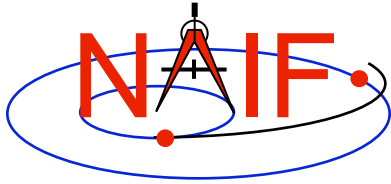


Navigation and Ancillary Information Facility

IDL Interface to CSPICE **“Icy”**

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

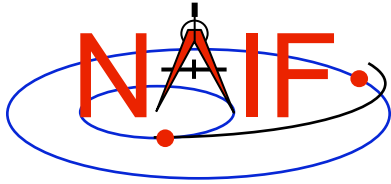
September 2009



Topics

Navigation and Ancillary Information Facility

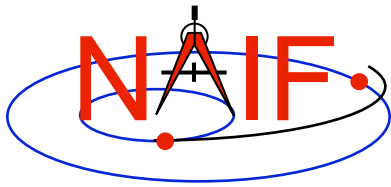
- **Icy Benefits**
- **How does it work?**
- **Distribution**
- **Icy Operation**
- **Vectorization**
- **Simple Use of Icy Functionality**



Icy Benefits

Navigation and Ancillary Information Facility

- **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.**

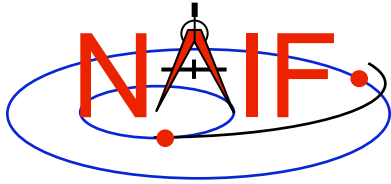


How Does It Work? (1)

Navigation and Ancillary Information Facility

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

Navigation and Ancillary Information Facility

- from the IDL interpreter (or from a command script), execute the **dml_register** command

```
IDL> dml_register, '_path_to_directory_containing_icy.dml_'  
» IDL> dml_register, '/naif/icy/lib/icy.dml'  
» IDL> dml_register, 'c:\naif\icy\lib\icy.dml'
```

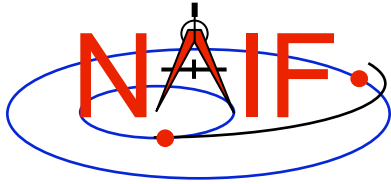
- copy **icy.dml** and **icy.so (icy.dll)** to IDL's binary directory

```
{The IDL install directory}/bin/bin.user_architecture  
» /usr/local/itt/idl64/bin/bin.linux.x86/  
» C:\ITT\IDL64\bin\bin.x86\
```

- append to the **IDL_DLM_PATH** environment variable the directory name containing **icy.dml** and **icy.so (icy.dll)** e.g.:

```
setenv IDL_DLM_PATH "<IDL_DEFAULT>: _path_to_directory_containing_icy.dml_"
```

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.dml**" but no "**icy.so**."



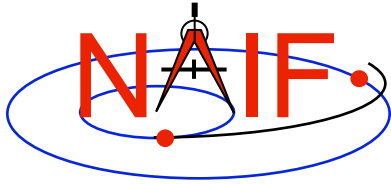
How Does It Work? (3)

Navigation and Ancillary Information Facility

When a user invokes a call to a DLM routine:

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

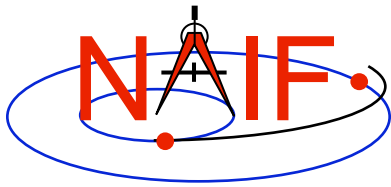
... transparent from the user's perspective.



Icy Distribution

Navigation and Ancillary Information Facility

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



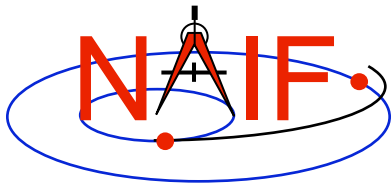
Icy Operation (1)

Navigation and Ancillary Information Facility

- **A user may occasionally encounter an IDL math exception:**

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

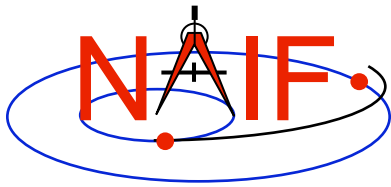
- This warning occurs most often as a consequence of CSPICE math operations.
- **In all known cases, the SIGFPE exceptions caused by CSPICE can be ignored. CSPICE assumes numeric underflow as zero.**
 - **A user can adjust IDL's response to math exceptions by setting the !EXCEPT variable:**
 - » **!EXCEPT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



Icy Operation (2)

Navigation and Ancillary Information Facility

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

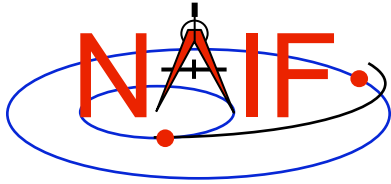


Icy Operation (3)

Navigation and Ancillary Information Facility

– Mitigation: two approaches

- » Load all needed SPICE kernels for your IDL session at the beginning of the session, paying careful attention to the files loaded and the loading order (loading order affects precedence)
 - Convince yourself that this approach will provide ALL of the scripts you will run during this IDL session with the appropriate SPICE data
- » At or near the end of every IDL script 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



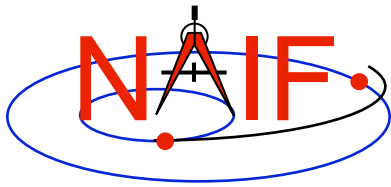
Icy Vectorization (1)

Navigation and Ancillary Information Facility

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

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

continued on next page



Icy Vectorization (2)

Navigation and Ancillary Information Facility

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

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

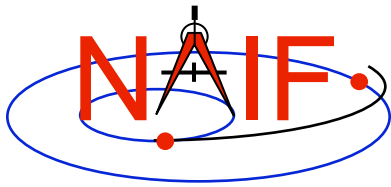
- Access the *ith* state 6-vector corresponding to the *ith* ephemeris time with the expression

```
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)

Navigation and Ancillary Information Facility

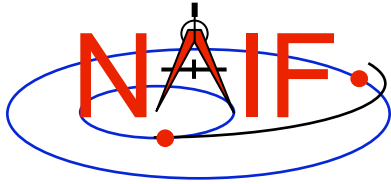
- The call returns `utcstr`, an array of 1000 strings each *ith* string the calendar date corresponding to `et[i]`. Access the *ith* string of `utcstr` corresponding to the *ith* ephemeris time with the expression

```
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

Navigation and Ancillary Information Facility

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

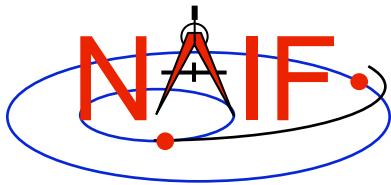
```
;; Define the number of divisions of the time interval and the time interval.
STEP = 10000
utc  = [ 'Jun 20, 2004', 'Dec 1, 2005' ]

;; Load the needed kernels
cspice_furnsh, 'standard.tm'
cspice_furnsh, '/kernels/cassini/spk/T18-5TDJ5.bsp'

cspice_str2et, utc, et
times = dindgen(STEP)*(et[1]-et[0])/STEP + et[0]
cspice_spkpos, 'Cassini', times, 'J2000', 'NONE', 'SATURN BARYCENTER', pos, ltime

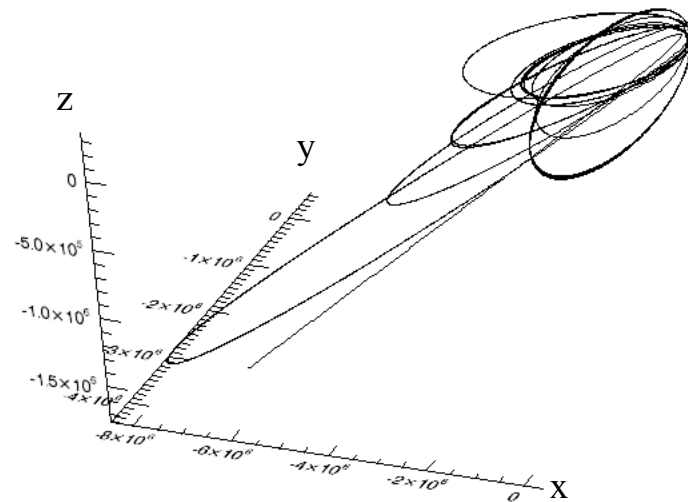
;; Plot the resulting trajectory.
x = pos[0,*]
y = pos[1,*]
z = pos[2,*]
iplot, x, y, z

cspice_kclear
```



Graphic Output using IDL iTool

Navigation and Ancillary Information Facility



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