Writing a SPICE-based Fortran Program

March 2006

Viewing This Tutorial

This coding example is an "animated" presentation that is best viewed using PowerPoint set to "Slide Show" mode.

Undefined variables are displayed in red; results are displayed in blue.
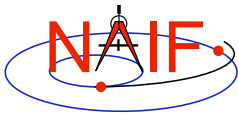
Introduction

First, let's go over the important steps in the process of writing a 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

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

When? TIME (UTC, TDB or TT)

On what object? SATNM

For which instrument? INSTNM

For what spacecraft? SCNM

Using what model? SETUPF

on-board clock

ephemeris time

UTC time

inertial frame

instrument frame

instrument boresight

surface normal

emission angle

solar incidence angle

spacecraft frame

spacecraft position

Phase angle

Surface intersection

body-fixed frame

planetocentric longitude

planetocentric latitude

Time transformation kernels

Orientation models

Instrument descriptions

Shapes of satellites, planets

Ephemerides for spacecraft,
Saturn barycenter and satellites.
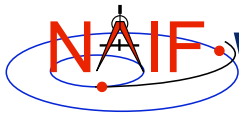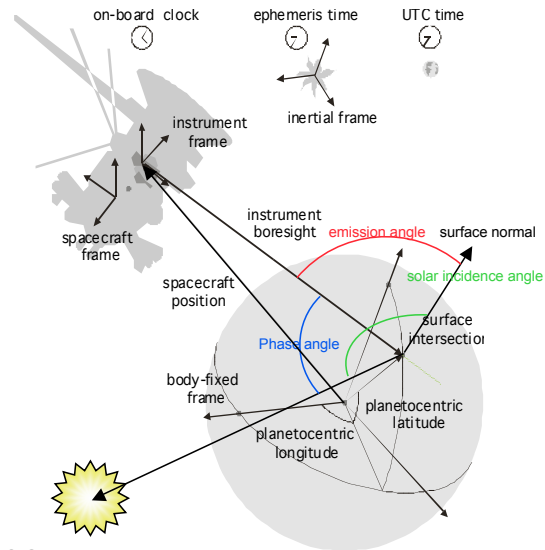
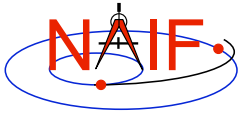
Which Kinds of Kernels are Needed?

NAIF

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	naif0007.tls
	CASSINI SCLK	cassini.tsc
satellite orientation	generic PCK	pck00007.tpc
satellite shape	generic PCK	pck00007.tpc
satellite position	planet/sat ephemeris SPK	020514_SE_SAT105.bsp
planet barycenter position	planet SPK	981005_PLTEPH-DE405S.bsp
spacecraft position	spacecraft SPK	tour9201.bsp
spacecraft orientation	spacecraft CK	cas_050215.bc
instrument alignment	CASSINI FK	cas_v37.tf
instrument boresight	Instrument IK	cas_iss_v09.ti

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 = ( 'naif0007.tls',           'cassini.tsc',
                       'pck00007.tpc',           '020514_SE_SAT105.bsp',
                       '981005_PLTEPH-DE405S.bsp', 'tour9201.bsp',
                       'cas_050215.bc',          'cas_v37.tf',
                       'cas_iss_v09.ti'                                  )
\begintext
```
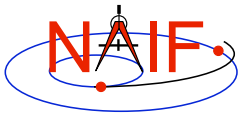
and we make the program prompt for the name of this setup file:

```
   CALL PROMPT ( 'Enter setup file name > ', SETUPF )
   CALL FURNSH ( SETUPF )
```

- 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 and spacecraft-to-target center 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.
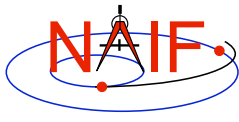
Compute Surface Intercept and Ranges

Compute the intercept point (**POINT**) of the instrument boresight vector (**INSITE**) with the satellite's (**SATNM**) surface at the TDB time of interest (**ET**). This call also returns the distance between the spacecraft and intercept point (**DIST**), the light-time corrected epoch at the intercept point (**TRGEPC**), the target center-to-spacecraft vector (**OBSPOS**), 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.

```
CALL SRFXPT ( 'Ellipsoid', SATNM, ET,    'CN+S',  SCNM,    IFRAME,
              INSITE,    POINT, DIST, TRGEPC, OBSPOS, FOUND  )
```

The ranges we want are obtained from the outputs of SRFXPT. These outputs are defined only if a surface intercept is found. If **FOUND** is true, the spacecraft-to-surface intercept range is the output argument **DIST**, and the spacecraft-to-target center range is the norm of the output argument **OBSPOS**. Units are km. We use the SPICELIB function VNORM to obtain the norm:

```
VNORM ( OBSPOS )
```

We'll write out the range data along with the other program results.
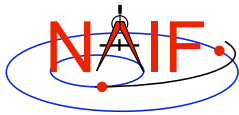
Compute Lat/Lon and Illumination Angles

Compute the planetocentric latitude (**PCLAT**) and longitude (**PCLON**), as well as the planetodetic latitude (**PDLAT**) and longitude (**PDLON**) of the intersection point.

```
        IF ( FOUND ) 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. For this call, normal light time and stellar aberration corrections suffice.

```
        CALL ILLUM ( SATNM,  ET,    'LT+S',  SCNM,
                     POINT, PHASE, SOLAR,  EMISSN )
```

```
C     Compute the boresight ray intersection with the surface of the
C     satellite. DIST and VNORM(OBSPOS) yield desired ranges.

      CALL SRFXPT ( 'Ellipsoid', SATNM,  ET,    'CN+S',  SCNM,   IFRAME,
                    INSITE,      POINT,  DIST,  TRGEPC,  OBSPOS, 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 ILLUM ( SATNM, ET,    'LT+S' SCNM,
                      POINT, PHASE, SOLAR, EMISSN )
         ...
      ELSE
         ...
```
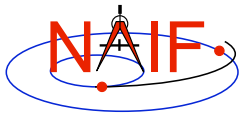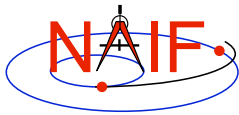
The code above used quite a few inputs that we don't have yet:

- **TDB epoch of interest (ET)**
- **satellite and s/c names (SATNM, SCNM)**
- **satellite 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), spacecraft name (SCNM), instrument name (INSTNM) and time of interest (TIME):

```
      CALL PROMPT ( 'Enter satellite name  > ', SATNM  )
      CALL PROMPT ( 'Enter spacecraft name > ', SCNM   )
      CALL PROMPT ( 'Enter instrument name > ', INSTNM )
      CALL PROMPT ( 'Enter time            > ', TIME   )
```
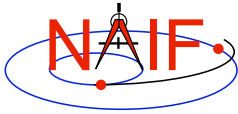
Then we can get the rest of the inputs from SPICELIB calls:

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             )
```
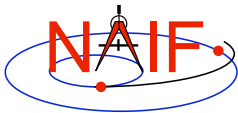
```
C      Prompt for the user-supplied inputs for our program.
       CALL PROMPT ( 'Enter setup file name > ', SETUPF )

       CALL FURNSH ( SETUPF )
       CALL PROMPT( 'Enter satellite name  > ', SATNM  )
       CALL PROMPT( 'Enter 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             )
```
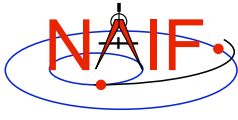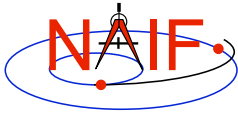
```
C       Display results.  Convert angles from radians to degrees
C       for output.
        WRITE ( *, '(1X,A,F12.6)' )
    .   'Intercept planetocentric longitude      (deg):  ',  DPR()*PCLON
        WRITE ( *, '(1X,A,F12.6)' )
    .   'Intercept planetocentric latitude       (deg):  ',  DPR()*PCLAT
        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):  ',  DIST
        WRITE ( *, '(1X,A,F12.6)' )
    .   'Range from spacecraft to target center   (km):  ',
    .   VNORM(OBSPOS)
        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
```
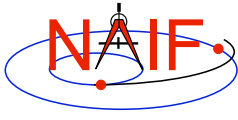
To finish up the program we need to declare the variables we've used.

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

```
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)   SCNM
CHARACTER*(FILESZ)   SETUPF
CHARACTER*(WORDSZ)   SHAPE
CHARACTER*(WORDSZ)   TIME

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

```
DOUBLE PRECISION     DIST
DOUBLE PRECISION     EMISSN
DOUBLE PRECISION     ET
DOUBLE PRECISION     F
DOUBLE PRECISION     INSITE(3)
DOUBLE PRECISION     OBSPOS(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
```

```
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 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           )
```

```
C       Compute the boresight ray intersection with the surface of the
C       satellite. DIST and VNORM(OBSPOS) yield desired ranges.
        CALL SRFXPT ( 'Ellipsoid', SATNM,   ET,      'CN+S',   SCNM,    IFRAME,
                       INSITE,      POINT,  DIST,  TRGEPC,  OBSPOS, 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 ILLUM ( SATNM, ET,     'LT+S', SCNM,
                        POINT, 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
```
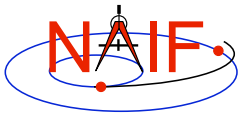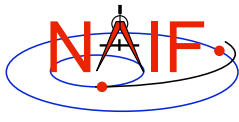
```
           WRITE ( *, '(1X,A,F12.6)' )
    .      'Intercept planetocentric latitude        (deg):  ',  DPR()*PCLAT
           WRITE ( *, '(1X,A,F12.6)' )
    .      'Intercept planetodetic longitude         (deg):  ',  DPR()*PDLON
           WRITE ( *, '(1X,A,F12.6)' )
    .      'Intercept planetodetic latitude          (deg):  ',  DPR()*PDLAT
           WRITE ( *, '(1X,A,F12.6)' )
    .      'Range from spacecraft to intercept point (km):  ',  DIST
           WRITE ( *, '(1X,A,F12.6)' )
    .      'Range from spacecraft to target center    (km):  ',
    .      VNORM(OBSPOS)
           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
```
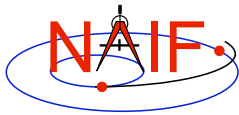
- **First be sure that both the SPICE Toolkit and a Fortran compiler are properly installed.**
  - A "hello world" Fortran program must be able to compile, link, and run successfully in your environment.
  - Any of the mkprodct.* scripts in the toolkit/src/* paths of the SPICE Toolkit installation should 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.

  - Or, compile the program on the command line. The program must be linked against the SPICE support object library support.a and the SPICELIB object library spicelib.a (support.lib and 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/support.a \
              /myhome/toolkit/lib/spicelib.a
```

```
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>
```
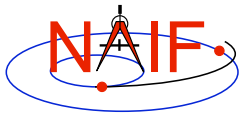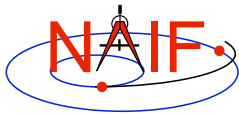
It looks like we have everything taken care of:

- **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.

```
                            Terminal Window

    Prompt> demo
    Enter setup file name   > setup.ker
    Enter satellite name    > titan
    Enter spacecraft name   > cassini
    Enter instrument name   > cassini_iss_nac
    Enter time              > 2005 feb 15 8:15 UTC

    Intercept planetocentric longitude      (deg):  -156.443003
    Intercept planetocentric latitude       (deg):    18.788539
    Intercept planetodetic longitude        (deg):  -156.443003
    Intercept planetodetic latitude         (deg):    18.788539
    Range from spacecraft to intercept point (km):  4810.941881
    Range from spacecraft to target center   (km):  7384.326555
    Intercept phase angle                   (deg):    43.274588
    Intercept solar incidence angle         (deg):    41.038424
    Intercept emission angle                (deg):     2.514613
    Prompt>
```