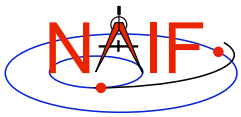
JNISPICE

March 2006

Navigation and Ancillary Information Facility

Topics

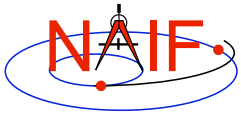
- User feedback
- Introduction
- Overview of the JNISPICE prototype system
- Possible Java SPICE Implementation Structures
- Possible OO SPICE Implementation Structures



User Feedback

Navigation and Ancillary Information Facility

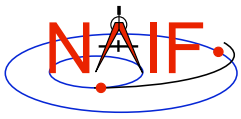
- **NAIF would like to hear your ideas on JNISPICE and more generally on the concept of an object-oriented SPICE Toolkit.**
  - **Would such a product be useful to you?**
  - **If so, what characteristics would this product have?**
- **To reply, please contact**
  - **Nat Bachman: Nathaniel.Bachman@jpl.nasa.gov**
  - **Chuck Acton: Charles.H.Acton-Jr@jpl.nasa.gov**



Introduction

Navigation and Ancillary Information Facility

- **JNISPICE is a prototype implementation of the SPICE system in Java, using the Java native interface (JNI) capability.**
  - **High level, object-oriented code is written in Java.**
  - **Low level implementation is largely based on CSPICE.**
- **Motivation for the JNISPICE experiment:**
  - **Various SPICE users are already using SPICE via JNI calls. Need for product of this type appears to exist.**
  - **Value of a complete, robust, well-documented, well-supported version of such a product is evident.**
  - **This development work may become the basis of a fully-OO version of SPICE, either in Java or other OO languages.**
  - **JNISPICE may facilitate development of higher-level SPICE-based tools, particularly GUI tools, by NAIF and others.**
  - **Arguably NAIF is the right team to do the job.**

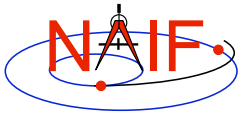


JNISPICE Implementation Layers

Navigation and Ancillary Information Facility

JNISPICE has three implementation layers:

- **API level: Object-oriented view of SPICE based on "natural" Java classes**
  - States, Times, Reference frames, Units, SPICE exceptions, etc.
  - Functionality of OO layer, when complete
    - » Will include functionality of CSPICE
    - » May include higher-level functionality not provided by CSPICE
      - Classes to support building SPICE-based GUIs?
- **JNI level: Java class or classes declaring native methods**
  - Methods correspond to CSPICE wrappers
  - Method functionality is as close as possible to that of CSPICE.
  - Methods provide error handling: trap SPICE errors, fetch SPICE error messages, throw exceptions
  - Methods are synchronized
- **CSPICE level: C shared object library**

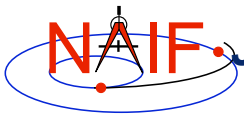


JNISPICE Documentation

Navigation and Ancillary Information Facility

JNISPICE documentation

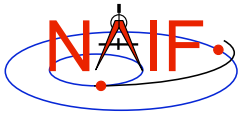
- **Java source code is documented via javadoc.**
  - All documentation is presented as HTML pages
  - System-scope and package-scope documentation resides in stand-alone files.
  - Java source code contains detailed documentation:
    - » Class-scope documentation is located at the start of each class source file.
    - » Method documentation in the style of SPICE module headers. Most "SPICE Toolkit header style" documentation is presented in the "Constructor Detail" or "Method Detail" portions of Java class documentation pages
  - For an example of javadoc-style documentation, see Sun's own Java documentation at
    - » http://java.sun.com/j2se/1.4.2/docs/api
- **CSPICE code has traditional CSPICE Toolkit documentation**



JNISPICE Implementation: Components

Navigation and Ancillary Information Facility

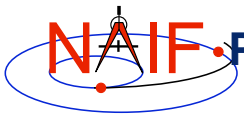
- Seven packages
  - spice.basic
  - spice.coverage
  - spice.daf
  - spice.geometry
  - spice.jni
  - spice.timesystem
  - spice.units
- About 60 classes and interfaces
- About 400 methods
  - As with most OO systems, there are many trivial methods.
    - » System is not as big as it looks.



Package spice.basic: Components

Navigation and Ancillary Information Facility

- Classes
  - AberrationCorrection
  - Body
  - Engineering Quaternion
  - GeodeticCoords
  - KernelDatabase
  - LatitudinalCoords
  - Matrix33
  - MatrixG
  - PositionRecord
  - PositionVector
  - Quaternion
  - ReferenceFrame
  - SCLK
  - SCLKTime
  - SpiceQuaternion
  - StateRecord
  - StateVector
  - Time
  - Vector3
  - VectorG
  - VelocityVector
- Interfaces
  - Coordinates
    - » Implemented by
      - GeodeticCoords
      - LatitudinalCoords
- Exceptions
  - SpiceException
    - » Superclass for all JNISPICE exceptions
    - » Non-error exceptions are also derived from SpiceException
      - PointingNotFoundException
  - SpiceErrorException
    - » All CSPICE errors cause a SpiceErrorException to be thrown



Package spice.basic: Selected Hierarchies

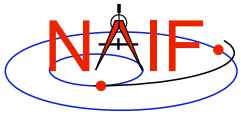
Navigation and Ancillary Information Facility

State vector class hierarchy

java.lang.Object
  spice.basic.VectorG
    spice.basic.StateVector
      spice.basic.StateRecord

Position vector class hierarchy

java.lang.Object
  spice.basic.Vector3
    spice.basic.PositionVector
      spice.basic.PositionRecord



Package spice.daf

Navigation and Ancillary Information Facility

- Classes
  - DAFArraySearch
    - » An array search is an object, rather than a set of variables comprising a "state" as in SPICELIB
  - DAFFileRecord
  - DAFNameRecord
  - DAFSegmentDescr
  - DAFSummaryRecord
  - ReadOnlyDAF
    - » A class implementing the DAFReader interface
- Interfaces
  - DAFReader
    - » The interface to be implemented by all classes that emulate traditional DAF access methods
- Exceptions
  - DAFRecordNotFoundException

spice.basic Example Program - 1

Navigation and Ancillary Information Facility

```
//
// Find the state of the moon relative to the earth at a specified
// UTC time.  Use light time and stellar aberration corrections.
// Express the state relative to the J2000 reference frame.  Display
// the results of the computation.
//

import java.awt.*;
import spice.basic.*;

public class SRExample extends Object
{
   //
   // Load the JNISpice C shared object library.
   //
   static
   {
      System.loadLibrary( "JNISpice" );
   }

   public static void main ( String[] args )
   {
      //
      // Get line terminator character for host system.
      //
      String nl = System.getProperty ( "line.separator" );
```


spice.basic Example Program - 2

Navigation and Ancillary Information Facility

```
      try
      {
         //
         // Load SPICE kernels:  a planetary ephemeris SPK file and a
         // leapseconds kernel.
         //
         KernelDatabase.load ( "de405.bsp"       );
         KernelDatabase.load ( "leapseconds.ker" );
         //
         // Declare and initialize inputs for state look-up.
         //
         Body                 targ   = new Body ( "moon"       );
         Body                 obs    = new Body ( "earth"      );
         Time                 t      = new Time ( "2003 nov 6" );
         ReferenceFrame       ref    = new ReferenceFrame       ( "J2000" );
         AberrationCorrection abcorr = new AberrationCorrection ( "LT+S"  );
         //
         // Create a new "state record."  This is a traditional SPICE state
         // vector with additional information grouped together in a
         // data structure.
         //
         StateRecord sr = new StateRecord ( targ, t, ref, abcorr, obs );
```

spice.basic Example Program - 3

Navigation and Ancillary Information Facility

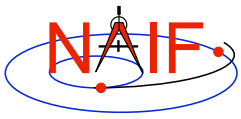
```
         // Display the state record using StateRecord class' default
         // formatting method toString(), which is implicitly invoked by
         // Java's System.out.println method.
         //
         System.out.println ( sr );

         //
         // Express the target position in latitudinal coordinates.
         //
         System.out.println ( new LatitudinalCoords(sr) + nl );
      }
      catch ( SpiceException se )
      {
         //
         // Display description of the exception and a traceback.
         //
         se.printStackTrace();
      }
   }
```

When this program is executed, the following output is produced:

spice.basic Program Output

Navigation and Ancillary Information Facility

```
Target               = MOON (NAIF ID 301)
Observer             = EARTH (NAIF ID 399)
Time                 = 2003-NOV-06 00:01:04.182  TDB
Reference frame      = J2000
Aberration correction = LT+S

State vector =

     3.9366537814844770E 005  (km)
     7.0833486231944120E 004  (km)
     4.3878137398047400E 003  (km)
    -1.2859001690543082E-001  (km/s)
     8.6712178855283640E-001  (km/s)
     4.4855824031090030E-001  (km/s)

Distance =

     4.0001133188191880E 005  (km)

Speed =

     9.8470304334942540E-001  (km/s)

One way light time =

     1.3342941798819998E 000  (seconds)

Radius    =  4.0001133188191880E 005 (km)
Longitude =  1.0200268551633691E 001 (degrees)
Latitude  =  6.2850282091979340E-001 (degrees)
```

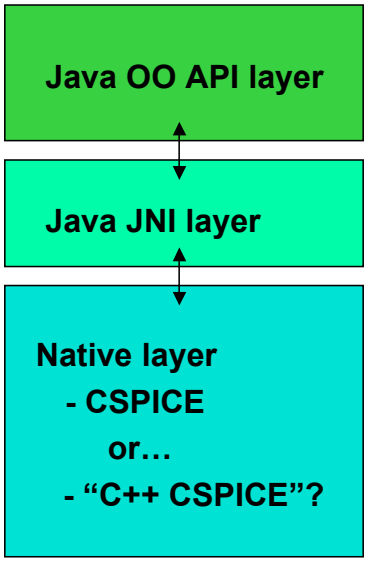


Alternative Java SPICE Implementations

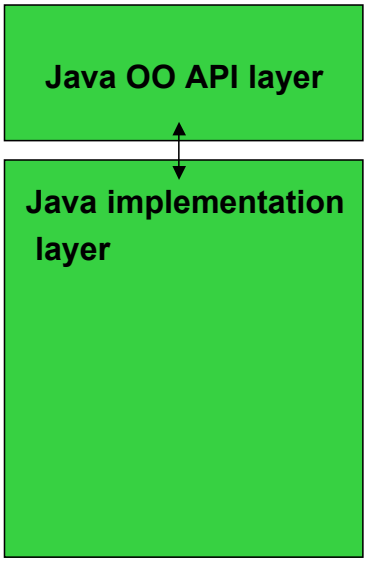
Navigation and Ancillary Information Facility

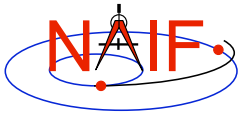
JNISPICE style

- Could be build on CSPICE
- Could be built on C++



Pure Java

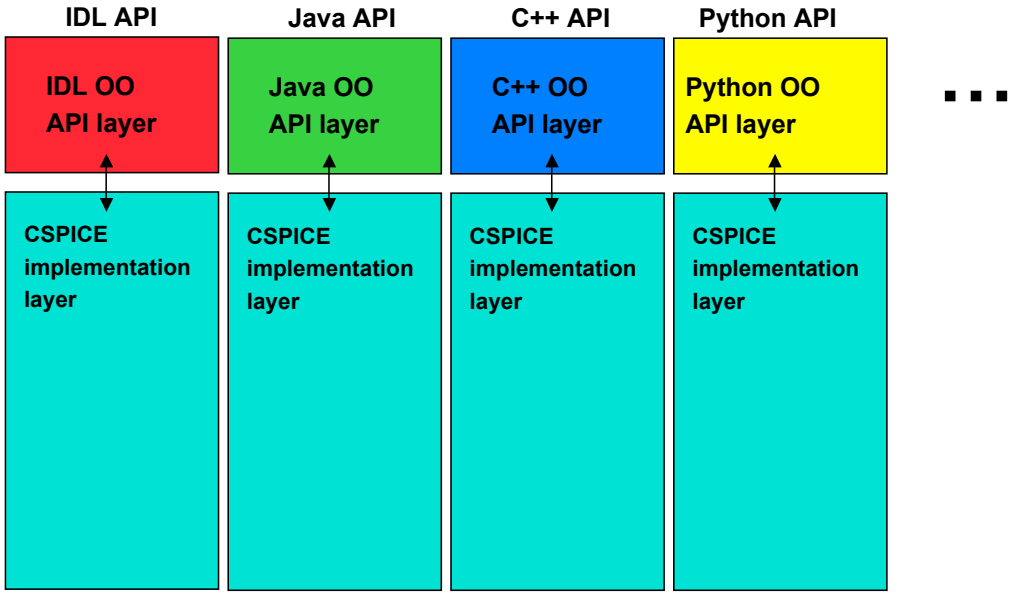


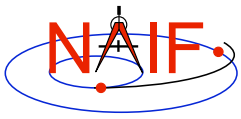


Possible OO SPICE Layered Structure -1

Navigation and Ancillary Information Facility

Partially OO: parallel APIs, CSPICE implementation

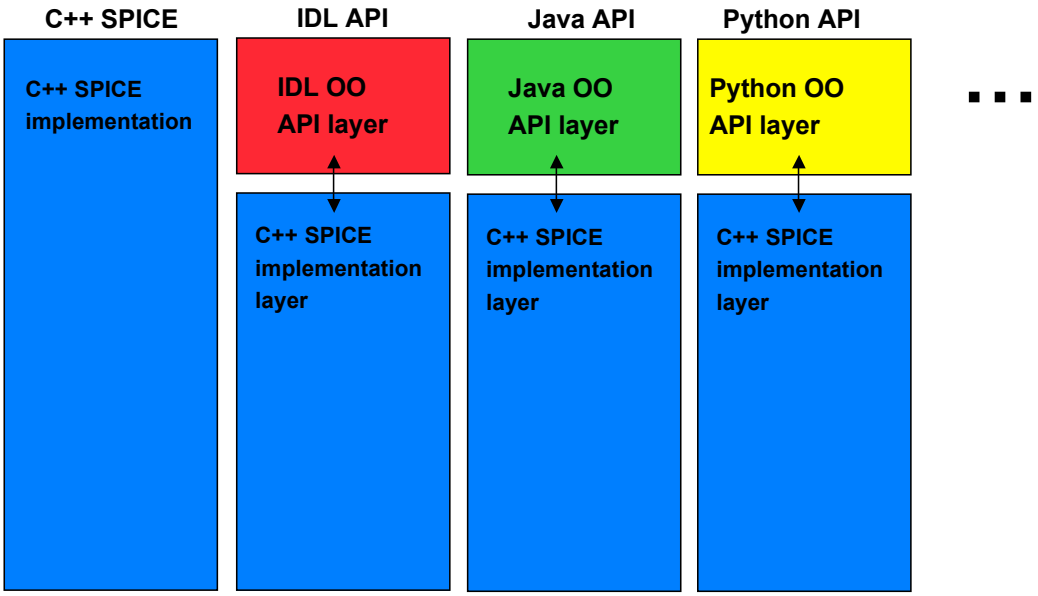


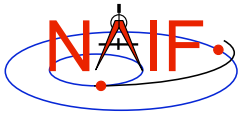


Possible OO SPICE Layered Structure -2

Navigation and Ancillary Information Facility

Fully OO: Parallel APIs for non-C++ Toolkits, C++ implementation



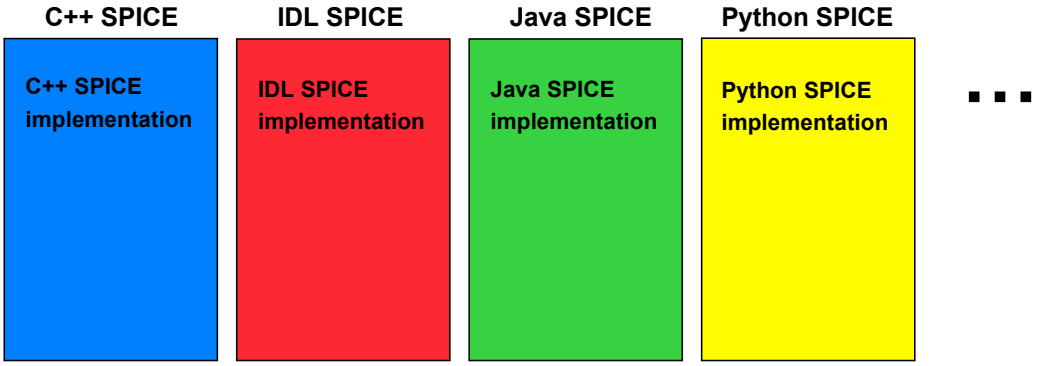


Independent OO SPICE Implementations

Navigation and Ancillary Information Facility

Multiple fully OO systems: completely independent implementations in different languages

- Functionality of systems in different languages may diverge as Toolkit grows.
  - Each system can take advantage of features unique to its implementation language
- Maintenance and development cost appears prohibitive
- But this design *might* be feasible using automatic translation techniques



Possible OO SPICE Distributed Implementation

Navigation and Ancillary Information Facility

Distributed OO implementation

- SPICE OO implementation language is transparent to clients
- May be unsuitable for number-crunching applications
  - Tight coupling with SPICE objects preferable for these applications