FORTRAN 95 Nuclear Structure Data File Library
November 14, 2005

This document describes the contents of the subprogram library NSDFLIB95 that contains FORTRAN 95 versions of the subprogram library NSDLIB and additional subprogram libraries.

With the exception of the subroutines GET_COMMAND_LINE, OPEN_SORT_INPUT, and OPEN_SORT_OUTPUT, all subprograms in this library are written to conform to the ANSI standard for FORTRAN 95 and are, therefore, generally machine independent.

List of Subprograms

1. String Processing Subprograms
   a. INTEGER(KIND=4) FUNCTION LENSTR
   b. INTEGER(KIND=4) FUNCTION LSTRING
   c. INTEGER(KIND=4) FUNCTION TYPSTR
   d. SUBROUTINE ADDSTR
   e. SUBROUTINE DELSTR
   f. SUBROUTINE SQZSTR
   g. SUBROUTINE REPCHR
   h. SUBROUTINE REPSTR
   i. INTEGER(KIND=4) FUNCTION INDEXF
   j. INTEGER(KIND=4) FUNCTION BREAK
   k. INTEGER(KIND=4) FUNCTION SPAN
   l. SUBROUTINE PADLFT
   m. SUBROUTINE UPSTR
   n. SUBROUTINE LOSTR
   o. CHARACTER(LEN=*) FUNCTION UPCASE
   p. CHARACTER(LEN=*) FUNCTION LOCASE
   q. INTEGER(KIND=4) FUNCTION KSEARCH
   r. SUBROUTINE TOKEN
      ENTRY TOKENL
   s. SUBROUTINE SCOUNT
   t. SUBROUTINE SUPBLANK
   u. SUBROUTINE NOLBLANK
      ENTRY LBSUP
      ENTRY NOLBLANKL
   v. SUBROUTINE TRANSNUC
2. Number to String and String to Number Subprograms
   a. REAL(KIND=4) FUNCTION VALSTR
      REAL(KIND=8) FUNCTION DVALST
   b. INTEGER(KIND=4) FUNCTION IVLSTR
   c. INTEGER(KIND=4) FUNCTION RLSCN
      INTEGER(KIND=4) FUNCTION DRLSCN
   d. INTEGER(KIND=4) FUNCTION INTSCN
   e. SUBROUTINE NUMSTR
   f. SUBROUTINE CNVS2U
      ENTRY DCNVSU
   g. SUBROUTINE ZSYM
      ENTRY IZEL
      ENTRY IZELW
   h. SUBROUTINE CNVU2S
      ENTRY DCNVUS
   i. SUBROUTINE SUPALF
   j. SUBROUTINE SUPEMB
   k. LOGICAL(KIND=4) FUNCTION IVRFLW
   l. SUBROUTINE KNVIX
   m. SUBROUTINE SCALDX
   n. SUBROUTINE SCALX
   o. SUBROUTINE SCAL10
   p. SUBROUTINE KNVI2S
1. Mathematics Subprograms
   a. SUBROUTINE UADD
   b. SUBROUTINE USUB
   c. SUBROUTINE UMULT
   d. SUBROUTINE UDIV
   e. COMPLEX(KIND=4) FUNCTION GAMA
   f. COMPLEX(KIND=4) FUNCTION GAMZ
   g. COMPLEX(KIND=4) FUNCTION GAM1
   h. COMPLEX(KIND=4) FUNCTION GAM2
   i. COMPLEX(KIND=4) FUNCTION HYPERG
2. Time & Date Subprograms
   a. SUBROUTINE GET_TIME
   b. SUBROUTINE DATE_20
   c. SUBROUTINE IDATE_20
3. Command Line Subprogram
   a. SUBROUTINE GET_COMMAND_LINE
4. File Manipulation Subprogram
   a. SUBROUTINE DELETE_FILE
5. Sorting Subprograms
   a. SUBROUTINE SORT
      ENTRY SET_SORT_FILE
   b. SUBROUTINE FSORT
   c. SUBROUTINE READ_SORT_IN
   d. SUBROUTINE SORT_WRITE
   e. SUBROUTINE SRTKEYS
   f. SUBROUTINE OPEN_SORT_INPUT
   g. SUBROUTINE OPEN_SORT_OUTPUT

String Processing Library

FORTRAN 95 provides many great and wondrous features for string processing. However, there are some features that are not provided for by the standard. Therefore, this subprogram library has been written to include some useful subroutines and functions to manipulate character strings or return information about character strings.

INTEGER(KIND=4) FUNCTION LENSTR (string)

Function LENSTR will return the character position of the last non-blank (or non-null) character of the argument string. This value also represents the length of the string viewed as terminating with the last non-blank (or non-null) character.

Example:

```
CHARACTER(LEN=10) :: str
INTEGER(KIND=4) :: i,j
INTEGER(KIND=4),INTRINSIC :: LEN
INTEGER(KIND=4),EXTERNAL :: LENSTR
str = 'ABC'
i = LEN(str)
j = LENSTR(str)
WRITE(*,*) i, j
```

will display the values 10, 3.

See also subroutine SCOUNT and function LSTRING.

Note: The FORTRAN 95 intrinsic function LEN_TRIM provides a similar functionality.

INTEGER(KIND=4) FUNCTION LSTRING (string)

Function LSTRING will return the position of the last non-blank character in a string.

Example:

```
CHARACTER(LEN=10) :: str
INTEGER(KIND=4) :: i,j
INTEGER(KIND=4),INTRINSIC :: LEN
INTEGER(KIND=4),EXTERNAL :: LSTRING
str = 'ABC'
i = LEN(str)
j = LSTRING(str)
WRITE(*,*) i, j
```

will display the values 10, 3.

See also subroutine SCOUNT and function LENSTR.

Note: The FORTRAN 95 intrinsic function LEN_TRIM provides a similar functionality.

INTEGER(KIND=4) FUNCTION TYPSTR (string)

Function TYPSTR will scan the argument **string** and return one of the following values as the type of the string:

 0 if the string contains all blanks.
 1 if the string contains all digits (0 - 9) only.
 2 if the string contains all upper case letters (A - Z).
-1 if the string contains other characters or is a mixture of types.
-2 if the string is a FORTRAN number

Example:

```
INTEGER(KIND=4),EXTERNAL :: TYPSTR
CHARACTER(LEN=5) :: string
string = '13579'
WRITE(*,*) TYPSTR(string)
```

will display the value 1.

Trailing blanks are ignored but leading blanks count as non-numeric, non-alpha characters except that for FORTRAN numbers beginning blanks are also allowed.

Note: In determining a FORTRAN number the functions INTSCN and RLSCN are used. Any side effects of these functions may affect the determination.

SUBROUTINE ADDSTR (string, pos, new)

Subroutine ADDSTR will insert the **new** string into the existing **string** by moving the characters of the existing string starting at the specified position, **pos**, to the right by an amount equal to the length of the **new** string.

Example:

```
CHARACTER(LEN=10) :: string
string = 'ABCD'
CALL ADDSTR(string, 3, 'XYZ')
WRITE(*,*) string
```

will display the value 'ABXYZCD '.

Note: If the resulting string exceeds the length of the first argument, then the resulting string will be truncated on the right before the argument is returned from the subroutine.

SUBROUTINE DELSTR (string, pos, len)

Subroutine DELSTR will delete the specified number of characters, **len**, from the argument **string** starting at the specified position, **pos**. The remaining characters to the right will be moved left. The resulting string will be padded on the right with blanks to fill out the argument **string**'s length.

Example:

```
CHARACTER(LEN=10) :: string
string = 'ABXYZCD'
CALL DELSTR(string, 3, 3)
WRITE(*,*) string
```

will display the value 'ABCD '.

SUBROUTINE SQZSTR (string, char)

Subroutine SQZSTR will scan the argument **string** looking for the specified character, **char**. When one is found it will be deleted (as in DELSTR). The scan will continue looking for all of the occurrences of the specified character. The possibly shorter string will be padded on the right with blanks as needed.

Example:

```
CHARACTER(LEN=10) :: STRING
string = 'NOW IS THE'
CALL SQZSTR(string, ' ')
WRITE(*,*) string
```

will display the value 'NOWISTHE '

See also subroutines NOLBLANK, LBSUP, NOLBLANKL, and SUPBLANK.

SUBROUTINE REPCHR (string, old, new)

Subroutine REPCHR will scan the argument **string** looking for any of the characters that exist within the **old** string. If any are found, they will be replaced by the corresponding character from the **new** string. The original and modified strings should be the same length.

Example:

```
CHARACTER(LEN=10) :: string
string = 'NOW IS THE'
CALL REPCHR(string, ' AEIOU', '_aeiou')
WRITE(*,*) string
```

will display the value 'NoW_iS_THe'.

See also subroutine REPSTR.

SUBROUTINE REPSTR (string, old, new)

Subroutine REPSTR will scan the argument **string** searching for all occurrences of the **old** string. If any are found, they will all be replaced by the **new** string. If the new string is CHAR(0) (*i.e.*, a string of length 1 with value the character null), then the **old** string will be removed and nothing will replace it. The length of the **old** and **new** strings may be greater than or equal to one byte.

Example:

```
CHARACTER(LEN=10) :: string
string = 'NOW IS THE'
CALL REPSTR(string, 'IS', 'WAS')
WRITE(*,*) string
```

will display the value 'NOW WAS TH'.

Please note that the subroutine REPSTR will call the subroutines DELSTR and ADDSTR to delete the old string and add the new string (if needed). Any side effects of these subroutines will hold true for subroutine REPSTR.

See also subroutine REPCHR.

INTEGER(KIND=4) FUNCTION INDEXF (string, pos, substr)

Function INDEXF behaves in a manner similar to the standard function INDEX except that INDEXF controls where the search for the sub-string should begin with the argument **pos**. Occurrences of the sub-string which precede position pos will not be considered. The value returned will be zero (0) or the index of the substring, **substr** with respect to the beginning of **string** as is the case with INDEX.

Example:

```
INTEGER(KIND=4),EXTERNAL :: INDEXF
INTEGER(KIND=4) :: i
CHARACTER(LEN=10) :: string
string = 'ABC ABC AB'
i = INDEXF(string, 3, 'A')
WRITE(*,*) I
```

will display the value 5.

INTEGER(KIND=4) FUNCTION BREAK (string, pos, brkstr)

Function BREAK will scan the argument **string** starting from position **pos** looking for the first character (the break character) which is one of the characters in the **brkstr** string. If found, the position of the break character in string will be returned. If not found, the length of string plus one (*i.e.*, the position of the character after the last character in string) will be returned. If the starting position, pos, is not within the limits of the argument string, the position pos will be returned. Characters should not be repeated in the break string.

Example:

```
CHARACTER(LEN=10) :: string
INTEGER(KIND=4),EXTERNAL :: BREAK
INTEGER(KIND=4) :: i
string = 'NOW IS THE'
i = BREAK(string, 3, 'AEIOU')
WRITE(*,*) I
```

will display the value 5.

See also integer function SPAN.

INTEGER(KIND=4) FUNCTION SPAN (string, pos, spnstr)

Function SPAN will scan the argument string starting from position **pos** looking for the first character (the break character, **spnstr**) after **pos** which is *not* one of the characters in the span string. If found, the position of the break character in string will be returned. If not found, the length of string plus one (*i.e.*, the position of the character after the last character in string) will be returned. If the starting position, **pos**, is not within the limits of the argument string, the position pos will be returned. Characters should not be repeated in the break string.

Example:

```
CHARACTER(LEN=10) :: STRING
INTEGER(KIND=4),EXTERNAL :: SPAN
INTEGER(KIND=4) :: i
STRING = '   NOW THE'
i = SPAN(string, 1, ' ')
WRITE(*,*) I
```

will display the value 4.

See also integer function BREAK.

SUBROUTINE PADLFT (string, len)

Subroutine PADLFT will insert blanks to the left of the argument **string** to make the string **len** characters long. If len is less than or equal to the length of the string (as defined by integer function LENSTR) or greater than the length of the string (as defined by the intrinsic integer function LEN), no action will be taken.

Example:

```
CHARACTER(LEN=10) :: string
string='ABC'
CALL PADLFT(string,6)
WRITE(*,*) string
```

will display '   ABC'.

SUBROUTINE UPSTR (string)

Subroutine UPSTR will convert a **string** to all upper case.

Example:

```
CHARACTER(LEN=10) :: string
string='ab12c'
Call UPSTR(string)
WRITE(*,*)string
```

will display 'AB12C'.

See also character function UPCASE.

SUBROUTINE LOSTR (string)

Subroutine LOSTR will convert a **string** to all lower case.

Example:

```
CHARACTER(LEN=10) :: string
string='AB12C'
Call LOSTR(string)
WRITE(*,*)string
```

will display 'ab12c'.

See also character function LOCASE.

CHARACTER(LEN=*) FUNCTION UPCASE (string)

Function UPCASE will convert a **string** to all upper case.

Example:

```
CHARACTER(LEN=10) :: string,newstring
CHARACTER(LEN=10),EXTERNAL :: UPCASE
string='ab12c'
newstring=UPCASE(string)
WRITE(*,*)newstring
```

will display 'AB12C'.

See also subroutine UPSTR.

CHARACTER(LEN=*) FUNCTION LOCASE (string)

Function LOCASE will convert a **string** to all lower case.

Example:

```
CHARACTER(LEN=10) :: string,newstring
CHARACTER(LEN=10),EXTERNAL :: LOCASE
string='AB12C'
newstring=LOCASE(string)
WRITE(*,*)newstring
```

will display 'ab12c'.

See also subroutine LOSTR.

INTEGER(KIND=4) FUNCTION KSEARCH (string, delim, idnum)

Function KSEARCH will return the position of the n^{th}, **idnum**, **o**ccurence of a delimiter, **delim**, in a **string**. If the desired occurence is a negative number zero is returned. If the end of the string is reached before the desired occurence, the length of the string+1 is returned.

Example:

```
CHARACTER(LEN=10) :: string

CHARACTER(LEN=1) :: delim

INTEGER(KIND=4),EXTERNAL :: KSEARCH

INTEGER(KIND=4) :: ipos

string='ABAB'

delim='B'

ipos=KSEARCH(string,delim,2)

WRITE(*,*) ipos
```

will display 4.

SUBROUTINE TOKEN (instr, delim, itok, outstr)
ENTRY TOKENL (instr, delim, itok, outstr, nstr)

Subroutine TOKEN extracts a delimited sub-string, **delim** from a string, **instr**.

Example:

```
CHARACTER(LEN=20) :: string,newstring
CHARACTER(LEN=2) :: delim
del='&&'
string='AB&&CD'
CALL TOKEN(string,delim,1,newstring)
WRITE(*,*)newstring
```

will display 'CD'.

Entry TOKENL will also return the length of the new string in **nstr**.

SCOUNT (string, lstring)

Subroutine SCOUNT will return the position of the last non-blank character. This value also represents the length of the string viewed as terminating with the last non-blank character.

Example:

```
CHARACTER(LEN=10) :: str
INTEGER(KIND=4) :: i,j
INTEGER(KIND=4),INTRINSIC :: LEN
str = 'ABC'
i = LEN(str)
CALL SCOUNT(str,j)
WRITE(*,*) i, j
```

will display the values 10, 3.

See also integer function LENSTR.

Note: The FORTRAN 95 intrinsic function LEN_TRIM provides a similar functionality.

SUBROUTINE SUPBLANK (string, nc)

Subroutine SUPBLANK will remove all blanks from a **string** and also return the length of the resultant string in **nc**.

Example:

```
CHARACTER(LEN=10) :: string
INTEGER(KIND=4) :: nc
string = 'NOW IS THE'
CALL SUPBLANK(string, nc)
WRITE(*,*) string,nc
```

will display the values 'NOWISTHE ' and 8.

See also subroutines NOLBLANK, LBSUP, NOLBLANKL, and SQZSTR.

SUBROUTINE NOLBLANK (string)
ENTRY LBSUP (string)
ENTRY NOLBLANKL (string, nstr)

Subroutine NOLBLANK and entry LBSUP will remove leading blanks from a string.

Example:

```
CHARACTER(LEN=10) :: string
string='      ABC'
CALL LBSUP(string)
WRITE(*,*) STRING
```

will display 'ABC’.

Entry NOLBLANKL will also return the length of the resultant string in NSTR.

See also subroutines SQZSTR and SUPBLANK.

SUBROUTINE TRANSNUC (instr, rstr, sstr, ierr)

Subroutine TRANSNUC translates between nuclear structure presentation of nuclides (AAAZZ) and nuclear reaction presentation (ZZ-AAA) as necessary.

Name	Description	Type
instr	Input string to be translated	CHARACTER(LEN=*)
rstr	Resultant reaction format string	CHARACTER(LEN=*)
sstr	Resultant structure format string	CHARACTER(LEN=*)
ierr	Error (=1) if input string cannot be parsed	INTEGER(KIND=4)

Library of Number to String and String to Number Routines

Most of the subprograms in this library deal with the conversion of numeric variables to string variables or the converse.

REAL(KIND=4) FUNCTION VALSTR (string)
REAL(KIND=8) FUNCTION DVALST (string)

Function VALSTR will scan the argument **string** looking for a leading real type numeric value (see RLSCN). The value returned will be the floating point value of the text scanned.

Example:

```
CHARACTER(LEN=10) :: string
REAL(KIND=4) :: value
REAL(KIND=4),EXTERNAL :: VALSTR
string = '123.45E2'
value = VALSTR(string)
WRITE(*,*) value
```

will display the value 12345.00.

Double precision function DVALST will return the double precision value of the scanned string.

Please note that the functions VALSTR and DVLSTR call the functions RLSCN and DRLSCN, respectively, to obtain the returned value. Any side effects of these functions will hold true for the functions VALSTR and DVALST.

See also integer functions RLSCN and DRLSCN and subroutine CNVS2U.

INTEGER(KIND=4) FUNCTION IVLSTR (string)

Function IVLSTR will scan the argument **string** looking for a leading integer type numeric value (see INTSCN). The value returned will be the integer value of the text scanned. Please note that any side effects of the function INTSCN will hold true for the function IVLSTR.

Example:

```
CHARACTER(LEN=10) :: string
INTEGER(KIND=4) :: ivalue
INTEGER(KIND=4),EXTERNAL :: IVLSTR
string = '123.45E2'
ivalue = IVLSTR(string)
WRITE(*,*) ivalue
```

will display the value 123.

See also integer function INTSCN.

INTEGER(KIND=4) FUNCTION RLSCN (string, pos, value)
INTEGER(KIND=4) FUNCTION DRLSCN (string, pos, value)

Function RLSCN will scan the argument **string** looking for a leading real type numeric value. Scanning begins at the position specified by **pos** and continues to the end of the string. Leading blanks will be ignored. **Value** is set to the numeric value of the string (internal floating point form). The function value is set to the position within the string where the numeric string ends plus one (*i.e.*, the break character).

The numeric string must have the form (after leading blanks):

[sign] d+ ['.' d*] ['E' [sign] d+] or

[sign] '.' d+ ['E' [sign] d+]

where sign is '+' or '-',
d* is zero or more digits,
d+ is one or more digits,
'.' and 'E' are literal ('e' is also accepted),
brackets ([and]) delimit optional sequences.

Example:

```
CHARACTER(LEN=10) :: string
INTEGER(KIND=4),EXTERNAL :: RLSCN
INTEGER(KIND=4) :: ipos
REAL(KIND=4) :: value
string = '123.45E2  '
ipos = RLSCN(string, 1, value)
WRITE(*,*) ipos, value
```

will display the values 9, 12345.00.

Integer function DRLSCN will perform the same function as RLSCN but dvalue returned will be in double precision.

Also note that since the ENSDF formats do not allow "D" floating-point numbers neither RLSCN or DRLSCN recognize these as FORTRAN numbers.

See also real function VALSTR, double precision function DVALST, and subroutine CNVS2U.

INTSCN (string, pos, signed, value)

Function INTSCN will scan the argument **string** looking for a leading integer type numeric value. Scanning begins at the position specified by **pos** and continues to the end of the string. Leading blanks will be ignored. The search may be for a **signed** (signed = .TRUE.) or unsigned (signed = .FALSE.) integer value. If signed, leading plus (+) or minus (-) is allowed. If unsigned, they will terminate the scan as they are invalid for an unsigned integer. **Value** is set to the numeric value of the string (internal integer form). The function value is set to the position within the string where the numeric string ends plus one (i.e., the break character). If the starting position, pos, is not within the limits of the argument string, the position pos will be returned and value will be set to zero. If the sign is the last character in the field (with no integer following it), the position pos or the index of the last leading blank will be returned and value will be set to zero.

Example:

```
CHARACTER(LEN=10) :: string
INTEGER(KIND=4),EXTERNAL :: INTSCN
INTEGER(KIND=4) :: ipos, jpos
INTEGER(KIND=4) :: ivalue,jvalue
string = '-123.45E2 '
ipos = INTSCN(string, 1, .TRUE.,  ivalue)
jpos = INTSCN(string, 1, .FALSE., jvalue)
WRITE(*,*) ipos, ivalue, jpos, jvalue
```

will display the values 5, -123, 1, 0.

See also integer function IVLSTR.

SUBROUTINE NUMSTR (number, string)

Subroutine NUMSTR will convert the argument integer **number** into character format and store it in **string**. The string will be right justified and blank filled on the left. If the length of string is too small to contain the integer number, string will be filled with asterisks ("*"'s).

Example:

```
CHARACTER(LEN=5) :: string
CALL NUMSTR(137, string)
WRITE(*,*) string
```

will display the value ' 137'.

See also subroutine KNVI2S.

SUBROUTINE CNVS2U (sx, sdx, y ,dy)
ENTRY DCNVSU (sx, sdx, x, dx)

Subroutine CNVS2U converts strings **sx** and **sdx** into two real numbers **y** and **dy** where **sdx** and **dy** are the uncertainties of **sx** and **y**.

Entry DCNVSU converts strings **sx** and **SDX** into two double precision numbers **x** and **dx** where **sdx** and **dx** are the uncertainties of **sx** and **x**.

The strings **sx** and **sdx** represent a value and its uncertainty in the ENSDF or Nuclear Data Sheets notation; that is a value and the uncertainty in the least significant digits of the value. If the string **sx** is a single non-numeric character, zero is returned for the number and uncertainty

Name	Description	Type
sx	Input character string	CHARACTER(LEN=*)
sdx	Input character string, uncertainty of SX	CHARACTER(LEN=*)
y	output real number	REAL(KIND=4)
dy	output real uncertainty	REAL(KIND=4)
x	output double precision number	REAL(KIND=8)
dx	output double precision uncertainty	REAL(KIND=8)

The input strings are stored in temporary strings of length 24.

See also integer functions RLSCN and DRLSCN and real function VALSTR.

SUBROUTINE ZSYM (el, sym)

Subroutine ZSYM translates element number (**el**) into symbol text (**sym**).

Name	Type
el	INTEGER(KIND=4)
sym	CHARACTER(LEN=2)

ENTRY IZEL (sym, el)

Entry IZEL translates symbol text (**sym**) into element number (**el**).

Name	Type
sym	CHARACTER(LEN=2)
el	INTEGER(KIND=4)

ENTRY IZELW (sym, el, izlmsg)

Entry IZELW translates symbol text (**sym**) into element number (**el**) and returns a warning (**izlmsg**).

Name	Type
sym	CHARACTER(LEN=2)
el	INTEGER(KIND=4)
izlmsg	CHARACTER(LEN=80)

Note: As of version 1.5 of NSDFLIB, the IUPAC adopted symbols for Z=104 through 109 have been implemented in ZSYM and IZEL; the adopted symbol for Z=110 was implemented in version 1.6c; and the adopted symbol for Z=111 was implemented in version 1.6e. IZEL and IZELW will still properly interpret the old ENSDF formalism of "04" through "11" for these elements but IZELW will return a warning message in **izlmsg**. Version 1.5c implemented the change of the chemical symbol for the neutron from "N " to "NN".

SUBROUTINE CNVU2S (y, dy, sx, lenx, sdx, lendx)
ENTRY DCNVUS (x, dx, sx, lenx, sdx, lendx)

Subroutine CNVU2S converts the real number **y**, with optional uncertainty **dy**, into string format.

Entry DCNVUS converts the double precision number **x**, with optional double precision uncertainty **dx**, into string format.

Name	Description	Type
y	Input real number to be converted.	REAL(KIND=4)
dy	Input real number uncertainty of y.	REAL(KIND=4)
x	Input double precision number to be converted.	REAL(KIND=8)
dx	Input double precision uncertainty of x.	REAL(KIND=8)
sx	Output string for x (and dx in format 2, see below).	CHARACTER(LEN=*)
lenx	Input length specifier for SX.	INTEGER(KIND=4)
sdx	Output string for dx (formats 1 and 3).	CHARACTER(LEN=*)
lendx	Input length specifier for sdx (formats 1 and 3) or a format flag(formats 2 and 4).	INTEGER(KIND=4)

One of the four formats is selected based on the values of dy (or dx) and lendx.

FORMAT 1: dy (or dx)>0.0, lendx>0.
sx and sdx are set.
sdx will be in the range of 1 to 25.
sx will be set as appropriate for the specified uncertainty.

FORMAT 2: dy (or dx)>0.0, lendx≤0.
sx only is set. sdx is not modified.
y (or x) and dy (or dx) are formatted into sx.
The uncertainty is not constrained to the range 1 to 25 if dx>25.0.
If lendx=0, the results will be set to the "natural" number of significant digits.
If lendx<0, the results will be set to -lendx significant digits.

FORMAT 3: dy (or dx)=0.0, lendx≥0.
sx and sdx are set.
sx will be set using 4 significant digits.
sdx will be blanked out to a length of lendx.

FORMAT 4: dy (or dx)=0.0, lendx<0.
sx only is set, sdx is not modified.
sx will be set using -lendx significant digits.

If the resultant string is too long to fit within the specified lengths or a correct string can not be derived, the returned string will be filled with asterisks ("*"'s).

CNVU2S (DCNVUS) calls subroutines IVRFLW, KNVIX, SCALDX, SCALX, SCAL10, and KNVI2S.

SUBROUTINE SUPALF (str)

Subroutine SUPALF converts all non-numeric characters in **str** to blanks (except ".", "E", "+", and "-").

Name	Description	Type
str	Input and output character string	CHARACTER(LEN=*)

SUBROUTINE SUPEMB (str)

Subroutine SUPEMB finds and eliminates unwanted embedded "+"'s and "-"'s so that "+"'s and "-"'s will appear only at the beginning of the string or right after E.

This subroutine should be used in addition to SUPALF when needed.

Name	Description	Type
str	Input and output character string	CHARACTER(LEN=*)

LOGICAL(KIND=4) FUNCTION IVRFLW (x, ipwr)

Logical Function IVRFLW checks to see if $\mathbf{x}\times10^{-\mathbf{ipwr}}$ may be converted to an integer without overflow errors. The valid range is between -2^{31} and $2^{31}-1$. If the value falls within this range, .FALSE. is returned. If it is outside this range, .TRUE. is returned.

Name	Description	Type
x	Input value	REAL(KIND=4)
ipwr	Input power	INTEGER(KIND=4)

SUBROUTINE KNVIX (ix, ipwr, sx, lenx)

Subroutine KNVIX converts **ix** with scale factor **ipwr** to a string **sx** of length **lenx**. If the string space is too small, asterisks ("*""s) are returned in string **sx**. If ipwr>0 or if $ix \times 10^{ipwr} < 10^{-4}$, exponential format is used.

Name	Description	Type
ix	Input value	INTEGER(KIND=4)
ipwr	Input power	INTEGER(KIND=4)
sx	Output character string	CHARACTER(LEN=*)
lenx	Length of string	INTEGER(KIND=4)

SUBROUTINE SCALDX (dx, idx, ipwr)

Subroutine SCALDX computes for the double precision value **dx**, the integer **idx** in the range from 3 to 25 and the integer **ipwr** which is the power of 10 to get back to the original. For example, if dx=0.0025D+0, then idx=25 and ipwr=-4 will be returned and if dx=35D+0, then idx=4 and ipwr=1 will be returned.

Name	Description	Type
dx	Input value	REAL(KIND=8)
idx	Integer value returned	INTEGER(KIND=4)
ipwr	Exponent returned	INTEGER(KIND=4)

SUBROUTINE SCALX (x, ix, ipwr)

Subroutine SCALX computes for the double precision value **x** and the integer value **ipwr** the integer value **ix** which corresponds to $x \times (10.0^{-ipwr})$. For example, if x=20.0D+0 and ipwr=-2, ix=0 will be returned and if x=-45.0D+0 and ipwr=-1, ix=-5 will be returned.

Name	Description	Type
x	Input value	REAL(KIND=8)
ix	Integer value returned	INTEGER(KIND=4)
ipwr	Input exponent value	INTEGER(KIND=4)

SUBROUTINE SCAL10 (ix, idx, ipwr)

Subroutine SCAL10 will, if both integer **ix** or **idx** are modulo 10 (exactly divisable by 10), reduce ix and idx by a factor of 10 and increase the integer **ipwr** by 1. For example if ix=300, idx=30, and ipwr=1, ix, then ix=30, idx=3, and ipwr=2 will be returned (both MOD(ix) and MOD(idx) are equal to zero) and if ix=300, idx=35, and ipwr=1, then the values are not changed (MOD(idx) is not equal to zero).

Name	Description	Type
ix	Input/return value	INTEGER(KIND=4)
idx	Input/return value	INTEGER(KIND=4)
ipwr	Input/return value	INTEGER(KIND=4)

SUBROUTINE KNVI2S (n, str, slen)

Subroutine KNVI2S converts the integer **n** into a right justified string, **str**, with string length **slen**. If slen is 0, the returned string is left justified. If n is too large for slen characters, asterisks ("*"'s) will fill str. The longest string considered is 11 characters according to the largest 4 byte character size.

Name	Description	Type
n	Input value	INTEGER(KIND=4)
str	Character string returned	CHARACTER(LEN=*)
slen	Length of character string	INTEGER(KIND=4)

See also subroutine NUMSTR.

Mathematical Routines Library

SUBROUTINE UADD(z, dz, x, dx, y, dy)

UADD computes the sum of two numbers and the uncertainty of the sum.

Name	Description	Type
x	input variable	REAL(KIND=4)
dx	uncertainty of x	REAL(KIND=4)
y	input variable	REAL(KIND=4)
dy	uncertainty of y	REAL(KIND=4)
z	output variable, z=x+y	REAL(KIND=4)
dz	uncertainty of z, dz=SQRT(dx*dx + dy*dy)	REAL(KIND=4)

SUBROUTINE USUB(z, dz, x, dx, y, dy)

USUB computes the difference of two numbers and the uncertainty of the difference.

Name	Description	Type
x	input variable	REAL(KIND=4)
dx	uncertainty of x	REAL(KIND=4)
y	input variable	REAL(KIND=4)
dy	uncertainty of y	REAL(KIND=4)
z	output variable, z=x-y	REAL(KIND=4)
dz	uncertainty of z, dz=SQRT(dx*dx + dy*dy)	REAL(KIND=4)

SUBROUTINE UMULT(z, dz, x, dx, y, dy)

UMULT computes the product of two numbers and the uncertainty of the product.

Name	Description	Type
x	input variable	REAL(KIND=4)
dx	uncertainty of x	REAL(KIND=4)
y	input variable	REAL(KIND=4)
dy	uncertainty of y	REAL(KIND=4)
z	output variable, $z=x \times y$	REAL(KIND=4)
dz	uncertainty of z, $dz=z \times SQRT((dx/x)^2+(dy/y)^2)$	REAL(KIND=4)

SUBROUTINE UDIV (z, dz, x, dx, y, dy)

UDIV computes the result of the division of one number by another and the uncertainty in the result.

Name	Description	Type
x	input variable	REAL(KIND=4)
dx	uncertainty of x	REAL(KIND=4)
y	input variable	REAL(KIND=4)
dy	uncertainty of y	REAL(KIND=4)
z	output variable, $z=x/y$	REAL(KIND=4)
dz	uncertainty of z, $dz=z\times SQRT((dx/x)^2+(dy/y)^2)$	REAL(KIND=4)

COMPLEX(KIND=4) FUNCTION GAMA (x)

Complex function GAMA returns value $\Gamma(x)$ for all values of complex variable **x**.

Name	Description	Type
x	Input variable	COMPLEX(KIND=4)

COMPLEX(KIND=4) FUNCTION GAMZ (x)

Complex function GAMAZ returns value Γ (x) for **x(real), x(imag)** $\geq$ **0**.

Name	Description	Type
x	Input variable x(real), x(imag) $\geq$ 0	COMPLEX(KIND=4)

COMPLEX(KIND=4) FUNCTION GAM1 (x)

Complex function GAM1 returns value Γ (x) for **x(real) ≥0, 0≤x(imag) ≤1**.

Name	Description	Type
x	Input variable x(real) ≥0, 0<x(imag) ≤1	COMPLEX(KIND=4)

COMPLEX(KIND=4) FUNCTION GAM2 (x)

Complex function GAM2 returns value Γ (x) for **0≤x(real) ≤1, 0≤x(imag) ≤1**, using Pade-Power approximation of 1/Γ (x).

Name	Description	Type
x	Input variable 0≤x(real) ≤1, 0≤x(imag) ≤1	COMPLEX(KIND=4)

COMPLEX(KIND=4) FUNCTION HYPERG(a, b, x)

Complex Function HYPERG returns HYPERGEOMETRIC(a, b, x). Adopted from 1604 subroutine of C.W. Nestor.

Name	Description	Type
a	Input variable	COMPLEX(KIND=4)
b	Input variable	COMPLEX(KIND=4)
x	Input variable	COMPLEX(KIND=4)

Time & Date Routines Library

The following subroutines use the intrinsic subroutine DATE_AND_TIME. This subroutine requires the environmental TZ to be properly set. For OpenVMS, this may be done by:

```
DEFINE TZ SYS$TIMEZONE_NAME
```

For other operating systems, please check your compiler's language reference manual.

SUBROUTINE GET_TIME (ipath, ieru)

Subroutine GET_TIME writes the begin run time or end run time depending on **ipath** to standard outout and to the unit defined by **ieru**.

Name	Description	Type
ipath	Output begin/end message 1 - Begin 2 - End (Default)	INTEGER(KIND=4)
ieru	Output unit number for message to be written to. No message written if ieru=0	INTEGER(KIND=4)

SUBROUTINE DATE_20 (date)

Subroutine DATE_20 returns the **date** as a character string of 11 characters in the form of *dd-mmm-yyyy*.

Name	Description	Type
date	returned string	CHARACTER(LEN=*)

SUBROUTINE IDATE_20 (imonth, iday, iyear, idate)

Subroutine IDATE_20 returns the date as components and in the form of *yyyymmdd*.

Name	Description	Type
Idate	Yyyymmdd	INTEGER(KIND=4)
iday	Day of the month	INTEGER(KIND=4)
Imonth	Month of the year (January=1)	INTEGER(KIND=4)
Iyear	Year	INTEGER(KIND=4)

Command Line Routines Library

SUBROUTINE GET_COMMAND_LINE (delim, carray, npar)

Subroutine GET_COMMAND_LINE checks for command line input and parses it.

Name	Description	Type
delim	Command line delimiter	CHARACTER(LEN=1)
carray	Array of command line parameters	CHARACTER(LEN=*), DIMENSION(*)
npar	Dimension of carray	INTEGER(KIND=4)

This subroutine is machine dependent and currently supports the following compilers:

Linux/UNIX	Lahey/Fujitsu FORTRAN 95 (UNX) Uses GETCL
Microsoft Windows	HP/COMPAQ/Digital FORTRAN (DVF) Uses GETARG
OpenVMS	Digital FORTRAN 90 (VMS) Uses LIB$GET_FOREIGN

File Manipulation Subprogram

SUBROUTINE DELETE_FILE (dfile)

Subroutine DELETE_FILE is a machine independent routine to delete a file.

Example:

```
      CHARACTER(LEN=20) :: Dfile
!
      Dfile='TEST.TMP'
      Call DELETE_FILE(Dfile)
```

Note: This subroutine uses unit 69 to open and close the file to be deleted.

Sorting Routines Library

The subroutine SORT is a generalized sorting routine which is able to process several different input file formats. Immediately prior to the call to SORT, the subroutine SET_SORT_FILE (entry in SORT) should be called. The remaining subroutines in this section are required by SORT as are other NSDFLIB95 subprograms.

SUBROUTINE SORT (namin, namout, keys, ierr)

The subroutine SORT can be used to perform sorts of ASCII or binary format disk files from a FORTRAN program. The subroutine SET_SORT_FILE (Entry in SORT) must be called first to provide the characteristics of the file to be sorted. If SET_SORT_FILE is not called, the defaults are sequential formatted ASCII file with a maximum record length of 0 bytes.

Name	Description	Type
namin	Input file path and name	CHARACTER(LEN=*)
namout	Output file path and name If this string is blank, then the sorted output file will replace the input file.	CHARACTER(LEN=*)
keys	Sort keys keys(1) - Number of sort fields specified (nkeys) Repeat the following for each sort [offset=4*(n-1)] **offset+1** Sort type 1 - Character 2 - Integer 3 - Floating point **offset+2** Sort order 0 - Ascending 1 - Descending **offset+3** Starting position of the field (byte number) **offset+4** Length of sort field in bytes The total number of elements used is 4×nkeys+1.	INTEGER(KIND=4), DIMENSION(*)
ierr	Error return 0 - No error in executing the subroutine. not 0 - Error encountered during execution of the subroutine.	INTEGER(KIND=4)

ENTRY SET_SORT_FILE (iaccf, iformf, mrecf)

The entry SET_SORT_FILE informs the sort routines about the format of the file to be sorted. If SET_SORT_FILE is not called, the defaults are sequential formatted ASCII file with a maximum record length of 0 bytes.

Name	Description	Type
iaccf	Type of file access 1 - Sequential 2 - Direct (Random) - Default	INTEGER(KIND=4)
iformf	Format of file 1 - Formatted 2 - Unformatted - Default	INTEGER(KIND=4)
mrecf	Record length For a sequential file, this number is the maximum record length in bytes. For a direct access file, this is the length of each record in bytes.	INTEGER(KIND=4)

SUBROUTINE FSORT (namin, namout, iunit, iacc, iform, keys, mrec, mkey, mbuf, ierr)

The subroutine FSORT sorts the file specified by **namin**.

This is an external sort. The input file must be closed before calling SORT. The sorted data will be returned in the same file as the input, but in sorted order, if **namout** is blank, then namout will be set to namin. The file(s) will be closed on exit.

The algorithm used is Multiway Merging and Replacement Selection (see The Art of Computer Programming - Volume 3 / Sorting and Searching by Donald E. Knuth, Addison-Wesley Publishing Co., 1973).

This implementation uses a tree of losers to organize the data in a buffer array to minimize the time it takes to find the least element of the buffer to send out to the temporary file.

The merge phase uses a repeated two into two method to merge the runs down to two runs which are finally merged back into the user's file.

There are various parameters which may be varied at compile time to either adjust the performance (*i.e.*, **mbuf**, the number of records stored in main memory at any time (the number of leaves in the sort tree)) or tailor the routine for other applications.

To simplify the implementation, it is required that the sort key be the first n characters of the record (n as appropriate for the application) and that this key will be sorted in the inherent character set of the host machine as a simple string of n characters.

Name	Description	Type	Name	Description	Type
namin	Input file path and name	CHARACTER(LEN=*)	**keys**	Sort keys See keys under SORT.	INTEGER(KIND=4), DIMENSION(*)
namout	Output file path and name If this string is blank, then the sorted output file will replace the input file.	CHARACTER(LEN=*)	**mrec**	Record length. Maximum of mrec as defined in SET_SORT_FILE or mkey in SORT.	INTEGER(KIND=4)
iunit	Input unit number Defined as 70 in SORT.	INTEGER(KIND=4)	**mkey**	Length of sort keys	INTEGER(KIND=4)
iacc	Type of file access See iaccf under SET_SORT_FILE.	INTEGER(KIND=4)	**mbuf**	Incore buffer size Defined in SORT as 2×MAX0(48,nrec/20) where nrec is the number of records in the input file	INTEGER(KIND=4)
iform	Format of file See iformf under	INTEGER(KIND=4)	**ierr**	Error return 0 - No error in executing	INTEGER(KIND=4)

SET_SORT_FILE. the subroutine.

READ_SORT_IN (iunit, iacc, iform, nrio, inprec, eof)

Subroutine READ_SORT_IN reads a record from the input file.

Name	Description	Type
iunit	Input unit number Defined as 70 in SORT.	INTEGER(KIND=4)
iacc	Type of file access See iaccf under SET_SORT_FILE.	INTEGER(KIND=4)
iform	Format of file See iformf under SET_SORT_FILE.	INTEGER(KIND=4)
nrio	Record number	INTEGER(KIND=4)
inprec	Record content	CHARACTER(LEN=*)
eof	End of file	LOGICAL(KIND=4)

SUBROUTINE SORT_WRITE (filout, iunit, iacc, iform, nrio, filin, merger, iend)

Subroutine SORT_WRITE writes a record to the sort file.

Name	Description	Type
Filout	Output unit number	INTEGER(KIND=4)
Iunit	Input unit number Defined as 70 in SORT.	INTEGER(KIND=4)
Iacc	Type of file access See iaccf under SET_SORT_FILE.	INTEGER(KIND=4)
Iform	Format of file See iformf under SET_SORT_FILE.	INTEGER(KIND=4)
Nrio	Record number	INTEGER(KIND=4)
Filin	Input unit number for merge	INTEGER(KIND=4)
merger	Record to be written	CHARACTER(LEN=*)
Iend	Flag for end of filin 0 - Not end of file 1 - End of file	INTEGER(KIND=4)

SUBROUTINE SRTKEYS (keys, buffin, keystr)

The subroutine SRTKEYS returned a keyed record based on the input buffer record and the type of sort.

Name	Description	Type
keys	Sort keys See keys under SORT.	INTEGER(KIND=4), DIMENSION(*)
buffin	Internal buffer record Defined in FSORT	CHARACTER(LEN=*)
keystr	Internal buffer key record Returned to calling subprogram	CHARACTER(LEN=*)

OPEN_SORT_INPUT (namin, iunit, iacc, iform, mrec, nrec)

Subroutine OPEN_SORT_INPUT opens the sort input file and returns the number of records.

Name	Description	Type
namin	Input file path and name	CHARACTER(LEN=*)
iunit	Input unit number Defined as 70 in SORT.	INTEGER(KIND=4)
iacc	Type of file access See iaccf under SET_SORT_FILE.	INTEGER(KIND=4)
iform	Format of file See iformf under SET_SORT_FILE.	INTEGER(KIND=4)
mrec	Record length Maximum of mrec as defined in SET_SORT_FILE or mkey in SORT.	INTEGER(KIND=4)
nrec	Number of records in the input file	INTEGER(KIND=4)

Note: This subroutine contains machine dependent coding if used under OpenVMS to convert the record length for a direct access file from bytes to words (one word equals four bytes).

SUBROUTINE OPEN_SORT_OUTPUT (namout, iunit, iacc, iform, nrio, mrec)

Subroutine OPEN_SORT_OUTPUT opens the sort output file.

Name	Description	Type
namout	Output file path and name If this string is blank, then the sorted output file will replace the input file.	CHARACTER(LEN=*)
Iunit	Input unit number Defined as 70 in SORT.	INTEGER(KIND=4)
Iacc	Type of file access See iaccf under SET_SORT_FILE.	INTEGER(KIND=4)
Iform	Format of file See iformf under SET_SORT_FILE.	INTEGER(KIND=4)
Nrio	Record number	INTEGER(KIND=4)
Mrec	Record length Maximum of mrec as defined in SET_SORT_FILE or mkey in SORT.	INTEGER(KIND=4)

Note: This subroutine contains machine dependent coding if used under OpenVMS to convert the record length for a direct access file from bytes to words (one word equals four bytes).