Exception Handling

April 2016

Topics

- **What Exceptions Are**
- **Language Dependencies**
- **C and Fortran Error Handling Features**
- **Error Messages**
- **Error Handling Actions**
- **Error Device**
- **Customize Error Handling**
- **Get Error Status**
- **Signal Errors**
- **Icy Error Handling**
- **Mice Error Handling**
- **Recommendations**

- **Run-time error conditions such as:**
  - **Files**
    - » **Required files not loaded.**
    - » **Gaps in data.**
    - » **Corrupted or malformed files (e.g. ftp'd in wrong mode).**
  - **Invalid subroutine/function arguments**
    - » **String values unrecognized.**
    - » **Numeric values out of range.**
    - » **Data type/dimension mismatch.**
  - **Arithmetic errors**
    - » **Divide by zero, square root of a negative number.**
  - **Environment problems**
    - » **Insufficient disk space for output files.**
   - » **Lack of required read/write permission/privileges.**

- **Valid but unusual conditions, such as:**
  - » **Normalize the zero vector.**
  - » **Find the rotation axis of the identity matrix.**
  - » **Find the boresight intercept lat/lon for a non-intercept case.**
  - » **Find a substring where the end index precedes the start index.**
  - – **Such cases are normally not SPICE "Error Conditions"**
  - – **Typically must be handled by a logical branch**

- **Errors found by analysis tools, such as:**
  - » **Invalid SQL query.**
  - » **Invalid string representing number (borderline case).**
  - – **Such cases are normally not SPICE "Error Conditions"**
  - – **However, if a SPICE parsing routine failed because it couldn't open a scratch file, that would be an "error condition."**

Exception Handling

Language Dependencies

- **SPICELIB and CSPICE provide essentially identical error handling capabilities.**

- **Icy and Mice provide similar error handling functionality; this functionality is quite different from that of CSPICE.**
  - These systems do rely on CSPICE for most error <u>detection</u>.
  - Icy and Mice provide no API for customizing underlying CSPICE error handling behavior.
  - Short, long, and traceback error messages are merged into a single, parsable, message.
  - Use IDL or MATLAB features to customize error handling…
    » to prevent your program from stopping
    » to capture SPICE error messages

- **Most of this tutorial deals with SPICELIB and CSPICE error handling.**
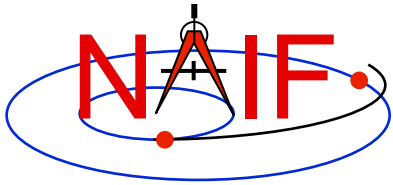 - There is a bit on Icy and Mice near the end.
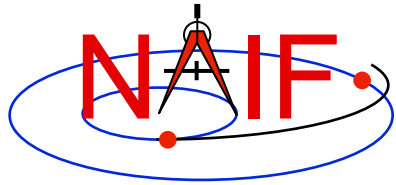
NAIF

- ## Error handling in SPICE: safety first
  - Trap errors where they occur; don't let them propagate
    - » Don't let errors "fall through" to the operating system.
  - Supply meaningful diagnostic messages
    - » Incorporate relevant run-time data.
    - » Supply context in human-readable form.
  - Don't depend on callers to handle errors
    - » Normally, "error flags" are not returned to callers.
  - Stop unless told not to
    - » Don't try to continue by making "smart guesses."

- ## Subroutine interface for error handling
  - Interface routines called within SPICE may be called by users' application programs
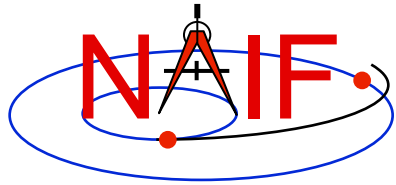
Navigation and Ancillary Information Facility

- **Signal errors**
  - Create descriptive messages when and where an error is detected
    » Short message, long message, (explanation), traceback
  - "Signal" the error: set error status, output messages
    » By default, CSPICE error output goes to stdout (not stderr)

- **Retrieve error information**
  - Get status and error messages via subroutine calls

- **Customize error response---actions taken when an error occurs.**
  - Set error handling mode ("action")
  - Set error output device
  - Set message selection

- **Inhibit tracing**
  - To improve run-time performance (only for thoroughly debugged code)

Error Messages

- ## Short message
  - **Up to 25 characters.**
  - **Can easily be compared with expected value.**
    - » **Example: SPICE(FILEOPENFAILED).**

- ## Long message
  - **Up to 1840 characters.**
  - **Can contain values supplied at run time.**
    - » **Example: 'The file <sat077.bsp> was not found.'**

- ## Traceback
  - **Shows call tree above routine where error was signaled.**
    - » **Not dependent on system tracing capability.**
   - » **Don't need a "crash" to obtain a traceback.**

- **ABORT**
  - **Designed for safety.**
    - » **Output messages and traceback to your screen or stdout.**
    - » **Stop program; return status code if possible.**

- **RETURN**
  - **For use in programs that must keep running.**
  - **Attempts to return control to the calling application.**
  - **Preserves error information so calling application can respond.**
    - » **Output messages to current error device.**
    - » **Set error status to "true": FAILED() will return "true."**
    - » **Set "return" status to "true": RETURN() will return "true."**
    - » **Most SPICE routines will return on entry. Very simple routines will generally execute anyway.**

--continues--
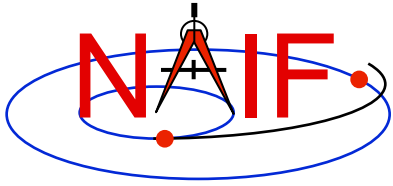
» **Capture traceback at point where error was signaled.**

» **Inhibit error message writing and error signaling.**

» **Must call RESET to resume normal error handling.**

Error Device

- ## Destination of error messages
  - Screen/stdout (default)
  - Designated file
    - » Error diagnostics are appended to the file as errors are encountered.
  - "NULL" --- suppress output
    - » When the NULL device is specified, error messages can still be retrieved using API calls.

- ## Limitations
  - In C, cannot send messages to stderr.
  - In C, writing to a file opened by means other than calling errdev_c is possible only if CSPICE routines were used to open the file.
    - » These limitations may be removed in a later version of CSPICE.

- ## Set error action
  - CALL ERRACT ('SET', 'RETURN')
  - erract_c ("set", LEN, "return");
    - » Length argument is ignored when action is "set"; when action is "get", LEN should be set to the available room in the output string, for example:
    - » erract_c ("get", ACTLEN, action);

- ## Set error device
  - CALL ERRDEV ('SET', 'errlog.txt')
  - errdev_c ("set", LEN, "errlog.txt");

- ## Select error messages
  - CALL ERRPRT ('SET', 'NONE, SHORT, TRACEBACK')
    - » If tracing is disabled (see next page), selecting TRACEBACK has no effect.
 - errprt_c ("set", LEN, "none, short, traceback");

- **Disable tracing**
  - **Normally done to speed up execution by a few percent**
  - **Benefit is highly dependent on application**
  - **NAIF normally recommends users not turn tracing off**
  - **Use TRCOFF:**
    - » **CALL TRCOFF   or   trcoff_c();**
      - **Do this at the beginning of your program.**
      - **Once disabled you cannot re-enable tracing during a program run.**

Get Error Status - 1

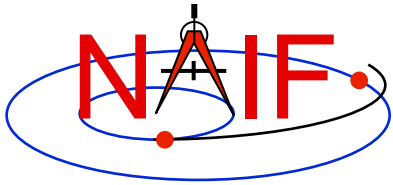- **Use FAILED to determine whether an error has been signaled**
  - `IF ( FAILED() ) THEN …`
  - `if ( failed_c() ) { …`

- **Use FAILED after calling one or more SPICE routines in a sequence**
  - Normally, it's safe to call a series of SPICE routines without testing FAILED after each call

- **Use GETMSG to retrieve short or long error messages**
  - `CALL GETMSG ( 'SHORT', SMSG )`
  - `getmsg_c ( "short", LEN, smsg );`
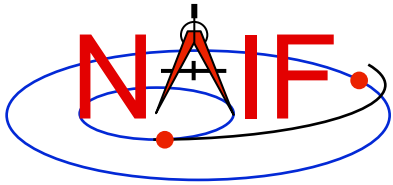
- **Use QCKTRC or TRCDEP and TRCNAM to retrieve traceback message**

- **Test value of RETURN() to determine whether routines should return on entry**
  - Only relevant if user code is designed to support RETURN mode

- **Handle error condition, then reset error status:**
  - `CALL RESET`
  - `reset_c();`
  - In Icy-based applications you only handle error condition; a reset is automatically performed by Icy

Navigation and Ancillary Information Facility
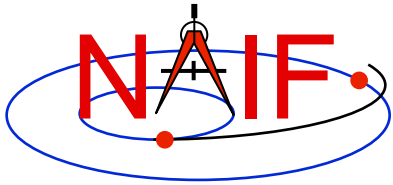
- # Create long error message
  - **Up to 1840 characters**
  - **Use SETMSG**
    - » `CALL SETMSG ( 'File <#> was not found.' )`
    - » `setmsg_c    ( "File <#> was not found." );`

- # Substitute string, integer, or d.p. values at run time
  - **Use ERRCH**
    - » `CALL ERRCH ( '#', 'cassini.bsp' )`
    - » `errch_c     ( "#", "cassini.bsp" );`
  - **Also can use ERRINT, ERRDP**
  - **In Fortran, can refer to files by logical unit numbers: ERRFNM**

Exception Handling

Navigation and Ancillary Information Facility

- ## Signal error
  - Use SIGERR to signal error. Supply short error message as input to SIGERR.
    - » `CALL SIGERR ( 'FILE OPEN FAILED' )`
    - » `sigerr_c    ( "FILE OPEN FAILED" );`
  - "Signaling" error causes SPICE error response to occur
    - » Output messages, if enabled
    - » Set error status
    - » Set return status, if error action is RETURN
    - » Inhibit further error signaling if in RETURN mode
    - » Stop program if in abort mode

- ## Reset error status after handling error
  - CALL RESET()
  - reset_c()

Icy Error Handling

- **Error action:**
  - By default, a SPICE error signal stops execution of IDL scripts; a SPICE error message is displayed; control returns to the execution level (normally the command prompt).
  - Icy sets the CSPICE shared object library's error handling system to RETURN mode. No other modes are used.
    - » The CSPICE error state is reset after detecting an error.
  - Use the IDL CATCH feature to respond to error condition.

- **Error status**
  - Value of !error_state.name
    - » ICY_M_BAD_IDL_ARGS - indicates invalid argument list.
    - » ICY_M_SPICE_ERROR - indicates occurrence of a SPICE error.

- **Error message**
  - CSPICE short, long, and traceback error messages are merged into a single, parsable, message.
    - » The merged error message is contained in the variable !error_state.msg.
    - » Example:

```
CSPICE_ET2UTC: SPICE(MISSINGTIMEINFO): [et2utc->ET2UTC->UNITIM]
                The following, needed to convert between the
                uniform time scales, could not be found in the
                kernel pool: DELTET/DELTA_T_A, DELTET/K,
                DELTET/EB, DELTET/M. Your program may have failed to load…
```

Mice Error Handling

- **Error action**
  - By default, a SPICE error signal stops execution of MATLAB scripts; a SPICE error message is displayed; control returns to the execution level.
  - Mice sets the CSPICE shared object library's error handling system to RETURN mode. No other modes are used.
    - » The CSPICE error state is reset after detecting an error.
  - Use the MATLAB try/catch construct to respond to error condition.

- **Error message**
  - CSPICE short, long, and traceback error messages are merged into a single, parsable, message.
    - » **Example:**
      ```
      ??? SPICE(MISSINGTIMEINFO): [et2utc->ET2UTC->UNITIM]
      The following, needed to convert between the
      uniform time scales, could not be found in the
      kernel pool: DELTET/DELTA_T_A, DELTET/K,
      DELTET/EB, DELTET/M. Your program may have failed to load…
      ```

- **Use the MATLAB function lasterror to retrieve SPICE error diagnostics. When a SPICE error occurs:**
  - the "message" field of the structure returned by lasterror contains the SPICE error message.
  - the "stack" field of this structure refers to the location in the m-file from which the Mice wrapper was called (and so is generally not useful).
  - the "identifier" field of this structure currently is not set.

Recommendations

- **For easier problem solving**
  - **Leave tracing enabled when debugging.**
  - **Always test FAILED after a sequence of one or more consecutive calls to SPICE routines.**
  - **Don't throw away error output. It may be the only useful clue as to what's going wrong.**
    - » **Programs that must suppress SPICE error output should trap it and provide a means for retrieving it.**
      - **Test FAILED to see whether an error occurred.**
      - **Use GETMSG to retrieve error messages**
      - **Use RESET to clear the error condition**
  - **Use SPICE error handling in your own code where appropriate.**
  - **When reporting errors to NAIF, have SPICE error message output available**
    - » **Note whether error output is actually from SPICE routines, from non-SPICE code, or was generated at the system level.**