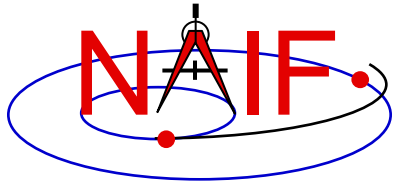


---

Navigation and Ancillary Information Facility

# Writing an Icy (IDL) Based Program

January 2018

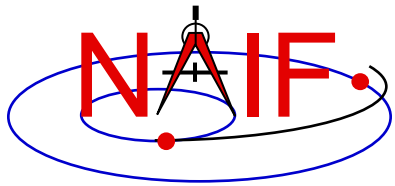


# Viewing This Tutorial

---

Navigation and Ancillary Information Facility

Undefined variables are displayed in **red**  
Results are displayed in **blue**



# Introduction

---

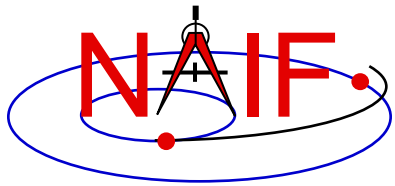
## Navigation and Ancillary Information Facility

First, let's go over the important steps in the process of writing an Icy-based program and putting it to work:

- Understand the geometry problem.
- Identify the set of SPICE kernels that contain the data needed to perform the computation.
- Select the SPICE APIs needed to compute the quantities of interest.
- Write and compile the program.
- Get actual kernel files and verify that they contain the data needed to support the computation for the time(s) of interest.
- Run the program.

To illustrate these steps, let's write a program that computes the apparent intersection of the boresight ray of a given CASSINI science instrument with the surface of a given Saturnian satellite. The program will compute:

- Planetocentric and planetodetic (geodetic) latitudes and longitudes of the intercept point.
- Range from spacecraft to intercept point.
- Illumination angles (phase, solar incidence, and emission) at the intercept point.



# Observation geometry

Navigation and Ancillary Information Facility

We want the boresight intercept on the surface, range from s/c to intercept, and illumination angles at the intercept point.

When? **TIME (UTC, TDB or TT)**

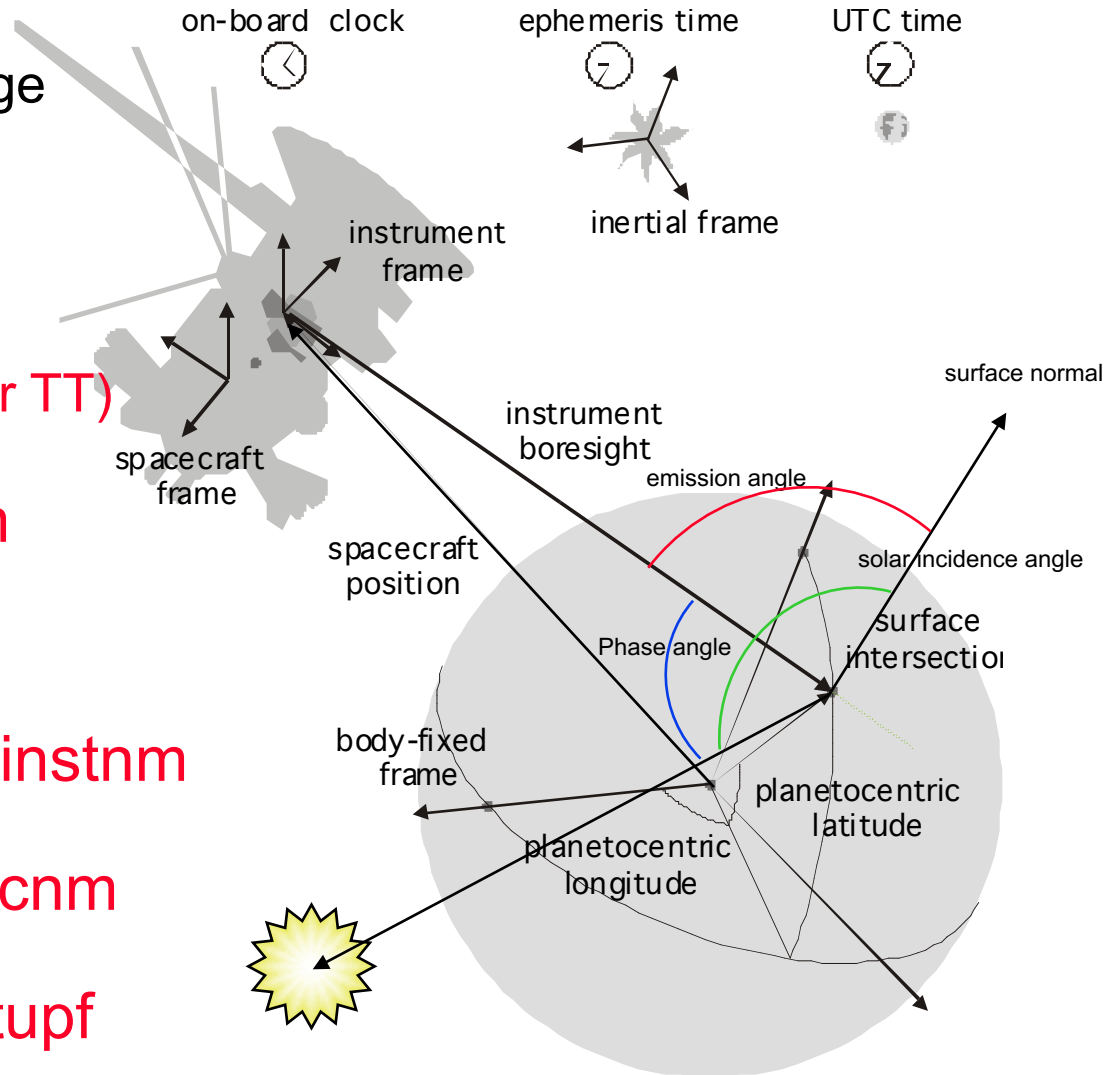
On what object? **satnm**

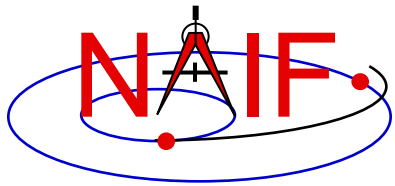
In what frame? **fixref**

For which instrument? **instnm**

For what spacecraft? **scnm**

Using what model? **setupf**





# Needed Data

Navigation and Ancillary Information Facility

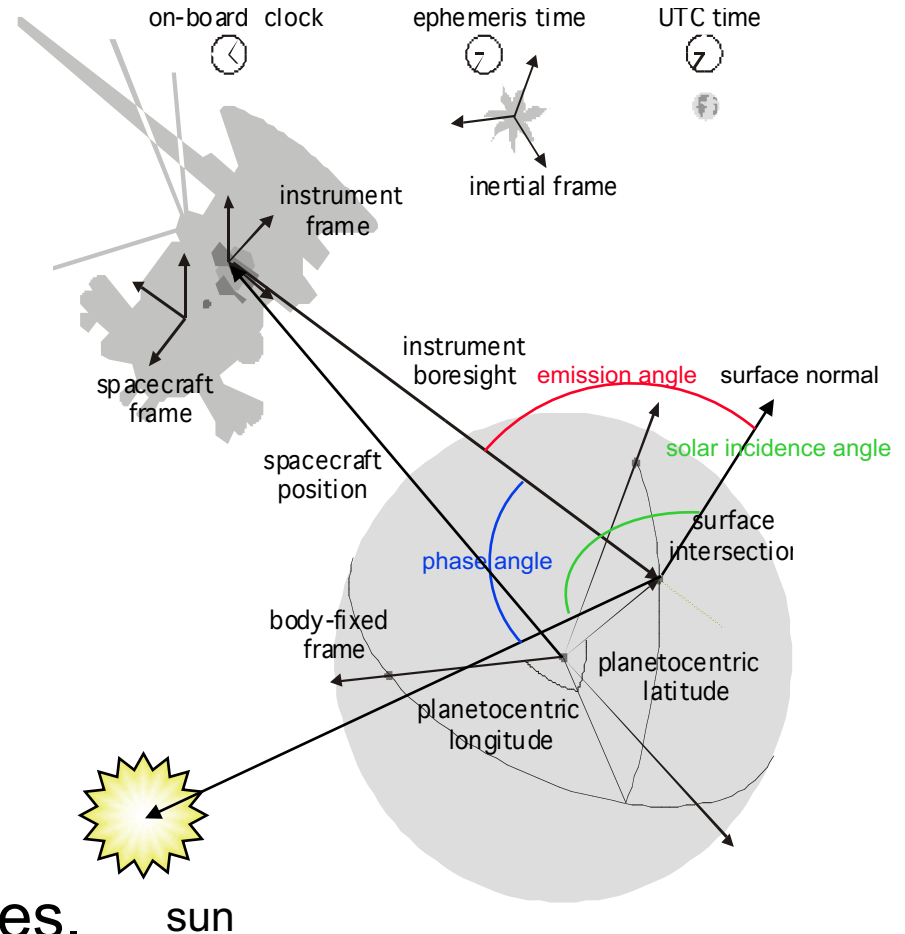
Time transformation kernels

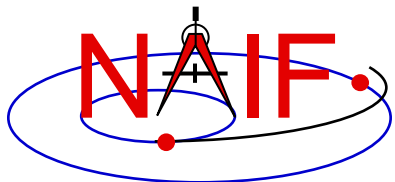
Orientation models

Instrument descriptions

Shapes of satellites, planets

Ephemerides for spacecraft,  
Saturn barycenter and satellites.





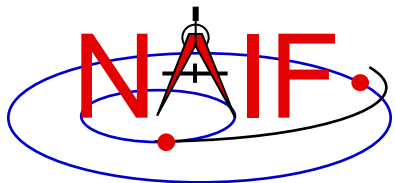
# Which Kernels are Needed?

## Navigation and Ancillary Information Facility

Data required to compute vectors, rotations and other parameters shown in the picture are stored in the SPICE kernels listed below.

Note: these kernels have been selected to support this presentation; they should not be assumed to be appropriate for user applications.

Parameter	Kernel Type	File name
-----	-----	-----
time conversions	generic LSK CASSINI SCLK	naif0009.tls 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.

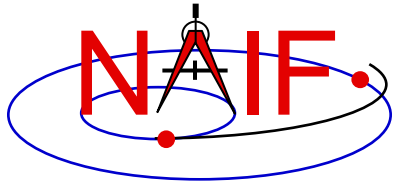
```
\begindata
```

```
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:

```
read, setupf, PROMPT='Enter setup file name > '  
cspice_furnsh, setupf
```



# Programming Solution

---

## Navigation and Ancillary Information Facility

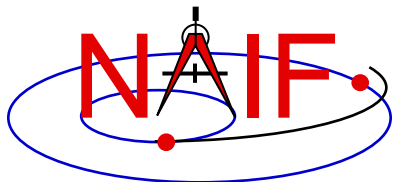
- **Prompt for setup file (“metakernel”) name; load kernels specified via setup file. (Done on previous chart.)**
- **Prompt for user inputs required to completely specify problem. Obtain further inputs required by geometry routines via lcy calls.**
- **Compute the intersection of the boresight direction ray with the surface of the satellite, presented as a triaxial ellipsoid.**

**If there is an intersection,**

- **Convert Cartesian coordinates of the intersection point to planetocentric latitudinal and planetodetic coordinates**
- **Compute spacecraft-to-intercept point range**
- **Find the illumination angles (phase, solar incidence, and emission) at the intercept point**
- **Display the results.**

**We discuss the geometric portion of the problem first.**





# Compute surface intercept

Navigation and Ancillary Information Facility

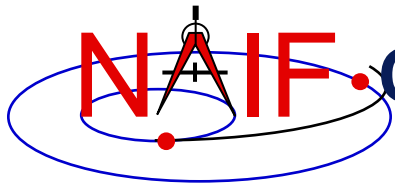
Compute the intercept point (`point`) of the boresight vector (`insite`) specified in the instrument frame (`iframe`) of the instrument mounted on the spacecraft (`scnm`) with the surface of the satellite (`satnm`) at the TDB time of interest (`et`) in the satellite's body-fixed frame (`fixref`). This call also returns the light-time corrected epoch at the intercept point (`trgepc`), the spacecraft-to-intercept point vector (`srfvec`), and a flag indicating whether the intercept was found (`found`). We use "converged Newtonian" light time plus stellar aberration corrections to produce the most accurate surface intercept solution possible. We model the surface of the satellite as an ellipsoid.

```
cspice_sincpt, 'Ellipsoid', satnm, et, fixref, 'CN+S', scnm, iframe, $
               insite, point, trgepc, srfvec, found
```

The range we want is obtained from the outputs of `cspice_sincpt`. These outputs are defined only if a surface intercept is found. If `found` is true, the spacecraft-to-surface intercept range is the norm of the output argument `srfvec`. Units are km. We use the lcy function `cspice_vnorm` to obtain the norm:

```
cspice_vnorm( srfvec )
```

We'll write out the range data along with the other program results.



# Compute Lat/Lon and Illumination Angles

Navigation and Ancillary Information Facility

Compute the planetocentric latitude (`pclat`) and longitude (`pclon`), as well as the planetodetic latitude (`pdlat`) and longitude (`pdlon`) of the intersection point.

```
if ( found ) then begin
    cspice_reclat, point, r, pclon, pclat

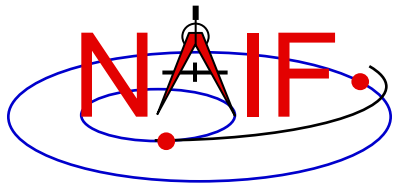
    ;; Let re, rp, and f be the satellite's longer equatorial
    ;; radius, polar radius, and flattening factor.

    re = radii[0]
    rp = radii[2]
    f = ( re - rp ) / re;

    cspice_recgeo, point, re, f, pdlon, pdlat, alt
```

The illumination angles we want are the outputs of `cspice_ilumin`. Units are radians.

```
cspice_ilumin, 'Ellipsoid', satnm, et, fixref, 'CN+S', scnm, $
    point, trgepc, srfvec, phase, solar, emissn
```



# Geometry Calculations: Summary

---

## Navigation and Ancillary Information Facility

```
;; Compute the boresight ray intersection with the surface of the
;; target body.

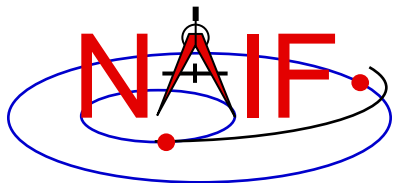
cspice_sincpt, 'Ellipsoid', satnm, et, fixref, 'CN+S', scnm, $
             iframe, insite, point, trgepc, srfvec, found

;; If an intercept is found, compute planetocentric and planetodetic
;; latitude and longitude of the point.

if ( found ) then begin
    cspice_reclat, point, r, pclon, pclat
    ;; Let re, rp, and f be the satellite's longer equatorial
    ;; radius, polar radius, and flattening factor.
    re = radii[0]
    rp = radii[2]
    f = ( re - rp ) / re;
    cspice_recgeo, point, re, f, pdlon, pdlat, alt

    ;; Compute illumination angles at the surface point.

    cspice_ilumin, 'Ellipsoid', satnm, et, fixref, 'CN+S', scnm, $
                 point, trgepc, srfvec, phase, solar, emissn
    ...
endif else begin
    ...
```



# Get inputs - 1

## Navigation and Ancillary Information Facility

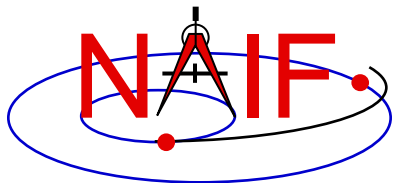
The code above used quite a few inputs that we don't have yet:

- TDB epoch of interest (`et`);
- satellite and s/c names (`satnm`, `scnm`);
- satellite body-fixed frame name (`fixref`);
- satellite ellipsoid radii (`radii`);
- instrument fixed frame name (`iframe`);
- instrument boresight vector in the instrument frame (`insite`);

Some of these values are user inputs; others can be obtained via lcy calls once the required kernels have been loaded.

Let's prompt for the satellite name (`satnm`), satellite frame name (`fixref`), spacecraft name (`scnm`), instrument name (`instnm`) and time of interest (`time`):

```
read, satnm , PROMPT='Enter satellite name > '  
read, fixref, PROMPT='Enter satellite frame > '  
read, scnm , PROMPT='Enter spacecraft name > '  
read, instnm, PROMPT='Enter instrument name > '  
read, time , PROMPT='Enter time > '
```



# Get Inputs - 2

## Navigation and Ancillary Information Facility

Then we can get the rest of the inputs from lcy calls:

To get the TDB epoch (`et`) from the user-supplied time string (which may refer to the UTC, TDB or TT time systems):

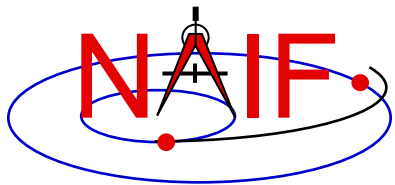
```
cspice_str2et, time, et
```

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

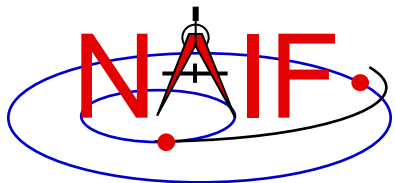
```
;; Prompt for the user-supplied inputs for our program
read, setupf, PROMPT='Enter setup file name > '
cspice_furnsh, setupf

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
if ( NOT found ) then begin
    print, "Unable to determine ID for instrument: ", instnm
return endif
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.
```

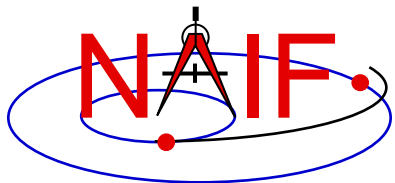
```
print  
print, 'Intercept planetocentric longitude      (deg): ', $  
                                     cspice_dpr()*pclon  
print, 'Intercept planetocentric latitude     (deg): ', $  
                                     cspice_dpr()*pclat  
print, 'Intercept planetodetic longitude      (deg): ', $  
                                     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

Navigation and Ancillary Information Facility

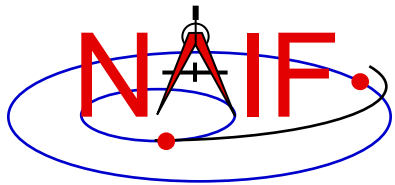
To finish up the program we need to declare the variables we've used.

- We'll highlight techniques used by NAIF programmers
- Add remaining IDL code required to make a syntactically valid program

```
PRO PROG_GEOMETRY

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





# Complete source code - 1

## Navigation and Ancillary Information Facility

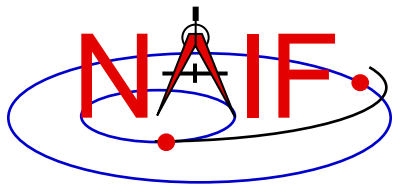
```
;; Prompt for the user-supplied inputs for our program.
read, setupf, PROMPT='Enter setup file name > '
cspice_furnsh, setupf
read, satnm, PROMPT='Enter satellite name > '
read, fixref, PROMPT='Enter satellite frame > '
read, scnm, PROMPT='Enter spacecraft name > '
read, instnm, PROMPT='Enter instrument name > '
read, time, PROMPT='Enter time > '

;; Get the epoch corresponding to the input time:
cspice_str2et, time, et

;; Get the radii of the satellite.
cspice_bodvrd, satnm, 'RADII', 3, radii

;; Get the instrument boresight and frame name.

cspice_bodn2c, instnm, instid, found
if ( NOT found ) then begin
    print, "Unable to determine ID for instrument: ", instnm
    return
endif
cspice_getfov, instid, ROOM, shape, iframe, insite, bundry
```



# Complete source code - 2

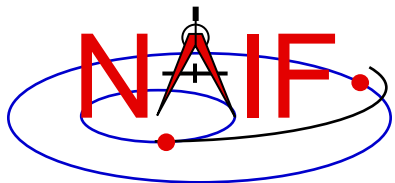
## Navigation and Ancillary Information Facility

```
;; Compute the boresight ray intersection with the surface of the
;; target body.
cspice_sincpt, 'Ellipsoid', satnm, et, fixref, 'CN+S', scnm, $
    iframe, insite, point, trgepc, srfvec, found

;; If an intercept is found, compute planetocentric and planetodetic
;; latitude and longitude of the point.
if ( found ) then begin
    cspice_reclat, point, r, pclon, pclat
    ;; Let re, rp, and f be the satellite's longer equatorial
    ;; radius, polar radius, and flattening factor.
    re = radii[0]
    rp = radii[2]
    f = ( re - rp ) / re
    cspice_recgeo, point, re, f, pdlon, pdlat, alt

    ;; Compute illumination angles at the surface point.
    cspice_ilumin, 'Ellipsoid', satnm, et, fixref, 'CN+S', scnm, $
        point, trgepc, srfvec, phase, solar, emissn

    ;; Display results. Convert angles from radians to degrees
    ;; for output.
    print
    print, 'Intercept planetocentric longitude      (deg): ', $
        R2D*pclon
```



# Complete source code - 3

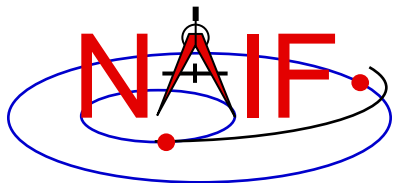
## Navigation and Ancillary Information Facility

```
print, 'Intercept planetocentric latitude      (deg): ', $
      R2D*pclat
print, 'Intercept planetodetic longitude      (deg): ', $
      R2D*pdlon
print, 'Intercept planetodetic latitude      (deg): ', $
      R2D*pdlat
print, 'Range from spacecraft to intercept point (km): ', $
      cspice_vnorm(srfvec)
print, 'Intercept phase angle                (deg): ', $
      R2D*phase
print, 'Intercept solar incidence angle      (deg): ', $
      R2D*solar
print, 'Intercept emission angle            (deg): ', $
      R2D*emissn

endif else begin
  print, 'No intercept point found at ' + time
endelse

;; Unload the kernels and clear the kernel pool
cspice_kclear

END
```



# Compile the program - 1

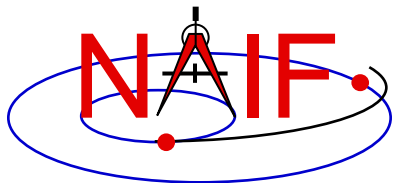
---

Navigation and Ancillary Information Facility

Though IDL functions in a manner similar to interpreted languages, it does compile source files to a binary form.

Ensure that both the Icy Toolkit, and an IDL installation are properly installed. IDL must load the Icy DLM, `icy.dlm/icy.so(dll)` to compile those scripts containing Icy calls. IDL loads DLMs from default locations and from the current directory when the user ran IDL. The user may also explicitly load a DLM with the `dml_register` command.

Now compile the code.



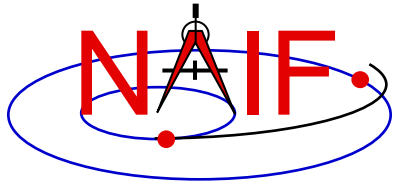
# Compile the program - 2

Navigation and Ancillary Information Facility

A screenshot of a terminal window titled "Terminal Window". The window has a standard Mac OS-style title bar with a close button in the top right corner. The terminal content shows the execution of an IDL command to compile a program. The text in the terminal is as follows:

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

On the left side of the terminal window, there is a vertical scrollbar and two arrow buttons (down and up) for navigating through the terminal history.



# Running the program

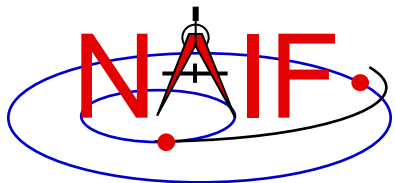
---

Navigation and Ancillary Information Facility

It looks like we have everything taken care of:

- We have all necessary kernels
- We made a setup file (metakernel) pointing to them
- We wrote the program
- We compiled the program

Let's run it.



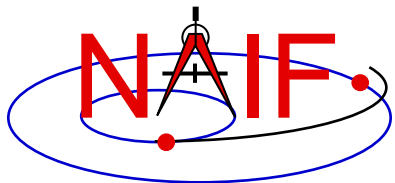
# Running the program

Navigation and Ancillary Information Facility

```
Terminal Window

IDL>  prog_geometry
Enter setup file name > setup.ker
Enter satellite name > PHOEBE
Enter satellite frame > IAU_PHOEBE
Enter spacecraft name > CASSINI
Enter instrument name > CASSINI_ISS_NAC
Enter time             > 2004 jun 11 19:32:00

Intercept planetocentric longitude      (deg):      39.843719
Intercept planetocentric latitude       (deg):      4.1958778
Intercept planetodetic longitude        (deg):      39.843719
Intercept planetodetic latitude         (deg):      5.0480106
Range from spacecraft to intercept point (km):    2089.1697
Intercept phase angle                   (deg):      28.139479
Intercept solar incidence angle          (deg):      18.247220
Intercept emission angle                 (deg):      17.858309
```



# Backup

Navigation and Ancillary Information Facility

- **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).

