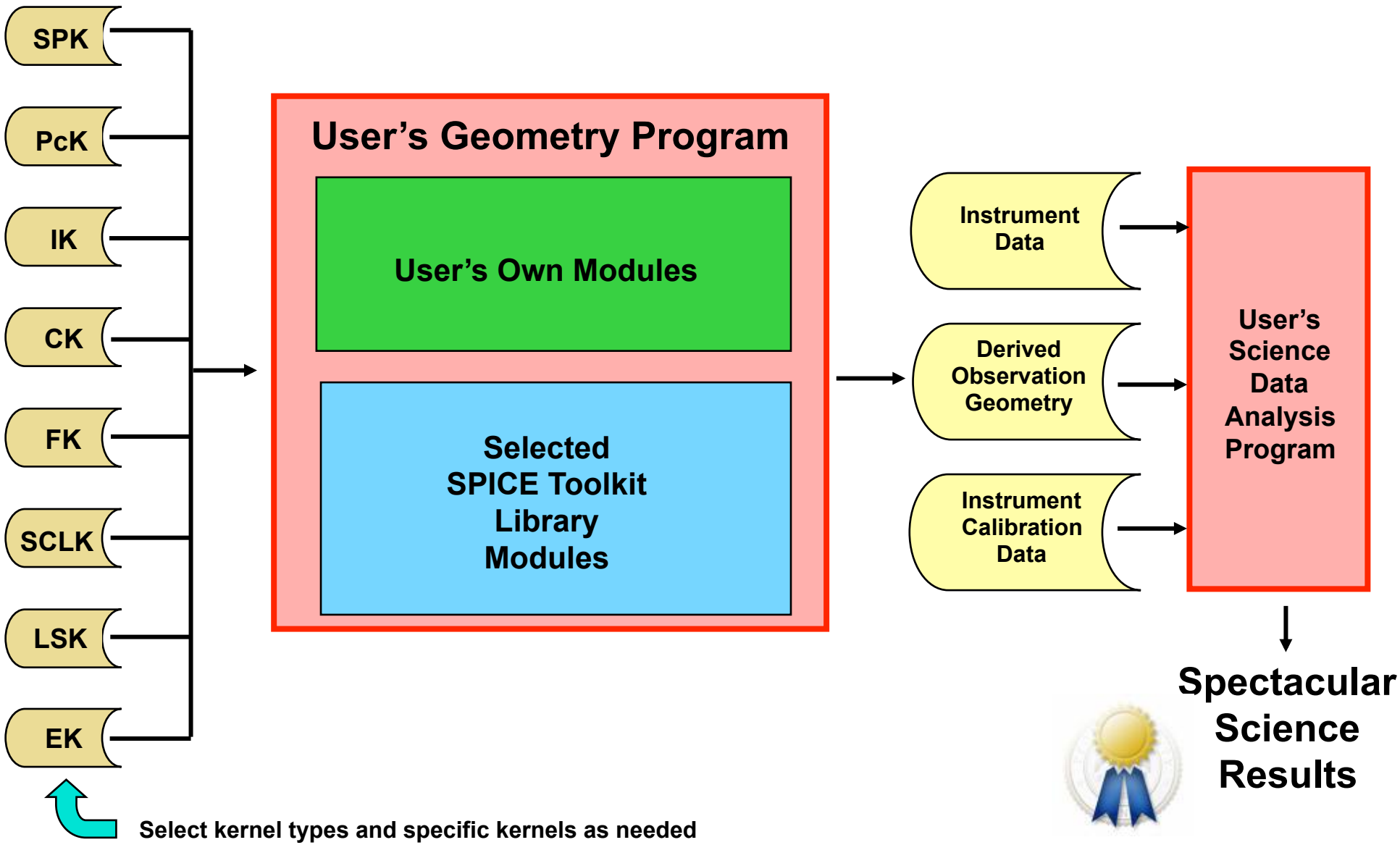
Example: Using SPICE in Mission Planning

Navigation and Ancillary Information Facility



Example:
Using SPICE in Science Data Analysis

Navigation and Ancillary Information Facility



Select kernel types and specific kernels as needed

SPICE System Characteristics - 1

- SPICE Toolkit software is portable between computers
- New Toolkits are released about every 10-15 months
- Code is 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
- SPICE tutorials are available
- “Open book” programming lessons are available
- NAIF offers a free annual training class

SPICE System Characteristics - 2

Navigation and Ancillary Information Facility

- All numeric computations use double precision numbers
- System includes built-in exception handling
  - Catches most invalid inputs
- kernel files are portable between computers
- Kernel files are separable
  - Use only those you need for a particular application
- Kernel files are extensible
  - New data “types” can be added within a family
- New kinds of kernels can be developed as needed
- SPICE users have access to JPL’s integrated ephemerides for spacecraft and natural bodies (planets, satellites, comets, asteroids)

SPICE System Characteristics - 3

Navigation and Ancillary Information Facility

- **Funding**
  - SPICE system development is funded by NASA's Planetary Science Division (PSD)
  - NASA PSD flight projects fund NAIF or others to deploy and operate SPICE in support of NASA's planetary missions
  - Foreign institutions fund their own people for deployment and operation of SPICE in support of their own projects
  - SPICE Toolkit software is free to individual end users
  - Access to SPICE kernels produced by NAIF is not restricted
    - » Includes mission operations kernels as well as those archived in the PDS
  - Consultation from NAIF is mostly restricted to paying, and paid for, users
    - » See chart near the end of this tutorial for details

- **Distribution of SPICE software and data is not restricted under U.S. Government regulations**
  - » SPICE is classified TSPA ("Technology and Software Publicly Available")
  - » There are no ITAR restrictions on data, training or consulting
    - NASA or the U.S. Government may nevertheless restrict training in specific situations

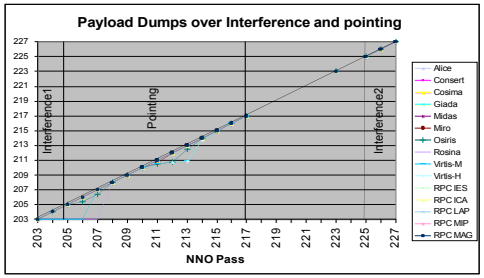
Supported Environments

- The SPICE Toolkit has been ported to a wide variety of popular "environments"
  - Each environment is characterized by…
    - » Language
    - » Hardware type (platform)
    - » Operating System
    - » Compiler (where applicable)
    - » Sometimes even selected compilation options
- 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

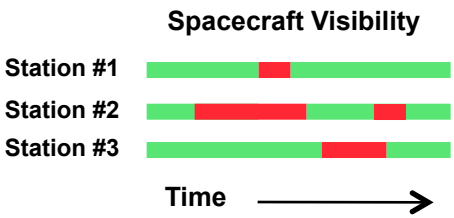
How is SPICE Used ?

Navigation and Ancillary Information Facility

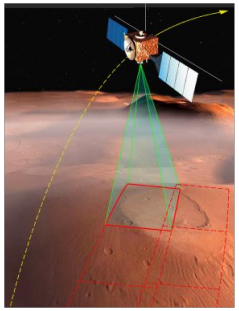
Evaluation of a planned trajectory



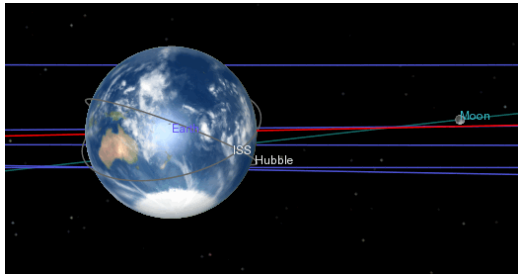
Mission engineering analyses



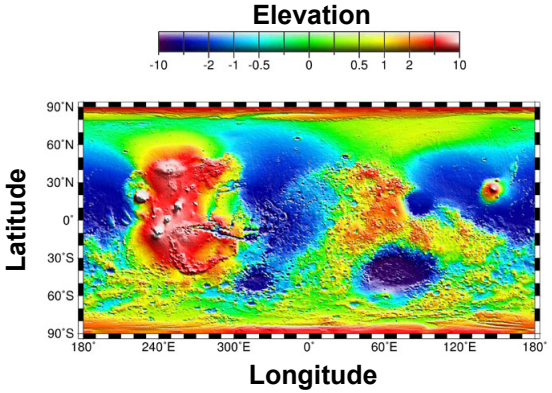
Planning an instrument pointing profile



Observation geometry visualization



Science data archiving and analysis

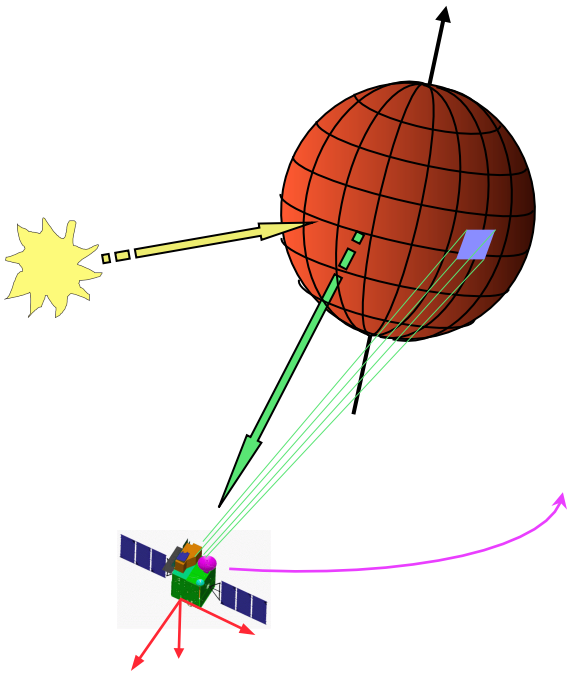


What Can One Do With SPICE?

Navigation and Ancillary Information Facility

Compute many kinds of observation geometry parameters at selected times

A Few Examples



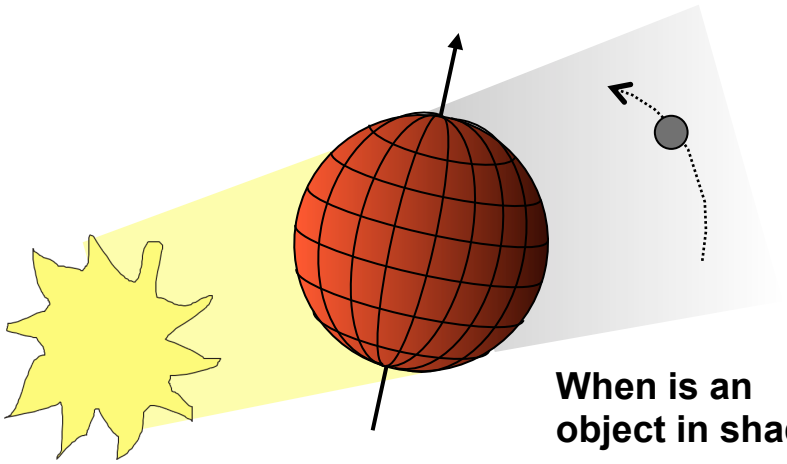
- Positions and velocities of planets, satellites, comets, asteroids and spacecraft
- Size, shape and orientation of planets, satellites, comets and asteroids
- Orientation of a spacecraft and its various moving structures
- Instrument field-of-view location on a planet's surface or atmosphere

What Can One Do With SPICE?

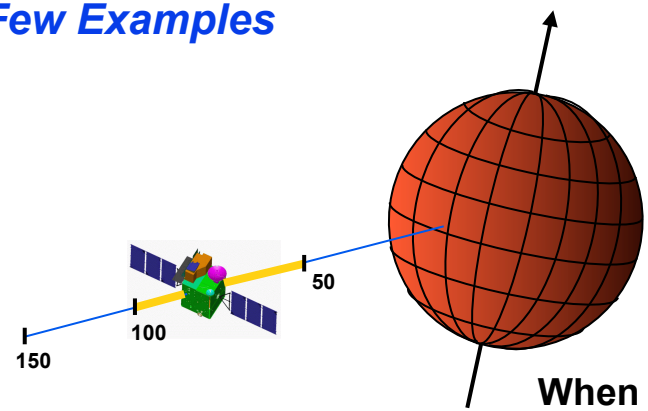
Navigation and Ancillary Information Facility

Find times when a selected "geometric event" occurs, or when a selected "geometric condition" exists

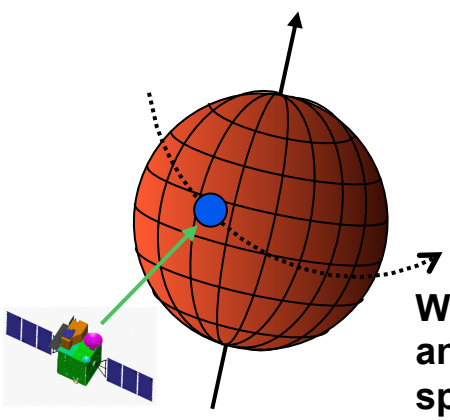
A Few Examples



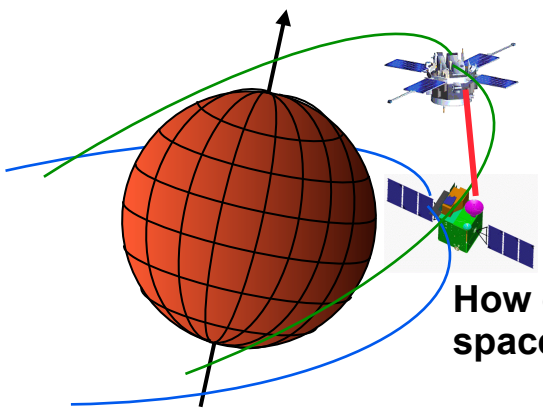
When is an object in shadow?



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?



How close will two spacecraft get?

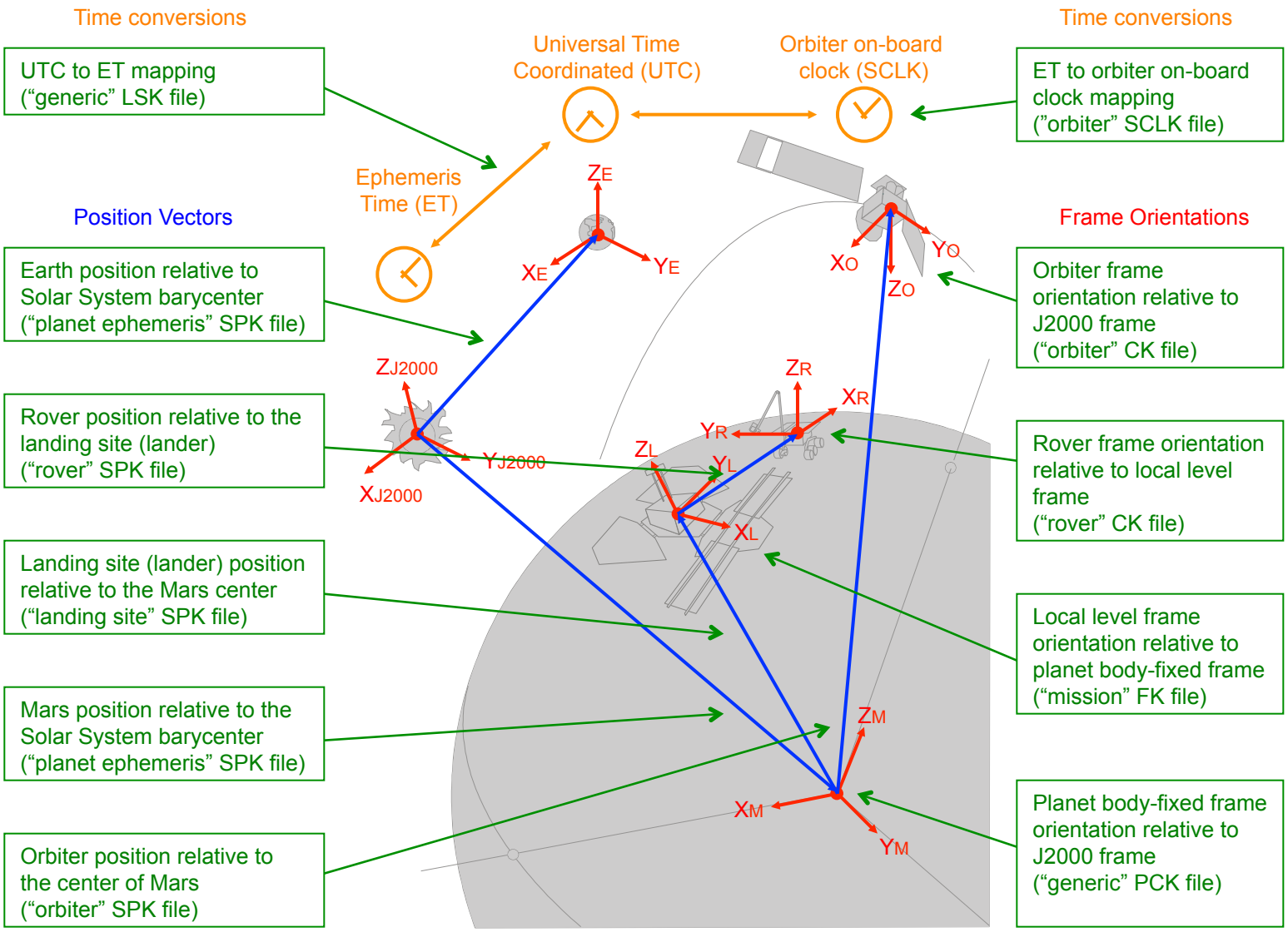
What "Vehicle" Types Can Be Supported?

- **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 observers
  - Tracking station needs

* Not yet demonstrated

Global SPICE Geometry

Navigation and Ancillary Information Facility



Orbiter Geometry

Navigation and Ancillary Information Facility

Position Vectors

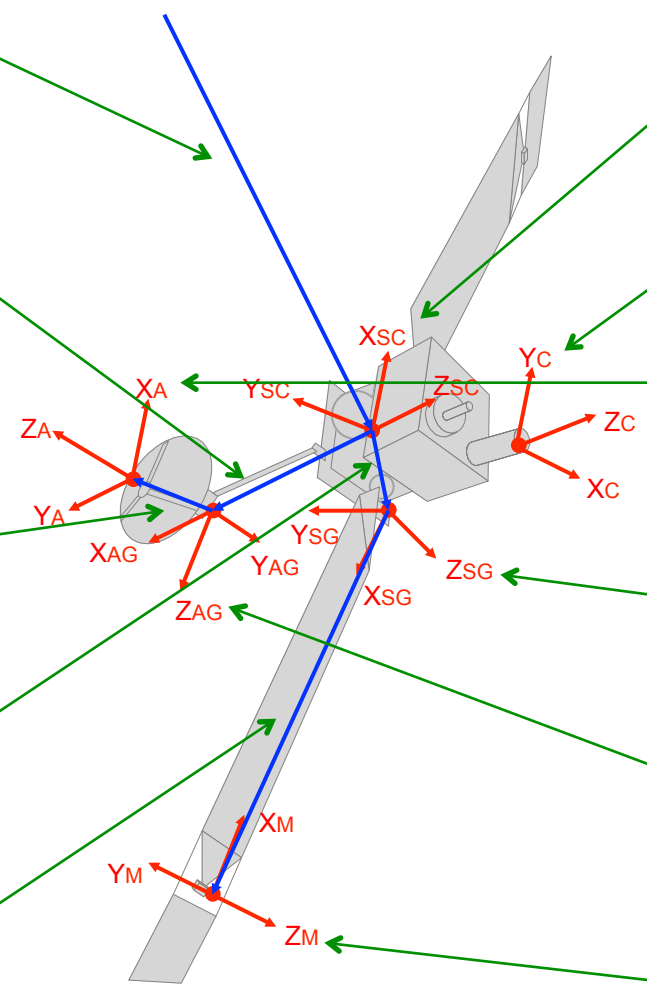
Spacecraft position relative to planet center (“spacecraft” SPK file)

High gain antenna gimbal position relative to spacecraft (“structures” SPK file)

High gain antenna phase center location relative to high gain antenna gimbal (“structures” SPK file)

Solar array gimbal position relative to spacecraft center (“structures” SPK file)

Magnetometer position relative to solar array gimbal (“structures” SPK file)



Frame Orientations

Spacecraft frame orientation relative to inertial frame (“spacecraft” CK file)

Camera frame orientation relative to spacecraft frame (“mission” FK file)

High gain antenna frame orientation relative to high gain antenna gimbal frame (“mission” FK file)

Solar array gimbal frame orientation relative to spacecraft frame (“solar array” CK file)

High gain antenna gimbal frame orientation relative to spacecraft frame (“antenna” CK file)

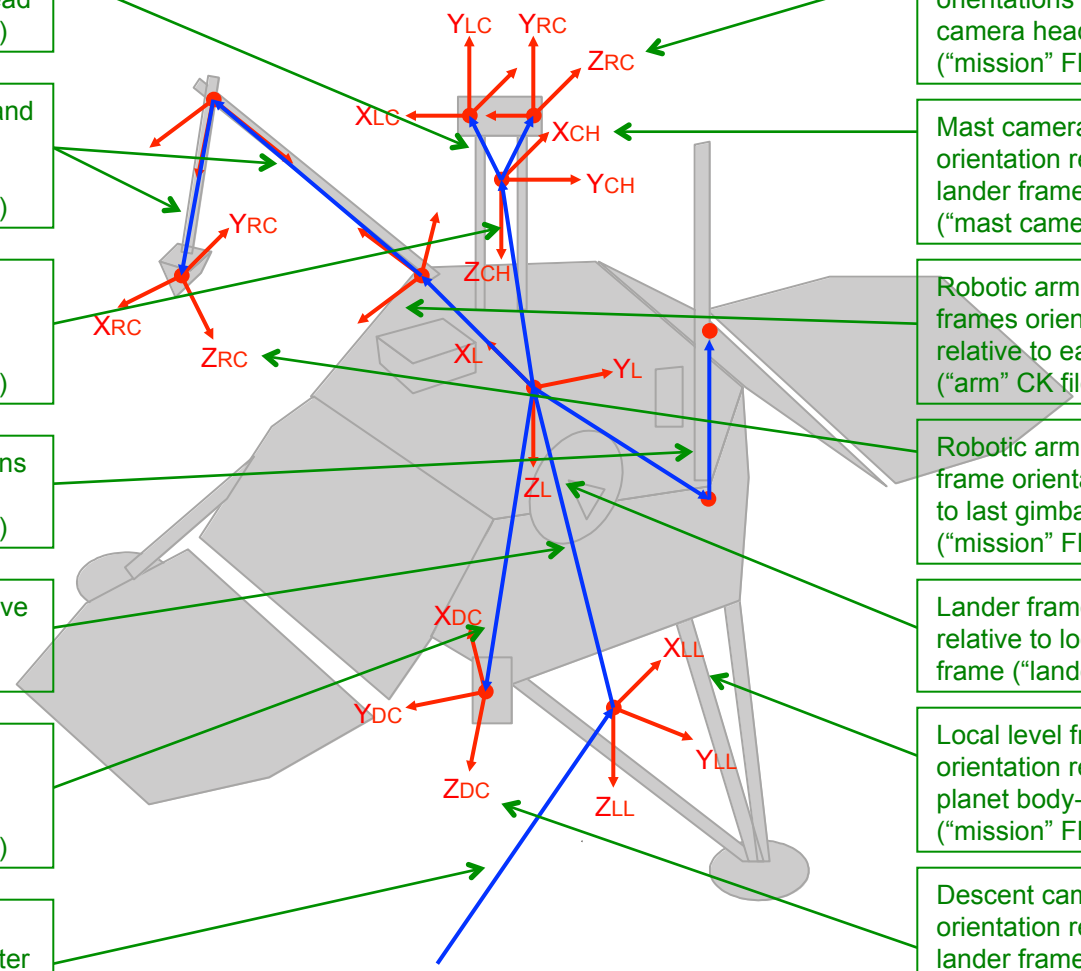
Magnetometer frame orientation relative to solar array gimbal frame (“mission” FK file)

Lander Geometry

Navigation and Ancillary Information Facility

Position Vectors

- Left and right mast camera positions relative to camera head (“structures” SPK file)
- Robotic arm gimbal and camera relative positions (“structures” SPK file)
- Mast camera head position relative to lander (“structures” SPK file)
- Meteo sensor positions relative to lander (“structures” SPK file)
- Lander position relative to landing site (“lander” SPK file)
- Descent camera position relative to lander (“structures” SPK file)
- Landing site position relative to planet center (“landing site” SPK file)



Frame Orientations

- Left and right mast camera frame orientations relative to camera head frame (“mission” FK file)
- Mast camera head frame orientation relative to lander frame (“mast camera” CK file)
- Robotic arm gimbal frames orientations relative to each other (“arm” CK file)
- Robotic arm camera frame orientation relative to last gimbal frame (“mission” FK file)
- Lander frame orientation relative to local level frame (“lander” CK file)
- Local level frame orientation relative to planet body-fixed frame (“mission” FK file)
- Descent camera frame orientation relative to lander frame (“mission” FK file)

Rover Geometry

Navigation and Ancillary Information Facility

Position Vectors

Left and right mast camera center positions relative to end of mast (“structures” SPK file)

End of mast position relative to elbow gimbal (“structures” SPK file)

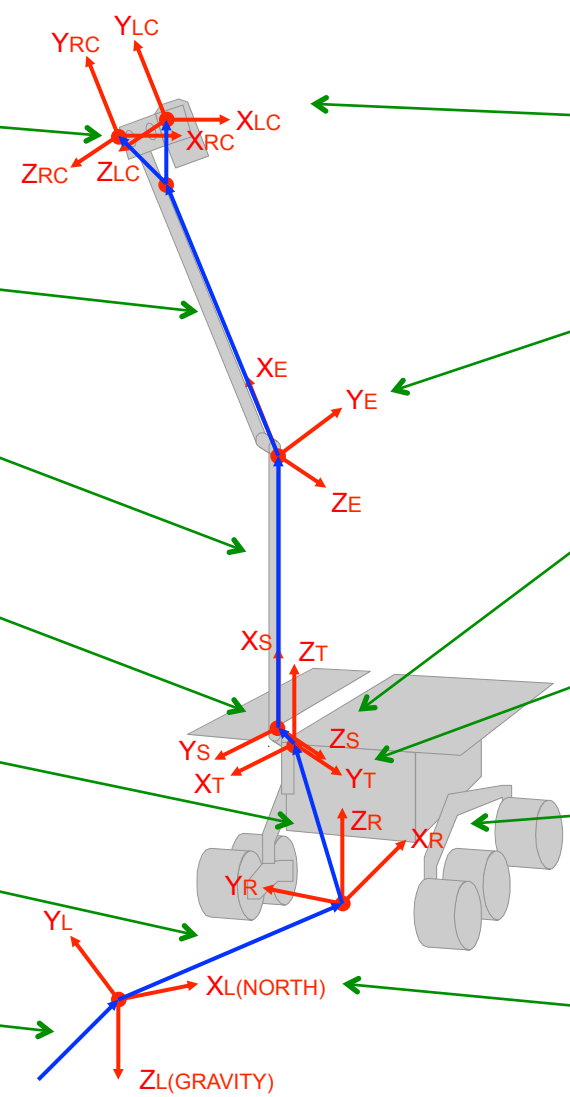
Mast elbow gimbal position relative to shoulder gimbal (“structures” SPK file)

Mast shoulder gimbal position relative to torso gimbal (“structures” SPK file)

Mast torso gimbal position relative to rover (“structures” SPK file)

Rover position relative to landing site (“rover” SPK file)

Landing site position relative to planet center (“landing site” SPK file)



Frame Orientations

Left and right mast camera frames orientation relative to mast elbow frame (“mission” FK file)

Mast elbow frame orientation relative to mast shoulder frame (“mast” CK file)

Mast shoulder frame orientation relative to mast torso frame (“mast” CK file)

Mast torso frame orientation relative to rover frame (“mast” CK file)

Rover frame orientation relative to local level frame (“rover” CK file)

Local level frame orientation relative to planet body-fixed rotating frame (“mission” FK file)

Flight Projects Using SPICE

Navigation and Ancillary Information Facility

12/28/11

Data Restorations	*Past Users*	*Current Users*	*Anticipated*
Apollo 15, 16 [L]	Magellan [L]	Cassini Orbiter	LADEE
Mariner 9 [L]	Clementine (NRL)	Mars Odyssey	Mars 2016/2018 (ESA and NASA))
Mariner 10 [L]	Mars Observer [F]	Mars Exploration Rover	NASA Discovery Program
Viking Orbiters [L]	Mars 96 [F] (RSA)	Mars Reconnaissance Orbiter	NASA New Frontiers Program
Viking Landers [L]	Mars Pathfinder	DAWN	BepiColombo (ESA)
Pioner 10/11 [L]	Mars Climate Orbiter [F]	Mars Science Lab	
Haley armada [L]	Mars Polar Lander [F]	Juno	
Phobos 2 [L] (RSA)	NEAR	SMAP	*Future ?*
Ulysses [L]	Deep Space 1	MAVEN	Jupiter (NASA)
Voyagers [L]	Galileo	GRAIL	Jupiter (ESA)
Lunar Orbiter [L]	Genesis	Osiris-Rex	Akatsuki (JAXA)
	Deep Impact		Luna-Resurs (ISRO/RSA)
	Huygens Probe (ESA)	Lunar Reconnaissance Orbiter	Luna-Glob (RSA)
	Stardust/NExT	New Horizons	
	Mars Global Surveyor	Messenger	
[L] = limited use	Phoenix		*Examples of Other SPICE Users*
[S] = special services	EPOXI	Mars Express (ESA)	NASA AMMOS
[F] = mission failed	Hubble Space Telescope [S]	Venus Express (ESA)	NASA Deep Space Network
	ISO [S] (ESA)	Rosetta (ESA)	STEREO
	CONTOUR [F]		Spitzer Space Telescope
	Space VLBI [L] (multinational)	Hayabusa (JAXA)	Kepler
	Smart-1 (ESA)	Kaguya (JAXA)	Planck (ESA)
	Chandrayaan-1 (ISRO)		WISE
	LCROSS	Planetary Data System	IBEX
	Phobos Sample Return (RSA)	Planetary Science Archive (ESA)	

- NAIF has/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 and students with using SPICE data that have been officially archived at the NAIF Node of the PDS.
- NAIF has NASA funding to support ESA/RSA in SPICE operations and review of SPICE archive, 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/had modest PDS-supplied funding to consult only on assembly of a SPICE archive.
- NAIF has PDS funding to help scientists and students with using SPICE data that have been officially archived at the NAIF Node of the PDS.
- User consultation is provided by ESA's Science Operations Department.

Flight Projects Using SPICE

Navigation and Ancillary Information Facility

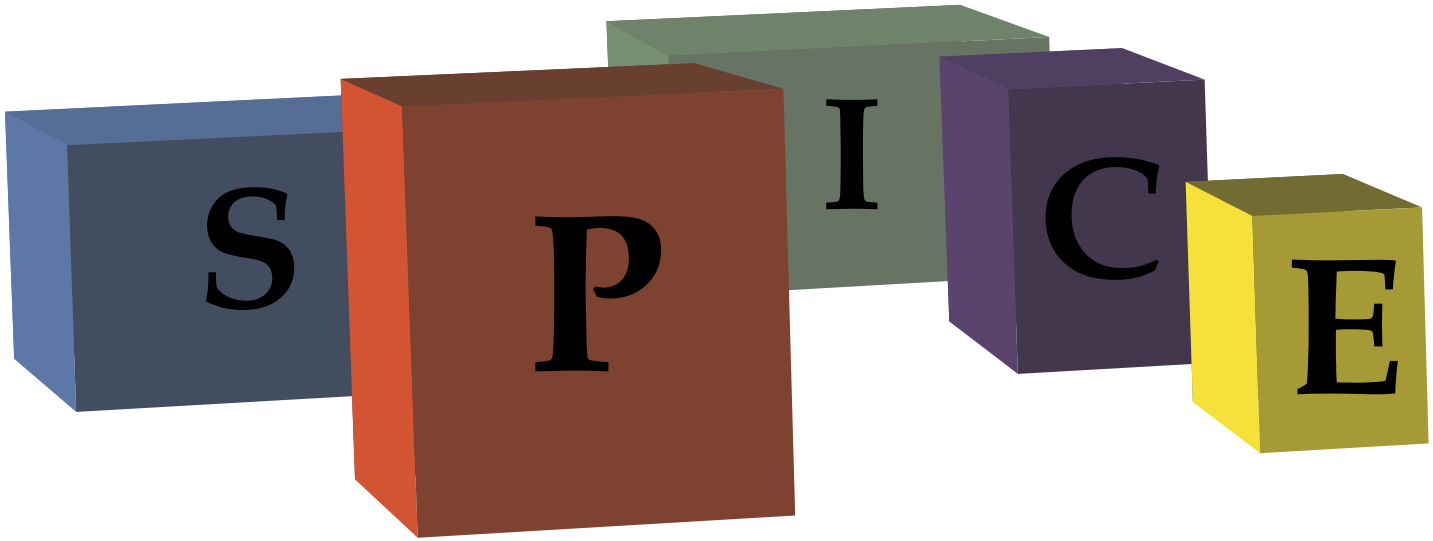
Data Restorations	*Past Users*	*Current Users*	*Anticipated*
Apollo 15, 16 [L]	Magellan [L]	Cassini Orbiter	LADEE
Mariner 9 [L]	Clementine (NRL)	Mars Odyssey	NASA Discovery Program
Mariner 10 [L]	Mars Observer [F]	Mars Exploration Rover	NASA New Frontiers Program
Viking Orbiters [L]	Mars 96 [F] (RSA)	Mars Reconnaissance Orbiter	BepiColombo (ESA)
Viking Landers [L]	Mars Pathfinder	DAWN	Osiris-Rex
Pioner 10/11/12 [L]	Mars Climate Orbiter [F]	Mars Science Lab	
Haley armada [L]	Mars Polar Lander [F]	Juno	*Examples of Possible Future Use ?*
Phobos 2 [L] (RSA)	NEAR	SMAP	Jupiter (NASA)
Ulysses [L]	Deep Space 1	MAVEN	Jupiter (ESA)
Voyagers [L]	Galileo	GRAIL	Akatsuki (JAXA)
Lunar Orbiter [L]	Genesis	Lunar Reconnaissance Orbiter	Luna-Resurs (ISRO/RSA)
Helios 1,2 [L]	Deep Impact	New Horizons	Luna-Glob (RSA)
	Huygens Probe (ESA)	Messenger	Solar Probe
	Stardust/NExT	Mars Express (ESA)	Solar Orbiter (ESA)
	Mars Global Surveyor	Venus Express (ESA)	
	Phoenix	Rosetta (ESA)	
	EPOXI		*Examples of Other SPICE Users*
	ISO [S] (ESA)		NASA Deep Space Network
	CONTOUR [F]		NASA AMMOS
	Space VLBI [L] (multinational)		STEREO
[L] = limited use	Smart-1 (ESA)		Spitzer Space Telescope
[S] = special services	Chandrayaan-1 (ISRO)		Kepler
[F] = mission failed	Hayabusa (JAXA)		Hubble Space Telescope [S]
	Kaguya (JAXA)	Planetary Data System	WISE
	Phobos Sample Return (RSA)	Planetary Science Archive (ESA)	IBEX

- NAIF has/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 and students with using SPICE data that have been officially archived at the NAIF Node of the PDS.
- NAIF has NASA funding to support ESA/RSA in SPICE deployment and review of a SPICE archive, 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/had modest PDS-supplied funding to consult on assembly of a SPICE archive.
- NAIF has PDS funding to help scientists and students with using SPICE data that have been officially archived at the NAIF Node of the PDS.
- User consultation is provided by ESA's Science Operations Department.

Building Blocks for Your Applications

Navigation and Ancillary Information Facility

The "SPICE" ancillary information system can serve as a set of blocks for building tools that can help execute a multi-mission, international space exploration program



SPICE: the ancillary information system that NAIF builds and often operates.
NAIF: the JPL entity responsible for development and deployment of SPICE.
NAIF Node of the PDS: one responsibility of the NAIF Group--archiving and providing long-term access to SPICE data for the worldwide science community.

Navigation and Ancillary Information Facility

Fundamental Concepts

January 2012

Topics

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

Preface

- 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

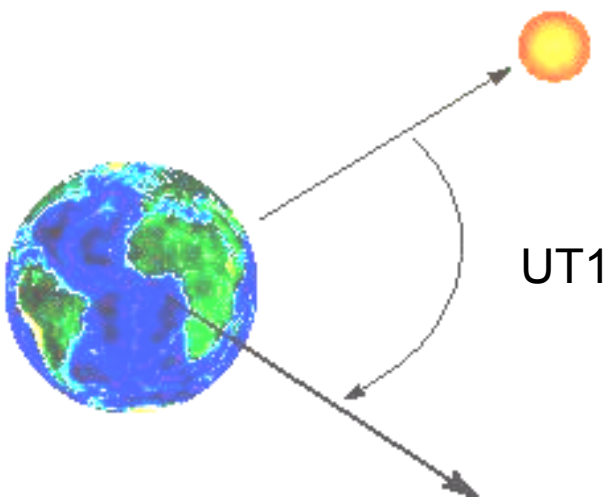
- **An epoch is an instant in time specified by some singular event**
  - Passage of a star across your zenith meridian
  - Eclipse of a spacecraft signal as it passes behind a solid body
- **Clocks**
  - Clocks count epochs specified by events such as: "regular" oscillations of a pendulum, quartz crystal, 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

- 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
  - UTC + 10 seconds + number of leap seconds = TAI
    - » Valid only after Jan 01, 1972

Astronomical Time – UT1

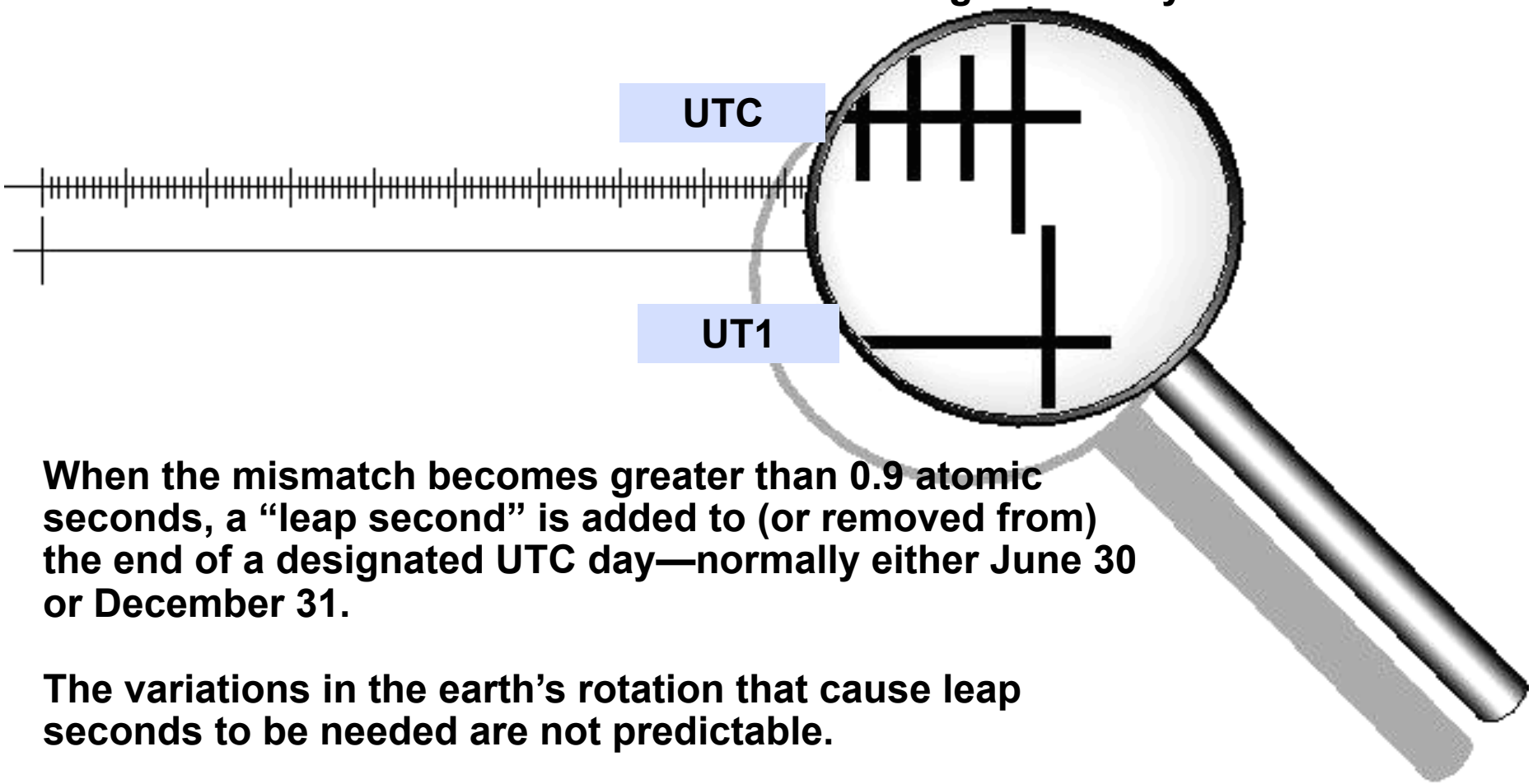
- Astronomical Time (UT1) is an hour representation of the angle between the Greenwich zenith meridian and the location of the "computed mean sun."
- Used prior to atomic time for civil time keeping



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.



When the mismatch becomes greater than 0.9 atomic seconds, a "leap second" is added to (or removed from) the end of a designated UTC day—normally either June 30 or December 31.

The variations in the earth's rotation that cause leap seconds to be needed are not predictable.

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

- **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 ideal time at the Solar System Barycenter (TCB) by a scale factor, so that TDB advances on average at the same rate as TAI.**

Terrestrial Dynamical Time

- **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 the same reference epoch (approximately 1 Jan 2000, 12:00:00 at Greenwich England)**
- **TDB and TDT advance at different rates.**
  - Variations are small: ~ 1.6 milliseconds
  - Variations are periodic with a period of 1 sidereal year (to first order)
  - Variations are due to relativistic effects
    - » TDB = TDT + 0.001657 sin(E + 0.01671sin(E))
- **Use of TDT in the SPICE system is quite limited.**
  - SCLK kernels
  - Duration computations involving UTC

Spacecraft Clocks

- 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 spacecraft clock
- 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

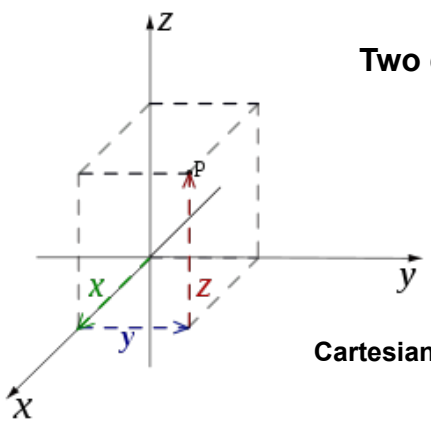
- **SCLK string formats vary from one spacecraft clock to the next.**
  - **Cassini: Maximum reading for partition 1 = 1/4294967295.255**
    - » **Partition number: 1**
    - » **Seconds: 4294967295**
    - » **Ticks (for Cassini, unit = 1/256 second): 255**
  - **Galileo: Maximum reading for partition 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

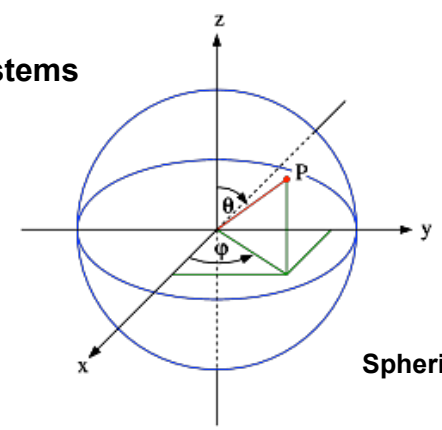
Navigation and Ancillary Information Facility

- **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

- **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 <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.

Types of Reference Frames

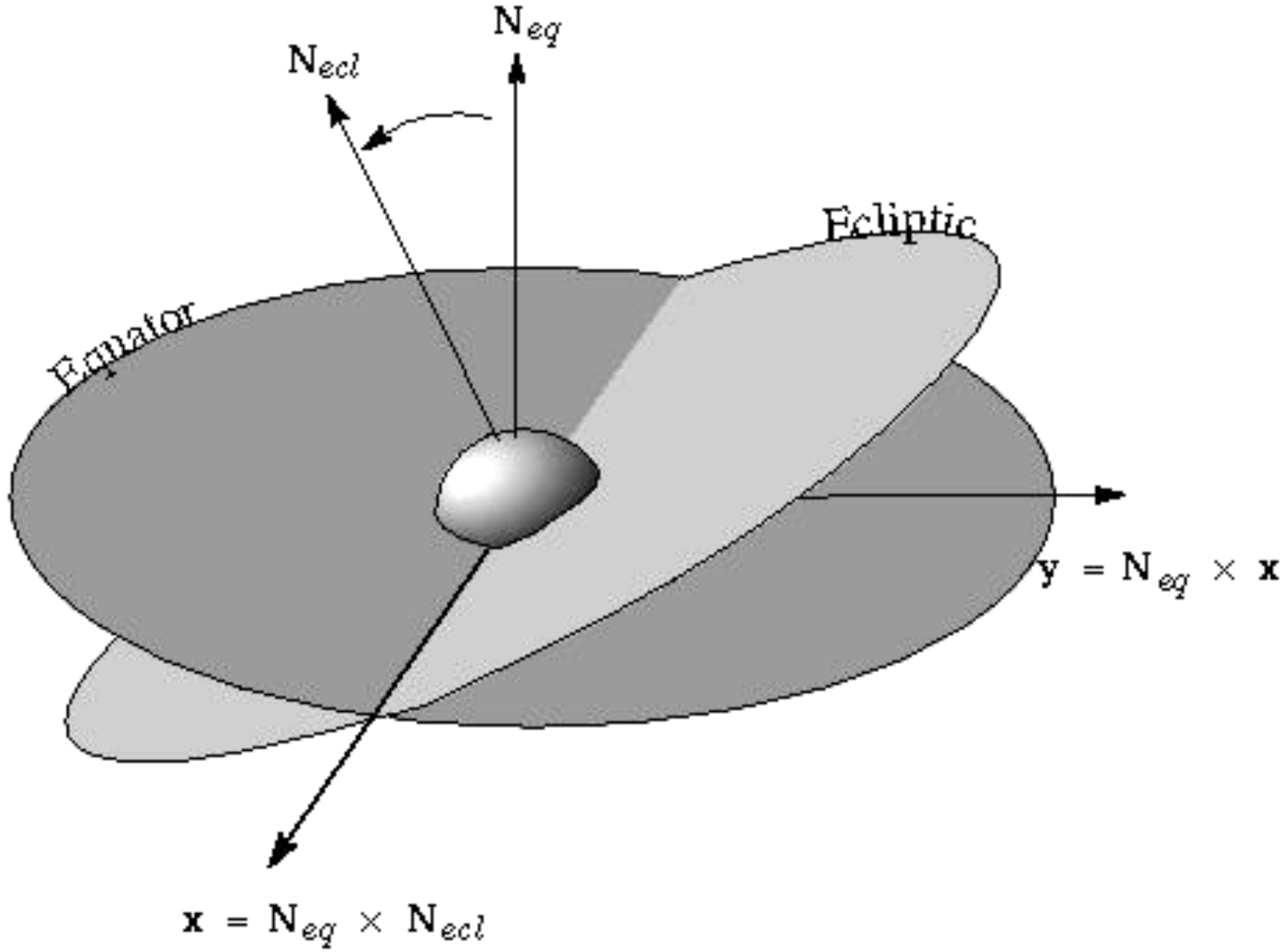
- **Inertial**
  - Non-rotating
    - » With respect to fixed stars
  - Non-accelerating origin
    - » Velocity is typically non-zero; acceleration is negligible
  - Examples:
    - » J2000 (also called ICRF), B1950
- **Non-Inertial**
  - Examples
    - » Body-fixed
      - Centered at body center
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

Frames Defined by Dynamics

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

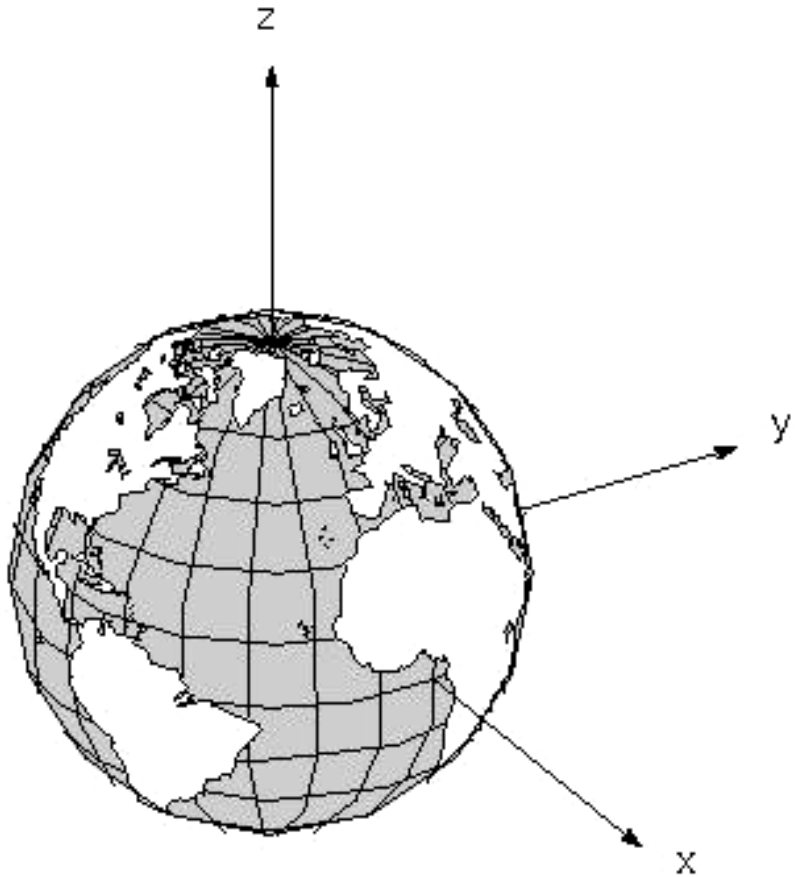
Ecliptic Plane

J2000 (ICRF) Frame



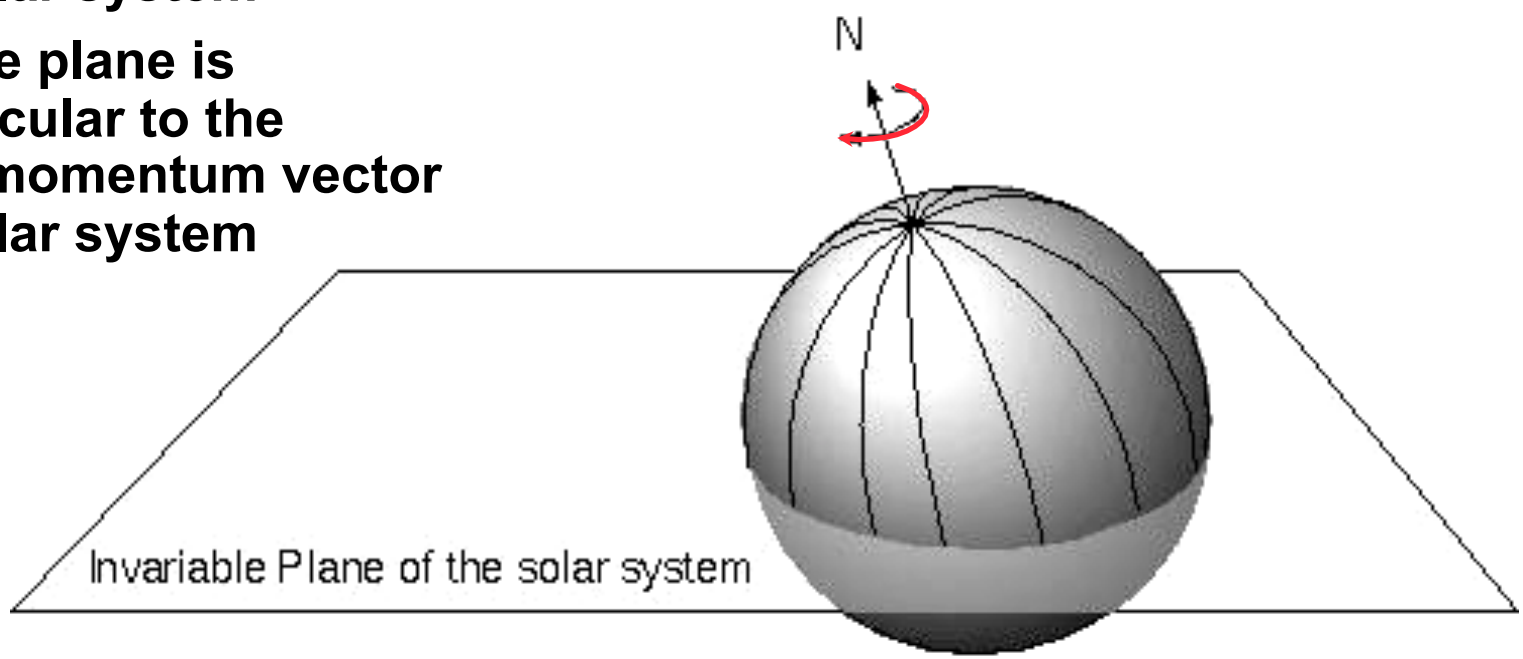
Rotating Frames

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



IAU Body-fixed Frames

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

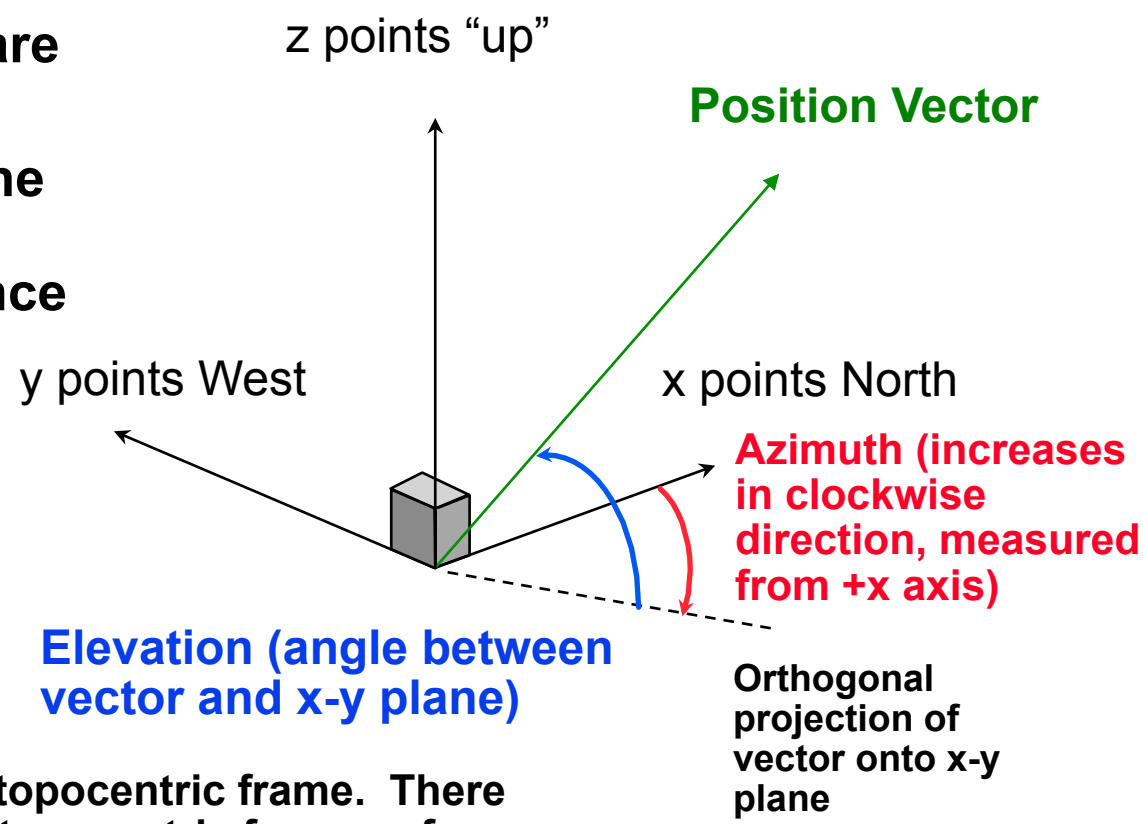


IAU = International Astronomical Union

Topocentric Frames

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.

Spacecraft and Instrument Frames

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

State Vectors

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

Transforming States

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

* Assuming both frames are right-handed

Transforming States

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

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

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

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

Coordinate Systems

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

Aberration Corrections: Introduction

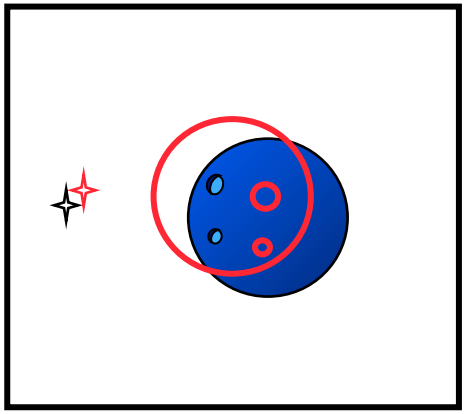
- **Within the SPICE system, "aberration corrections" are adjustments made to state vectors and time-dependent reference frames to accurately reflect the apparent–as opposed to the actual–state and attitude of a target object as seen from a specified observer at a specified time.**
  - Actual, uncorrected 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?

Example: Predicted vs Actual Photo

Navigation and Ancillary Information Facility

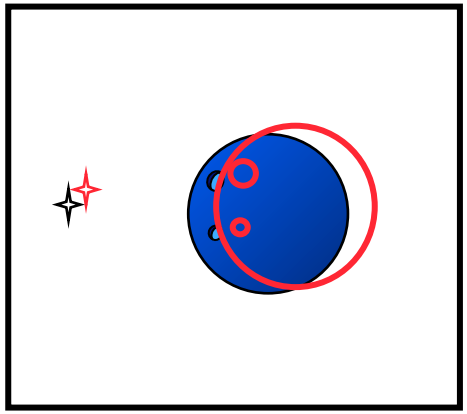
We compare the predicted appearance of a photograph from an optical camera against the actual photograph. We show three predictions derived using different aberration corrections: NONE, LT ("light time only"), and LT+S ("light time plus stellar aberration").

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.



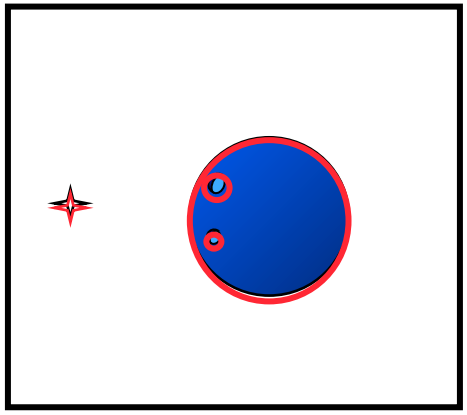
NONE

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



LT

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

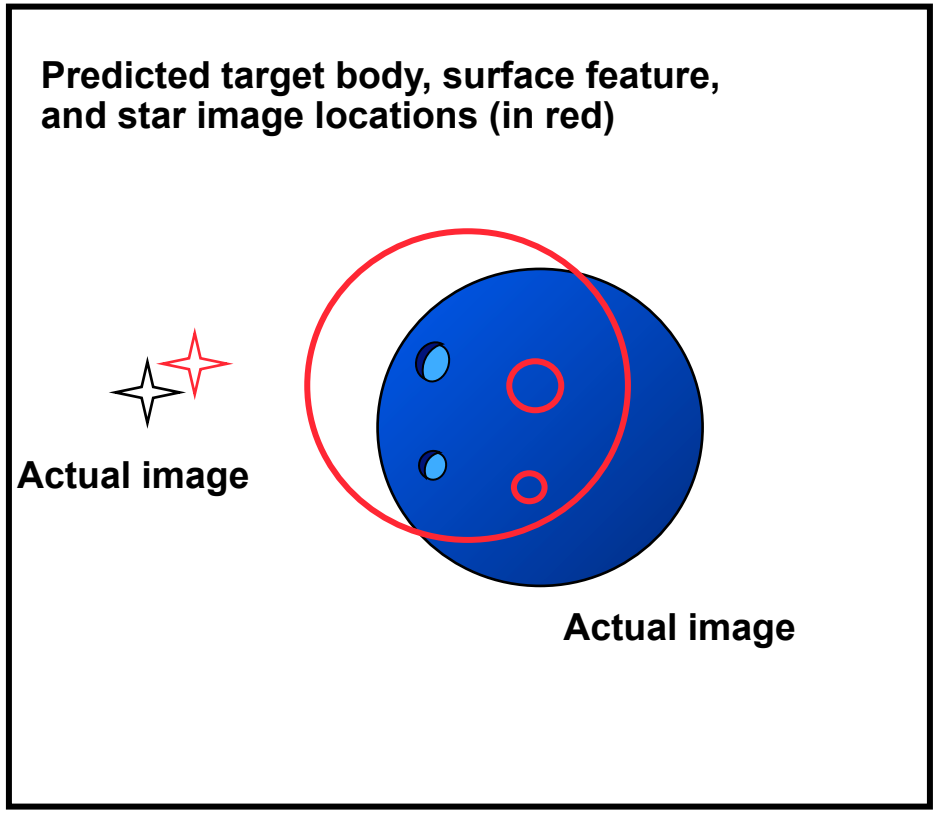


LT+S

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

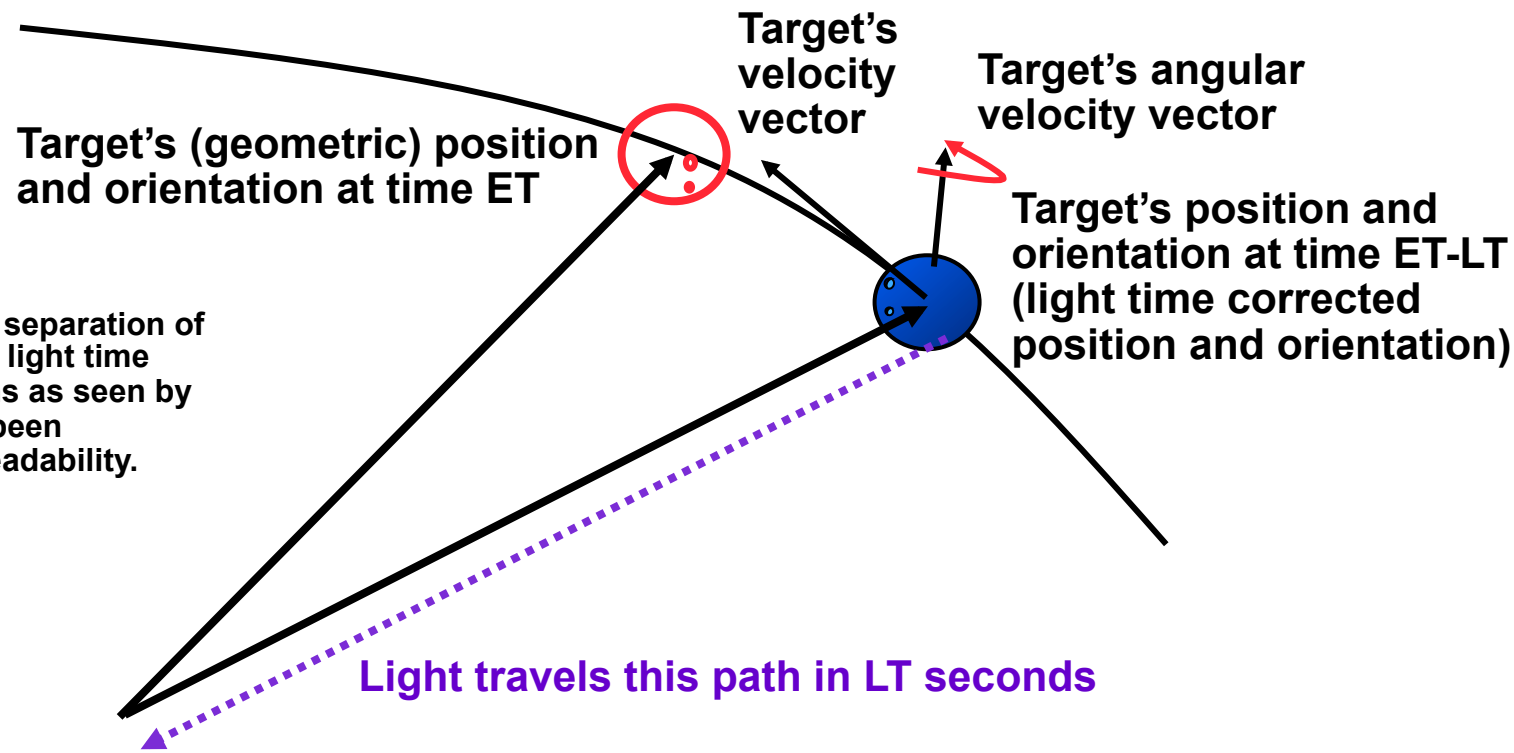
Navigation and Ancillary Information Facility



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



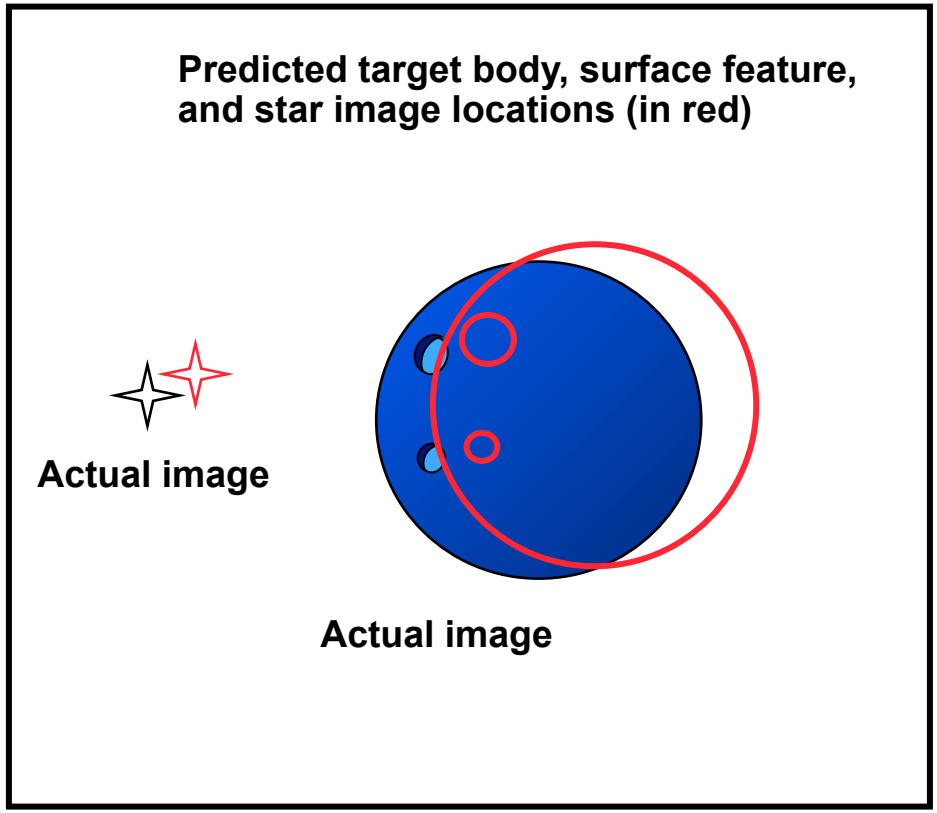
Note: The angular separation of the geometric and light time corrected positions as seen by the observer has been exaggerated for readability.

Observer's position at time ET

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

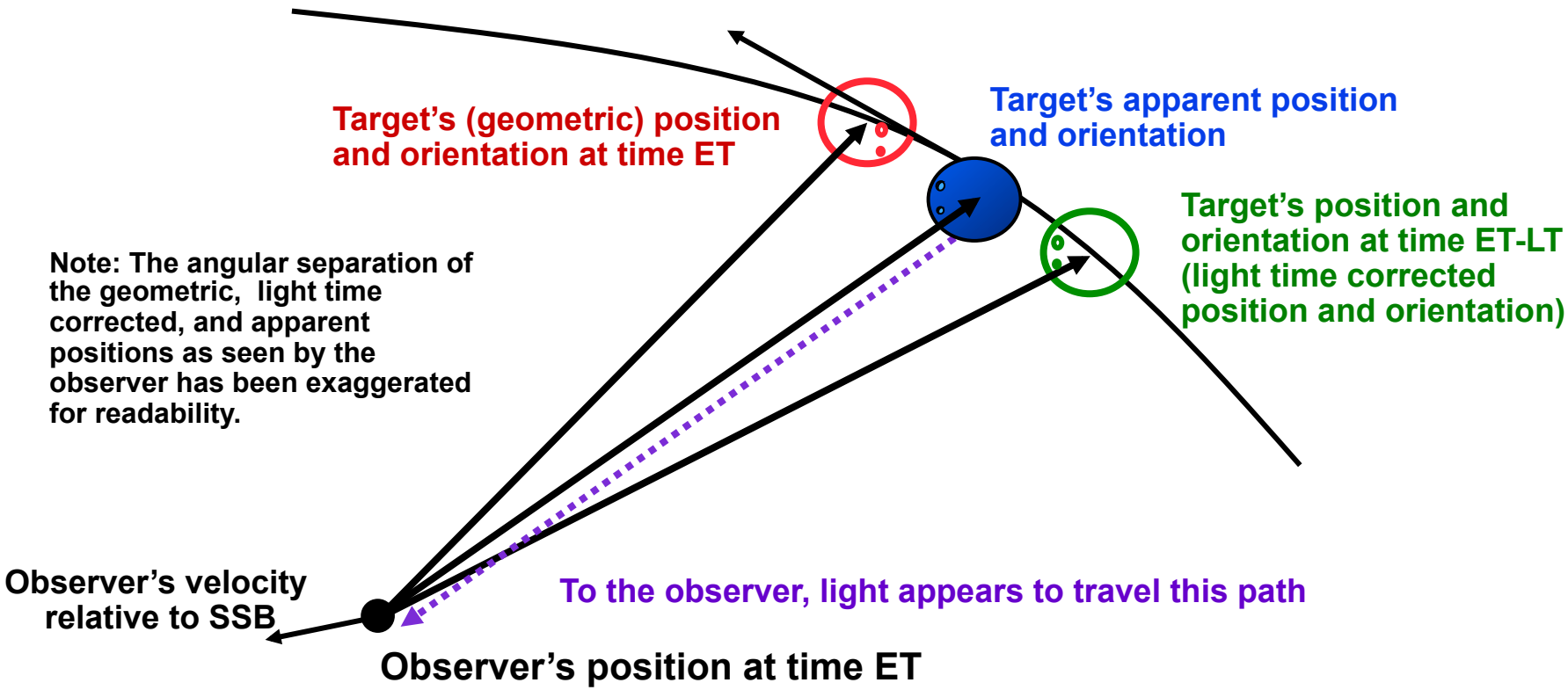
Navigation and Ancillary Information Facility



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

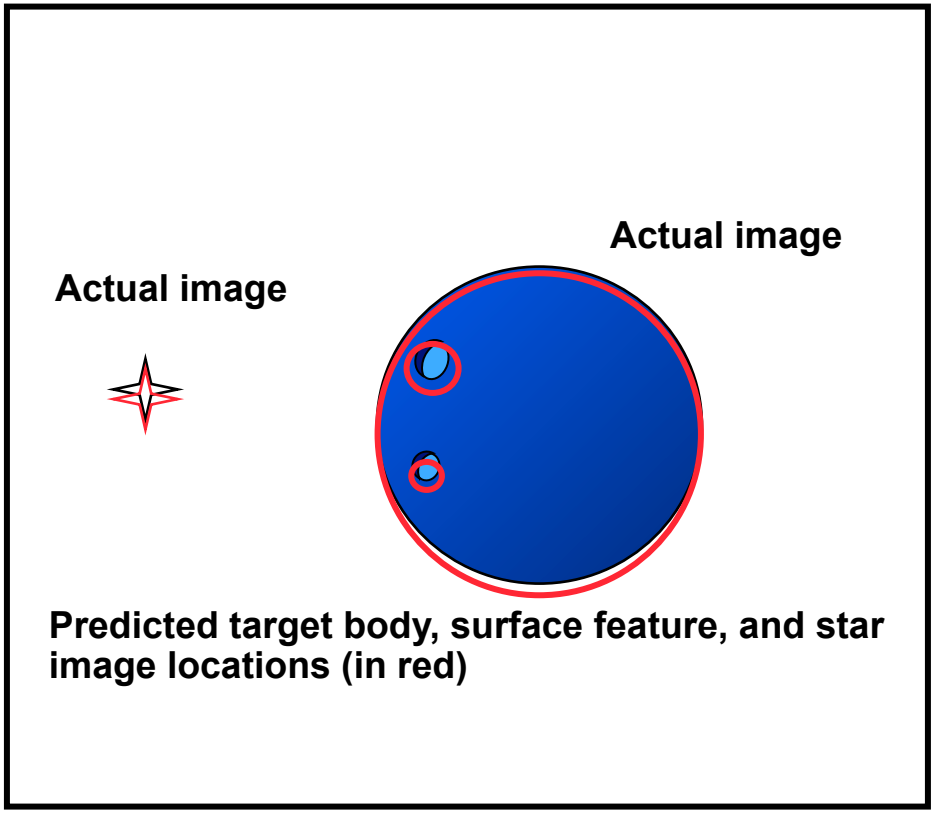


At time ET, the observer's camera records photons emitted from the target at time ET-LT, where LT is the one-way light time.

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.

Effect of Aberration Corrections - 1

- **Angular offsets between corrected and uncorrected position vectors over the time span 2004 Jan 1--2005 Jan1**
  - 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

- **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 summary of standards, lingo and common usage within SPICE

January 2012

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 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	• Used to refer 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

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 ("Furnsh")
  - Any set of text kernels 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 <u>cannot</u> 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 so-called "binary kernel run-time translation" capability added in Toolkit N0052 (1/2002).
    - » But still has a role in making native binary kernels required for some operations.

SPICE Lexicon - 4

- "Command file"
  - Many SPICE application and utility programs either require, or optionally accept, an input file containing program directives, and sometimes input data.
  - Unfortunately NAIF has not used a consistent approach for referring to such files. The following names have been used:
    - » setup file
    - » preferences file
    - » command file
    - » specifications file
    - » definitions file
- "Found flag"
  - A Boolean output from a SPICE API that informs your program whether or not a result was obtained
- 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

- **Deprecated software**
  - Code that, while still useable, has been superseded with a newer and presumably better version
  - We encourage you to not use deprecated SPICE software
    - » (But, for your convenience, we won't remove it from the Toolkit packages)
- **Toolkit version naming**
  - "Nxxxx" e.g. N0064 is Version 64
    - » Often shortened to just Nxx (e.g. N64)
  - Used for all instances of a given toolkit release
- **"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 and Furnsh Kernel
- Module, routine, subroutine, procedure, function and API
- Application, program, utility, executable
- Metadata, comments
- Time, Epoch
- Encoded SCLK, ticks*
- Frame, Reference Frame**
- Ephemeris, trajectory
- Rectangular coordinates, Cartesian coordinates**
- Geodetic, Planetodetic (coordinate system)
- Ephemeris time (ET), Barycentric Dynamical Time (TDB)
- Attitude and orientation
- International Celestial Reference Frame (ICRF) and Earth Mean Equator and Equinox of 2000 reference frame (J2000)
- "Body", "solar system object" and "ephemeris object"

* Encoded SCLK <u>always</u> 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), and to use "frame" in the sense of a set of three orthogonal vectors.

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 PDS standard: "36.3"
  - <1 to 36 alphanumeric characters>.<1 to 3 chars>
  - (Note: This is a change from the old 27.3 standard.)
- 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. pck00008.tpc)
b*	binary format	(e.g. de421.bsp)
x*	transfer format	(e.g. de421.xsp)

Common SPICE Kernel File Name Extensions

Navigation and Ancillary Information Facility

SPK:

.bsp	binary SPK file
.xsp	transfer format SPK file

PcK:

.tpc	text PcK file (This is the most common type PcK)
.bpc	binary PcK file (few instances of this)
.xpc	transfer format PcK file (few instances of this)

IK:

.ti	text IK file

FK:

.tf	text FK file

LSK:

.tls	text LSK file

CK:

.bc	binary CK file
.xc	transfer format CK file

SCLK:

.tsc	text SCLK file

MK:

.tm	text meta-kernel file ("FURNSH kernel")

DSK:

.bds	binary DSK file

EK Family (ESP, ESQ, ENB)

ESP:

.tep	text Science Plan EK file

ESQ:

.bes	binary Sequence Component EK file
.xes	transfer format Sequence Component EK file

ENB:

n/a	(www interface)

These are suggestions, not requirements

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
- Alternate formats of some of the Toolkit documents are available from the NAIF anonymous ftp server

.pdf	PDF documents

Public and Private Modules

Navigation and Ancillary Information Facility

- All Toolkits include public and private modules
- Public modules are available for your 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
  - The API Reference Guide included in the Toolkit HTML documentation provides the complete list of all public SPICE APIs available in a specific implementation of the Toolkit
- 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
  - Private APIs are not listed in the API Reference Guide but can be seen in the source code directories for SPICELIB, CSPICE, ICY and MICE
  - Do not use "private" modules in your code – they may be changed by NAIF without notice

Reference Frame and Coordinate System Conventions - 1

Navigation and Ancillary Information Facility

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

- Planetocentric reference frames
  - 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

  - For dwarf planets*, minor planets (asteroids), their satellites, and comets, these objects spin in the right hand sense about the "positive pole."
    - » In SPICE documentation, what the IAU now calls the "positive pole" is still referred to as the "north pole"
    - » 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, Eris, Haumea, Makemake, Pluto

Reference Frame and Coordinate System Conventions - 2

- **In planetographic reference frames:**
  - 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 latitude increases positively northward

Quaternions

Navigation and Ancillary Information Facility

- The SPICE system uses quaternions 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 JPL ACS/AACS and other aerospace applications
- The relationship between SPICE and MSOP quaternions:
  - Let M be a rotation matrix such that for any vector v, M*v is the result of rotating v by Θ radians in the counterclockwise direction about unit vector A. Then the quaternions representing M are:
    - » SPICE: (+/-) (cos(Θ/2), sin(Θ/2)A(1), sin(Θ/2)A(2), sin(Θ/2)A(3))
    - » MSOP: (+/-) (-sin(Θ/2)A(1), -sin(Θ/2)A(2), -sin(Θ/2)A(3), cos(Θ/2))
- Details about SPICE quaternions are found in:
  - Rotations Required Reading document
  - NAIF white paper on quaternions: ftp://naif.jpl.nasa.gov/pub/naif/misc/Quaternion_White_Paper/
  - SPICE quaternion conversion routines: M2Q, Q2M

Names and IDs

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

Navigation and Ancillary Information Facility

IDs and Names
for Physical Objects and Reference Frames

January 2012

Topics

- Summary of naming/numbering schemes used in SPICE
- Naming/numbering of <u>physical objects</u>
- Naming/numbering of <u>reference frames</u>
- Connection between the schemes

Caution: users sometimes confuse the ID assigned to an object and the ID(s) assigned to reference frame(s) associated with that object. Read on for details.

Overview - 1

- SPICE uses IDs and names to identify:
  - physical objects
  - reference frames
- An ID is an integer number
- A name is a text string

- IDs are used inside SPICE kernels
- IDs, and often any associated name(s), can be used in software interfaces (APIs) to those kernels
  - See the header for each API for details

Overview - 2

- **The schemes used for assigning IDs and names to physical 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 the physical object names and IDs**
    - » **There are some exceptions; they will be mentioned later**

Names and IDs associated with Physical Objects

Physical Object IDs and Names

Navigation and Ancillary Information Facility

- A single ID is assigned to the following types of physical objects:
  - 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
    - » 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 those same objects
- Within SPICE software there is a 1-to-MANY mapping between the ID and the 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

- **Physical object IDs are used:**
  - in kernels as data identifiers:
    - » SPKs -- to identify a body and its center of motion
    - » text PCKs -- in keywords associated with a body
    - » IKs -- in keywords associated with an instrument
    - » FKs -- to specify the center used in computing light-time correction, and to identify the body in PCK-based frames
    - » FKs -- to identify target and observer in dynamic frame specifications
    - » 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

- **Physical object IDs are used:**
  - in <u>some APIs</u> as input and/or output arguments:
    - » Older SPK routines -- SPKEZ, SPKEZP, SPKGEO, …
    - » Older derived geometry routines -- ET2LST, …
    - » Older PCK routines -- BODVAR, BODMAT, …
    - » IK routines -- GETFOV, indirectly in G*POOL, …
    - » SCLK routines -- SCE2C, SCT2E, …
    - » Coverage routines -- SPKOBJ, SPKCOV, CKOBJ, CKCOV
    - » … and more…

Object Names – Where Used?

- **Physical object names are used in the following APIs as input and/or output arguments:**
  - » **Newer SPK routines -- SPKEZR, SPKPOS**
  - » **Newer derived geometry routines -- SINCPT, ILUMIN, SUBPNT, SUBSLR, …**
  - » **Newer PCK routines -- BODVRD, …**

- **Physical object names are not used as data identifiers within kernels**

Object IDs and Names – Where Defined?

Navigation and Ancillary Information Facility

- **Name-to-ID mappings used by SPICE may be defined in two places**
  - **In Toolkit software**
    - » **Hard-coded in the source code**
    - » **Defined here for both physical objects and reference frames**
    - » **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 can be placed in any text kernel**
    - » **Mappings defined in text kernels take precedence over those defined in Toolkit source code**

Spacecraft and Ground Stations

- **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'
    - -7 'PIONEER-7, 'P7',
    - -82 'CASSINI', 'CAS'
    - -94 'MARS GLOBAL SURVEYOR', 'MGS'
    - …
  - Unfortunately sometimes NASA re-uses a number
    - » 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'
    - …

Planets

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

- **Planet-only mass centers (planet barycenter ID * 100 + 99)**
  - 199 ‘MERCURY’
  - 299 ‘VENUS’
  - 399 ‘EARTH’
  - 499 ‘MARS’
  - …
  - 999 ‘PLUTO’

Satellites

- **Satellites (planet barycenter ID*100 + number <1… 98>)**
  - 301 ‘MOON’
  - 401 ‘PHOBOS’
  - 402 ‘DEIMOS’
  - 501 ‘IO’
  - …
  - 901 ‘CHARON’, ‘1978P1’

See the BACKUP section for details about how to handle more than 98 satellites.

Comets & Asteroids

- **Periodic Comets (1000000 + sequence number*)**
  - 1000001 ‘AREND’
  - 1000002 ‘AREND-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 on SPK ID or name and find the other item here: http://ssd.jpl.nasa.gov/sbdb.cgi**

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

Instruments

- **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 physical object IDs to names, and vice versa**
  - **To get the ID for a given physical 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 physical 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 Name-to-ID Mappings

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

```
NAIF_BODY_NAME += ( 'my_spacecraft_name'     )
NAIF_BODY_ID   += (  my_spacecraft_ID        )
```

- See “Kernels Required Reading” for details

Names and IDs associated with Reference Frames

Frame IDs and Names

- **Text kernels define spacecraft frames, instrument frames, spacecraft subsystem frames, DSN station frames, etc.**
  - See comments/data sections in a text kernel for the complete listing of the frames defined in that kernel
- **IDs and Names are assigned to the following kinds of reference frames**
  - 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 Used?

Navigation and Ancillary Information Facility

- **Reference frame IDs are used**
  - in the following kernels as data identifiers:
    - » FKs -- to "glue" frame definition keywords together
    - » SPKs -- to identify base reference frames
    - » PCKs -- to identify base reference frames
    - » CKs -- to identify base reference frames
  - in the following APIs as input and/or output arguments:
    - » Almost nowhere -- users rarely or never need to deal with or be aware of reference frame IDs
- **Reference frame names are used**
  - as arguments in all high level APIs that require a reference frame to be specified as an input
    - » Derived geometry routines -- SINCPT, ILUMIN, SUBPNT, …
    - » Frame transformation routines -- PXFORM, SXFORM
    - » SPK routines -- SPKEZR, SPKPOS, …
- **Frame names are NOT used as data identifiers within kernels**

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 the Toolkit: 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 in FKs (MER, MEX, JUNO, MSL, …)**
    - » **Very rarely in other kernels, but can be in any text kernel**
      - **(For example during operations MGS frames were defined in IKs and SCLK)**
- **Unlike for physical 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**

Examples of Frame IDs and Names
Inertial and Body-fixed

- **Inertial frames (positive integers starting at 1)**
  - 1 ‘J2000’
  - …
  - 16 ‘MARSIAU’
  - 17 ‘ECLIPJ2000’
  - …

- **Body-fixed frames (positive integers starting at 10001)**
  - 10001 ‘IAU_MERCURY_BARYCENTER’
  - …
  - 10011 ‘IAU_MERCURY’
  - …
  - 10020 ‘IAU_MOON’
  - …
  - 10081 ‘EARTH_FIXED’
  - …

- **NOTE: SPICE users would rarely if ever need to know or use the frame IDs; you use only the frame name**

Examples of Frame IDs and Names
Spacecraft and Instrument

- **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 name**

Frame IDs/Names -- Mapping APIs

Navigation and Ancillary Information Facility

- SPICE provides two 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

Connections between Objects and Frames

Navigation and Ancillary Information Facility

- Although physical 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

- **CK IDs**
  - Historically IDs used in CKs are called structure IDs but in reality they are much more closely related to frames than to physical 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 the same as the spacecraft ID
  - 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

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

Name/IDs Example -- CASSINI (2)

- **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

Backup

How to handle more than 98 satellites for one planet

Object IDs/Names -- Satellites (2)

- **The scheme described for satellites can accommodate only 98 natural satellites. How do we handle more than 98?**
  - For IAU <u>provisional</u> assignments, use "psbbb" also, where:
    - » p = planet barycenter ID
    - » s = separator, set equal to "5"
    - » bbb = satellite number
      - Start with the next available unused IAU two-digit number with a "0" pre-pended, when we elect to put this scheme into effect
      - Increment by one thereafter
  - For IAU <u>permanent</u> assignments, use "psbbb" where:
    - » p = planet barycenter ID
    - » s = separator, set equal to "0"
    - » bbb = satellite number
      - Use "099" for the first new permanent assignment after "p98" from the current range is used

Object IDs/Names -- Satellites (3)

- The satellite extended numbering scheme described on the previous page attempts to, but may not be able to, maintain consistency in the last three digits as the boundary between 98 and 99 is crossed.
  - This is because the IAU sometimes changes the numbering order of new satellites when migrating from provisional to permanent status.
- It has been suggested that some JPL entity maintain a public web page showing the history of all assignments.
  - (Not done so far…)
- It is recommended that any SPK file made using provisional IDs be trashed (or hidden away) when a new file using permanent IDs is obtained.
  - If no new file is expected, the *bspidmod* program can be used to replace provisional ID(s) with official ID(s) in the old file

Navigation and Ancillary Information Facility

Introduction to Kernels

January 2012

“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 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.

SPICE Kernels Family

Navigation and Ancillary Information Facility

- **SPK**
  - Spacecraft and Planet Ephemeris
- **PcK**
  - Planetary Constants, for natural bodies
    - » Orientation
    - » Size and shape
- **IK**
  - Instrument
- **CK**
  - Pointing ("C-matrix")
- **EK**
  - Events, up to three distinct components
    - » ESP: science plan
    - » ESQ: sequence
    - » ENB: experimenter's notebook
- **FK**
  - Reference frame specifications
- **SCLK**
  - Spacecraft clock correlation data
- **LSK**
  - Leapseconds
- **Meta-Kernel** (a.k.a. "FURNSH kernel")
  - Mechanism for aggregating and easily loading a collection of kernel files
- **DSK (under development)**
  - Digital shape kernel
    - » Tesselated plate model
    - » Digital elevation model

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 ("Furnsh" meta-kernel)

SPICE binary kernels are:

- SPK
- binary PCK (exists only for Earth and moon)
- CK
- ESQ (part of the E-kernel)
- DBK (database kernel)
- DSK (digital shape kernel)*

* New kind of kernel under development

SPICE Kernel Forms

Navigation and Ancillary Information Facility

- **Binary form: SPK, binary PCK, CK, EK/ESQ[1], DSK**
  - Binary kernels are not human-readable and require the use of Toolkit software to examine the data contents.
- **Text form: text PcK, IK, FK, LSK, SCLK, FURNSH (MK)**
  - Text kernels contain only printing characters (ASCII values 32-126), i.e. human-readable text.
- **"Transfer" form of a binary kernel**
  - This is an ASCII representation of a binary kernel
  - Was used for porting the file between computers with incompatible binary representations (e.g. PC and UNIX)
  - Use of the transfer format is no longer needed for porting due to the run-time translation capability added to SPICE long ago
    - » But it is one way to convert a non-native binary kernel into native format, needed for modifying the kernel or improving read efficiency

[1] The ESP and ENB components of the EK might be binary, or text, or html, depending on specific implementation.

Text Kernel Data

- **A text kernel contains data provided in simple assignments of the form:**

```
variable name = value(s)
```

or

```
variable name += value(s)
```

- **A text kernel can, and should, also contain descriptive meta-data that characterize the data**

Example Text Kernel

Navigation and Ancillary Information Facility

This is a sample SPICE text kernel. The \begindata and \begintext markers on lines by themselves set off the start of data and text blocks respectively.

Note the forward slash!

Note the backward slashes!

```
KPL/<kernel type>
\begindata
  NAME            = 'Sample text value'
  NaMe            = 'Keywords are case sensitive'

  NUMBERS         = ( 10.123, +151.241, -1D14 )
  NUMBERS        += ( 1.0,    1,        -10   )
  NUMBERS        += ( 1.542E-12, 1.123125412  )

  NAIF_BODY_NAME += ( 'SPEEDO', 'NEETO' )
  NAIF_BODY_CODE += ( -678,     -679    )

  TIME            = @1972-JAN-1

\begintext
   < some comments about the data >
\begindata
   < more data, given again in keyword = value syntax >
\begintext
      < etc., etc. >
```

- **Text kernels can contain characters, times, and numeric values.**
- **Text kernels can contain scalar or vector values.**
- **For more information see "Kernel Required Reading."**

Text Kernel Operators

- An assignment using the "=" operator creates a variable name with one or more associated values.

- An assignment using the "+=" operator adds additional values to an existing variable. It creates a new variable if the referenced variable doesn't already exist.

- The "@" symbol preceding a calendar date identifies a date string.
  - The string must not contain embedded blanks
  - The string will be parsed and converted to an internal double precision representation of that epoch.
    - » This conversion does not need a leapseconds kernel
  - The date is interpreted as ephemeris time (ET, also called TDB).

Example Binary Kernel

A binary kernel contains lots of non-printing (unintelligible) data, usually interspersed with occasional occurrences of ASCII characters.

The only way to use a binary kernel is to read it, or add to it, using a SPICE routine or program.

Comments In SPICE Kernels

- **All SPICE kernels can and 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.**

Loading Kernels - 1

- To make kernels available to SPICE programs you "load" them.

- When you load a <u>text</u> 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 <u>binary</u> kernel:
  - the file is opened
  - for SPK, CK, and binary PCK 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.**
  - **Sample FORTRAN, C, IDL and MATLAB calls:**
    - » `CALL FURNSH ( 'name.ext' )`
    - » `furnsh_c ( "name.ext" );`
    - » `cspice_furnsh, 'name.ext'`
    - » `cspice_furnsh ( 'name.ext' )`

- **Best practice: don't hard code filenames–list these in a "meta-kernel" and load the meta-kernel using FURNSH.**
  - `CALL FURNSH ('meta-kernel_name')` **(Fortran example)**
  - **See the next page for more information on meta-kernels**

- **Caution: "Transfer format" versions of binary kernels can <u>not</u> be loaded; they must first be converted to binary with the Toolkit utility program *tobin* or *spacit*.**

What is a "Meta-Kernel"

- 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
  - You can simply load the meta-kernel, causing 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
- A meta-kernel is implemented using the generic SPICE text kernel standards
  - Refer to the Kernels Required Reading technical reference for details
- The terms "meta-kernel" and "FURNSH kernel" are used synonymously

Sample Meta-Kernel Contents (1)

Navigation and Ancillary Information Facility

- This is a sample meta-kernel that the Toolkit routine FURNSH could use to load a collection of kernels.

```
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',
                       )
```

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 sizes by the text kernel standards.

Sample Meta-Kernel Contents (2)

- This sample 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'
                     )
```

- Although the OS environment variable notation $NAME is used to refer to the symbols set by 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 also be used to handle path value strings that exceed 80 characters.

Limits on Loaded Kernels

- **The number of binary kernels that may be loaded at any time is large, but limited.**
  - For SPK, CK, and binary PCK files:
    - » Loaded SPKs + Loaded CKs + Loaded binary PCKs <= 1000
  - For ESQ files:
    - » Loaded ESQs <= 20
  - For all kernels:
    - » Loaded kernels <= 1300
      - Assumes each has been loaded only once, and not unloaded.
- **There are also limits on the number of keywords and values for all loaded text kernels:**
  - Maximum number of keywords is 5003.
  - Maximum number of numeric data items is 200,000.
  - Maximum number of character data items is 4000.

Kernel Precedence Rule

- **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).
    - » 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 have been replaced with the last loaded value(s).
  - Orientation data from a binary PCK <u>always</u> supersedes orientation data (for the same object) obtained from a text PCK, no matter the order in which the kernels are loaded.

Unloading Kernels

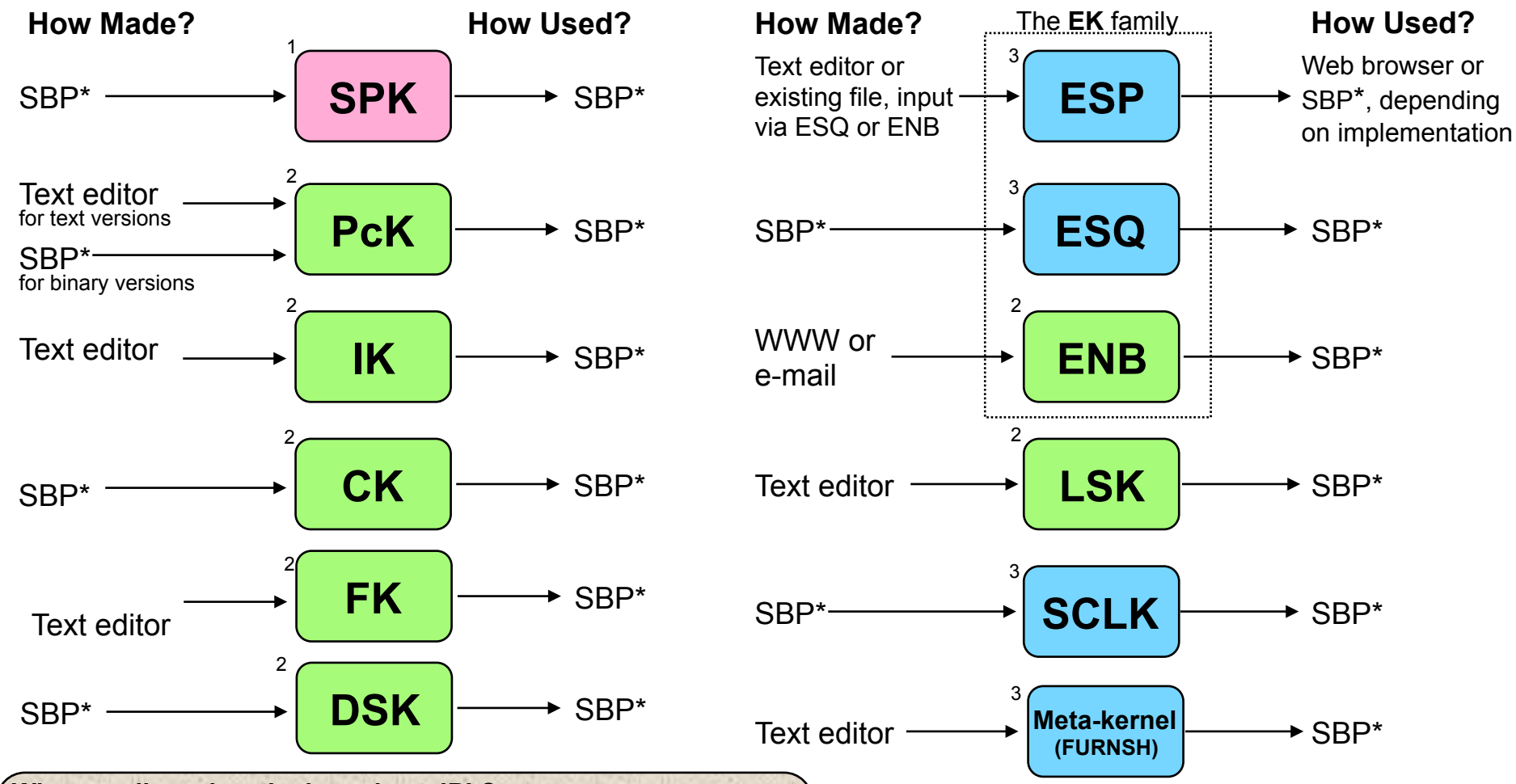
- The unloading of a kernel is infrequently needed for FORTRAN or CSPICE applications but is essential for Icy and Mice scripts
  - Because of the way IDL and MATLAB interact with external shared object libraries any kernels loaded during an IDL or MATLAB session will stay loaded until the end of the 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 FURNSH. They will not unload any files that have been loaded with older load routines such as SPKLEF (those used prior to availability of FURNSH).
- Caution: unloading text kernels with UNLOAD will also remove any kernel pool data provided through the kernel pool API (P*POOL)

Backup

- 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



Who usually makes the kernels at JPL?

1 NAV and NAIF

2 NAIF

3 NAIF 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 NAIF 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 metadata (comments) -To merge files or subset a file -To correct/revise an object ID	- COMMNT, SPACIT or SPICELIB module - SPKMERGE - 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 metadata (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 metadata (comments) -To merge files or subset a file	- COMMNT, SPACIT or SPICELIB module - DSKMERGE

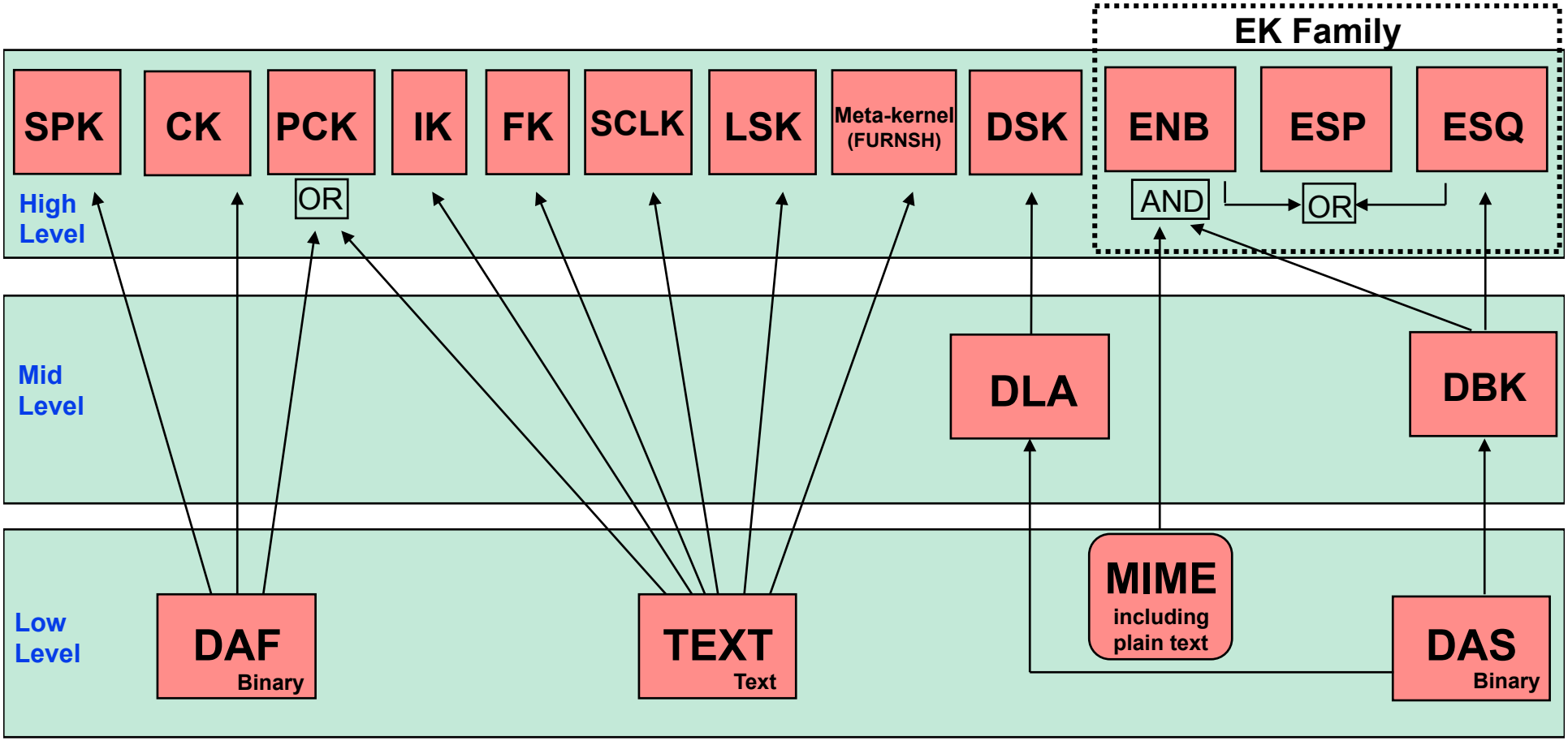
Why & How Kernels are “Modified” - 2

Navigation and Ancillary Information Facility

File Type	Why Modified	How Modified
The **EK** family		
ESP	-To add, revise or delete “data” -To add metadata (comments)	- (Depends on implementation) - (Depends on implementation)
ESQ	-To add additional data -To revise data -To delete data -To add metadata (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 metadata	- Text editor
Meta-kernel (FURNSH)	- To revise contents in any way	- 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

- **Cutting/pasting complete, or pieces of, data assignments or `\begindata` or `\begintext` 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”

Some Useful Tools - 1

- For a Unix or Linux (including Mac) environment
  - To display all non-printing characters, 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)
  - 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

- **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>
  - For converting either style text kernel to the other style, use the SPICE "bingo" executable
    - » 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

Porting Kernels

January 2012

Porting Issues - 1

- 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 used on the computer you are using.

Porting Issues - 2

- Toolkit software can often read kernels obtained from an incompatible platform
  - Binary SPK, CK, or PCK kernels from one system can be read on an incompatible system (e.g. any pair of PC, Mac, Sun).
  - Text kernels from one system can be read on an incompatible system (e.g. any pair of PC, Mac, Sun) when using a C, IDL or MATLAB toolkit.

- The Toolkit cannot read certain kernels from incompatible platforms
  - Text kernels, if using a FORTAN toolkit
  - DAS-based files, such as E-kernels (ESQ) or shape model kernels (DSK)

Porting Issues - 3

- When conversion to native format is required to make the kernel usable, 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 dos2unix and unix2dos, or bingo, 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…

	Groupings of Text Compatible Environments	End of line indicator
1	PC using Windows or N T	<CR><LF>
2	Unix PC with LINUX Macintosh OSX (Motorola or Intel chip)	<LF>

Compatible Environments for Binary Kernels

Navigation and Ancillary Information Facility

	Groupings of Binary Compatible Environments	Binary Representation
1	PC/ Windows PC/Linux Mac Pro (Intel chip) (the new ones)	IEEE - Little endian
2	Sun Mac Power PC (Motorola chip)	IEEE - Big endian

Caution Using Email

- **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

- **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.**
  - **You cannot write information into the comment area, or delete information from the comment area.**
  - **You cannot append additional data to the kernel.**
- **Run-time conversion does not work for E-kernel (ESQ) or shape model (DSK) kernels.**
  - **More generally, it does not yet work for any file built upon the SPICE "DAS" architecture.**

Binary Kernels
Allowed Operations

- You may “load” and read both non-native and native binary kernels in the same runtime instance
- You may merge any combination of native and non-native SPK files
  - The resultant, merged SPK file will be in native format

Navigation and Ancillary Information Facility

“Comments” In SPICE Kernels

Also known as “meta-data”

January 2012

What are Comments?

- Comments 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 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
  - 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
  - Companion files
- In SPICE, we sometimes refer to “comments” as “meta-data”

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
  - Comments may be inserted between blocks of data using `\begintext` and `\begindata` as start and end markers:

```
\begintext
  Some comments
\begindata
  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)
    - » This capability is not yet available in Mice
- **Text Kernels**
  - Use a text editor
    - » Begin comment sections with the "`\begintext`" marker alone on a line
      - (The marker is not needed for comments placed at the beginning of a text kernel)
    - » End comment sections with a "`\begindata`" marker 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

- **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

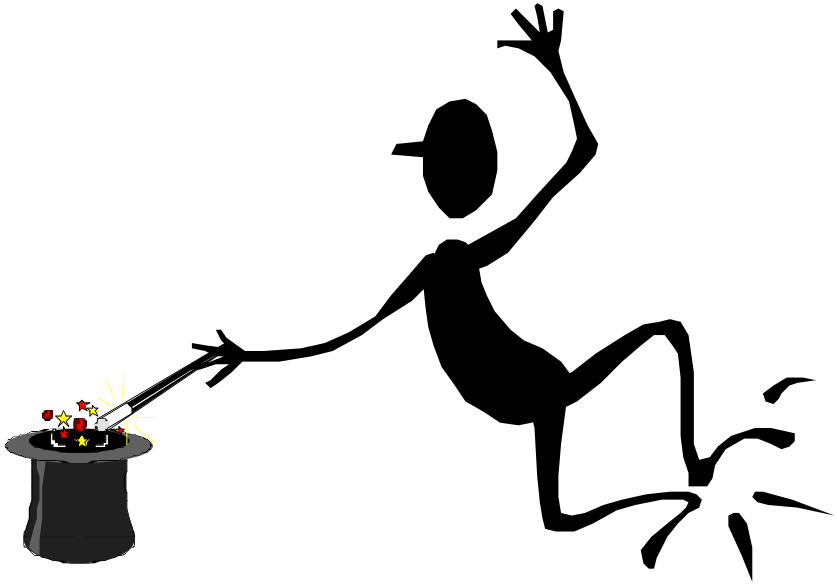
This example show use of the unix “more” processor to examine an entire text kernel, both comments and data.

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

Getting and Installing the SPICE Toolkit



January 2012

Getting Toolkit

Navigation and Ancillary Information Facility

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

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

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

Don’t Port it Yourself

- **The packages provided on the NAIF server have been built and tested by NAIF on these particular environments.**

- **We highly recommend you NOT try to port any instance of the Toolkit to some other environment, especially without consulting with NAIF first.**
  - **There are portability issues and compiler optimization issues that must be carefully dealt with.**

Installing Toolkit

Navigation and Ancillary Information Facility

Terminal Window

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

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

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

```
> toolkit
```

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

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

- **Try some executables**
  - Use *tobin* to convert the SPICE transfer format SPK and CK files supplied with the Toolkit to local binary.
    - » cook_01.tsp, cook_02.tsp, cook_01.tc, and cook_02.tc are found in the ../data directory
  - Use *brief*, *ckbrief* or *spacit* to summarize the converted kernels.
- **Problems may occur if operating systems or compiler versions are out of sync**
  - Rebuild the Toolkit using the script “makeall.csh” (or “makeall.bat”) located in the “top level” directory (toolkit or cspice or icy or mice).
- **In the rare circumstance that things still don’t work, contact your System Administrator or NAIF.**

Backup

- **Getting the Toolkit using command line FTP**

Command line FTP - 1

Navigation and Ancillary Information Facility

Terminal Window

```
prompt> ftp naif.jpl.nasa.gov

Connected to naif.jpl.nasa.gov
220  Welcome to the NAIF FTP service.

Name (your.sight:your_name): anonymous
331 Please specify the password
Password: your@e.mail.address
230-
230-  ======================================================================
230-  |                    Jet Propulsion Laboratory                       |
230-  |                 * * *    W A R N I N G   * * *                     |
230-  |                        Property of the                             |
230-  |                     UNITED STATES GOVERNMENT                       |
230-  ======================================================================
230-
230 Login successful. Have fun.
Remote system type is UNIX.
Using binary mode to transfer files.

ftp> cd pub/naif/toolkit/<FORTRAN or C or IDL or MATLAB>
250 CWD command successful.
ftp> dir
```

Command line FTP - 2

Navigation and Ancillary Information Facility

Terminal Window

```
ftp> dir
Mac_OSX_Absoft
Mac_OSX_IFORT
Mac_OSX_g77
PC_Cygwin
PC_Linux
PC_Windows_Digital
PC_Windows_IFORT
PC_Windows_Lahey
Sun_Solaris
```

FORTRAN

```
ftp> dir
Mac_OSX_Apple_C
Mac_OSX_Intel_C
PC_Cygwin_C
PC_Linux_C
PC_Linux_C_64bit
PC_Windows_Visual_C
Sun_Solaris_C
Sun_Solaris_GCC
Sun_Solaris_GCC_64bit
```

C

```
ftp> dir
Mac_OSX_Apple_C
Mac_OSX_Intel_C
PC_Linux_C
PC_Windows_Visual_C
Sun_Solaris_C
Sun_Solaris_GCC
```

IDL

```
ftp> dir
Mac_OSX_Apple_C
Mac_OSX_Intel_C
PC_Linux_C
PC_Windows_Visual_C
```

MATLAB

The environments available at the time you download the Toolkit may differ from those shown here.

Command line FTP - 3

Navigation and Ancillary Information Facility

Terminal Window

```
ftp> cd <environment>/packages
ftp> binary
200 Type set to I
ftp> get toolkit.tar.Z
         ( or toolkit.exe
           or cspice.tar.Z or cspice.exe
           or icy.tar.Z or icy.exe
           or mice.tar.Z or mice.exe )
. . .
ftp> ascii
200 Type set to A
ftp> get importSpice.csh
         ( or importCSpice.csh
           or importIcy.csh
           or importMice.csh )
         ( not availabe for Windows environment )
ftp> get README
ftp> get dscriptn.txt
ftp> get whats.new
ftp> quit
```

Navigation and Ancillary Information Facility

Preparing for Programming Using the SPICE Toolkits

January 2012

Setting Path to Toolkit Executables

Navigation and Ancillary Information Facility

Recommended for all languages

- **Unix**
  - **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)`
  - **bash**
    - » `PATH=$PATH:/`*`my_directory`*`/toolkit/exe`
    - » `PATH=$PATH:/`*`my_directory`*`/cspice/exe`
    - » `PATH=$PATH:/`*`my_directory`*`/icy/exe`
    - » `PATH=$PATH:/`*`my_directory`*`/mice/exe`
- **Windows**
  - **Add location of toolkit executables to the environment variable PATH from the *Advanced* pane on the *System* Control Panel (*Control Panel->System->Advanced*).**
    - » *`drive:\my_directory`*`\toolkit\exe`
    - » *`drive:\my_directory`*`\cspice\exe`
    - » *`drive:\my_directory`*`\icy\exe`
    - » *`drive:\my_directory`*`\mice\exe`

Replace the *italics* with the path in which you installed the toolkit on your computer.

Unix/Linux: Builds

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, say *program*, against the CSPICE or SPICELIB library.**

  » **For C:**

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

  » **For FORTRAN:**

```
$ g77 program.f /naif/toolkit/spicelib.a
```

  » **Note: some FORTRAN compilers (e.g. Absoft) require an additional flag "`-lU77`" to pull in the standard Unix symbols when linking against SPICELIB.**

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

  `gcc … -L/`*`path_to_libs`*`/ -l`*`name`*

  unless you rename the SPICE library.

Windows: Compiler settings

- **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 for a 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`
  - **If available on your system, you can execute the "Visual Studio *version* Command Prompt" utility from the**

    Programs -> Microsoft Visual Studio version -> Visual Studio Tools

    menu. The utility spawns a DOS shell set with the appropriate environment variables.

Windows: Builds

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:**

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

Icy: Register the Icy DLM to IDL (1)

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.**
    - » **Unix, X86 architecture**

```
cp icy.dlm icy.so /usr/local/itt/idl64/bin/bin.linux.x86/
```

    - » **Windows, X86 architecture**

```
cp icy.dlm icy.dll C:\ITT\IDL64\bin\bin.x86\
```

continued on next page

Icy: Register the Icy DLM to IDL (2)

- **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_directory_containing_icy.dlm_"
```

Caveat: 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 to IDL_DLM_PATH the directory. This directory contains an "icy.dlm" but not "icy.so."

continued on next page

Icy: Register the Icy DLM to IDL (3)

- **Windows specific:**
  - **Set environment variable IDL_DLM_PATH from the *Advanced* pane of the *System* Control Panel.**
- **Once registered (by whatever means) 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

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 lessons' code.
- Optional: Place your `dlm_register` command in a start up script. Specify the script using the IDL IDE's preferences panel.

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

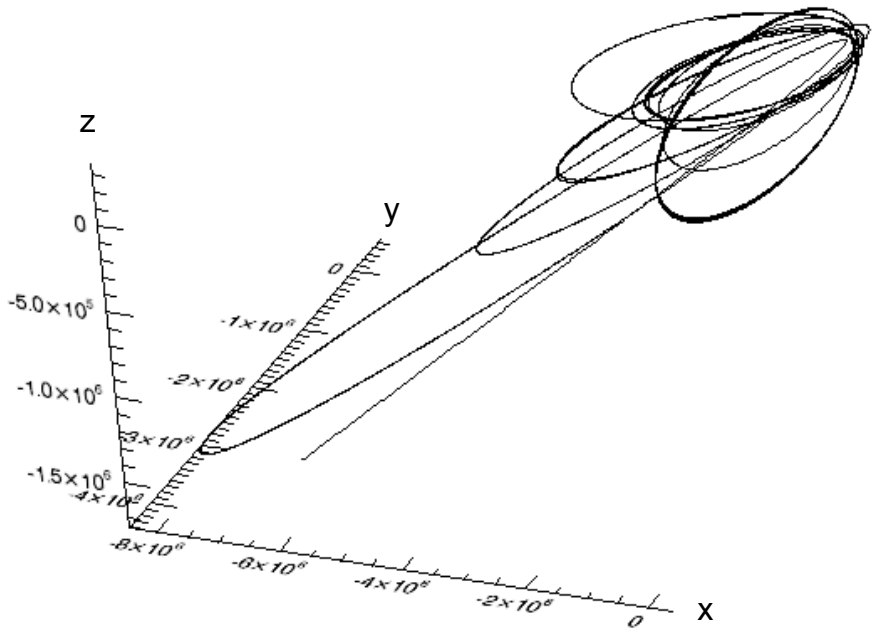
Simple Icy Example

- **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

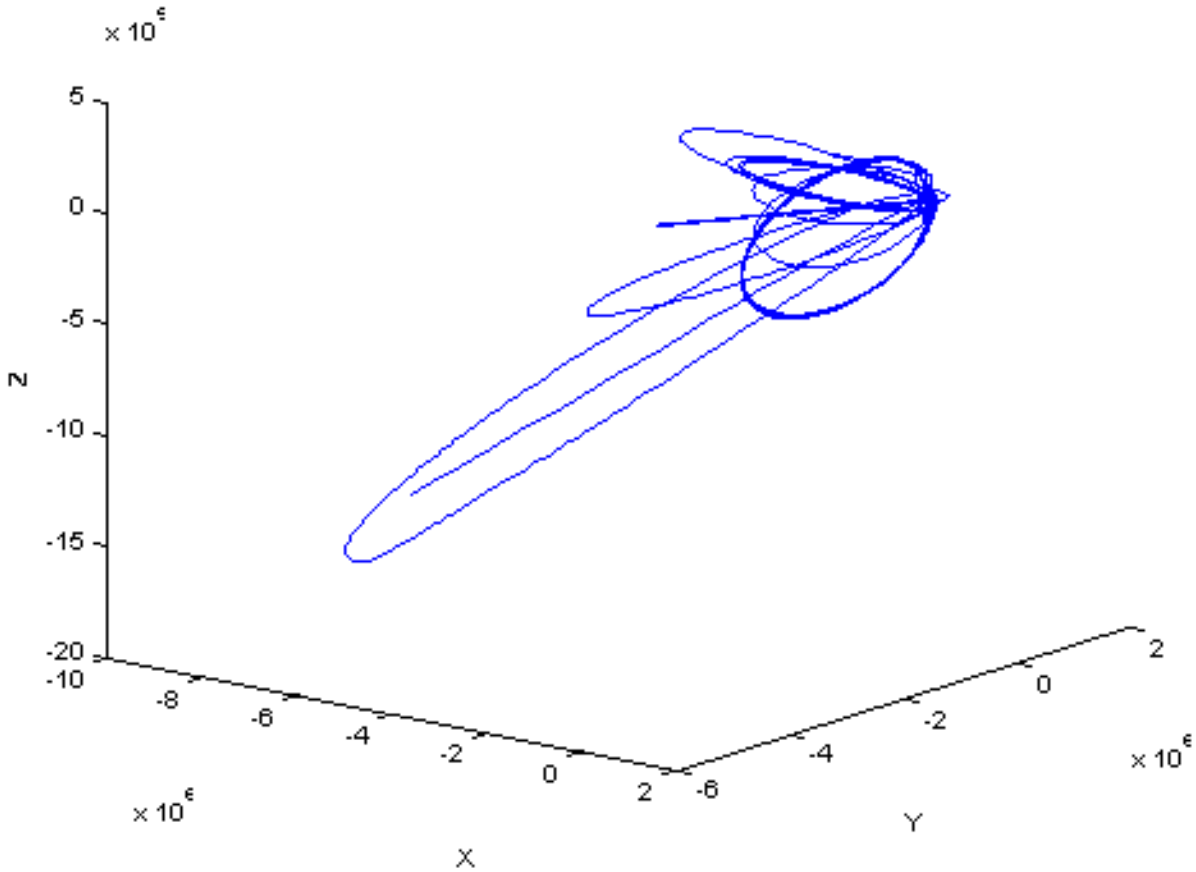
Simple Mice Example

- **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

- "icy.req," Icy Required Reading
- "mice.req," Mice Required Reading
- "Introduction to the Family of SPICE Toolkits," tutorial

Navigation and Ancillary Information Facility

Introduction to the Family of SPICE Toolkits

January 2012

Topics

Navigation and Ancillary Information Facility

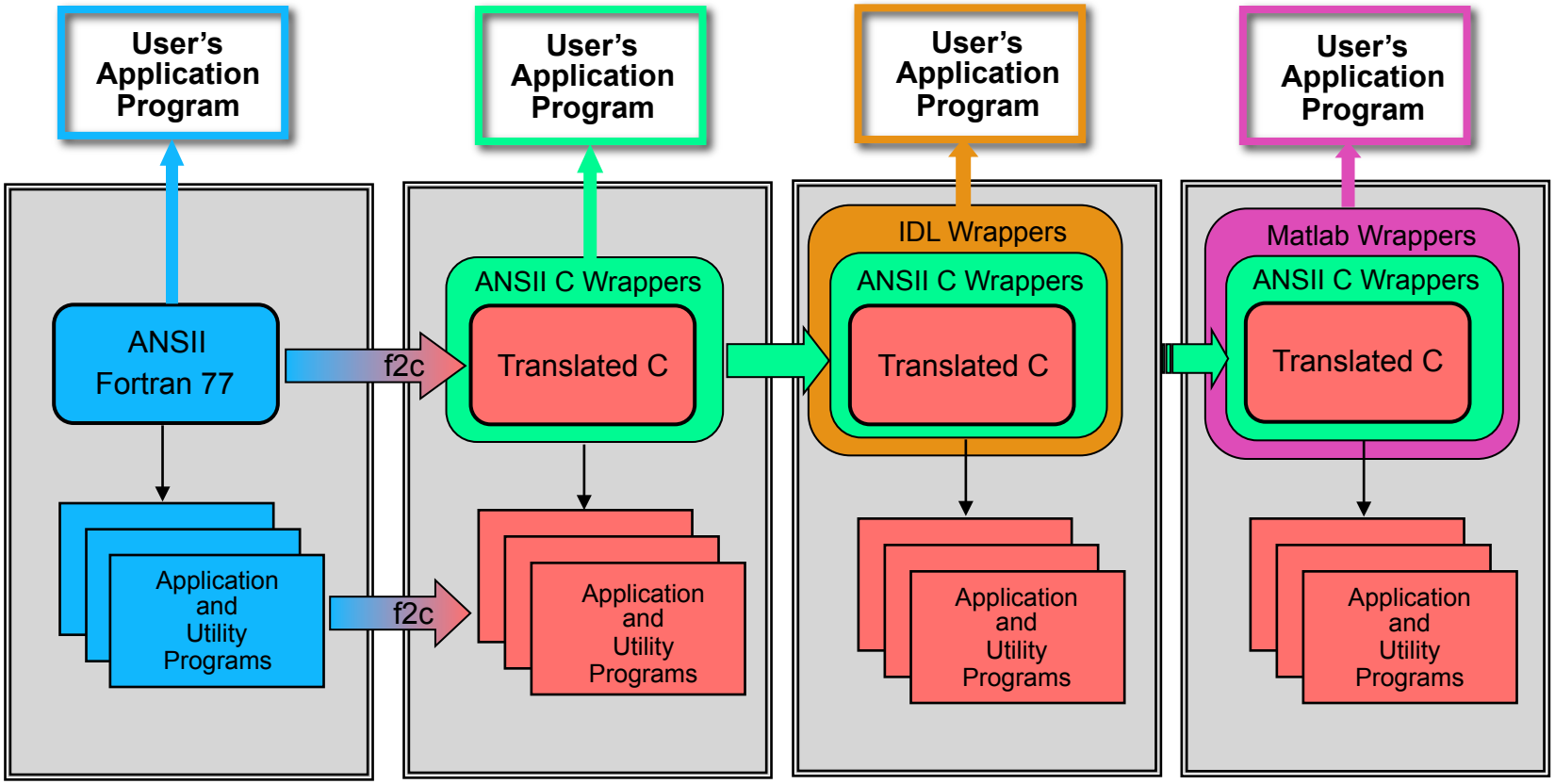
- Toolkit architecture
- Toolkit contents
- Toolkit characteristics
- Toolkit versions
- Toolkit capabilities
- Introduction to the four Toolkits
  - "Toolkit" (FORTRAN)
  - "CSPICE" (C)
  - "Icy" (IDL)
  - "Mice" (Matlab)
- Toolkit Directory Structure
- Toolkit Utility Programs
- Toolkit Application Programs
- Toolkit Documentation
- Backup: Currently Supported Environments

Toolkit Architecture

- The SPICE Toolkit is available in Fortran, C, IDL (Interactive Data Language), and Matlab.
- The Fortran, C, IDL, and Matlab Toolkits are delivered as standalone products.
  - The IDL and Matlab Toolkits, by necessity, also include the complete C Toolkit.

Toolkit Architecture Pictorial

Navigation and Ancillary Information Facility



Language →	FORTRAN	C	IDL	Matlab
Product name →	"Toolkit"	"CSPICE"	"Icy"	"Mice"

Toolkit Contents

Navigation and Ancillary Information Facility

- **Software**
  - Subroutine libraries, with source code
    - » SPICELIB (Fortran)
    - » CSPICE (C)
    - » Icy (C)
    - » Mice (C and Matlab script)
  - Executable programs
    - » application and utility programs
    - » cookbook examples
  - Installation/build scripts
- **Documentation**
  - Available in ASCII and HTML
- **Data**
  - Sample kernel files (supplied only for use with cookbook programs, not valid for general use).
- **Toolkits delivered to missions or other special customers may be augmented with mission- or customer-specific products.**

Toolkit Characteristics

- 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.
    - » 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

- **Toolkit Version**
  - SPICE Toolkits have an associated Version number
    - » Example: "N0064" (also written as "N64")
  - The version number applies to the Fortran, C, IDL and Matlab implementations for all supported platforms.
- **When does NAIF release a new SPICE toolkit version?**
    - » Not according to a fixed schedule
    - » Primarily driven by addition of significant new capabilities
      - For example, Icy or Mice or the geometry finder subsystem
    - » 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

- Toolkit libraries contain a broad set of capabilities related to the computations needed for “observation geometry” and time conversions.

- 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) Toolkit duplicates most functionality available in the C Toolkit wrapper routines.
  - The Matlab (Mice) Toolkit provides interfaces to those routines NAIF considers the most often needed by users.

Toolkit Capabilities

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)
  - Query binary EK files (EK-ESQ)
- **Kernel write access**
  - SPK writers
  - CK writers
  - EK writers (sequence component, ESQ)
  - PCK writers (only for binary PCK files)

Toolkit Capabilities

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 transformation**
  - Obtain 3x3 matrices for frame transformations of positions
  - Obtain 6x6 matrices for frame transformations of states
- **Time conversion**
  - Conversion between standard systems: TDB, TT (TDT), UTC
  - Conversion between SCLK and other systems
  - Parsing and formatting
- **Geometry finder**
  - 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 Capabilities

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 Capabilities

- **Arrays**
  - Sorting, finding order vector, reordering
  - Searching: linear, binary
  - Insertion and deletion
- **Name/code conversion**
  - Bodies
  - Frames
- **I/O support**
  - Logical unit management (Fortran toolkits)
  - Open, read, write text files
  - Kernel pool API
- **Exception handling**
  - Control exception handling behavior: mode, set message, assign output device.

Toolkit Capabilities

- **Advanced data types**
  - Cells, Sets
  - Windows (sometimes called schedules)
  - Symbol Tables
  - Planes, Ellipses

Fortran Toolkit

- "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

- "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

- 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.
    - Includes 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, &ltime);
      - 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

- "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, &ltime);
      - 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 a HTML page.
    - » Mice Required Reading
      - Provides background information essential for programming with Mice
- See the "Matlab_Interface" tutorial for details

Directory Structure

- The top level directory name for each Toolkit is:
  - "toolkit" for Fortran Toolkits.
  - "cspice" for C Toolkits.
  - "icy" for IDL Toolkits.
  - "mice" for Matlab Toolkits.
- Directory structures for the Toolkits are almost identical. However…
  - The CSPICE, Icy and Mice Toolkits also have a directory for include files.
  - The names for application source code directories in CSPICE, Icy and Mice differ slightly from those in the Fortran toolkit.
  - Icy and Mice include additional directories for :
    - » Icy/Mice source code
    - » Icy/Mice cookbook programs

Directory Structure

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

Directory Structure

Navigation and Ancillary Information Facility

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

Toolkit Application Programs

Navigation and Ancillary Information Facility

- SPICE Toolkit application programs perform various tasks. Some examples are:
  - create a new SPK file from a text file of state vectors or elements
    - » mkspk
  - compare (diff) two SPKs
    - » spkdiff
  - compare (diff) two reference frames
    - » frmdiff
  - create a new CK from a text file of attitude data
    - » msopck
  - carry out a wide assortment of time conversions
    - » chronos
  - query Event Kernels (EKs)
    - » inspekt

Toolkit Utility Programs

- **SPICE Toolkit utility programs are available to:**
  - port binary SPICE kernels between incompatible systems*
    - » tobin, toxfr, spacit
    - » bingo (available only from the NAIF webpage)
  - port text SPICE kernels between incompatible systems
    - » bingo (available only from the NAIF webpage)
  - 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, spacit
  - merge or subset SPK files
    - » spkmerge
  - indicate current Toolkit version
    - » version

* Usually not needed

Toolkit Documentation

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
    - » Act as primary functional specification: I/O, exceptions, particulars defining behavior of module
    - » Contain code examples
    - » A standard format is used for each routine or entry point
    - » 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

continues on next page

Toolkit Documentation

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 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 of Required Readings were adapted for all languages
    - Some of the Required Reading documents provided with CSPICE are still based upon Fortran SPICE
    - Some of the Required Readings for Icy or Mice toolkits are still based upon CSPICE
- User's Guides
  - » Interface specifications for the Toolkit utility programs and applications.
  - » 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

Navigation and Ancillary Information Facility

- **Other documents**
  - Permuted Index
    - » Maps phrases describing functionality to corresponding module names and file names
    - » Shows names of all entry points in Fortran toolkit APIs
    - » HTML version is accessible using the "Permuted 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

- **Other documents (continued)**
  - 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.
    - » 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

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.**
    - » **Please don't try to use or port a Toolkit built for another environment.**

Supported Environments - Fortran

Navigation and Ancillary Information Facility

Product Name	Operating System	Compiler
Mac/Intel, OS-X, Intel FORTRAN, 32bit	OS X 10.6.8	Intel Fortran 10.1
Mac/Intel, OS-X, Intel FORTRAN, 64bit	OS X 10.6.8	Intel Fortran 10.1
Mac/Intel, OS-X, gfortran, 32bit	OS X 10.6.8	gfortran, GNU Fortran 4.3
Mac/Intel, OS-X, gfortran, 64bit	OS X 10.6.8	gfortran, GNU Fortran 4.3
PC, CYGWIN, g77, 32bit	Windows/Cygwin	g77, GNU Fortran 3.2
PC, Linux, Intel FORTRAN, 32bit	Red Hat Linux (RHE4)	Intel Fortran 10.0
PC, Linux, g77, 32bit	Red Hat Linux (RHE4)	g77, GNU Fortran 3.4
PC, Linux, gfortran, 32bit	Red Hat Linux (RHE4)	gfortran, GNU Fortran 4.3
PC, Linux, gfortran, 64bit	Red Hat Linux (RHE4)	gfortran, GNU Fortran 4.3
PC, Windows, Intel FORTRAN, 32bit	Windows XP	Intel Fortran 9.1
PC, Windows, Intel FORTRAN, 64bit	Windows XP	Intel Fortran 9.1
Sun/Intel, Solaris, SUN FORTRAN, 32bit	Solaris 9	Sun FORTRAN 95 8.2
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 10.4.x	gcc, GNU C 4.2.1
Mac/Intel, OS-X, Apple C, 64bit	OS X 10.4.x	gcc, GNU C 4.2.1
PC, CYGWIN, gCC, 32bit	Windows/Cygwin	gcc, GNU C 3.2
PC, Linux, gCC, 32bit	Red Hat Linux (RHE4)	gcc, GNU C 3.4.6
PC, Linux, gCC, 64bit	Red Hat Linux (RHE4)	gcc, GNU C 3.4.6
PC, Windows, Microsoft Visual C, 32bit	Windows NT/2K/XP	Microsoft Visual Studio 2008 C
PC, Windows, Microsoft Visual C, 64bit	Windows NT/2K/XP	Microsoft Visual Studio 2008 C
Sun/Intel, Solaris, gCC, 32bit	Solaris 9	gcc, GNU C 3.3.2
Sun/Intel, Solaris, gCC, 64bit	Solaris 9	gcc, GNU C 3.3.2
Sun/SPARC, Solaris, gCC, 32bit	Solaris 9	gcc, GNU C 3.3.2
Sun/SPARC, Solaris, gCC, 64bit	Solaris 9	gcc, GNU C 3.3.2
Sun/SPARC, Solaris, SUN C, 32bit	Solaris 9	Sun C 5.8
Sun/SPARC, Solaris, SUN C, 64bit	Solaris 9	Sun C 5.8

Supported Environments - IDL*

Navigation and Ancillary Information Facility

Product Name	Operating System	Compiler
Mac/Intel, OS-X, Apple C/IDL, 32bit	OS X 10.6.8	gcc, GNU C 4.2.1
Mac/Intel, OS-X, Apple C/IDL, 64bit	OS X 10.6.8	gcc, GNU C 4.2.1
PC, Linux, gcc/IDL, 32bit	Red Hat Linux (RHE4)	gcc, GNU C 3.4.6
PC, Linux, gcc/IDL, 64bit	Red Hat Linux (RHE4)	gcc, GNU C 3.4.6
PC, Windows, Microsoft Visual C/IDL, 32bit	Windows XP	Microsoft Visual Studio 2008 C
PC, Windows, Microsoft Visual C/IDL, 64bit	Windows XP	Microsoft Visual Studio 2008 C
Sun, Solaris, gcc/IDL, 32bit	Solaris 9	gcc, GNU C 3.3.2
Sun, Solaris, gcc/IDL, 64bit	Solaris 9	gcc, GNU C 3.3.2
Sun, Solaris, SUN C/IDL, 32bit	Solaris 9	Sun C 5.8

***NAIF built and tested Icy using IDL version 8.1.**

Supported Environments - Matlab*

Product Name	Operating System	Compiler
Mac/Intel, OS-X, Apple C, 32bit	OS X 10.6.8	gcc, GNU C 4.2.1
Mac/Intel, OS-X, Apple C, 64bit	OS X 10.6.8	gcc, GNU C 4.2.1
PC, Linux, gCC, 32bit	Red Hat Linux (RHE4)	gcc, GNU C 3.4.6
PC, Linux, gCC, 64bit	Red Hat Linux (RHE4)	gcc, GNU C 3.4.6
PC, Windows, Microsoft Visual C/ Matlab, 32bit	Windows XP	Microsoft Visual Studio .NET 7.0 C
PC, Windows, Microsoft Visual C/ Matlab, 64bit	Windows XP	Microsoft Visual Studio 2008 C
Sun, Solaris, SUN C/Matlab, 32bit	Solaris 9	Sun C 5.8

***Mice requires use of Matlab version 7.2 (R2006a) or higher**

Navigation and Ancillary Information Facility

Matlab Interface to CSPICE "Mice"

How to Access the CSPICE library Using Matlab©

January 2012

© The MathWorks Inc.

Topics

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

Mice Benefits

- 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)

- 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.

How Does It Work? (2)

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

- 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 the library and wrapper files.
- Note: You do not need a C compiler to use Mice.

Mice Operation (1)

- **A possible irritant exists in loading kernels using the `cspice_furnsh` function.**
  - **Kernels load into your Matlab <u>session</u>, not into your Matlab <u>scripts</u>. This means:**
    - » **loaded <u>binary</u> kernels remain accessible ("active") throughout your Matlab session**
    - » **data from loaded <u>text</u> 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)

- **Mitigation: two approaches**
  - **Load all needed SPICE kernels for your Matlab <u>session</u> 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 <u>script</u>:**
    - » **include a call to** `cspice_unload` **for each kernel loaded using** `cspice_furnsh`
    - » **or include a call to** `cspice_kclear` **to remove <u>ALL</u> kernel data from the kernel pool loaded using** `cspice_furnsh`

Mice Vectorization (1)

- **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 the array of 1000 ephemeris times in steps 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)

– **Access the *ith* state 6-vector (6x1 array) corresponding to the *ith* 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 *ith* string is the calendar date corresponding to `et(i)`.**

continued on next page

Mice Vectorization (3)

- Access the *ith* string of utcstr corresponding to the *ith* 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)

- **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.**

```
% 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);

[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)

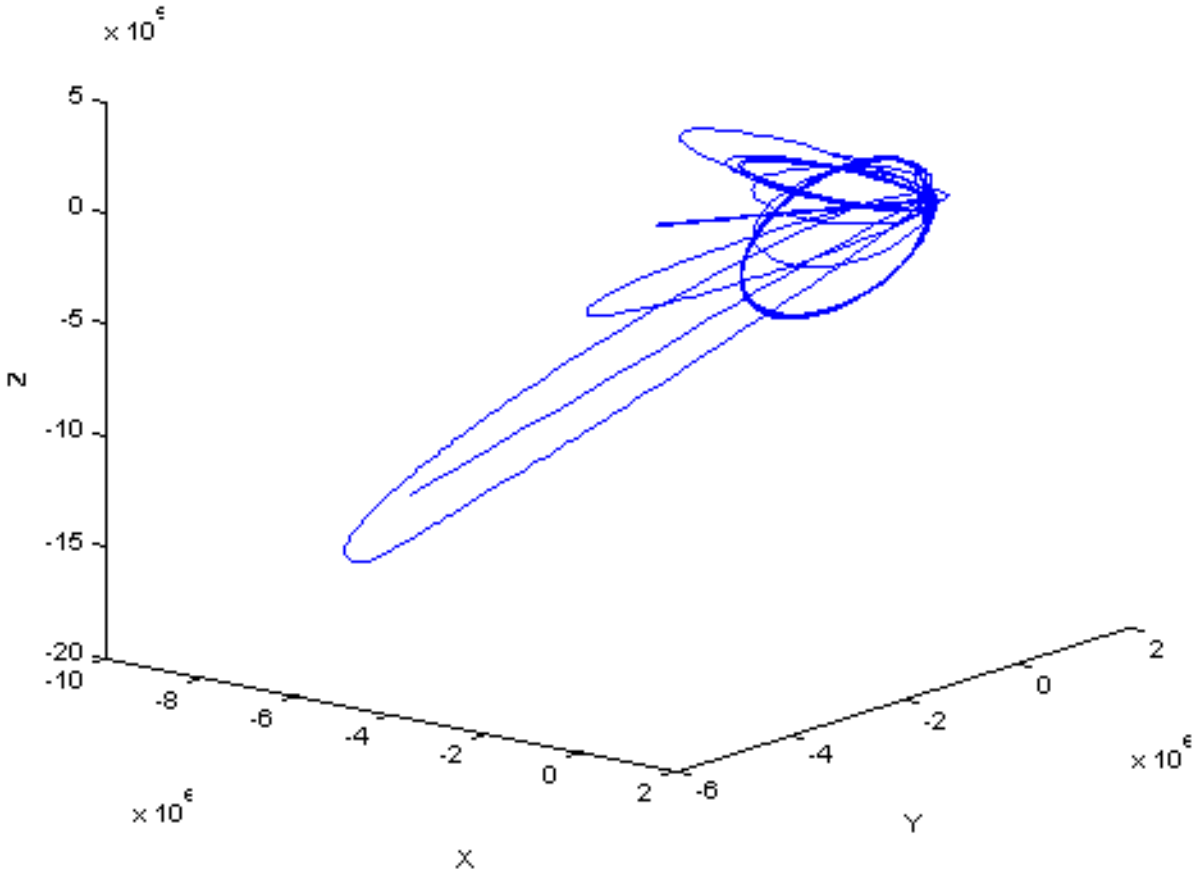
- **Repeat the example of the previous page, except use the `mice_spkezr` function to retrieve full state vectors.**

```
% 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
```

continued on next page

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

IDL Interface to CSPICE "Icy"

How to Access the CSPICE library from the Interactive Data Language (IDL)©

January 2012

© Exelis

Topics

- Icy Benefits
- How does it work?
- Distribution
- Icy Operation
- Vectorization
- Simple Use of Icy Functionality

Icy Benefits

- Ease of use: Icy operates as an extension to the IDL language regime.
- Icy supports more than three-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)

- 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.
- 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

continued on next page

How Does It Work? (2)

- **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  /usr/local/itt/idl64/bin/bin.linux.x86/`
    - » `cp icy.dlm icy.dll  C:\ITT\IDL64\bin\bin.x86\`
- **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_"`

Caveat: 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 to IDL_DLM_PATH the directory. This directory contains an "icy.dlm" but no "icy.so."

How Does It Work? (3)

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

- 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 the these files
- Note: You do not need a C compiler to use Icy.

Icy Operation (1)

- **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)

- **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.)**

Icy Operation (3)

- 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 you write:
    - provide a call to `cspice_unload` for each kernel loaded using `cspice_furnsh`
    - provide a call to `cspice_kclear` to remove ALL kernel data from the kernel pool

Icy Vectorization (1)

- 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)

– **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 *ith* state 6-vector corresponding to the *ith* 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)

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

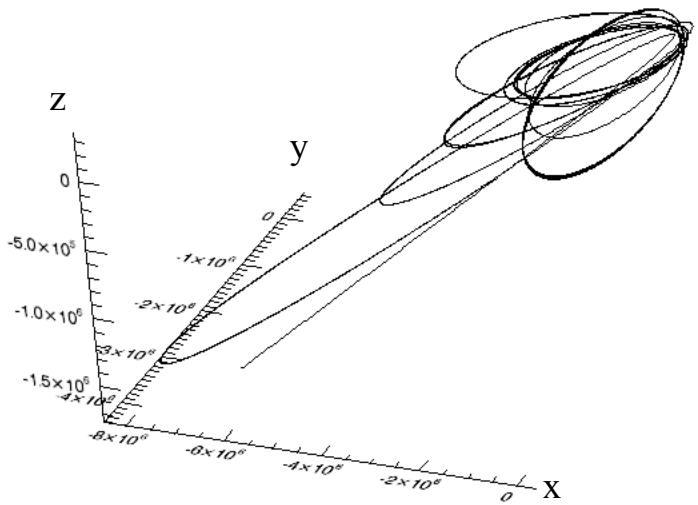
Simple Use of Icy Functionality

- **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:  "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 using IDL iTool

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

Using Module Headers

January 2012

Topics

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

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

Module Header Purpose

Navigation and Ancillary Information Facility

- NAIF uses module "headers" to provide SPICE users with detailed information describing a module's function and design.
  - In FORTRAN, C and MATLAB Toolkits the "headers" are comment blocks inserted in the source code
  - In IDL Toolkits, where there are no source code files, the "headers" exist as independent files
- All Toolkit distributions include HTML versions of the module headers.
- Using the HTML formats is usually the best approach because they are hyperlinked with other NAIF documentation
- The next charts provide the header locations

Fortran Module Header Locations

Navigation and Ancillary Information Facility

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

C Module Header Locations

Navigation and Ancillary Information Facility

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

Icy Module Header Locations

- In IDL ("Icy") toolkits, two sets of headers are provided.
  - Icy headers in HTML format:
    - » <path to Icy>/icy/doc/html/icy/index.html
  - CSPICE headers, in text and HTML formats:
    - » <path to Icy>/icy/src/cspice/<name>_c.c
    - » <path to Icy>/icy/doc/html/cspice/index.html
- The information provided in an "Icy" 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.

Mice Module Header Locations

Navigation and Ancillary Information Facility

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

```
>> help cspice_str2et
```

  - **CSPICE headers, in text and HTML formats:**
    - » **<path to Mice>/mice/src/cspice/<name>_c.c**
    - » **<path to Mice>/mice/doc/html/cspice/index.html**
- **The information provided in a "Mice" 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

- **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 principle ephemeris access module
str2et is a key time conversion module

Navigation and Ancillary Information Facility

Time Conversion and Time Formats

January 2012

Topics

- Time Systems and Kernels
- Converting Time Strings
- Converting Numeric Times
- Additional Time Conversions
- Pictorial Layout of the Time Conversions
- Backup

Time Systems and Kernels

Navigation and Ancillary Information Facility

- Time inputs and outputs in users' SPICE-based programs are usually strings representing epochs in these three time systems:
  - Coordinated Universal Time (UTC)
  - Spacecraft Clock (SCLK)
  - Ephemeris Time (ET, also referred to as Barycentric Dynamical Time, TDB)
- Time stamps in kernel files, and time inputs and outputs to SPICE routines reading kernel data and computing derived geometry, are double precision numbers representing epochs in these two time systems:
  - Numeric Ephemeris Time (TDB), expressed as ephemeris seconds past J2000
  - Encoded Spacecraft Clock, expressed as clock ticks since the clock start
- SPICE provides routines to perform conversions between string and numeric times using data from these two kernels:
  - Leapseconds Kernel (LSK) containing data for UTC <=> ET conversion
  - Spacecraft Clock Kernel (SCLK) containing data for ET <=> SCLK conversion

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, excepting SCLK. For example:**

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

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

Converting Numeric Times - 1

- **Numeric Ephemeris Time to Calendar, DOY or Julian Date UTC, TDB, or TDT String**
  - **TIMOUT (*et*, *fmtpic*, STRING)**
    - » ***fmtpic* is an output time string format specification, giving the user great flexibility in setting the appearance of the output time string and the time system used (UTC, TDB, TDT).**
      - **See a backup slide for examples of format pictures to produce a variety of output time strings**
      - **See the TIMOUT header for complete format picture syntax**
      - **The module TPICTR may be useful in constructing a format picture specification from a sample time string**
    - » **Requires LSK Kernel**
  - **ETCAL (*et*, STRING)**
    - » **STRING, fixed format ephemeris calendar time string, for example**
      '2000 JAN 01 12:16:40.123'
    - » **No LSK Kernel is required**

Converting Numeric Times - 2

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

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

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

Additional Time Conversions

- **Conversion between uniform time systems – numeric representations of TDB(ET), TAI, TDT, JDTDB(JED), JDTDT**
  - **Return value = UNITIM (*epoch*, *insys*, *outsys*)**
    - » **Requires LSK kernel**

- **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**

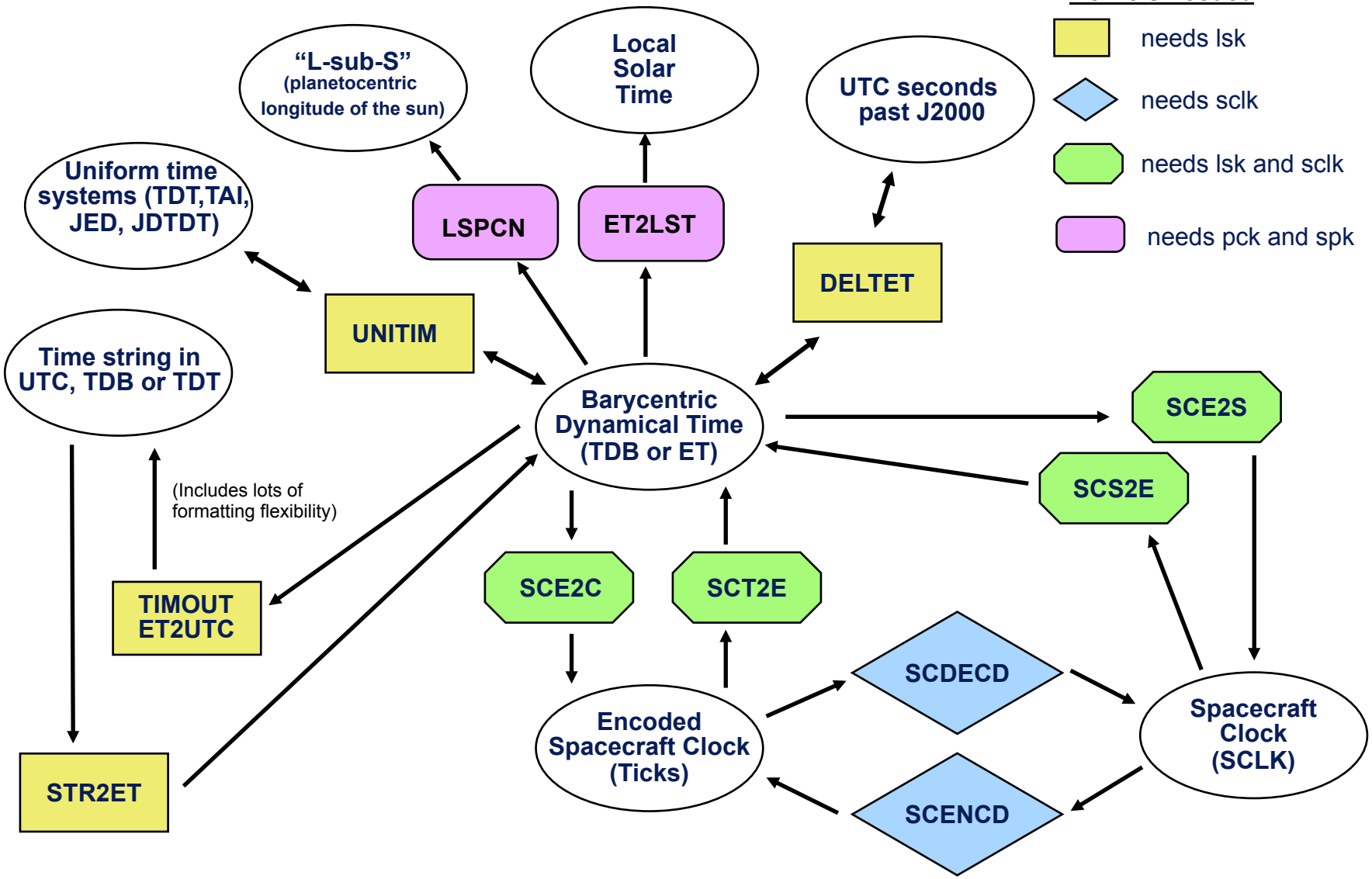
- **Numeric Ephemeris Time to planetocentric longitude of the Sun (Ls)**
  - **Return value = LSPCN (*body*, *et*, *abcorr*)**
    - » **While Ls is not a time system, it is frequently used to determine *body* season for a given epoch**
      - **Spring – 0° Ls; Summer – 90° Ls; Autumn – 180° Ls; Winter – 270° Ls**
    - » **Requires SPK and PCK kernels**

Conversions of Future Times

- **Caution: the long-term <u>future</u> relationships between UTC, TDB, and SCLK time systems cannot be accurately predicted due to:**
  - **Leapseconds**
  - **Spacecraft clock drift**

Principal Time System Interfaces

Navigation and Ancillary Information Facility



Backup

- Customizing the Time System
- Use of time format picture

Customizing the Time System

Navigation and Ancillary Information Facility

- **Defaults**
  - Two digit year (a bad idea but supported):1969-2068
  - Time System:
    - » For inputs and outputs: UTC
    - » For internal computations: TDB (ET)
  - Calendar: Gregorian
- **Adjustments**
  - The one hundred year interval to which two digit years belong may be set. For example 1980-2079
  - Time Systems: UTC, TDB, TT (Terrestrial Time)
  - Calendar: Gregorian, Julian, or Mixed.
- **See the TIMDEF module header and *Time Required Reading* (time.req) for details**

Use of Time Format Picture

Navigation and Ancillary Information Facility

Example Time Strings and the Corresponding Format Pictures

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

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

Navigation and Ancillary Information Facility

Leapseconds and Spacecraft Clock Kernels

LSK and SCLK

January 2012

Topics

- **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.)**

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.**
  - Utility programs using LSK: *spkmerge*, *chronos*, *spacit*, etc.
  - Subroutines using LSK: STR2ET, TIMOUT, ET2UTC, etc.
- **As with all SPICE kernels, load it using FURNSH.**
- **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 <u>always</u> the best one to use.
  - Announcement of each new LSK is made using the "spice_announce" system.
    - » http://naif.jpl.nasa.gov/mailman/listinfo/spice_announce
  - Updates can occur ONLY on January 1st and July 1st

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> . . .
                           32,   @1999-JAN-1
                           33,   @2006-JAN-1
                           34,   @2009-JAN-1 )

\begintext
```

Out of Date LSKs

- **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 contains data used in conversions from SCLK to other time systems.
- It is required by Toolkit utilities and routines that utilize SCLK time.
  - For example, the SPICE CK subsystem makes heavy use of spacecraft clock time.
- As with all SPICE kernels, 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's database.
    - » 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> . . .

\begindata

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   = ( 0.0000000000000E+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    = (
    0.0000000000000E+00     -6.3119514881600E+08     1.0000000000000E+00
    1.2098765056000E+10     -5.8393434781600E+08     1.0000000000000E+00
                  . . . <more coefficient records> . . .
    2.4179319365000E+11      3.1330950356800E+08     9.9999997500000E-01 )

\begintext
```

Forms of SCLK Time Within SPICE

- 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, whether for science or engineering/housekeeping data.
- 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 (ET, also called TDB).

Sample SCLK String

Navigation and Ancillary Information Facility

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

Partition Delimiter

Clock Field Delimiter* (not a decimal point)

`1/1609504792.123`

Partition: Accounts for clock resets or counter roll-over.

Least Significant Clock Field: Ranges from 0 to 255. Nominally 1/256th of a second increment.

Most Significant Clock Field: Ranges from 0 to 4294967295 (2^{32}-1). Nominally 1 second increment.

* 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.

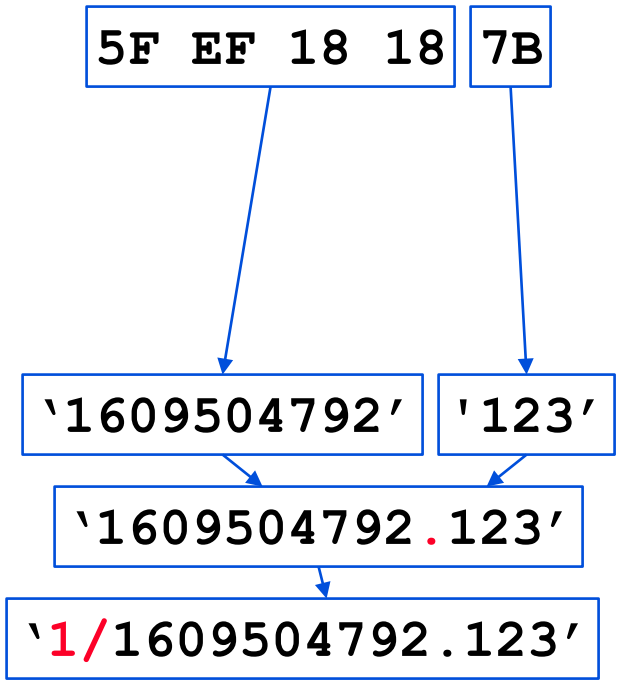
- **A partition is a NAIF-created construct to handle spacecraft clock rollovers or resets.**
- **When referring to epochs in the first partition, the leading '1/' may be omitted.**
- **Many modern spacecraft don't use a partition other than 1.**

Constructing an 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
  - First four bytes are an unsigned integer representing seconds
  - 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)



Encoded SCLK (Ticks)

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
  - 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

Backup

- Examples of SCLK strings
- SCLK Interface Routines

Sample Galileo SCLK String

Navigation and Ancillary Information Facility

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

Clock Field Delimiters

Partition Delimiter

`1/16777214:90:9:7`

Least Significant Clock Field: Ranges from 0 to 7. Nominally 1/120th of a second.

Partition: Accounts for clock resets or counter roll-over.

Intermediate Clock Field: Ranges from 0 to 9. Nominally 1/15th of a second.

Most Significant Clock Field: Ranges from 0 to 16777214. Nominally 60 2/3rd second increment.

Intermediate Clock Field: Ranges from 0 to 90. Nominally 2/3rd of a second.

More Sample SCLK Strings

Navigation and Ancillary Information Facility

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

- **Cassini**
  `1/1334314108.134`
- **DS1**
  `1/67532406.010`
- **Galileo**
  `1/16777214:90:9:7`
- **Genesis**
  `1/666230496.204`
- **MGS**
  `1/655931592.103`
- **MPF**
  `1/559627908.058`
- **Mariner 9**
  `1/11542909`
- **Mars Odyssey**
  `1/687231994.091`
- **NEAR**
  `1/40409721942`
- **Stardust**
  `1/697451990.042`
- **Viking 1&2**
  `1/32233616`
- **Voyager 1&2**
  `1/05812:00:001`
- **Mars Express**
  `1/0090979196.29713`
- **Venus Express**
  `1/0033264000.50826`
- **Rosetta**
  `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

SCLK Interface Routines

Navigation and Ancillary Information Facility

Convert SCLK times using the following routines

`SCS2E (SC, SCLKCH, ET)`	(SCLK String⇨ET)
`SCE2S (SC, ET, SCLKCH)`	(ET⇨SCLK String)
`SCT2E (SC, SCLKDP, ET)`	(Encoded SCLK⇨ET)
`SCE2C`[1] `(SC, ET, SCLKDP)`	(ET⇨Continuous Encoded SCLK)
`SCE2T (SC, ET, SCLKDP)`	(ET⇨Discrete Encoded SCLK)
`SCENCD(SC, SCLKCH, SCLKDP)`	(Encode SCLK)
`SCDECD(SC, SCLKDP, SCLKCH)`	(Decode SCLK)

[1] Use SCE2C (not SCE2T) for C-kernel data access.

Navigation and Ancillary Information Facility

Introduction to the SPICE Ephemeris Subsystem SPK

Focused on <u>reading</u> SPK files

January 2012

SPICE Ephemeris Data

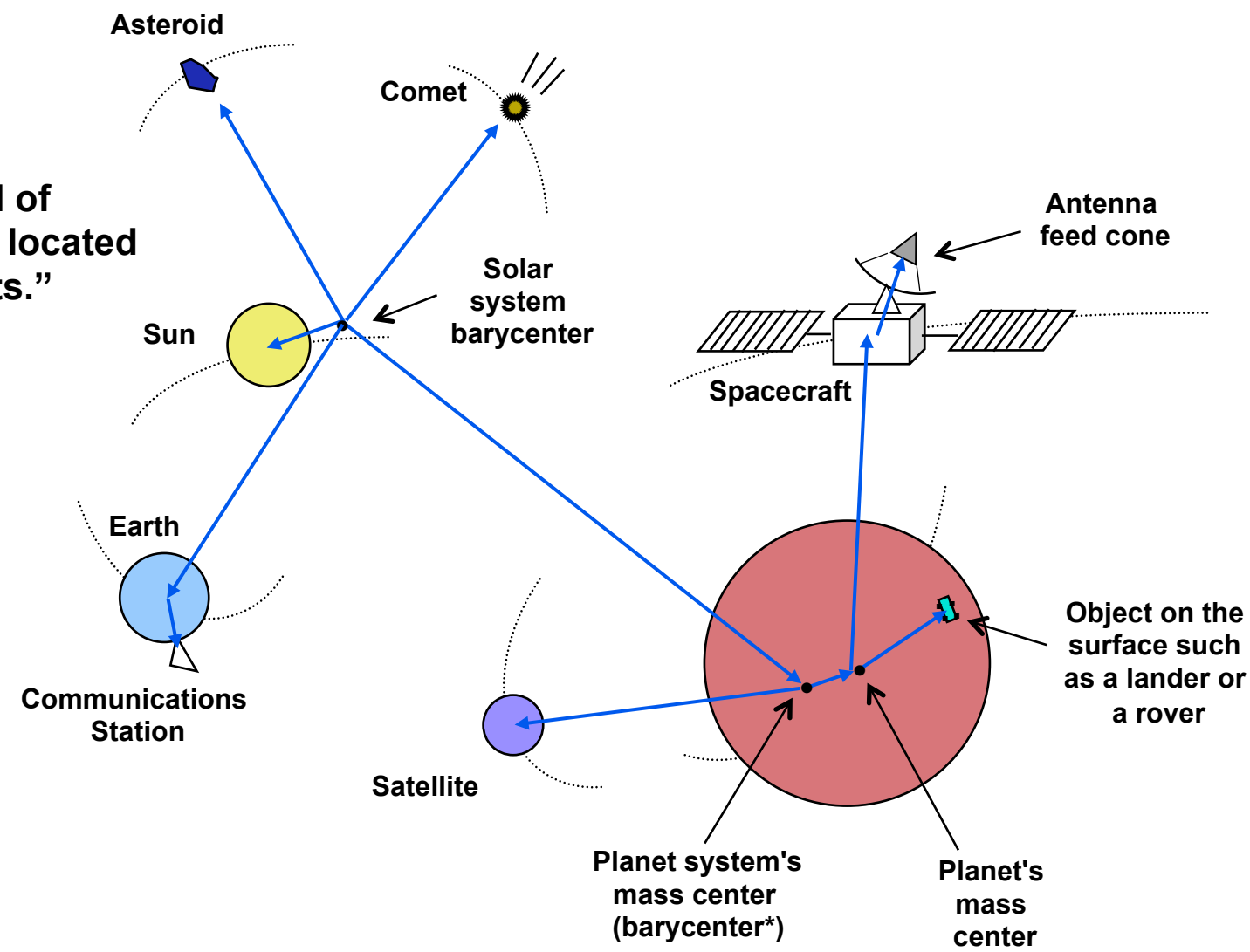
Navigation and Ancillary Information Facility

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

Examples of Ephemeris Objects

Navigation and Ancillary Information Facility

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



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

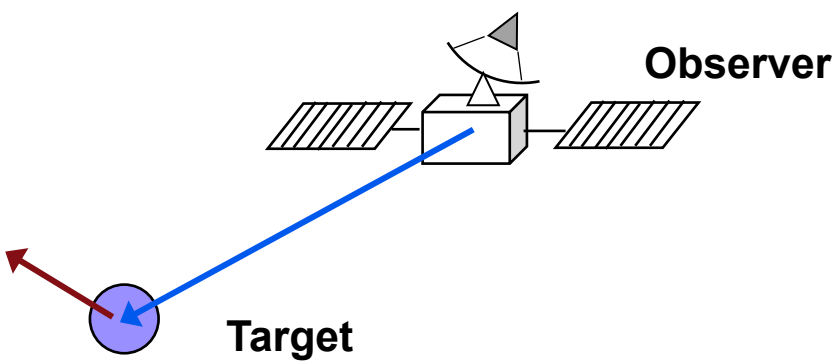
Inside an SPK: Bodies and Centers of Motion

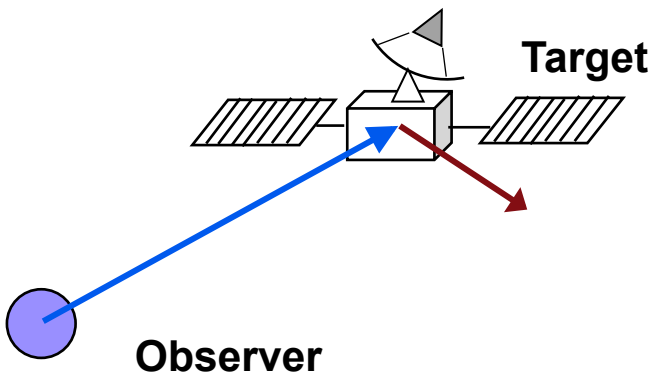
- **Inside an SPK file, ephemeris objects come in pairs: a "body" and its "center of motion."**
  - **The ephemeris is given for the body moving relative to the center of motion.**
    - » **For the position component, the vector points TO the body FROM the center of motion.**
  - **There can be, and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

Reading an SPK: Observers and Targets

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. ("State" means position and velocity)
  - The position data point from the "observer" to the "target."
  - The velocity is that of the "target" relative to the "observer."





Caution: state (observer, target) ≠ - state (target, observer) unless the state is geometric (no aberration corrections)

SPK File Coverage

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

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

Reference Frames Used in Writing and Reading SPKs

On Writing

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

On Reading

- The application "reading" an SPK file(s) must specify relative to what reference frame the output state or position vectors are to be given
  - This frame must be "known" to the SPICE-based program
  - "Known" means either a built-in frame ("hard coded") or one fully specified at run-time
  - The user's program may need to have access to additional SPICE data in order to construct some of these frames

Using SPK Files

An Introduction

Retrieving Position or State Vectors

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

Retrieving a State Vector

Navigation and Ancillary Information Facility

Fortran syntax used here

Initialization…typically done once per program execution

Tell your program which SPICE files to use ("loading" files)

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

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

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

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

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

Retrieve state vector from the SPK file at your requested time

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

inputs

outputs

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

Arguments of SPKEZR - 1

INPUTS

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

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

Arguments of SPKEZR - 2

- **CORRECTION: Specification of what kind of aberration correction(s), if any, to apply in computing the output state vector.**
  - Use 'LT+S' to obtain the apparent state of the target as seen by the observer. 'LT+S' invokes light time and stellar aberration corrections. ('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. 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.**

Retrieving a Position Vector

Navigation and Ancillary Information Facility

- **The SPICE routine 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**
- **When velocity data are not needed, using SPKPOS offers several advantages over using SPKEZR**
  - **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**

Retrieve a position vector from the SPK file at your requested time

CALL SPKPOS (target, tdb, 'frame', 'correction', observer, *positn, light time*)

inputs

outputs

Use the returned position vector in other SPICE routines

A Simple Example of Retrieving State Vectors

Navigation and Ancillary Information Facility

Initialization - typically do this just once per program execution

```
CALL FURNSH ( 'NAIF0009.TLS' )
CALL FURNSH ( 'HUYGENS_3_MERGE.BSP' )
```

Better to use a "furnsh kernel" instead of these individual FURNSH statements

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', 'HUYGENS PROBE',
             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 Huygens probe 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) and has been corrected for both light time and stellar aberration (LT+S). The one-way light time (LT) is also returned.

A SPICE leapseconds file (NAIF0009.TLS) is used, as is a SPICE ephemeris file (HUYGENS_3_MERGE.BSP) containing ephemeris data for the Huygens probe (-150), Saturn barycenter (6), Saturn mass center (699), Saturn's satellites (6xx) and the sun (10), relative to the solar system barycenter.

A Slightly More Complex Example - 1
Kernel Data Needed

- **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 planet in the planet's IAU body-fixed reference frame you'll need:**
    - » **PcK file containing orientation data for the planet**
    - » **SPK(s) for the object, planet, and (typically) planet barycenter**
    - » **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 - 2
Retrieving State Vectors

Obtaining a state vector in a body-fixed reference frame

Initialization...typically once per program execution

Tell your program which SPICE files to use ("loading" files)

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

Better to use a "furnsh kernel" instead of these three individual FURNSH statements

Loop... do as many times as you need

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

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

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

```
CALL SPKEZR (target, tdb, 'IAU_<body_name>', 'correction',
  observer, state, lightime)
```

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

Chaining Data

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

Example of Chaining

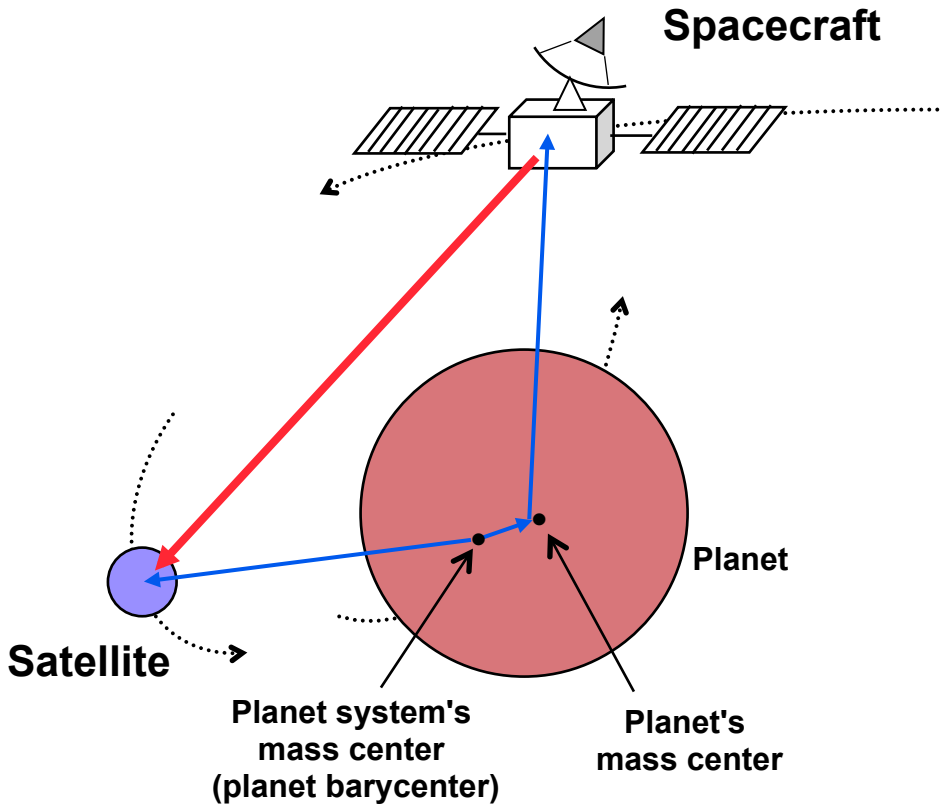
Navigation and Ancillary Information Facility

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

Your SPK may not contain <u>exactly</u> the ephemeris you want.

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

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



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

More About SPK Files

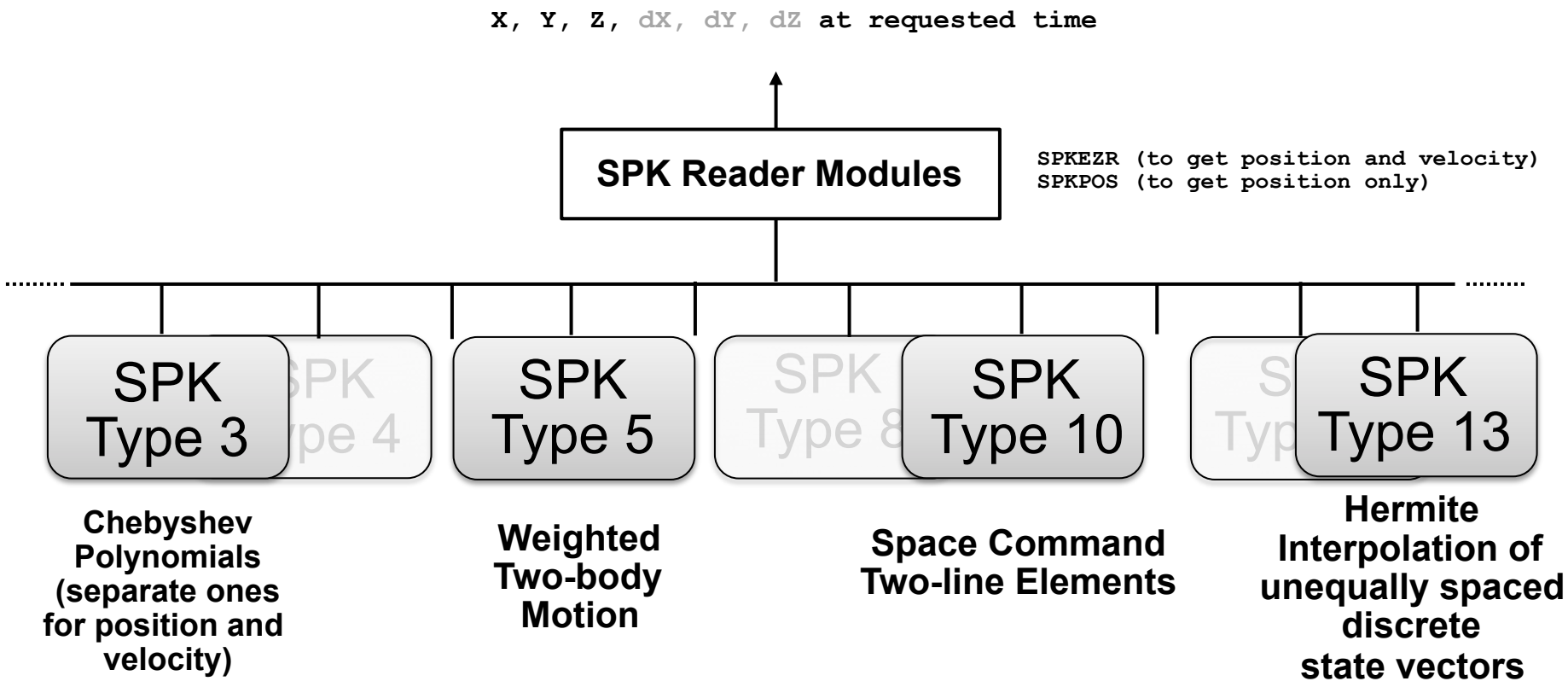
Here we provide some more details on the design and structure of SPK files.

However, still more information is provided in the "Making an SPK" tutorial.

SPK Data Type Concept

Navigation and Ancillary Information Facility

SPK files 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?

- To allow SPK producers to choose an ephemeris representation well-suited for their applications. 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 other formats.
  - Types 1, 2, 3, 8, 10, 14, 15, 17 and 18 were developed to enable accurate duplication of data obtained from ephemeris developers.

Widely Used SPK Data Types

- 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 non-JPL ephemeris producers and by MKSPK users
- Type 18 (Sliding window Hermite or Lagrange interpolation)
  - Used in SPKs made by ESA missions

SPK Segments - 1

- **An SPK file is mostly made up of one or more blocks of ephemeris data called "segments."**
  - **Each segment contains position and velocity data for one body, relative to one center of motion, given in one reference frame, using one mathematical representation (called the SPK "data type"), for a specified time interval.**
    - » **There are often many such segments within an SPK, each for a different time interval.**
  - **Additional data for a different body, or relative to a different center of motion, or given in a different reference frame, or using a different SPK data type, may be placed in one or more additional segments.**
    - » **This is also a common occurrence.**
  - **You can see the segment-by-segment contents of an SPK file using the SPACIT utility program.**

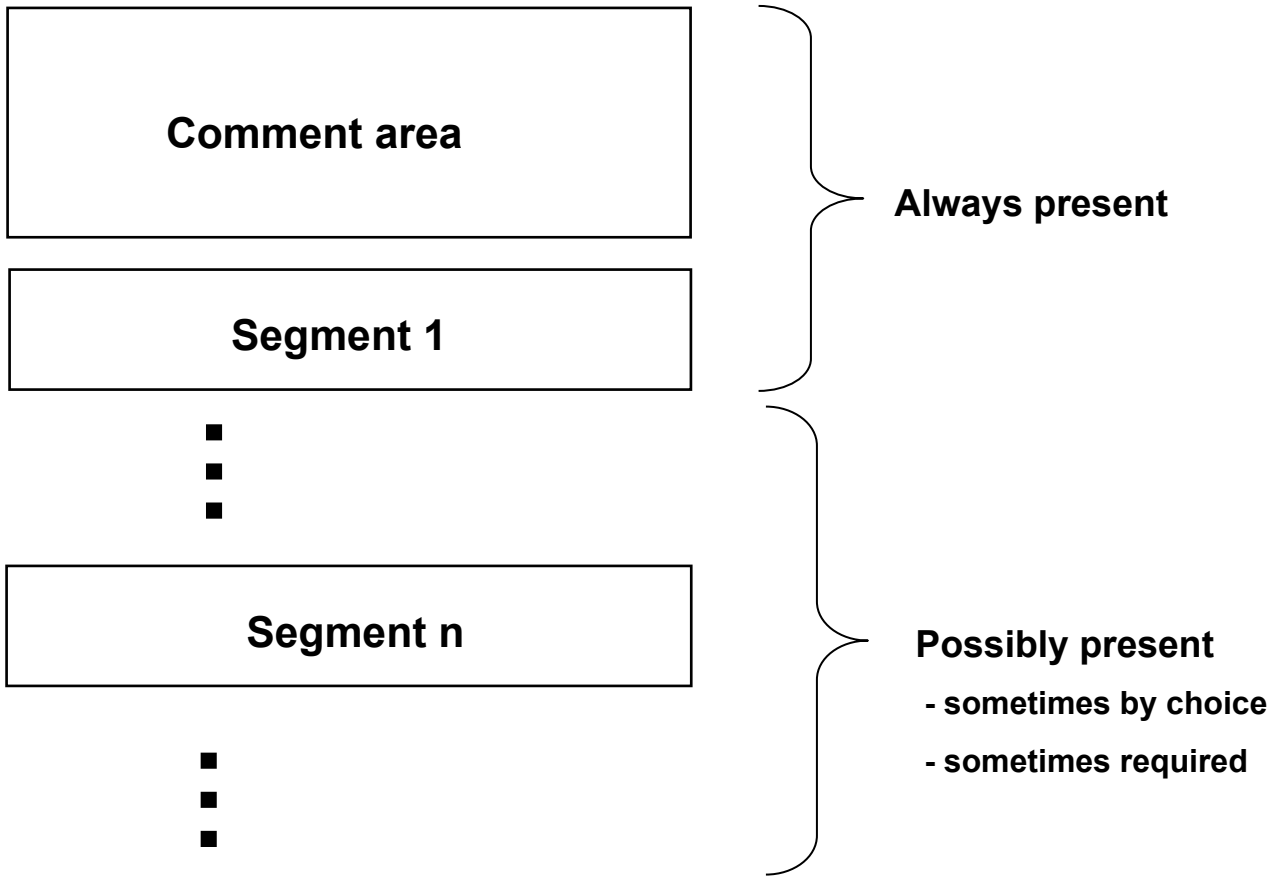
SPK Segments - 2

- **By definition an SPK segment contains a continuous representation of ephemeris data over an interval of time.**
  - **You can request a position or state vector at any time within the covered interval.**
    - » **Achieved through interpolation of discrete data, or by evaluating an analytical function, depending on the SPK type.**
- **However, interpolation across segments and extrapolation beyond the time bounds of a segment are not permitted by the SPK "readers."**

SPK File Structure: The User's View

Navigation and Ancillary Information Facility

Logical Organization of an SPK File



See the tutorial named Making an SPK for details about segment architecture and contents.

SPK Segment Order and Priority

- **Within a single SPK file…**
  - The segments in an SPK file need not be ordered according to time or body.
  - Segment order does imply priority. When 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 located 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: when 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.

Planets and Planet Barycenters

Navigation and Ancillary Information Facility

- A planet and its satellites orbit the planet system's barycenter
  - For example, the planet Jupiter's mass center (599) and each of Jupiter's satellites (501 - 5xx) orbit the Jupiter system barycenter (5)
- Planet system barycenters (i.e. 1 through 9) and the sun (10) orbit the solar system barycenter (0)
- 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)*
- Because the masses of Phobos and Deimos are so small compared to the mass of Mars, the mass center for Mars (499)* is treated as equivalent to the Mars barycenter (4)

* These equivalences hold true ONLY in the SPK subsystem, not in the PCK subsystem

SPK File Ephemeris Contents

- **A single SPK file can hold data for one ephemeris object, or for many ephemeris objects.**
  - **This is illustrated in the next three charts**

Example of Flight Project SPK Files

Navigation and Ancillary Information Facility

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

Probe

-150 = Huygens probe

One object

Orbiter

-82 = Cassini S/C

One object

Planets

0 = solar system bc
.
.
3 = Earth barycenter
.
6 = Saturn barycenter
.
.
10 = Sun mass center
299 = Venus mass center
399 = Earth mass center
301 = Moon

Multiple objects

Satellites - 1

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

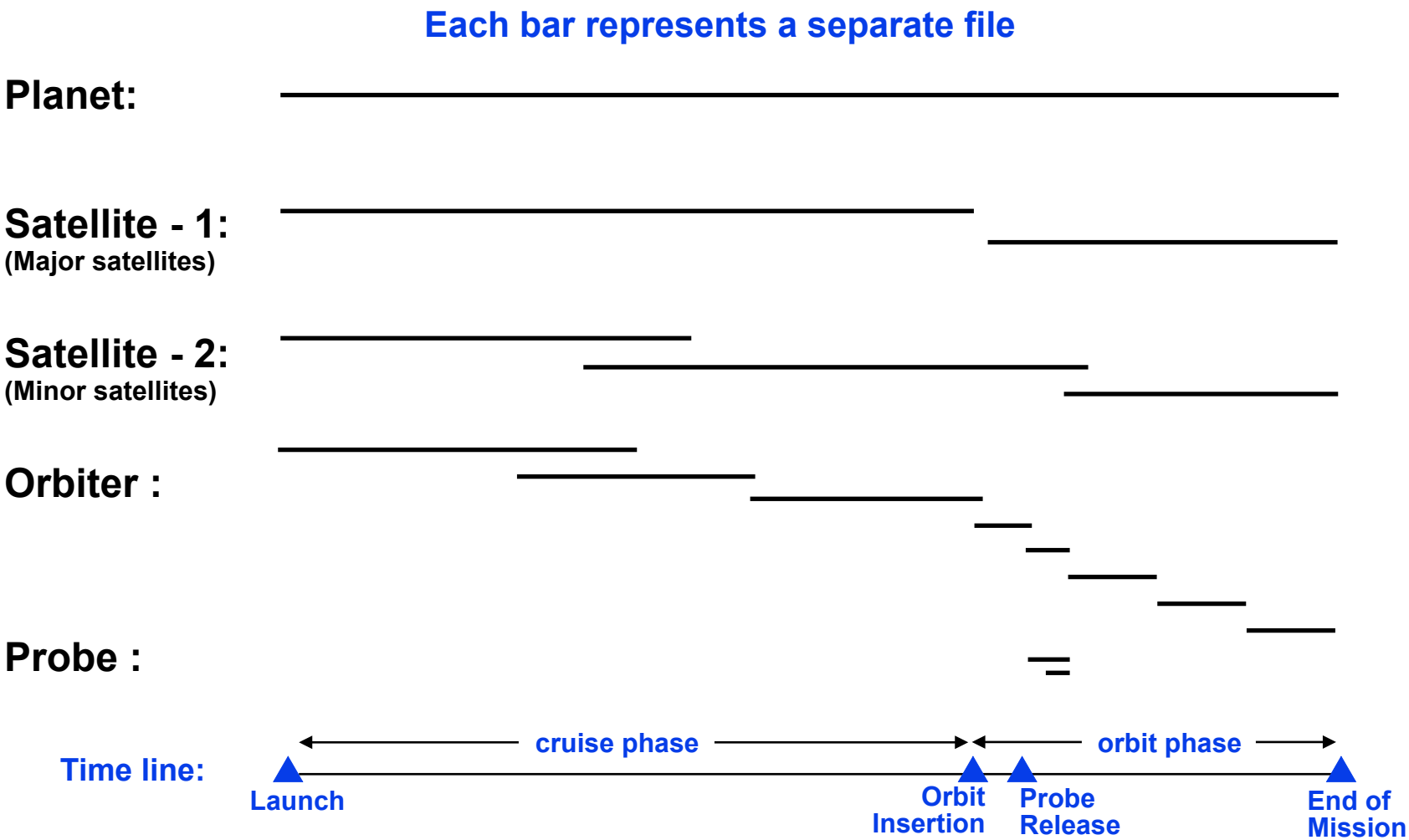
Satellites - 2

The user's program must "load" as many of these SPK files as needed to satisfy her/his requirements. NAIF strongly recommends that such programs have the flexibility to load a list of SPK files provided to the program at run-time; this is easily accomplished by listing the SPK files in a "meta kernel" using the Toolkit's FURNSH routine.

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



* 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.

Examples of Generic SPK File Contents

Navigation and Ancillary Information Facility

de421.bsp
Planet Ephemeris

0	S.S. BC
1	Merc. BC
199	Mercury
2	Venus BC
299	Venus
3	Earth BC
301	Moon
399	Earth
4	Mars BC
499	Mars
5	Jupiter BC
6	Saturn BC
7	Uranus BC
8	Neptune BC
9	Pluto BC
10	Sun

my_asteroids.bsp
Asteroid Ephemeris

2000253	Mathilde
2000433	Eros

jup230.bsp
Merged Planet and Satellite Ephemeris

3	Earth BC
399	Earth
5	Jupiter BC
501	Io
502	Europa
503	Ganymede
504	Callisto
505	Amalthea
514	Thebe
599	Jupiter
10	Sun

Generic SPK files may be obtained from the NAIF server

SPKs for Objects Located on the Surface of a Natural Body

- 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
    - » Needed to rotate body-fixed vectors to the J2000 frame

Understanding an SPK File

- 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
- These comments may be extracted or viewed using an API (subroutine) or a SPICE utility program.

Manipulating and Using SPK Files

Navigation and Ancillary Information Facility

- You can subset an SPK, or merge two or more SPKs
  - The merge may be keyed off of objects, or time, or both
- 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

  (* The allowed number of simultaneously loaded DAF-based files is currently set to 1000.)
- You can convert an SPK that is in non-native binary format to native binary format if you need to add data or comments

Summarizing an SPK File - 1

Navigation and Ancillary Information Facility

- A brief summary can be made using the SPICE Toolkit utility "brief"
  - Summary is for objects present and their start and stop epochs
- At your command line prompt, type the program name (with path), followed by the name of the binary SPK file that is to be summarized
- See the brief User's Guide or on-line help (%brief -h) for more details

```
% brief 070413BP_SCPSE_07097_07121.bsp
Brief.  Version: 2.3.1        (SPICE Toolkit N0061)


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)              End of Interval (ET)
        -------------------------------     -------------------------------
        2007 APR 07 16:22:23.000            2007 MAY 01 09:34:03.000
```

Note, the default is ET, not UTC!

Summarizing an SPK File - 2

Navigation and Ancillary Information Facility

- A detailed summary can be made using the Toolkit utility “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 - 3

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.

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 or merges one or more SPK files

- **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

Additional Information on SPK

- For more information about SPK, look at the following:
  - Backup slides in this tutorial
  - STATES cookbook program (source code) and its User's Guide
  - 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

- Problems Using SPK Files
- Don’t Mix Planet Ephemerides
- Barycenters and Mass Centers
- Effect of Aberration Corrections
- Retrieving State Vectors: "Under the Hood”
- SPK File Structure

Problems Using SPK Files - 1

- 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 DELTAET is about 65 seconds as of 9/11)
  - 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

- **You have requested aberration-corrected states but the 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:**

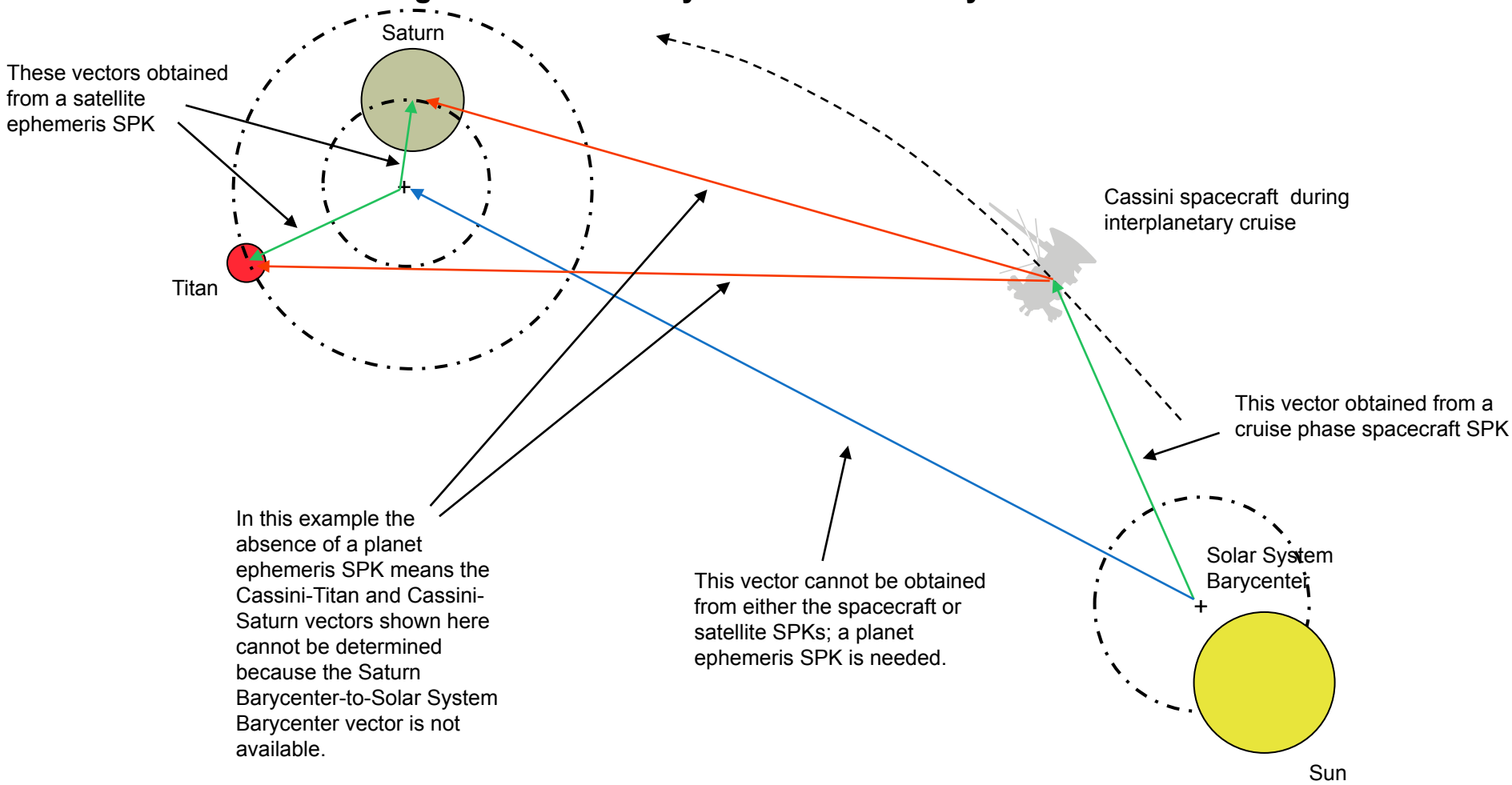
```
SPICE(SPKINSUFFDATA) - -
Insufficient ephemeris data has been loaded to
compute the state of xxx relative to yyy.
```

Problems Using SPK Files – 3a

- **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 relative to the solar system barycenter. 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.**

Problems Using SPK Files - 3b (drawing)

Navigation and Ancillary Information Facility



Given…

available in SPK files

not available in SPK files

Therefore…

can't be computed

Problems Using SPK Files - 4

- 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

- **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

- **Other missing data… not obvious.**
  - You may need CK (and SCLK), FK or PcK data to construct a state (or position) vector in your requested output frame.
- **Mistaking ET for UTC, or vice-versa.**
- **You must have loaded sufficient SPKs to be able to chain states to the solar system barycenter if doing aberration corrections.**
- **Using light time corrections requires target ephemeris data at the light time-corrected epoch.**
  - If you're working near the beginning of an SPK, the light time-corrected epoch may occur earlier than available data.

Problems Using SPK Files - 7

- **You've assumed that:**

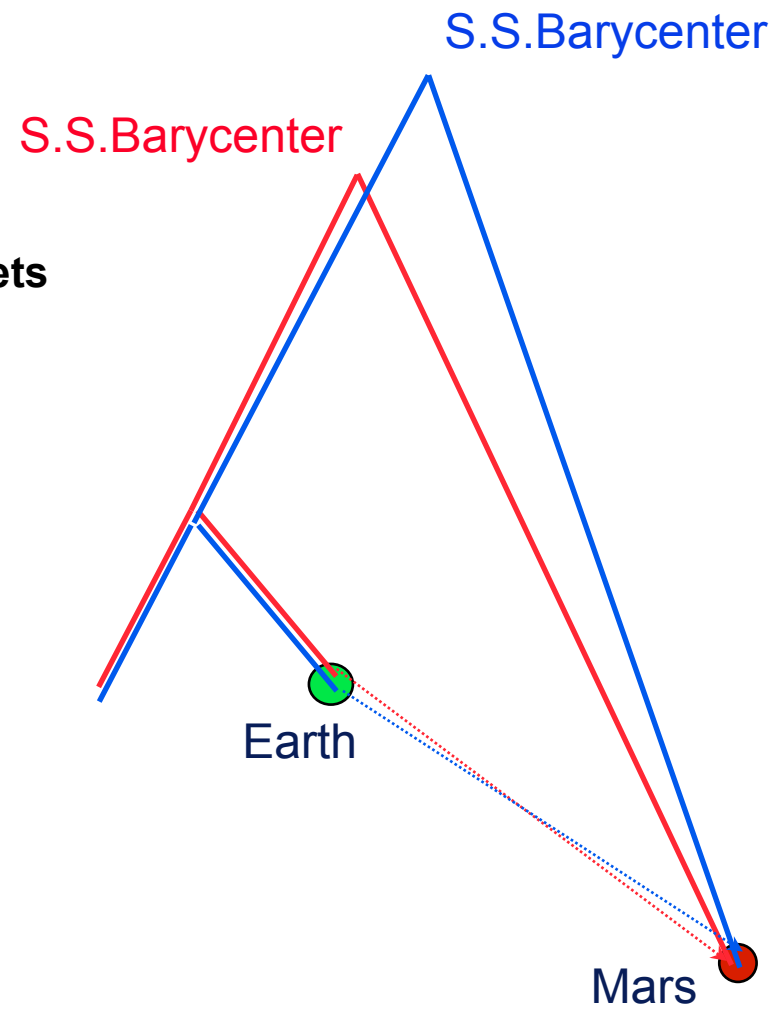
  state (observer, target) = - state (target, observer)

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

Comparing Apples and Oranges-1

Navigation and Ancillary Information Facility

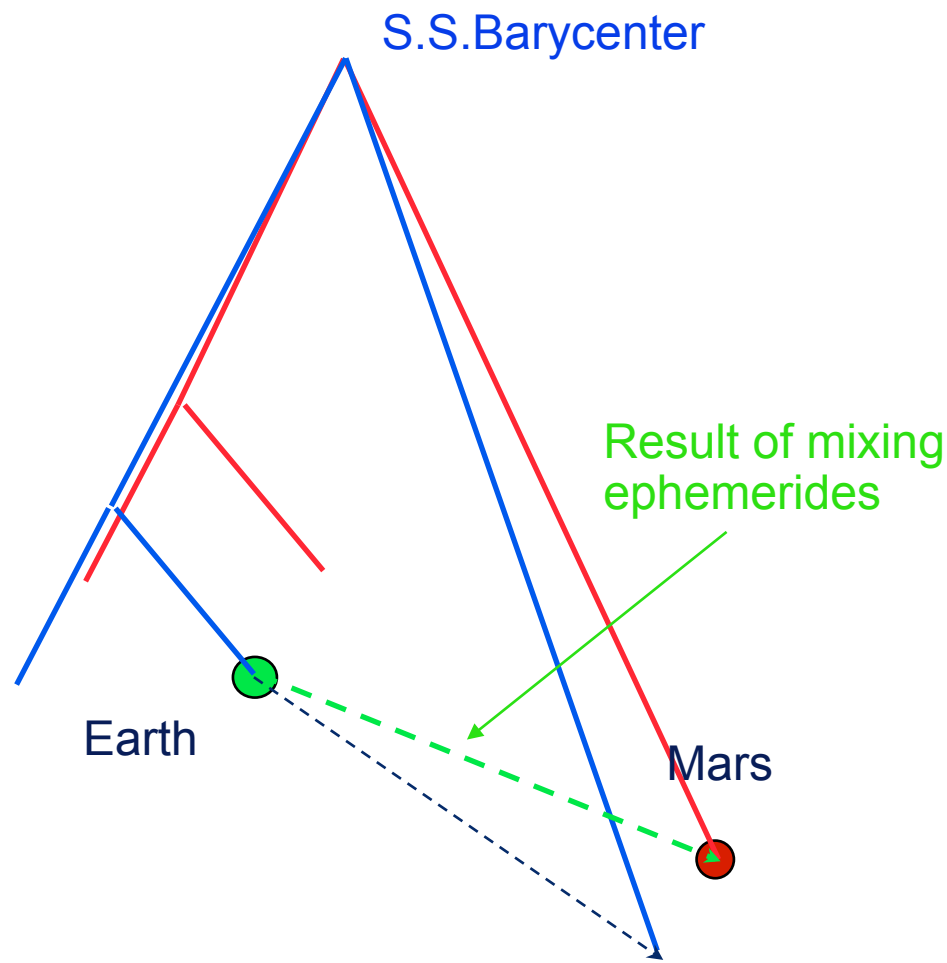
- With each new integration of the solar system, the solar system barycenter moves w.r.t. the planets
- Planet to planet offset variations are much smaller than the barycenter to planet variations



Comparing Apples and Oranges-2

Navigation and Ancillary Information Facility

- SPICE allows you to "load" different planetary ephemerides (or portions of them)
  - » Potentially can 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



Barycenter Offset Magnitude

Navigation and Ancillary Information Facility

Body Mass Center	System Barycenter	Barycenter offset from body mass center (km)*	Offset as % of body radius*
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	~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)	2080	174%

* Estimated maximum values over the time range 2000-2050

Effect of Aberration Corrections - 1

- Angular offsets between corrected and uncorrected position vectors over the time span 2004 Jan 1 to 2005 Jan1
  - 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

- **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

- **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)

Examples of Retrieving State Vectors – 2

- **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

  - » 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

- 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 2012

Topics

- Overview
- Using PCKs
- Text PCKs
  - IAU Models
- Binary PCKs
- Interface Routines
- PCK Reference Frames

Overview

Navigation and Ancillary Information Facility

- The Planetary Constants Kernel (PCK or PcK) is a component of the logical "planet kernel."
  - As implemented in SPICE, PCKs are distinct from the planet ephemeris information provided in SPKs.
- The PCK subsystem supports text and binary PCK file formats.
  - Usually text PCK data consist of:
    - » Orientation (also known as "rotation") models for extended, natural solar system bodies: sun, planets, natural satellites, a few asteroids
      - Location of the pole and prime meridian
      - Axis directions of a body-fixed, body-centered reference frame
      - Spin rate
    - » Physical and cartographic constants
      - Sets of radii for triaxial shape models.
      - Additional items could be included, such as
        - prime meridian offset from the principal axis
        - magnetic dipole location
        - gravity parameters: GM, J2, higher order gravity field terms
        - ring model parameters
  - Binary PCKs are used only for high-accuracy orientation data.
    - » At NAIF, binary PCKs are available only for the earth and the moon.

Text PCKs - 1

Navigation and Ancillary Information Facility

- **Text PCK files contain orientation, shape and other data associated with natural solar system bodies.**
- **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 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 MAKSPK 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

- The SPICE text kernel mechanism is used to implement IAU/IAG-based generic PCK files.
  - Users may easily visually inspect data.
  - 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.
  - Kernel variables contain the mathematical terms appearing in rotation or shape models. For example:
    - » `BODY699_RADII = ( 60268 60268 54364 )`
    - » `BODY699_POLE_RA = ( 40.589 -0.036 0. )`
    - » `BODY699_POLE_RA = ( 83.537 -0.004 0. )`
  - 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.

IAU Orientation Models - 1

- **SPICE text PCK orientation models use data from the IAU/IAG:**
  - **for the sun and planets:**
    - » **IAU 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.**
    - » **Trigonometric polynomial terms are supported by SPICE**
      - **but are rarely used in IAU models for planet orientation**
  - **for natural satellites:**
    - » **Additional trigonometric polynomial terms are used to more accurately represent precession and nutation.**
    - » **A few satellites exhibit chaotic rotation and so are not modeled.**
  - **for some major asteroids (e.g. Ida, Eros, Gaspra, Vesta)**

IAU Orientation Models - 2

- IAU body-fixed frames are planetocentric. For planets and satellites:
  - Z-axis is aligned with +/- spin axis. The positive Z-axis points toward the north side of the "invariable plane of the solar system."
  - The "invariable plane" is normal to the solar system's angular momentum vector. It is:
    - » approximately the same as Jupiter's orbital plane.
    - » roughly parallel to the ecliptic plane.
  - X-axis defines the prime meridian.
  - Y-axis completes the right-handed frame.
- The IAU chooses as its base frame the International Celestial Reference Frame (ICRF), as defined by the International Earth Rotation Service (IERS).
  - For historical and backwards compatibility reasons SPICE uses the names "J2000" and "EME2000" as synonyms for the ICRF inertial reference frame, even though J2000 and ICRF are, in fact, not identical. (The difference is "well under 0.1 arc seconds.")
- The IAU reference epoch for rotational models is 2000 Jan 1 12:00:00 TDB, frequently referred to as "J2000."
  - Note: This use of "J2000" is to identify an epoch in time, and should NOT be confused with "J2000" used in SPICE and elsewhere to refer to the ICRF inertial reference frame.

IAU Shape Models

- SPICE text PCK shape models use data from the IAU/IAG
- IAU shape models are nominally triaxial ellipsoids
  - For many bodies, two of the axes (equatorial axes) have the same value (spheroidal)
  - For some bodies, one or more radii have not been determined.
- Although many bodies are in fact modeled as spheres or spheroids, SPICE deals with the general, triaxial case.
  - Exception: SPICE supports geodetic coordinate transformations only for bodies modeled as spheres or spheroids.
    - » RECGEO and GEOREC are the modules performing these transformations.
  - Exception: SPICE supports planetographic coordinate transformations only for bodies modeled as spheres or spheroids.
    - » PGRREC, RECPGR, DPGRDR and DRDPGR are the modules supporting these transformations.

Special Case: Binary PCKs

- **The SPICE system stores high-accuracy orientation models in binary PCKs.**
  - Binary PCKs are implemented using the DAF file architecture (as are SPK files)
  - SPICE Toolkit utilities enable reading and writing comments, summarizing, and porting binary PCKs.
  - Like SPK files, binary PCKs support multiple data representations ("data types").
    - » Type 2: Chebyshev polynomials for Euler angles, angular velocity obtained by differentiation, constant interval length.
    - » Type 3: Separate Chebyshev polynomials for Euler angles and their derivatives, variable interval length.
- **Binary PCKs are limited to storing <u>orientation</u> data.**
  - Applications that require <u>shape</u> data must also load a text PCK.
- **Orientation data from a binary PCK <u>always</u> supersede orientation data (for the same object) obtained from a text PCK, no matter the order in which the kernels are loaded**
- **Binary PCKs are available for the Earth and Moon.**
  - The orientation data provided by these kernels are much more accurate than those provided by generic text PCKs based on the IAU/IAG reports.
  - These kernels are the topic of a tutorial on high-accuracy orientation data and associated frames for the Earth and Moon.

PCK Reference Frames

- Many PCK reference frame specifications are built-in to SPICE.
  - Just load PCK files to use these frames. Examples:
    - » IAU frames: IAU_SATURN, IAU_TITAN, IAU_EARTH, IAU_MOON, etc.
    - » IERS frames: ITRF93
- Other PCK frames are not built in and must be specified at run time by loading frame kernels, for example:
  - 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.
  - Lunar body-fixed frames: MOON_ME, MOON_PA
    - » See the tutorial on “high-accuracy” orientation data and associated frames for the Earth and Moon” for details.
- SPICE makes default associations between bodies and built-in PCK frames
  - For example, the default PCK frames for the planets are IAU_MERCURY, IAU_VENUS, IAU_EARTH, etc.
  - You can look up the default PCK frame associated with a body by calling CNMFRM or CIDFRM.

Using PCKs

Navigation and Ancillary Information Facility

- Load PCKs using FURNSH
- PCK orientation data are usually accessed via Frame system or SPK calls
  - 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 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)
  - Example: 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 specified reference frame is inertial
        - This call, in the course of its work, converts the position of the DSN station relative to the Earth’s center from an Earth-fixed, earth-centered frame to the J2000 frame.
- Access to PCK shape and other data is discussed in the section titled “Interface Routines”

Interface Routines - 1

Navigation and Ancillary Information Facility

- **Call FURNSH 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 specified reference frame is inertial
      - This call, in the course of its work, converts the position of the DSN station relative to the Earth's center from an Earth-fixed, earth-centered frame to the J2000 frame.

Interface Routines - 2

- **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

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

PCK Utility Programs

- **The following utility programs included in the Toolkit work with PCKs:**

BRIEF	summarizes coverage for one or more binary PCK files
SPACIT	generates segment-by-segment summary of a binary PCK file
COMMNT	reads, appends, or deletes comments in a binary PCK file
FRMDIFF	samples or compares orientation of a PCK-based frame

- **These additional utility programs, provided on the NAIF Web site, also work with PCKs:**

DAFMOD	alters frame IDs in a binary PCK file
DAFCAT	concatenates together binary PCK files
BFF	displays binary file format of an binary PCK file
BINGO	converts binary PCK files between big-endian and little-endian formats

Additional Information on PCK

- 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

Backup

Changing the Default Frame

- Some (mostly deprecated) SPICE routines implicitly use the default PCK frames (IAU_<body name>).
- You can change the default PCK frame associated with a body by loading a frame kernel that assigns a new default frame to that body.
  - For the Earth or Moon, you can load a "frame association kernel" provided by NAIF.
  - For any body, you can load a frame kernel containing the assignment

    ```
    OBJECT_<body name>_FRAME = '<new default frame name>'
    ```

    » Example: `OBJECT_MOON_FRAME = 'MOON_ME'`
- For high-accuracy work involving the Earth or Moon and any SPICE routines that use the default PCK frames, you normally would override the SPICE default frames by loading frame association kernels.
  - Reference the tutorial on "high-accuracy" orientation data and associated frames for the Earth and Moon for details.

Navigation and Ancillary Information Facility

“Camera-matrix” Kernel CK

(Orientation or Attitude Kernel)

Emphasis on reading CK files

January 2012

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" ⇒ 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

- **A single CK file can hold orientation data for one, or for any combination of spacecraft or their 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 practice CKs usually contain data for just one structure**

CK File Varieties

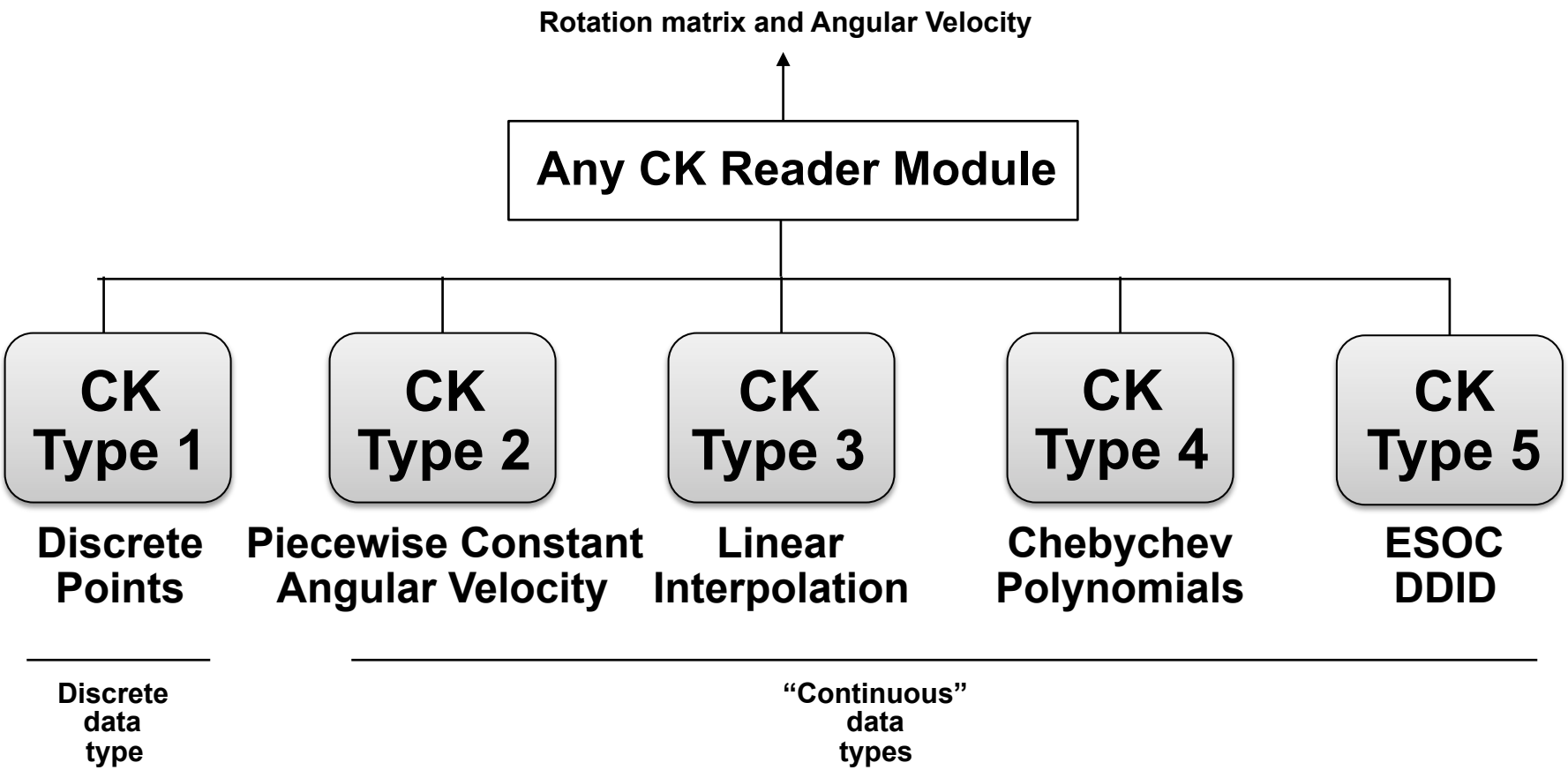
- "Reconstruction" CK (also called "definitive" CK)
  - A CK file can be made from orientation telemetry returned from a spacecraft or other structure
  - A CK might also be made from some process that improves upon the pointing determined from downlinked telemetry ("C-smithing")
  - These kinds of files are generally 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
  - This kind of file is generally used for engineering assessment, science observation planning, software testing and quick-look science data analysis
    - » If it has good fidelity, such a file might be used to "fill in the data gaps" of a reconstruction CK file
    - » In some cases (ESA in particular) the predict meets the reconstruction accuracy requirements; a true reconstruction CK is not produced

CK Data Types Concept

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

- To get orientation (rotation matrix) and angular velocity of a spacecraft or one of its moving structures, one needs at least three SPICE kernel types: CK, SCLK and LSK. One may also need an FK if planning to access CK data via high level SPICE interfaces.
  - CK contains spacecraft or other structure orientation
  - SCLK and LSK contain time correlation coefficients used to convert between ephemeris time (ET) and spacecraft clock time (SCLK)
    - » Sometimes an LSK is not needed in this conversion, but best to have it available as it is usually needed for other purposes
  - FK associates reference frames with CK IDs
    - » Needed if high level SPICE interfaces are used to access CK data (see next page)

What SPICE Routines Access CKs?

- 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: 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 this tutorial, along with topics applicable to all of the routines mentioned above

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

```
CALL CKGP   (instid,sclkdp,tol,'ref_frame',cmat,    clkout,found)
or
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

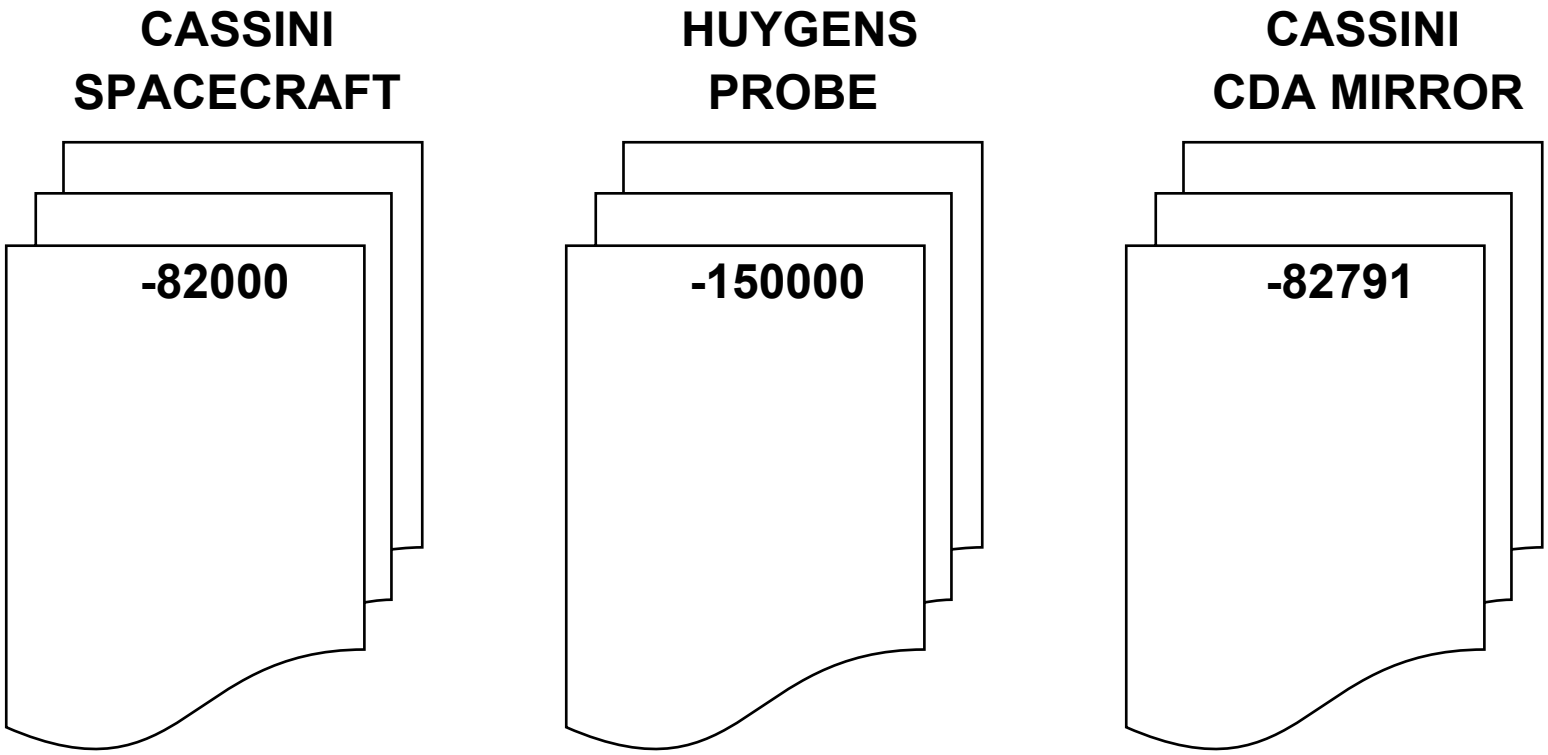
cmat	the 3x3 rotation matrix that you requested
av	the angular velocity that you requested
clkout	the exact time for which the orientation and angular velocity was computed
found	the logical flag indicating whether the orientation and angular velocity data were found. Note that if the loaded CK file(s) 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 should not be used!

Always check the FOUND flag returned by CKGPAV and CKGP!

* tol is explained in detail in backup slides

An Example of Project CK Files

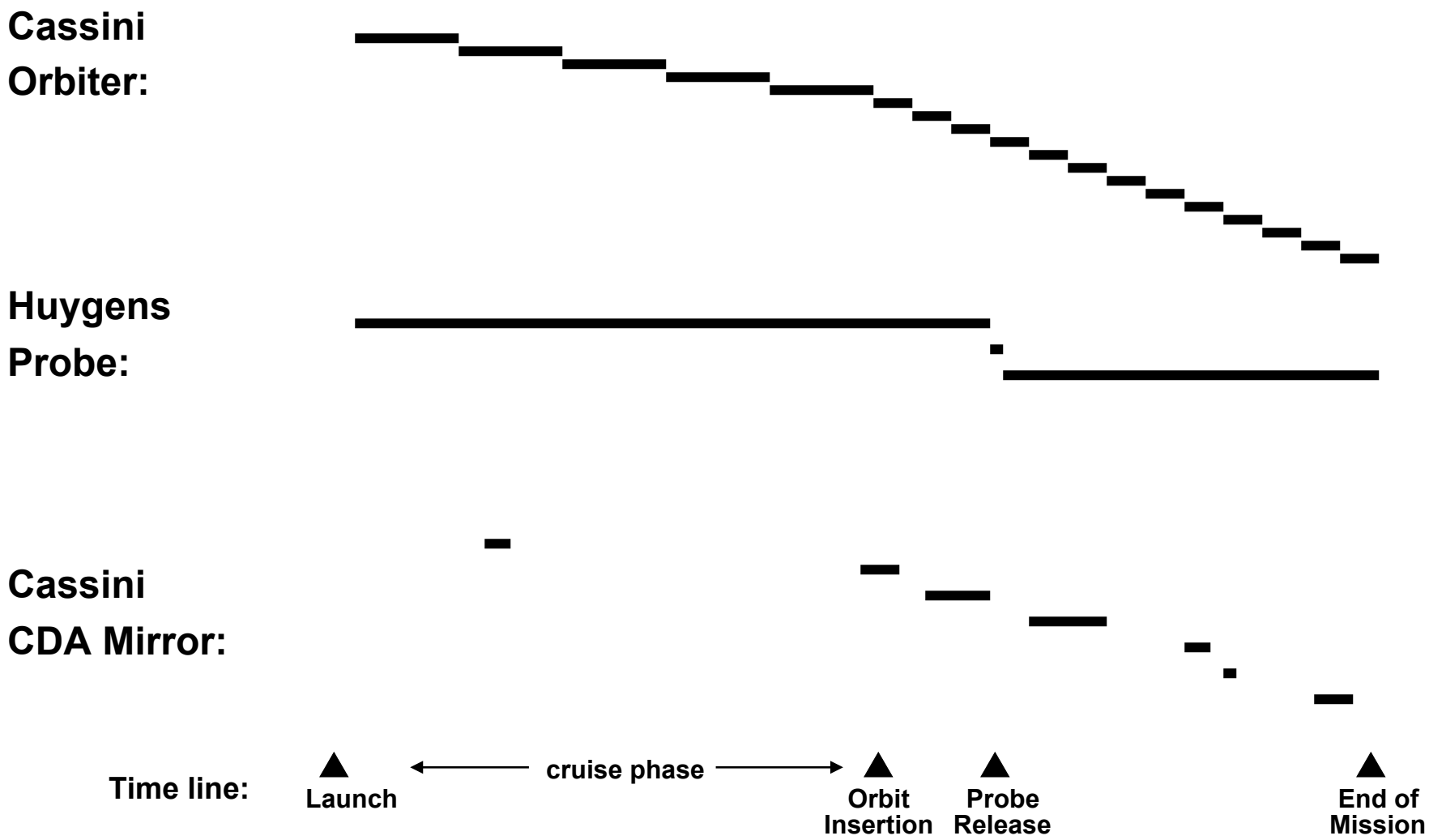
Navigation and Ancillary Information Facility



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 user's 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 FURNSH routine.

Sample* CK File Coverage - 1

Navigation and Ancillary Information Facility

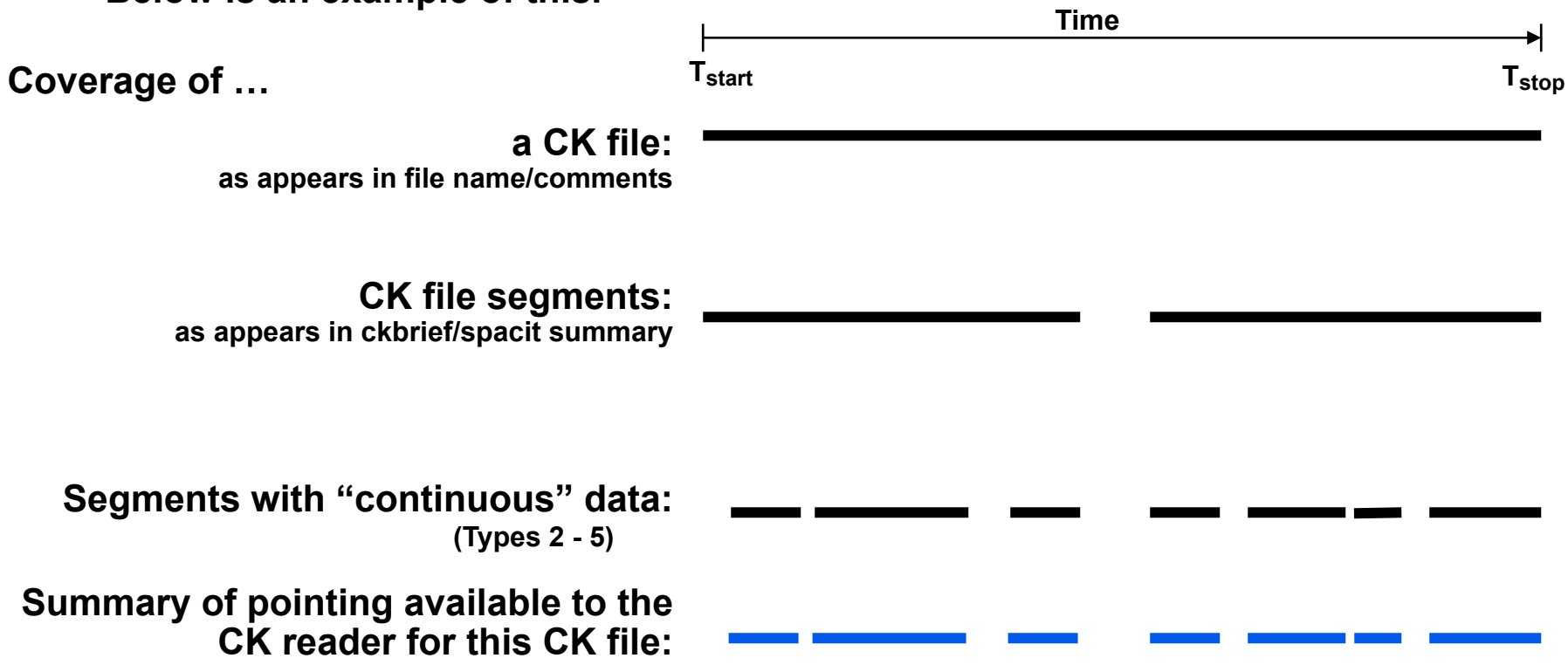


* Note: This is not an actual Cassini/Huygens scenario; it is a highly simplified illustration of some of the possibilities for orientation delivery on a planetary mission.

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 almost always contain gaps in coverage! Below is an example of this.



Blue line segments represent interpolation intervals–times when pointing will be returned and the FOUND flag set to “TRUE.”

What is an Interpolation interval?

Navigation and Ancillary Information Facility

- An interpolation interval is a time period for which the CK reader routines can compute and return pointing.
  - For CK Types 3 and 5 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 the CK readers 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.

Obtaining Coverage of CK File

- 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: an 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.

Make Use of CKCOV

- 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.
- 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

Summarizing a CK file - 1

- A brief 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)

```
% ckbrief 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

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

```
% ckbrief -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

continued from previous page

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

```
% frmdiff -t dumpg \
          -k cas_v40.tf naif0008.tls cas00106.tsc \
          -f 'YYYY-MM-DDTHR: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:**

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

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

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

(*) DAFCAT and SKSMRG 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

- **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

- The meaning of Tolerance
- Examples of Problems Encountered When Using CK Subsystem

The Meaning of Tolerance - 1

- The 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 - 5), if pointing information is not found within +/- tol of your pointing request time, no pointing will be returned and the "found flag" will be set to "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 later instance will be selected.
  - For Types 2 - 5 (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.

*Ignoring segment priority

The Meaning of Tolerance - 2

Navigation and Ancillary Information Facility

When reading a Type 1 CK containing discrete pointing instances

```
                          SCLKDP    TOL
                                \   /
                                 | |
                                 |/ \
Your request                  [---+---]
                              .   .   .
Available data   (0----0--0--0--0---0--0--0--0-----0--0--0)
                                 ↑
          A SPICELIB CK reader returns this pointing instance
```

- "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 or 5 CK (continuous pointing), with a "pointing request" that falls within a span of continuous pointing (an "interpolation interval")

```
                                  SCLKDP
                                       \  TOL
                                        | /
                                        |/\
Your request                          [--+--]
                                      .  .  .
Available data              (==----==============--======------===--)
                                        ↑
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 or 5 CK (continuous pointing), with a "pointing request" that is NOT within a span of continuous pointing (an "interpolation interval")

```
                                        SCLKDP
                                              \   TOL
                                               | /
                                               |/\
Your request                                [--+--]
                                            .  .  .
Available data                (--==============---=====------===--)
```

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

- "==" 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

- **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.

- **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

- **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 (outside of interpolation intervals)
- **One of the frames routines (SPKEZR, SPKPOS, SXFORM, PXFORM, SINCPT) signals an error**
  - All frames 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

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

Problems using CK - 4

- **The FOUND flag comes back "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 comes back "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 also prepared and can provide a "white paper" explaining differences between various quaternion styles commonly used in space applications.

Problems using CK - 5

- 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)
    - » Sun, HP, SGI and MAC (Motorola cpu) use the unix binary standard
    - » PC (Windows or Linux) uses its own 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

Navigation and Ancillary Information Facility

Frames Kernel
FK

January 2012

Introduction

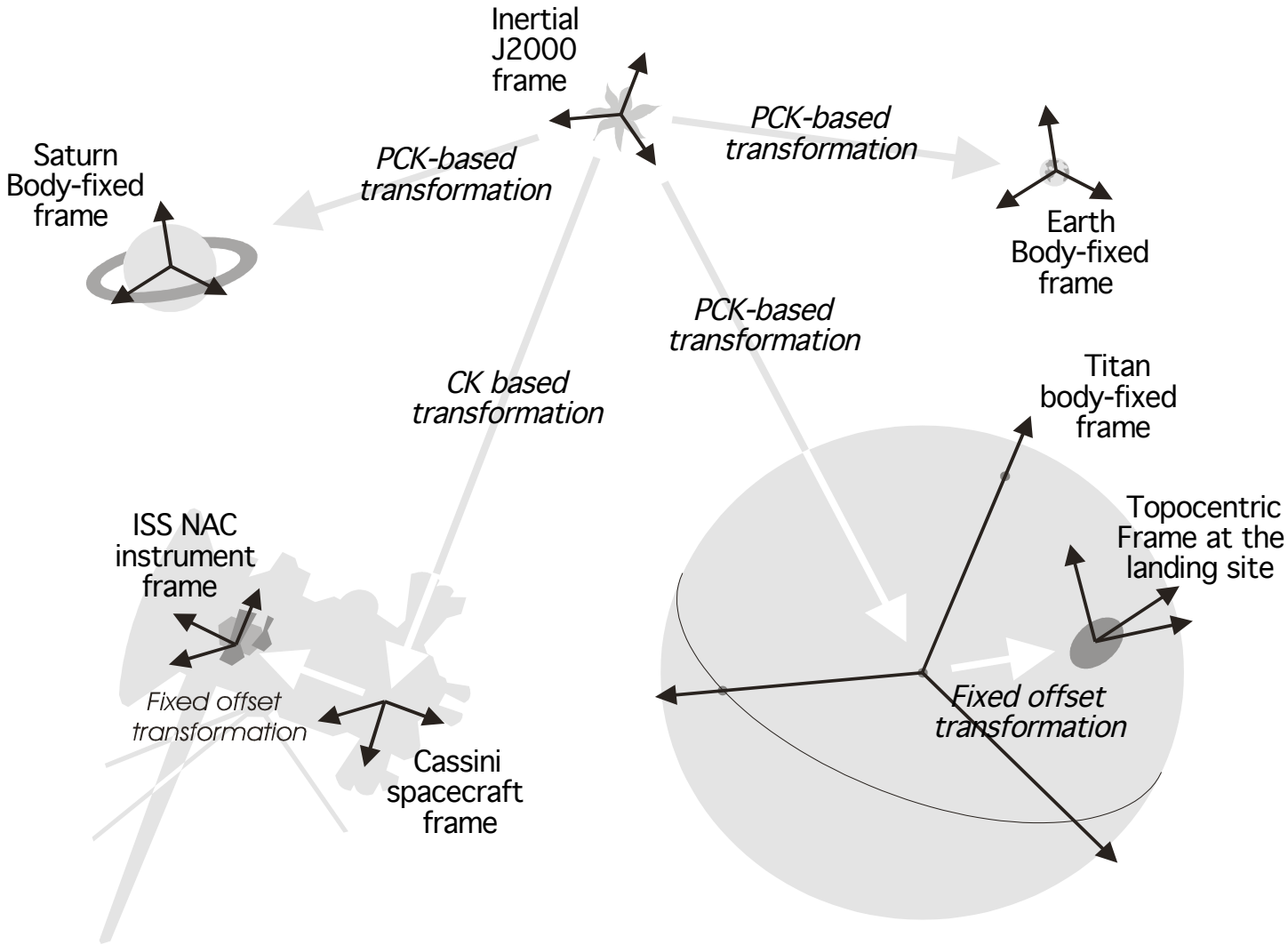
What does the FRAMES subsystem do?

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

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

Sample Frame Tree

Navigation and Ancillary Information Facility



Examples of Frames for Each Frame Class

Frame class	***Examples***
Inertial	• Earth Equator/Equinox of Epoch (J2000, …) • 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

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

Frames Subsystem Interfaces

Navigation and Ancillary Information Facility

SXFORM/PXFORM returns state or position transformation matrix

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

SPKEZR/SPKPOS returns state or position vector in specified frame

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

The above are FORTRAN examples, using SPICELIB modules.

The same interfaces exist for C, using CSPICE modules, and for Icy and Mice.

What are the Names of Frames?

- Refer to 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 project’s Frames Kernel (FK) file for a list of frames defined for the spacecraft, its subsystems and instruments
- Refer to an earth stations FK for a list of frames defined for the DSN and other stations
- Refer to the moon FKs for descriptions of the body-fixed frames defined for the moon

Frames Kernel File

Navigation and Ancillary Information Facility

- Uses the SPICE text kernel standards
- Loaded using the FURNSH 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 on 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:**
  - **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 <u>not</u> the same for the body-fixed frames defined in the Toolkit but they <u>can</u> 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

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 – 5) 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 a 1000.

Frame Tree Example: ASPERA Instrument on Mars Express

Navigation and Ancillary Information Facility

```
                                "J2000" <-inertial
                                   |
        +-------------------------------------------------------+
        |                          |                            |
        |<-pck                     |                            |<-pck
        |                          |                            |
        v                          |                            v
    "IAU_MARS"                     |                       "IAU_EARTH"
  MARS BODY-FIXED                  |<-ck                  EARTH BODY-FIXED
  ---------------                  |                     ----------------
                                   v
                             "MEX_SPACECRAFT"
        +-------------------------------------------------------+
        |                                                       |
        |<-fixed                                                |<-fixed
        |                                                       |
        v                                                       v
"MEX_ASPERA_URF"                                       "MEX_ASPERA_IMA_URF"
-----------------                                      --------------------
        |                                                       |
        |<-ck                                                   |<-fixed
        |                                                       |
        v                                                       v
"MEX_ASPERA_SAF"                                        "MEX_ASPERA_IMA"
---------------------------------------------- ...      ----------------
        |                 |                 |
        |<-fixed          |<-fixed          |<-fixed
        |                 |                 |
        v                 v                 v
"MEX_ASPERA_ELS"   "MEX_ASPERA_NPI"   "MEX_ASPERA_NPD1"
----------------   ----------------   ----------------
```

Blue text indicates frame class

FK Utility Programs

- **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

- 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 2012

What is the Power of Frames?

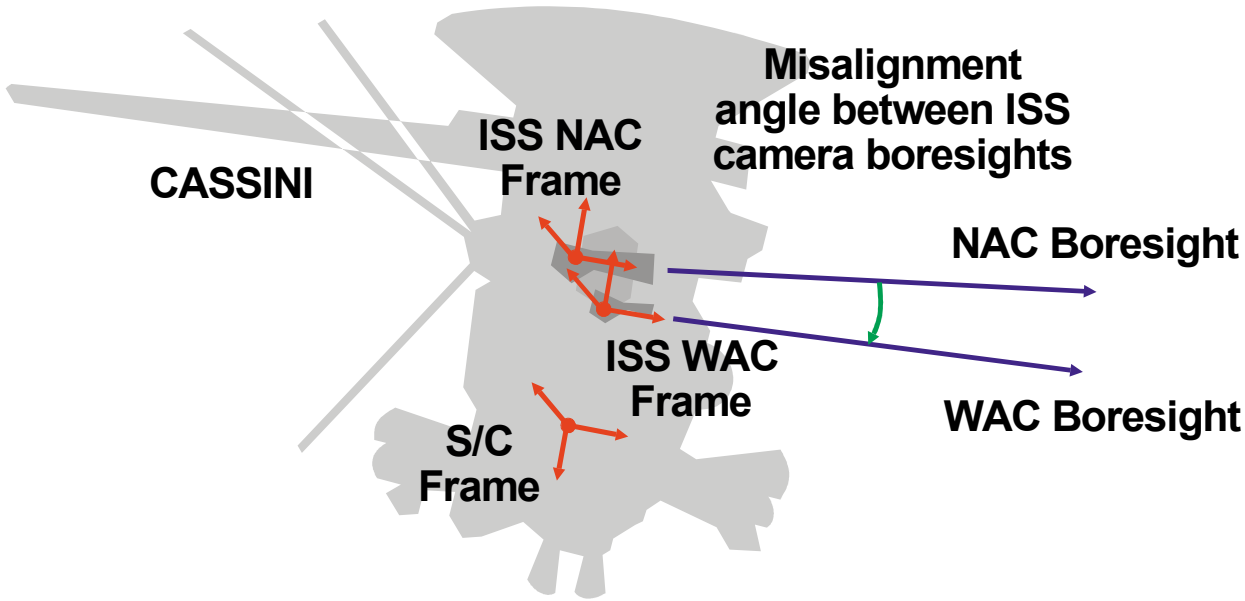
Navigation and Ancillary Information Facility

- 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 and PXFORM)
- The remaining pages illustrate typical use of frames
- Several VERY IMPORTANT usage issues are mentioned in the core Frames tutorial; be sure to also read that.

In SPICE terminology: "reference frame" ≠ "coordinate system"

Offset Between Instruments

Navigation and Ancillary Information Facility



Required Kernels:

- Generic LSK
- Mission FK
- Camera IK(s)

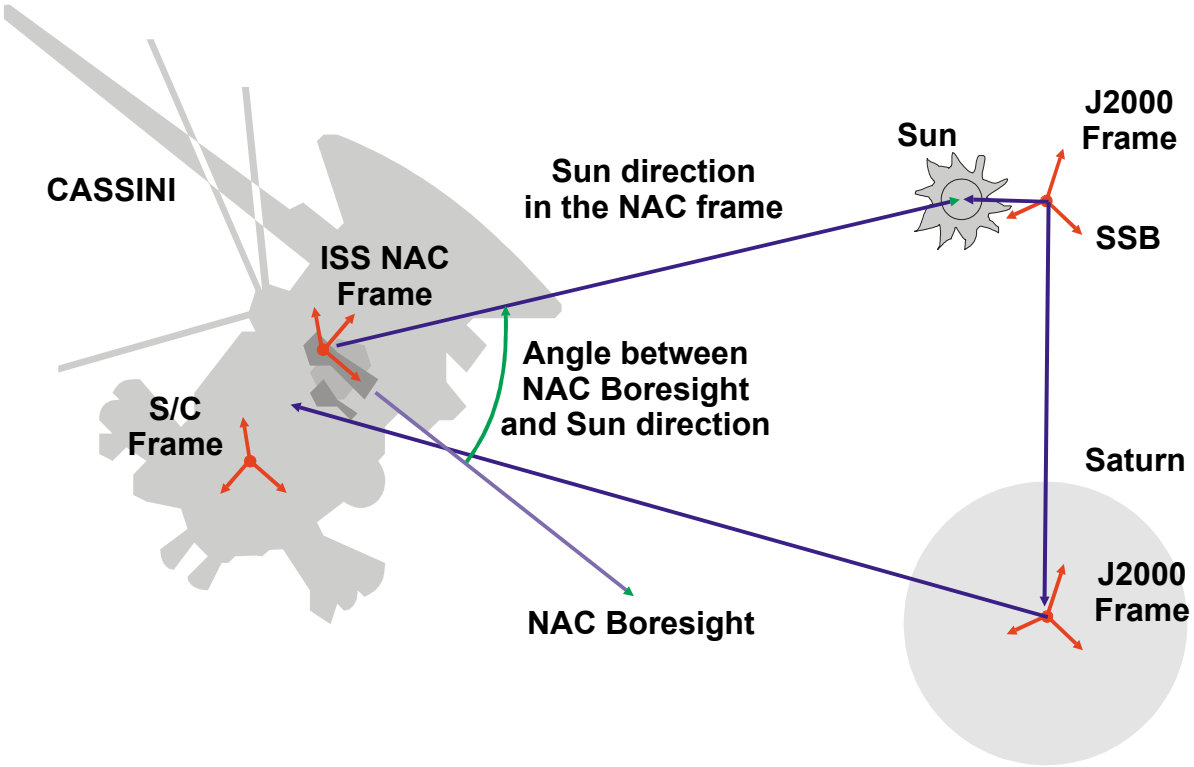
ISS = Imaging Science System

Compute the angular separation between the 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 )
```

Angular Constraints

Navigation and Ancillary Information Facility



Required Kernels:

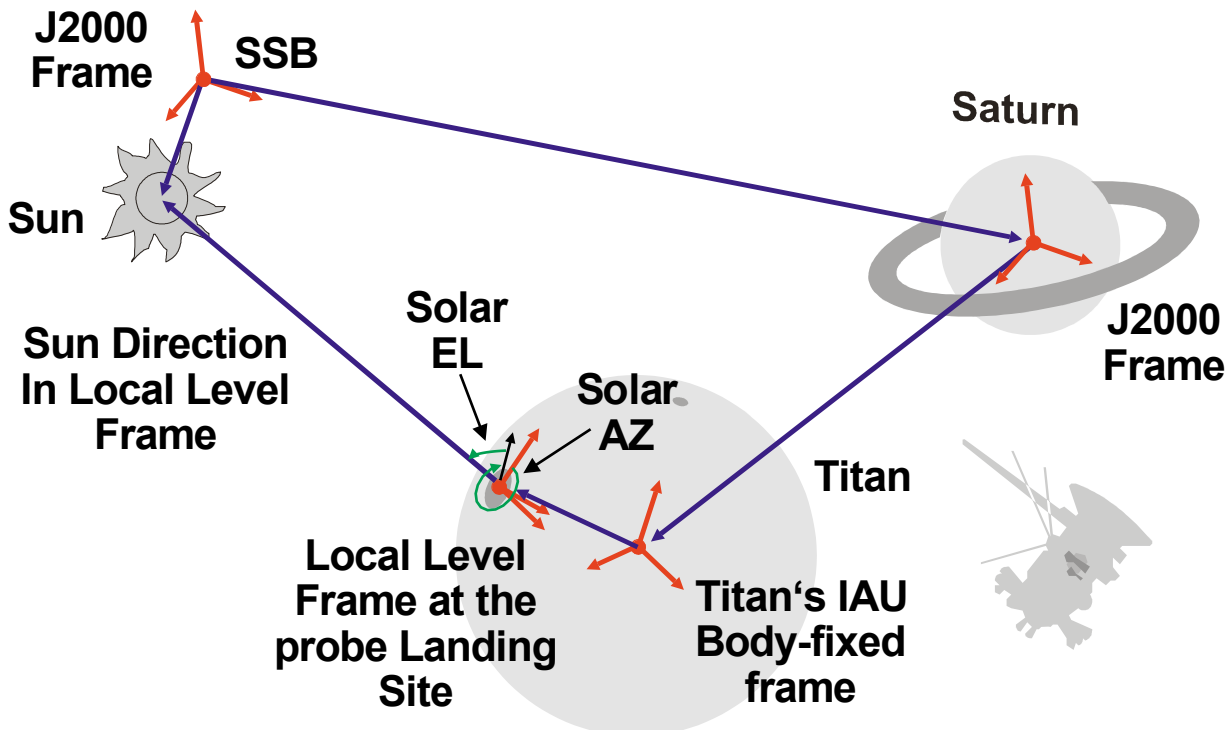
- Generic LSK
- Mission FK
- Spacecraft SCLK
- Camera IK
- Planetary Ephemeris SPK
- Spacecraft SPK
- Spacecraft CK

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!'
```

Angles at the Surface

Navigation and Ancillary Information Facility



Required Kernels:

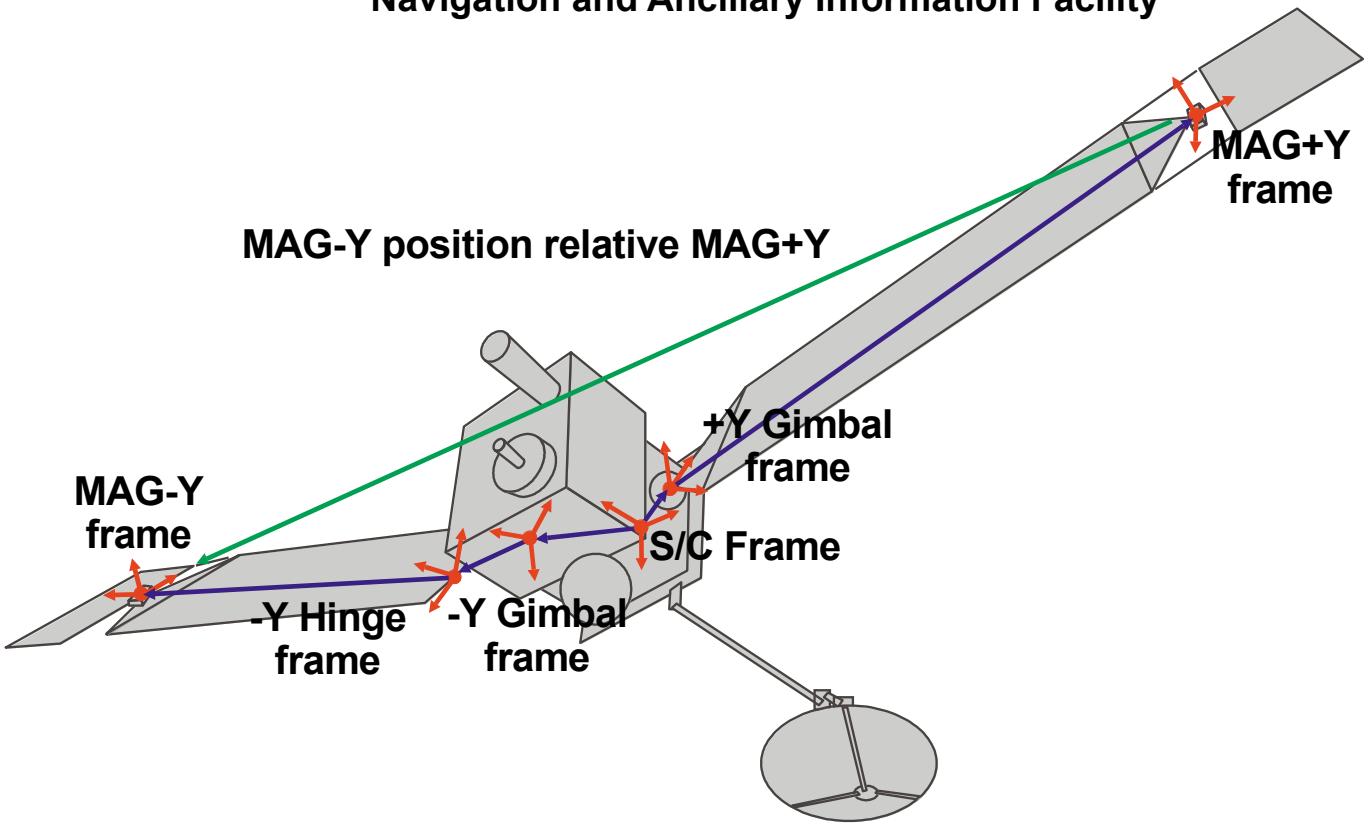
- Generic LSK
- Generic PCK
- Mission FK
- Planetary Ephemeris SPK
- Satellite Ephemeris SPK
- Landing Site SPK

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
```

Relative Position of Sensors

Navigation and Ancillary Information Facility



Required Kernels:

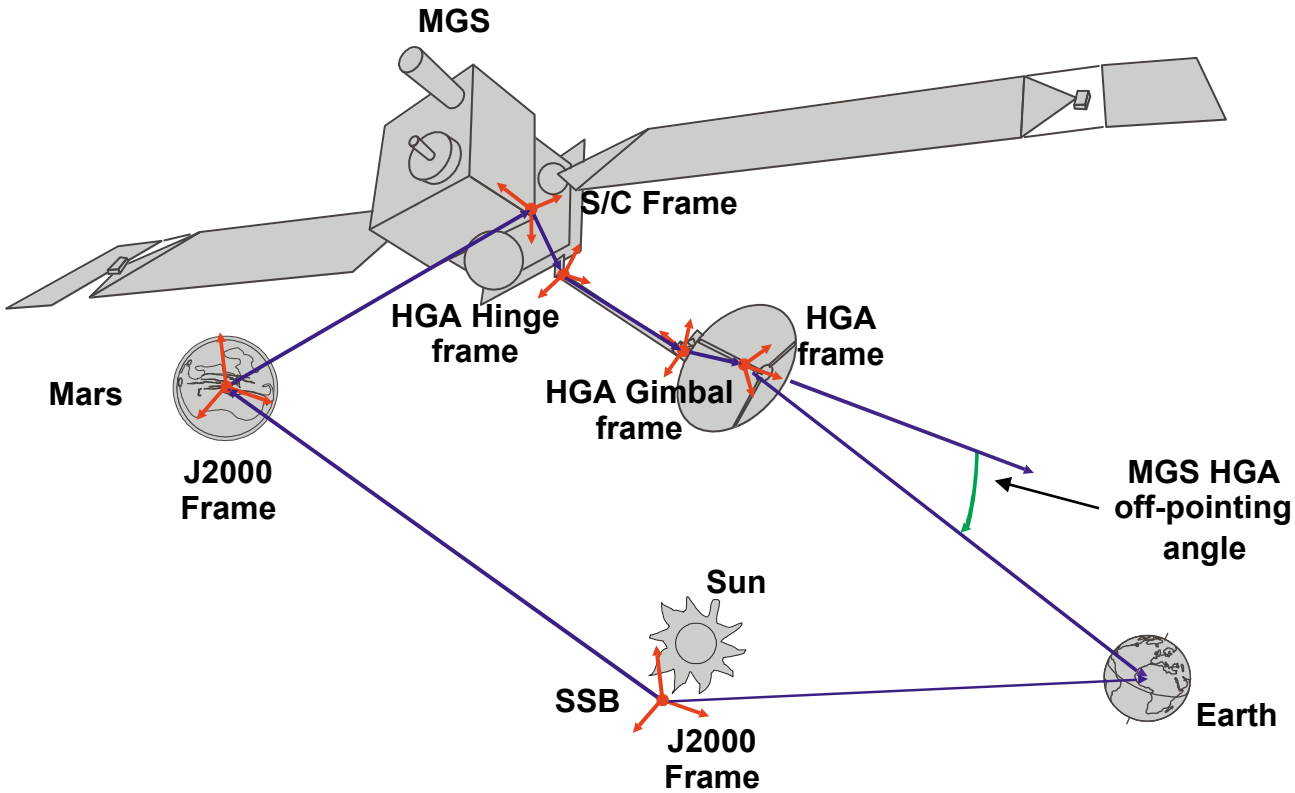
- Generic LSK
- Mission FK
- Structure Locations SPK
- Spacecraft SCLK
- Solar Array CK

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)
```

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

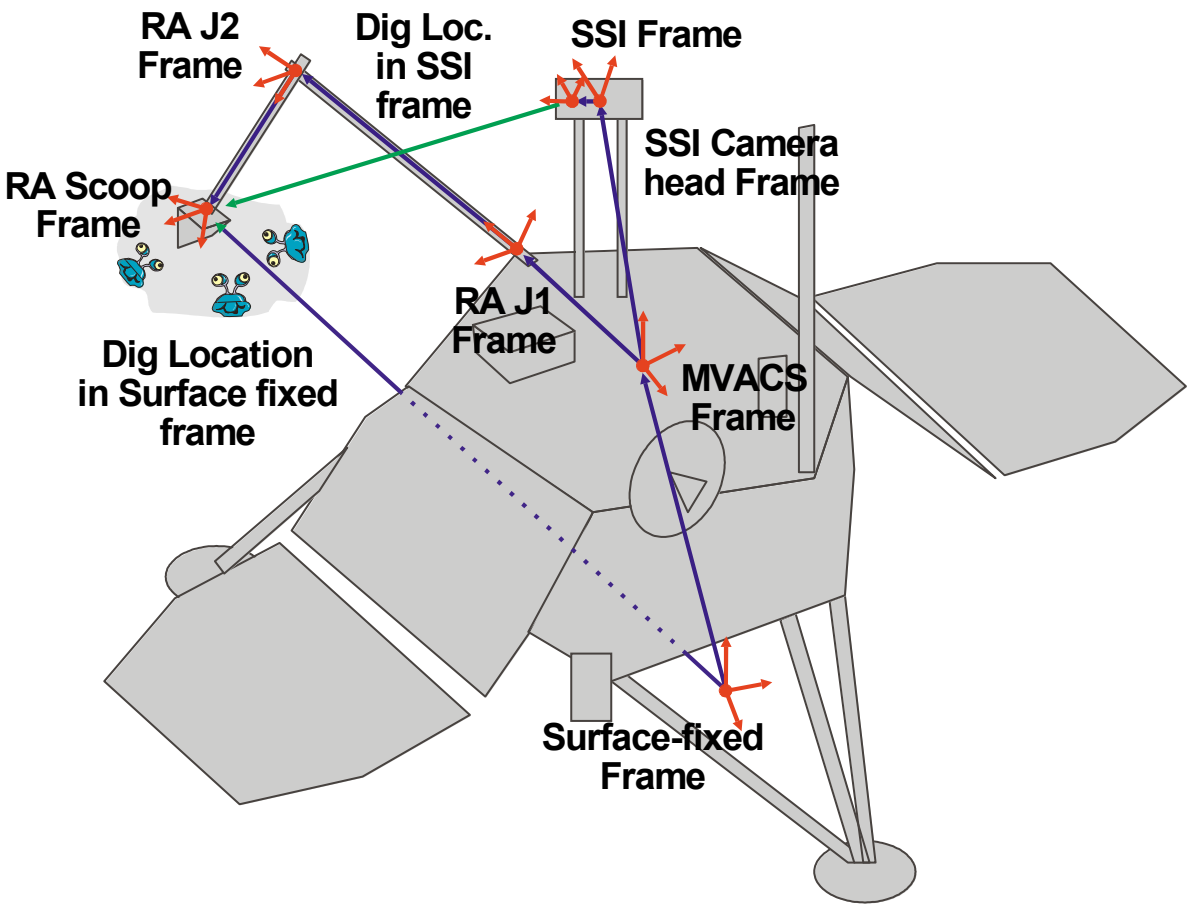
HGA = High Gain Antenna

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 )
```

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 )
```

Navigation and Ancillary Information Facility

Dynamic Reference Frames

January 2012

Topics

- Introduction to Dynamic Reference Frames
- Terminology
- Parameterized Dynamic Reference Frames
- Defining Dynamic Reference Frames
  - Two-Vector Frame Concepts
  - Two-Vector Frame Examples
  - "Of-Date" Frames
  - Euler Frames
  - Frozen Dynamic Frames
  - Inertial Dynamic Frames
- Generic Dynamic Reference Frame Kernel
- Backup

What are Dynamic Frames

- **What are dynamic reference frames?**
  - Dynamic reference frames ("dynamic frames" for short) have time-dependent orientation.
  - Dynamic frames are specified using a frames kernel (FK).
  - CK- and PCK-based frames are not considered to be dynamic frames although they are time-varying.
- **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

- 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
- RTN ("radial, tangential, normal”)

Using a Dynamic Frame

- **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.**
    - » **Load any kernels on which the dynamic frame depends.**
      - **Some dynamic frames are constructed using data using SPK, FK, PCK, CK, SCLK or other SPICE kernels.**
  - **Then, refer to the dynamic frame by name in calls to SPICE routines, just as you would do with built-in frames such as "J2000."**

Examples of Using Dynamic Frames

- **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 refer to dynamic frames in SPK or CK files, for example:**
  - **When you create an SPK file, you can have an SPK segment reference its ephemeris data to the true earth equator and equinox of date reference frame.**
    - » **However, some restrictions apply to use of dynamic frames in SPICE kernels (see Backup slides).**

Creating a Frames Kernel
Defining Dynamic Frames

- 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 for defining dynamic frames.
- 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 frame definition examples

Rationale

- **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.

Terminology - 1

- "Frame" is an abbreviation for "reference frame."
- A frame can be thought of 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 = axis 1; Y = axis 2; Z = axis 3
- All of the frames we'll deal with are "right-handed": this means +Z is the cross product +X x +Y.
- A reference frame's orientation is always defined relative to another specified frame: the "base frame."

Terminology - 2

- 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.
  - 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

- 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 frame kernel.
- The term "API" stands for "Application Programming Interface." This term refers to the set of SPICE routines that are intended to be called directly by SPICE-based programs.
- The notation

$$[\mathtt{theta}]_{\mathtt{n}}$$

indicates a frame rotation of theta radians about axis n, where n is one of {1, 2, 3}. This transformation rotates vectors by –theta radians about axis n.

Parameterized Dynamic Frames

- This is the only frame definition style currently supported by the dynamic frames subsystem.
  - Future versions of SPICE might support additional styles such as script-based dynamic frames
- Frames are defined via parameterized formulas
  - The code implementing the formulas is built into SPICE.
  - The parameters 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
    - » Mean Equator and Equinox
    - » True Equator and Equinox
    - » Mean Ecliptic and Equinox
  - Euler Frames

Defining Parameterized Dynamic Frames

- Parameterized dynamic frames are defined using "keyword=value" assignments in a frames kernel.
- 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
    - » Frame definition specifies mapping from dynamic frame to the base frame.
  - Frame family
  - Family-specific assignments

continued on next page

Defining Parameterized Dynamic Frames

continued from previous page

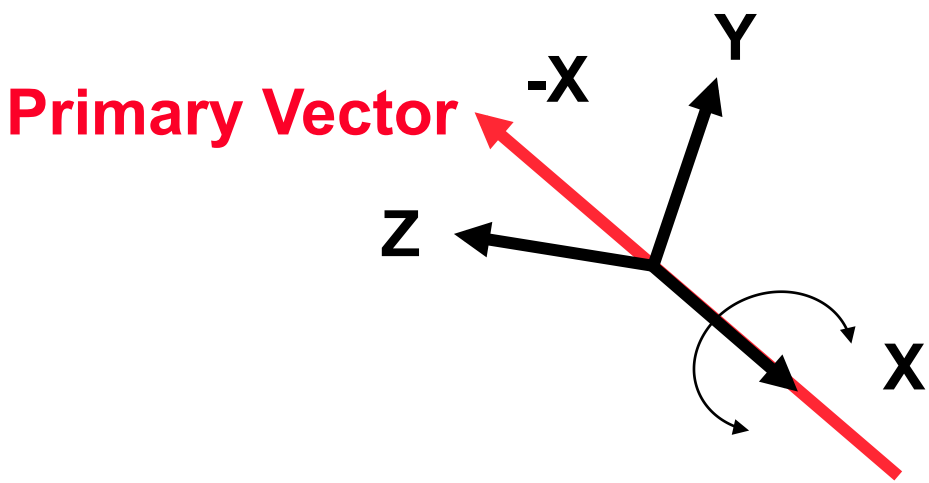
- Rotation state
  - » Possible states are 'ROTATING' and 'INERTIAL'.
    - A frame is treated as rotating or inertial for the purpose of velocity transformations.
  - » The default dynamic frame rotation state is 'ROTATING'.
  - » For rotating two-vector and Euler frames, the rotation state assignment 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.
- Freeze epoch
  - » 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.

Two-Vector Frame Concepts - 1

- **Two-vector frames are defined using two time-dependent vectors: the "primary" and "secondary" vectors.**
  - Each 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

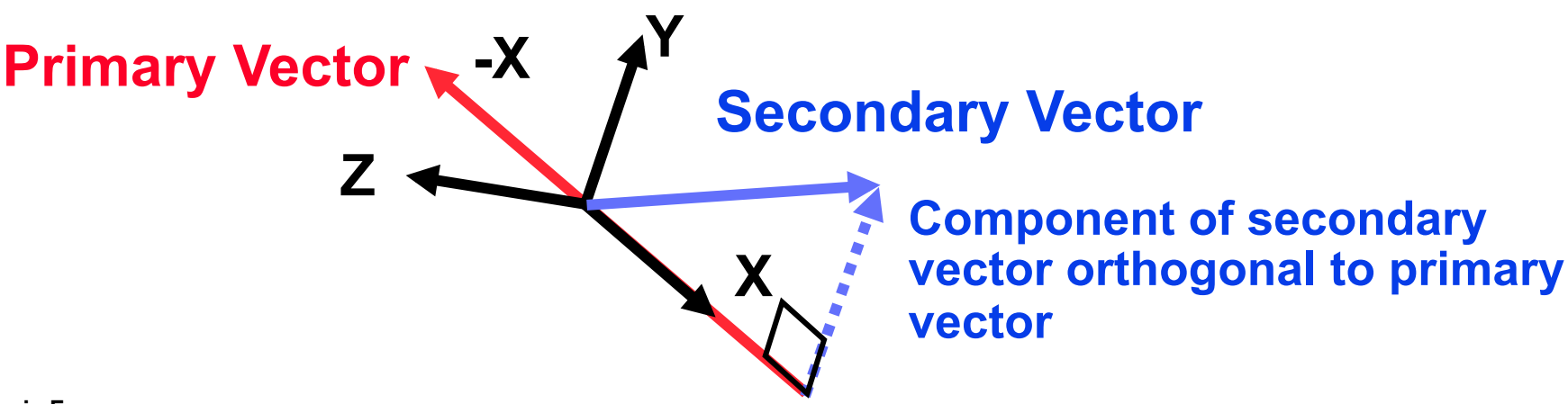
- **Primary Vector**
  - A specified positive or negative axis of the two-vector frame is aligned with this vector.
    - » The frame kernel creator assigns to this vector one of the axis designations { +X, -X, +Y, -Y, +Z, -Z }.
  - Two degrees of freedom of the frame orientation are removed by association 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:



Two-Vector Frame Concepts - 3

Navigation and Ancillary Information Facility

- **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

- **Secondary Vector, continued**
  - Typically the secondary vector itself is not orthogonal to the primary vector.
  - 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

- **Position Vector**
  - Is 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 routine:
    - » 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 - 6

- **Target Near Point Vector**
  - Is 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, as for position vectors.
  - 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

- **Velocity Vector**
  - Is 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, as for position vectors.

Two-Vector Frame Concepts - 8

- **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 - 9

» 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.

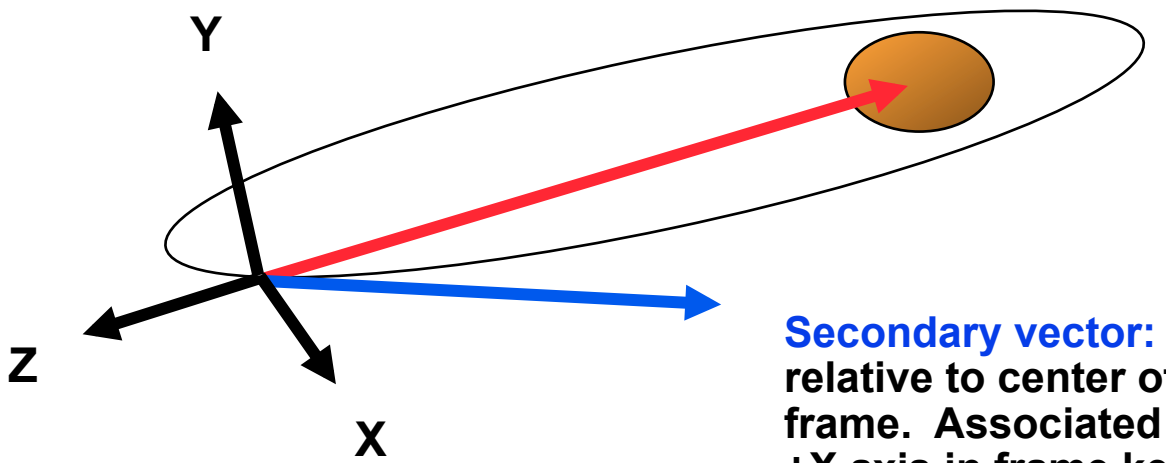
Two-Vector Frame Examples - 1

Navigation and Ancillary Information Facility

Nadir-Oriented Spacecraft-Centered Frame

Y = Z x X, completing the right-handed frame.

Primary vector: spacecraft nadir direction vector. Associated with nadir frame's -Z axis in frame kernel.



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 center of motion in J2000 frame. Associated with nadir frame's +X axis in frame kernel.

Normalized component of secondary vector orthogonal to primary vector. This vector is aligned with the nadir frame's +X axis.

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 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

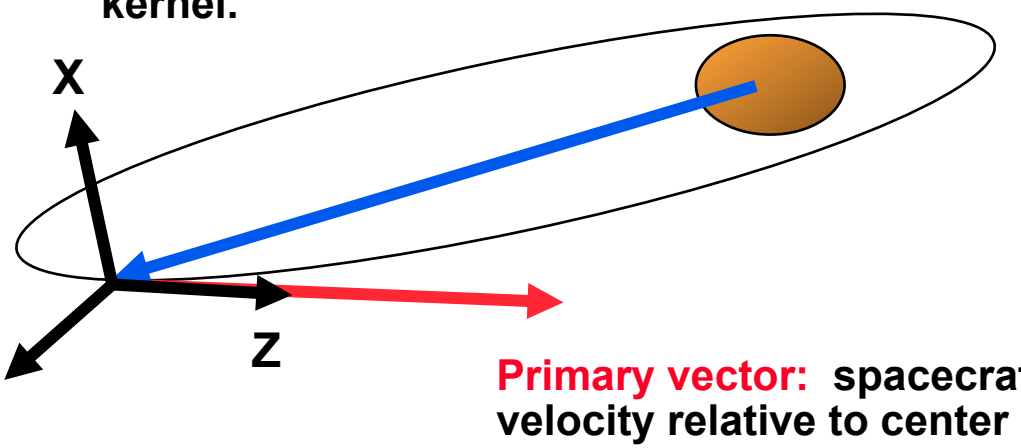
Two-Vector Frame Examples - 3

Navigation and Ancillary Information Facility

Spacecraft "View Frame"

Secondary vector:
spacecraft position relative to center of motion. Associated with view frame's +Y axis in frame kernel.

X = Y x Z, completing the right-handed frame. ("Out of plane" direction)



Primary vector: spacecraft velocity relative to center of motion in J2000 frame. Associated with view frame's +Z axis in frame kernel. ("Down track" direction)

Normalized component of secondary vector orthogonal to primary vector. This vector is aligned with the view frame's +Y axis. ("In plane" direction)

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 +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 = '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'
```

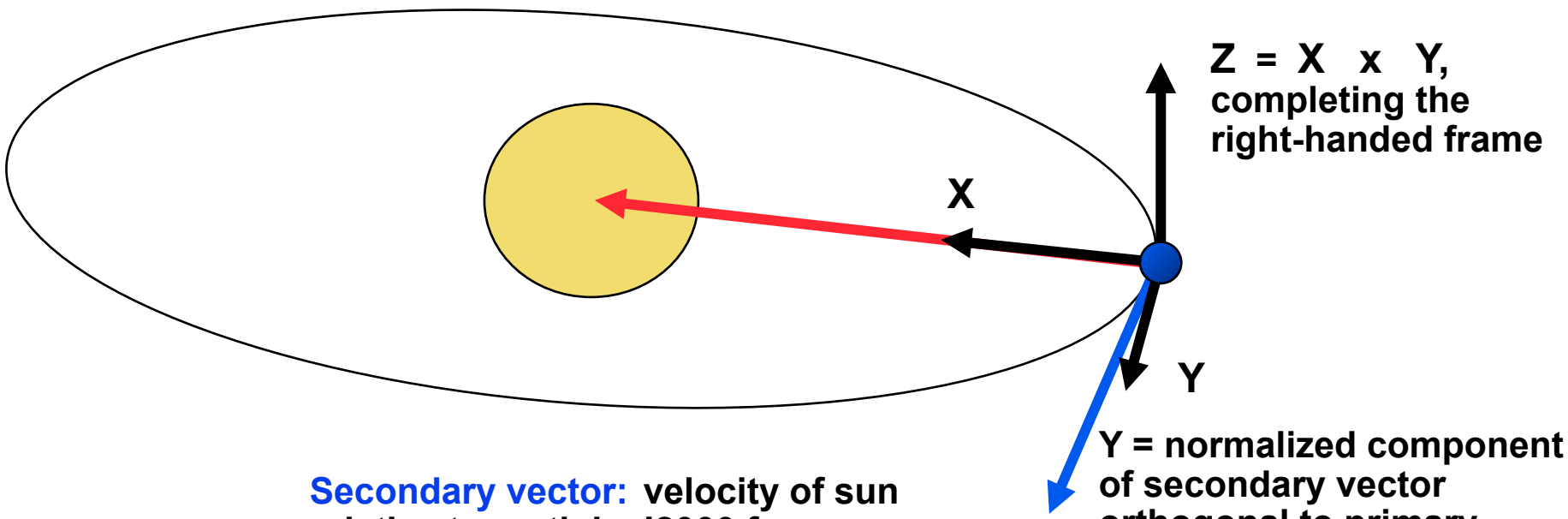
Definitions

<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

Geocentric Solar Ecliptic Frame (GSE)

Primary vector: position of sun relative to earth
Associated with GSE frame's +X axis in frame kernel.



Secondary vector: velocity of sun relative to earth in J2000 frame.
Associated with GSE frame's +Y axis in frame kernel.

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

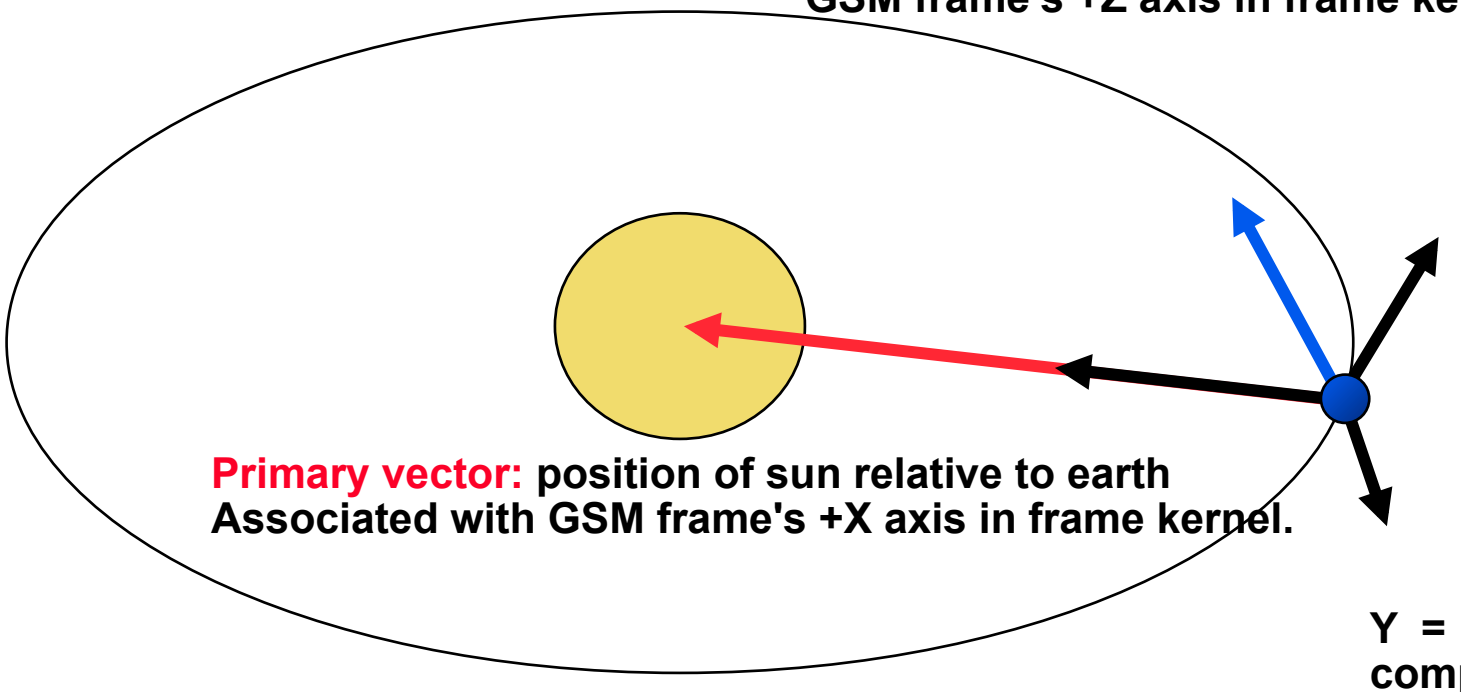
<frame_ID> = integer frame ID code

Two-Vector Frame Examples - 7

Navigation and Ancillary Information Facility

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.



Z = normalized component of secondary vector orthogonal to primary vector

Primary vector: position of sun relative to earth Associated with GSM frame's +X axis in frame kernel.

Y = Z x X, completing the right-handed 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
```

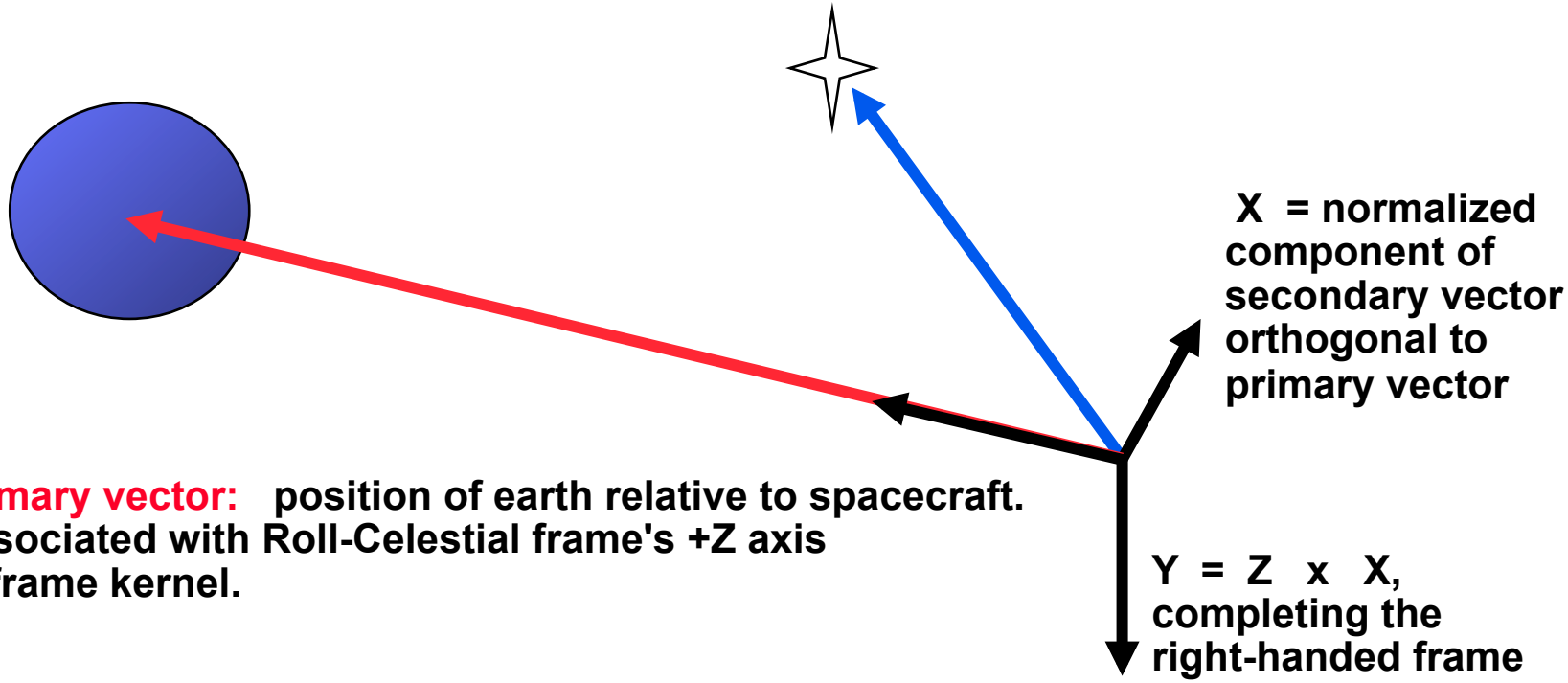
Definition

<frame_ID> = integer frame ID code

Two-Vector Frame Examples - 9

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. Associated with Roll-Celestial frame's +Z axis in frame kernel.

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

- 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 frames.

"Of-Date" Frames - 2

- 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
  - The set of supported frame families can grow, if necessary.

"Of-Date" Frames - 3

- **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
      - 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."

"Of-Date" Frames - 4

Navigation and Ancillary Information Facility

```
Earth mean equator and equinox of date frame:

     +Z axis is perpendicular to 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_ID>
   FRAME_<frame_ID>_NAME               =  <frame_name>
   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             = 'MEAN_EQUATOR_AND_EQUINOX_OF_DATE'
   FRAME_<frame_ID>_PREC_MODEL         = 'EARTH_IAU_1976'
   FRAME_<frame_ID>_ROTATION_STATE     = 'ROTATING'
```

Definitions

<frame_ID>	= integer frame ID code
<frame_name>	= user-specified frame name

"Of-Date" Frames - 5

- 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
      - 1976 IAU precession model (Lieske model)
      - 1980 IAU nutation model
    - » 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 - 6

Navigation and Ancillary Information Facility

```
Earth true equator and equinox of date frame:

     +Z axis is perpendicular to 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 mean ecliptic of date.

     +Y axis 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             =  399
   FRAME_<frame_ID>_RELATIVE           = 'J2000'
   FRAME_<frame_ID>_DEF_STYLE          = 'PARAMETERIZED'
   FRAME_<frame_ID>_FAMILY             = 'TRUE_EQUATOR_AND_EQUINOX_OF_DATE'
   FRAME_<frame_ID>_PREC_MODEL         = 'EARTH_IAU_1976'
   FRAME_<frame_ID>_NUT_MODEL          = 'EARTH_IAU_1980'
   FRAME_<frame_ID>_ROTATION_STATE     = 'ROTATING'
```

Definitions

<frame_ID>	= integer frame ID code
<frame_name>	= user-specified frame name

"Of-Date" Frames - 7

- **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
      - 1976 IAU precession model (Lieske model)
      - 1980 IAU mean obliquity model
    - » 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.

\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             =  399
   FRAME_<frame_ID>_RELATIVE           = 'J2000'
   FRAME_<frame_ID>_DEF_STYLE          = 'PARAMETERIZED'
   FRAME_<frame_ID>_FAMILY             = 'MEAN_ECLIPTIC_AND_EQUINOX_OF_DATE'
   FRAME_<frame_ID>_PREC_MODEL         = 'EARTH_IAU_1976'
   FRAME_<frame_ID>_OBLIQ_MODEL        = 'EARTH_IAU_1980'
   FRAME_<frame_ID>_ROTATION_STATE     = 'ROTATING'
```

Definitions

<frame_ID>	= integer frame ID code
<frame_name>	= user-specified frame name

Euler Frames - 1

- **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. Angles are converted to radians internally by SPICE.
    - » 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

      $[angle_1]_{axis_1}\ [angle_2]_{axis_2}\ [angle_3]_{axis_3}$ (units are radians)

Euler Frames - 2

- **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

- Mean or true body equator and earth 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.

```
       angle_3 is  90 + RA                      angle_1 is -90 - RA
 PCK:  angle_2 is  90 - Dec       Euler Frame:  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

- **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 given in SPICE text kernel format.

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 mean equator of date.

     +X axis is parallel to cross product of +Z axis and
        vector normal to mean ecliptic of date.

     +Y axis 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             =  399
   FRAME_<frame_ID>_RELATIVE           = 'J2000'
   FRAME_<frame_ID>_DEF_STYLE          = 'PARAMETERIZED'
   FRAME_<frame_ID>_FAMILY             = 'MEAN_EQUATOR_AND_EQUINOX_OF_DATE'
   FRAME_<frame_ID>_PREC_MODEL         = 'EARTH_IAU_1976'
   FRAME_<frame_ID>_FREEZE_EPOCH       = @1949-DEC-31/22:09:46.861901
```

Definitions

<frame_ID>	= integer frame ID code
<frame_name>	= user-specified frame name

Inertial Dynamic Frames - 1

- **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 zero derivative block: the state transformation matrix has the form**

```
        R(t)   |    0
      --------|------
Derivative
block   0      |  R(t)
```

where R(t) is a time-dependent rotation.

Inertial Dynamic Frames - 2

– 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

```
      R(t)     |    0
     -------|------
   dR(t)/dt|   R(t)
```

– 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 true equator of date.

     +X axis is parallel to cross product of +Z axis and
        vector normal to mean ecliptic of date.

     +Y axis 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             =  399
   FRAME_<frame_ID>_RELATIVE           = 'J2000'
   FRAME_<frame_ID>_DEF_STYLE          = 'PARAMETERIZED'
   FRAME_<frame_ID>_FAMILY             = 'TRUE_EQUATOR_AND_EQUINOX_OF_DATE'
   FRAME_<frame_ID>_PREC_MODEL         = 'EARTH_IAU_1976'
   FRAME_<frame_ID>_NUT_MODEL          = 'EARTH_IAU_1980'
   FRAME_<frame_ID>_ROTATION_STATE     = 'INERTIAL'
```

Definitions

<frame_ID>	= integer frame ID code
<frame_name>	= user-specified frame name

Generic Dynamic Frames Kernel

- NAIF may develop a "generic" dynamic frames kernel.
  - Would contain widely applicable dynamic frame definitions.
  - Analogous to generic PCK file.
  - Examples of possible included dynamic frames:
    - » GSE, GSM, MAG
    - » Earth mean equator and equinox of date, 1976 version
    - » Earth true equator and equinox of date, 1980 version

Backup

- **Numerical Issues**
- **Limitations**

Numerical Issues - 1

- **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

- **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

- **Simulated recursion:**
  - ANSI Fortran 77 doesn't support recursion, so the SPICE dynamic frame system implements limited, simulated recursion.
    - » Basically, 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

- **Run-time efficiency:**
  - Dynamic frame evaluation typically requires more computation than is needed for CK or PCK 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

“High Accuracy” Orientation and Body-fixed Frames for the Moon and Earth

January 2012

Topics

- 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

- 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 Moon or the Earth.

Introduction-2

- **NAIF provides "High accuracy" orientation data for the Earth and 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 each time an official new JPL "Developmental Ephemeris" (DE) is released.
    - » 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

Earth Binary PCKs

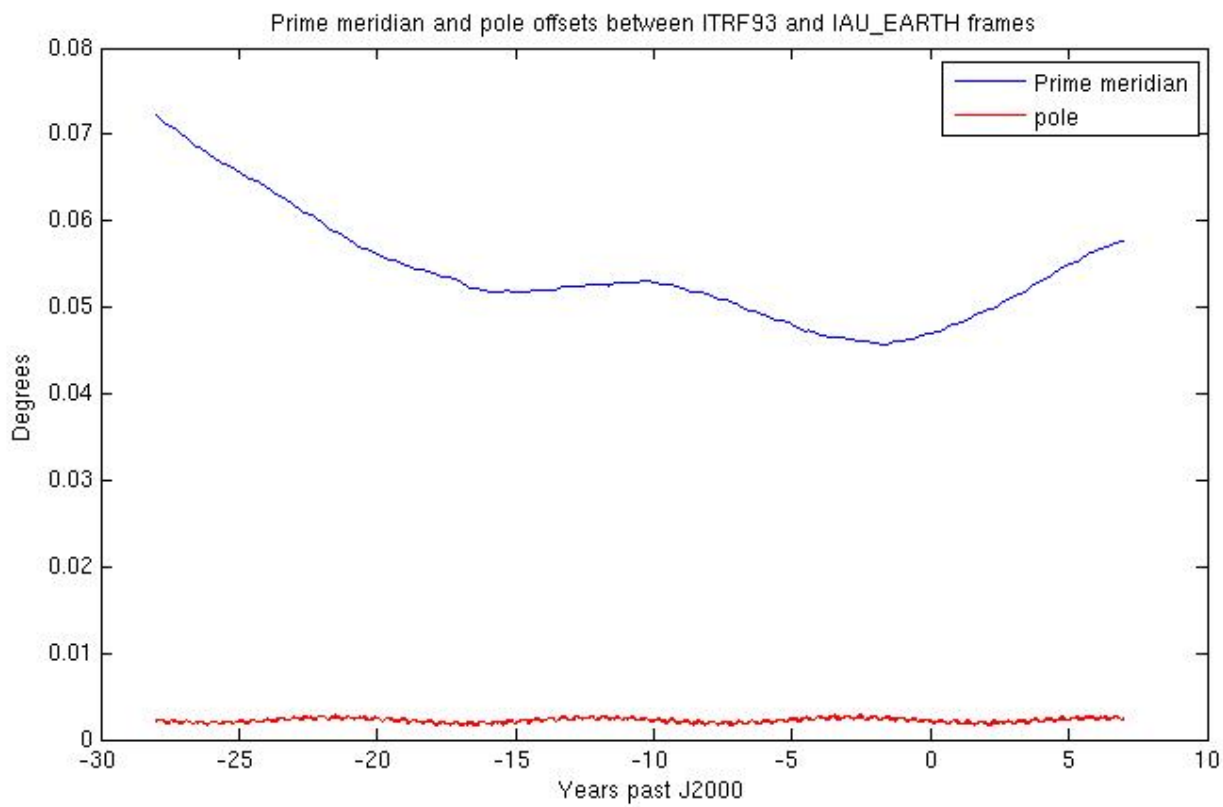
“High Accuracy” Earth Rotation Model

- **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 ~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

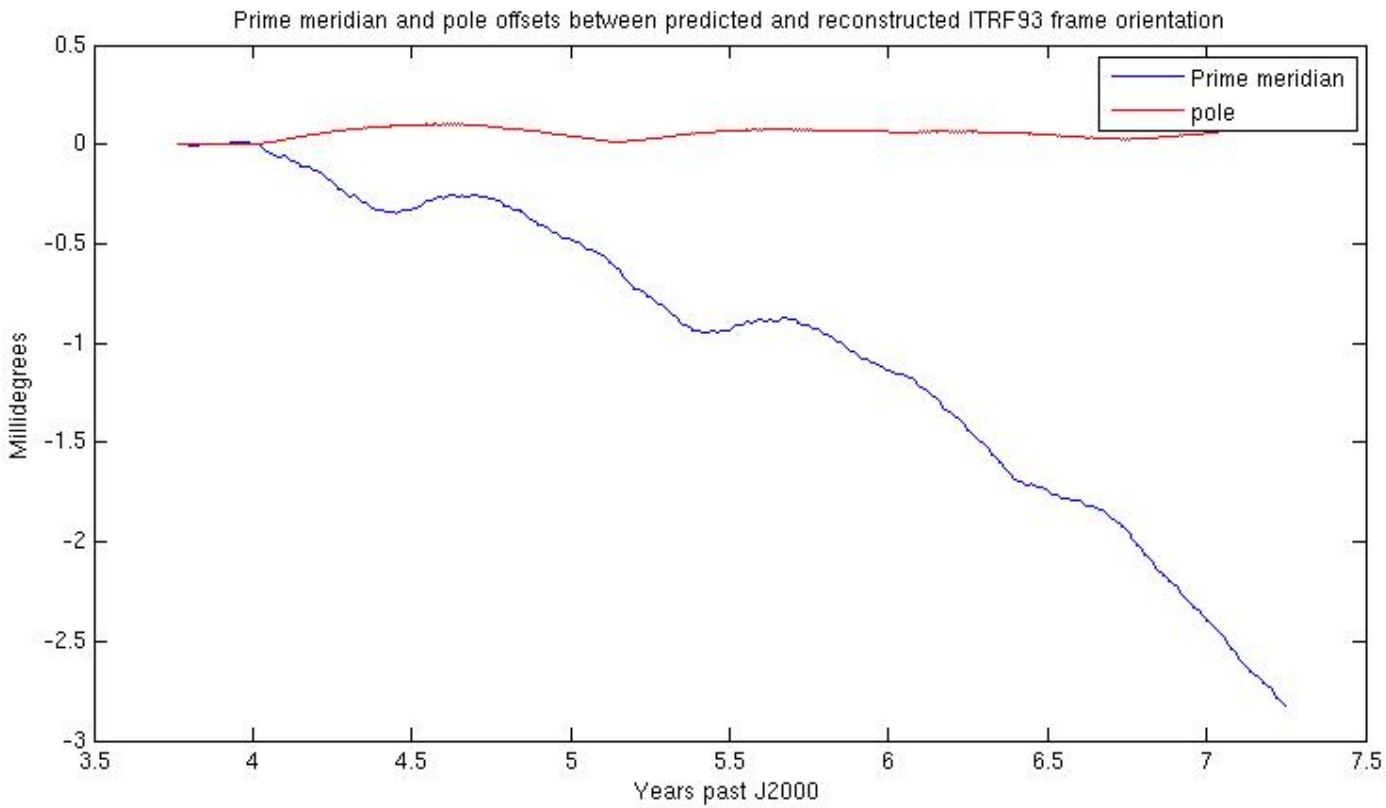
Difference between the 'IAU_EARTH' frame and the 'ITRF93' frame



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

- **Data for the earth come from a JPL Earth Orientation Parameters file (EOP).**
  - **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.**

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
IERS = International Earth Rotation Service

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 automatic script**
  - **Long term predict, for future uses not requiring high accuracy**
    - » **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:** ftp://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

- 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.
  - The other three non-rotational effects affecting surface locations are not accounted for by any PCK, or by any other SPICE component. But magnitude is under one meter.

Kernel Usage Summary: Earth

- **To use high accuracy Earth orientation data**
  - **Load one or more binary Earth PCKs**
    - » **If a long-term predict is used, load this 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**

Lunar Binary PCKs

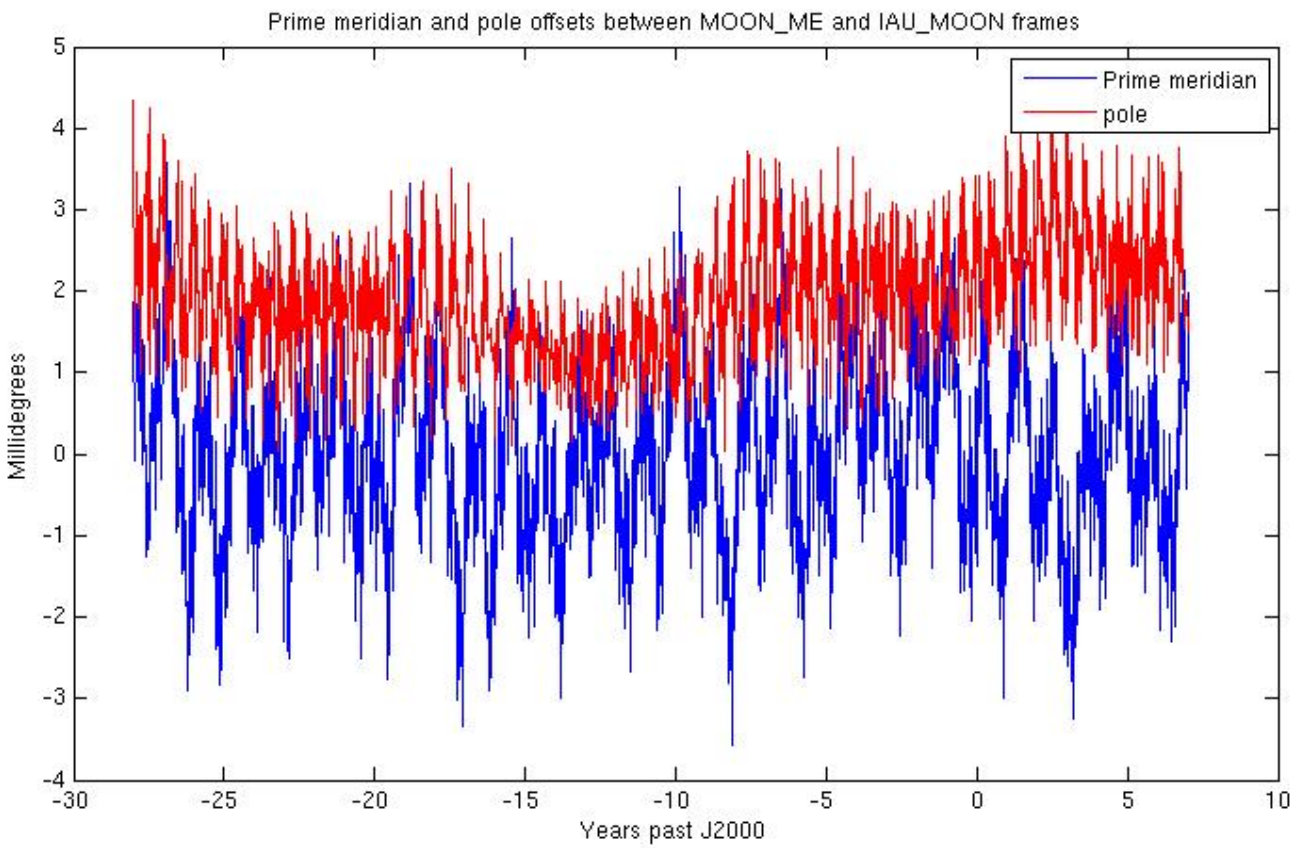
"High Accuracy" Lunar Rotation Model

- **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').**
  - Rotation error between IAU_MOON and the corresponding "high accuracy" MOON_ME (mean Earth/rotation axis) frame for the DE-421 and 2000 IAU data sets and for the time period of 2000-2020 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)



Lunar Rotation Model Effects

- **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-A)/B 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)

Data Sources for “High Accuracy” Models

- **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 Frames Kernel

Lunar Frames Kernel

- **A lunar frames kernel is maintained and available from NAIF. It has four functions.**
  1. Make two lunar frames–Principal Axes (PA) and Mean Earth/Polar 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 (relative to the ICRF).
  3. Provide specifications, in the SPICE context, 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.
- **To access the PA or ME frame you must load the lunar FK into your program in addition to the lunar binary PCK that implements the lunar PA frame orientation.**

Kernel Usage Summary: Moon

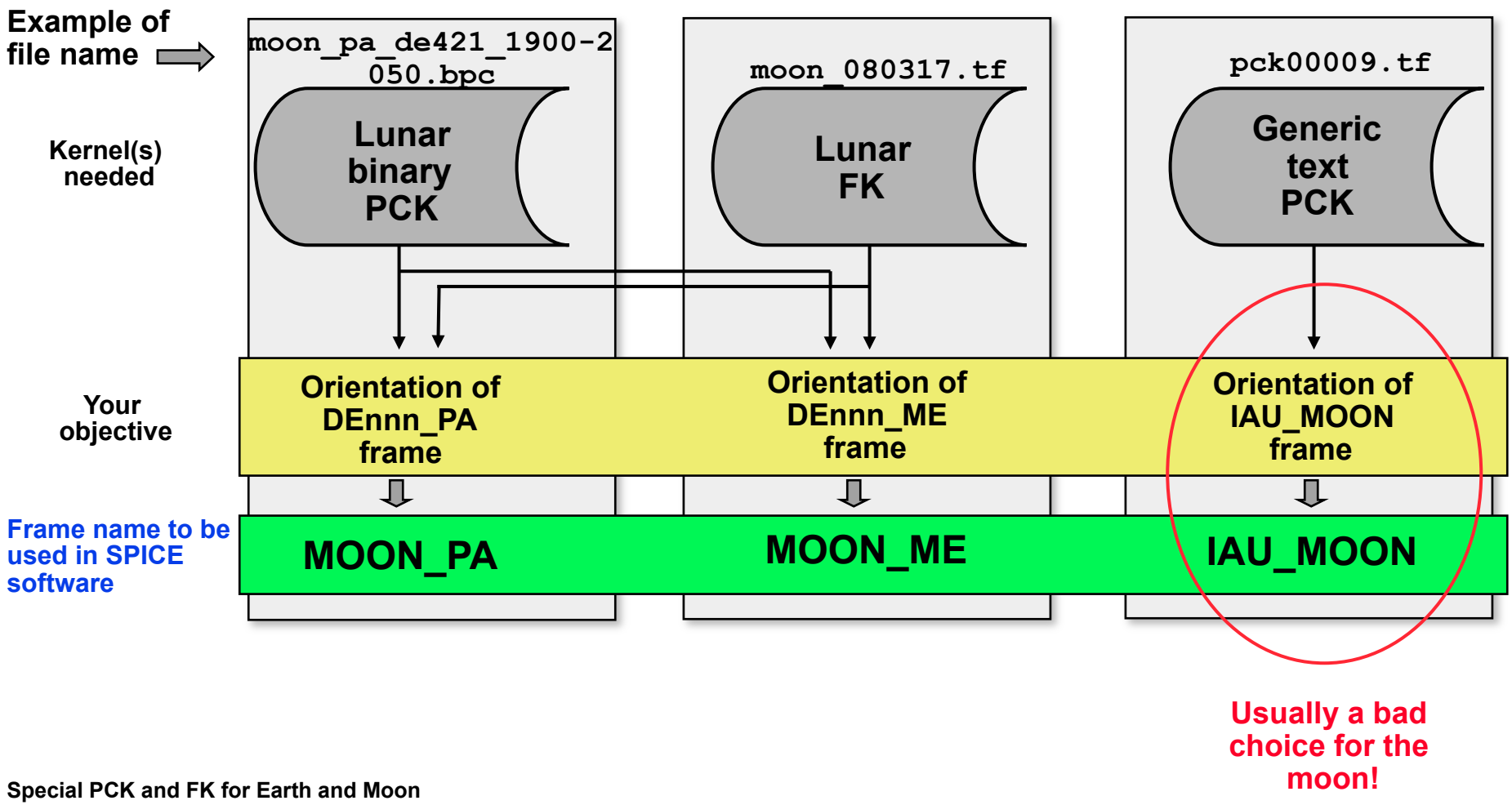
- **To use high accuracy Moon orientation data**
  - Load the current binary lunar PCK
  - Load the current lunar FK
  - If your application uses any of the old, pre-N0062 APIs that make use of the <u>default</u> lunar body-fixed frame (see Backup), load a moon frame association kernel making either 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 pck00009.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?



Binary PCK File Format

Binary PCK File Format

- 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.
      - Not currently used for Earth or 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
    - » Cheby polynomials are fit to source Euler angles; these evaluate very quickly

Using Binary PCKs

Precedence Rules for Text and Binary PCKs

- 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. pck00009.tpc), so there will be no conflict.

Tools for use with Binary PCKs

- 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
  - If you need to write to a non-native binary PCK you must first convert it to native binary form using *bingo* or the pair of *toxfr* and *tobin*
    - » *toxfr* and *tobin* are avaiable in each Toolkit; bingo is available only from the NAIF website
  - Converting a non-native binary PCK to native form will also speed up data access somewhat

Backup

Association Frames Kernels for the Earth and the Moon

Association FKs: Introduction

- **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 <u>default</u> body-fixed frame**
- **This was a bad assumption… at least for the Earth and the Moon!**
  - **Multiple body-fixed frames exist for the Moon and Earth**
  - **The <u>default</u> body-fixed frames for the Moon and the Earth, for which the defining data are provided in a generic text PCK (taken from an IAU report) are very inaccurate representations of the actual orientations of these bodies**

Better Choice for the Default

- **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**

Body	SPICE Default Body-fixed Frame	Better choice
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**

Old: still available, but better to use those noted below

- **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 <u>default</u> 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**

New: safer to use, and offer improved accuracy in some cases

Changing the Default Body-Fixed Frame Name

- **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 <u>default</u> 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.**

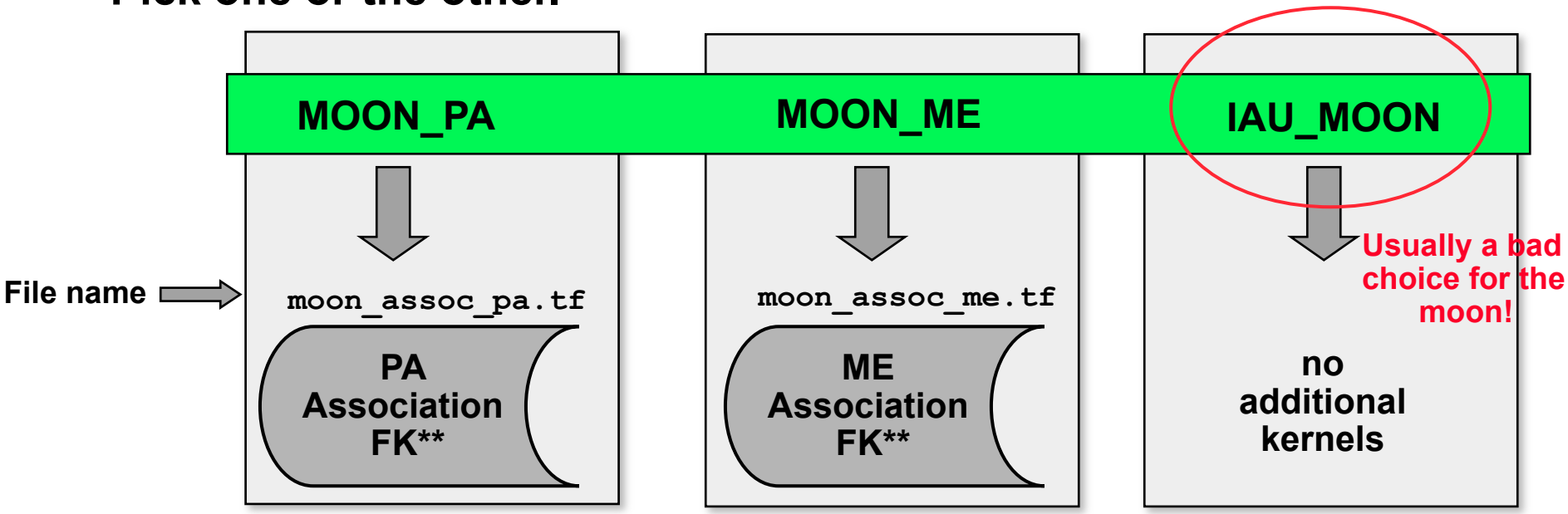
Using Association FKs to Change the Default

- **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:**
    - » **ftp://naif.jpl.nasa.gov/pub/naif/generic_kernels/fk/planets/**
  - **For the Moon:**
    - » **ftp://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)

But best to use the replacements for these four, which don't use a default body-fixed frame:
- ILUMIN
- SINCPT
- SUBPNT
- SUBSLR

**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.

Navigation and Ancillary Information Facility

Instrument Kernel
IK

January 2012

Purpose

- **The Instrument Kernel serves as a repository for instrument specific information that may be useful within the SPICE context.**
  - **Always included:**
    - » **Specifications for an instrument's field-of-view (FOV) size, shape, and orientation**
  - **Other possibilities:**
    - » **Internal instrument timing parameters and other data relating to SPICE computations might also be placed in an I-kernel**
    - » **Instrument geometric calibration data**

- **Note: instrument mounting alignment data are specified in a mission's Frames Kernel (FK)**
  - **(Wasn't true for some of the earliest missions that used SPICE)**

I-Kernel Structure

- **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)

- **Examples of IK keywords, with descriptions:**
  - INS-94031_FOCAL_LENGTH — MGS MOC NA focal length
  - INS-41220_IFOV — 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)

- **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

- **This documentation exists 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 )
CALL GIPOOL ( NAME, START, ROOM, N, VALUES, FOUND )
CALL GCPOOL ( NAME, START, ROOM, N, VALUES, FOUND )
```

for DP values
for integer values
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)

- **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 defining reference frame in which the boresight vector and FOV boundary vectors are specified**

```
INS#_FOV_FRAME      = 'frame name'
```

  - **Keyword defining the boresight vector**

```
INS#_BORESIGHT      = ( X, Y, Z )
```

continued on next page

FOV Definition Keywords (2)

– 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)

2) By providing half angular extents of the FOV (possible only for circular, elliptical or rectangular FOVs)

```
INS#_FOV_CLASS_SPEC        = 'ANGLES'
INS#_FOV_REF_VECTOR        = ( X, Y, Z )
INS#_FOV_REF_ANGLE         = halfangle1
INS#_FOV_CROSS_ANGLE       = halfangle2
INS#_FOV_ANGLE_UNITS       = '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.

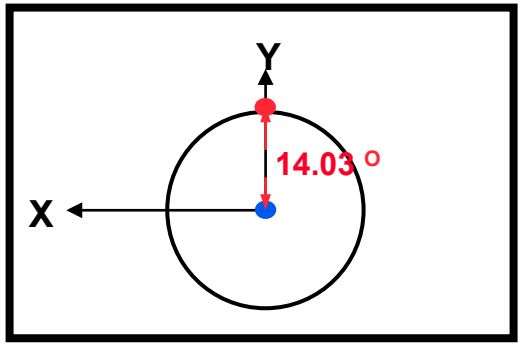
FOV Definition Keywords (4)

- 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
- Neither the boresight nor corner nor 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

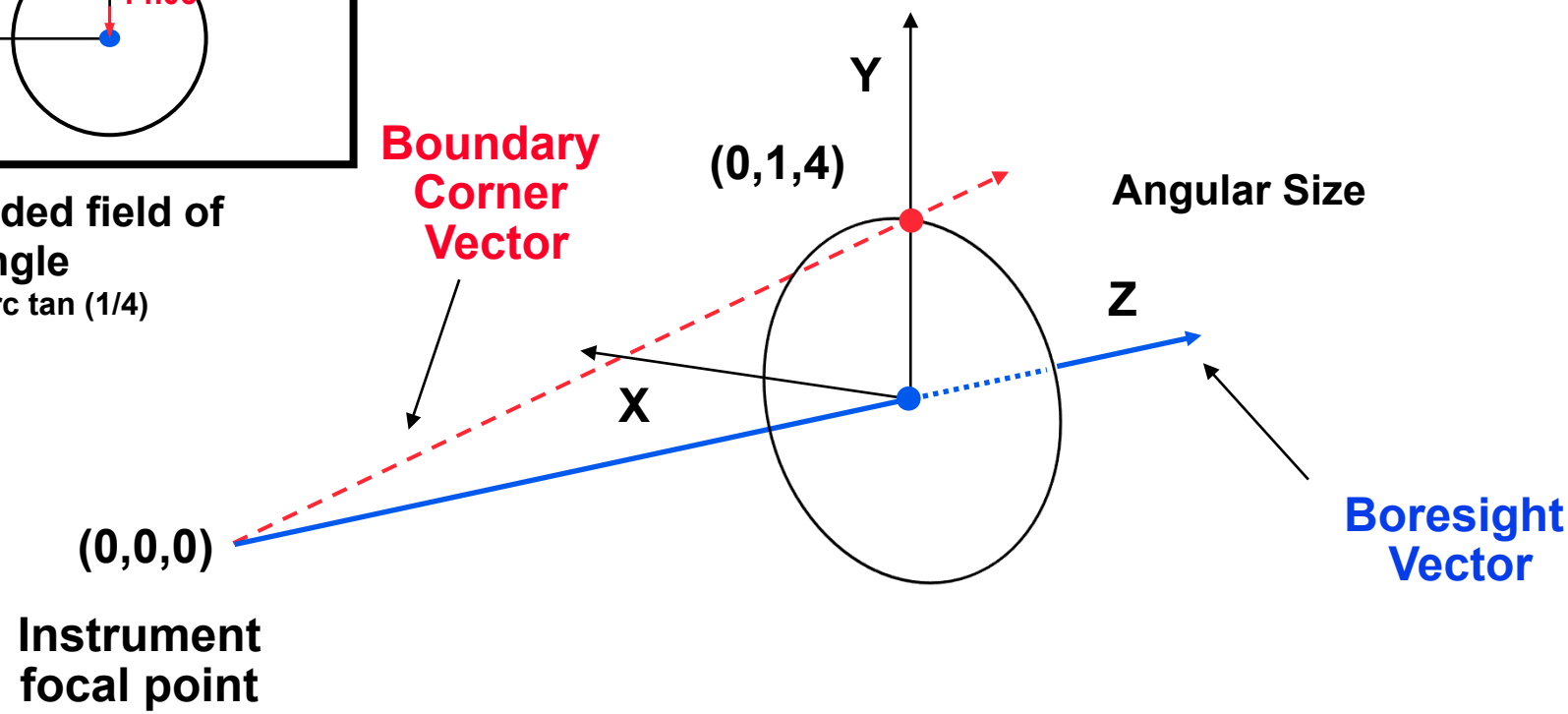
Navigation and Ancillary Information Facility

Consider an instrument with a circular field of view.



Subtended field of view angle

14.03 = arc tan (1/4)



Circular FOV Definition

The following sets of keywords and values describe this circular field of view:

Specifying boundary vectors explicitly:

```
INS-11111_FOV_SHAPE            = 'CIRCLE'
INS-11111_FOV_FRAME            = 'FRAME_FOR_INS-11111'
INS-11111_BORESIGHT            = ( 0.0  0.0  1.0 )
INS-11111_FOV_BOUNDARY_CORNERS = ( 0.0  1.0  4.0 )
```

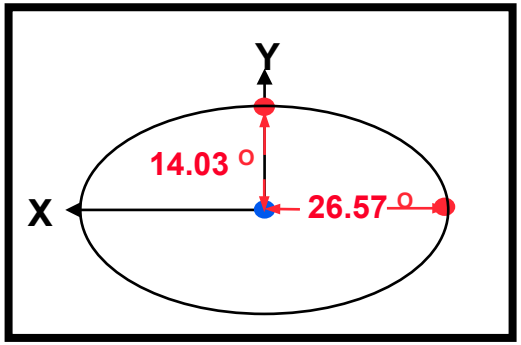
Specifying half angular extents of the FOV:

```
INS-11111_FOV_SHAPE            = 'CIRCLE'
INS-11111_FOV_FRAME            = 'FRAME_FOR_INS-11111'
INS-11111_BORESIGHT            = ( 0.0  0.0  1.0 )
INS-11111_FOV_CLASS_SPEC       = 'ANGLES'
INS-11111_FOV_REF_VECTOR       = ( 0.0  1.0  0.0 )
INS-11111_FOV_REF_ANGLE        = 14.03624347
INS-11111_FOV_ANGLE_UNITS      = 'DEGREES'
```

Elliptical Field of View

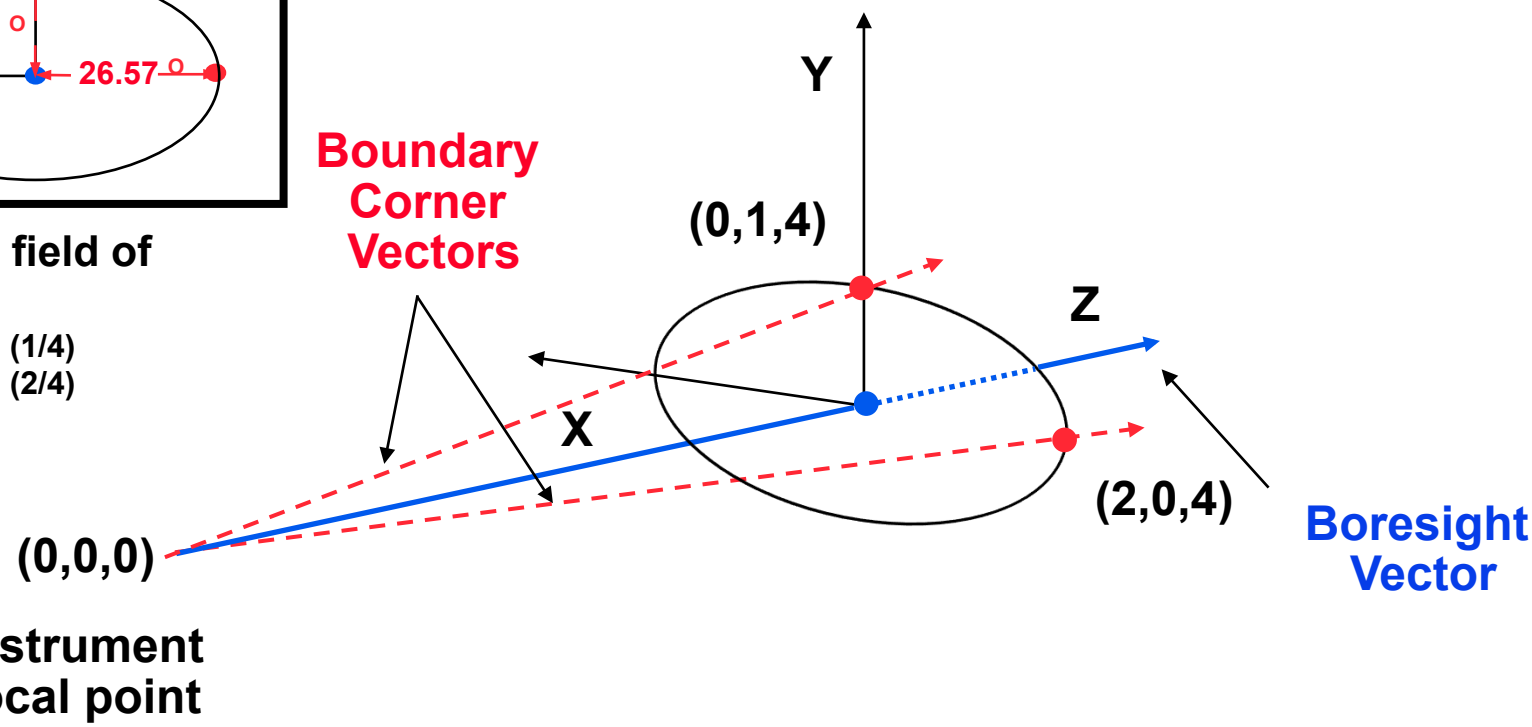
Navigation and Ancillary Information Facility

Consider an instrument with an elliptical field of view.



Subtended field of view angle

14.03 = arc tan (1/4)
26.57 = arc tan (2/4)



Elliptical FOV Definition

The following sets of keywords and values describe this elliptical field of view:

Specifying boundary vectors explicitly:

```
INS-22222_FOV_SHAPE            = 'ELLIPSE'
INS-22222_FOV_FRAME            = 'FRAME_FOR_INS-22222'
INS-22222_BORESIGHT            = ( 0.0  0.0  1.0 )
INS-22222_FOV_BOUNDARY_CORNERS = ( 0.0  1.0  4.0
                                   2.0  0.0  4.0 )
```

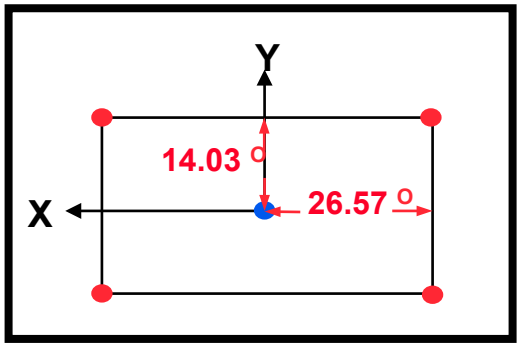
Specifying half angular extents of the FOV:

```
INS-22222_FOV_SHAPE            = 'ELLIPSE'
INS-22222_FOV_FRAME            = 'FRAME_FOR_INS-22222'
INS-22222_BORESIGHT            = ( 0.0  0.0  1.0 )
INS-22222_FOV_CLASS_SPEC       = 'ANGLES'
INS-22222_FOV_REF_VECTOR       = ( 0.0  1.0  0.0 )
INS-22222_FOV_REF_ANGLE        = 14.03624347
INS-22222_FOV_CROSS_ANGLE      = 26.56505118
INS-22222_FOV_ANGLE_UNITS      = 'DEGREES'
```

Rectangular Field of View

Navigation and Ancillary Information Facility

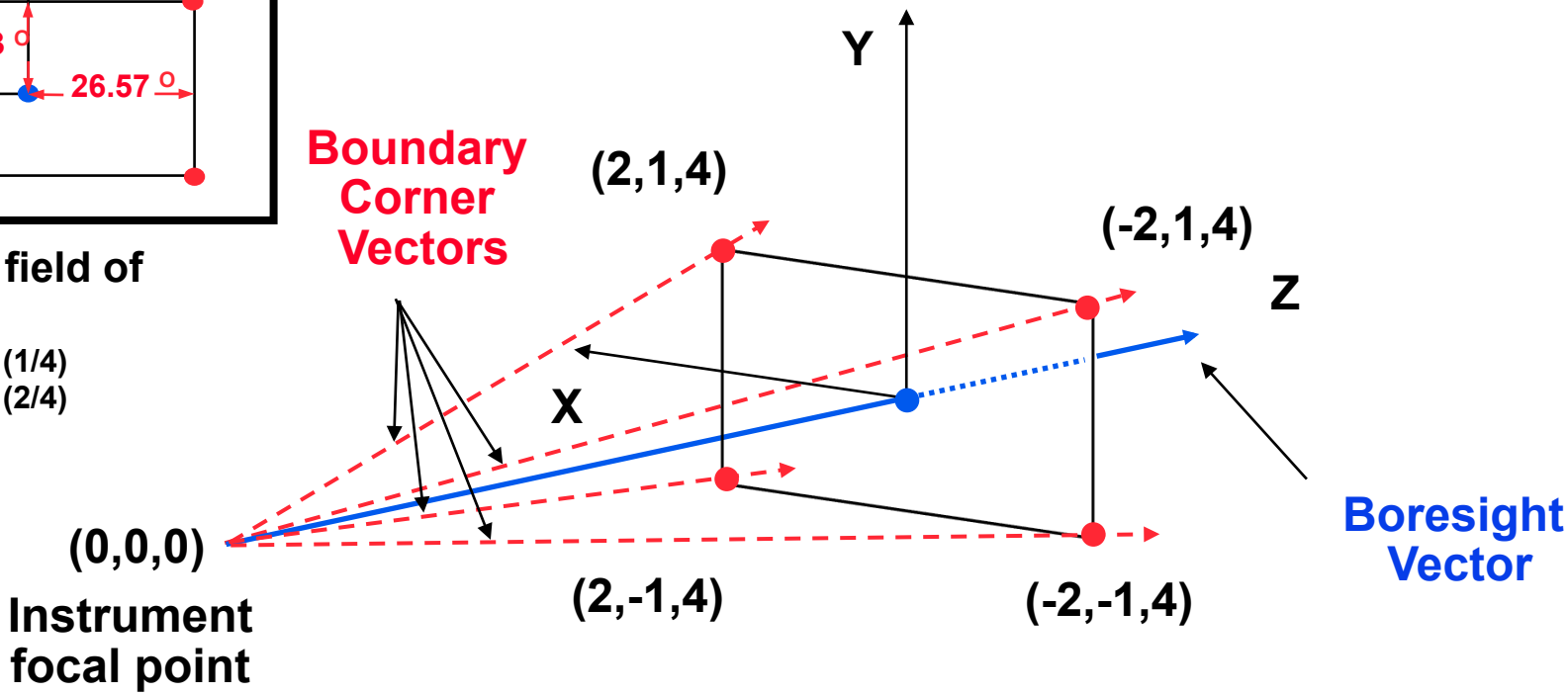
Consider an instrument with a rectangular field of view.



Subtended field of view angle

14.03 = arc tan (1/4)

26.57 = arc tan (2/4)



Rectangular FOV Definition

The following sets of keywords and values describe this rectangular field of view:

Specifying boundary vectors explicitly:

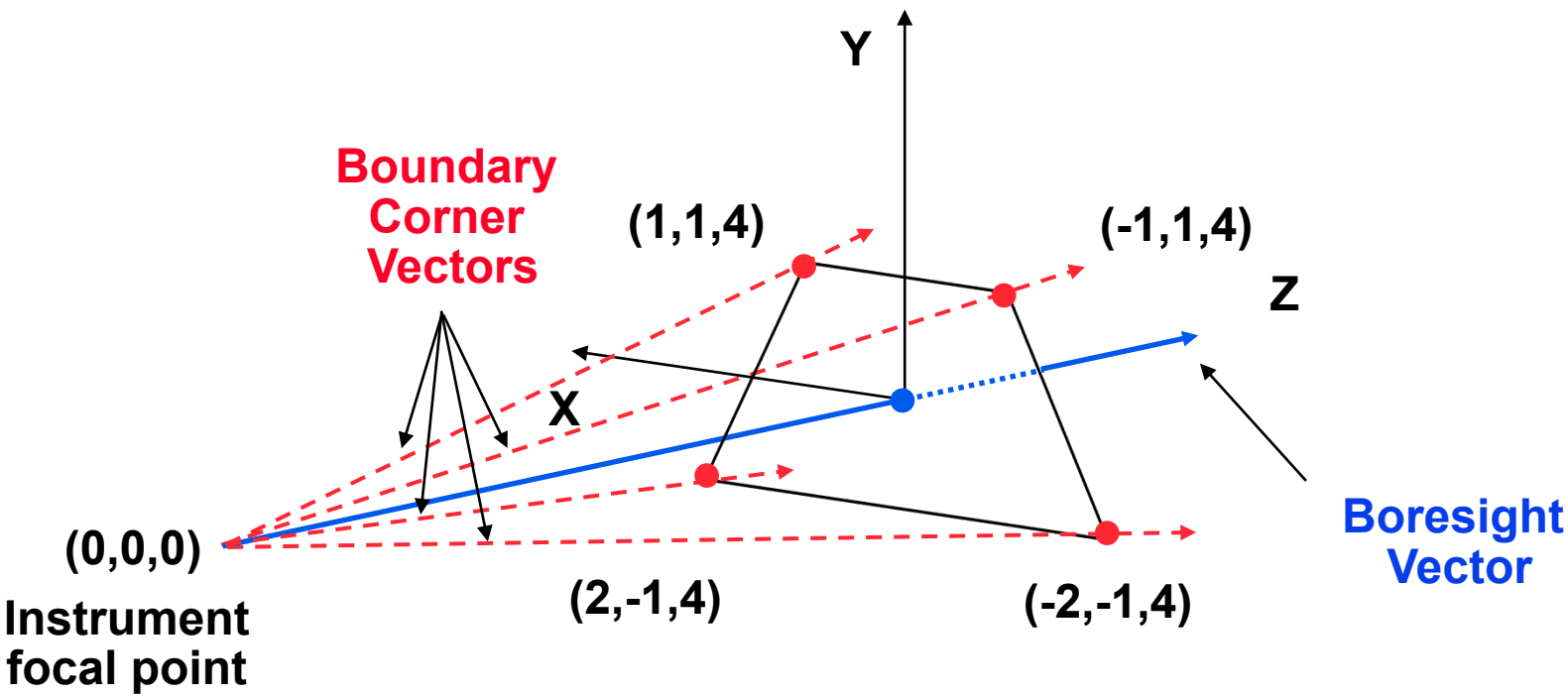
```
INS-33333_FOV_SHAPE            = 'RECTANGLE'
INS-33333_FOV_FRAME            = 'FRAME_FOR_INS-33333'
INS-33333_BORESIGHT            = ( 0.0  0.0  1.0 )
INS-33333_FOV_BOUNDARY_CORNERS = ( 2.0  1.0  4.0
                                  -2.0  1.0  4.0
                                  -2.0 -1.0  4.0
                                   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

Consider an instrument with a trapezoidal field of view.



Polygonal FOV Definition

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

- **No IK utility programs are included in the Toolkit**

- **Two IK utility programs are provided on the NAIF Web site (http://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

- **The best way to learn more about IKs is to examine some found in the NAIF Node archives.**
  - **Start looking here:**

    `http://naif.jpl.nasa.gov/naif/data_archived.html`
- **Unfortunately 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

- **IK file example**

- **Computing angular extents from corner vectors returned by GETFOV**

Sample IK Data

The following LEMMS1 FOV definition was taken from the Cassini MIMI IK (cas_mimi_v11.ti):

```
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.)


                                  ^ Y
                                  |  ins
                                  |
                                  |        /|
                                  |      /  |
                                  |    /    |
                                  |  /    o |
                                  |/  7.50  |
                                  x--------------->
                                X  \        |    Z
                                 ins \      |     ins
                                       \    |
                                         \  |
                                           \|

                                  |-- 1.0 --|
```

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:

           Y Component = 1.0 * tan ( 7.50 degrees )
                       = 0.131652498

   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

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, bnds);

/*  Compute the angular separation. */
    angsiz = vsep_c( bsght, &(bnds[0][0]));
```

Elliptical FOV Angular Size - 1

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)
```

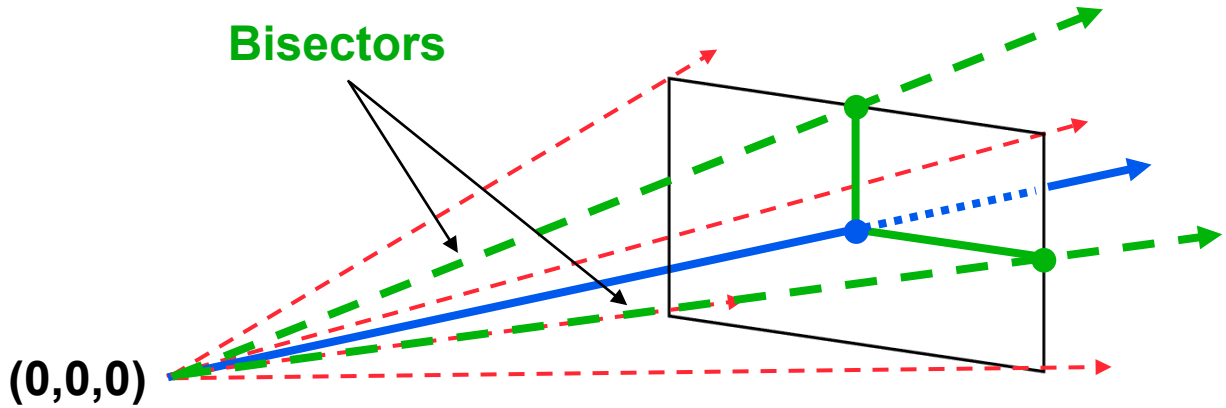
Elliptical FOV Angular Size - 2

C EXAMPLE

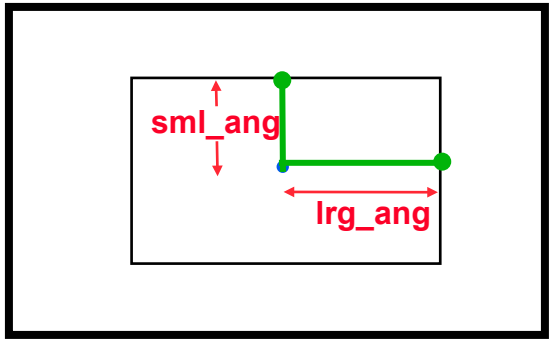
```
/*  Define the string length parameter. */
    #define STRSIZ         80

/*  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

The angular extents of the FOV are computed by calculating the angle between the bisector of adjacent unit boundary vectors and the boresight.



Instrument



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 2012

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.

- DEEP IMPACT:
  - ftp://naif.jpl.nasa.gov/pub/naif/pds/data/di-c-spice-6-v1.0/disp_1000/data/fk/*.tf
  - ftp://naif.jpl.nasa.gov/pub/naif/pds/data/di-c-spice-6-v1.0/disp_1000/data/ik/*.ti
- CASSINI:
  - ftp://naif.jpl.nasa.gov/pub/naif/pds/data/co-s_j_e_v-spice-6-v1.0/cosp_1000/data/fk/*.tf
  - ftp://naif.jpl.nasa.gov/pub/naif/pds/data/co-s_j_e_v-spice-6-v1.0/cosp_1000/data/ik/*.ti
- MESSENGER:
  - ftp://naif.jpl.nasa.gov/pub/naif/pds/data/mess-e_v_h-spice-6-v1.0/messsp_1000/data/fk/*.tf
  - ftp://naif.jpl.nasa.gov/pub/naif/pds/data/mess-e_v_h-spice-6-v1.0/messsp_1000/data/ik/*.ti
- MARS EXPRESS:
  - ftp://naif.jpl.nasa.gov/pub/naif/MEX/kernels/fk/*.TF
  - ftp://naif.jpl.nasa.gov/pub/naif/MEX/kernels/ik/*.TI

Derived Quantities

January 2012

Overview

- What are "derived quantities?"
- A quick tour of some of the routines provided for the computation of derived quantities
  - Vector/Matrix Routines
  - Geometry Routines
  - Coordinate System Routines
  - Lighting angles, intercepts and season
  - Geometry-finder Routines
- Examples of use:
  - Computing Illumination Angles
  - Computing Ring Plane Intercepts
  - Computing Occultation Events

What are Derived Quantities?

Navigation and Ancillary Information Facility

- **Derived quantities are produced using data from kernels**
  - **These are the primary reason that SPICE exists!**
- **Examples are:**
  - **Angles, Angular Rates**
  - **Distances, Speeds**
  - **Directions**
  - **Lighting conditions**
  - **Cartographic parameters (LAT/LON)**
  - **Time windows of events**
- **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.**

A Quick Tour

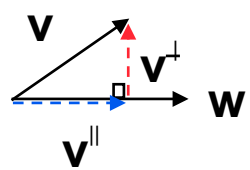
Navigation and Ancillary Information Facility

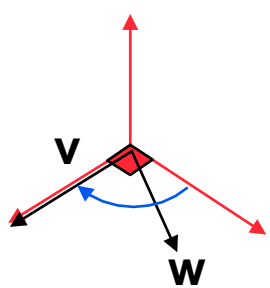
- **Vector/Matrix Routines**
  - Vector and vector derivative arithmetic
  - Matrix arithmetic
- **Geometric "Objects"**
  - Planes
  - Ellipses
  - Ellipsoids
  - Rays
- **Coordinate Systems**
  - Spherical: latitude/longitude, co-latitude/longitude, right ascension/declination; Geodetic, Cylindrical, Rectangular, Planetographic
- **Others**

The lists on the following pages are just a subset of what's available in the Toolkit.

Vectors

Navigation and Ancillary Information Facility

- **Function**
  - **<v,w>**
  - **v** x **w**
  - **v/|v|**
  - **v** x **w** / | **v** x **w**|
  - **v + w**
  - **v - w**
  - a**v** [**+** b**w** [**+** c**u**]]
  - 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

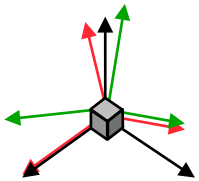
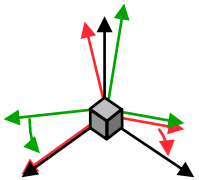
- **Function**
  - M x v
  - M x M
  - M^t x v
  - M^t x M
  - M x M^t
  - v^t x M x v
  - M^t
  - M^{-1}

- **Routine**
  - `MXV`
  - `MXM`
  - `MTXV`
  - `MTXM`
  - `MXMT`
  - `VTMV`
  - `XPOSE`
  - `INVERSE, INVSTM`

`M` = Matrix
`V` = Vector
`X` = Multiplication
`T` = Transpose

Matrix Conversions

Navigation and Ancillary Information Facility

Function			Routines
 Euler angles	⟷ Transform between	$\begin{matrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{matrix}$ 3x3 rotation matrix	– EUL2M, M2EUL
 Euler angles and Euler angle rates *or* rotation matrix and angular velocity vector	⟷ Transform between	$\begin{matrix} a_x & a_y & a_z & & \\ b_x & b_y & b_z & 0 & \\ c_x & c_y & c_z & & \\ \alpha_x & \alpha_y & \alpha_z & a_x & a_y & a_z \\ \beta_x & \beta_y & \beta_z & b_x & b_y & b_z \\ \gamma_x & \gamma_y & \gamma_z & c_x & c_y & c_z \end{matrix}$ 6x6 state transformation matrix	– EUL2XF, XF2EUL RAV2XF, XF2RAV
Rotation axis and angle	⟷ Transform between	$\begin{matrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{matrix}$ 3x3 rotation matrix	– RAXISA, AXISAR ROTATE, ROTMAT
(Q_0,Q_1,Q_2,Q_3) SPICE Style Quaternion	⟷ Transform between	$\begin{matrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{matrix}$ 3x3 rotation matrix	– Q2M, M2Q

Geometry

Navigation and Ancillary Information Facility

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

Position Coordinate Transformations

Navigation and Ancillary Information Facility

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

Velocity Coordinate Transformations - 1

Navigation and Ancillary Information Facility

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

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

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

Velocity Coordinate Transformations - 2

- Example: 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 (colatitude) / dt, d (longitude) /dt)

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

Other Derived Geometric Quantities

- Illumination angles (phase, incidence, emission)
  - ILUMIN*
- Subsolar point
  - SUBSLR*
- Subobserver point
  - SUBPNT*
- Surface intercept point
  - SINCPT*
- Longitude of the sun (L_s), an indicator of season
  - LSPCN

* These routines supercede the now deprecated routines ILLUM, SUBSOL, SUBPT and SRFXPT

Geometric Events Finder

- **Most SPICE routines are used to determine a quantity at a specified time.**

- **The SPICE Geometry Finder (GF) subsystem takes the opposite approach: find times, or time spans, when a specified geometric condition or event occurs.**
  - **This is such a large topic that a separate tutorial ("geometry_finder") has been written for it.**

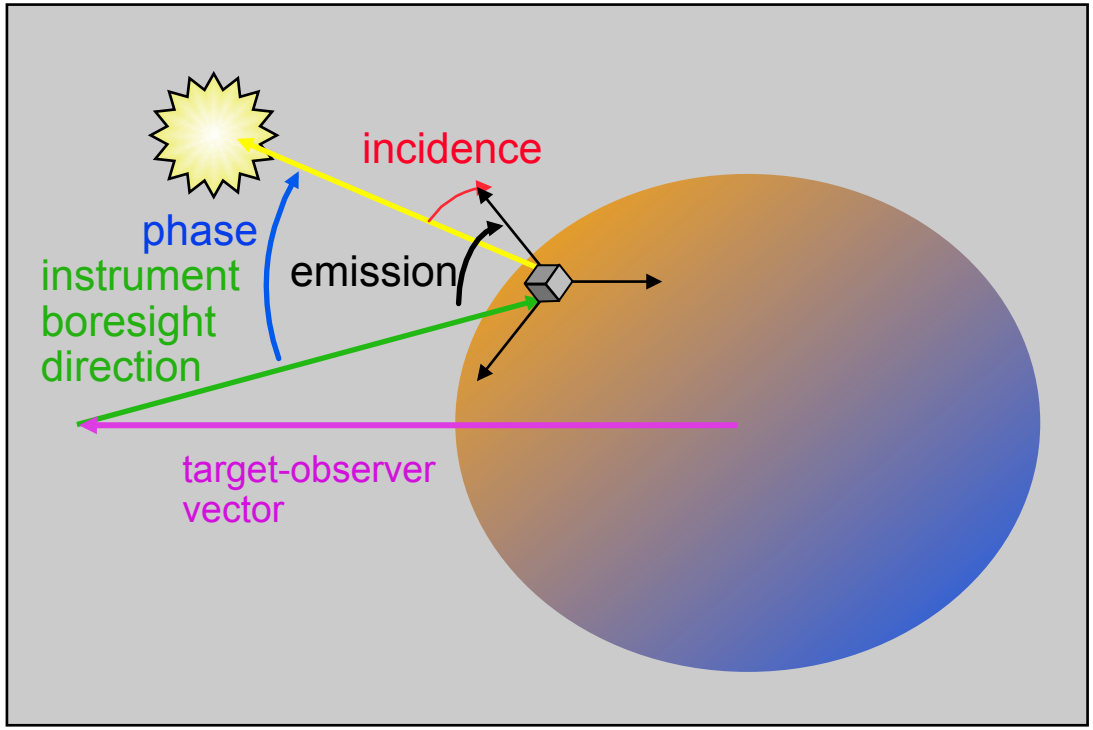
Examples

- On the next several pages we present examples of using some of the “derived quantity” APIs.

Computing Illumination Angles

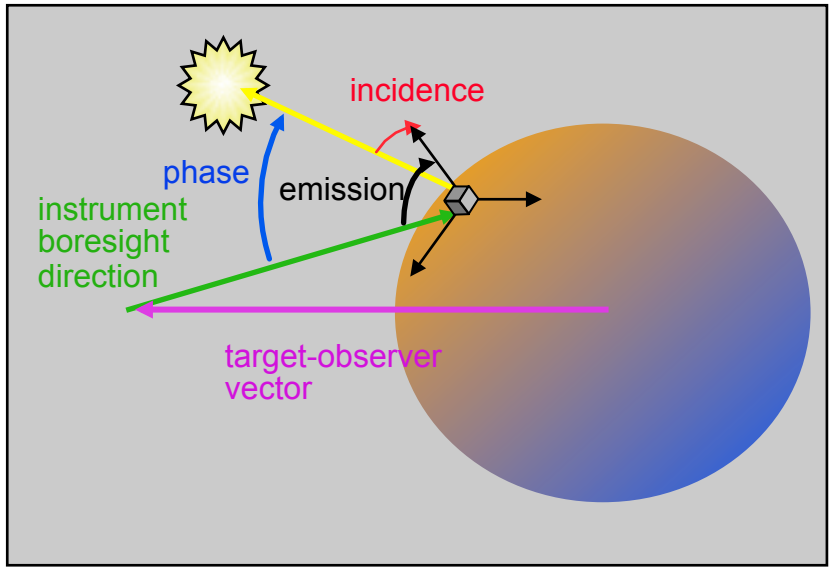
Navigation and Ancillary Information Facility

- **Given the direction of an instrument boresight in a bodyfixed frame, return the illumination angles (incidence, phase, emission) at the surface intercept on a tri-axial ellipsoid**



Computing Illumination Angles

Navigation and Ancillary Information Facility

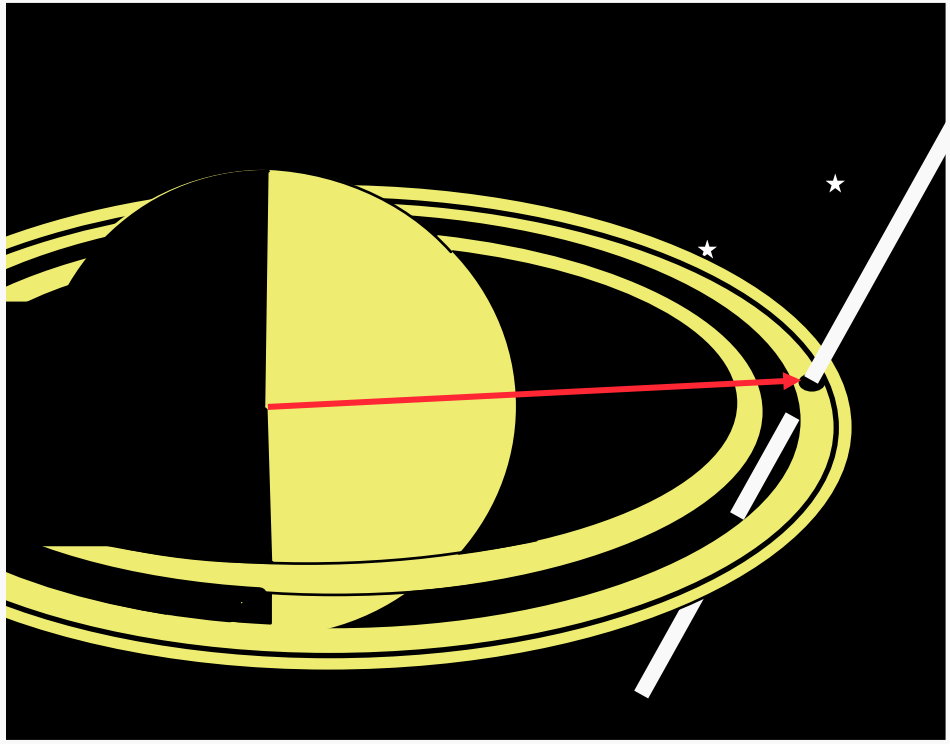


- CALL GETFOV to obtain boresight direction vector
- CALL SINCPT to find intersection of boresight direction vector with surface
- CALL ILUMIN to determine illumination angles

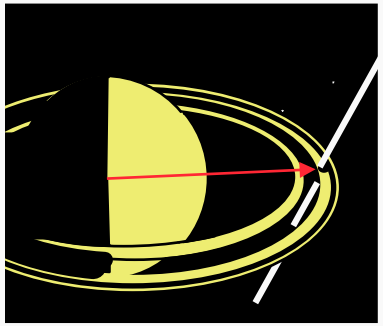
Computing Ring Plane Intercepts

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



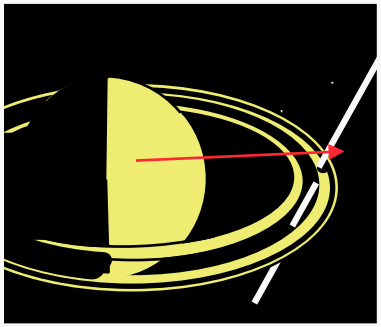
This computation is for the reception case; radiation is received at the earth at a given epoch “ET”.

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 can be re-computed using the light time from earth to the intercept; the intercept can be re-computed until convergence is attained.

- **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

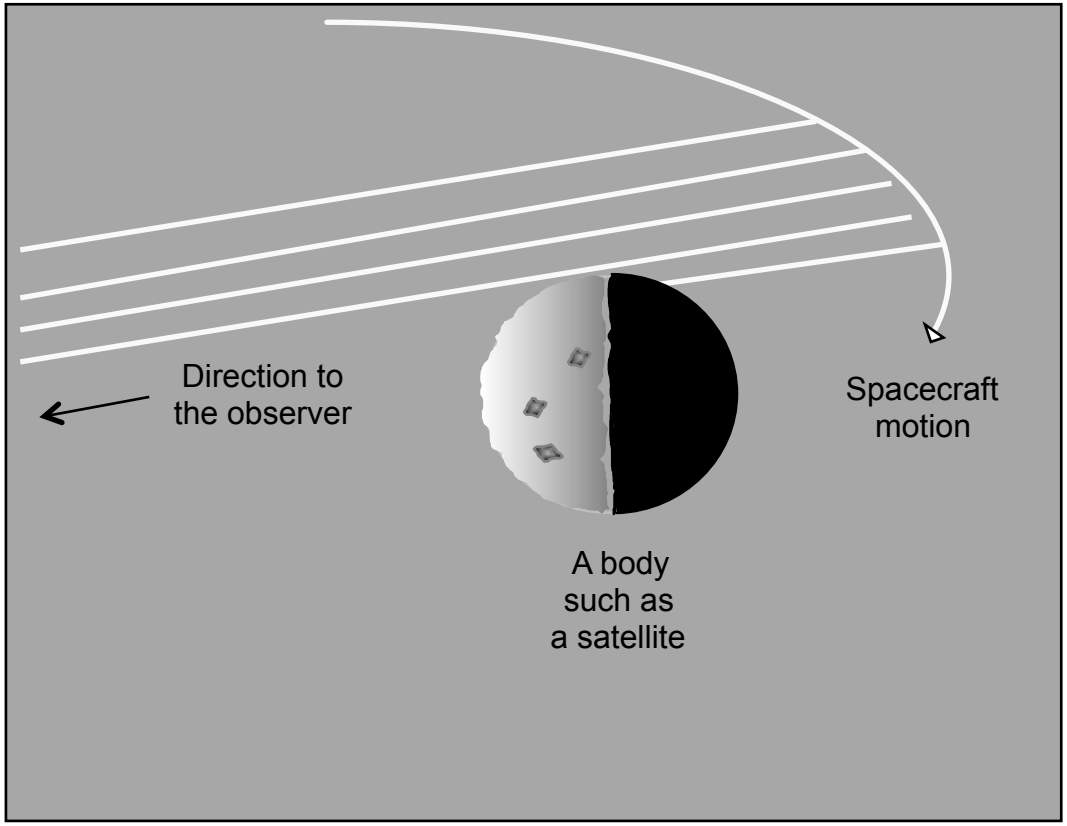


An alternate approach

- 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.

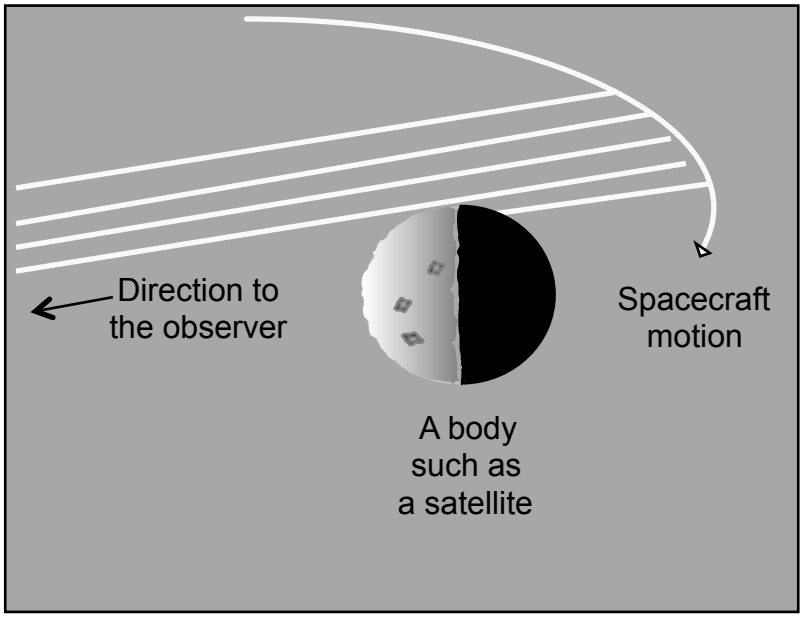
Computing Occultation Events

- **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 shorter than the minimum interval between occultation events that are to be distinguished, but large enough to solve 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 or points (but at least one must be an ellipsoid).
  - GFOCLT can search for different occultation or transit geometries: full, partial, annular, or "any."

Other Useful Functions

January 2012

Topics

- 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

- 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.

Text I/O (1)

- 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.

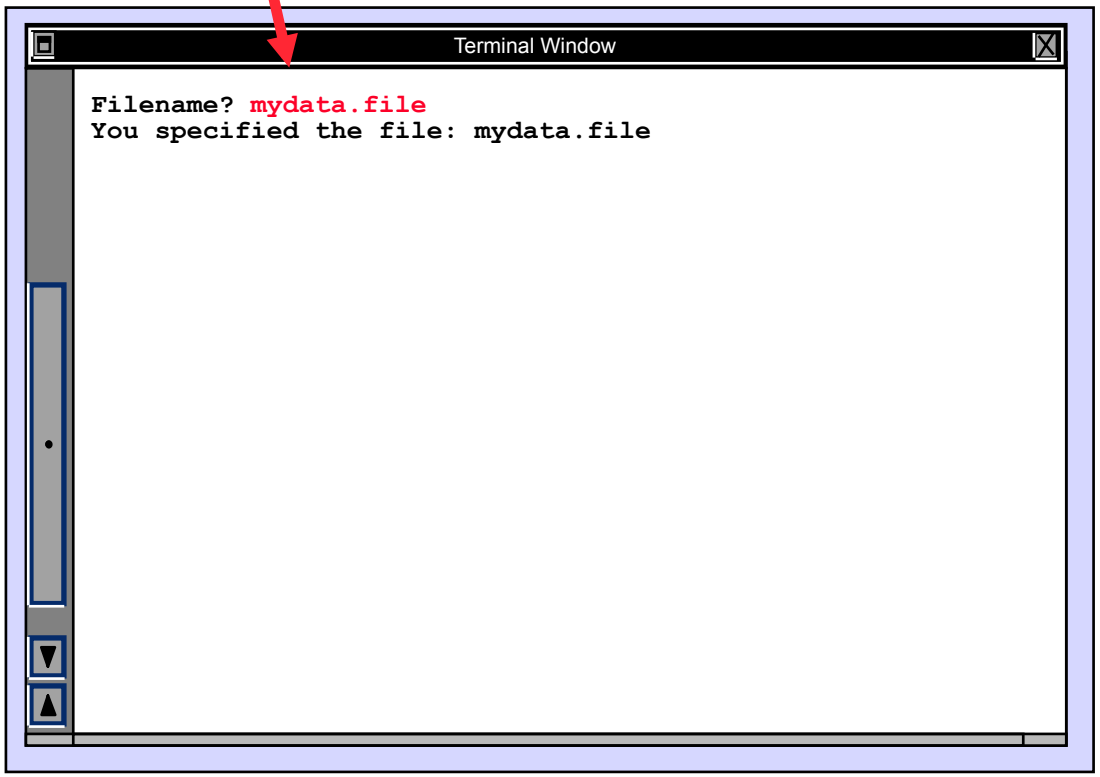
Text I/O (2)

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

- 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.
- Determining whether or not a file exists
  - [F,C,I] EXISTS
- Deleting an existing file
  - [F] DELFIL

String Manipulation - Parsing (1)

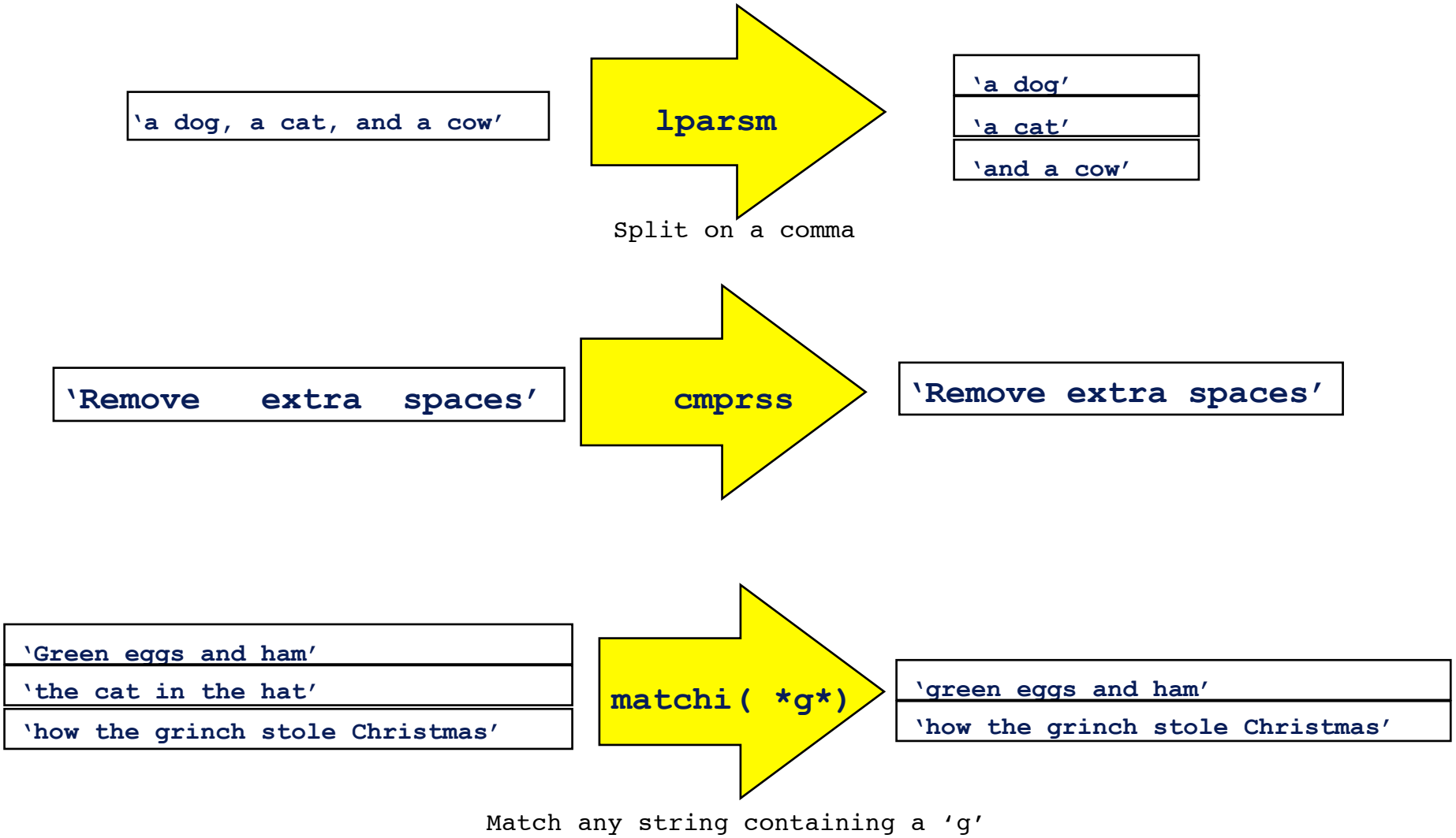
- Breaking apart a list
  - [F,C,I] LPARSE: parses a list of items delimited by a single character.
  - [F,C] LPARSM: parses a list of items separated by multiple delimiters.
  - [F] NEXTWD: returns the next word in a given character string.
  - [F] NTHWD: returns the nth word in a string and the location 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)

- 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, NPARSI, DXTRCT, TPARTV
  - [F,C,I] PRSDP, PRSINT, TPARSE
- Heavy duty parsing
  - [F] SCANIT

String Manipulation - Parsing (3)

Navigation and Ancillary Information Facility



String Manipulation - Creating (1)

- **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] REPMI: Replace a marker with an integer.**

    ```
    CALL REPMI ( '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)

- **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, INTTXT, INTORD**

String Manipulation - Creating (3)

- 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

Searching, Sorting and Other Array Manipulations (1)

- 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.[1]
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.[1]
earth	00.983
jupiter	05.440
mars	01.531
mercury	00.445
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.

[1] Distance in A.U. at Jan 01, 2006.

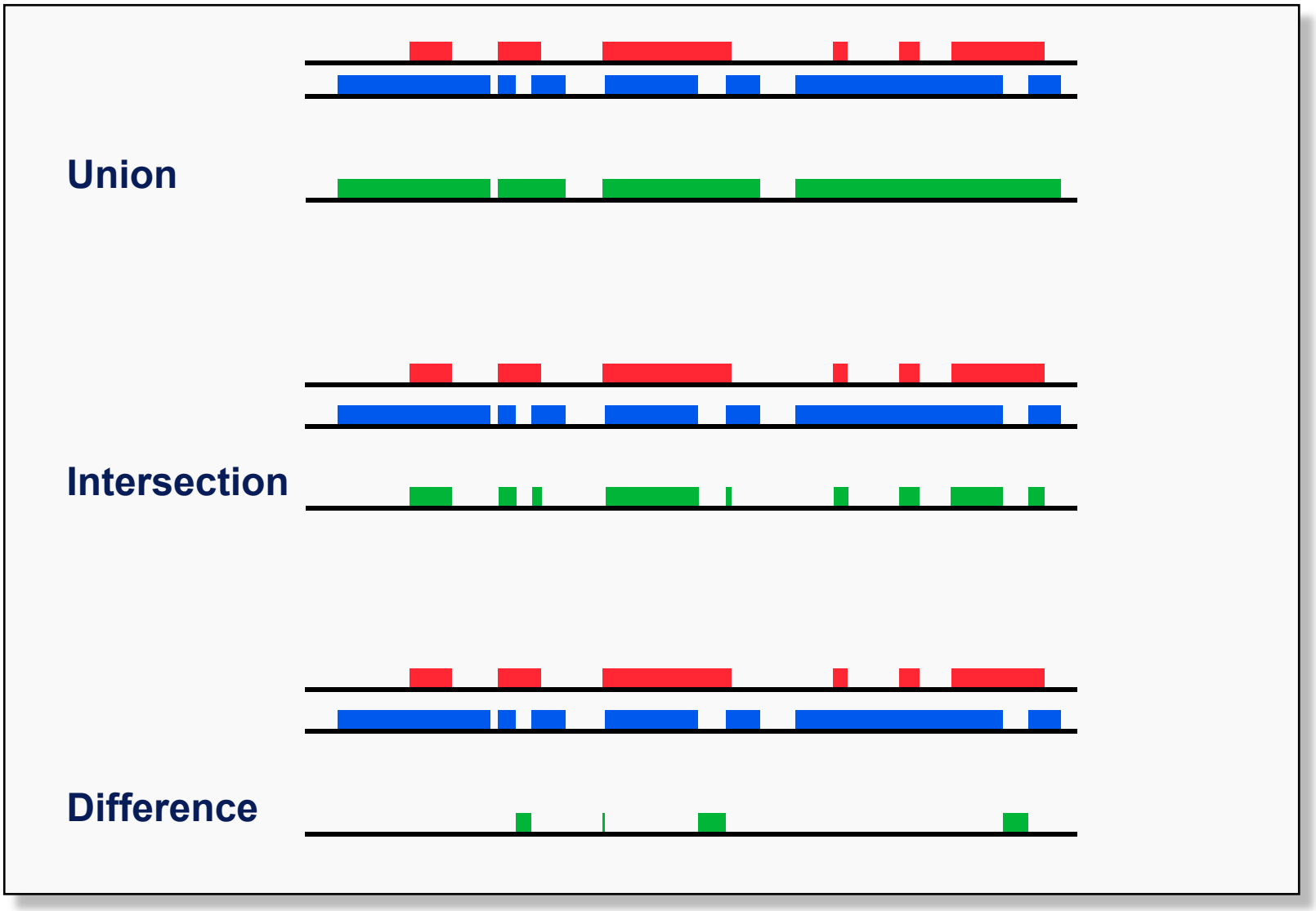
Windows

- 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



Symbol Tables

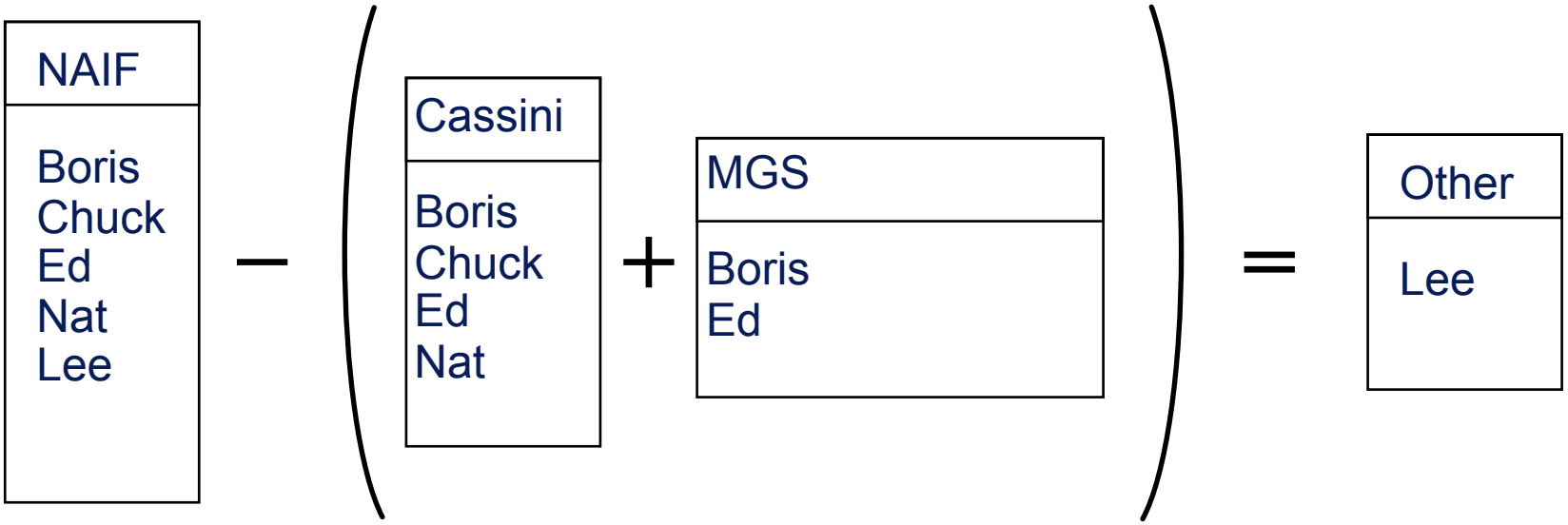
- **SPICELIB (Fortran) supports the use of associative arrays/ hashes through the use of an abstract data type called symbol tables.**
  - **These are used to associate a set of names with collections of associated values.**
  - **Values associated with a name are exclusively character, exclusively integer 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 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)

- 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.
  - Sets and cells aren't currently needed in the MATLAB Toolkits since MATLAB 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)

- 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

Numerical Functions (2)

- 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 events occur

January 2012

Topics

- GF Subsystem Overview
- GF Search Examples
- SPICE Windows
- Geometric Search Types and Constraints
- Development Plans
- Backup

GF Subsystem Overview

Purpose

- **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 Mars Reconnaissance Orbiter is in the penumbral shadow of Mars
    - » The Saturn phase angle as seen by the Cassini orbiter is 60 degrees
  - Each GF search is conducted over a user-specified time 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.

GF High-Level API Routines

- 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 containment in a specified 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 appearances in a specified instrument's field of view
  - GFUDB: user-defined boolean quantity
  - GFUDS: user-defined scalar quantity (Fortran and C only)

GF Mid-Level API Routines

- The current Fortran and C SPICE Toolkits provide some mid-level API routines 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 API routines are:
  - GFEVNT: All scalar numeric quantity searches
  - GFFOVE: Target or ray in FOV searches
  - GFOCCE: Occultation searches

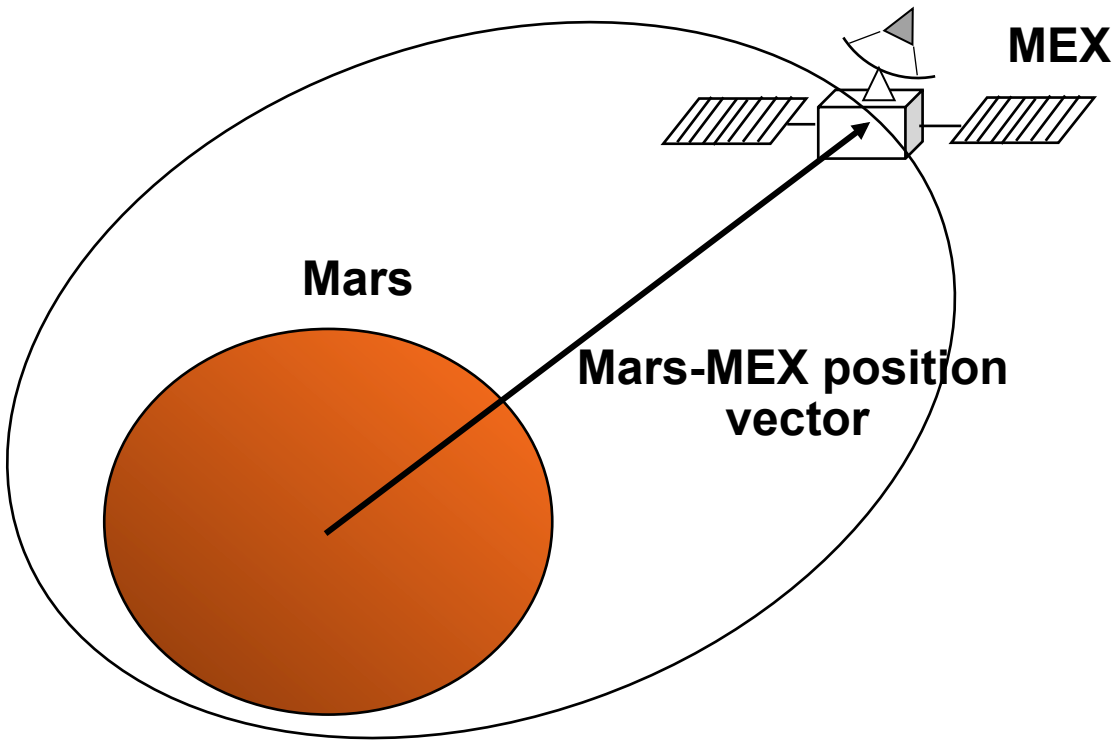
GF Documentation

- The GF module headers and API reference guides contain complete example programs for each GF API routine.
  - These may be the best source of documentation for quickly getting started using the GF routines.
- The GF Required Reading document gf.req contains
  - Extensive, moderately advanced example programs
  - A set of “computational recipes” that outline how to use the GF subsystem to solve various popular search problems
  - Discussions of mid-level GF APIs available only in the Fortran and C Toolkits
  - Discussion of technical details of the GF subsystem
  - Discussion of anticipated problems that may arise while using the GF subsystem
- Documentation on SPICE windows:
  - The WINDOWS Required Reading windows.req
  - The Other Functions tutorial
  - API documentation for SPICE window routines

GF Search Examples

Distance Local Extremum Search

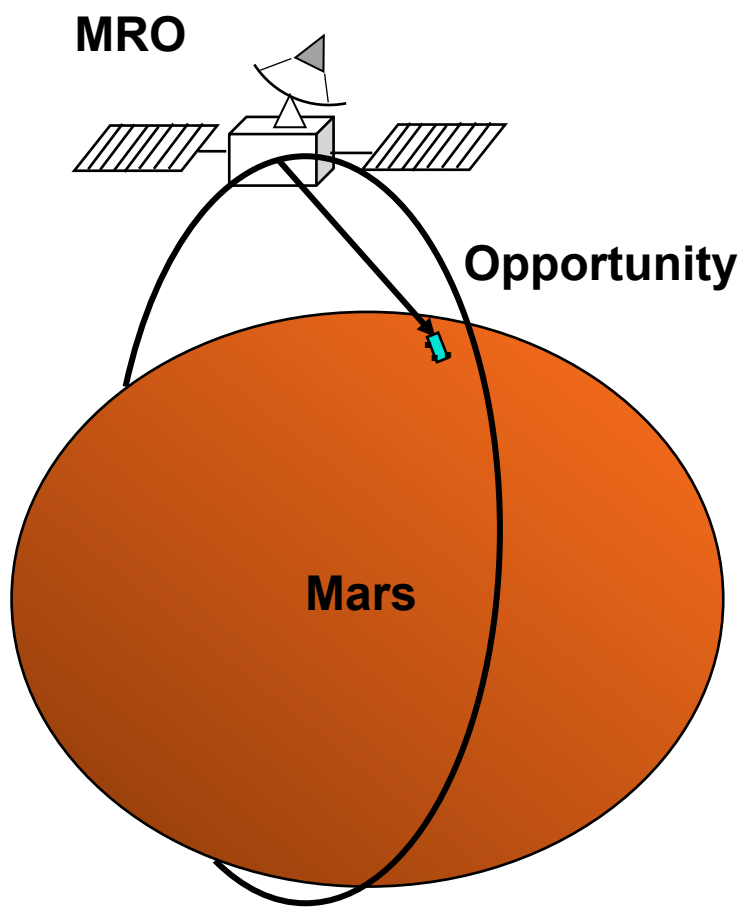
Find the time of apoapse of the Mars Express Orbiter (MEX)



Distance Inequality Search

Navigation and Ancillary Information Facility

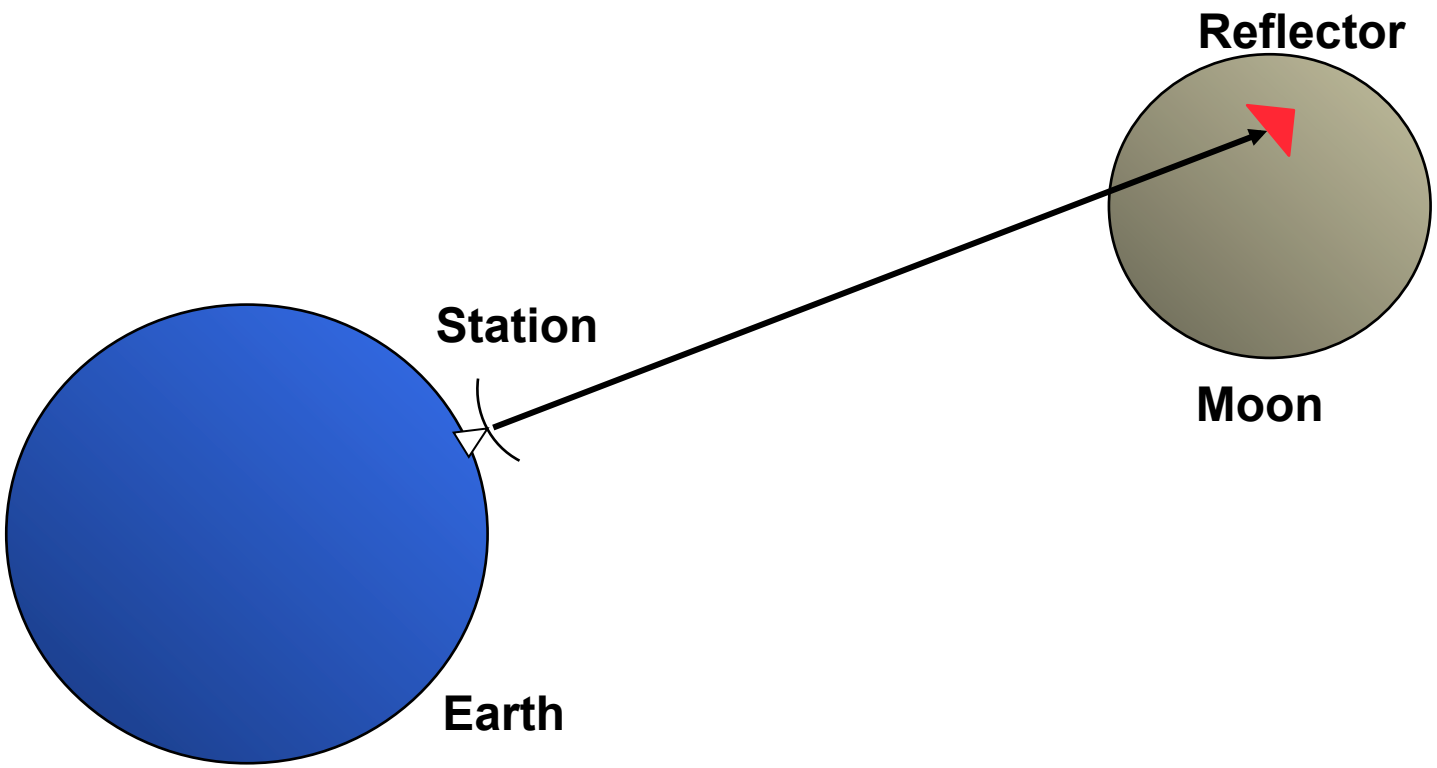
Find the time period when the Mars Reconnaissance Orbiter (MRO) is within 500km of the Opportunity rover.



Range Rate Extremum Search

Navigation and Ancillary Information Facility

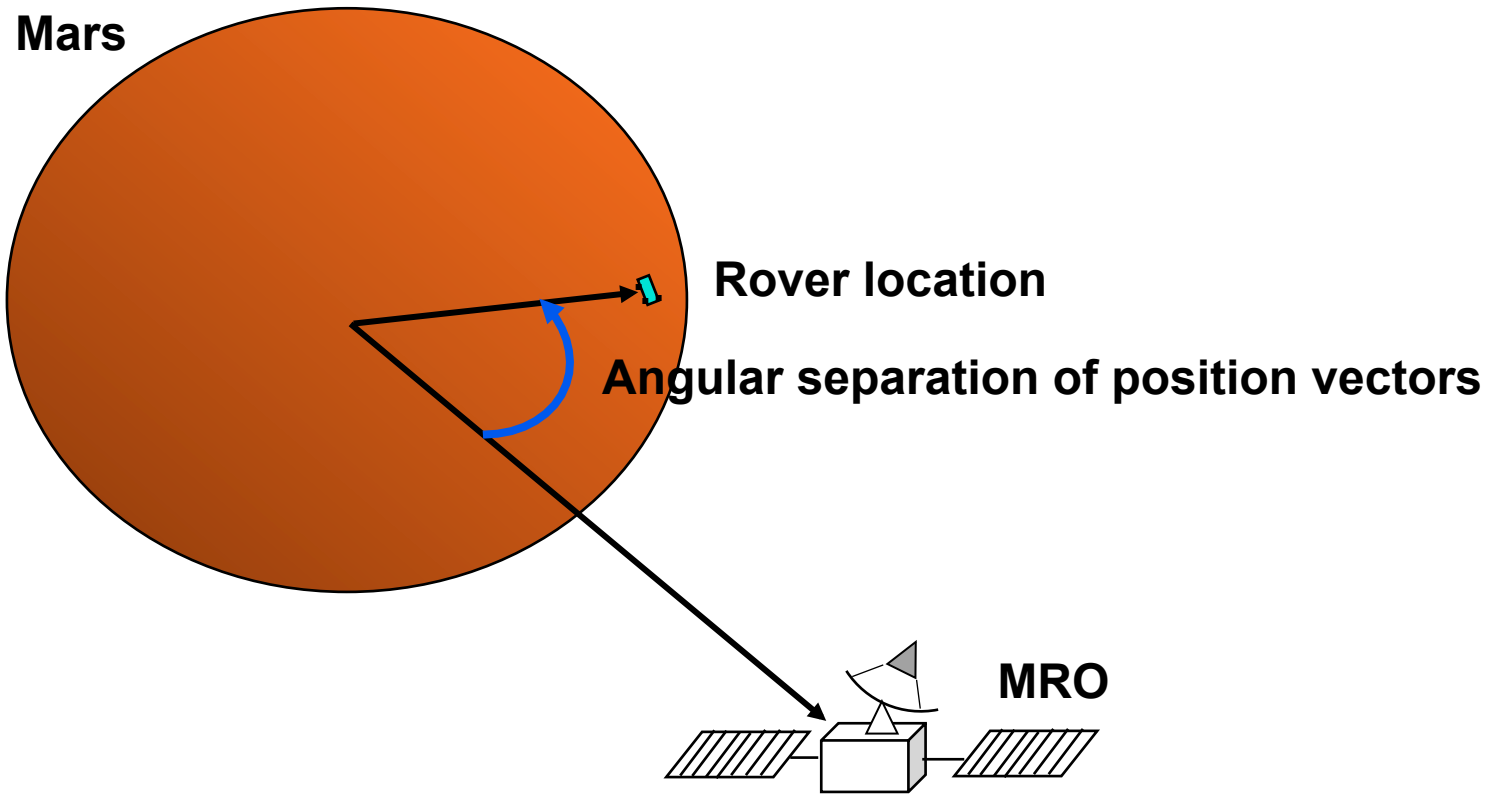
Find the times when the range rate of a lunar reflector, as seen by an Earth station, attains an absolute extremum.



Angular Separation Inequality Search -1

Navigation and Ancillary Information Facility

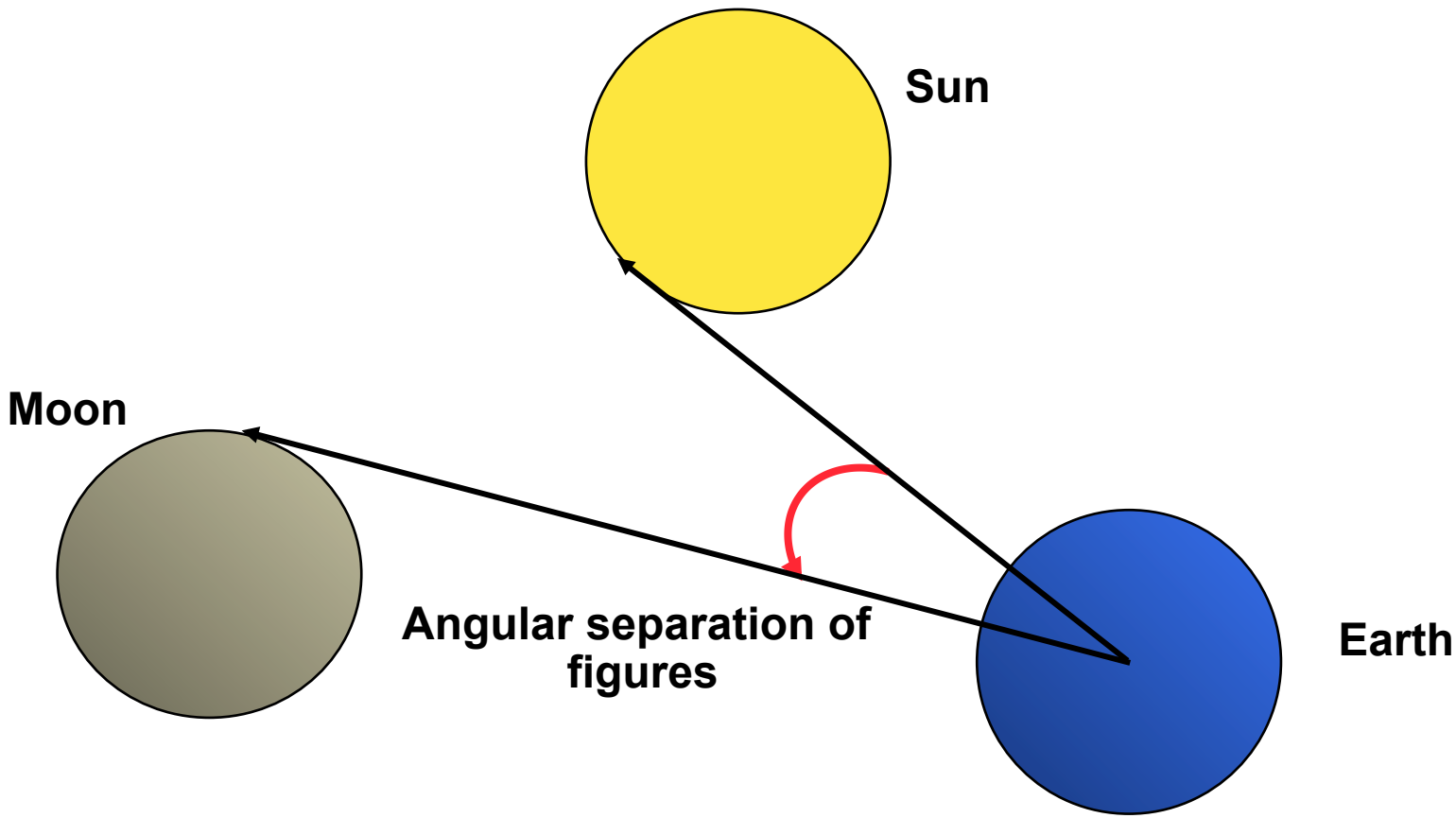
Find the time period when the angular separation of the Mars-to Mars Reconnaissance Orbiter (MRO) and Mars-to-Opportunity Rover position vectors is less than 3 degrees. Both targets are modeled as points.



Angular Separation Inequality Search -2

Navigation and Ancillary Information Facility

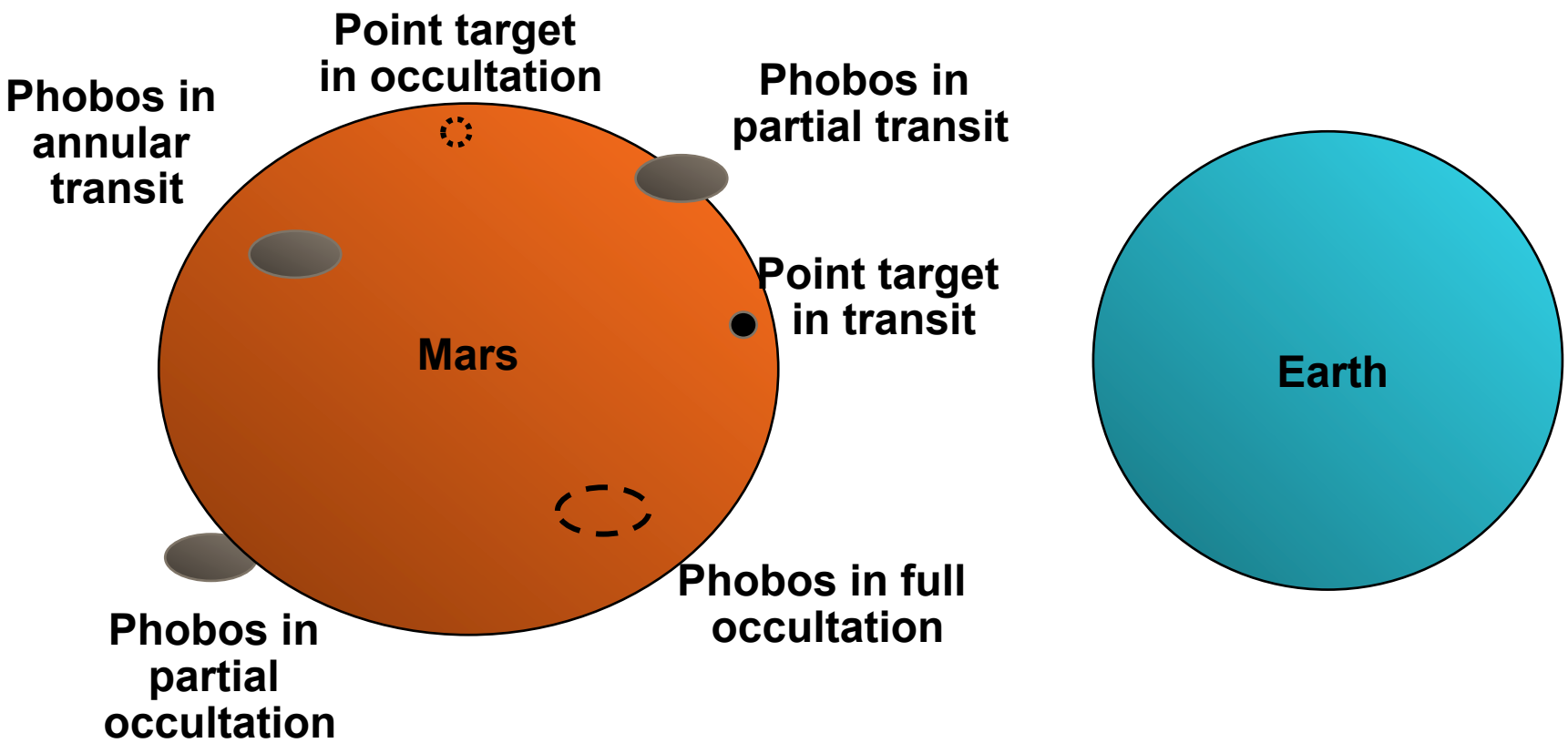
Find the time period when the angular separation of the figures of the Moon and Sun, as seen from the Earth, is less than 1 degree. Both targets are modeled as spheres.



Occultation/Transit Search

Navigation and Ancillary Information Facility

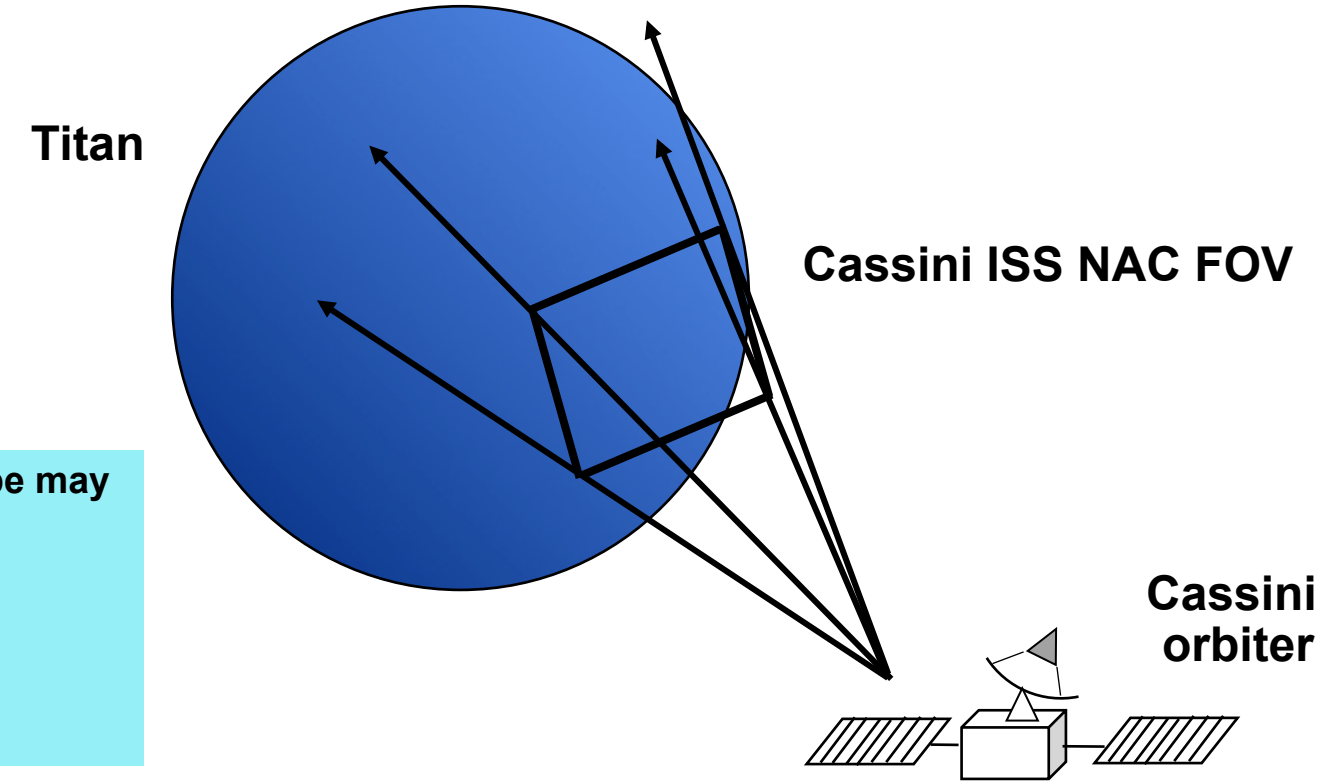
Find the ingress and egress times of an occultation of Phobos by Mars, as seen from Earth. Both targets are modeled as triaxial ellipsoids.



Target in FOV Search

Navigation and Ancillary Information Facility

Find the time period when Titan appears in the FOV of the Cassini ISS Narrow Angle Camera (NAC). The target is an ephemeris object; the target shape is modeled as an ellipsoid. (Point targets are also supported.)



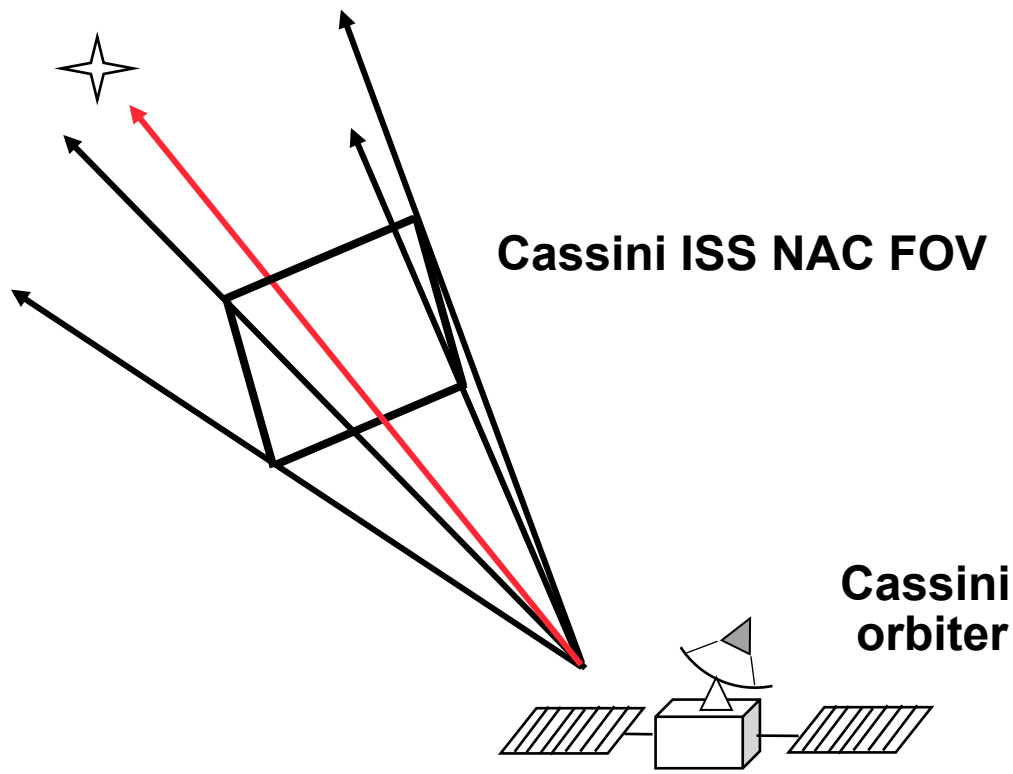
The FOV shape may be any of:

- Rectangle
- Circle
- Ellipse
- Polygon

Ray in FOV Search

Navigation and Ancillary Information Facility

Find the time period when the star having Hipparcos catalog number 6000 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.

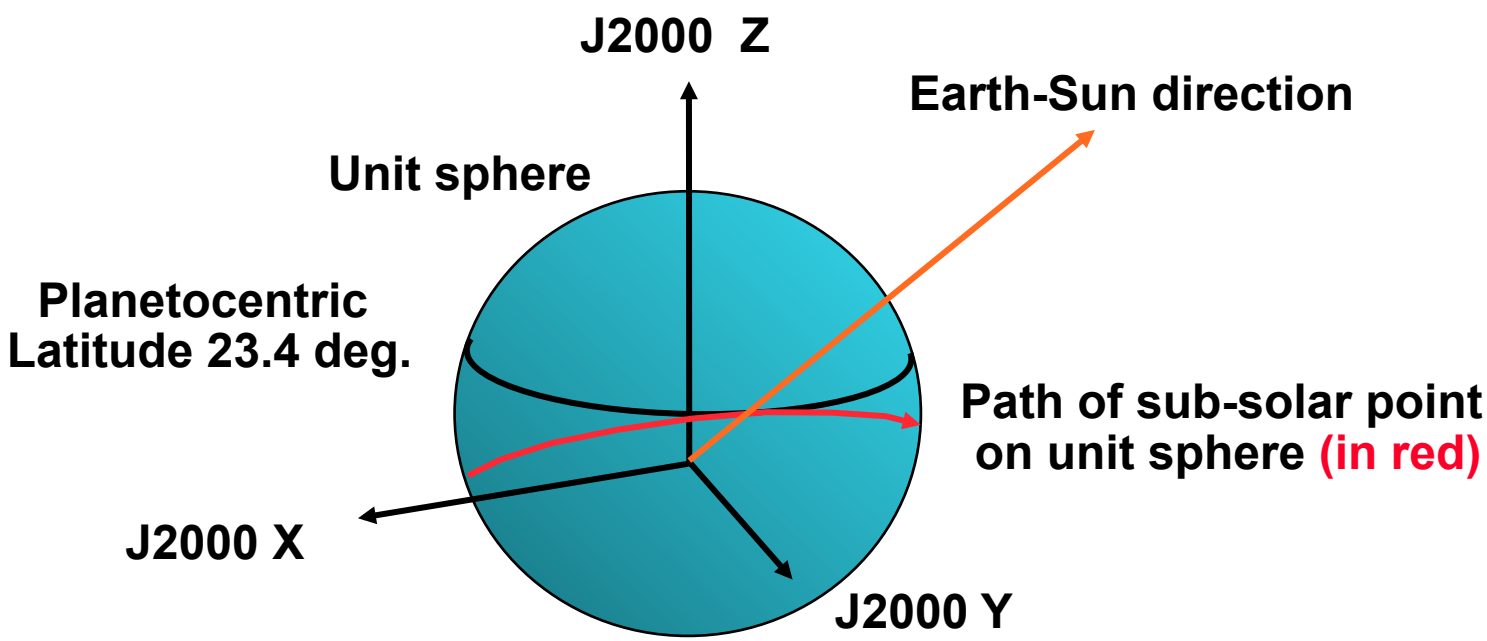


The FOV shape may be any of:

- Rectangle
- Circle
- Ellipse
- Polygon

Position Coordinate Local Extremum Search

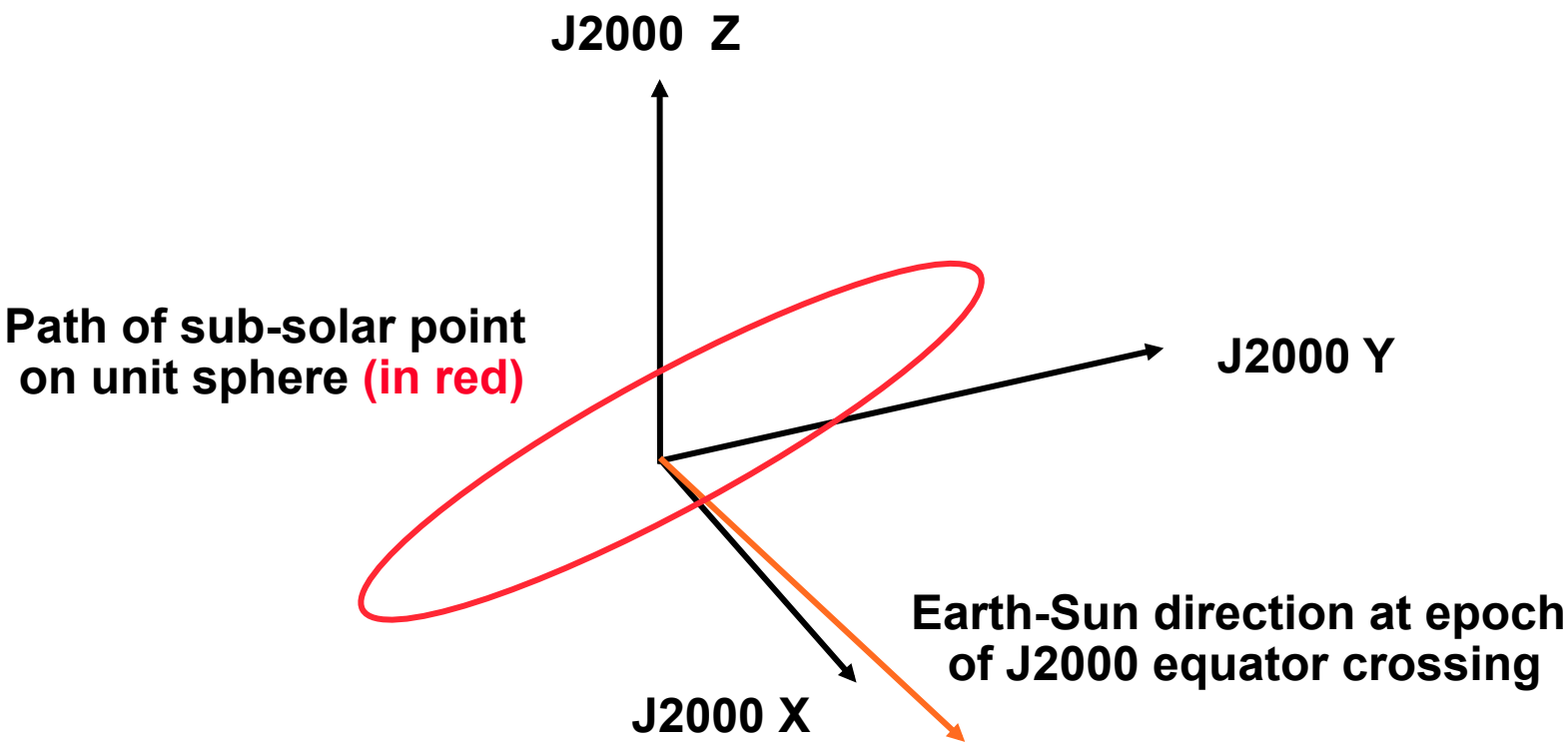
Find the time(s) at which the planetocentric latitude of the Earth-Sun vector, expressed in the J2000 frame, is at a local maximum.



Position Coordinate Equality Search

Navigation and Ancillary Information Facility

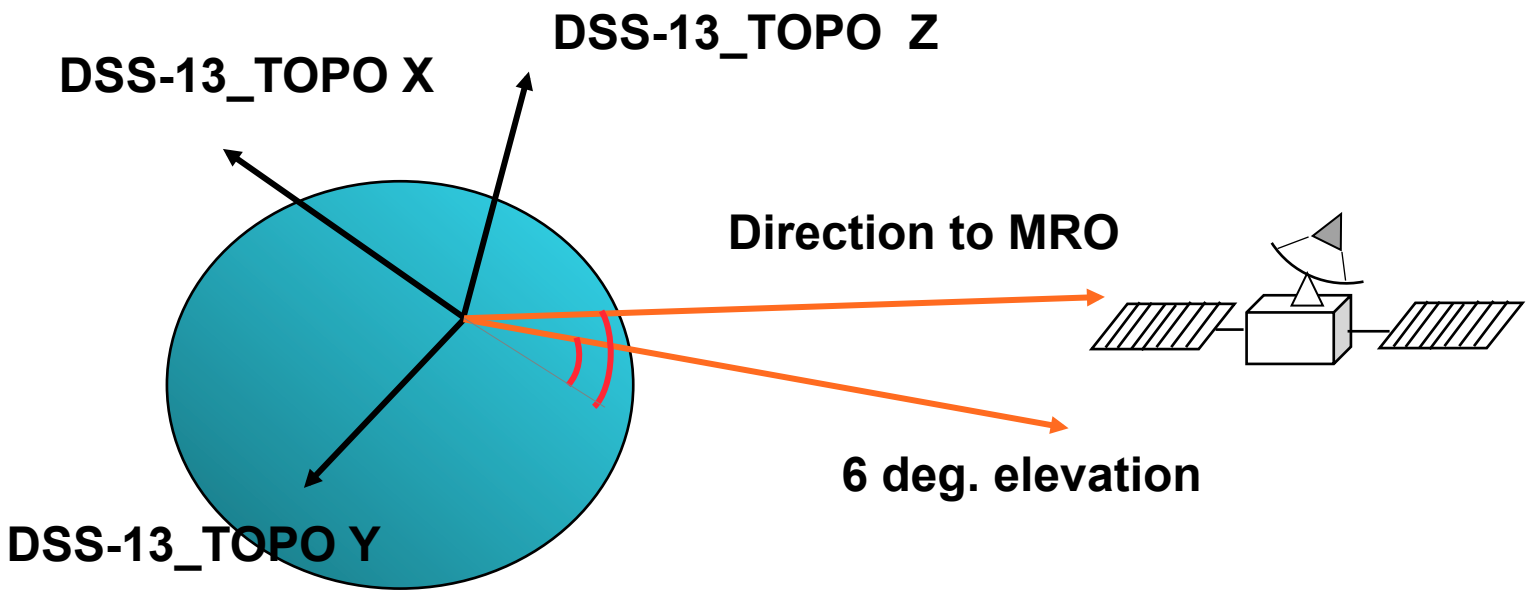
Find the time(s) at which the Z component of the Earth-Sun vector, expressed in the J2000 frame, is 0.



Position Coordinate Inequality Search -1

Navigation and Ancillary Information Facility

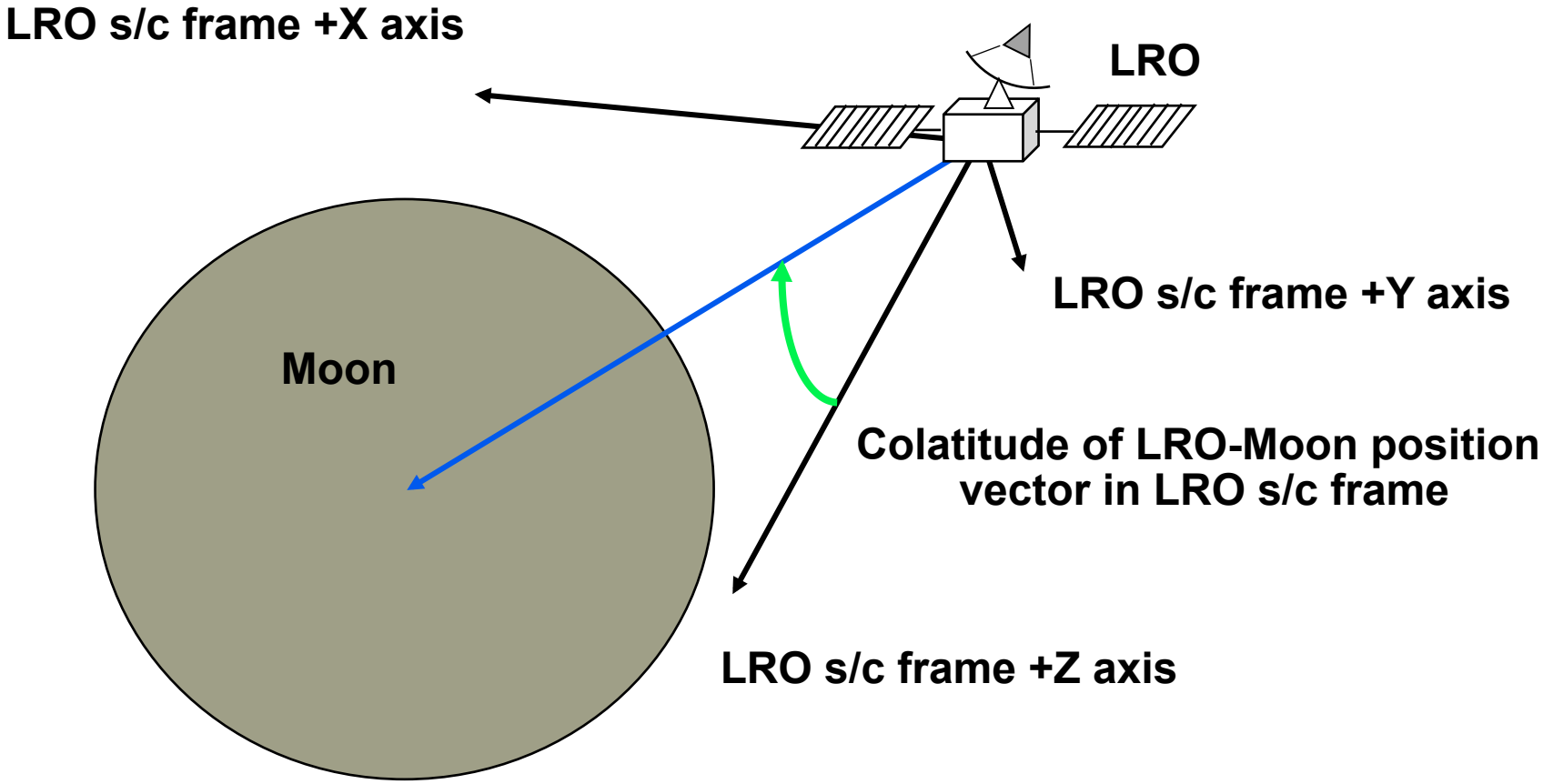
Find the time period when the elevation of the DSS-13 to Mars Reconnaissance Orbiter (MRO) spacecraft vector, expressed in the DSS-13 topocentric frame, is greater than 6 degrees.



Position Coordinate Inequality Search -2

Navigation and Ancillary Information Facility

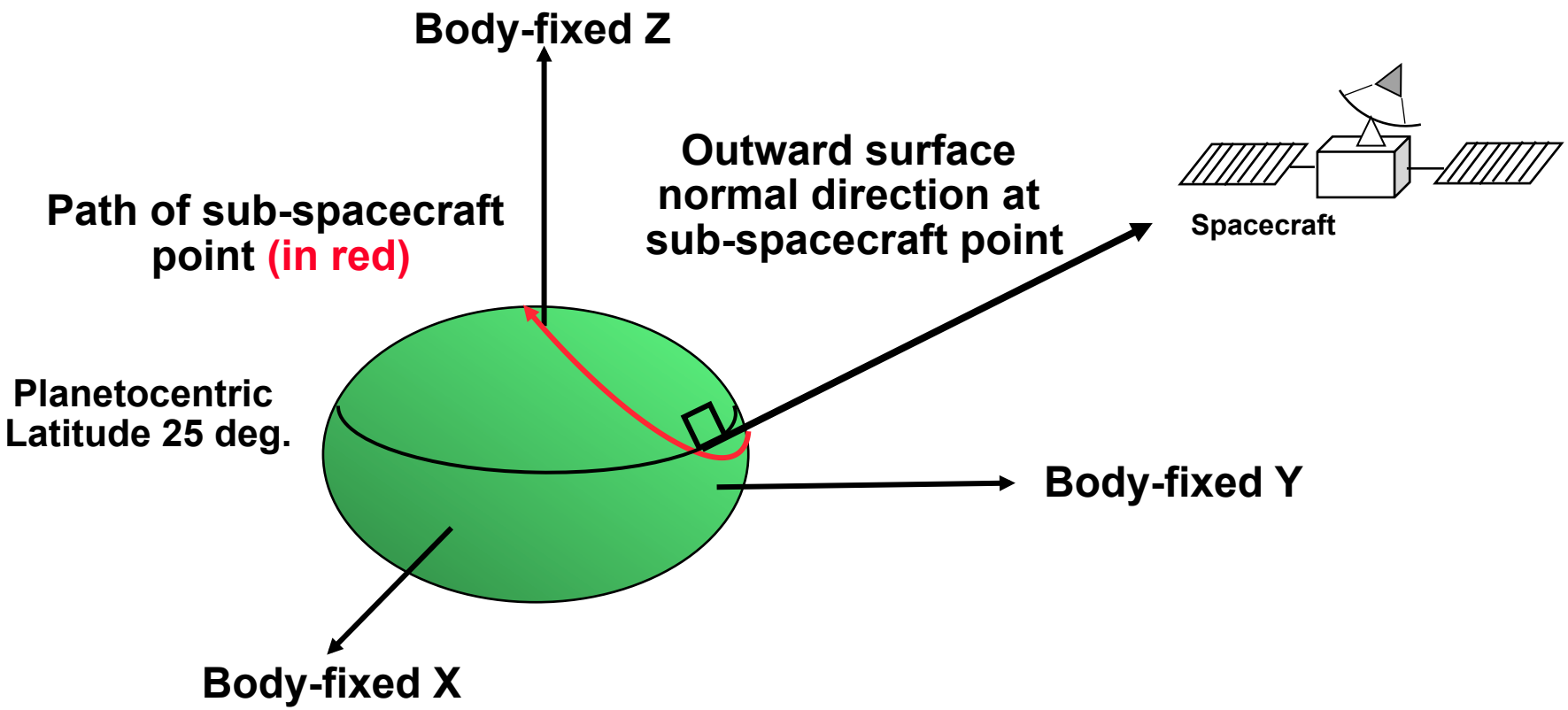
Find the time period when the Lunar Reconnaissance Orbiter's (LRO) off-nadir angle exceeds 5 degrees. Solution requires CK-based "LRO spacecraft frame."



Sub-Observer Point Coordinate Equality Search

Navigation and Ancillary Information Facility

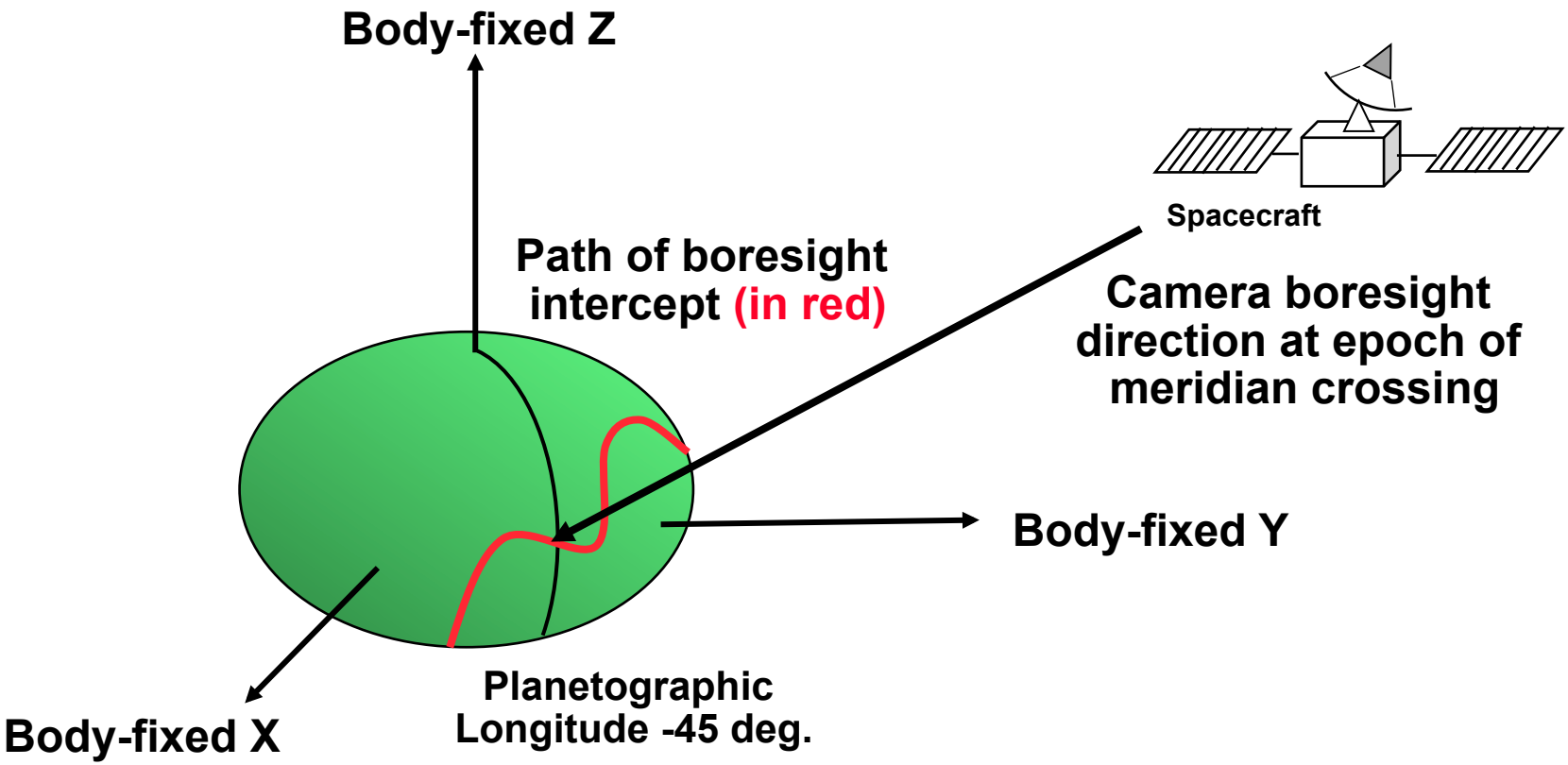
Find the time at which the planetocentric latitude of the sub-spacecraft point, using the "near point" definition, is 25 degrees.



Surface Intercept Coordinate Equality Search

Navigation and Ancillary Information Facility

Find the time at which the planetographic longitude of a given camera boresight surface intercept is -45 degrees.

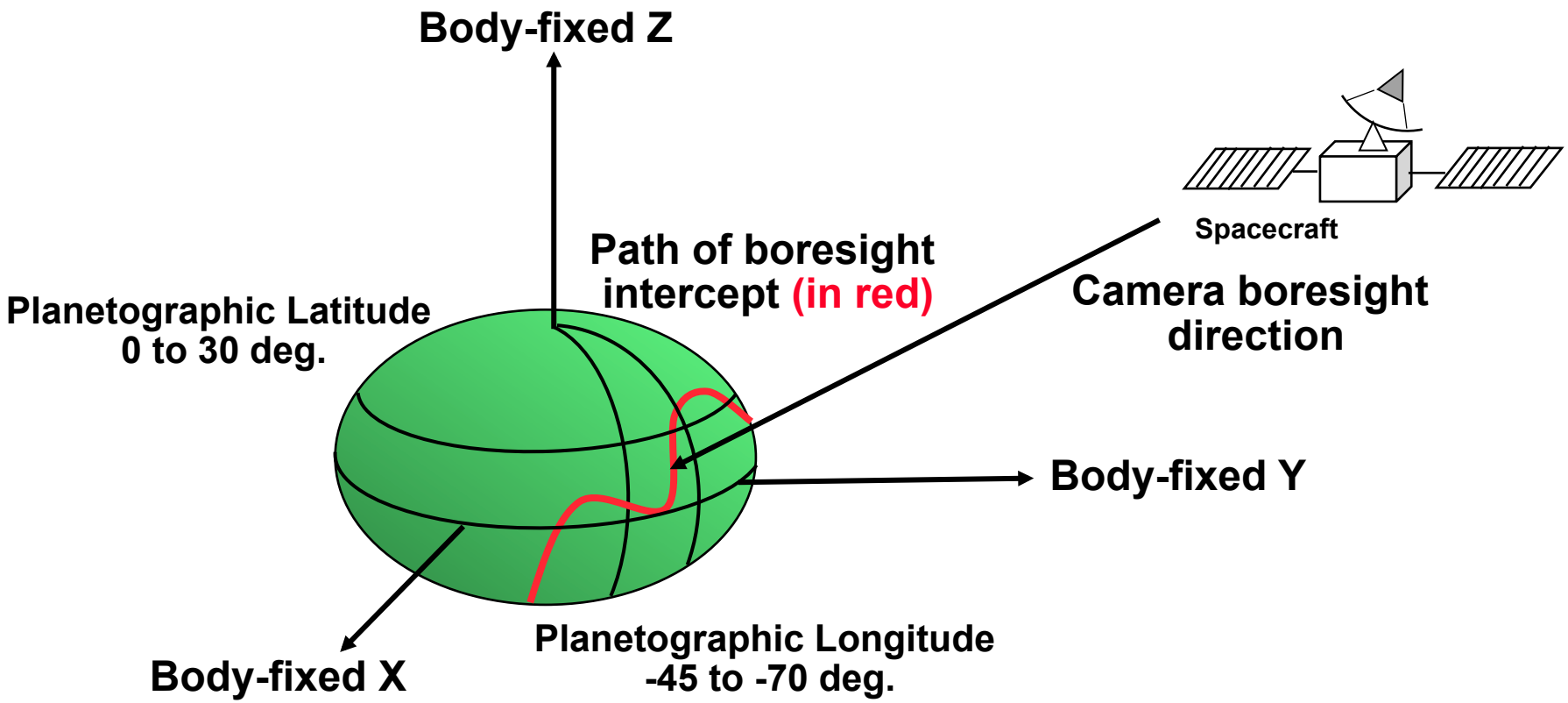


Surface Intercept "Box" Search

Navigation and Ancillary Information Facility

Find the time period when the planetographic longitude of a given 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.

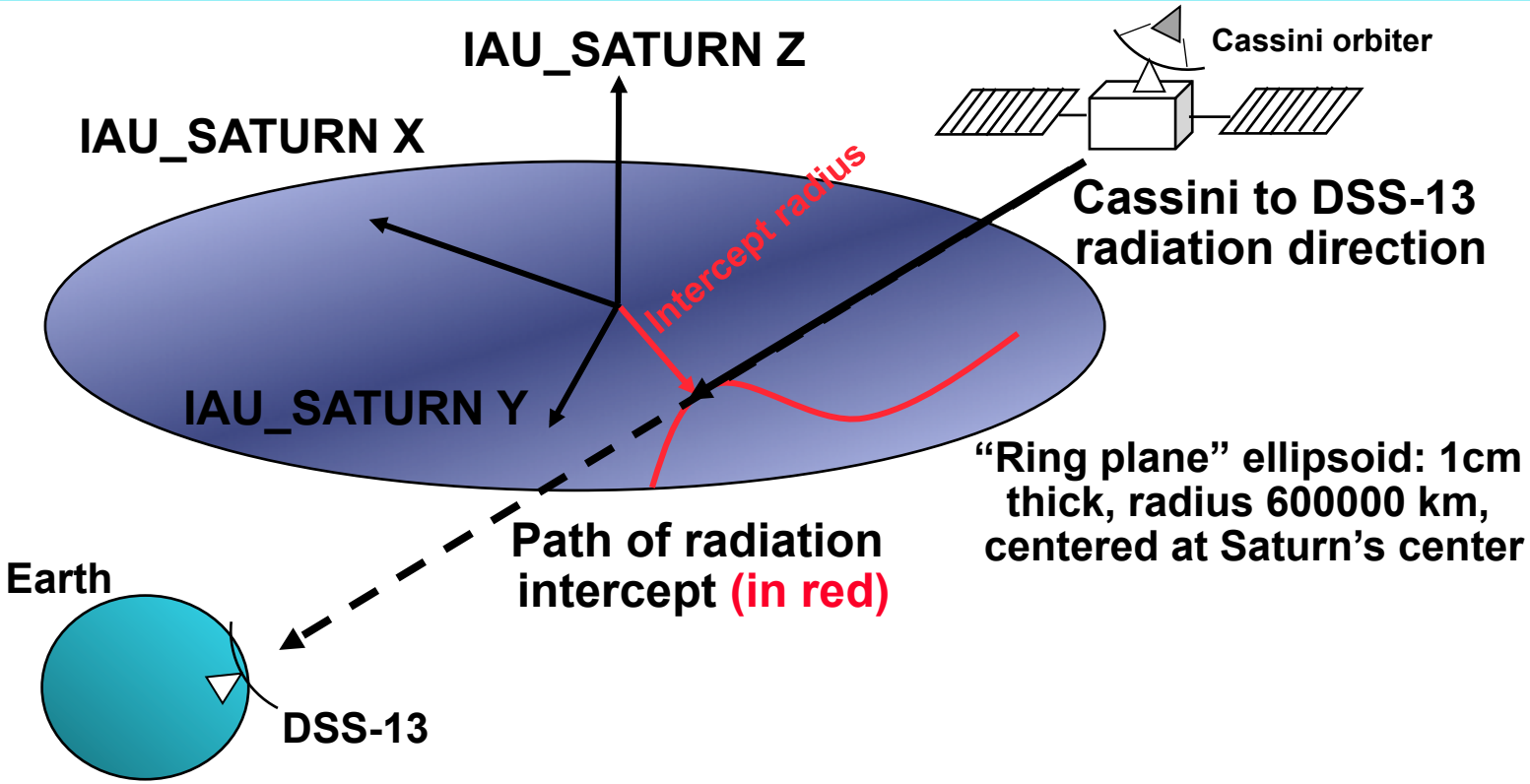


Surface Intercept Coordinate Equality Search

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.

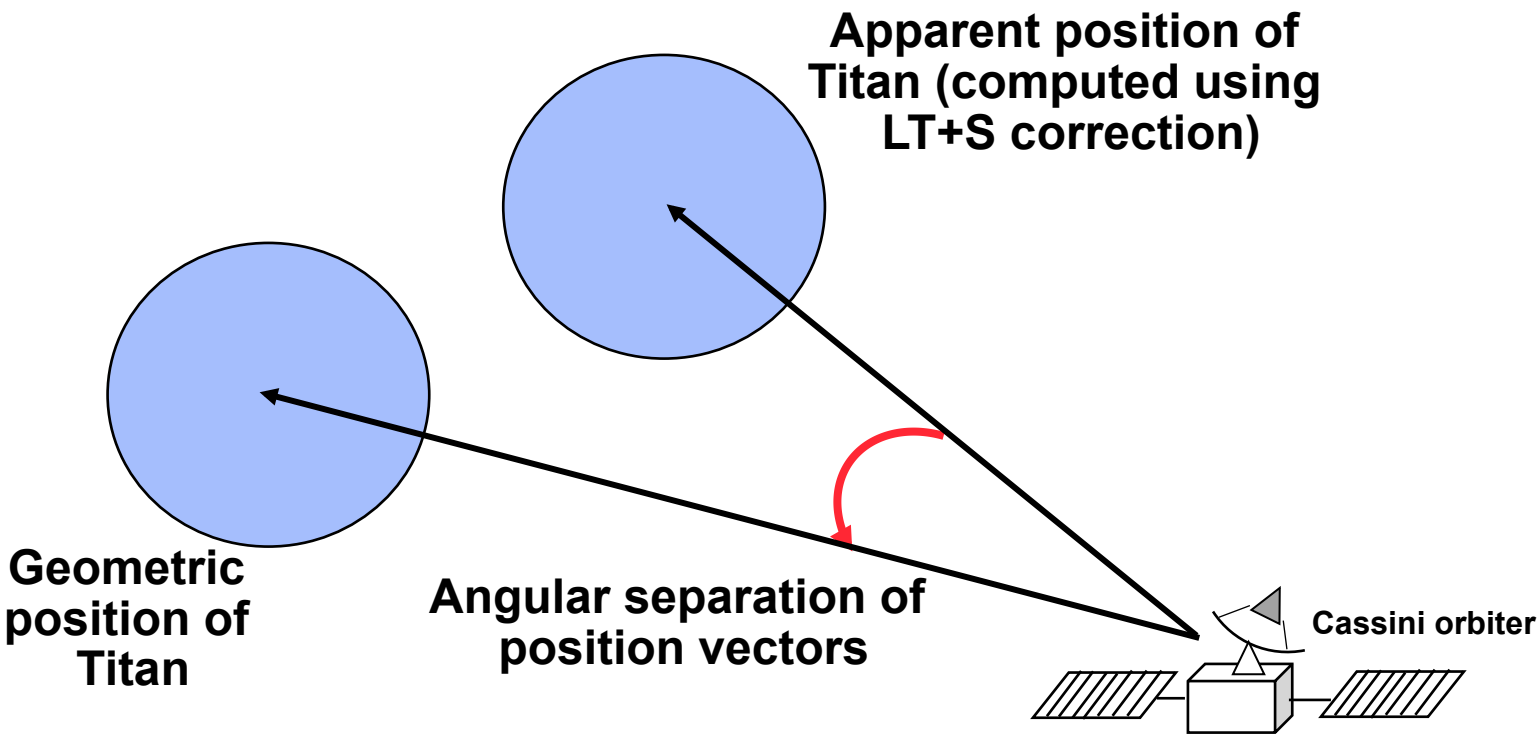
The solution requires a dynamic frame for which one axis points along the radiation path.



User-Defined Quantity Extremum Search

Navigation and Ancillary Information Facility

Find the time period when the angular separation of the geometric and apparent positions of Titan as seen from the Cassini orbiter attains an absolute maximum.

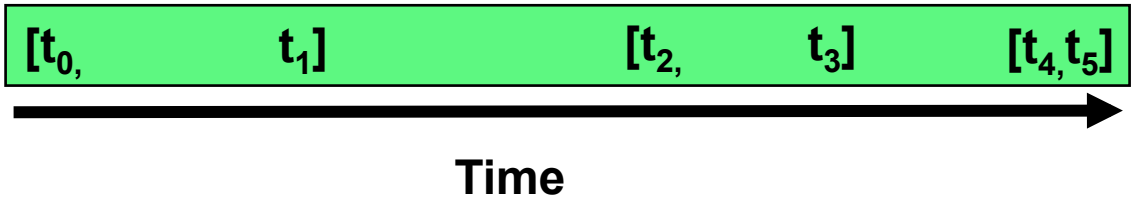


SPICE Windows

SPICE Windows -1

Navigation and Ancillary Information Facility

- **The GF system uses the SPICE "window" data structure to represent time periods.**
  - A SPICE window contains zero or more disjoint intervals arranged in ascending order. The endpoints of the intervals are double precision numbers.



- **The time period over which a GF search is conducted is called the "confinement window."**
  - Confinement windows often consist of a single time interval, but may contain multiple intervals.
- **The time period representing the result of a GF search is called the "result window."**
  - Result windows often consist of multiple intervals.
  - Result windows can contain "singleton" intervals: intervals for which the left and right endpoints coincide.
    - » Results of equality, local extremum, and unadjusted absolute extremum searches will consist of zero or more singleton intervals.

SPICE Windows -2

- SPICE windows support a form of set arithmetic:
  - SPICE provides routines to compute unions, intersections, and differences of windows, plus numerous additional window functions.
- An important, non-arithmetic SPICE window operation is contraction:
  - Shrinking each window interval by increasing the left endpoint and decreasing the right endpoint
- The result window from one search can be used as the confinement window for another.
  - 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 a fast search can be performed to produce a small confinement window for a second, slower search.
    - » See the example program “CASCADE” in gf.req.

Cascading Search -1

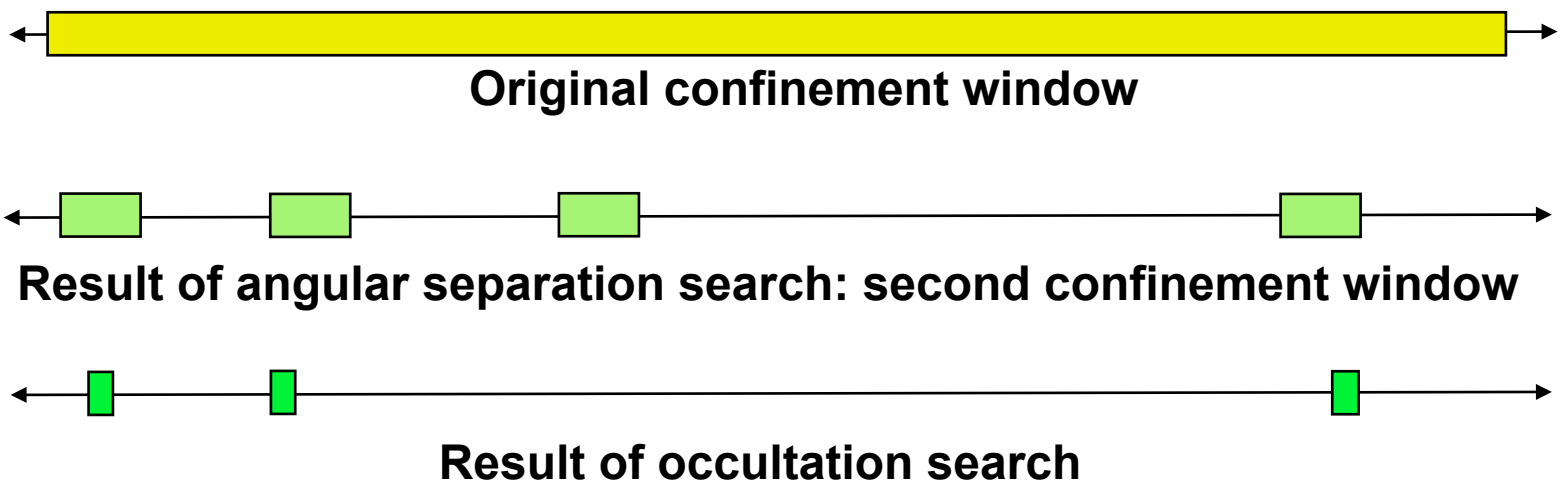
Navigation and Ancillary Information Facility

Example: accelerate a solar occultation search.

First search for times when the angular separation of the figures of the Sun and Moon, as seen from DSS-14, 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 the occultation search (on the original confinement window), the total search time is greatly reduced.



Cascading Search -2

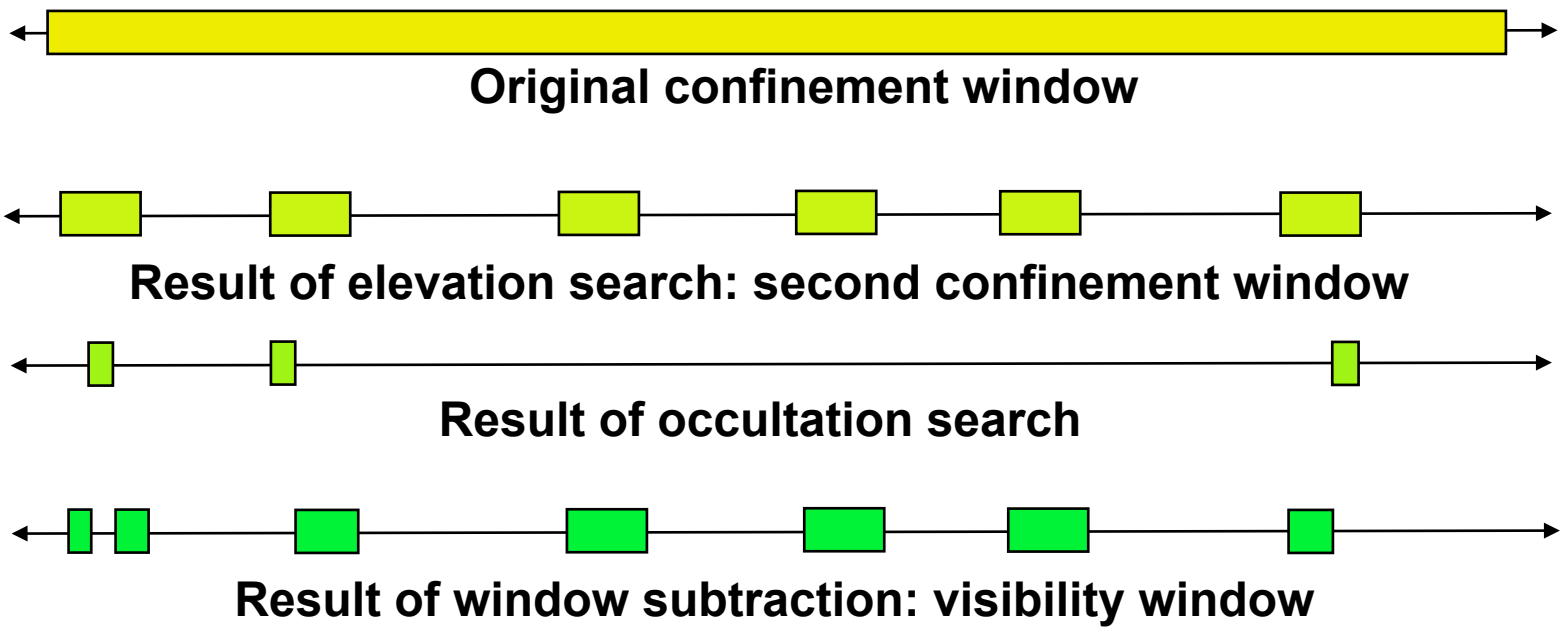
Navigation and Ancillary Information Facility

Example: find times of visibility of MRO from DSS-14.

First find times when the elevation of MRO in the DSS-14_TOPO frame is above 6 degrees.

Use the result window of the elevation search as the confinement window of a Mars occultation search.

Subtract the result of the occultation search from that of the elevation search.

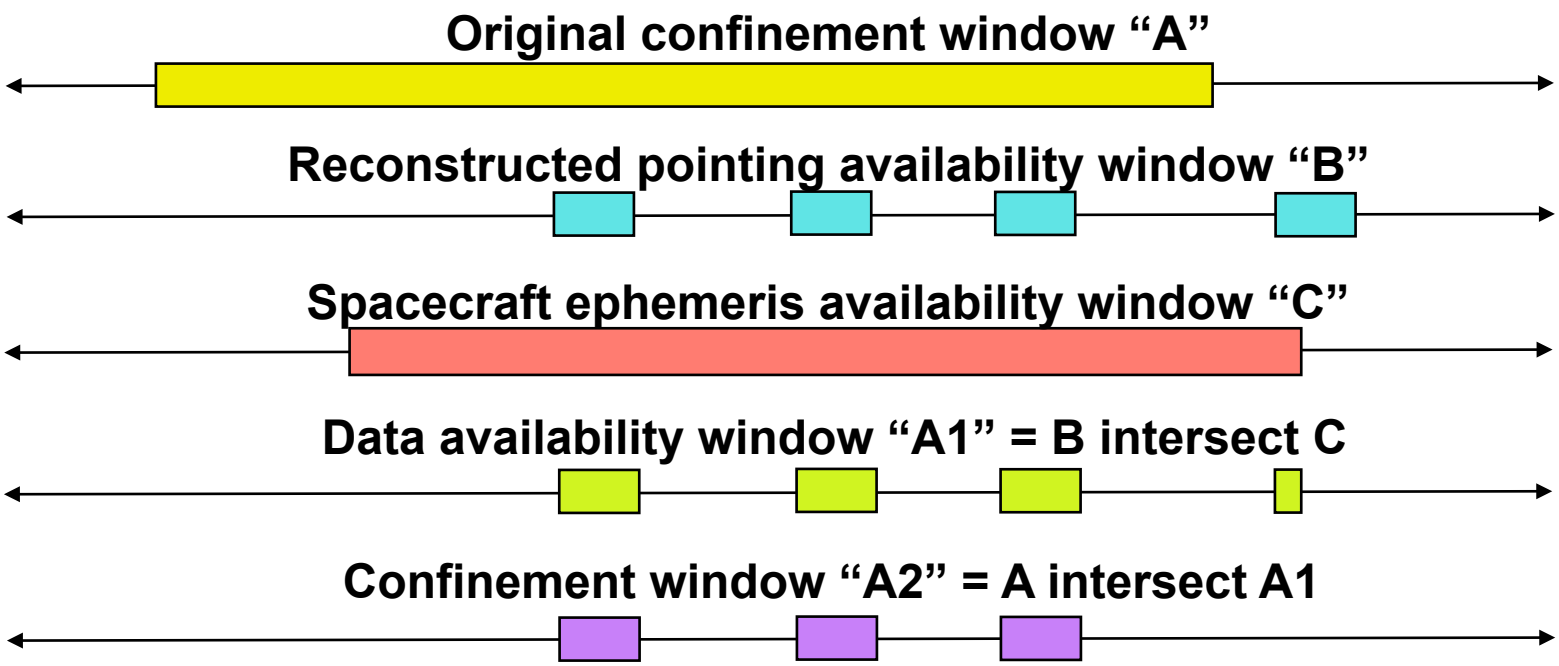


Kernel Data Availability Window

Navigation and Ancillary Information Facility

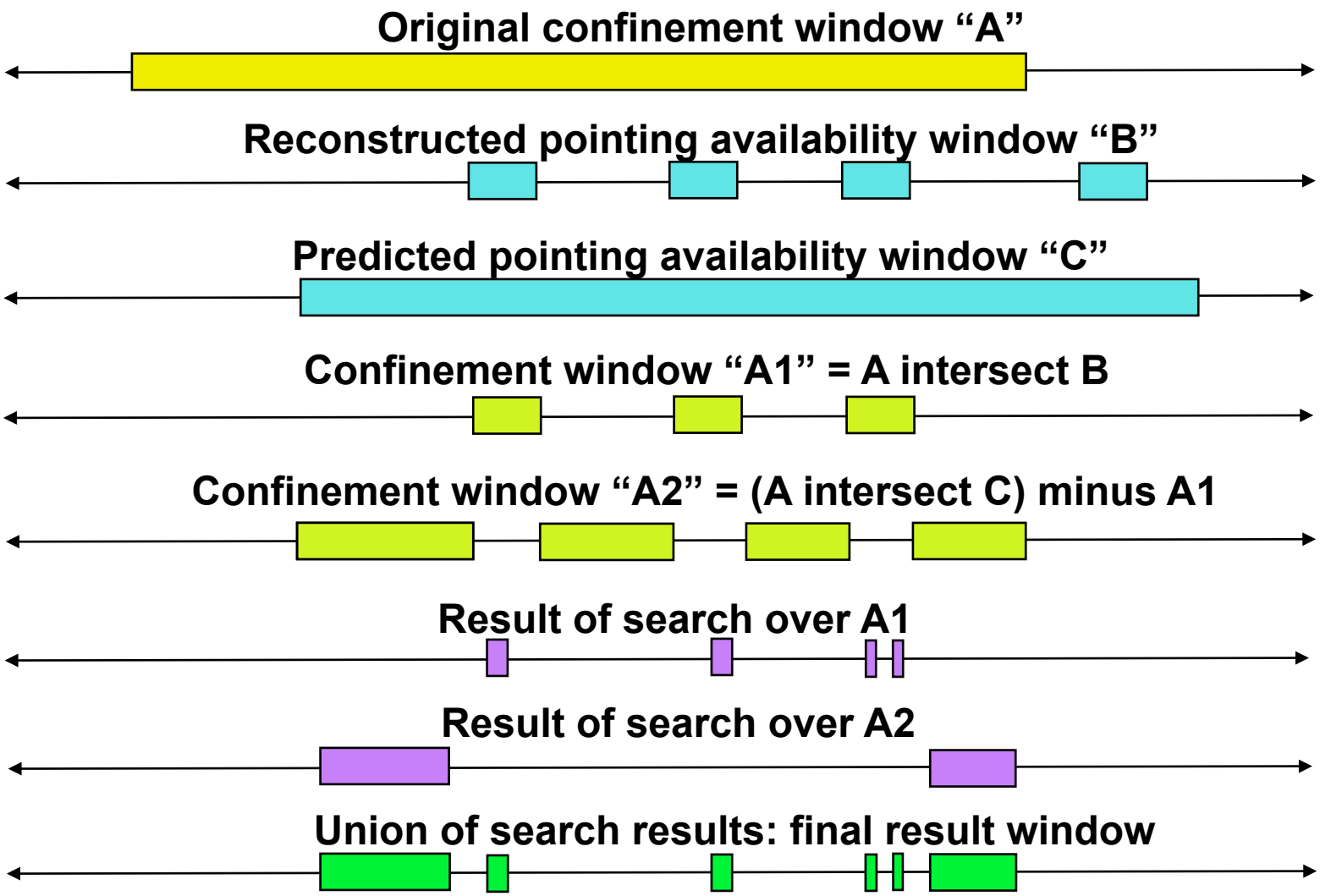
Given an initial confinement window, restrict that window to times when required CK and SPK data are available.

Use CKCOV and SPKCOV to find CK and SPK availability windows. Contract CK window slightly to avoid round-off problems. Contract SPK window by a few seconds if discrete differentiation is used by search algorithms (e.g. for acceleration or for "is function decreasing?" tests).



CK Data Availability: Divide and Conquer

Navigation and Ancillary Information Facility



Geometric Search Types and Constraints

Types of Geometric Quantities

- 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

- **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

- **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.**

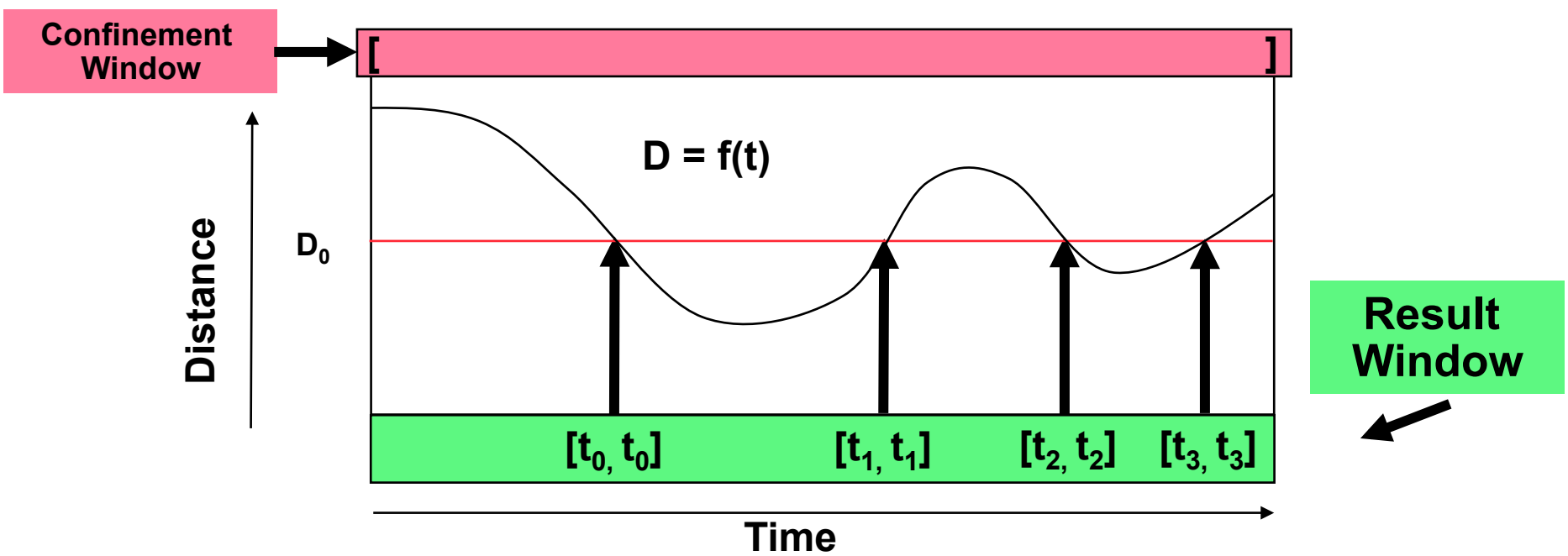
Solve Distance Equality

Navigation and Ancillary Information Facility

Find time window during which Distance = D_0

GF API input arguments defining constraint:

```
RELATE  =  '='
ADJUST  =  0.D0
REFVAL  =  D0
```



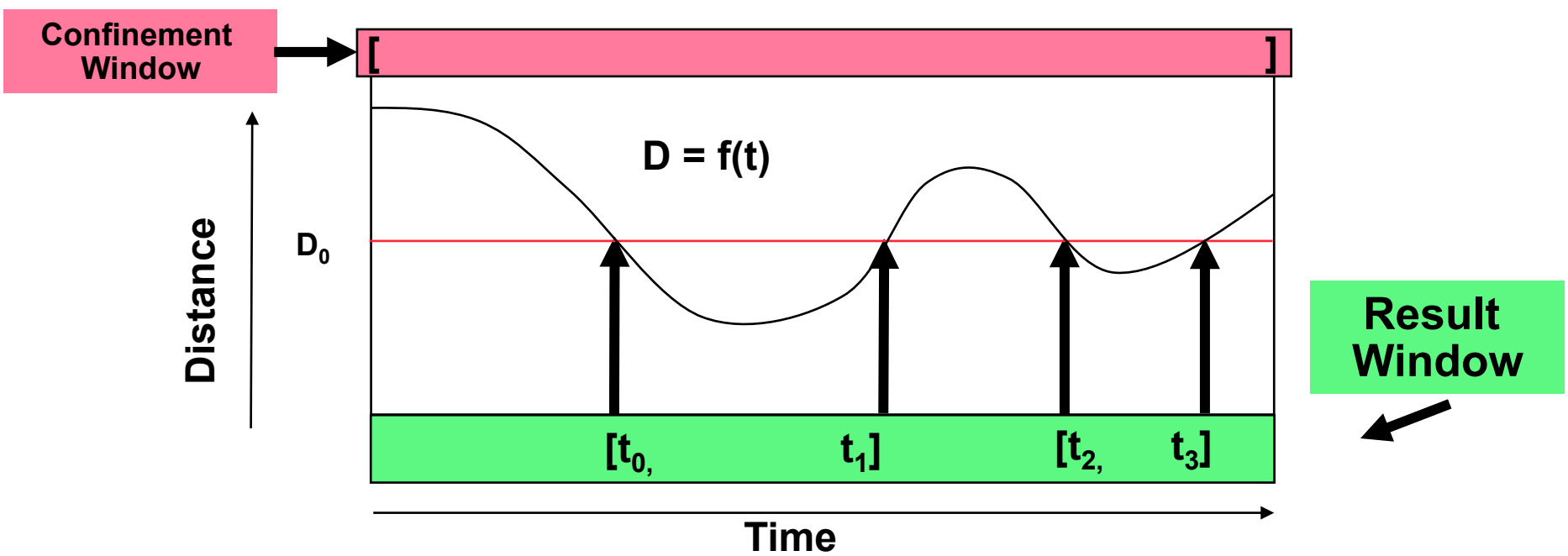
Solve Distance < Inequality

Navigation and Ancillary Information Facility

Find time window during which Distance < D_0

GF API input arguments defining constraint:

```
RELATE  =  '<'
ADJUST  =  0.D0
REFVAL  =  D0
```



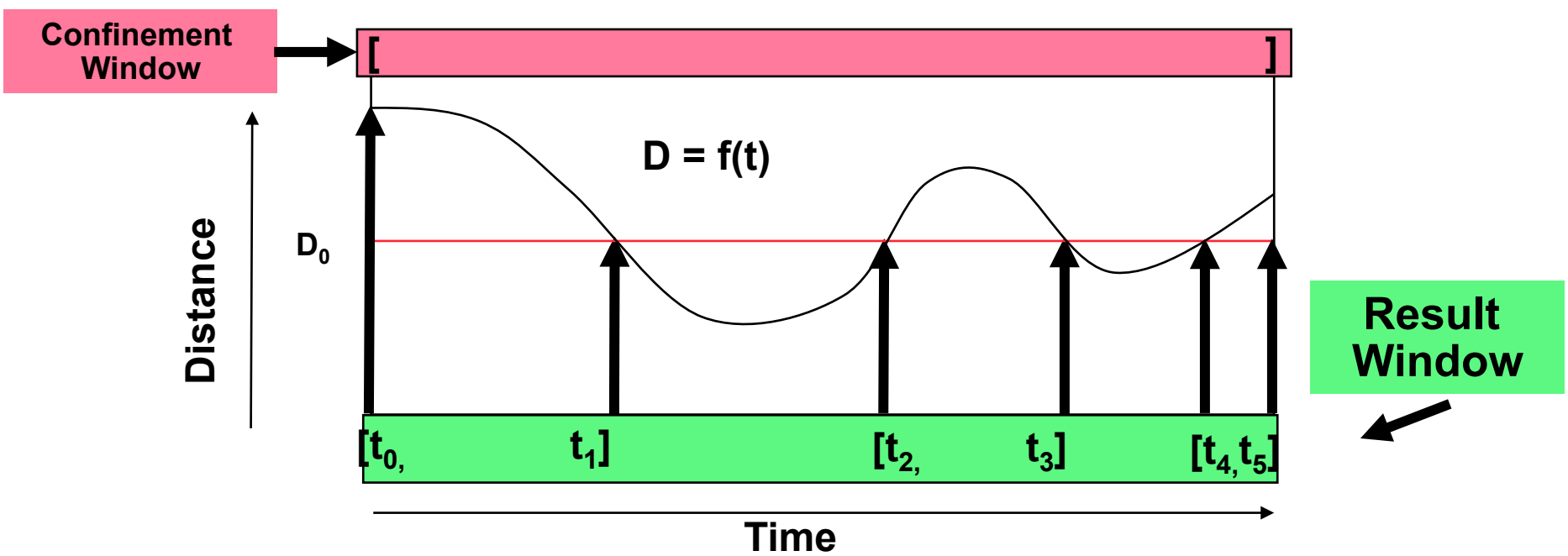
Solve Distance > Inequality

Navigation and Ancillary Information Facility

Find time window during which Distance > D_0

GF API input arguments defining constraint:

```
RELATE  =  '>'
ADJUST  =  0.D0
REFVAL  =  D0
```



Find Local Minima

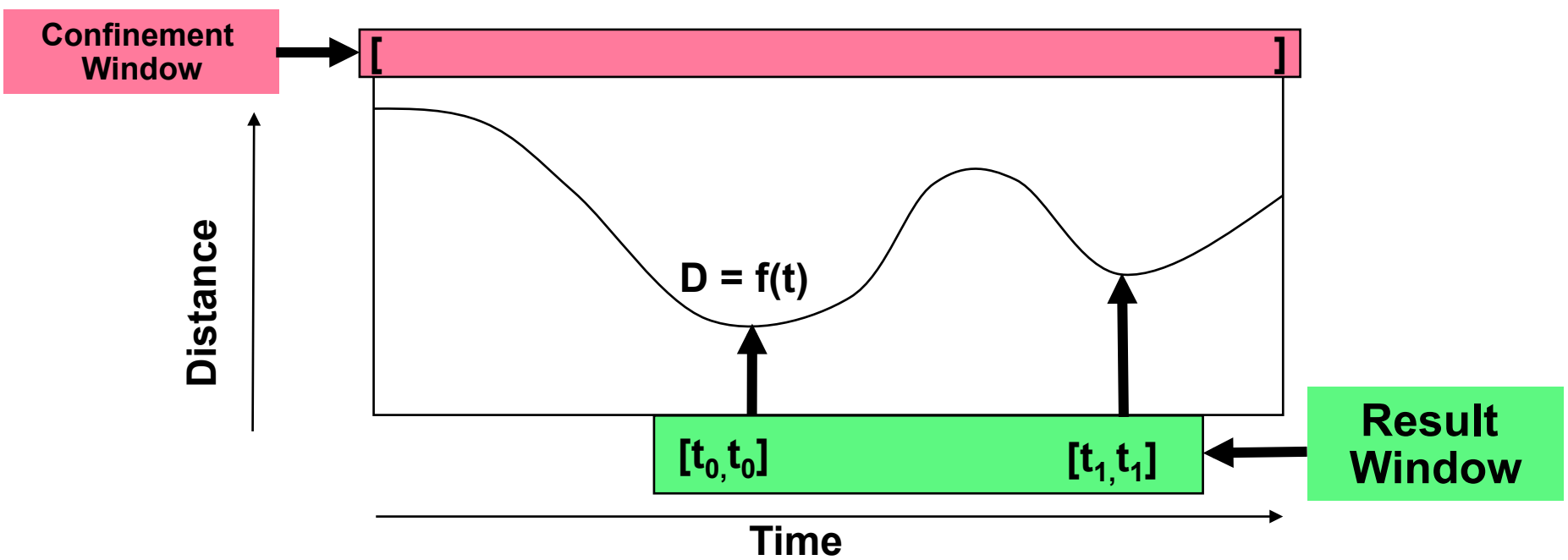
Navigation and Ancillary Information Facility

Find time window during which Distance is a local minimum.

GF API input arguments defining constraint:

```
RELATE = 'LOCMIN'
ADJUST = 0.D0
REFVAL = 0.D0
```

(REFVAL is not used in this case but should be initialized for portability)



Find Local Maxima

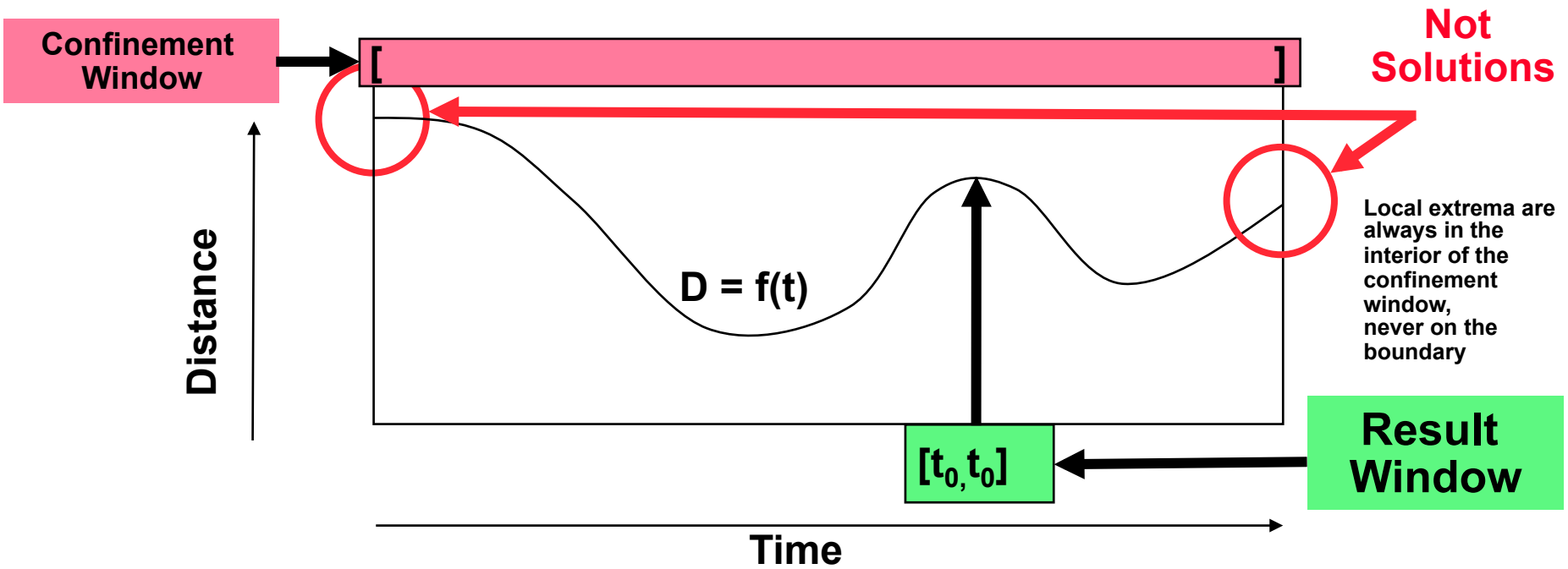
Navigation and Ancillary Information Facility

Find time window during which Distance is a local maximum.

GF API input arguments defining constraint:

```
RELATE = 'LOCMAX'
ADJUST = 0.D0
REFVAL = 0.D0
```

(REFVAL is not used in this case but should be initialized for portability)



Find Absolute Minimum

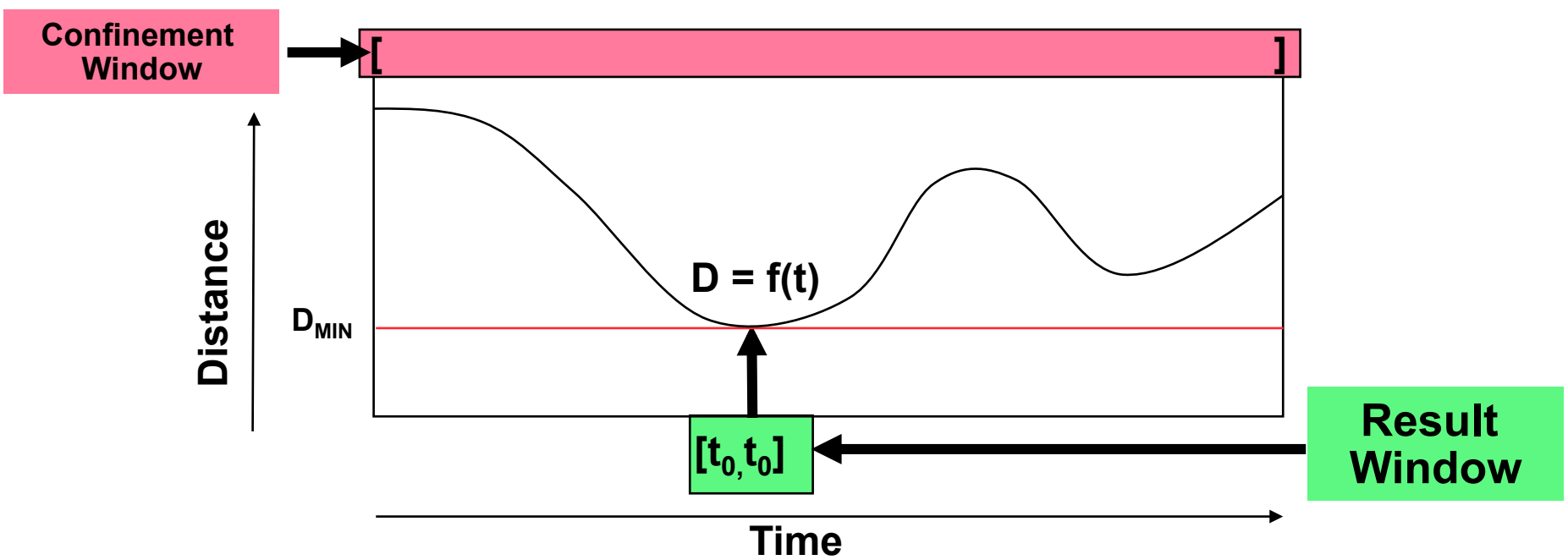
Navigation and Ancillary Information Facility

Find the time at which Distance is an absolute minimum.

GF API input arguments defining constraint:

```
RELATE = 'ABSMIN'
ADJUST = 0.D0
REFVAL = 0.D0
```

(REFVAL is not used in this case but should be initialized for portability)



Find Absolute Maximum

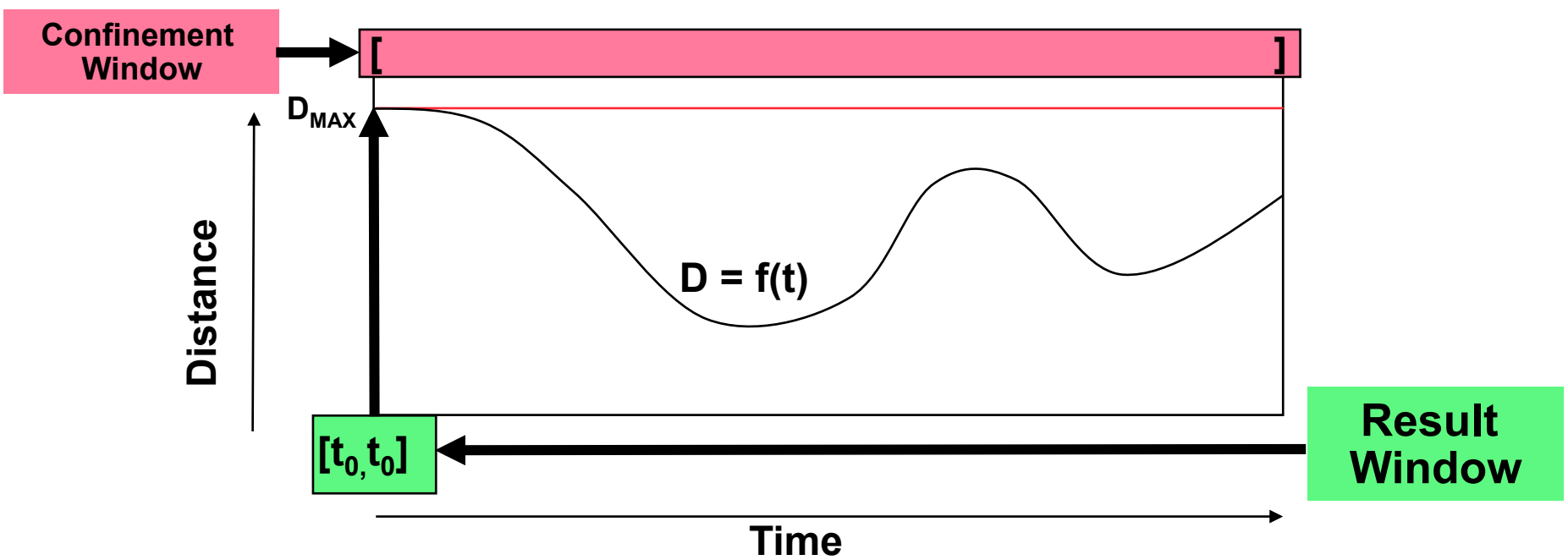
Navigation and Ancillary Information Facility

Find the time at which Distance is an absolute maximum.

GF API input arguments defining constraint:

```
RELATE = 'ABSMAX'
ADJUST = 0.D0
REFVAL = 0.D0
```

(REFVAL is not used in this case but should be initialized for portability)



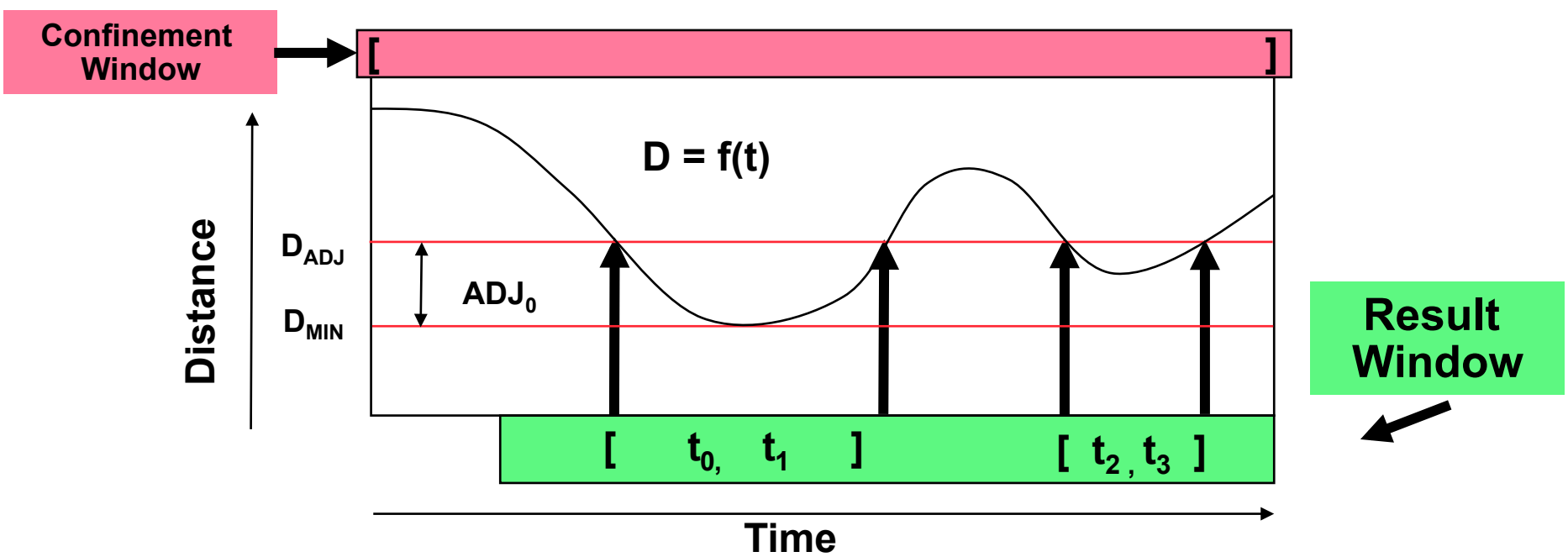
Find Adjusted Absolute Minimum

Navigation and Ancillary Information Facility

Find time window during which Distance < $D_{ADJ} = D_{MIN} + ADJ_0$

GF API input arguments defining constraint:

```
RELATE = 'ABSMIN'
ADJUST = ADJ0     (ADJ0 > 0)
REFVAL = 0.D0     (REFVAL is not used in this case but should be initialized for portability)
```



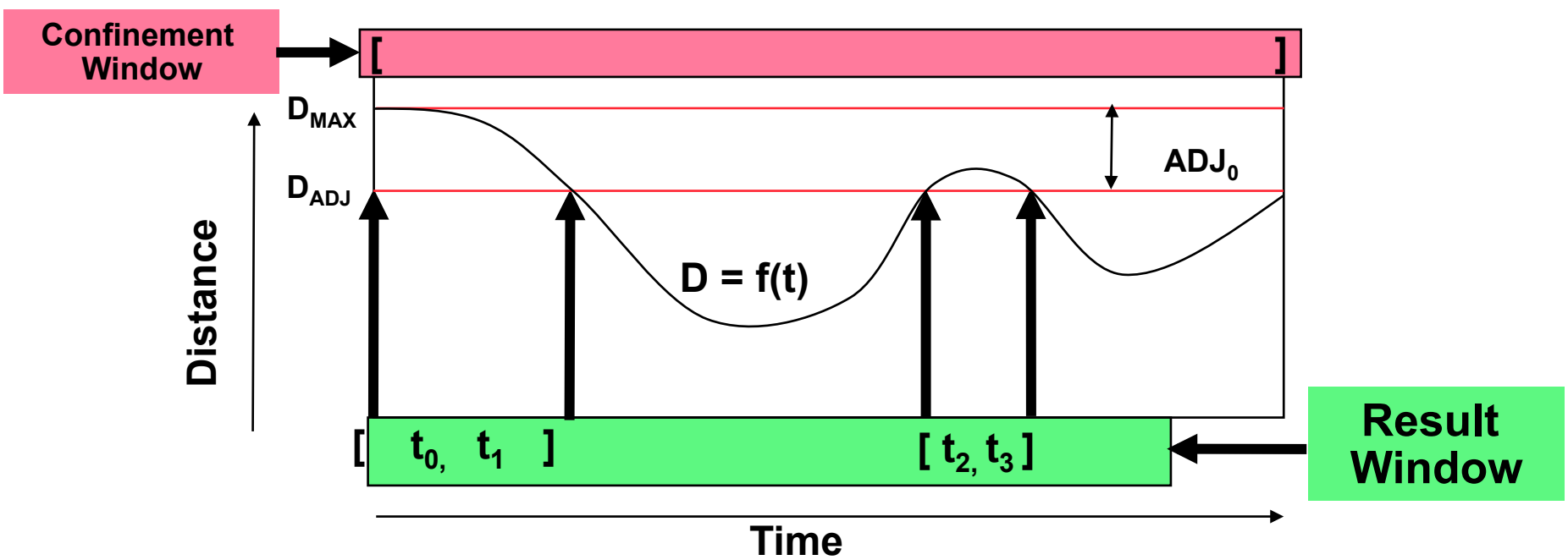
Find Adjusted Absolute Maximum

Navigation and Ancillary Information Facility

Find time window during 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)
```



Plans for further GF Development

Plans

- NAIF plans to add the following GF capability in a future SPICE Toolkit release
  - Eclipse search using extended bodies
- Additional GF capabilities might be added

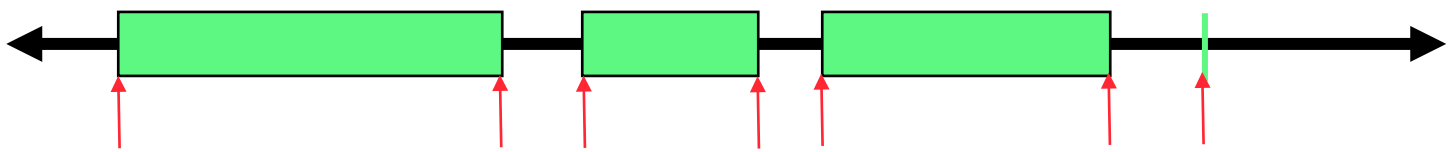
Backup

- Root Finding
- Workspace
- API Example: GFDIST

Root Finding

Root Finding

- 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

- 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.

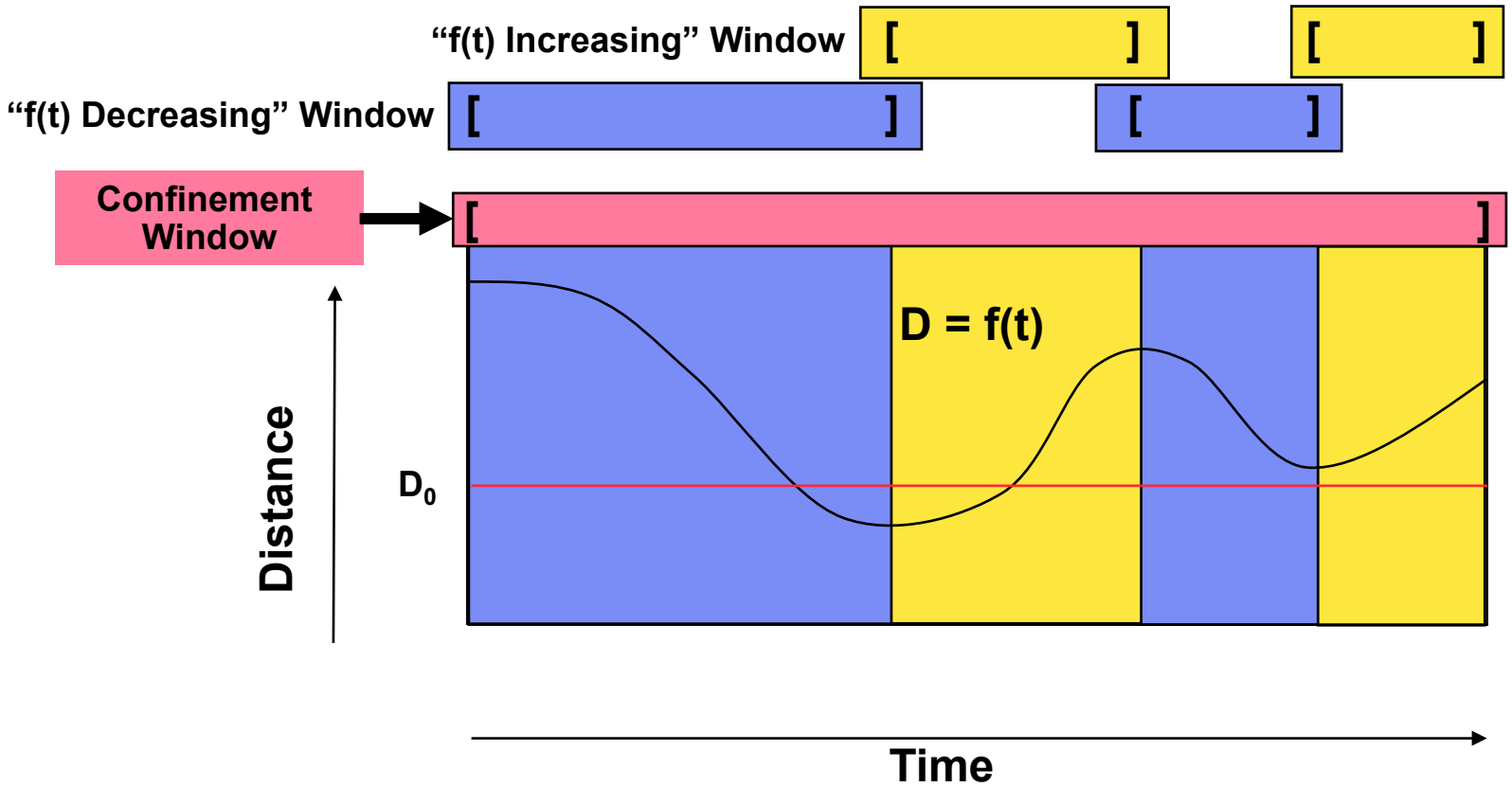
Monotone Windows -1

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.

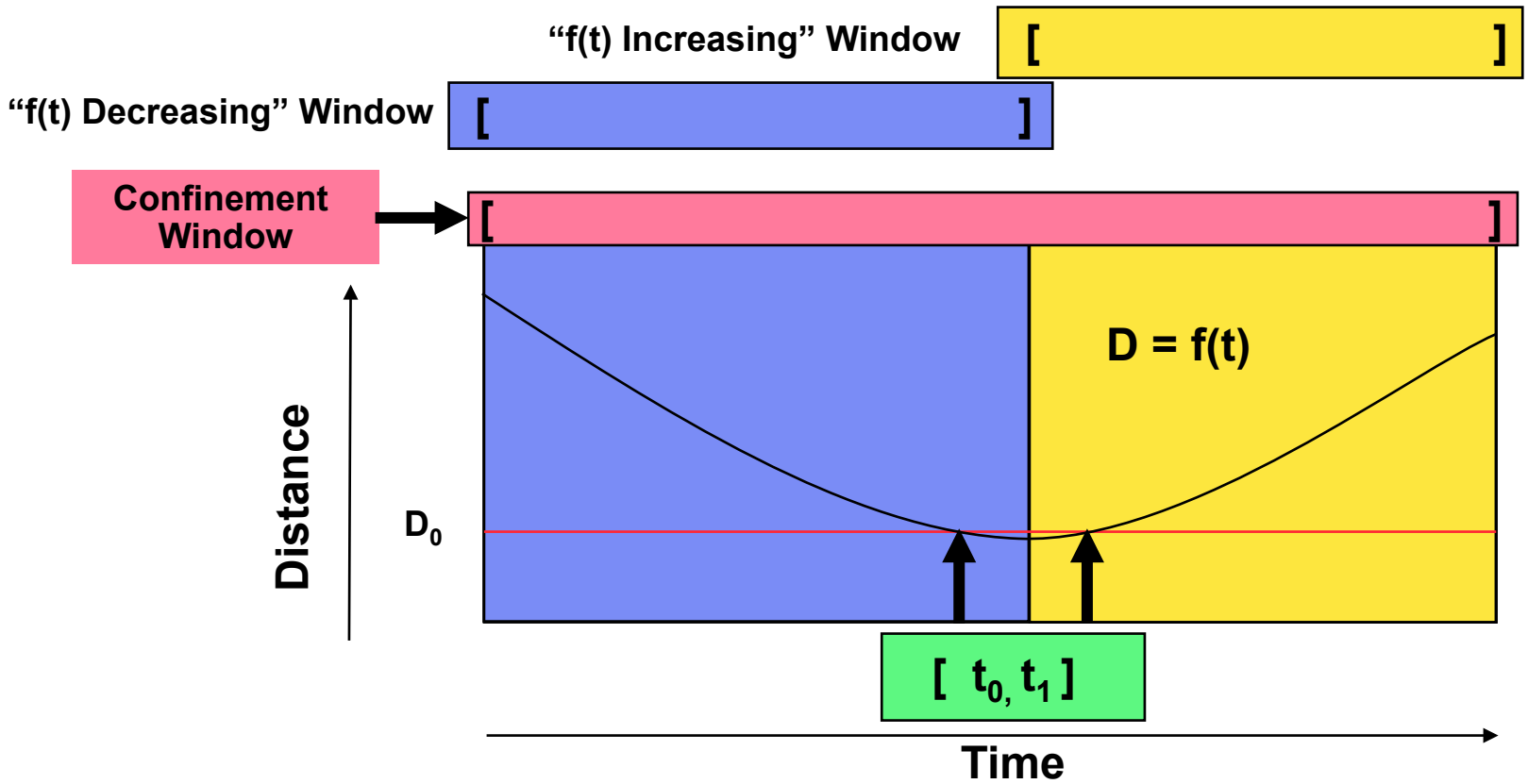


Monotone Windows-2

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.

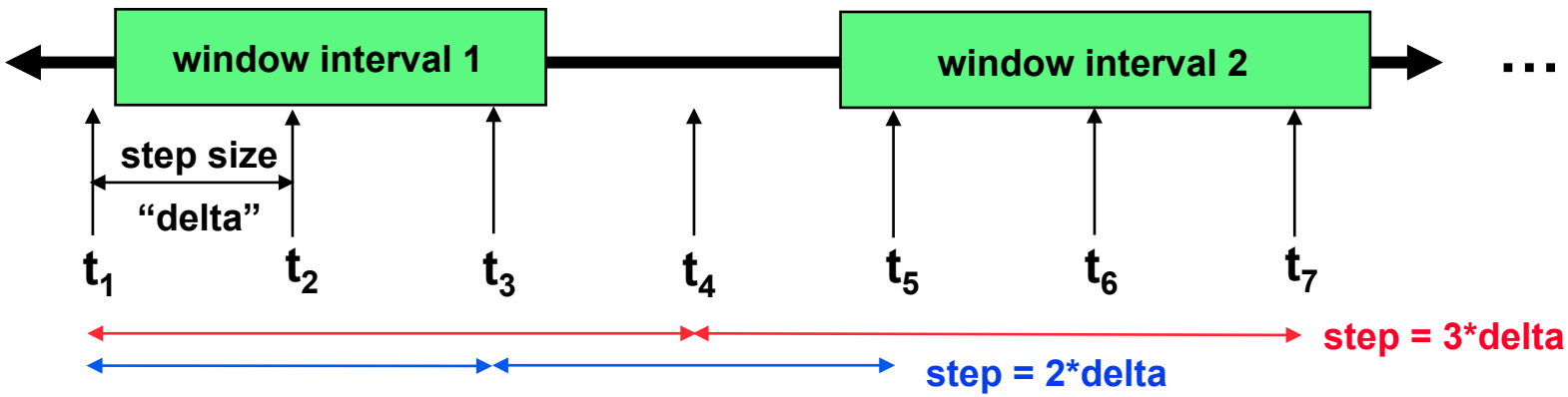
For example, consider the search for times when $D < D_0$. The result window consists of the interval $[t_0, t_1]$ shown below.



Bracketing Interval Endpoints

Navigation and Ancillary Information Facility

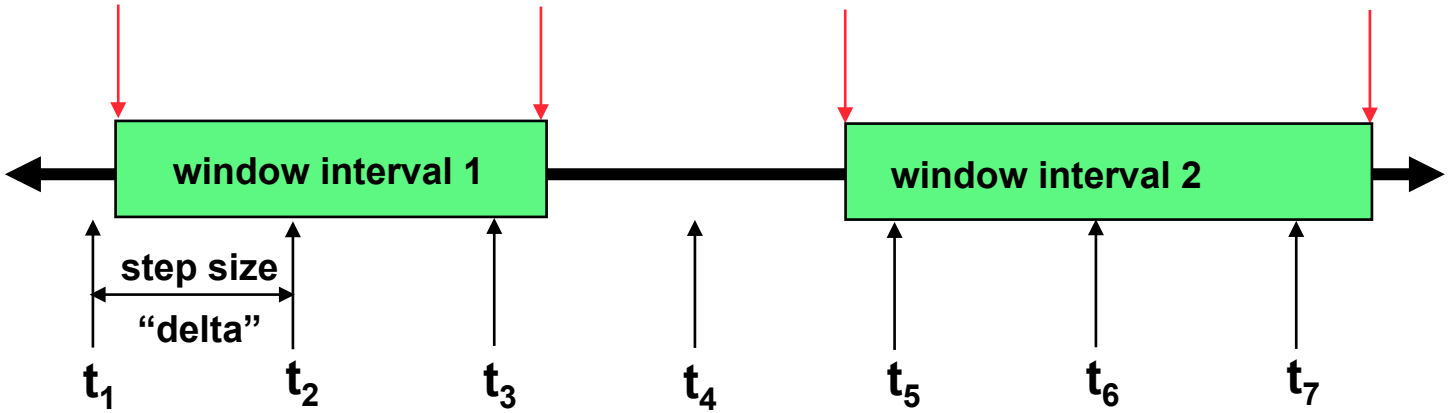
- In the diagram below, the green boxes denote intervals of a window.
- The start of the first interval is bracketed by the first and second times; the end of the first interval is bracketed by the third and fourth times. The start of the second interval is bracketed by the fourth and fifth times.



- 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

- 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.

Workspace

Specifying Workspace Dimensions

- 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.

API Example: GFDIST

Solve for Distance Constraints

API Example: GFDIST

- **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

- **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'
  ...
INTEGER            LBCELL
PARAMETER        ( LBCELL =  -5 )
INTEGER            MAXWIN
PARAMETER        ( MAXWIN = 200000 )
DOUBLE PRECISION   CNFINE  ( LBCELL : MAXWIN )
DOUBLE PRECISION   RESULT ( LBCELL : MAXWIN )
DOUBLE PRECISION   WORK    ( LBCELL : MAXWIN, NWDIST )
```

Include GF parameters such as NWDIST

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

C constant and variable declarations

Declare confinement and result windows, as well as size of workspace.

```
#include "SpiceUsr.h"
  ...

#define  NINTVL        100000
#define  MAXWIN        ( 2 * NINTVL )

SPICEDOUBLE_CELL   ( cnfine,  MAXWIN );
SPICEDOUBLE_CELL   ( result,  MAXWIN );
```

Include CSPICE macro, typedef, and prototype declarations

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.

These macro calls declare CSPICE window structures and set their maximum sizes.

IDL Set-up

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

Initialization…typically done once per program execution

Tell your program which SPICE files to use ("loading" files)

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

Better yet, replace these two calls with a single call to a "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

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 )
```

inputs Input/output output

Execute the Search -2

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 WNCOND to shrink the intervals of the result window slightly, so the constraint is met on the entire result window.
    - » Caution: DON'T use WNCOND for minimum, maximum, or equality searches---the result window will disappear! (WNCOND is ok for adjusted absolute extrema search results, though, since the result intervals are not singletons.)
    - » Caution: using WNCOND may not be desirable if the window is an intermediate result: subsequent, derived results might be made less accurate.

Arguments of GFDIST - 1

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 "Reading an 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

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

Exception Handling

January 2012

Topics

- 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

- **Run time error conditions**
  - 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, square root of a negative number.
  - Environment problems
    - » Insufficient disk space for output files.
    - » Lack of required read/write permission/privileges.

Exceptions Are… - 2

- **Valid but unusual conditions**
  - Examples are:
    - » 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 parsers**
  - Examples are:
    - » Invalid SQL query.
    - » Invalid string representing number (borderline case).
  - 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."

Language Dependencies

- 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 <u>detection</u>.
  - 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

- **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

- **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

- **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

- **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.

--continues--

Error Handling Actions - 2

- 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

- **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

- **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, selecting TRACEBACK has no effect.
  - errprt_c ("set", LEN, "none, short, traceback");

Customize Error Handling - 2

- **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

- **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

- Use QCKTRC or TRCDEP and TRCNAM to retrieve traceback message
  - In CSPICE, only f2c'd versions of these routines are available
- 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 handle error condition; a reset is automatically performed by Icy

Signal Errors - 1

- **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

- **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, parsable, 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

- 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

Toolkit Applications

January 2012

Toolkit Applications

Toolkit applications create or manipulate kernels, or perform other functions such as time conversion.

Each of the following applications is included in the generic Toolkits.

- **Time conversion tool: *chronos***
- **SPK generation tool: *mkspk***
- **SPK merge and subset tool: *spkmerge***
- **SPK comparison tool: *spkdiff***
- **CK generation tool: *msopck***
- **Frame comparison tool: *frmdiff***
- **Kernel summary tools: *brief, ckbrief, spacit***
- **Comments manipulation tools: *commnt, spacit***
- **File format converters: *tobin, toxfr***

CHRONOS

Navigation and Ancillary Information Facility

chronos **is an application that 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
```

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

- *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 <cental body ID>]
                [-LANDINGTIME <UTC time of the landing>]
                [-SOL1INDEX <index of the first SOL>]
                [-NOLABEL]
                [-TRACE]
```

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

- *mkspk* may be used to generate an SPK file from any of several types of data, such as discrete states, classic elements, and 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

- **The contents of an SPK file or set of SPK files may be merged or subsetted using *spkmerge***
  - **Extract an interval of <u>time</u> of interest from a single SPK file or a set of SPK files.**
  - **Extract data for one or more <u>objects</u> 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

- **SPK files created with *spkmerge* have no overlapping ephemeris data. The order in which the source files are specified determines precedence when sources 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.**

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

- *spkdiff* is a command line program for comparing trajectories provided by two SPK files
- *spkdiff* compares SPKs 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 one SPK file, then computing another set of geometric states for the same or different body, center, and frame at the same times using the other SPK file, 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.

SPKDIFF - Usage

Navigation and Ancillary Information Facility

Terminal Window

```
$ spkdiff

   spkdiff computes differences between geometric states obtained from
   two SPK files and either displays these differences or shows statistics
   about them (see the User's Guide for more details.) The program usage is:

      % spkdiff [options] <first SPK file> <second SPK file>

   Options are shown below. Order and case of keys are not significant.
   Values must be space-separated from keys, i.e. '-n 10', not '-n10'.

      -b1 <first body name or ID>
      -c1 <first center name or ID>
      -r1 <first reference frame name>
      -b2 <second body name or ID>
      -c2 <second center name or ID>
      -r2 <second reference frame name>
      -k  <other kernel file name(s)>
      -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)>
      -t  <report type: basic|stats|dump|dumpvf (default: basic)>
```

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

- *msopck* is a program for making CK files from orientation provided in a text file as a time tagged, space-delimited table
  - has a simple command line interface
  - requires all setups to be provided in a setup file that follows the SPICE text kernel syntax
  - can process quaternions (SPICE and non-SPICE flavors), Euler angles, or matrixes, tagged with UTC, SCLK, or ET
  - for more details see the “Making a CK File” Tutorial

FRMDIFF

- *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 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 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, matrixes, 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_TIMOUT>
      -t  <report: basic|stats|dumpaa|dumpm|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)
```

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 5.0000000000000 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 10.000000000000 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 seco...
#    to   '2009 JAN 10 00:01:06.184' TDB (284817666.18419 TDB seco...
#
# 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....
```

Kernel Summary Applications

The contents of binary kernels can be summarized with the 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.**
- ***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.**

BRIEF

- *brief* is a simple command line program for summarizing the contents of SPK or binary PCK files
- the files to be summarized can 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 motions along with the bodies
  - treating all input files as if they were a single file
  - displaying summary only for files covering a specified time or time range or containing data for a specified body
  - displaying summary in tabular format or grouped by coverage
  - and many others ...

BRIEF - Usage

Navigation and Ancillary Information Facility

Terminal Window

```
$ brief

BRIEF -- Version 3.0.0, January 14, 2008 -- Toolkit Version N0063

   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
```

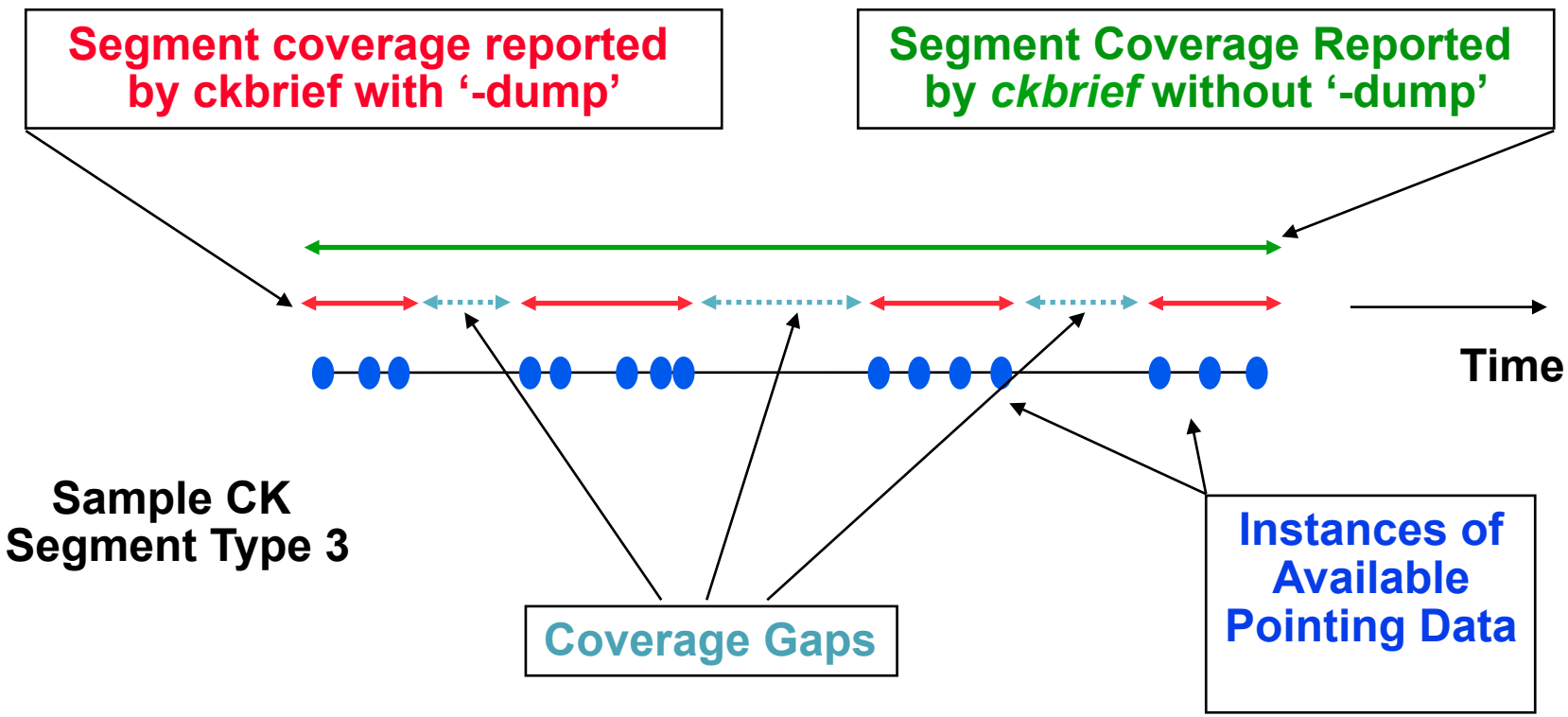
CKBRIEF

- *ckbrief* is a simple command line program for summarizing the contents of CK files
- the files to be summarized can 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 as date UTC, DOY UTC, SCLK, or encoded SCLK (default time format is calendar ET)
    - » to display times as ET, UTC, or SCLK, an LSK file and 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
  - 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 to get a complete list of continuous coverage intervals for each segment



CKBRIEF – Usage

Navigation and Ancillary Information Facility

Terminal Window

```
$ ckbrief

CKBRIEF -- Version 5.0.0, February 11, 2009 -- Toolkit Version N0063

   CKBRIEF is a command-line utility program that displays a summary for
   one or more binary CK files. The program usage is:

      % ckbrief [-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
      -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

```
$ ckbrief -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


$ ckbrief -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
```

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
    - » It displays short menus where you choose the action desired
- *spacit* may also be used to manage comments, and to convert between binary and transfer format

Comment Manipulation Tools

- **Every kernel should contain metadata – called "comments" – describing the file contents, intended usage, etc.**
- **In binary kernels – SPKs, CKs, binary PCKs, 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).**

COMMNT

- *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 to be 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 avail
able from NAIF or from the author: Dr. Myles Standish of JPL's Solar System Dy
namics Group. Because this memo was produced using the TeX processor and inclu
des 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; it must be downloaded from the NAIF website

* DAF-based binary kernels are SPK, CK and binary PCK

Non-Toolkit Applications

January 2012

Summary

- **NAIF makes available a set of applications *not* included in the generic Toolkits. This set includes programs for:**
  - making, modifying, validating, inspecting, and analyzing SPK files:
    - » *pinpoint, dafcat, bspidmod, dafmod, spy*
  - making and modifying CK files
    - » *prediCkt, ckslicer, ckspanit, dafcat, cksmrg, dafmod*
  - making SCLK files
    - » *makclk*
  - computing derived quantities
    - » *orbnum, optics, spy*
  - determining SPICE kernel type and binary format
    - » *archtype, bff*
  - converting between binary and text kernel formats
    - » *bingo*
- **Executables and User's Guides for selected computer environments are available from the NAIF server at:**
  - http://naif.jpl.nasa.gov/naif/utilities.html

PINPOINT

Navigation and Ancillary Information Facility

- *pinpoint* is a program for creating SPK files and topocentric frames FK files for objects whose 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
  - Works on SPKs, CKs, and binary PCKs
    - » will not merge different types of kernels together, i.e. 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
```

BSPIDMOD

Navigation and Ancillary Information Facility

- *bspidmod* is a program for altering the object IDs in a binary SPK file
  - can be used to modify IDs in an SPK file(s) produced with a “bogus” spacecraft ID (or a simulation spacecraft ID)
  - can be used to replace “good” 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)
- *bspidmod* is a command line program with 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 the in 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 utility for validating, inspecting, and analyzing SPK files
- Spy can:
  - Dump SPK file contents
    - » Data
    - » Summary information
    - » Comment area
    - » Bookkeeping information
  - Sample data from a set of loaded kernels
    - » Sample position, distance, velocity, derived velocity, speed, acceleration, acceleration magnitude, osculating elements
  - Check SPK files
    - » Validate SPK structure
    - » Check sampled data for bounds violations
    - » Locate invalid double precision numbers
  - 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
=================================================================
```

PREDICKT

Navigation and Ancillary Information Facility

- *prediCkt* is a program for making CK files from a set of orientation specification rules and schedules defining when these rules are to be applied
  - has a simple command line interface
  - requires orientation and schedule specification to be provided in a setup file that follows the SPICE text kernel syntax
  - requires all supporting kernels -- SPK, PCK, etc -- to be loaded using a meta kernel
  - for more details see “Making CK Tutorial”

CKSLICER

- *ckslicer* is a program for subsetting a CK file
- *ckslicer* is a command line program with 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 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 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* is a command line program with 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*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

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

- *cksmrg* is a program for merging data from two or more uniform CK segments (same ID, base frame and type) provided in a single CK file
- *cksmrg* is used for eliminating gaps between segments (that cannot be removed by *ckspanit*) and removing duplicate data points contained in different segments
- *cksmrg* is a command line program with 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 CKs from different sources, nor should it be used to merge overlapping predict CKs

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

- *makclk* is a program for converting a SCLKSCET file to an SCLK kernel
  - SCLKSCET 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_____     ________SCET0________ _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)

Terminal Window

```
$ more flc_template.tsc
KPL/SCLK
   \begindata
      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 = ( 0.0000000000000E+00 )
      SCLK_PARTITION_END_77   = ( 1.0995116277750E+12 )
      SCLK01_COEFFICIENTS_77  = ( 0.E+00 0.E+00 1.E+00 )
   \begintext

$ 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 start/stop times and orbit numbers 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 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
- If -pref, -num or -file have not been provided, the program will prompt for them
- If -d is not specified the program will prompt for the time span within which the search for orbit events is to be performed

ORBNUM Example

Navigation and Ancillary Information Facility

Terminal Window

```
$ more mex_orbnum.setup
\begindata
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'
\begintext

$ 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 utility 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 a command line program 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 two 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

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', 'FK'
- 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

```
$ archtype 020514_SE_SAT105.bsp
DAF SPK

$ archtype 04135_04171pc_psiv2.bc
DAF CK

$ archtype cas00084.tsc
KPL SCLK

$ archtype cas_v37.tf
KPL FK

$ archtype cpck05Mar2004.tpc
KPL PCK

$ archtype naif0008.tls
KPL LSK

$ archtype .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 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

SPICE Toolkit
Common Problems

January 2012

Topics

- Prevention
- The “Common Problems Required Reading” document
- Reporting a Problem to NAIF

Prevention - 1

- **Get well setup to use SPICE**
  - 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 application to the latest Toolkit
  - Read the pertinent documentation
    - » Tutorials, module headers, Required Reading technical reference documents, comments inside kernels
  - Get the correct (usually latest) kernel files
    - » Verify that coverage and intended use are suitable
  - 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, if needed, is one solution.

continued on next page

Prevention - 2

- **Avoid common programming problems**
  - Verify use of the correct time system for your need
    - » e.g., TDB, 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?
    - » e.g. which version of the IAU_Mars body-fixed frame?
  - Check definitions of geometric quantities
    - » Geodetic versus latitudinal coordinates
    - » Oblate versus spherical body shapes
  - 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 <u>reference frame</u> ID with the ID of the instrument itself (the object ID)

Useful Documentation

- NAIF has compiled a list of common problems, probable causes, and solutions encountered by users of the various SPICE Toolkits:
  - Refer to …/doc/html/req/problems.html or …doc/req/PROBLEMS.REQ, both of which are provided in each Toolkit package.
- Some tutorials (e.g. SPK and CK) contain a section near the end describing common problems.
- 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. At a minimum 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 (IDL, Matlab).
  - The Toolkit version you are using, i.e. N0064 (also called N64).
  - Names of the kernel files being used.
    - » You may need to provide the kernels themselves if these are not available to NAIF.
  - Your inputs to SPICE modules that signaled the error.
  - If possible, a code fragment from where the error seems to occur.
- Send the email to anyone on the NAIF team.

Navigation and Ancillary Information Facility

Obtaining SPICE Components Offered by NAIF and Horizons

Emphasis on Kernels

January 2012

Overview

- **Many SPICE products are available from the NAIF server**
  - These are mostly products produced at JPL by NAIF
  - Access is available using the http or ftp protocol
    - » Note: starting with “http” leads to “ftp”
  - 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
  - As a general rule, NAIF has no cognizance of these products
- **Horizons is an on-line natural body ephemeris generator**

NAIF Server HTTP URLs

Navigation and Ancillary Information Facility

- NAIF home page
  http://naif.jpl.nasa.gov
  - Here you may access all official SPICE products produced by NAIF
    - kernels (generic, mission ops, PDS archived)
    - software (Toolkits and individual application programs)
    - documents
    - tutorials
    - programming lessons
    - problem solving tips
    - rules about using SPICE
    - links to useful resources

- SPICE announcements (by NAIF)
  http://naif.jpl.nasa.gov/mailman/listinfo/spice_announce

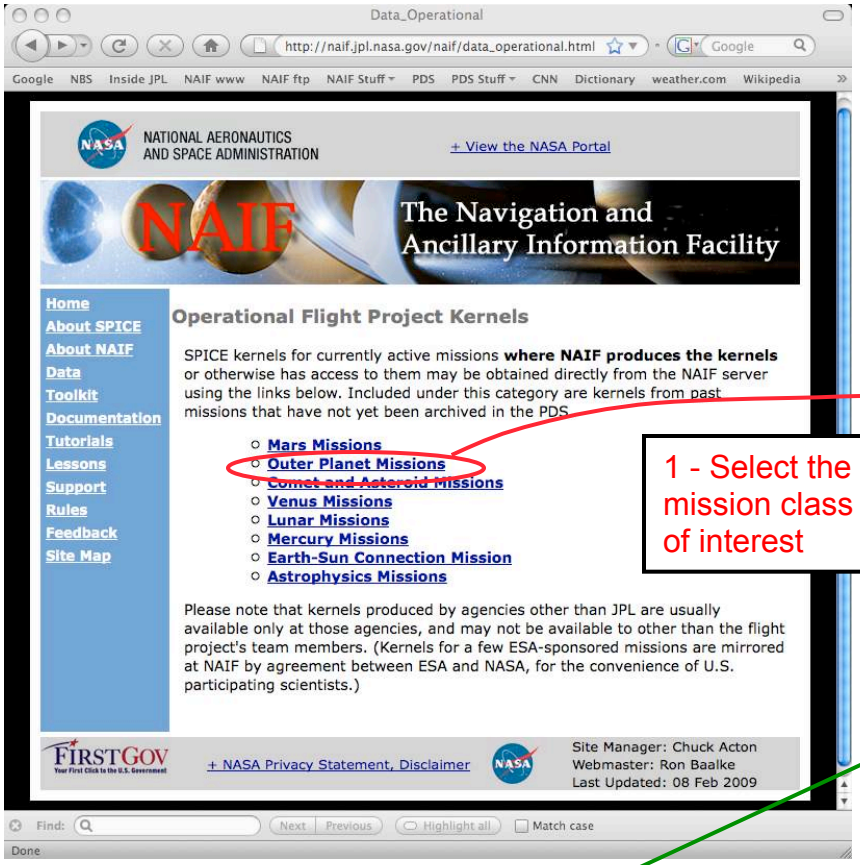
- SPICE discussion (by anyone)
  http://naif.jpl.nasa.gov/mailman/listinfo/spice_discussion

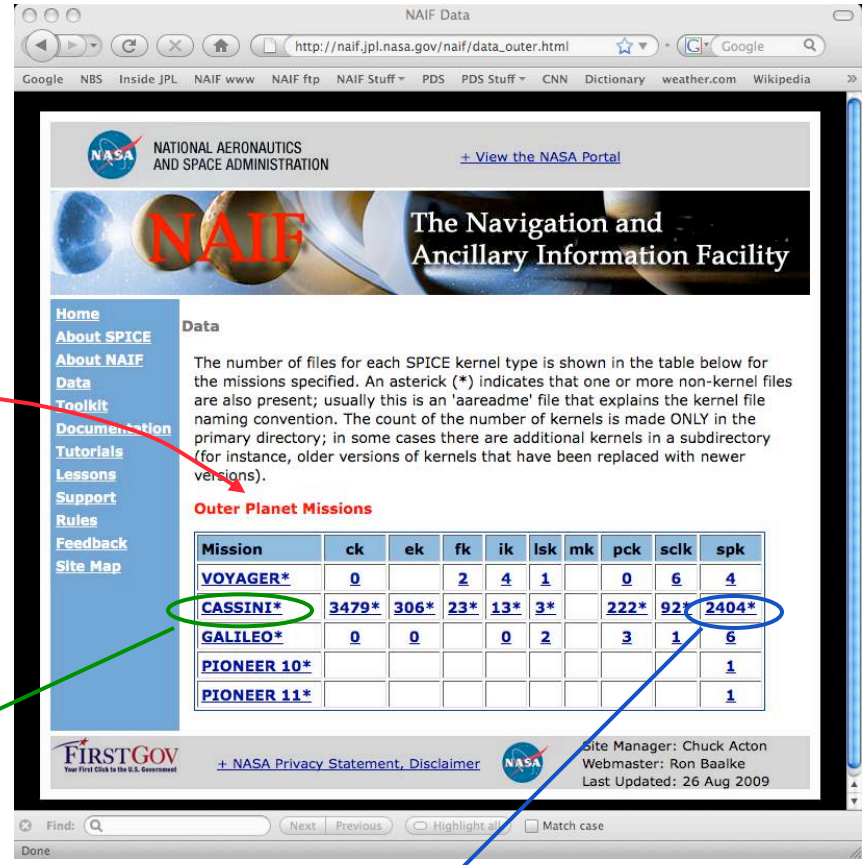
Getting SPICE Kernels

- **The remaining charts discuss where to find the various categories of SPICE kernel files**
  - **Operational flight project kernels, for (mostly JPL) active flight projects**
  - **PDS archived kernels, those formally delivered to and accepted by NASA's Planetary Data System**
  - **Generic kernels, used by many flight projects and other endeavors**

Obtaining Operational Flight Project Kernels - 1

Navigation and Ancillary Information Facility





1 - Select the mission class of interest

2a - Select the project name to get access to all products available for that project (see next page)

- or -

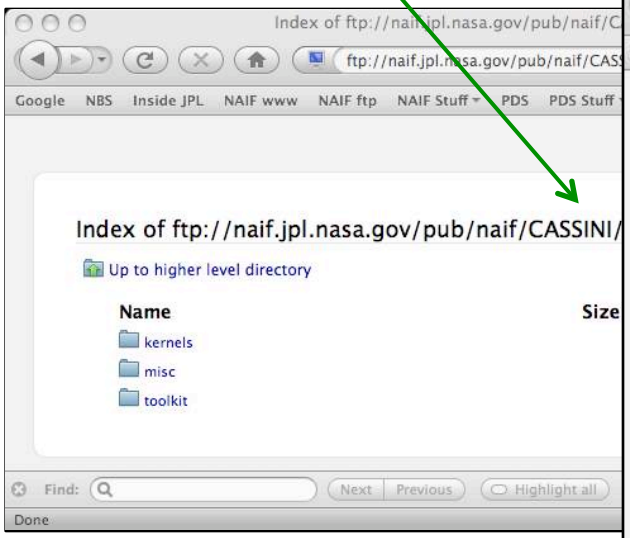
2b - Select the kernel type to get access to all kernels of that type for that project. The number tells how many kernels of that type are available. (see next page)

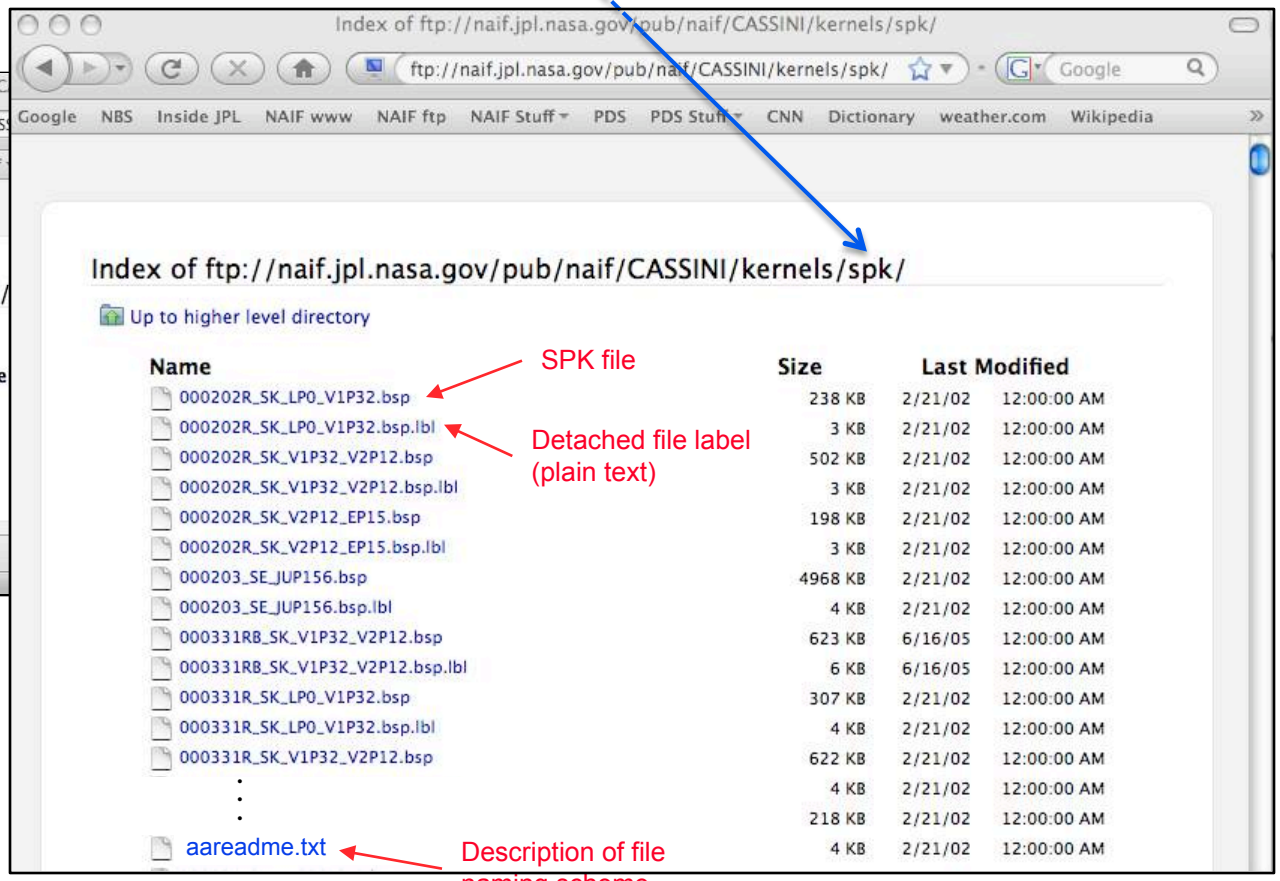
Obtaining Operational Flight Project Kernels - 2

Navigation and Ancillary Information Facility

Access to kernels and other products available for the named project

Access to kernels of the selected type for the named project





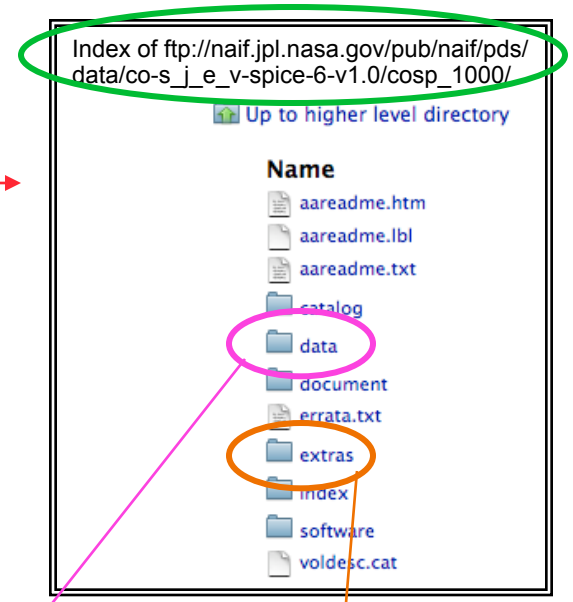
Overview: Obtaining PDS Archived Kernels

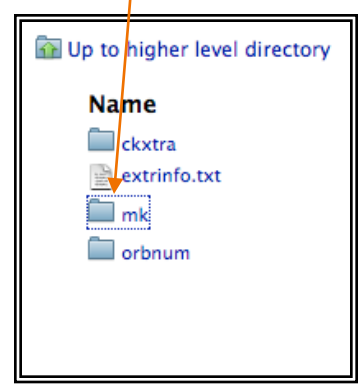
- **Two methods are available for obtaining PDS archived kernels.**
  - **Directly from the NAIF server, using your browser: recommended!**
    - » **Unless you have reason to do otherwise, download the entire archival data set using the ftp URL**
      - **That way you'll get all the latest data, the 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.**
  - **Using a web browser to access the PDS central catalog interface, typing "SPICE" and the mission name or acronym in the text search box**
    - » **NAIF suggests you use this method <u>only</u> if you wish to obtain one or a few kernels that fit specific search criteria**
- **Pictorial examples are shown on the next several pages**

Obtaining Archived Kernels from the NAIF Server - 1

Navigation and Ancillary Information Facility

Mission Name	Archive Overview	Volume FTP Link	Data Size (gbytes)	Start Time	Stop Time	Data Set Subsetter Link
Cassini Orbiter	aareadme.htm	cosp_1000	18.6	1996-11-06	2008-12-31	subset
Clementine	aareadme.htm	clsp_1000	0.8	1994-01-26	1994-05-07	subset
Deep Impact	aareadme.htm	disp_1000	0.5	2005-01-12	2009-01-15	subset
Deep Space 1	aareadme.htm	ds1sp_1000	0.9	1998-10-24	2003-12-31	subset
MER 1 (Opportunity)	aareadme.htm	mer1sp_1000	1.8	2003-07-07	2009-05-19	subset
MER 2 (Spirit)	aareadme.htm	mer2sp_1000	1.5	2003-06-10	2009-04-28	subset
MESSENGER	aareadme.htm	messsp_1000	12.7	2004-08-03	2008-10-20	subset
Mars Express	AAREADME.TXT	mexsp_1000	0.9	2003-06-02	2008-07-31	subset
Mars Global Surveyor	aareadme.htm	mgsp_1000	15.4	1996-11-06	2006-11-02	subset
Mars Odyssey	aareadme.htm	odsp_1000	9.0	2001-04-07	2009-03-31	subset





If you select “PDS Archive Area” on the NAIF web page you can follow a path like this one.

- You can use the ftp URL along with Unix “wget” or the FileZilla tool, or some other equivalent, to download the entire data set–**recommended, if not too large! Otherwise see the next page.**
- Or you can select specific kernels from the kernel folders, and/or “furnsh” meta- kernels and other items from the extras folder

Obtaining Archived Kernels from the NAIF Server - 2

Navigation and Ancillary Information Facility

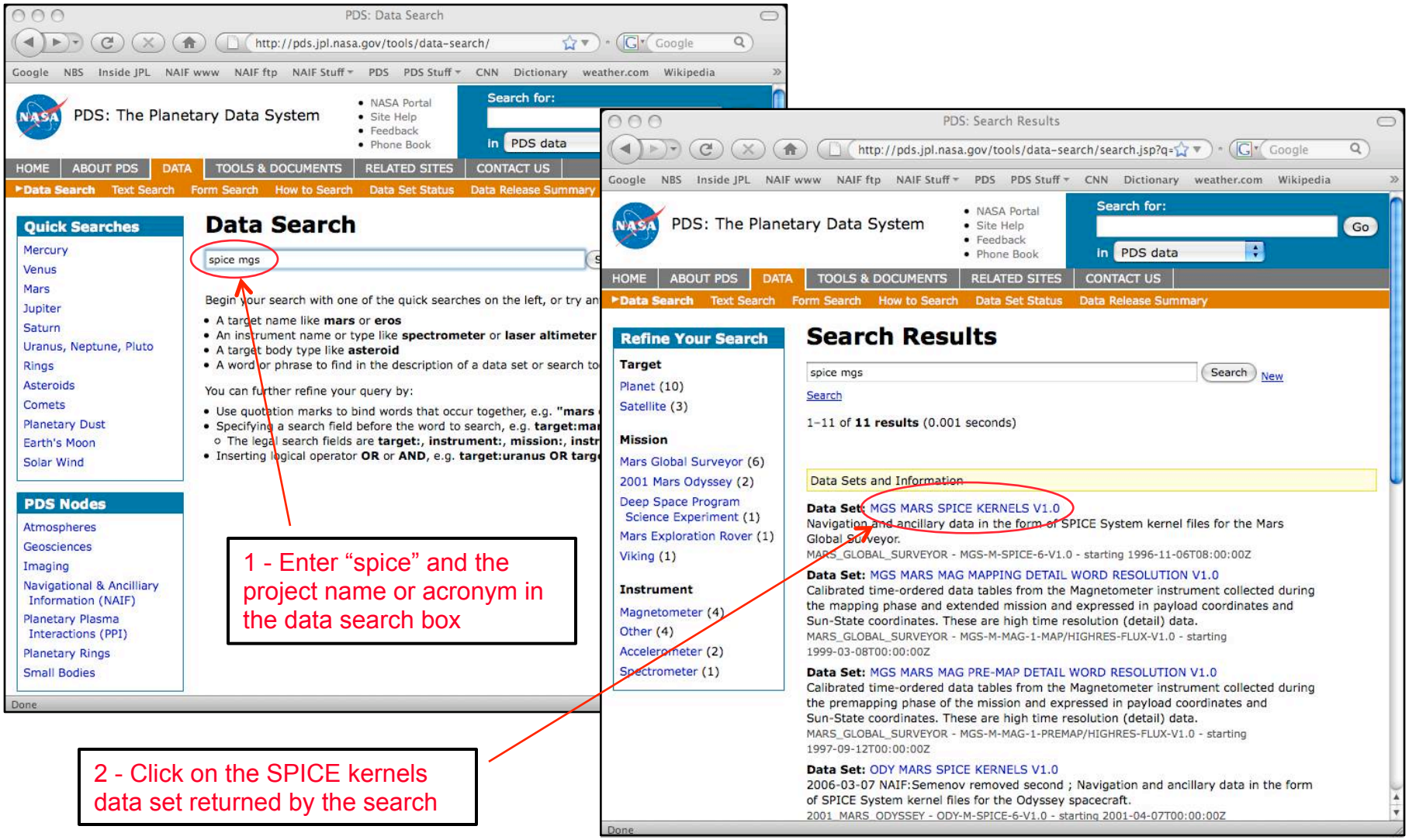
Mission Name	Archive Overview	Volume FTP Link	Data Size (gbytes)	Start Time	Stop Time	Data Set Subsetter Link
Cassini Orbiter	aareadme.htm	cosp_1000	18.6	1996-11-06	2008-12-31	subset
Clementine	aareadme.htm	clsp_1000	0.8	1994-01-26	1994-05-07	subset
Deep Impact	aareadme.htm	disp_1000	0.5	2005-01-12	2009-01-15	subset
Deep Space 1	aareadme.htm	ds1sp_1000	0.9	1998-10-24	2003-12-31	subset
MER 1 (Opportunity)	aareadme.htm	mer1sp_1000	1.8	2003-07-07	2009-05-19	subset
MER 2 (Spirit)	aareadme.htm	mer2sp_1000	1.5	2003-06-10	2009-04-28	subset
MESSENGER	aareadme.htm	messsp_1000	12.7	2004-08-03	2008-10-20	subset
Mars Express	AAREADME.TXT	mexsp_1000	0.9	2003-06-02	2008-07-31	subset
Mars Global Surveyor	aareadme.htm	mgsp_1000	15.4	1996-11-06	2006-11-02	subset
Mars Odyssey	aareadme.htm	odsp_1000	9.0	2001-04-07	2009-03-31	subset

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 "Subsetter 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.

Obtaining Archived Kernels from the PDS Central Catalog - 1

Navigation and Ancillary Information Facility



1 - Enter "spice" and the project name or acronym in the data search box

2 - Click on the SPICE kernels data set returned by the search

continued on next page

Obtaining Archived Kernels from the PDS Central Catalog - 2

Navigation and Ancillary Information Facility

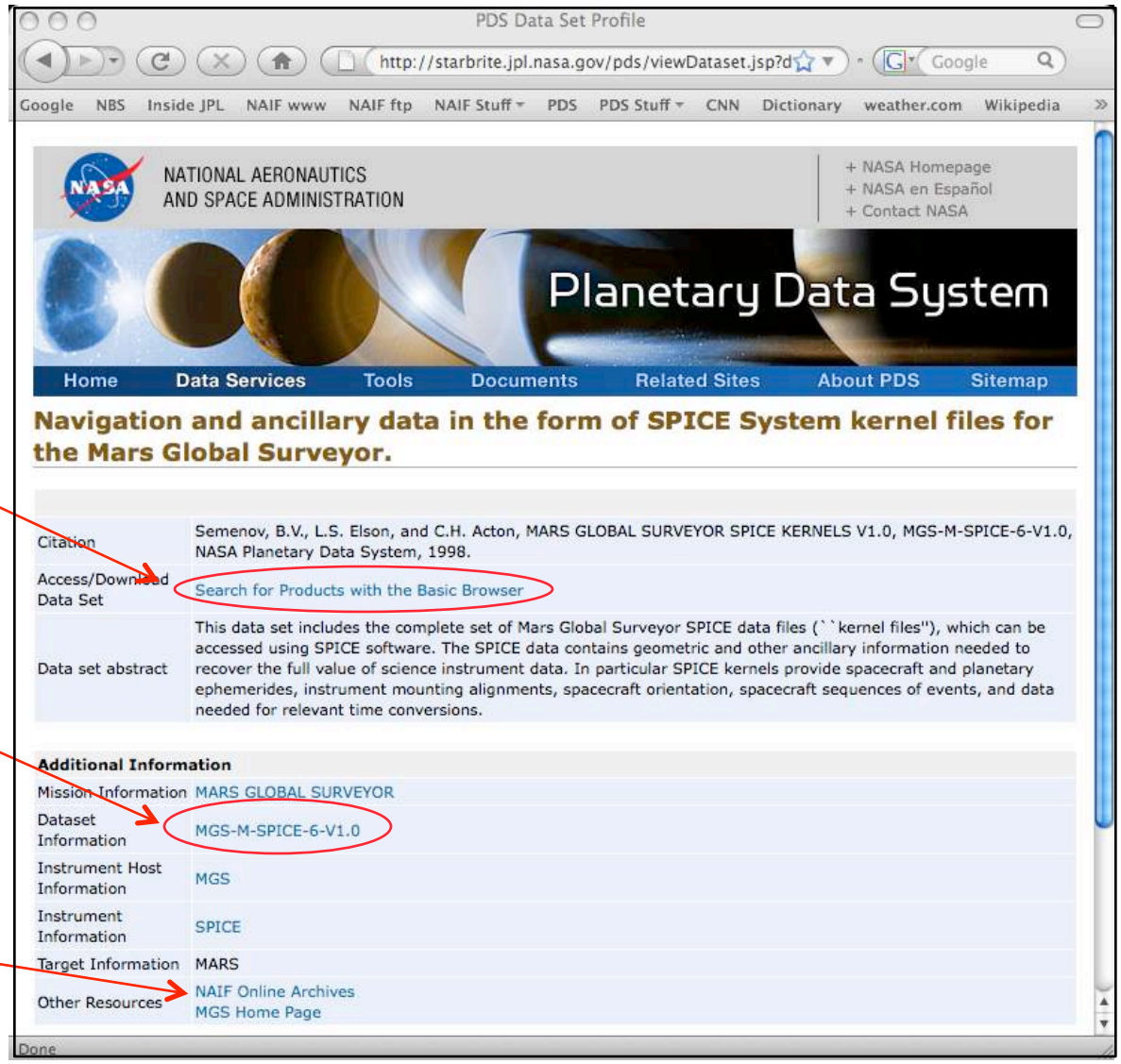
continued from previous page

Use the PDS browser if you wish to query for kernels meeting specific criteria.

Recommended

Click on the data set ID to see a summary of the entire data set

Click on “NAIF Online Archives” to get to the data set. From there you can download the complete data set (recommended!) or individual components.



continues on next page

Obtaining Archived Kernels from the PDS Central Catalog - 3

- Unless you have a specific reason to do otherwise you should download the complete archived SPICE data set for the mission of interest
- Complete SPICE data sets exist on the NAIF server fully expanded–not bundled in a Zip or tar file
- Use GNU wget or FileZilla or a similar utility to download the complete SPICE data set
  - Possible wget usage, and an example using Deep Impact
    - » wget -m -nH --cut-dirs=5 -nv (insert the URL of the "Volume FTP Link" for the SPICE data set here). For example:
      - wget -m -nH --cut-dirs=5 -nv ftp://naif.jpl.nasa.gov/pub/naif/pds/data/di-c-spice-6-v1.0/disp_1000/
  - FileZilla info
    - » http://filezilla-project.org/client_features.php

Horizons

- Horizons is an on-line ephemeris generator for natural bodies (and more)
- Operated by JPL's Solar System Dynamics Group
- Of primary interest to SPICE users is its ability to generate up-to-date SPK files for comets and asteroids
  - Horizons home: http://ssd.jpl.nasa.gov/?horizons
    - » Provides instructions for using the telnet interface to the SPK generation function
    - » The user interface takes some work to understand

Summary of Key Points

January 2012

Which Pieces of SPICE Must I Use?

Navigation and Ancillary Information Facility

- There's not a simple answer
  - Depends on what task you wish to accomplish
  - Depends on what mission you are working on
- Don't feel overwhelmed
  - Many complex calculations 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**
- **PCK: Size/shape/orientation of target bodies**
  - **For binary PCKs, only orientation is provided; use a text PCK to obtain size/shape**
- **IK: Instrument field-of-view geometry**
- **CK: Orientation of spacecraft or spacecraft structures that move**
- **FK: Definition/specification of non-core reference frames, including instrument mounting alignments**
- **LSK: UTC (SCET) ⟺ ET (TDB) time conversions**
- **SCLK and LSK: SCLK ⟺ ET (TDB) time conversions**

Primary Kernel Interfaces - 1

Navigation and Ancillary Information Facility

Which SPICE modules are most commonly called to use data obtained from 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
EK/ESQ	EKFIND, EKG*

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 module, which kind(s) of kernel(s) will or may be needed?

Module Name	Kernel Type(s) Needed						
	SPK	PCK	IK	CK	FK	LSK	SCLK
SPKEZR, SPKPOS	Y	M		M	M	M	M
SXFORM, PXFORM	M	M		M	M	M	M
CKGP, CKGPAV		M		Y	M	M	M
GETFOV			Y				
G*POOL		M	M				
STR2ET, TIMOUT						Y	
SCS2E, SCE2S						Y	Y
CHRONOS (time conversion app.)	M	M		M	M	Y	M

Yes = the indicated kernel type <u>is</u> needed
Maybe = the indicated kernel type <u>may be</u> 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 of interest to you
    - » Watch out for data gaps within that time span
    - » Watch out for the difference of ~66 seconds between ET and UTC
  - contain all the kernel types needed by SPICE to answer your question
    - » As the previous charts allude, 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?

- It depends on the mission or task you are working on…
- If you're working with JPL mission data, there are three categories of kernel data available to you.
  - Mission operations kernels – those used by the flight teams to fly the mission and prepare the archival science products
  - Archived kernels – those that have been selected from (or made from) the mission ops kernels, and then are well organized and documented for the permanent PDS archive
  - 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 SPICE archived kernels data sets mentioned above
- The situation may be very similar for non-JPL missions, but this is really up to whatever agency/institution is producing the kernels.

How Can I Find Possibly Useful Toolkit Modules?

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 Modules?

- The primary user-oriented documentation about each module is found in the "header" located at the top of each source code file and in the module's HTML page in the API reference guide.
  - (More documentation is found at the additional entry points for those FORTRAN modules 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.
- See "SPICE Documentation Taxonomy" in the tutorials collection for additional reading suggestions.

Does NAIF Provide Any Examples?

Navigation and Ancillary Information Facility

- Nearly all module 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
- 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/

Navigation and Ancillary Information Facility

Writing an Icy (IDL) Based Program

January 2012

Viewing This Tutorial

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 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.
- Formulate an algorithm to compute the quantities of interest using SPICE.
- 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)

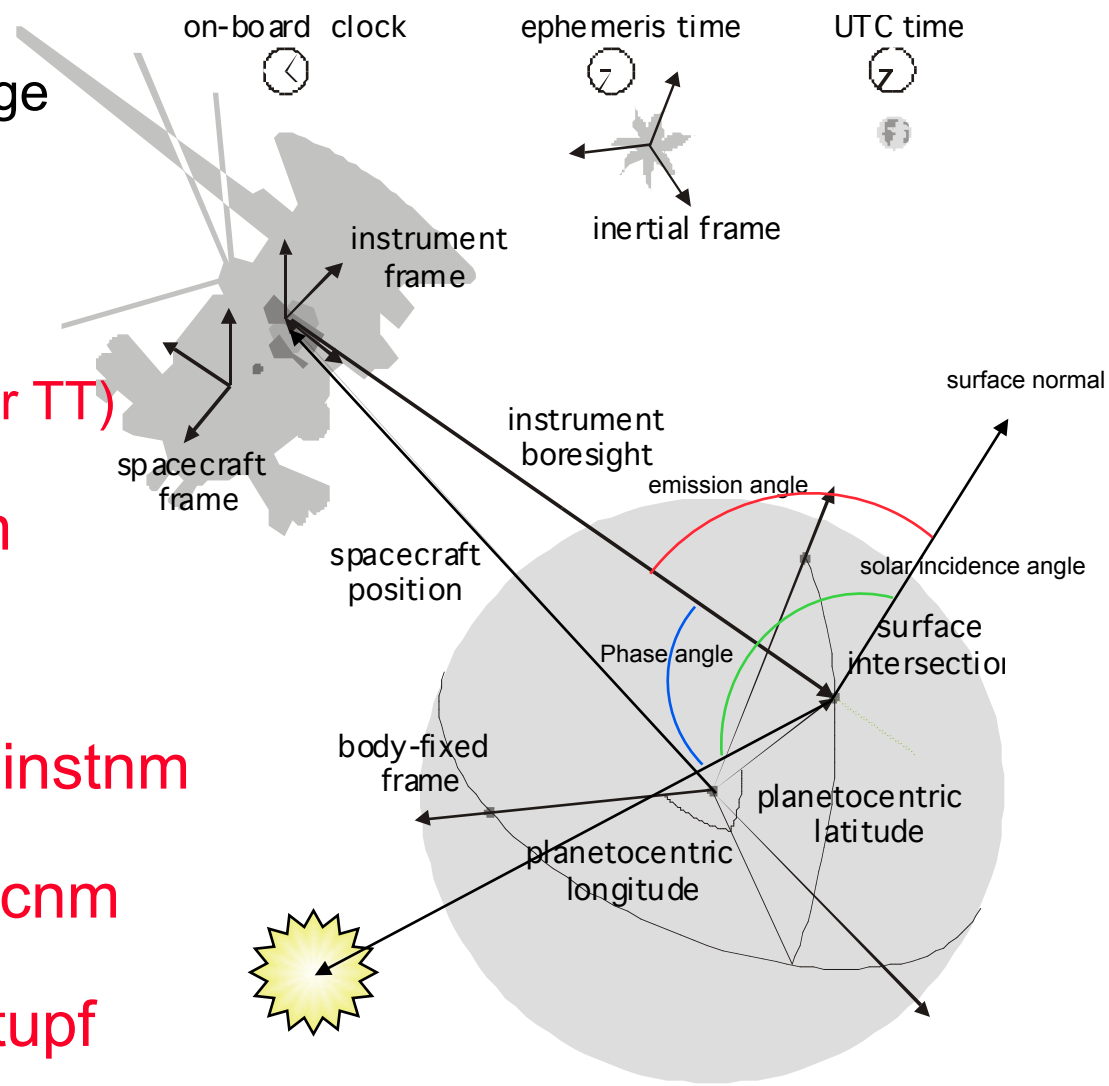
On what object? satnm

In what frame? fixref

For which instrument? instnm

For what spacecraft? scnm

Using what model? setupf



Needed Data

Navigation and Ancillary Information Facility

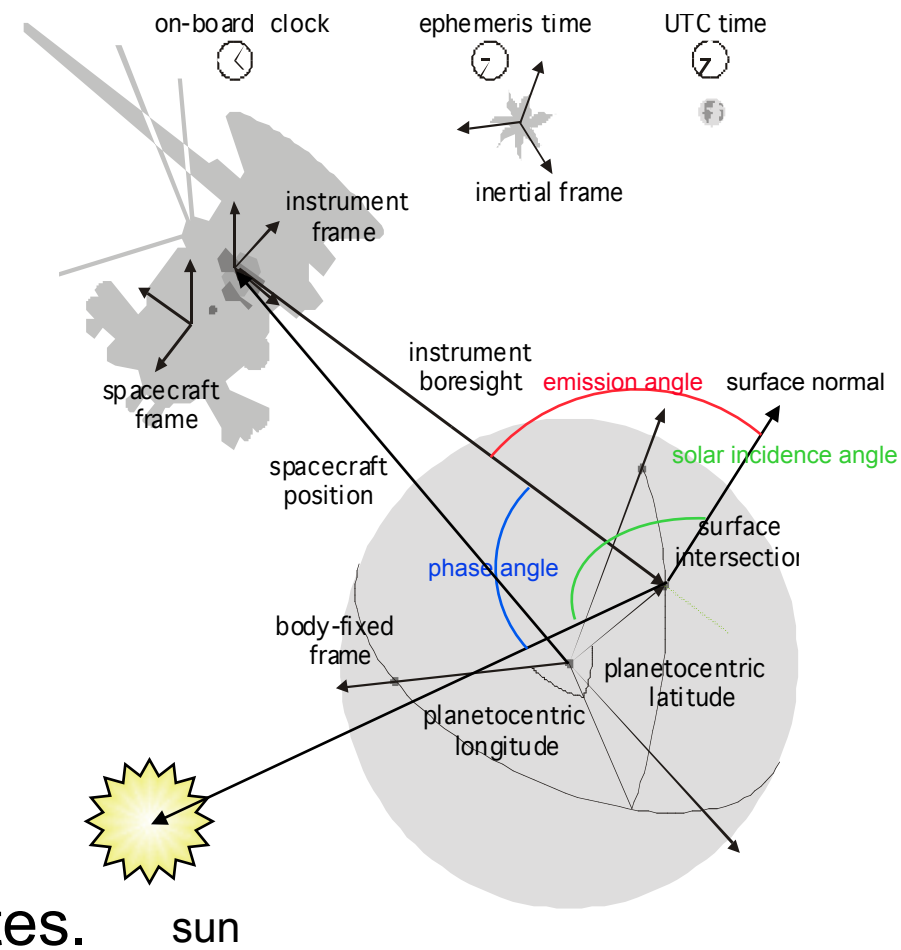
Time transformation kernels

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
	CASSINI SCLK	cas00084.tsc
satellite orientation	CASSINI PCK	cpck05Mar2004.tpc
satellite shape	CASSINI PCK	cpck05Mar2004.tpc
satellite position	planet/sat ephemeris SPK	020514_SE_SAT105.bsp
planet barycenter position	planet SPK	981005_PLTEPH-DE405S.bsp
spacecraft position	spacecraft SPK	030201AP_SK_SM546_T45.bsp
spacecraft orientation	spacecraft CK	04135_04171pc_psiv2.bc
instrument alignment	CASSINI FK	cas_v37.tf
instrument boresight	Instrument IK	cas_iss_v09.ti

Load kernels

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

- 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

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_illum`. 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

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`):

```
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

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, bundry
```

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
cspice_getfov, instid, ROOM, shape, iframe, insite, bundry
```

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

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

   ABCORR = 'CN+S'
   ROOM   = 10L
   setupf = ''
   satnm  = ''
   fixref = ''
   scnm   = ''
   instnm = ''
   time   = ''
   R2D    = cspice_dpr()
```

Complete source code -1


```
;; 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


```
;; 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 -4


```
      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

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 and link the program - 2

Navigation and Ancillary Information Facility

Terminal Window

```
IDL>  .compile prog_geometry.pro
% Compiled module: PROG_GEOMETRY.
```

Running the program

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

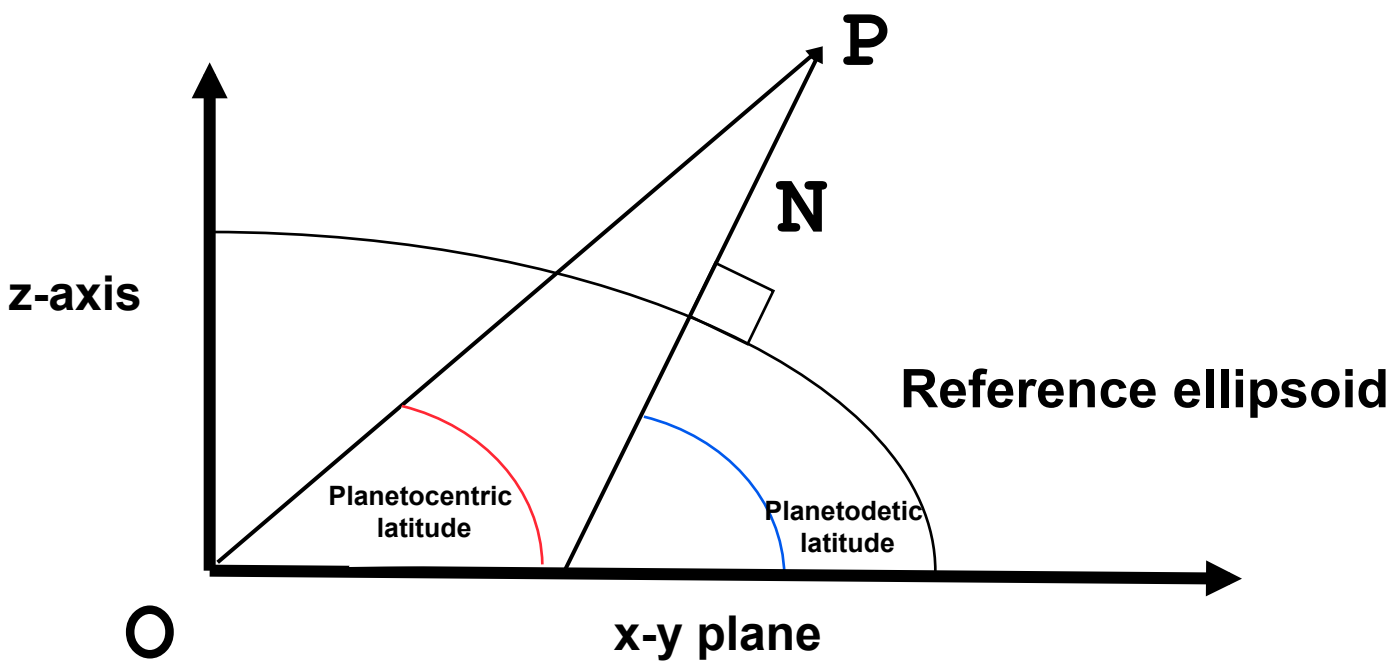
Navigation and Ancillary Information Facility

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
```

Backup

- **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 an Mice (MATLAB) Based Program

January 2012

Viewing This Tutorial

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.
- Formulate an algorithm to compute the quantities of interest using SPICE.
- 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)

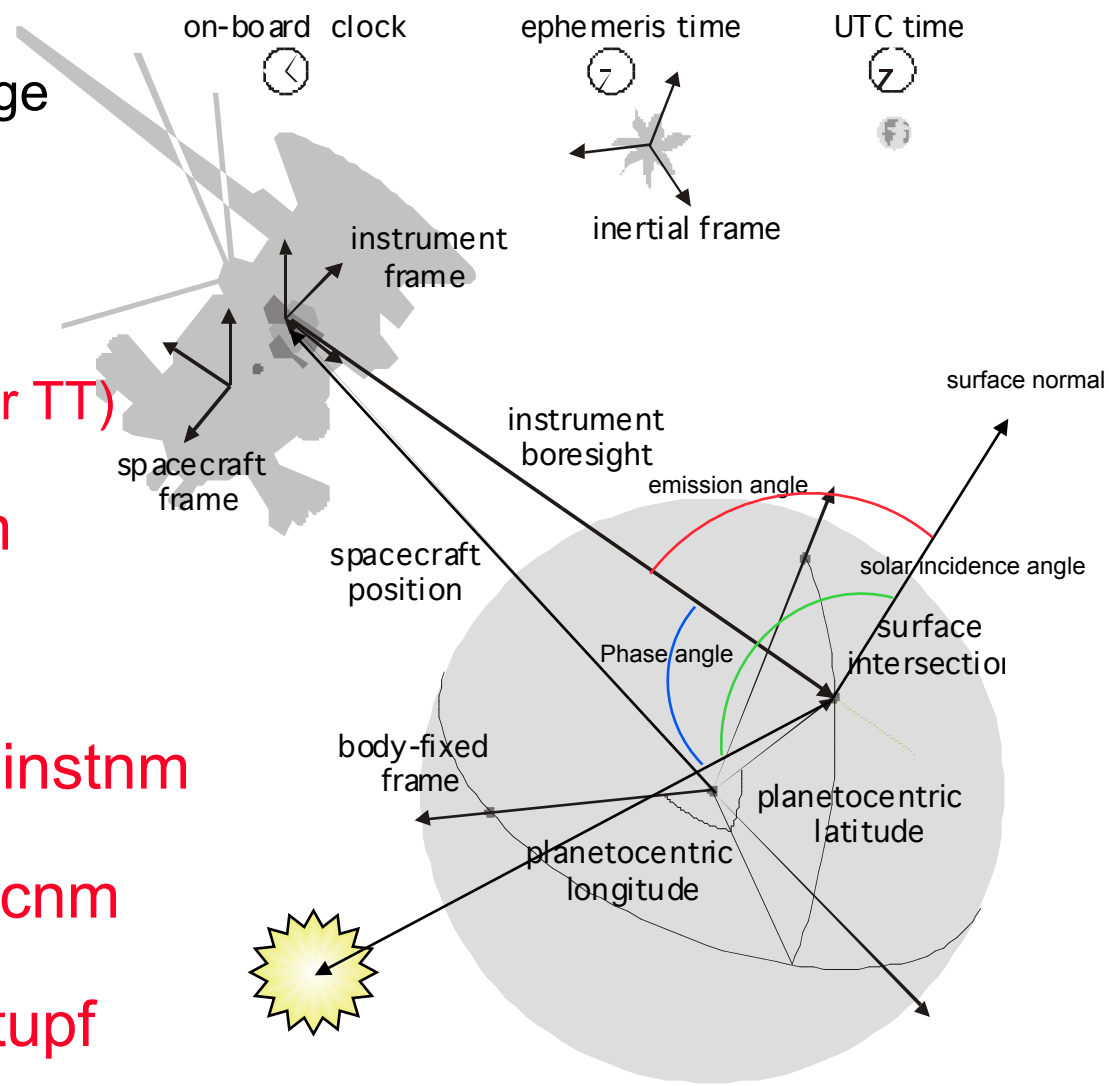
On what object? satnm

In what frame? fixref

For which instrument? instnm

For what spacecraft? scnm

Using what model? setupf



Needed Data

Navigation and Ancillary Information Facility

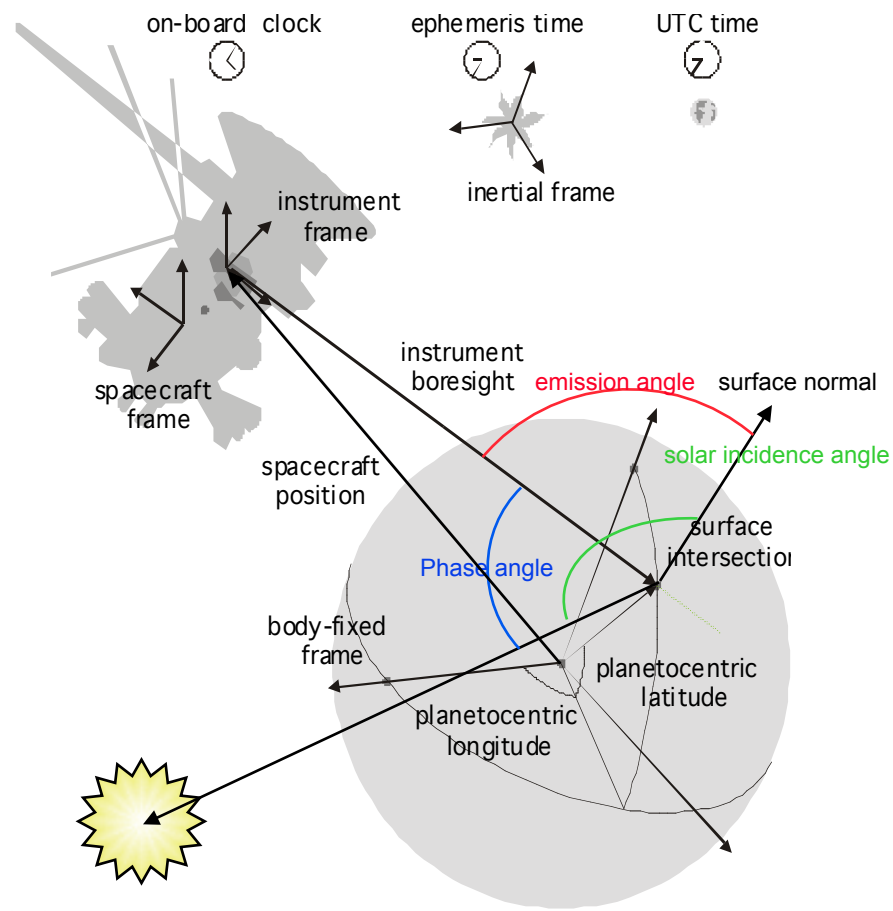
Time transformation kernels

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
	CASSINI SCLK	cas00084.tsc
satellite orientation	CASSINI PCK	cpck05Mar2004.tpc
satellite shape	CASSINI PCK	cpck05Mar2004.tpc
satellite position	planet/sat ephemeris SPK	020514_SE_SAT105.bsp
planet barycenter position	planet SPK	981005_PLTEPH-DE405S.bsp
spacecraft position	spacecraft SPK	030201AP_SK_SM546_T45.bsp
spacecraft orientation	spacecraft CK	04135_04171pc_psiv2.bc
instrument alignment	CASSINI FK	cas_v37.tf
instrument boresight	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

- 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

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, pdlat, 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
   ...
```

Get inputs - 1

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`):

```
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

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, bundry] = 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', ...
                                                    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):  %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
```

Complete the program

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

```
ABCORR = 'CN+S';
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


```
% 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 );
```

Complete source code - 2


```
% 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


```
   % 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):  %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

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

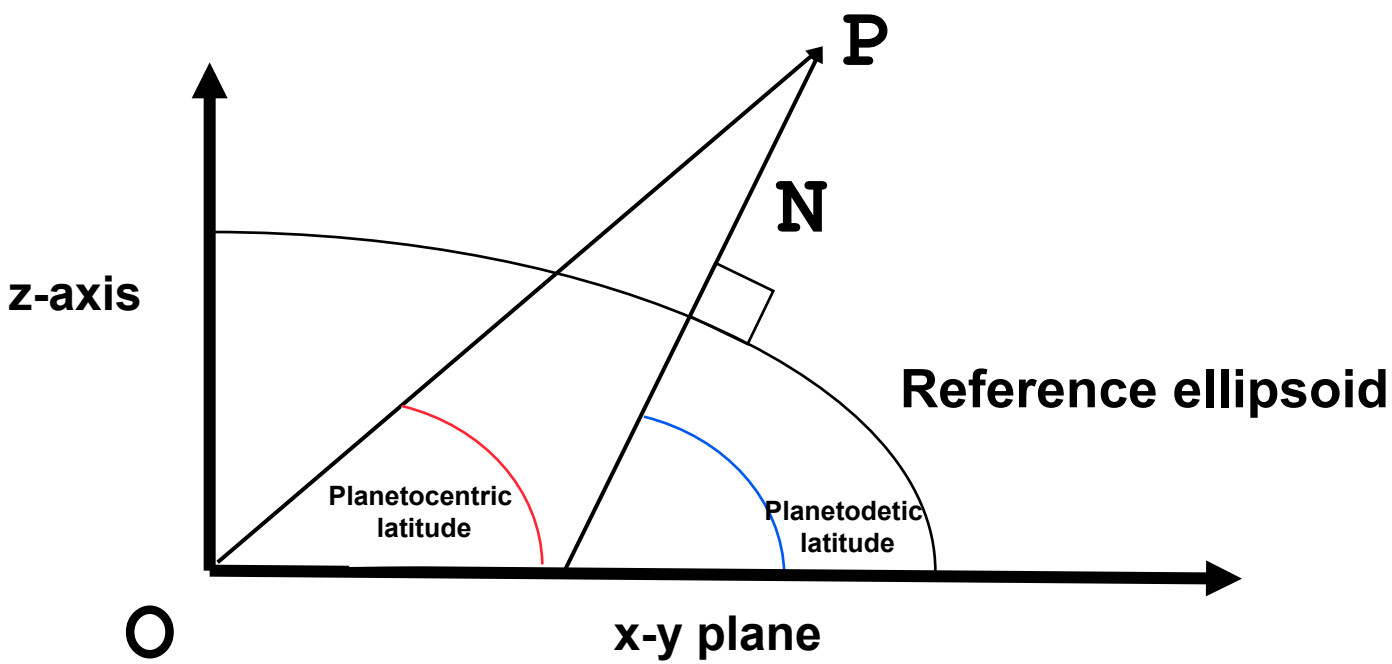
Navigation and Ancillary Information Facility

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
```

Backup

- **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 2012

Viewing This Tutorial

Undefined variables are displayed in red

Results are displayed in blue

Introduction

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.
- Formulate an algorithm to compute the quantities of interest using SPICE.
- 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 and from spacecraft to target center.
- 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)

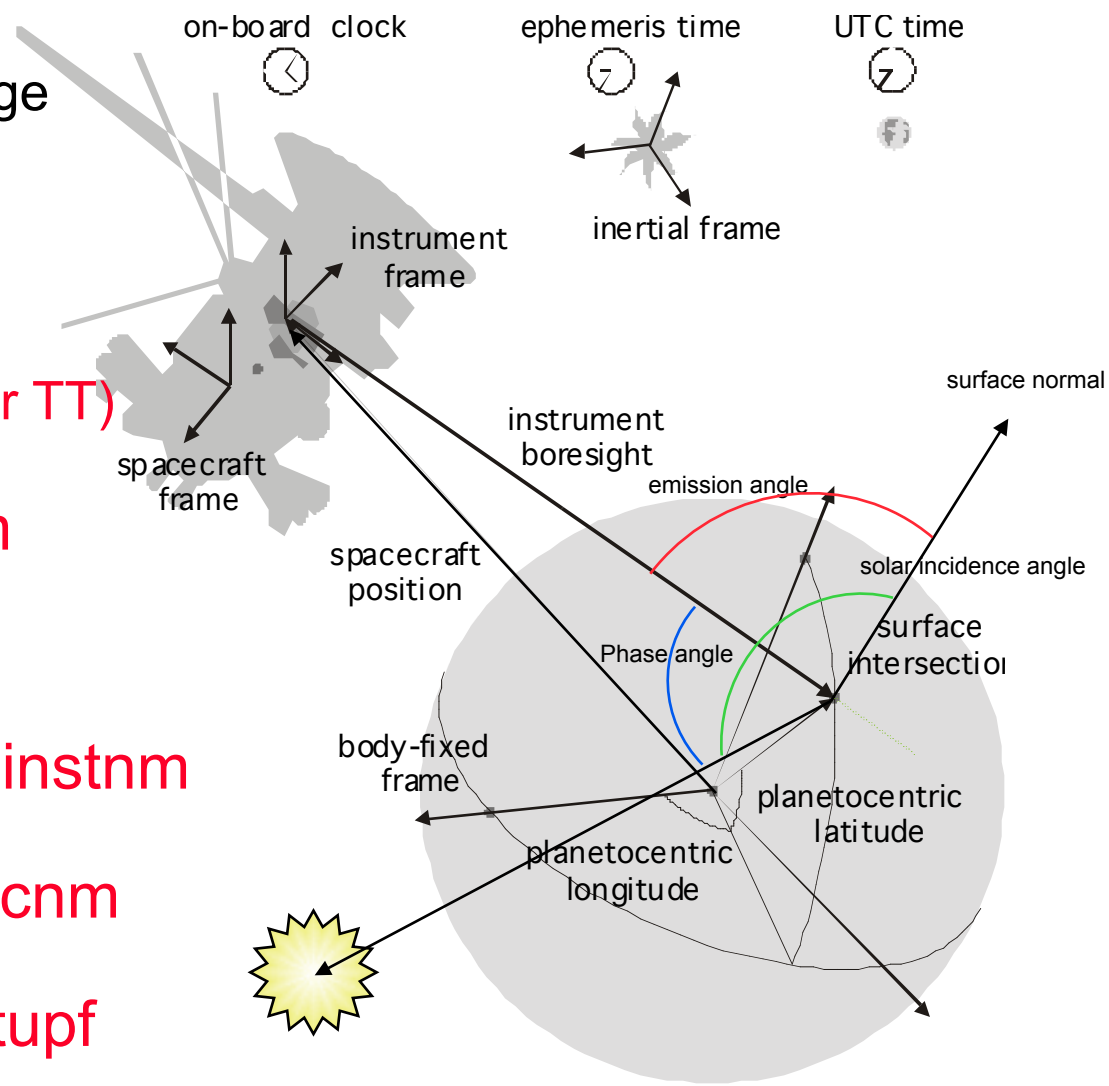
On what object? satnm

In what frame? fixref

For which instrument? instnm

For what spacecraft? scnm

Using what model? setupf



Needed Data

Navigation and Ancillary Information Facility

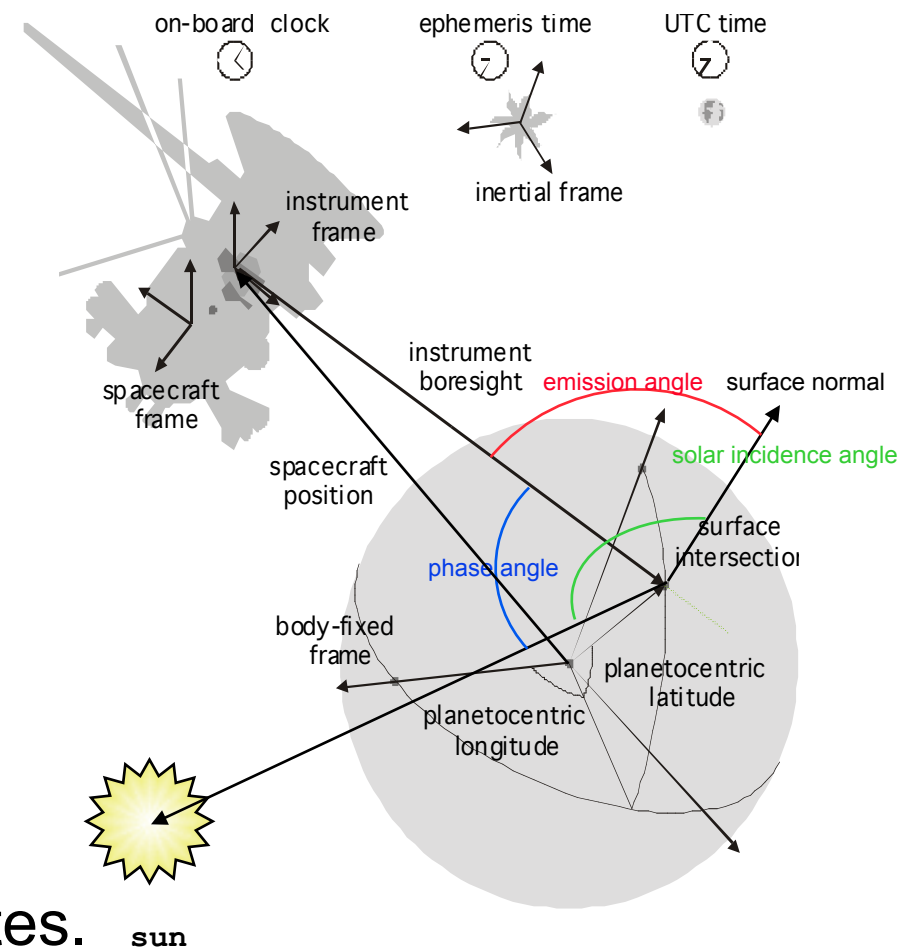
Time transformation kernels

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
	CASSINI SCLK	cas00084.tsc
satellite orientation	CASSINI PCK	cpck05Mar2004.tpc
satellite shape	CASSINI PCK	cpck05Mar2004.tpc
satellite position	planet/sat ephemeris SPK	020514_SE_SAT105.bsp
planet barycenter position	planet SPK	981005_PLTEPH-DE405S.bsp
spacecraft position	spacecraft SPK	030201AP_SK_SM546_T45.bsp
spacecraft orientation	spacecraft CK	04135_04171pc_psiv2.bc
instrument alignment	CASSINI FK	cas_v37.tf
instrument boresight	Instrument IK	cas_iss_v09.ti

Load Kernels

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

- 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 boolean 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
   satellite.                                                      */

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

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

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,     bundry );
```

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,     bundry );
```

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\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 );
   }
```

Complete the Program

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    bundry[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, 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,     bundry );
```

Complete Source Code - 3


```
/* Compute the boresight ray intersection with the surface of the
   satellite.                                                      */

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 you 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

– 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

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

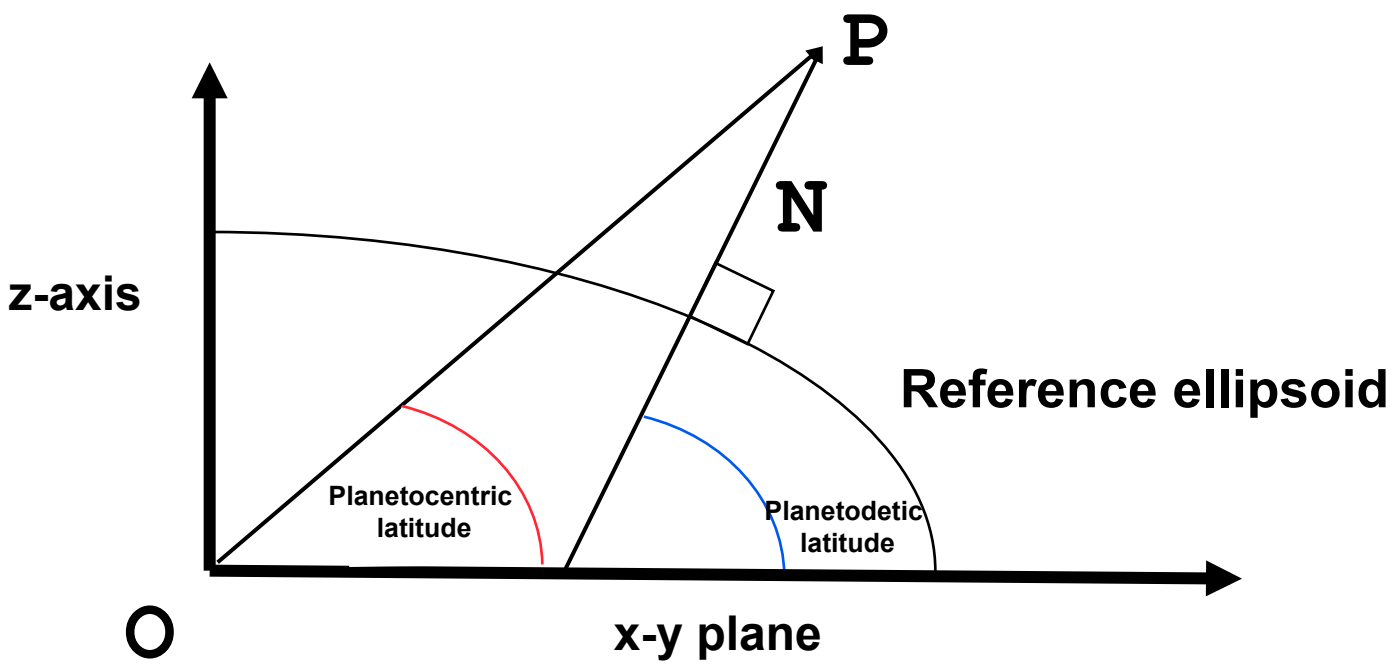
Navigation and Ancillary Information Facility

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>
```

Backup

- **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 2012

Viewing This Tutorial

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.
- Formulate an algorithm to compute the quantities of interest using SPICE.
- 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 and from spacecraft to target center.
- 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)

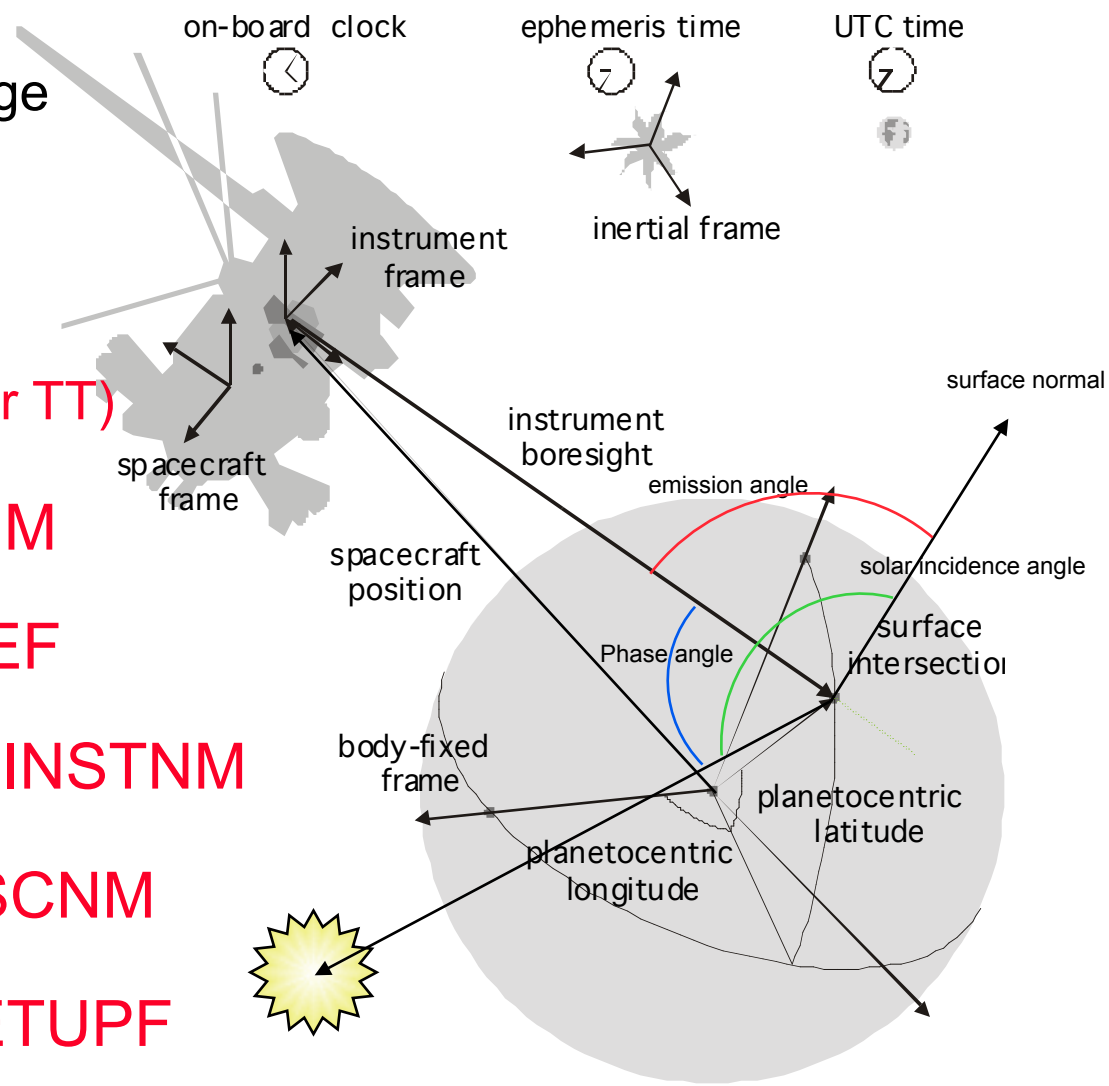
On what object? SATNM

In what frame? FIXREF

For which instrument? INSTNM

For what spacecraft? SCNM

Using what model? SETUPF



Needed Data

Navigation and Ancillary Information Facility

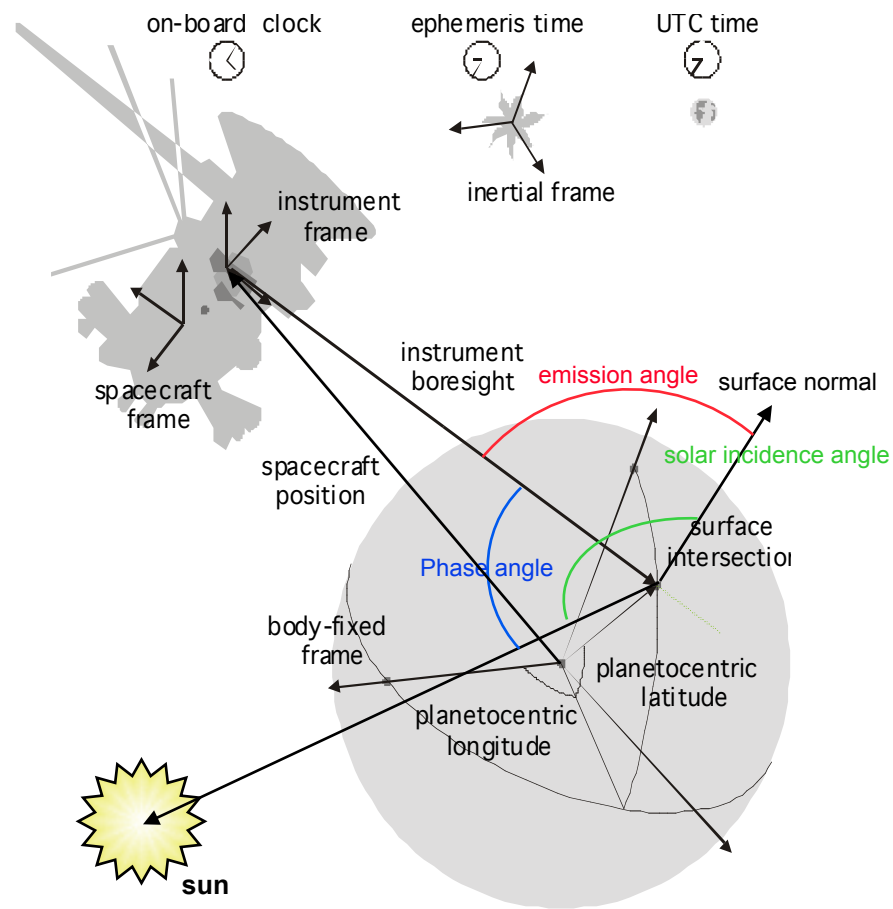
Time transformation kernels

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
	CASSINI SCLK	cas00084.tsc
satellite orientation	CASSINI PCK	cpck05Mar2004.tpc
satellite shape	CASSINI PCK	cpck05Mar2004.tpc
satellite position	planet/sat ephemeris SPK	020514_SE_SAT105.bsp
planet barycenter position	planet SPK	981005_PLTEPH-DE405S.bsp
spacecraft position	spacecraft SPK	030201AP_SK_SM546_T45.bsp
spacecraft orientation	spacecraft CK	04135_04171pc_psiv2.bc
instrument alignment	CASSINI FK	cas_v37.tf
instrument boresight	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

- 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 ILLUM. 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     satellite.

      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 )
         ...
      ELSE
         ...
```

Get Inputs - 1

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 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

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,    BUNDRY          )
```

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

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     VNORM


INTEGER              FILESZ
PARAMETER          ( FILESZ =     255 )
INTEGER              WORDSZ
PARAMETER          ( WORDSZ =      40 )
INTEGER              ROOM
PARAMETER          ( ROOM   =      10 )

CHARACTER*(WORDSZ)   IFRAME
CHARACTER*(WORDSZ)   INSTNM
CHARACTER*(WORDSZ)   SATNM
CHARACTER*(WORDSZ)   FIXREF
CHARACTER*(WORDSZ)   SCNM
CHARACTER*(FILESZ)   SETUPF
CHARACTER*(WORDSZ)   SHAPE
CHARACTER*(WORDSZ)   TIME

DOUBLE PRECISION     ALT
DOUBLE PRECISION     BUNDRY(3, ROOM)
```

```
DOUBLE PRECISION     EMISSN
DOUBLE PRECISION     ET
DOUBLE PRECISION     F
DOUBLE PRECISION     INSITE(3)
DOUBLE PRECISION     SRFVEC(3)
DOUBLE PRECISION     PCLAT
DOUBLE PRECISION     PCLON
DOUBLE PRECISION     PDLAT
DOUBLE PRECISION     PDLON
DOUBLE PRECISION     PHASE
DOUBLE PRECISION     POINT (3)
DOUBLE PRECISION     R
DOUBLE PRECISION     RADII (3)
DOUBLE PRECISION     RE
DOUBLE PRECISION     RP
DOUBLE PRECISION     SOLAR
DOUBLE PRECISION     TRGEPC


INTEGER              I
INTEGER              INSTID
INTEGER              N

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,    BUNDRY          )
```

Complete Source Code - 3

Navigation and Ancillary Information Facility

```
C     Compute the boresight ray intersection with the surface of the
C     satellite.
      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

- **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

– 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

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

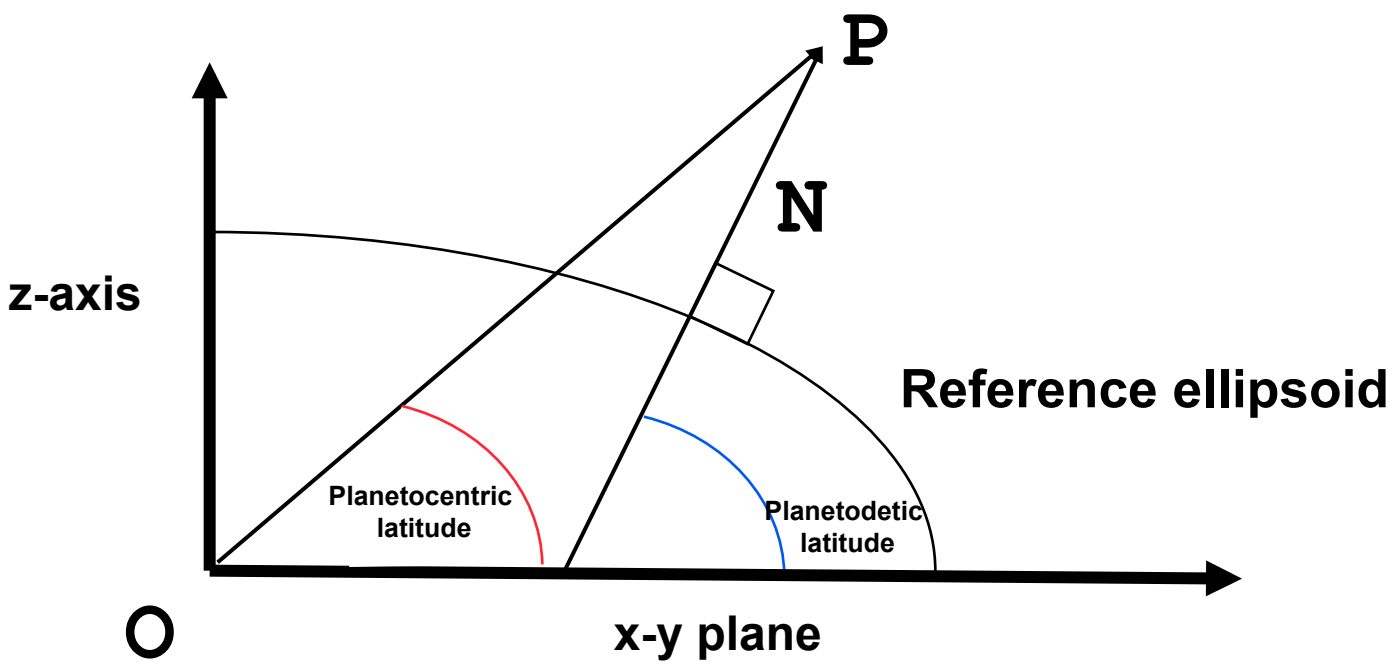
Navigation and Ancillary Information Facility

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>
```

Backup

- **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 2012

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 (subroutines)
- 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 <u>not</u> making JPL NAV-style "p-files"
- Issues affecting performance (reading efficiency) and usability

Purpose

- **This tutorial provides guidance for writing an SPK file using software provided by NAIF:**
  - **the MKSPK application program**

  or
  - **SPK writer modules from the SPICELIB (FORTRAN) or CSPICE (C-language) library, or from the Icy (IDL) system**
    - » **Only partial implementation in Icy**
    - » **No SPK writers implemented in Mice**

Scope

Navigation and Ancillary Information Facility

- **This tutorial addresses production of SPK files**
  - **for general purposes**
  - **for use with NASA's Deep Space Network (the "SPS")**

- **(Note: This tutorial does *not* address SPK production by JPL navigation teams using the NIOSPK application, which was specially built to process JPL's NAVIO-format ephemeris/trajectory files.)**
  - **Those NAV teams may simply learn how to use the NIOSPK program and any useful SPK-related utilities.**

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:

    ftp://naif.jpl.nasa.gov/pub/naif/toolkit_docs/Tutorials/
- It is assumed the reader has read the "SPK Tutorial" that characterizes much of the SPK subsystem, but with emphasis on reading SPK files.
  - The SPK "reading" tutorial is available at:

    ftp://naif.jpl.nasa.gov/pub/naif/toolkit_docs/Tutorials/ (named 19_spk)
- 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:

    http://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” (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

- **Understand the physics of your data and how that relates to SPK type. For instance:**
  - **Type 5 implies 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

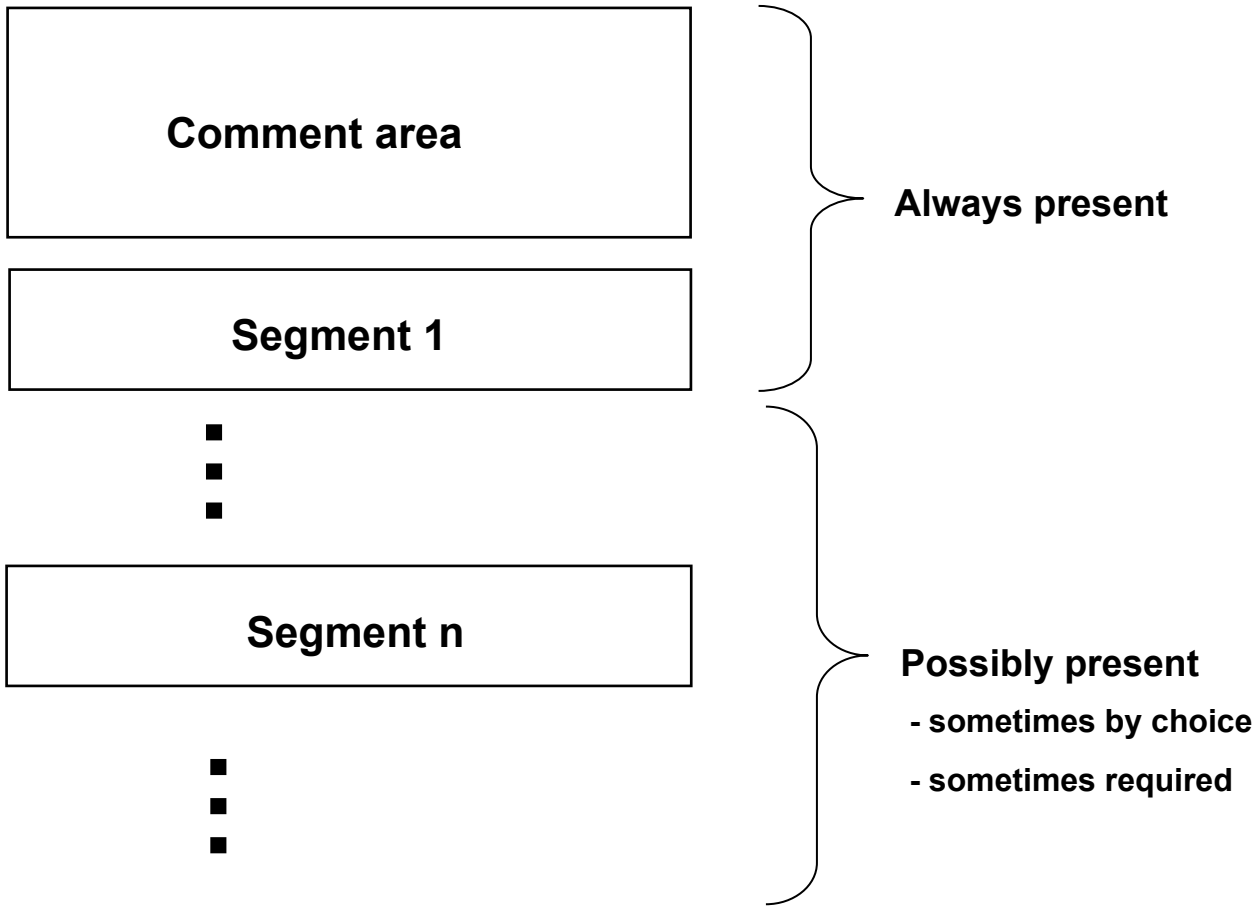
- **Ordinarily, use the NAIF MKSPK application to create SPKs from Cartesian state data or conic elements.**
  - **Depending on your source data, SPK types 5, 9, 10, and 13 will satisfy the requirements for most users.**
    - » **Type 5, yields compact SPK files when the trajectory is well approximated by piecewise two-body motion. 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.**

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										U*
N/P/C	D1	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

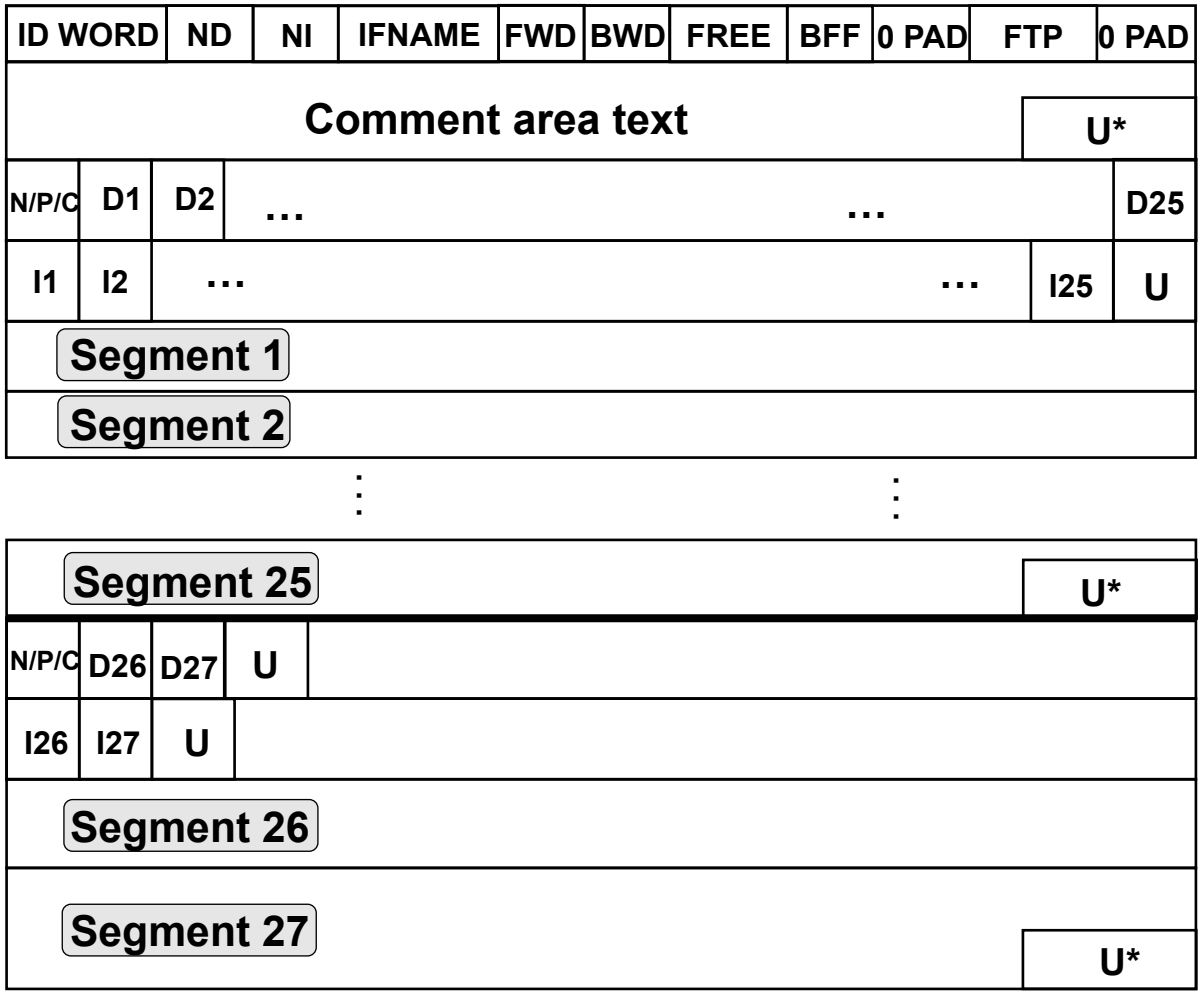
U: Unused space
U*: Possibly unused space

SPK File Structure - 2

Navigation and Ancillary Information Facility

An SPK file containing 27 segments

Records are fixed-length: 1024 bytes



File record: One record

Comment area: Always present but could be empty. One or more records.

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.)

Diagram not to scale

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 30000 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 the Hermite interpolation with variable length intervals SPK type (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

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	Discreet 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; fixed length intervals between states	Use Type 13 unless state spacing is truly uniform when measured in the TDB system.
13	Hermite interpolation; 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

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 non-JPL ephemeris producers and 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.
- For good SPK reading performance, the total number of segments for any given body in a file should be kept under the dimension of the SPKBSR segment buffer, currently set to 30,000.
  - More details about reading efficiency are provided at the end of this tutorial.
  - When reading data from multiple SPK files, a more stringent limit applies: the total number of loaded segments for any body, possibly contributed by multiple files, should be less than the SPKBSR segment buffer size.
  - For best efficiency, the total number of segments loaded should be less than this buffer size.
- One <u>may</u> 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 write 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-15. 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 3-mod-4, meaning degree 3, 7, 11 or 15.

Reference Frame

- No matter the SPK type, all ephemeris data must be provided in a well identified reference frame.
- 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.
- Some examples of typical reference frames used:
  - Inertial (non-rotating):
    - » Earth Mean Equator and Equinox of J2000 (EME2000, a.k.a. J2000)
    - » Ecliptic of J2000
  - Body fixed (non-inertial)
    - » ITRF93 (for Earth)
    - » MOON_ME (MOON_MeanEarth)

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 a 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 described above.**
- **Both methods are described in the next few pages.**

Making Your Choice - 1

Navigation and Ancillary Information Facility

- **Using the MKSPK program provided in the Toolkit could be easiest for "simple" situations.**
  - **Provides considerable flexibility for accepting a wide assortment of input data formats.**
  - **Does allow 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 files when the amount of input data requires so, is automatically handled.**

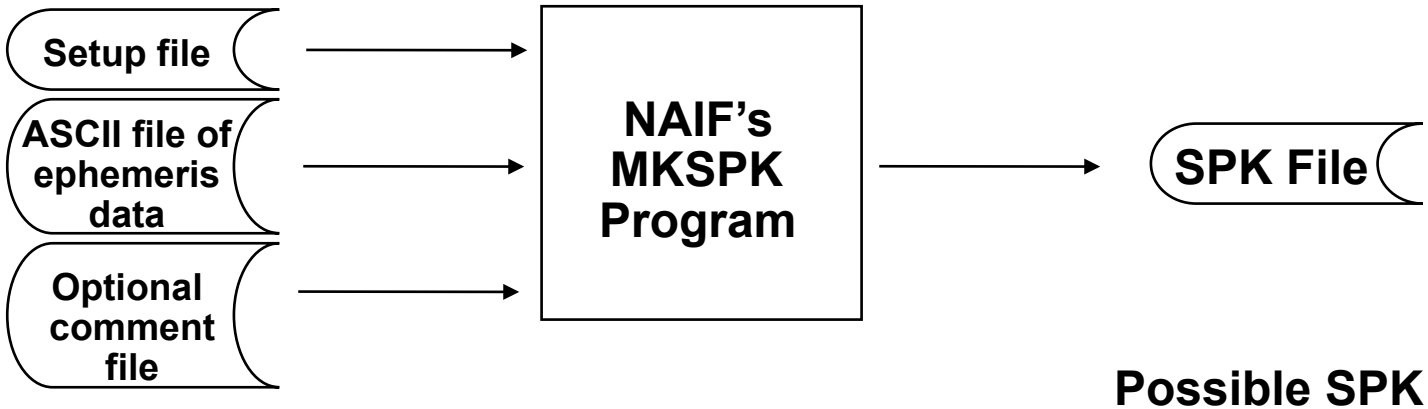
Making Your Choice - 2

- **Using the SPK "writer" modules found in SPICELIB, CSPICE, and Icy 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.**

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 will 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.**

Using the MKSPK Utility - 4

- **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.**

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.
  - SPICELIB (FORTRAN)
  - CSPICE (C)
    - » All types supported except Type 1
  - Icy (IDL)
    - » All types supported except Type 1, 15, 18
  - Mice (MATLAB)
    - » Currently no SPK writer modules are supported

- These routines could be embedded in your existing trajectory propagator program, or they could be used to build a separate conversion program analogous to MKSPK.

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, 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	**Repeat as needed**
SPK14A	Add data to segment	
[SPK14A]	Add more data	
SPK14E	End the segment	
etc.		
SPKCLS	Close the file	

[…] 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.**

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 consider 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 recommended that the producer of an SPK file add to the file, in the "comment area," appropriate descriptive information.**
  - 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.

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 this 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 only the states placed in the SPK file.
  - Verify a uniformly good fit on the whole time interval covered by the file.

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

- **If your project provides funding to NAIF for help with development:**
  - **Ask 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**
  - **Ask NAIF for samples of SPK files from other missions that could help you check your process.**

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 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 Preditcs (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 validation program that has some restrictions.**
- **SPK files intended for use in any of these software sets may face some restrictions. See the next pages.**
  - **(Note: The restrictions that apply as of October 2008 are far less than before this date, due to full retirement of the SPS predecessor–the NSS.)**

SPS Validation Gate

Navigation and Ancillary Information Facility

- **The SPS' front-end validation tool 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**
  - **the spacecraft SPK must be of Type 1 or Type 13**

What SPK Type to Use for Interfaces with the DSN?

- **The Metric Predicts Generator does not inherently place any restrictions on the SPK files used.**
  - **However, current rules of the SPS nevertheless restrict the SPK choice to only Type 1 or Type 13. (Other types not yet fully, formally tested.)**
  - **This restriction may be lifted in the near future: contact a DSN representative for the latest news.**
  - **Note: Only JPL NAV teams are able to produce Type 1 SPKs.**
- **The telecommunications prediction software does not inherently place any restrictions on the SPK files used.**
- **The radiometric modeling and calibration software requires only Type 1 or Type 13 SPKs**
  - **This restriction will be lifted approximately January 2009: contact a DSN representative for the latest news.**
- **The delta-DOR software requires only Type 1 or Type 13 SPKs**
  - **This restriction is likely to be lifted by approximately November 2009, maybe even sooner.**

SPK Reading: Efficiency Issues - 1

Navigation and Ancillary Information Facility

- **SPK file creators should design files to support efficient read access.**
  - 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
  - Number of segments for a body in one SPK file
  - 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 segments.**
      - **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 30,000.**
        - **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.**

continued on next page

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 1000 SPK files may be loaded at one time by an application.
    - » The "1000" 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 1000 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 1000 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 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 2012

Summary

- **SPICE provides means to create CK files, either by packaging orientation computed elsewhere or by first computing orientation and then packaging it in a CK file**
- **Packaging of <u>already existing</u> 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 attitude data to a CK using the Toolkit's *msopck* program**
- **<u>Computing as well as packaging</u> 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 attitude using SPICE routines is not discussed here**
  - **Convert orientation rules and schedules to a CK using the *prediCkt* program available at the NAIF website**

CK Writer Routines

- 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)
- Only the Type 3 writer is discussed in this tutorial
  - Writers for Types 1 and 2 have very similar interfaces
  - Types 4 and 5 are are not commonly used

Type 3 Writer Example - 1

- **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,
         "reference_frame", avflag, "segment_id",
         nrec, sclkdp, quats, avvs, nints, starts );

ckcls_c ( handle );
```

Type 3 Writer Example - 2

- `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

- `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.
- `quats` - a list of SPICE quaternions that rotate vectors from the base frame specified by the `ref` argument to the `inst` frame.
  - `m2q_c ( C_matrix, quaternion );`
- `avvs` - angular rate vectors given in the base frame specified by the `ref` argument.
- `starts` - a list of SCLK ticks indicating the start of interpolation intervals. They must correspond to entries in `sclkdp`.
- `nints` - number of entries in starts.

Type 3 writer - Making Up Rates

- **One of the easiest ways to accomplish this 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`.**

continues on next page

Type 3 Writer - Making Up Rates (2)

- Constructing angular rates in this fashion assumes that no more than a 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 particular approach, however.

MSOPCK

- *msopck* is a program for making CK files from orientation provided as a time tagged, space-delimited table in a text file
- *msopck* can process quaternions (SPICE and non-SPICE flavors), Euler angles, or matrixes, tagged with UTC, SCLK, or ET
- *msopck* requires all program directives 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

MSOPCK
List of Setup File Keywords

Navigation and Ancillary Information Facility

Supporting Kernels/Files

```
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'
```

Output CK Specifications

```
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 data Specifications

```
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 'RADIANS'
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_ROTATION_ANGLES   = (  angle3,  angle2,  angle1 )
OFFSET_ROTATION_AXES     = ( 'axis3', 'axis2', 'axis1' )
OFFSET_ROTATION_UNITS    = 'DEGREES' or 'RADIANS'
DOWN_SAMPLE_TOLERANCE    = down sampling tolerance, radians
INCLUDE_INTERVAL_TABLE   = 'YES' or 'NO' (default 'YES')
```

Optional and conditional keywords are shown in green

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 = '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)

- **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 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 and provides on request a “white paper” explaining differences between various quaternion styles.

MSOPCK - Input Details (3)

- **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 uniform rotation algorithm implemented in Type 3 CK, 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, SCLK ticks or ET, as specified using the INPUT_TIME_TYPE keyword**
  - Time tags must not have embedded spaces

MSOPCK - Output Details (1)

- *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 spinners
    - » 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 given in 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

```
Vinst = [ROTinput] * [ROToffset] * Vref
```

MSOPCK - Output Details (2)

- The time tags may be adjusted by a constant value specified in seconds using the TIME_CORRECTION 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)

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

- *prediCkt* is a program for making CK files from a set of orientation specification rules, and schedules defining when these rules are to be followed
- *prediCkt* has a simple command line interface
- *prediCkt* requires orientation and schedule specification to be provided in a setup file that follows the SPICE text kernel syntax
- *prediCkt* requires the names of all supporting kernels -- SPK, PCK, etc -- be provided in a meta-kernel (a “furnsh kernel”)
- *prediCkt* is available only from the Utilities link of the NAIF webpages

PREDICKT - Usage

- *prediCkt* has the following command line arguments

```
prediCkt -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

- 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 following 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 spec file 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

- **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 spec file 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)

- **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)

- 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 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
  - "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
\begindata
         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_THEMIS_IR = CameratoMars'
         CK-53000START        += @2004-FEB-10-00:00
         CK-53000STOP         += @2004-FEB-15-00:00
\begintext
```

PREDICKT - Example (2)

Navigation and Ancillary Information Facility

Terminal Window

```
$ cat m01_map_nadir.furnsh
\begindata
   KERNELS_TO_LOAD = ( 'naif0007.tls'
                       'm01_v26.tf'
                       'mar033-5.bsp'
                       'm01_map_rec.bsp'
                       'm01.tsc' )
\begintext
$ 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 2012

Note: the EK is infrequently used by NASA flight projects.

Only a brief overview of the EK subsystem is provided.

Scope

- This tutorial provides an overview of the entire Events Kernel subsystem, comprised of three components:
  - Science Plan ESP
  - Sequence ESQ
  - Notebook ENB
- Depending on specific circumstances:
  - the three components may exist as three distinct and different products
  - two components may be implemented with a single mechanism
  - one or more components may not be used at all

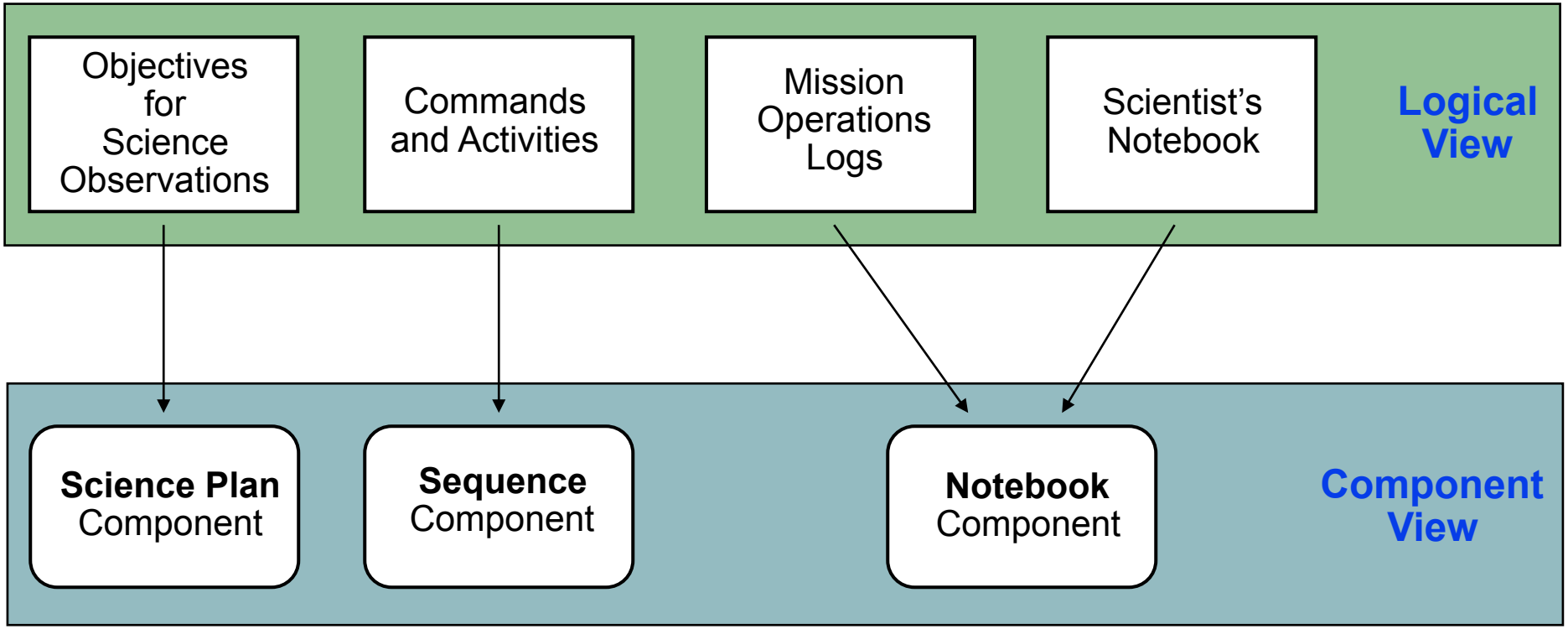
E-Kernel Subsystem Objectives

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**

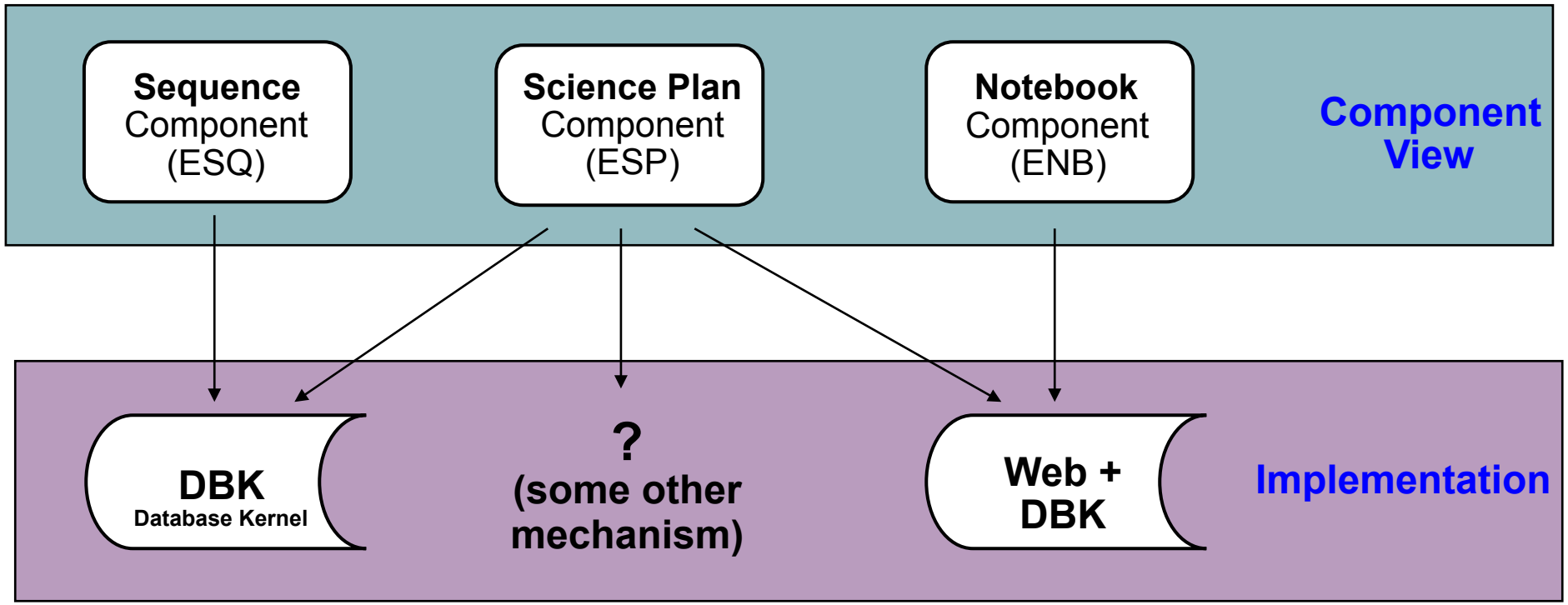
Nominal E-kernel Composition

Navigation and Ancillary Information Facility



Nominal E-kernel Implementation

Navigation and Ancillary Information Facility



Science Plan - ESP

- 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

- This component 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

- **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.)

Notebook - ENB

- 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
- 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

- The E-kernel is the least well developed and least used component of the SPICE system
  - Not of as much interest to flight project instrument and engineering teams as 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 input 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 2012

Outline

- Work in progress
- Future possibilities
- Your suggestions?

Work In Progress - 1

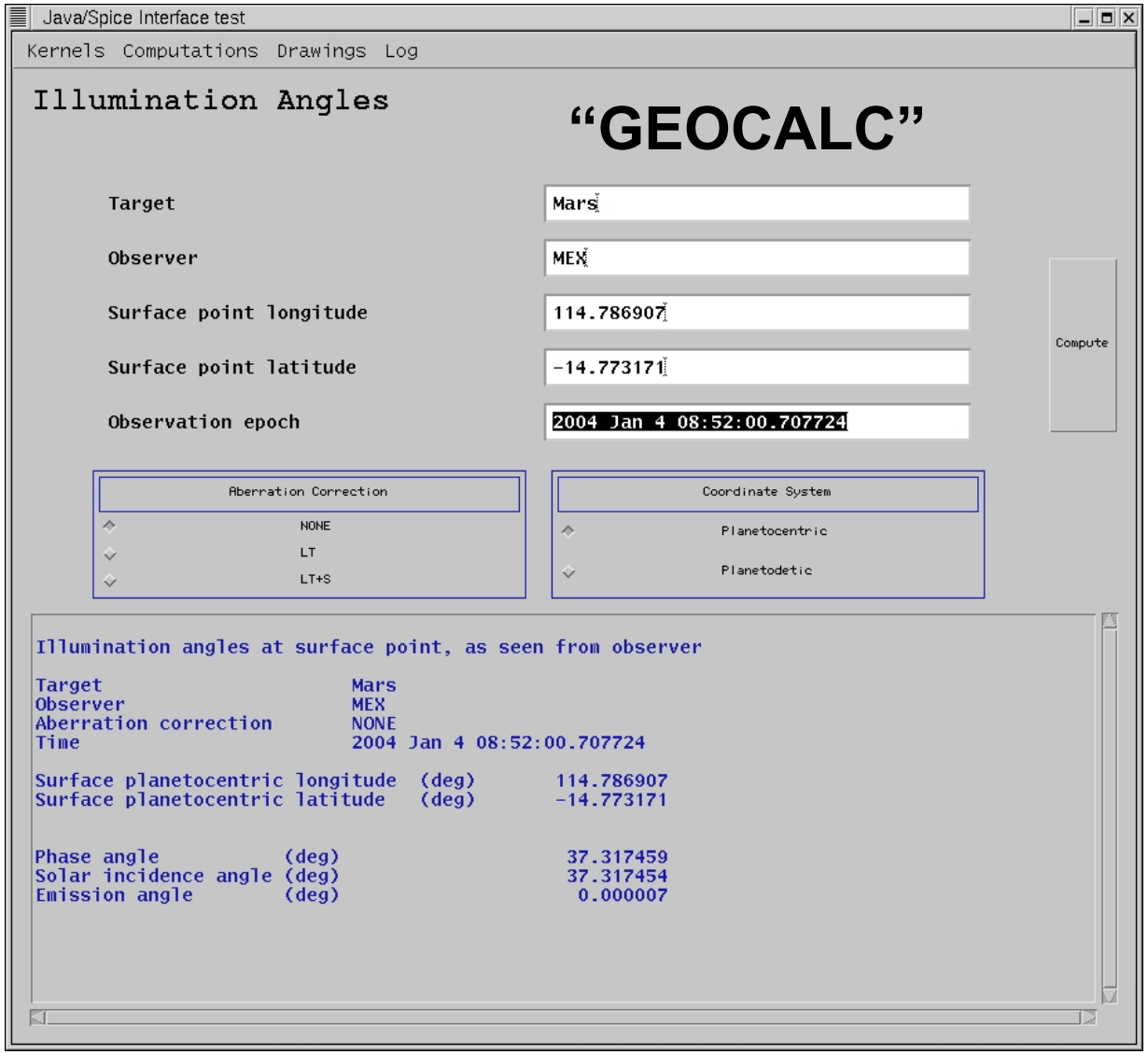
- **Extension of the shape model subsystem**
  - The task is to add two new shape model capabilities…
    - » tessellated plate model, for small, irregularly shaped bodies
    - » digital elevation model

    to the existing tri-axial shape model found in PCK

  - Status
    - » An alpha-test prototype of the plate model has been given to several projects and persons
    - » Date for release of a “final” version is not yet determined

Work in Progress - 2

Navigation and Ancillary Information Facility



Provide a GUI interface to many SPICE computations. Availability date is unknown.

In this example, compute the illumination angles on Mars at LON 114.7 and LAT -14.7 as seen from Mars Express on 2004 JAN 4 08:52:00. The user can pick either a planetocentric or planetodetic reference frame.

Work in Progress - 3

- **Java Native Interface (JNISpice)**
  - An alpha-test release was made in February, 2010
  - Official addition to the Toolkit perhaps in the Fall of 2012
- **Python**
  - Considerable prototyping has been done
  - Whether or not this effort will proceed, and when, is uncertain

Some Other Possibilities - 1

- Provide a GUI tool that will contrast a set of SPK files, thus aiding you in selecting the one(s) of interest
- Provide a GUI tool for easier creation of a SPICE frame, and subsequent visualization thereof
- Provide a “predict spk” tool that makes it easy to construct an SPK file from simple rules
- Add more high-level computations, such as instrument footprint coverage
- Provide a star catalog integrated with SPICE capabilities

Some Other Possibilities - 2

- Additional target models: rings, gravity, atmosphere, magnetosphere, …
- Develop a more flexible and extensible instrument modeling mechanism

What do You Suggest?

- **NAIF solicits suggestions from the user community.**
  - Caution: we're a small team and have a large backlog, so we can't promise any particular action.
- **We're interested in programmatic ideas as well as technical ones.**
  - Should NAIF promote use of SPICE beyond NASA's planetary science program?
  - What amount of cooperation and interoperability with foreign partners is appropriate and achievable?

Navigation and Ancillary Information Facility

Shape Model Subsystem Preview (DSK)

January 2012

SPICE DSK Topics

- Overview
- Requirements
- DSK Data Representations
- DSK System Components
- DSK Software Components
- DSK API Examples
- Using Shape with Orientation Data
- DSK Development Status

Overview

- NAIF is developing a new SPICE kernel type
  - DSK: Digital Shape Kernel
- The DSK subsystem deals with data sets describing topography of solar system objects, or more generally, shapes of 3-dimensional objects. Examples:
  - Digital elevation models (DEM) for the surfaces of Mars or the Moon
  - Tessellated plate model for the surface of a natural satellite, asteroid or comet nucleus
- The DSK system facilitates high-accuracy, SPICE-based geometric computations using “complex” shape data
  - Currently SPICE uses only triaxial ellipsoid shape models.

Requirements -1

- All "requirements" listed here are of an informal nature
  - Derived from customer interaction and NAIF team members' experience using SPICE
- Overall requirement: facilitate high-accuracy geometry computations involving realistic models of surfaces of extended bodies.
- Examples of computations that should be supported:
  - » Location of "sub-observer point" and height of observer above surface
  - » Ray-surface intercept point
  - » Occultation/transit state of a point target
  - » Limb and terminator location
  - » Illumination angles at a specified surface point
  - » Determine if a target is in an instrument's field of view (FOV)

Requirements -2

- System should support efficient random access data search
  - For example: for a given (LONGITUDE, LATITUDE) coordinate pair, return radius (distance from body center) of the corresponding surface point
- System should support rapid, high volume data extraction ("bulk read")
  - Required for efficient use by graphics applications
- System should be able to use data sets spread across multiple files
  - Some current data sets exceed 2Gbytes in size
  - Larger data sets should be expected in the future
  - Impractical to store all needed data in one file
- System should be able to work with models for different bodies simultaneously.
  - For example: support simultaneous use of data sets for Mars and Phobos.

Requirements -3

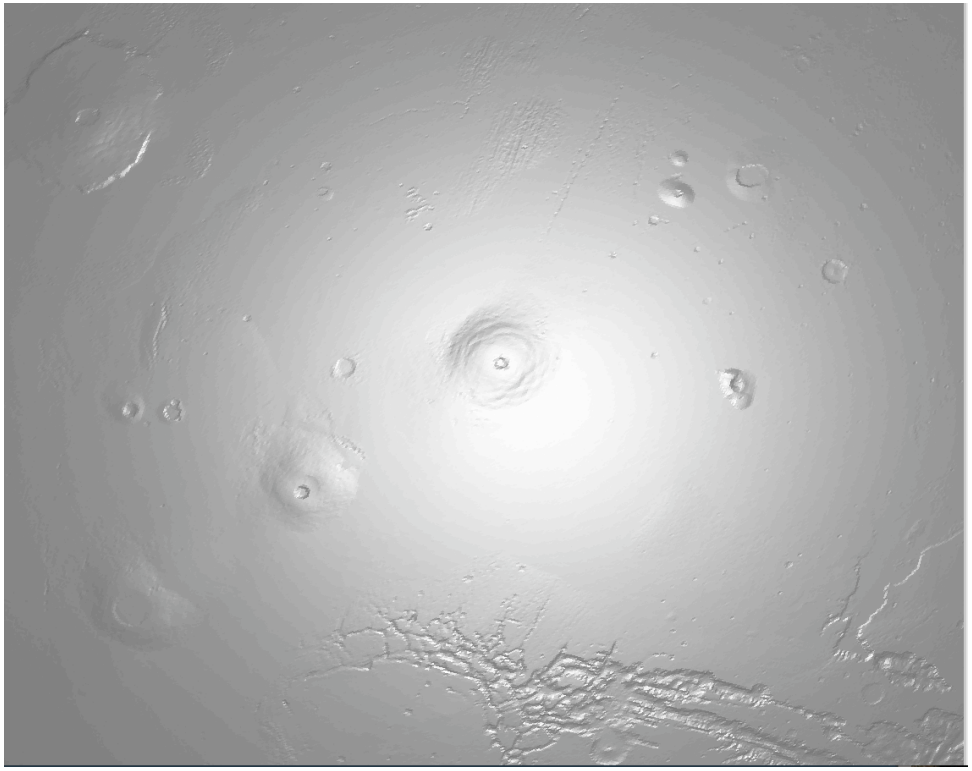
- System should be able to work with multiple models for different parts of the surface of a specified body simultaneously.
  - Support simultaneous use of multiple data sets having different resolutions, or even different mathematical representations, for different regions of the surface.
- DSK files should be portable
- DSK files should support inclusion of metadata
- Tools should be provided for:
  - summarizing contents of DSK files
  - accessing metadata in DSK files
  - merging or subsetting DSK files
  - ingesting data from other types of files. Examples:
    - » Dr. Robert Gaskell's ASCII plate model files
    - » Dr. Peter Thomas' ASCII format used for Vesta model
    - » DLR ASCII formats used for Phobos

DSK Shape Representations

- The DSK subsystem will handle two representations of shape data
  - Digital elevation model
  - Tessellated plate model
- DSK supplements the only existing SPICE shape model: tri-axial ellipsoid

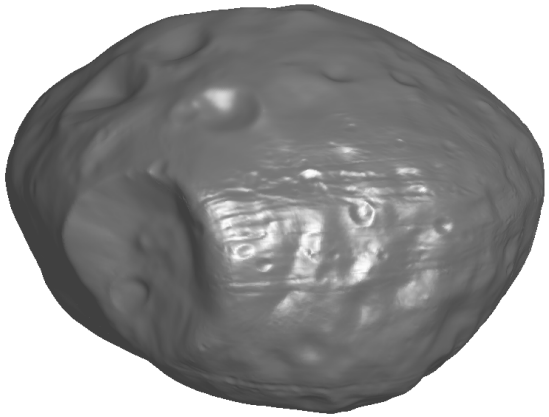
DSK Data Representations -1

- Digital elevation model (DEM)
  - Maps longitude/latitude to “elevation”
    - » Elevation of a surface point can be defined as distance from the origin of a body-fixed reference frame
    - » Elevation can be defined as height above a reference ellipsoid
  - Example: image created from MGS laser altimeter (MOLA) Mars DEM

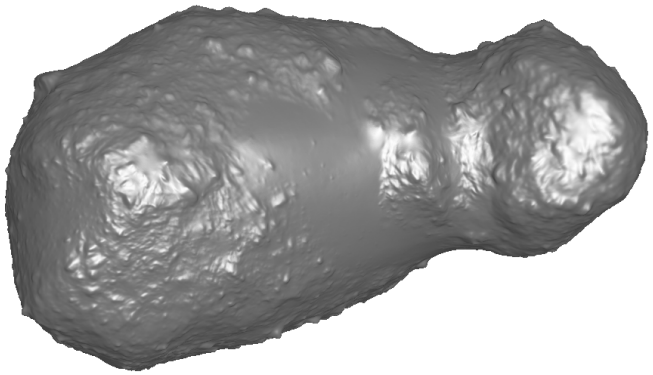


DSK Data Representations -2

- **Tessellated plate model**
  - **Surface of object is represented as a collection of triangular plates**
  - **More flexible than digital elevation model: 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 of similar resolution in terms of storage and computational speed**



Phobos

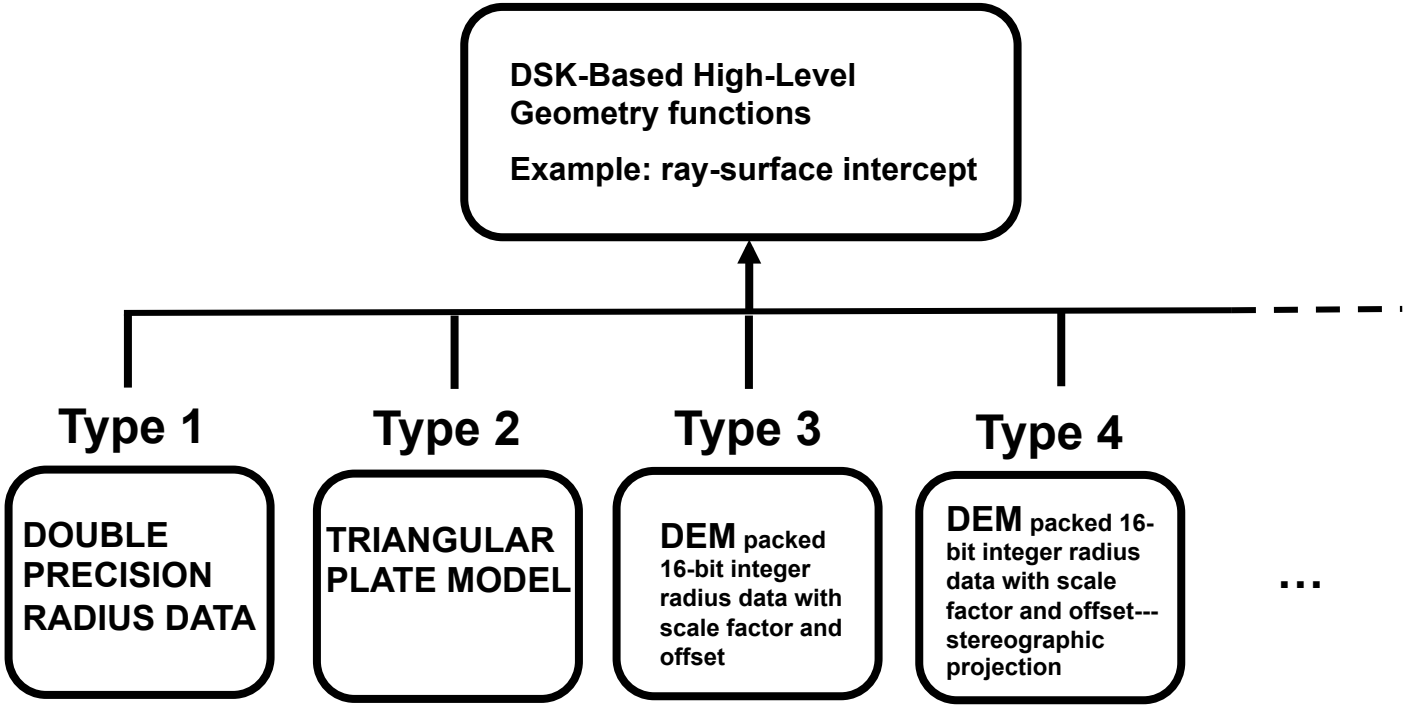


Itokowa

DSK Data Representations -3

Navigation and Ancillary Information Facility

- **DSK shape representations are polymorphic:**
  - DSK shape representations are called "Data Types."
  - Each data type has its own mathematical representation of a surface
  - Each data type has associated software that implements common functionality, such as the ability to return a radius (distance of surface point from body center) value for a specified latitude and longitude.
  - Each data type may have additional, unique functionality.
    - » For example, type 2 has accessor routines that return plate and vertex data. These functions are not applicable to other data types.



DSK System Components

- **DSK Files**
  - Built upon the SPICE DAS file architecture
    - » Binary, direct access
    - » System-independent buffering built in
    - » Comment area built in
- **DSK Software**
  - SPICE software which enables users to create and use DSK kernels
    - » Writer routines
    - » Reader routines
    - » High-level API routines
      - For example: routines dealing with observer-target geometry
    - » Supporting utility programs

DSK Software Components -1

Navigation and Ancillary Information Facility

- **Writers**
  - Routines that enable a SPICE-based application to create a DSK kernel
    - » Open new DSK kernel for write access
    - » Open existing DSK kernel for write access
    - » Start new DSK segment ("segments" are partial DSK data sets containing data for a given region on a specified object)
    - » Add data to DSK segment
- **Readers**
  - Routines that extract data from a DSK file
    - » Return elevation of surface at given longitude/latitude
    - » Return specified attributes, for example the surface normal vector, for a specified longitude and latitude
    - » Rapidly obtain data for large portion of surface ("bulk read")
    - » Return DSK attributes such as number of plates, pixel size, min/max elevation, etc.
- **High-level functions (including, but not limited to, the following):**
  - Compute sub-observer point on surface and height of observer above surface
  - Compute intercept of ray with surface
  - Determine whether a portion of a target body's surface is within the FOV of specified instrument at specified time.
  - Determine occultation/transit state of a point target
  - Compute limb and terminator location
  - Compute Illumination angles at a specified surface point

DSK Software Components -2

- **Utility programs that**
  - Create DSK files: import other surface shape data sets into SPICE DSK format
  - Port DSK files
  - Provide comment area access
  - Summarize DSK file contents
  - Subset or merge DSK files
  - Downsample DSK files
  - Convert one DSK data type to another
    - » Example: create type 2 DSK file from type 1

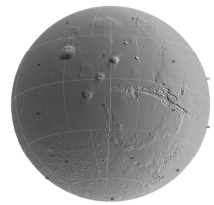
DSK API Examples

- **Get radius at surface point (inputs are in red, outputs in blue):**
  - CALL DSKRAD (TARGET, LON, LAT, RADIUS)
    - » Inputs: target body name, longitude and latitude of point of interest
    - » Output: radius (distance from target center) at surface point
- **Find sub-observer point on target:**
  - CALL SUBPT (METHOD, TARGET, ET, ABCORR, OBSRVR, SPOINT, ALT)
    - » SUBPT is a generic, high-level API. SUBPT doesn't assume the surface is modeled by a DSK.
    - » Input "METHOD" indicates surface model and sub-point definition
      - For ellipsoids, METHOD may be set to 'near point' or 'intercept'
      - For DSKs, set METHOD to 'DSK intercept', indicating that the sub-point is defined as the closest intersection to the observer of the observer-target center ray with the surface, and a DSK shape model is to be used.
      - Note that SPICE should not assume DSK is to be used just because a DSK for the target body is loaded; may be too inefficient for some applications. Caller must say which model is to be used.
    - » Other inputs: target body name, epoch, aberration correction, observer name.
    - » Outputs: sub-observer point in Cartesian coordinates, expressed in the body-fixed frame associated with the target, and altitude of the observer above the sub-point.

Writing Shape and Orientation Kernels

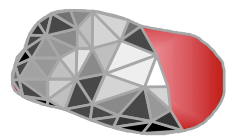
Navigation and Ancillary Information Facility

LAT/LON and height above ellipsoid or distance from center of frame → **MKDSK Program** (SPICE Toolkit) →



Digital Terrain Shape Model

Lists of plate model vertices and associated plates, and optionally, albedo data for each plate → **MKDSK Program** (SPICE Toolkit) →

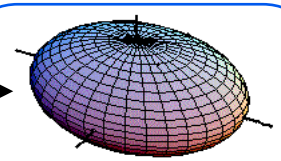


Tessellated Plates Shape Model

DSK

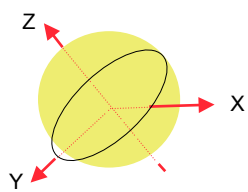
Digital shape kernel

Axes dimensions for tri-axial ellipsoid → **Text editor** (Usually done by NAIF) →



Triaxial Ellipsoid Shape Model

Some source of rotation state information (pole RA/DEC and prime meridian location) → **Text editor** (Usually done by NAIF) →



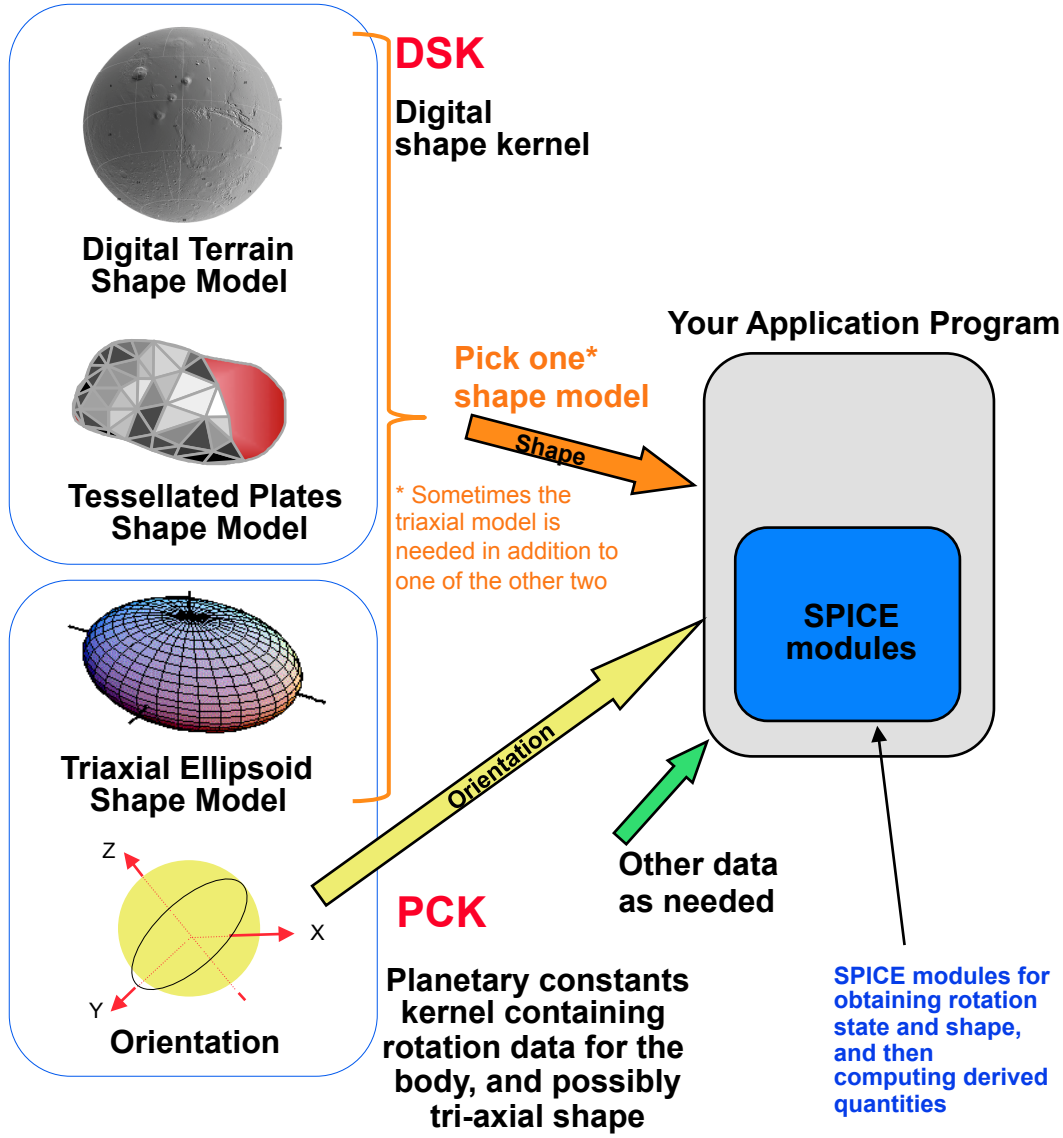
Orientation

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



DSK Development Status

- **History**
  - Alpha Test DSK Toolkit was released in July, 2010.
    - » Available in Fortran, C, and IDL.
    - » Contains support only for plate model data type
    - » The DSK File format used in this prototype has been finalized; the format will be supported by the official SPICE Toolkit.
- **Plans**
  - Release date of a first official version of the DSK system is TBD.

SPICE Documentation Taxonomy

Arranged by Category

January 2012

General Reading, including installing the SPICE Toolkit

Document Name	File Name	Location	Description/Comments
Installing the SPICE Toolkit	Installing_toolkit	TUTOR	A collection of viewgraph-style packages providing tutorial information on nearly all components of the SPICE system.
Instructions for getting the Toolkit components from NAIF's FTP server	README	GETTK	Description of the files to be FTP'd in order to get, install and use the SPICE Toolkit.
Preparing your computer for programming with SPICE	Preparing_for_programming	TUTOR	
Toolkit Contents	dscriptn.txt	T	Describes the structure and contents of the Toolkit
New features and major changes	whats.new	T	Describes significant new features added to the Toolkit since the last version.

TUTOR = Tutorials on NAIF web pages (http://naif.jpl.nasa.gov/tutorials.html)
T = SPICE Toolkit, in the /doc subdirectory
GETTK = Go to the desired language and then platform under ftp://naif.jpl.nasa.gov/pub/naif/toolkit

General SPICE Programming

Document Name	File Name	Location	Description/Comments
Must Useful SPICE Subroutines	Mostused.html	T	Practical but terse specifications, including examples, for many popular routines.
Permuted Index	spicelib_idx.html, cspice_idx.html, icy_idx.html, mice_idx.html	T	Permuted index built from the “Brief Abstract” found in every routine. Helps focus your search for a routine that meets your needs.
CSPICE Required Reading ICY Required Reading MICE Required Reading	cspice icy mice	T	A discussion of how the product is produced and how to use it. (Note: only cspice is available in C toolkits; cspice and icy are available in IDL toolkits; cspice and mice are available in MATLAB toolkits.)
Summary of Key Points	summary_of_key_points	TUTOR	Tips for getting started on programming with SPICE modules.
Module headers (Also wee "wrappers" below)	*.f or *_c.c	T	Each module (subroutine) in SPICELIB and CSPICE contains an extensive "header" providing the detailed specifications for the routine needed by a programmer. Examples are included.
ICY and MICE “wrappers” around corresponding CSPICE modules	…/doc/html/<mice or icy>/index.html	T	IDL and MATLAB interface “wrappers,” providing some instructions on use of the named API. More detail is provided in the corresponding CSPICE header.
NAIF IDs Required Reading	naif_ids	T	A summary of numeric ID codes used throughout the SPICE system
Error Required Reading	error	T	Reference for configuring and using the exception handling system built-in to SPICELIB and CSPICE
Common Problems Required Reading	problems	T	A discussion of the most commonly encountered problems using SPICE

TUTOR = Tutorials on NAIF web pages (http://naif.jpl.nasa.gov/tutorials.html)
T = SPICE Toolkit:
Plain text, in the /doc subdirectory
HTML under the /doc/html/… subdirectory

Ephemerides for spacecraft and solar system bodies (SPK Subsystem)

Document Name	File Name	Location	Description/Comments
SPK Tutorial	spk	TUTOR	Tutorial on using SPK files
Making an SPK Tutorial	making_an_spk	TUTOR	Tutorial on making an SPK file
Using Frames Tutorial	using_frames	TUTOR	Tutorial on using frames, including in SPK routines
SPK Required Reading	spk	T	Reference for the SPK subsystem
Frames Required Reading	frames	T	Reference for working with reference frames
NAIF IDs Required Reading	naif_ids	T	Summarizes numeric ID codes used throughout the SPICE system
SPC Required Reading	spc	T	Reference for use of the "comment area" in binary kernels
Kernel Required REading	kernel	T	Reference for general specifications of text kernels
BRIEF User's Guide	brief	T	BRIEF produces a concise summary of the contents/coverage of an SPK file.
SPACIT User's Guide	spacit	T	SPACIT provides file conversion, detailed summarization and read access to internal comments (metadata).
Convert User's Guide	conver	T	Describes use of the command line utilities named TOBIN and TOXFR used to convert binary kernels to transfer format and vice-versa.
Comment User's Guide	commnt	T	Comment is used to add, extract, read and delete comments (metadata) in binary kernels.
SPK Merge User's Guide	spkmerge	T	SPKMERGE is a utility program used to merge two or more SPK files, or to subset a single SPK file.
SPKDIFF User's Guide	spkdiff	T	SPKDIFF computes differences between geometric states obtained from two SPK files and either displays these or shows statistics about them.
SPY User's Guide	spy.txt	UTIL	SPY is a utility for validating, inspecting and analyzing SPKs.
MKSPK User's Guide	mkspk	T	MKSPK is a utility for making an SPK file from a set of state vectors or conic elements or two-line elements.

TUTOR = Tutorials on NAIF web pages (http://naif.jpl.nasa.gov/tutorials.html)
T = SPICE Toolkit:
Plain text, in the /doc subdirectory
HTML under the /doc/html/… subdirectory
UTIL = Utilities link on the NAIF website

Target body size, shape and orientation (PCK Subsystem)

Document Name	File Name	Location	Description/Comments
PCK Tutorial	pck	TUTOR	Tutorial viewgraphs on using PC-kernels
High Accuracy Orientation and Body-fixed Frames for the Moon and Earth tutorial	lunar-earth_pck-fk	TUTOR	Tutorial viewgraphs on special orientation files (binary PCKs) and body-fixed frames for the moon and the earth
PCK Required Reading	pck	T	Reference for the PCK subsystem
Frames Required Reading	frames	T	Reference for working with reference frames
NAIF IDs Required Reading	naif_ids	T	Summarizes numeric ID codes used throughout the SPICE system
Kernel Required Reading	kernel	T	Reference for general specifications of text kernels

TUTOR = Tutorials on NAIF web pages (http://naif.jpl.nasa.gov/tutorials.html)
T = SPICE Toolkit:
Plain text, in the /doc subdirectory
HTML under the /doc/html/… subdirectory

Instrument Information Pertinent to SPICE (IK Subsystem)

Document Name	File Name	Location	Description/Comments
IK Tutorial	ik	TUTOR	Tutorial viewgraphs on using I-kernels
IK Required Reading	ik	--	(Not yet written!)
n/a	*.ti	D	Look at an existing I-kernel; these are text files that contain substantial internal documentation
Frames Required Reading	frames	T	Reference for working with reference frames
NAIF IDs Required Reading	naif_ids	T	Summarizes numeric ID codes used throughout the SPICE system
Kernel Required Reading	kernel	T	Reference for general specifications of text kernels

D = Project Data on NAIF web pages (http://naif.jpl.nasa.gov/naif/data.html)
TUTOR = Tutorials on NAIF web pages (http://naif.jpl.nasa.gov/tutorials.html)
T = SPICE Toolkit:
Plain text, in the /doc subdirectory
HTML under the /doc/html/… subdirectory

Orientation of a Spacecraft or Structure (CK Subsystem)

Document Name	File Name	Location	Description/Comments
CK Tutorial	ck	TUTOR	Tutorial viewgraphs on using C-kernels
Using Frames Tutorial	using_frames	TUTOR	Tutorial on using frames, including in transformation modules
CK Required Reading	ck	T	Reference for the CK subsystem
Frames Required Reading	frames	T	Reference for working with reference frames
NAIF IDs Required Reading	naif_ids	T	Summarizes numeric ID codes used throughout SPICE
SPC Required Reading	spc	T	Reference for use of the "comment area" in binary kernels
Rotations Required Reading	rotation	T	Reference for construction and use of rotation matrices within the SPICE context
CKBRIEF User's Guide	ckbrief	T	CKBRIEF produces a concise summary of the contents/coverage of an SPK file.
SPACIT User's Guide	spacit	T	SPACIT provides file conversion, detailed summarization and read access to internal comments (metadata).
Convert User's Guide	convert	T	Describes use of the command line utilities TOBIN and TOXFR used to convert binary kernels to transfer format and vice-versa.
Comment User's Guide	commnt	T	COMMENT is used to add, extract, read and delete comments (metadata) in binary kernels.
DAFCAT User's Guide	dafcat	UTIL	DAFCAT provides a very simple and simplistic file merge capability for CK files.
CKslicer User's Guide	ckslicer.txt	UTIL	CKSLICER subsets a CK into another CK file.
CKsmrg	chsmrg.txt	UTIL	CKSMRG merges segments in Type 3 CK files.
MSOPCK User's Guide	msopck	T	MSOPCK is a utility for making a CK file from orientation data in the form of quaternions, Euler angles or rotation matrices.
FRMDIFF User's Guide	frmdiff	T	Provides a statistical comparison of the orientations of two frames, one or both of which might be specified using CK(s).

TUTOR = Tutorials on NAIF web pages (http://naif.jpl.nasa.gov/tutorials.html)
T = SPICE Toolkit:
Plain text, in the /doc subdirectory
HTML under the /doc/html/… subdirectory
UTIL = Utilities link on the NAIF website

Connectivity of Reference Frames (FK Subsystem)

Document Name	File Name	Location	Description/Comments
Frames Tutorial	fk	TUTOR	Tutorial viewgraphs on contents of a Frames kernel
Using Frames	using_frames	TUTOR	Tutorial viewgraphs on using Frames kernels
Dynamic Frames	dynamic_frames	TUTOR	Tutorial on defining/implementing custom so-called dynamic frames
n/a	*.tf	D	Look at an existing Frames kernel; these are text files and contain substantial internal documentation
Frames Required Reading	frames	T	Reference for the Frames subsystem
NAIF IDs Required Reading	naif_ids	T	Summarizes numeric ID codes used throughout the SPICE system
Rotations Required Reading	rotation	T	Reference for construction and use of rotation matrices within the SPICE context
Kernel Required Reading	kernel	T	Reference for general specifications of text kernels
FRMDIFF User's Guide	frmdiff	T	Provides a statistical comparison of the orientations of two frames, one or both of which might be specified using CK(s).

D = Project Data on NAIF web pages (http://naif.jpl.nasa.gov/naif/data.html)
TUTOR = Tutorials on NAIF web pages (http://naif.jpl.nasa.gov/tutorials.html)
T = SPICE Toolkit:
Plain text, in the /doc subdirectory
HTML under the /doc/html/… subdirectory

"EVENTS", broken down into three sub-products (EK Subsystem)

Document Name	File Name	Location	Description/Comments
Introduction to EK subsystem	ek_intro	TUTOR	Tutorial Introduction to the Events subsystem
EK Required Reading	ek	T	Reference for the Events-kernel subsystem

TUTOR = Tutorials on NAIF web pages (http://naif.jpl.nasa.gov/tutorials.html)
T = SPICE Toolkit:
Plain text, in the /doc subdirectory
HTML under the /doc/html/… subdirectory

Time Conversion

Document Name	File Name	Location	Description/Comments
Time Tutorial	time	TUTOR	Tutorial viewgraphs on time conversions
Time Required Reading	time	T	Reference on time systems (excluding SCLK)
SCLK Required Reading	SCLK	T	Reference on spacecraft clock time
CHRONOS User's Guide	chronos	T	CHRONOS is a full-featured, flexible time conversion utility program
Kernel Required Reading	kernel	T	Reference for general specifications of text kernels

TUTOR = Tutorials on NAIF web pages (http://naif.jpl.nasa.gov/tutorials.html)
T = SPICE Toolkit:
Plain text, in the /doc subdirectory
HTML under the /doc/html/… subdirectory

Geometry Finder (GF Subsystem)

Document Name	File Name	Location	Description/Comments
Geometry Finder Tutorial	geometry_finder	TUTOR	Tutorial viewgraphs on geometry finder subsystem
Geomeetry Finder Required Reading	gf	T	Reference on geometry finder

TUTOR = Tutorials on NAIF web pages (http://naif.jpl.nasa.gov/tutorials.html)
T = SPICE Toolkit:
Plain text, in the /doc subdirectory
HTML under the /doc/html/… subdirectory