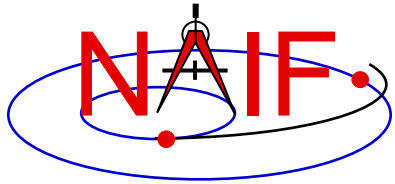


Navigation and Ancillary Information Facility

Writing a CSPICE (C) Based Program

January 2020

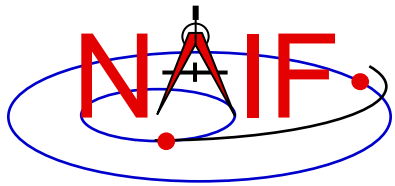


Viewing This Tutorial

Navigation and Ancillary Information Facility

Undefined variables are displayed in **red**

Results are displayed in **blue**



Introduction

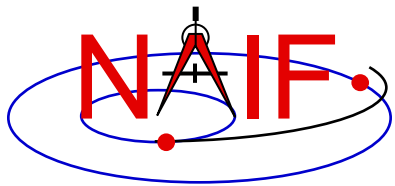
Navigation and Ancillary Information Facility

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

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

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

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



Observation geometry

Navigation and Ancillary Information Facility

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

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

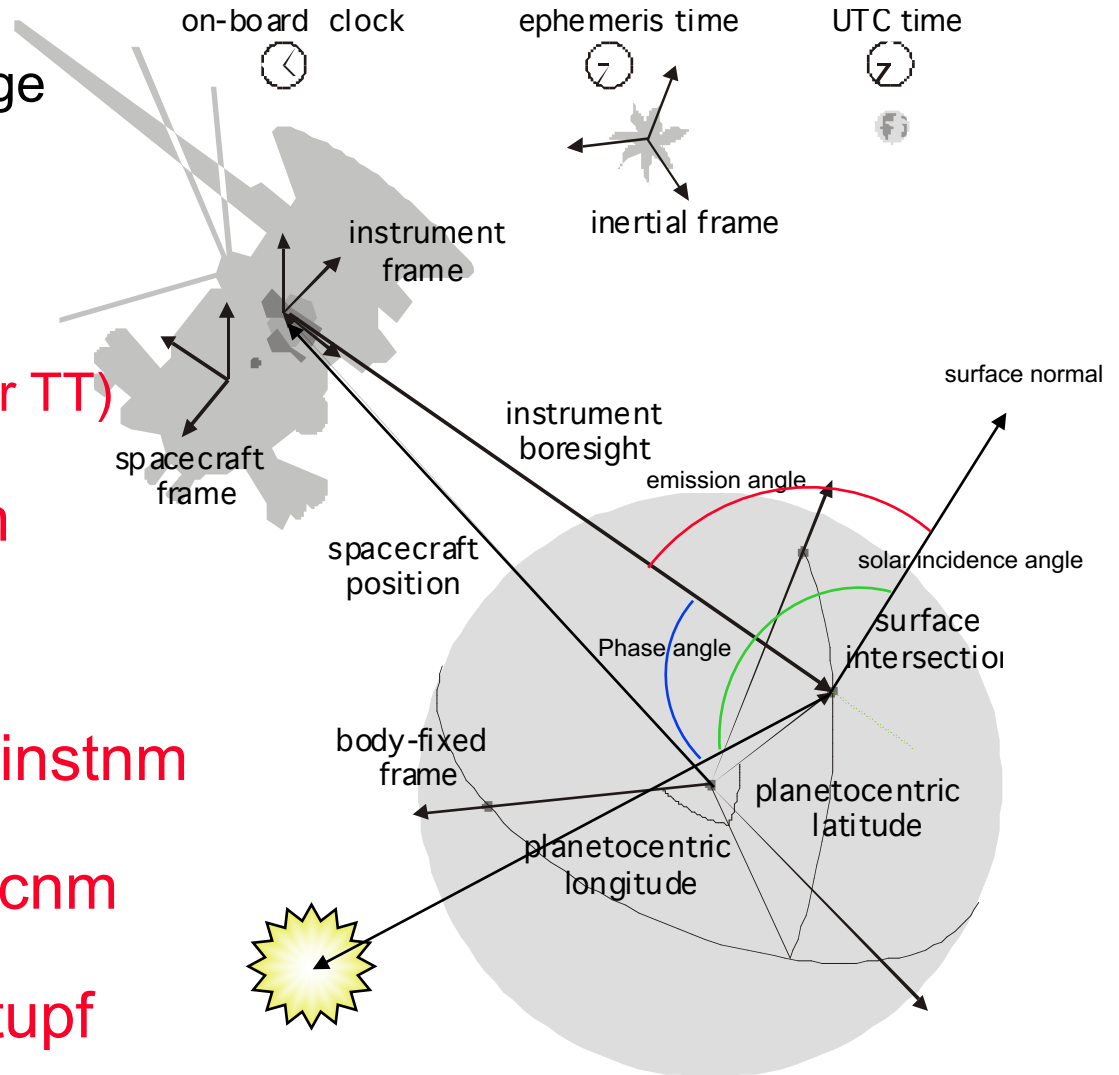
On what object? **satnm**

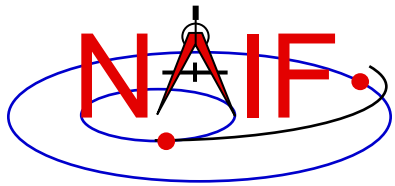
In what frame? **fixref**

For which instrument? **instnm**

For what spacecraft? **scnm**

Using what model? **setupf**





Needed Data

Navigation and Ancillary Information Facility

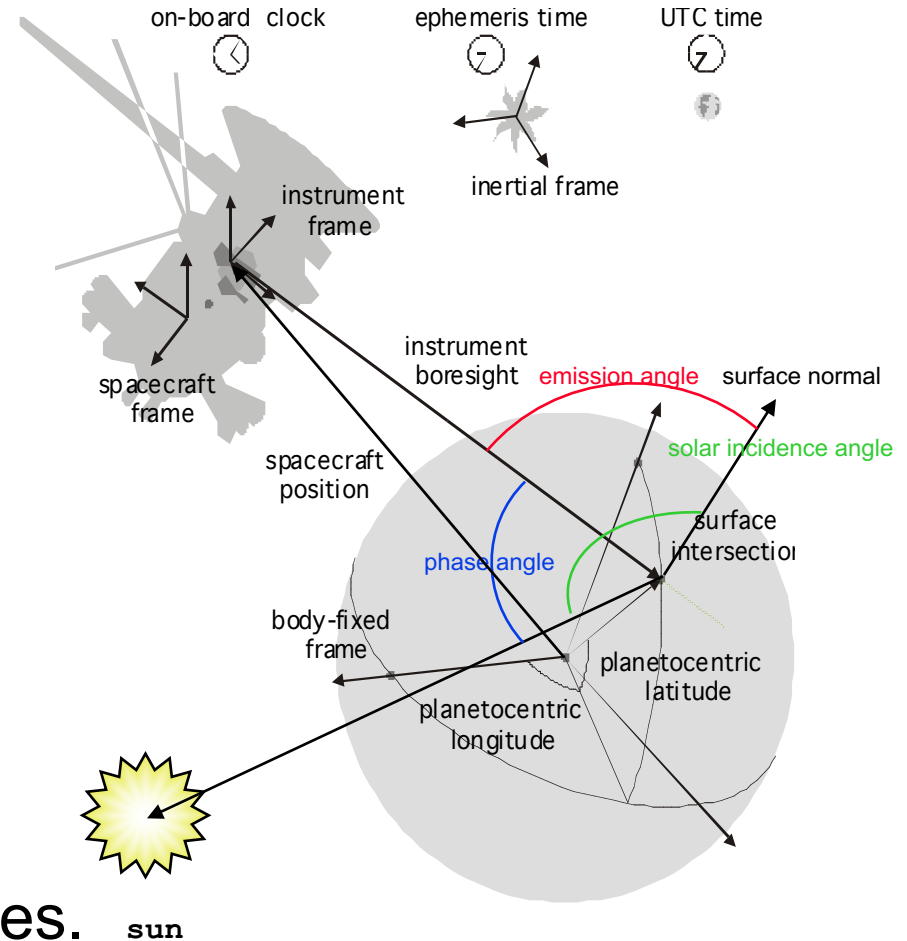
Time transformation kernels

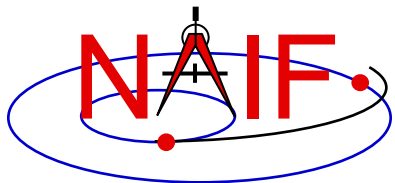
Orientation models

Instrument descriptions

Shapes of satellites, planets

Ephemerides for spacecraft,
Saturn barycenter and satellites.





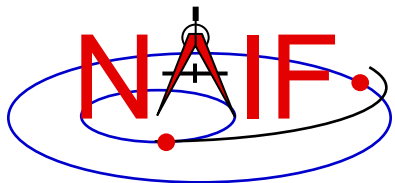
Which Kernels are Needed?

Navigation and Ancillary Information Facility

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

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

Parameter	Kernel Type	File name
time conversions	generic LSK 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 required kernels available to the program is via `furnsh_c`. For this example we make a setup file (also called a “metakernel” or “furnsh kernel”) containing a list of kernels to be loaded:

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

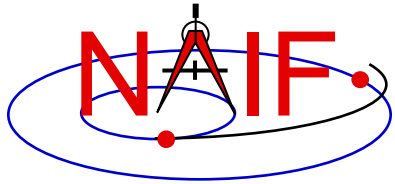
```
\begindata
```

```
KERNELS_TO_LOAD = ('naif0009.tls', 'cas00084.tsc', 'cpck05Mar2004.tpc',  
                   '020514_SE_SAT105.bsp', '981005_PLTEPH-DE405S.bsp',  
                   '030201AP_SK_SM546_T45.bsp', '04135_04171pc_psiv2.bc',  
                   'cas_v37.tf', 'cas_iss_v09.ti')
```

```
\begintext
```

and we make the program prompt for the name of this setup file:

```
prompt_c ( "Enter setup file name > ", FILESZ, setupf );  
furnsh_c ( setupf );
```



Programming Solution

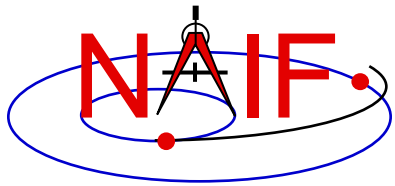
Navigation and Ancillary Information Facility

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

If there is an intersection,

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

We discuss the geometric portion of the problem next.



Compute Surface Intercept and Ranges

Navigation and Ancillary Information Facility

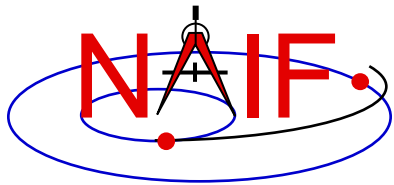
Compute the intercept point (`point`) of the boresight vector (`insite`) specified in the instrument frame (`iframe`) of the instrument mounted on the spacecraft (`scnm`) with the surface of the satellite (`satnm`) at the TDB time of interest (`et`) in the satellite's body-fixed frame (`fixref`). This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

```
sincpt_c ( "Ellipsoid", satnm, et, fixref, "CN+S", scnm, iframe, insite,  
          point, &trgepc, srfvec, &found );
```

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

```
vnorm_c ( srfvec )
```

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



Compute Lat/Lon and Illumination Angles

Navigation and Ancillary Information Facility

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

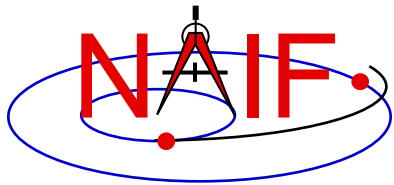
```
if ( found )
{
    reclat_c ( point, &r, &pclon, &pclat );

    /* Let re, rp, and f be the satellite's longer equatorial
       radius, polar radius, and flattening factor. */
    re = radii[0];
    rp = radii[2];
    f = ( re - rp ) / re;

    recgeo_c ( point, re, f, &pdlon, &pdlat, &alt);
}
```

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

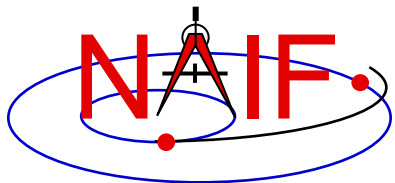
```
ilumin_c ( "Ellipsoid", satnm, et, fixref, "CN+S", scnm, point,
          &trgepc, srfvec, &phase, &solar, &emissn );
```



Geometry Calculations: Summary

Navigation and Ancillary Information Facility

```
/* Compute the boresight ray intersection with the surface of the
   target body. */
sincpt_c ( "Ellipsoid", satnm, et, fixref, "CN+S", scnm,
           iframe, insite, point, &trgepc, srfvec, &found );
/* If an intercept is found, compute planetocentric and planetodetic
   latitude and longitude of the point. */
if ( found )
{
  reclat_c ( point, &r, &pclon, &pclat );
  /* Let re, rp, and f be the satellite's longer equatorial
     radius, polar radius, and flattening factor. */
  re = radii[0];
  rp = radii[2];
  f = ( re - rp ) / re;
  recgeo_c ( point, re, f, &pdlon, &pdlat, &alt );
  /* Compute illumination angles at the surface point. */
  ilumin_c ( "Ellipsoid", satnm, et, fixref, "CN+S", scnm, point,
            &trgepc, srfvec, &phase, &solar, &emissn );
  ...
}
else
{ ... }
```



Get Inputs - 1

Navigation and Ancillary Information Facility

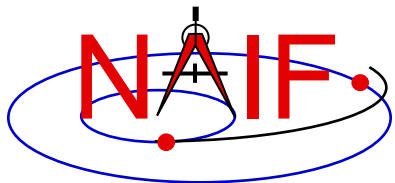
The code above used quite a few inputs that we don't have yet:

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

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

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

```
prompt_c ( "Enter satellite name > ", WORDSZ, satnm );
prompt_c ( "Enter satellite frame > ", WORDSZ, fixref );
prompt_c ( "Enter spacecraft name > ", WORDSZ, scnm );
prompt_c ( "Enter instrument name > ", WORDSZ, instnm );
prompt_c ( "Enter time                > ", WORDSZ, time );
```



Get Inputs - 2

Navigation and Ancillary Information Facility

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

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

```
str2et_c ( time, &et );
```

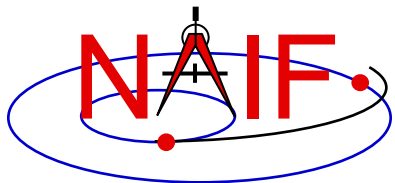
To get the satellite's ellipsoid radii (`radii`):

```
bodvrd_c ( satnm, "RADII", 3, &i, radii );
```

To get the instrument boresight direction (`insite`) and the name of the instrument frame (`iframe`) in which it is defined:

```
bodn2c_c ( instnm, &instid, &found );

if ( !found )
{
    setmsg_c ( "Instrument name # could not be "
              "translated to an ID code."      );
    errch_c ( "#", instnm                       );
    sigerr_c ( "NAMENOTFOUND"                   );
}
getfov_c ( instid, ROOM, WORDSZ, WORDSZ,
           shape, iframe, insite, &n, bundry );
```



Getting Inputs: Summary

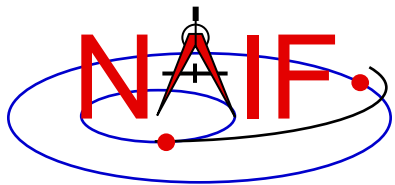
Navigation and Ancillary Information Facility

```
/* 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 );

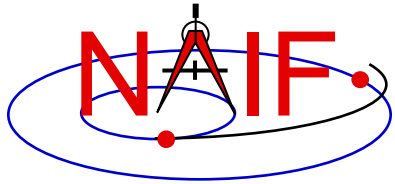
/* Get the instrument boresight and frame name. */
bodn2c_c ( instnm, &instid, &found );
if ( !found )
{
    setmsg_c ( "Instrument name # could not be "
              "translated to an ID code." );
    errch_c ( "#", instnm );
    sigerr_c ( "NAMENOTFOUND" );
}
getfov_c ( instid, ROOM, WORDSZ, WORDSZ,
           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 );
}
```

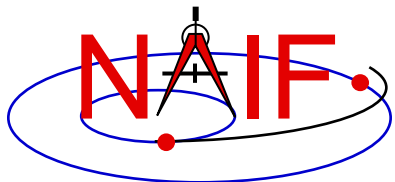


Complete the Program

Navigation and Ancillary Information Facility

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

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



Complete Source Code - 1

Navigation and Ancillary Information Facility

```
#include <stdio.h>
#include "SpiceUsr.h"

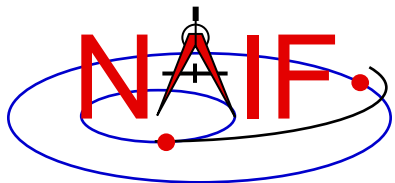
int main ()
{
    #define    FILESZ        256
    #define    WORDSZ        41
    #define    ROOM          10

    SpiceBoolean    found;

    SpiceChar        iframe [WORDSZ];
    SpiceChar        instnm [WORDSZ];
    SpiceChar        satnm  [WORDSZ];
    SpiceChar        fixref [WORDSZ];
    SpiceChar        scnm   [WORDSZ];
    SpiceChar        setupf [FILESZ];
    SpiceChar        shape  [WORDSZ];
    SpiceChar        time   [WORDSZ];

    SpiceDouble      alt;
    SpiceDouble      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

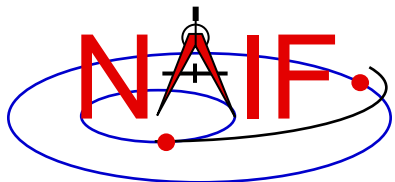
Navigation and Ancillary Information Facility

```
/* 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 );

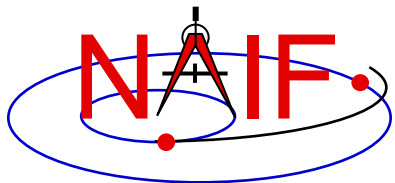
/* Get the instrument boresight and frame name. */
bodn2c_c ( instnm, &instid, &found );
if ( !found )
{
    setmsg_c ( "Instrument name # could not be "
              "translated to an ID code." );
    errch_c ( "#", instnm );
    sigerr_c ( "NAMENOTFOUND" );
}
getfov_c ( instid, ROOM, WORDSZ, WORDSZ,
          shape, iframe, insite, &n, bundry );
```



Complete Source Code - 3

Navigation and Ancillary Information Facility

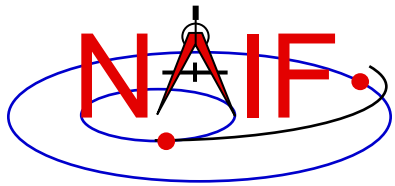
```
/* Compute the boresight ray intersection with the surface of the
   target body. */
sincpt_c ( "Ellipsoid", satnm, et, fixref, "CN+S", scnm,
           iframe, insite, point, &trgepc, srfvec, &found );
/* If an intercept is found, compute planetocentric and planetodetic
   latitude and longitude of the point. */
if ( found )
{
    reclat_c ( point, &r, &pclon, &pclat );
    /* Let re, rp, and f be the satellite's longer equatorial
       radius, polar radius, and flattening factor. */
    re = radii[0];
    rp = radii[2];
    f = ( re - rp ) / re;
    recgeo_c ( point, re, f, &pdlon, &pdlat, &alt );
    /* Compute illumination angles at the surface point. */
    ilumin_c ( "Ellipsoid", satnm, et, fixref, "CN+S", scnm, point,
              &trgepc, srfvec, &phase, &solar, &emissn );
    /* Display results. Convert angles to degrees for output. */
    printf ( "\n"
             "Intercept planetocentric longitude      (deg):   %11.6f\n"
             "Intercept planetocentric latitude         (deg):   %11.6f\n"
```



Complete Source Code - 4

Navigation and Ancillary Information Facility

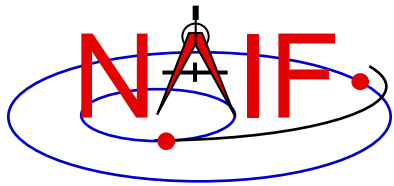
```
"Intercept planetodetic longitude           (deg):   %11.6f\n"  
"Intercept planetodetic latitude           (deg):   %11.6f\n"  
"Range from spacecraft to intercept point  (km):   %11.6f\n"  
"Intercept phase angle                     (deg):   %11.6f\n"  
"Intercept solar incidence angle           (deg):   %11.6f\n"  
"Intercept emission angle                  (deg):   %11.6f\n",  
dpr_c() * pclon,  
dpr_c() * pclat,  
dpr_c() * pdlon,  
dpr_c() * pdlat,  
vnorm_c( srfvec ),  
dpr_c() * phase,  
dpr_c() * solar,  
dpr_c() * emissn  
    );  
}  
else {  
    printf ( "No intercept point found at %s\n", time );  
}  
return(0);  
}
```



Compile and Link the Program - 1

Navigation and Ancillary Information Facility

- **First be sure that both the CSPICE Toolkit and a C compiler are properly installed.**
 - A "hello world" C program must be able to compile, link, and run successfully in your environment.
 - Any of the mkprodct.* scripts in the cspice/src/* paths of the CSPICE installation should execute properly.
- **Ways to compile and link the program:**
 - If you're familiar with the "make" utility, create a makefile. Use compiler and linker options from the mkprodct.* script found in the cspice/src/cook_c path of your CSPICE installation.
 - Or, modify the cookbook mkprodct.* build script.
 - » Your program name must be *.pgm, for example demo.pgm, to be recognized by the script.
 - » Change the library references in the script to use absolute pathnames.
 - » Change the path for the executable to the current working directory.
 - » If your compiler supports it, add a -I option to reference the cspice/include path to make CSPICE *.h files available. Otherwise, copy those files from the include path to your current working directory.
 - » On some platforms, you must modify the script to refer to your program by name.



Compile and Link the Program - 2

Navigation and Ancillary Information Facility

- Or, compile the program on the command line. The program must be linked against the CSPICE object library `cspice.a` (`cspice.lib` under MS Visual C++/C) and the C math library. On a PC running Linux and `gcc`, if

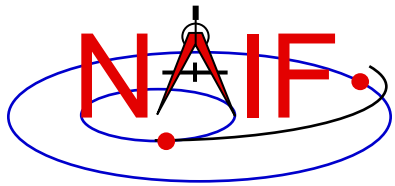
- » The `gcc` compiler is in your path
 - As indicated by the response to the command "which gcc"
- » the Toolkit is installed in the path (for the purpose of this example) `/myhome/cspice`
- » You've named the program `demo.c`

then you can compile and link your program using the command

```
» gcc -I/myhome/cspice/include \  
    -o demo \  
    demo.c /myhome/cspice/lib/cspice.a -lm
```

- Note: the preprocessor flag
`-DNON_UNIX_STUDIO`

used in the `mkprodct.csh` script is needed for code generated by `f2c`, but is usually unnecessary for compiling user code.



Compile and Link the Program - 3

Navigation and Ancillary Information Facility

```
Terminal Window

Prompt> mkprodct.csh

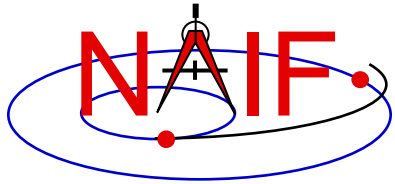
        Setting default compiler:
gcc

        Setting default compile options:
-c -ansi -O2 -DNON_UNIX_STDIO

        Setting default link options:
-lm

        Compiling and linking:  demo.pgm
Compiling and linking:  demo.pgm

Prompt>
```



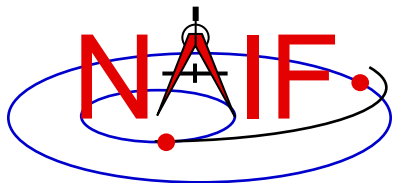
Running the Program - 1

Navigation and Ancillary Information Facility

It looks like we have everything taken care of:

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

Let's run it.



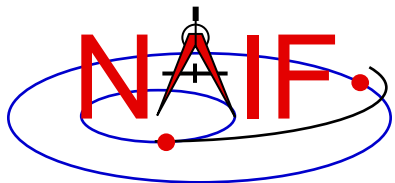
Running the Program - 2

Navigation and Ancillary Information Facility

```
Terminal Window

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

Intercept planetocentric longitude (deg) : 39.843719
Intercept planetocentric latitude (deg) : 4.195878
Intercept planetodetic longitude (deg) : 39.843719
Intercept planetodetic latitude (deg) : 5.048011
Range from spacecraft to intercept point (km) : 2089.169724
Intercept phase angle (deg) : 28.139479
Intercept solar incidence angle (deg) : 18.247220
Intercept emission angle (deg) : 17.858309
Prompt>
```



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).

