SUN/40.6

Starlink Project
Starlink User Note 40.6

A C Charles
P C T Rees
A J Chipperfield
T Jenness
D Berry

16 May 2018

CHR
Character Handling Routines
3.0
Programmer's Manual

Abstract

This document describes the Character Handling Routine library, CHR, and its use. The CHR library augments the limited character handling facilities provided by the Fortran 77 standard. It offers a range of character handling facilities: from formatting Fortran data types into text strings and the reverse, to higher level functions such as wild card matching, string sorting, paragraph reformatting and justification. The library may be used simply for building text strings for interactive applications or as a basis for more complex text processing applications.

Contents

1 Introduction

The Fortran 77 standard provides a data type for the storage of character strings (the type `CHARACTER`), an operator specific to character data (the `//` operator for string concatenation), and eight intrinsic functions specifically for handling `CHARACTER` typed variables (`CHAR`, `ICHAR`, `INDEX`, `LEN`, `LGE`, `LGT`, `LLE`, `LLT`). Facilities to write and read Fortran data types to and from character strings (using internal files and the `WRITE` and `READ` statements respectively) are also provided by the Fortran 77 standard. Although these features of the Fortran language are of considerable utility when handling character variables in Fortran programs, they constitute only the basic tools for the more extensive processing of textual data sometimes required in applications.

The CHR library augments the limited character handling facilities provided by the Fortran 77 standard. It offers a range of character handling facilities: from formatting Fortran data types into text strings and the reverse, to higher level functions such as wild card matching, string sorting, paragraph reformatting and justification. The library may be used simply for building text strings for interactive applications or as a basis for more complex text processing applications.

The functions performed by the CHR library may be categorised as follows:

- change case
- compare strings
- decode Fortran data types
- edit strings
- encode Fortran data types
- enquire
- facilitate portability
- search strings

A classified list of of these routines is given in Appendix B.

Equivalent functions written entirely in C are available for a subset of the CHR routines (see Appendix D). These are provided mainly for use when porting Fortran application code to C.

2 Error Handling

None of the CHR routines report error messages; *i.e.* they do not use the Starlink Error Reporting System , ERR (see SUN/104). However, some routines do use the inherited status conventions (described in SUN/104) to indicate success or failure. Those CHR routines which do not have a status argument handle any errors internally and have a specified behaviour on error. Those CHR routines which do have a status argument fall into two categories: those which obey the

full inherited status conventions and return without action if given a status value other than SAI__OK, and those which ignore the given status and just return a status value on exit. The following routines obey the full inherited status conventions:

- CHR_BTOI
- CHR_CTOD
- CHR_CTOI
- CHR_CTOL
- CHR_CTOR
- CHR_FIWE
- CHR_FIWS
- CHR_HTOI
- CHR_OTOI

The routines CHR_COPY, CHR_DCWRD and CHR_TRCHR all just return a status set to a value not equal to SAI__OK on error.

The symbolic names of the error values used by CHR are given in Appendix A.

3 Compiling and Linking

The two include files available for use with the Character Handling Routines are named **sae_par** and **chr_err** on UNIX machines, and reside in the directory **/star/include**.

When including these files within Fortran code, the Starlink convention is that the name in upper case with no path or extension is specified when including these files within Fortran code, *e.g.*

```
*  Global Constants:
      INCLUDE 'SAE_PAR'
      INCLUDE 'CHR_ERR'
```

Assuming that the software has been installed in the standard way and **/star/bin** has been added to the environment variable **PATH**, soft links with these upper-case names pointing to the required file are set up in the user's working directory by the the commands:

```
% star_dev
% chr_dev
```

Then to compile and link a non-ADAM program, the command line would be, *e.g.*

```
% f77 -o program program.f -L/star/lib `chr_link`
```

The CHR library is included automatically when programs are linked using the ADAM application linking commands, **alink** *etc*.

If it is necessary to link explicitly with the ADAM version of CHR (*e.g.* to produce a shareable library), the script **chr_link_adam** is available in **/star/bin**. The link command might be:

```
% ld -shared -o libmypkg.so.1.0 -lmypkg `chr_link_adam`
```

4 Efficiency Considerations

Several routines provided by CHR have implications for the efficiency of applications which make heavy use of them. These routines when used judiciously present no efficiency problem, but when they are used indiscriminantly they can have a marked effect on execution times.

CHR_CTOx These routines make use of the Fortran internal `READ` statements which can have an impact upon execution times when used heavily. Within an application it is only absolutely necessary to perform this type conversion once, when it is needed. If both representations are needed within an application, store both.

CHR_LEN Checking the used length of a given `CHARACTER` argument within every subroutine can have a significant impact upon execution times and should be avoided. For character strings which are not modified, CHR_LEN need only be called once. The used length of the string may then be passed as an additional subroutine argument, *e.g.*

```
STRLEN = CHR_LEN( STRING )
CALL SUBN( STRING, STRLEN, STATUS )
```

or the substring that represents the filled part of the string may be passed to the subroutine, *e.g.*

```
STRLEN = CHR_LEN( STRING )
CALL SUBN( STRING( 1 : STRLEN ), STATUS )
```

CHR_*x*TOC These routines make use of the Fortran internal `WRITE` statements which can have an impact upon execution times when used heavily. Within an application it is only absolutely necessary to perform this type conversion once, when it is needed. If both representations are needed within an application, store both.

A Include Files

There are two include files used by the Character Handling Routines to define global constants during compilation. These files have the logical names SAE_PAR and CHR_ERR. The contents of each of these include files are given below.

CHR_ERR Defines the Character Handling Routine errors.

CHR__EOSNT – End of sentence.
CHR__WNOTF – Word not found.

SAE_PAR Defines the global constants SAI__OK and SAI__ERROR.

SAI__ERROR – Error encountered.
SAI__OK – No error.
SAI__WARN – Warning.

B Classified List of Routines

The classifications are in alphabetical order as follows:

Change case – Change the case of a character string.

Compare strings – Compare two character strings.

Decode Fortran data types – Convert a character string into a Fortran data type and return its value.

Editing strings – Edit character strings by replacing, adding or removing defined substrings.

Encode Fortran data types – Convert a Fortran data type into a character string representation of its value.

Enquire – Return information about a character string.

Facilitate portability – Tools for assisting portability, especially to or from non-ASCII environments.

Search strings – Search a character string.

B.1 Change case

CHR_LCASE – Convert a string to lower case.

CHR_LOWER – Return the lower case equivalent of a character.

CHR_UCASE – Convert a string to upper case.

CHR_UPPER – Return the upper-case equivalent of a character.

B.2 Compare strings

CHR_ABBRV – Return whether two strings are equal apart from case, permitting abbreviation.

CHR_SCOMP – Compare two character strings using the ASCII character set.

CHR_SIMLR – Return whether two strings are equal, apart from case.

CHR_WILD – Return whether a string matches a wild-card pattern.

B.3 Decode Fortran data types

CHR_BTOI – Read an INTEGER value from a binary string.

CHR_CTOC – Write a CHARACTER string into another string.

CHR_CTOD – Read a DOUBLE PRECISION value from a string.

CHR_CTOI – Read an INTEGER value from a string.

CHR_CTOL – Read a LOGICAL value from a string.

CHR_CTOR – Read a REAL value from a string.

CHR_HTOI – Read an INTEGER value from a hexadecimal string.

CHR_OTOI – Read an integer from an octal string.

B.4 Edit strings

CHR_APPND – Copy one string into another, ignoring trailing blanks.

CHR_CLEAN – Remove all unprintable characters from a string.

CHR_COPY – Copy one string into another, checking for truncation.

CHR_DCWRD – Split a string into its component words.

CHR_FILL – Fill a string with a given character.

CHR_LDBLK – Remove any leading blanks from a string.

CHR_LINBR – Break a line of text into a sequence of shorter lines.

CHR_PFORM – Reformat a paragraph to a new width.

CHR_PREFX – Prefix a string with a substring.

CHR_RJUST – Right-justify a string.

CHR_RMBLK – Remove all blanks from a string.

CHR_RMCHR – Remove all specified characters from a string.

CHR_SORT – Sort an array of character strings into alphabetical order.

CHR_SWAP – Swap two single-character variables.

CHR_TERM – Terminate a string by padding out with blanks.

CHR_TRCHR – Translate the specified characters in a string.

CHR_TRUNC – Truncate a string at a given delimiter.

B.5 Encode Fortran data types

CHR_DTOAN – Write a DOUBLE PRECISION value into a string as hr/deg:min:sec.

CHR_DTOC – Write a DOUBLE PRECISION value into a string.

CHR_ITOB – Write an INTEGER value as a binary string.

CHR_ITOC – Write an INTEGER value as a decimal string.

CHR_ITOH – Write an INTEGER value as a hexadecimal string.

CHR_ITOO – Write an INTEGER value as an octal string.

CHR_LTOC – Write a LOGICAL value into a string.

CHR_PUTC – Put a CHARACTER string into another at a given position.

CHR_PUTD – Put a DOUBLE PRECISION value into a string at a given position.

CHR_PUTI – Put an INTEGER value into a string at a given position.

CHR_PUTL – Put a LOGICAL value into a string at a given position.

CHR_PUTR – Put a REAL value into a string at a given position.

CHR_RTOAN – Write a REAL value into a string as hr/deg:min:sec.

CHR_RTOC – Write a REAL value into a string.

B.6 Enquire

CHR_NTH – Return the two-character abbreviation for a specified integer.

CHR_INSET – Return whether a string is a member of a given set.

CHR_ISALF – Return whether a character is alphabetic.

CHR_ISALM – Return whether a character is alphanumeric.

CHR_ISDIG – Return whether a character is a digit.

CHR_ISNAM – Return whether a string is a valid name.

CHR_LEN – Return the length of a string, ignoring trailing blanks.

B.7 Facilitate Portability

CHR_ACHR – Return the character for a given ASCII value.

CHR_ATOK – Return the character for a given ASCII character token.

CHR_ATOM – Translate a string from ASCII to the machine's character set.

CHR_ETOM – Translate a string from EBCDIC to machine's character set.

CHR_IACHR – Return the ASCII value for a given character.

CHR_MTOA – Translate a string from machine's character set to ASCII.

CHR_MTOE – Translate a string from machine's character set to EBCDIC.

B.8 Search strings

CHR_DELIM – Locate a substring using a given delimiter character.

CHR_FANDL – Find the first and last non-blank characters in a string.

CHR_FIND – Find the next occurrence of a given substring within a string.

CHR_FIWE – Find the next end-of-word within a string.

CHR_FIWS – Find the start of the next word within a string.

CHR_FPARX – Find a parenthesised expression in a character string.

CHR_LASTO – Find the last occurrence of character in a string.

CHR_SKCHR – Skip over all specified characters in a string.

CHR_TOCHR – Skip to the next specified character in a string.

C Routine Descriptions

CHR_ABBRV
Return whether two strings are equal apart from case, permitting abbreviations

Description:

Returns a logical result indicating whether two strings are the same, apart from case. In assessing this, the first string is allowed to be an abbreviation of the second string, as long as it contains a specified minimum number of characters.

Invocation:

```
RESULT = CHR_ABBRV( STR1, STR2, NCHAR )
```

Arguments:

STR1 = CHARACTER * (*) (Given)

The first string, which may be an abbreviation.

STR2 = CHARACTER * (*) (Given)

The second string.

NCHAR = INTEGER (Given)

The minimum number of characters to which the first string may be abbreviated (a smaller number will be accepted if there are actually fewer than NCHAR characters in STR2).

Returned Value:

CHR_ABBRV = LOGICAL

Whether the two strings match after allowing for case and abbreviation of the first string to no less than NCHAR characters.

CHR_ACHR
Return the character for a given ASCII value

Description:

The given ASCII value is converted to a single returned character in the machine's character set. If no such character exists within the machine's character set, the character code 0 (the ASCII NUL character) is returned.

Invocation:

```
RESULT = CHR_ACHR( ASCII )
```

Arguments:

ASCII = INTEGER (Given)

The position of the character within the ASCII character set.

Returned Value:

CHR_ACHR = CHARACTER * 1

A character value within the machine's character set.

CHR_APPND
Copy one string into another, ignoring trailing blanks

Description:

The string STR1 (or as much of it as there is room for) is copied into the part of STR2 beginning at position IPOSN+1. IPOSN is updated to indicate the final length of STR2 after this operation. Trailing blanks in STR1 are ignored.

Invocation:

```
CALL CHR_APPND( STR1, STR2, IPOSN )
```

Arguments:

STR1 = CHARACTER * (*) (Given)

The string to be copied.

STR2 = CHARACTER * (*) (Given and Returned)

The string to be updated.

IPOSN = INTEGER (Given and Returned)

The position in STR2 at which STR1 is to be appended. This value is returned updated to be the position of the last non-blank character in STR2 after the copy.

CHR_ATOK
Return the character for a given ASCII character token

Description:

The given ASCII character token is converted to a single returned character in the machine's character set. All non-printable ASCII characters are represented by their equivalent token strings. If no such ASCII character exists, the character code 0 (the ASCII NUL character) is returned. The routine is intended for the portable initialisation of unprintable characters.

Invocation:

```
RESULT = CHR_ATOK( TOKEN )
```

Arguments:

TOKEN = CHARACTER * (*) (Given)

A printable character string representing the character to be returned, e.g., 'BEL', 'BS', etc.

Returned Value:

CHR_ATOK = CHARACTER * 1

The character code within the ASCII character set.

CHR_ATOM
Translate a string from ASCII to the machine's character set

Description:

The string STR1, which has been written on a machine which uses the ASCII character set and subsequently read on another machine is returned in STR2 translated into the correct character set for that machine.

Invocation:

```
CALL CHR_ATOM( STR1, STR2 )
```

Arguments:

STR1 = CHARACTER * (*) (Given)

The character string written on a machine with an ASCII character set and read on a machine which may not use ASCII.

STR2 = CHARACTER * (*) (Returned)

The character string translated into the machine's character set. If STR2 is shorter than STR1, the translated string will be truncated; if STR2 is longer than STR1, STR2 will be padded with blanks beyond the translated string.

CHR_BTOI
Read an INTEGER value from a binary string

Description:

The given binary string is decoded into an INTEGER value.

Invocation:

```
CALL CHR_BTOI( STRING, IVALUE, STATUS )
```

Arguments:

STRING = CHARACTER * (*) (Given)

String to be decoded, e.g. '10101100'.

IVALUE = INTEGER (Returned)

Value decoded from the given string.

STATUS = INTEGER (Given and Returned)

The status value. If this value is not SAI__OK on input, the routine returns without action. If the routine fails to complete successfully, STATUS is returned set to SAI__ERROR.

Notes:

This subroutine assumes a 32-bit, twos-complement representation of an INTEGER.

CHR_CLEAN
Remove all unprintable characters from a string

Description:

Replace all unprintable characters in the given string with blanks.

Invocation:

```
CALL CHR_CLEAN( STRING )
```

Arguments:

STRING = CHARACTER * (*) (Given and Returned)

String to be cleaned.

CHR_COPY
Copy one string to another, checking for truncation

Description:

This routine copies one character string to another, checking for truncation caused by the returned string being too short to accommodate the entire given string. As much of the given string as possible is copied to the returned string, ignoring any trailing blanks. If truncation is found, it is indicated by the returned status. Optionally, the last character of the returned string may also be set to '#' if truncation occurs.

Invocation:

```
CALL CHR_COPY( STR1, TRUNC, STR2, LSTAT )
```

Arguments:

STR1 = CHARACTER * (*) (Given)
The given string.

TRUNC = LOGICAL (Given)
A logical flag indicating the action to be taken if truncation occurs: if TRUNC is .TRUE., a '#' will be written into the last element of the returned string on truncation; if TRUNC is .FALSE., no '#' is written to the returned string.

STR2 = CHARACTER * (*) (Returned)
The returned string. This will contain the given string, possibly truncated.

LSTAT = INTEGER (Returned)
The status: 0 for success, 1 if truncation occurs.

CHR_CTOC
Write a CHARACTER string into another string

Description:

Write the given character string into the returned character string. If the given string is longer than the returned string, the given string is truncated. If the returned string is longer than the given character variable, the remainder of the returned string is padded with blanks.

Invocation:

```
CALL CHR_CTOC( STR1, STR2, NCHAR )
```

Arguments:

STR1 = CHARACTER * (*) (Given)

The value to be written.

STR2 = CHARACTER * (*) (Returned)

The character string into which the value is to be written.

NCHAR = INTEGER (Returned)

The resulting length of the character string, ignoring trailing blanks.

CHR_CTOD
Read a DOUBLE PRECISION value from a string

Description:

Read a DOUBLE PRECISION value from the given character string.

Invocation:

```
CALL CHR_CTOD( STRING, DVALUE, STATUS )
```

Arguments:

STRING = CHARACTER * (*) (Given)

The string from which a DOUBLE PRECISION value is to be read.

DVALUE = DOUBLE PRECISION (Returned)

The resulting DOUBLE PRECISION value.

STATUS = INTEGER (Given and Returned)

The status value: if this value is not SAI__OK on input, the routine returns without action; if the routine does not complete successfully, STATUS is returned set to SAI__ERROR.

CHR_CTOI
Read an INTEGER value from a string

Description:

Read an INTEGER value from the given character string.

Invocation:

```
CALL CHR_CTOI( STRING, IVALUE, STATUS )
```

Arguments:

STRING = CHARACTER * (*) (Given)

The string from which an INTEGER value is to be read.

IVALUE = INTEGER (Returned)

The resulting INTEGER value.

STATUS = INTEGER (Given and Returned)

The status value. If this value is not SAI__OK on input, the routine returns without action; if the routine does not complete successfully, STATUS is returned set to SAI__ERROR.

CHR_CTOL
Read a LOGICAL value from a string

Description:

The given string is decoded as a logical value. TRUE, T, YES, Y and FALSE, F, NO, N are recognised, regardless of case. Other strings result in STATUS being set to SAI__ERROR.

Invocation:

```
CALL CHR_CTOL( STRING, LVALUE, STATUS )
```

Arguments:

STRING = CHARACTER * (*) (Given)

The string from which a LOGICAL value is to be read.

LVALUE = LOGICAL (Returned)

The resulting LOGICAL value.

STATUS = INTEGER (Given and Returned)

The status value: if this value is not SAI__OK on input, the routine returns without action; if the routine does not complete successfully, STATUS is returned set to SAI__ERROR.

CHR_CTOR
Read a REAL value from a string

Description:

Read a REAL value from the given character string.

Invocation:

```
CALL CHR_CTOR( STRING, RVALUE, STATUS )
```

Arguments:

STRING = CHARACTER * (*) (Given)

The string from which a REAL value is to be read.

RVALUE = REAL (Returned)

The resulting REAL value.

STATUS = INTEGER (Given and Returned)

The status value: if this value is not SAI__OK on input, the routine returns without action; if the routine does not complete successfully, STATUS is returned set to SAI__ERROR.

CHR_DCWRD
Split a string into its component words

Description:

All the words in the given character string are detected and returned as individual elements of a character array. In this context, a word is defined as a continuous string of non-blank characters. Hence words must be separated from each other by one or more blanks.

Invocation:

```
CALL CHR_DCWRD( STRING, MXWRD, NWRD, START, STOP, WORDS, LSTAT )
```

Arguments:

STRING = CHARACTER * (*) (Given)
The string to be split into its constituent words.

MXWRD = INTEGER (Given)
The maximum number of words that can be extracted from the given string: if there are more than MXWRD words in the string, only the first MXWRD will be returned.

NWRD = INTEGER (Returned)
The number of words located in the string.

START(MXWRD) = INTEGER (Returned)
The Ith element contains the position of the first element of the Ith word in the given string.

STOP(MXWRD) = INTEGER (Returned)
The Ith element contains the position of the last element of the Ith word in the given string.

WORDS(MXWRD) = CHARACTER * (*) (Returned)
The Ith element contains the Ith word located in the given string.

LSTAT = INTEGER (Returned)
The local status. This is a return status only: the routine is not affected by the value on input. It has the following values: SAI__OK for successful completion, SAI__ERROR if the number of words exceeds MXWRD.

CHR_DELIM
Locate a substring using a given delimiter character

Description:

The given character string is examined to see if it contains a substring delimited by the character, DELIM. The indices of the first and last characters of the substring are returned as INDEX1 and INDEX2 respectively. If no occurrence of the specified delimiter is found, or if the only occurrence is the last character of the string, then the indices are returned pointing to the whole of the input string. If only one occurrence of the delimiter is found and it is not the last character in the string, INDEX1 will point to this position and INDEX2 will point to the last character in the string. If there are more than two of the occurrences of the delimiter character, INDEX1 will point to the first occurrence and INDEX2 to the last occurrence.

Invocation:

```
CALL CHR_DELIM( STRING, DELIM, INDEX1, INDEX2 )
```

Arguments:

STRING = CHARACTER * (*) (Given)
The character string to be searched.

DELIM = CHARACTER * 1 (Given)
The substring delimiting character.

INDEX1 = INTEGER (Returned)
The position of the first occurrence of the delimiter, or the first character in the string.

INDEX2 = INTEGER (Returned)
The position of the last occurrence of the delimiter, or the last character in the string.

CHR_DTOAN
Write a DOUBLE PRECISION value into a string as hr/deg:min:sec

Description:

Format a DOUBLE PRECISION value as hours/degrees:minutes:seconds and write it into a character string. This routine is for writing angular measures into a character string in a format suitable for presentation to an astronomer.

If the absolute value of the number to be written exceeds a predefined maximum a conversion is not attempted, but the number is written as a real number in Fortran 'exponential' format and a couple of question marks are appended to its end. This prevents silly results when very large numbers are input. The variable UNITS controls the maximum permitted value for the conversion to be carried out.

The value is written into the part of the string beginning at position IPOSN+1 and IPOSN is returned updated to the position of the end of the encoded angle in STRING.

Invocation:

```
CALL CHR_DTOAN( DVALUE, UNITS, STRING, IPOSN )
```

Arguments:

DVALUE = DOUBLE PRECISION (Given)
The value to be encoded into the string. This value should represent an angular measure.

UNITS = CHARACTER * (*) (Given)
This string controls the maximum value which will be formatted as hr/deg:min:sec: if UNITS = 'HOURS', the maximum permitted value is 24.0; if UNITS = 'DEGREES', the maximum permitted is 360.0. In all other cases the maximum is 1000.0.

STRING = CHARACTER * (*) (Given and Returned)
The string into which DVALUE is written.

IPOSN = INTEGER (Given and Returned)
Given as the last element in STRING before the beginning of the encoded angle. Returned as the element in STRING corresponding to the end of the encoded angle.

CHR_DTOC
Encode a DOUBLE PRECISION value as a string

Description:

Encode a DOUBLE PRECISION value as a character string, using as concise a format as possible, and return the number of characters used. In the event of an error, '*'s are written to the string.

Invocation:

```
CALL CHR_DTOC( DVALUE, STRING, NCHAR )
```

Arguments:

DVALUE = DOUBLE PRECISION (Given)

The value to be encoded.

STRING = CHARACTER * (*) (Returned)

The string into which the value is to be encoded.

NCHAR = INTEGER (Returned)

The field width used in encoding the value.

CHR_EQUAL
Return whether two strings are equal

Description:

Determine whether the two given strings are the same, with case distinction. Their lengths must be identical after removing trailing blanks.

Invocation:

```
RESULT = CHR_EQUAL( STR1, STR2 )
```

Arguments:

STR1 = CHARACTER * (*) (Given)
The first string.

STR2 = CHARACTER * (*) (Given)
The second string.

Returned Value:

CHR_EQUAL = LOGICAL
Returned as .TRUE. if the two given strings are the same, otherwise .FALSE.

Notes:

This routine is OBSOLETE. It exists for historical reasons. Its function is better performed by a Fortran relational expression.

CHR_ETOM
Translate a string from EBCDIC to the machine's character set

Description:

The string STR1, which has been written on a machine which uses the EBCDIC character set and subsequently read on a machine which may not use the EBCDIC character set to represent characters in Fortran, is returned in STR2 translated into the correct character set for the host machine.

Invocation:

```
CALL CHR_ETOM( STR1, STR2 )
```

Arguments:

STR1 = CHARACTER * (*) (Given)

The character string written on a machine with an EBCDIC character set and read on a machine which may not use EBCDIC to represent characters in Fortran.

STR2 = CHARACTER * (*) (Returned)

The translated EBCDIC character string. If STR2 is shorter than STR1, the translated string will be truncated; if STR2 is longer than STR1, STR2 will be padded with blanks beyond the translated string.

Notes:

This subroutine has been implemenred for machines which use the ASCII character set.

CHR_FANDL
Find the first and last non-blank characters in a string

Description:

Find the indices of the first and last non-blank characters in the given string. If the string is all blank, the first index is returned set to the end of the string and the last index is returned set to zero, i.e. INDEX1 is greater than INDEX2. If the string has no length, i.e. it is a substring with the first index greater than the second, both indices are returned set to zero.

Invocation:

```
CALL CHR_FANDL( STRING, INDEX1, INDEX2 )
```

Arguments:

STRING = CHARACTER * (*) (Given)
The character string.

INDEX1 = INTEGER (Returned)
The position of first non-blank character.

INDEX2 = INTEGER (Returned)
The position of last non-blank character.

CHR_FILL
Fill a string with a given character

Description:

The given character string is filled with the specified character.

Invocation:

```
CALL CHR_FILL( CVALUE, STRING )
```

Arguments:

CVALUE = CHARACTER (Given)

The character specified to fill the string.

STRING = CHARACTER * (*) (Returned)

The string to be filled.

CHR_FIND
Find the next occurrence of given substring within a string

Description:

Increments a pointer to a character position within the given string and checks if the following sequence of characters matches the specified substring, ignoring differences in case. The search may be performed either forwards or backwards. If a match is found, the position of the substring is returned. If no match exists, the pointer is set to one more than the length of the string if the search is forwards, zero if the search is backwards.

Invocation:

```
CALL CHR_FIND( STRING, SUBSTR, FORWD, IPOSN )
```

Arguments:

STRING = CHARACTER * (*) (Given)

The string to be searched.

SUBSTR = CHARACTER * (*) (Given)

The substring to be searched for, ignoring case.

FORWD = LOGICAL (Given)

The search direction: if .TRUE., proceed through the string in a forward direction, otherwise work backwards.

IPOSN = INTEGER (Given and Returned)

The starting position for the search. If the initial value of IPOSN does not point at a character within the string, the routine returns without action.

CHR_FIWE
Find the next end-of-word within a string

Description:

Find the next end-of-word, signified by the following character being a word delimiter (SPACE, TAB or COMMA). Note that the start of the next word is not found before looking for the next word delimiter so it is possible for IPOSN to remain unchanged and indeed to point to a word delimiter rather than a true end of a word. This routine is expected to be used in conjunction with CHR_FIWS.

Invocation:

```
CALL CHR_FIWE (STRING, IPOSN, STATUS)
```

Arguments:

STRING = CHARACTER * (*) (Given)

The string to be searched.

IPOSN = INTEGER (Given and Returned)

The given value is the character position within the string at which searching is to start. If IPOSN is less than 1, the search starts at position 1. The returned value is the character position preceding the next word delimiter. If IPOSN already points to a character preceding a delimiter, it is returned unchanged. If no delimiter is found, IPOSN is returned pointing to the end of the string, and STATUS is returned set.

STATUS = INTEGER (Given and Returned)

The status value: if this value is not SAI__OK on entry, the routine returns without action; if the next word delimiter is not found before the end of the string, STATUS is returned set to CHR__EOSNT. Note: The CHR__EOSNT symbolic constant is defined in the CHR_ERR include file.

CHR_FIWS
Find the start of the next word within a string

Description:

Find the start of the next word, signified by the character not being a word delimiter, i.e. SPACE, TAB, or COMMA. Note that the end of the current word is not found before looking for the start of the next. This routine is expected to be used in conjunction with CHR_FIWE.

Invocation:

```
CALL CHR_FIWS( STRING, IPOSN, STATUS )
```

Arguments:

STRING = CHARACTER * (*) (Given)

The string to be searched.

IPOSN = INTEGER (Given and Returned)

The given value is the character position within the string at which searching is to start. If IPOSN is less than 1, the search starts at position 1. The returned value is the character position at which the next word starts. If IPOSN already points to a character within a word, it is returned unchanged. If no word is found, IPOSN is returned pointing to the end of the string, and STATUS is returned set.

STATUS = INTEGER (Given and Returned)

The status value: if this value is not SAI__OK on entry, the routine returns without action; if no word is found, STATUS is returned set to CHR__WNOTF. Note: The CHR__WNOTF symbolic constant is defined in the CHR_ERR include file.

CHR_FPARX
Find a parenthesised expression in a character string

Description:

The routine searches the string STR to identify a sub-string containing a parenthesised expression and returns the character positions of the opening and closing parentheses in the F and L arguments. Allowance is made for nested parentheses. If a parenthesised expression was not found, then the returned value of F will be greater than the returned value of L.

Invocation:

```
CALL CHR_FPARX( STR, OPPAR, CLPAR, F, L )
```

Arguments:

STR = CHARACTER * (*) (Given)
String to be searched.

OPPAR = CHARACTER * (1) (Given)
The opening parenthesis character.

CLPAR = CHARACTER * (1) (Given)
The closing parenthesis character.

F = INTEGER (Returned)
Character position of the opening parenthesis.

L = INTEGER (Returned)
Character position of the closing parenthesis.

CHR_HTOI
Read an INTEGER value from a hexadecimal string

Description:

The given hexadecimal string is decoded into an INTEGER value.

Invocation:

```
CALL CHR_HTOI( STRING, IVALUE, STATUS )
```

Arguments:

STRING = CHARACTER * (*) (Given)

String to be decoded.

IVALUE = INTEGER (Returned)

Value decoded from the given string.

STATUS = INTEGER (Given and Returned)

The status value. If this value is not SAI__OK on input, the routine returns without action. If the routine fails to complete successfully, STATUS is returned set to SAI__ERROR.

Notes:

This subroutine assumes a 32-bit, twos-complement representation of an INTEGER.

CHR_IACHR
Return the ASCII value for the given character

Description:

The given character, encoded using the machine's character set, is converted to an integer indicating its position in the ASCII character set. If no such character exists, zero is returned.

Invocation:

```
RESULT = CHR_IACHR( CVALUE )
```

Arguments:

CVALUE = CHARACTER * 1 (Given)

The character to be converted to its position within the ASCII character set.

Returned Value:

CHR_IACHR = INTEGER

An integer position within the ASCII character set.

CHR_INDEX
Return the index of a substring in a string

Description:

Find the position of a substring within a given string. If no substring is found, the value zero is returned.

Invocation:

```
RESULT = CHR_INDEX( STRING, SUBSTR )
```

Arguments:

STRING = CHARACTER * (*) (Given)
The string to be searched.

SUBSTR = CHARACTER * (*) (Given)
The substring to be used in the search.

Returned Value:

CHR_INDEX = INTEGER
The position of SUBSTR within STRING.

Notes:

This routine is OBSOLETE. It exists for historical reasons. Its function is identical to the Fortran intrinsic function INDEX. It is recommended that the INDEX intrinsic function be called directly.

CHR_INSET
Return whether a string is a member of a given set

Description:

The character string is compared with each of the values given in the given set. The strings in the set can be any length and can differ in length throughout the set. Each value is separated by a comma. Upper and lowercase are treated as being equivalent and trailing blanks are ignored.

Invocation:

```
RESULT = CHR_INSET( SET, STRING )
```

Arguments:

SET = CHARACTER * (*) (Given)

The set of character values. It takes the form 'string1,string2,.....,stringN' where each of the substring values from string1 to stringN can be of different lengths.

STRING = CHARACTER * (*) (Given)

The character string to be checked for membership of the set.

Returned Value:

CHR_INSET = LOGICAL

Returns .TRUE. if the character string is a member of the given set, returns .FALSE. otherwise.

CHR_ISALF
Return whether a character is alphabetic

Description:

The given character is tested for being alphabetic, i.e. A - Z or a - z.

Invocation:

```
RESULT = CHR_ISALF( CVALUE )
```

Arguments:

CVALUE = CHARACTER (Given)

The character to be tested.

Returned Value:

CHR_ISALF = LOGICAL

Returns .TRUE. if the given character is alphabetic, returns .FALSE. otherwise.

CHR_ISALM
Return whether a character is alphanumeric

Description:

Determine whether a character is alphanumeric, i.e. A - Z, a - z, 0 - 9 or _. Note that this routine treats the underscore character as an alphanumeric character.

Invocation:

```
RESULT = CHR_ISALM( CVALUE )
```

Arguments:

CVALUE = CHARACTER (Given)

The character to be tested.

Returned Value:

CHR_ISALM = LOGICAL

Returns .TRUE. if the given character is alphanumeric, returns .FALSE. otherwise.

CHR_ISDIG
Return whether a character is a digit

Description:

Determine whether the given character is a digit, i.e. 0 - 9.

Invocation:

```
RESULT = CHR_ISDIG( CVALUE )
```

Arguments:

CVALUE = CHARACTER (Given)

The character to be tested.

Returned Value:

CHR_ISDIG = LOGICAL

Returns .TRUE. if the given character is a digit, returns .FALSE. otherwise.

CHR_ISNAM
Return whether a string is a valid name

Description:

Determine whether the given string is a valid name: i.e. whether it starts with an alphabetic character and continues with alphanumeric or underscore characters.

Invocation:

```
RESULT = CHR_ISNAM( STRING )
```

Arguments:

STRING = CHARACTER * (*) (Given)

The string to be tested.

Returned Value:

CHR_ISNAM = LOGICAL

Returns .TRUE. if the given string is a valid name, returns .FALSE. otherwise.

CHR_ITOB
Write an INTEGER value into a binary string

Description:

Encode an INTEGER value into a binary string. The result is right-justified in the returned string. In the event of an error, '∗'s are written to the string.

Invocation:

```
CALL CHR_ITOB( IVALUE, STRING, STATUS )
```

Arguments:

IVALUE = INTEGER (Given)

Value to be encoded.

STRING = CHARACTER ∗ (∗) (Returned)

Binary string encoded from the given value.

Notes:

This subroutine assumes a 32-bit, twos-complement representation of an INTEGER.

CHR_ITOC
Encode an INTEGER value as a string

Description:

Encode an integer value as a (decimal) character string, using as concise a format as possible, and return the number of characters used. In the event of an error, '∗'s will be written into to the string.

Invocation:

```
CALL CHR_ITOC( IVALUE, STRING, NCHAR )
```

Arguments:

IVALUE = INTEGER (Given)

The value to be encoded.

STRING = CHARACTER ∗ (∗) (Returned)

The string into which the integer value is encoded.

NCHAR = INTEGER (Returned)

The field width used in encoding the value.

CHR_ITOH
Write a hexadecimal string from an INTEGER value

Description:

Encode an INTEGER value into a hexadecimal string using the machine's character set. The result is right-justified in the returned string. In the event of an error, '∗'s are written to the string.

Invocation:

```
CALL CHR_ITOH( IVALUE, STRING, STATUS )
```

Arguments:

IVALUE = INTEGER (Given)

Value to be encoded.

STRING = CHARACTER ∗ (∗) (Returned)

Hexadecimal string encoded from the given value.

STATUS = INTEGER (Given and Returned)

The status value. If this value is not SAI__OK on input, the routine returns without action. If the routine fails to complete successfully, STATUS is returned set to SAI__ERROR.

Notes:

This subroutine assumes a 32-bit, twos-complement representation of an INTEGER.

CHR_ITOO
Write an octal string from an INTEGER value

Description:

Encode an INTEGER value into an octal string using the host machine's character set. The result is right-justified in the returned string. In the event of an error, '∗'s are written to the string.

Invocation:

```
CALL CHR_ITOO( IVALUE, STRING, STATUS )
```

Arguments:

IVALUE = INTEGER (Given)

Value to be encoded.

STRING = CHARACTER ∗ (∗) (Returned)

Octal string encoded from the given value.

Notes:

This subroutine assumes a 32-bit, twos complement representation of an INTEGER.

CHR_LASTO
Locates the last occurence of CVAL in STRING

Description:

The routine locates the last occurence of the single character CVAL in STRING. If an occurence is not located then IAT is returned as 0.

Invocation:

```
CALL CHR_LASTO( STRING, CVAL, IAT )
```

Arguments:

STRING = CHARACTER * (*) (Given)
String to be searched for occurences of CVAL.

CVAL = CHARACTER * (1) (Given)
Character whose last occurence is to be located.

IAT = INTEGER (Returned)
Position within STRING at which last occurence of CVAL is located. Set to 0 if the character is not found.

CHR_LCASE
Convert a string to lowercase

Description:

The characters in the string are all converted to lowercase in situ.

Invocation:

```
CALL CHR_LCASE( STRING )
```

Arguments:

STRING = CHARACTER * (*) (Given and Returned)

The string to be converted to lowercase.

CHR_LDBLK
Remove any leading blanks from a string

Description:

Remove any leading blanks from the character string. The remaining characters are moved to the left to eliminate the resulting empty space, and the end of the string is filled with blanks.

Invocation:

```
CALL CHR_LDBLK( STRING )
```

Arguments:

STRING = CHARACTER * (*) (Given and Returned)

The string from which the leading blanks are to be removed.

CHR_LEN
Return the length of a string, ignoring trailing blanks

Description:

Find length of string, ignoring trailing blanks.

Invocation:

```
RESULT = CHR_LEN( STRING )
```

Arguments:

STRING = CHARACTER * (*) (Given)

The string whose length is to be determined.

Returned Value:

CHR_LEN = INTEGER

Returns the used length of the string.

CHR_LINBR
Break a line of text into a sequence of shorter lines

Description:

Break a long line of text into a sequence of shorter lines, making the breaks between words at spaces if possible. The maximum length of an output line is determined by the size of the character variable supplied to contain it. This routine should be called repeatedly to generate successive output lines from a single long input line. Initially, the context argument IPOSN should be set to zero; it will be updated after each call, ready to generate the next output line. A value of zero is returned for IPOSN when there are no more output lines. Any unprintable characters (e.g. tabs) are treated as if they were blanks for the purpose of identifying line-breaks.

Invocation:

```
CALL CHR_LINBR( STR1, IPOSN, STR2 )
```

Arguments:

STR1 = CHARACTER * (*) (Given)

The line of text to be broken into shorter lines. Leading blanks are ignored.

IPOSN = INTEGER (Given and Returned)

On entry, this argument specifies the character position in STR1 from which to start generating the next returned line. If a value less than 1 is given, then 1 will be used.

On exit, this argument is set to one more than the position in STR1 of the last non-blank character which appears in the returned line STR2 (i.e. the position at which generation of the next returned line should begin). If STR2 is blank because there are no more characters to process, then IPOSN is returned set to zero.

STR2 = CHARACTER * (*) (Returned)

The returned line, left justified. The length of this argument determines the maximum length of the returned line.

CHR_LOWER
Return the lowercase equivalent of a character

Description:

If the given character is uppercase, the lowercase equivalent is returned, otherwise the character will be returned unchanged.

Invocation:

```
RESULT = CHR_LOWER( CVALUE )
```

Arguments:

CVALUE = CHARACTER * 1 (Given)

The character to be converted.

Returned Value:

CHR_LOWER = CHARACTER * 1 (Returned)

Lowercase equivalent of the given character, if the given character is an uppercase letter; otherwise the character is returned unchanged.

CHR_LTOC
Encode a LOGICAL value as a string

Description:

Encode the given LOGICAL value as one of the character strings 'TRUE' or 'FALSE'.

Invocation:

```
CALL CHR_LTOC( LVALUE, STRING, NCHAR )
```

Arguments:

LVALUE = LOGICAL (Given)
The value to be encoded.

STRING = CHARACTER * (*) (Returned)
The string into which the value is to be encoded.

NCHAR = INTEGER (Returned)
The field width used in encoding the value.

CHR_MOVE
Move one string into another

Description:

The string STR1, or as much of it as there is room for, is copied into STR2 beginning at position 1.

Invocation:

```
CALL CHR_MOVE( STR1, STR2 )
```

Arguments:

STR1 = CHARACTER * (*) (Given)
The given string.

STR2 = CHARACTER * (*) (Returned)
The returned string.

Notes:

This routine is OBSOLETE. It exists for historical reasons. Its function is identical to a Fortran assignment statement. It is recommended that an assignment statement be used instead of CHR_MOVE.

CHR_MTOA
Translate a string from the machine's characters set to ASCII

Description:

The string STR1, encoded in the host machine's character set, is returned in STR2 translated into a form which can be written and subsequently read correctly by a machine which uses the ASCII character set.

Invocation:

```
CALL CHR_MTOA( STR1, STR2 )
```

Arguments:

STR1 = CHARACTER * (*) (Given)

A string represented by the host machine's character set.

STR2 = CHARACTER * (*) (Returned)

A string represented by the ASCII character set. If STR2 is shorter than STR1, the translated string will be truncated; if STR2 is longer than STR1, STR2 will be padded with blanks beyond the translated string.

CHR_MTOE
Translate a string from the machine's character set to EBCDIC

Description:

The string STR1, which is a Fortran 77 CHARACTER string, is returned in STR2 translated into a form which can be written and subsequently read correctly by a machine which uses the EBCDIC character set.

Any characters which are not represented in the EBCDIC character set are translated to EBCDIC SPACE. Non-printable characters are translated where possible.

Invocation:

```
CALL CHR_MTOE( STR1, STR2 )
```

Arguments:

STR1 = CHARACTER * (*) (Given)
The Fortran 77 character string.

STR2 = CHARACTER * (*) (Returned)
A character string which may be written and subsequently read correctly by a machine which uses the EBCDIC character set to represent characters in Fortran. If STR2 is shorter than STR1, the translated string will be truncated; if STR2 is longer than STR1, STR2 will be padded with blanks beyond the translated string.

System-specific :

This subroutine has been implemented for machines which use the ASCII character set.

CHR_NTH
Return the two-character ordinal abbreviation for a specified integer

Description:

Return the two character ordinal abbreviation (i.e. st, nd, rd, th) appropriate for the given integer value.

Invocation:

```
RESULT = CHR_NTH( IVALUE )
```

Arguments:

IVALUE = INTEGER (Given)

The integer for which the abbreviation is required.

Returned Value:

CHR_NTH = CHARACTER * 2

The appropriate two character abbreviation for the given integer value.

CHR_OTOI
Read an INTEGER value from an octal string

Description:

The given octal string is decoded into an INTEGER value.

Invocation:

```
CALL CHR_OTOI( STRING, IVALUE, STATUS )
```

Arguments:

STRING = CHARACTER * (*) (Given)

String to be decoded.

IVALUE = INTEGER (Returned)

Value decoded from the given string.

STATUS = INTEGER (Given and Returned)

The status value. If this value is not SAI__OK on input, the routine returns without action. If the routine fails to complete successfully, STATUS is returned set to SAI__ERROR.

Notes:

This subroutine assumes a 32-bit, twos-complement representation of an INTEGER.

CHR_PFORM
Reformat a paragraph to a new width

Description:

This subroutine is called repeatedly to reformat the given paragraph to a new width (given by the declared length of the returned character variable). The output may be optionally justified to the right margin (i.e. the end of the returned character variable). This routine should be called repeatedly to generate successive returned lines from the given paragraph array. Initially, the context argument IPOSN should be set to zero; it will be updated after each call, ready to generate the next output line. A value of zero is returned for IPOSN when there are no more lines to return. Any unprintable characters (e.g. tabs) are treated as if they were spaces for the purpose of generating line-breaks.

Invocation:

```
CALL CHR_PFORM( MXPAR, PARRAY, JUSTFY, IPOSN, STRING )
```

Arguments:

MXPAR = INTEGER (Given)

The maximum length of the given paragraph array, PARRAY.

PARRAY(MXPAR) = CHARACTER * (*) (Given)

The character array which contains the paragraph text to be reformatted, one line per array element. Leading blanks are ignored. A line-break is interpreted as the start of a new word.

JUSTFY = LOGICAL (Given)

The right justification flag: if this is given as .TRUE., the text is returned right justified; otherwise the text is returned with a ragged right margin.

IPOSN = INTEGER (Given and Returned)

On entry, this argument specifies the character position in PARRAY from which to start generating the next returned line. It is given as the number of characters from the start of the first character in the first element in PARRAY. If a value less than 1 is used, then 1 will be used.

On exit, this argument is set to one more than the character offset of the start of PARRAY of the last non-blank character which appears in the returned line STRING (i.e. the position at which the generation of the next output line should start). If STRING is blank because there are no more characters to process, then IPOSN is returned set to zero.

STRING = CHARACTER * (*) (Returned)

The returned line of text in the paragraph, left justified. The length of this argument defines the maximum length of the returned paragraph line.

CHR_PREFX
Prefix a string with a substring

Description:

The substring STR1 is prefixed to the string STR2, moving the string STR2 along to make room. The given string in STR2 may be truncated by adding the prefix. The final length of the string STR2, ignoring trailing blanks, is returned in LEN2.

Invocation:

```
CALL CHR_PREFX( STR1, STR2, LEN2 )
```

Arguments:

STR1 = CHARACTER * (*) (Given)

The prefix string.

STR2 = CHARACTER * (*) (Given and Returned)

The string to be prefixed.

LEN2 = INTEGER (Returned)

The resultant length of the string STR2, ignoring trailing blanks.

CHR_PUTC
Put a CHARACTER string into another at a given position

Description:

The string STR1 (or as much of it as there is room for) is copied into the part of STR2 beginning at position IPOSN+1. IPOSN is updated to indicate the end position of the copy of STR1 within STR2 after this operation. If no copying is done, IPOSN is returned unchanged. The sizes of STR1 and STR2 are based on the declared Fortran 77 size given by the intrinsic function LEN.

Invocation:

```
CALL CHR_PUTC( STR1, STR2, IPOSN )
```

Arguments:

STR1 = CHARACTER * (*) (Given)
The string to be copied.

STR2 = CHARACTER * (*) (Given and Returned)
The string into which STR1 is to be copied.

IPOSN = INTEGER (Given and Returned)
The position pointer within STR2.

CHR_PUTD
Put a DOUBLE PRECISION value into a string at a given position

Description:

The DOUBLE PRECISION value is encoded into a concise string which is then copied into the given string beginning at position IPOSN+1. IPOSN is returned updated to indicate the end position of the encoded number within STRING. This is a combination of CHR_DTOC and CHR_PUTC.

Invocation:

```
CALL CHR_PUTD( DVALUE, STRING, IPOSN )
```

Arguments:

DVALUE = DOUBLE PRECISION (Given)
The value to be encoded into the string.

STRING = CHARACTER * (*) (Given and Returned)
The string into which DVALUE is to be copied.

IPOSN = INTEGER (Given and Returned)
The position pointer within STRING.

CHR_PUTI
Put an INTEGER value into a string at a given position

Description:

The INTEGER value is encoded into a concise string which is then copied into the given string beginning at position IPOSN+1. IPOSN is returned updated to indicate the end position of the encoded number within STRING. This is a combination of CHR_ITOC and CHR_PUTC.

Invocation:

```
CALL CHR_PUTI( IVALUE, STRING, IPOSN )
```

Arguments:

IVALUE = INTEGER (Given)

The value to be encoded into the string.

STRING = CHARACTER * (*) (Given and Returned)

The string into which IVALUE is to be copied.

IPOSN = INTEGER (Given and Returned)

The position pointer within STRING.

CHR_PUTL
Put a LOGICAL value into a string at a given position

Description:

The LOGICAL value is encoded into 'T' or 'F' which is then copied into the given string beginning at position IPOSN+1. IPOSN is is returned updated to indicate the end position of the encoded logical value within STRING.

Invocation:

```
CALL CHR_PUTL( LVALUE, STRING, IPOSN )
```

Arguments:

LVALUE = LOGICAL (Given)

The LOGICAL value to be encoded into the string.

STRING = CHARACTER * (*) (Given and Returned)

The string into which LVALUE is to be copied.

IPOSN = INTEGER (Given and Returned)

The position pointer within STRING.

CHR_PUTR
Put a REAL value into a string at a given position

Description:

The REAL value is encoded into a concise string which is then copied into the given string beginning at position IPOSN+1. IPOSN is returned updated to indicate the end position of the encoded number within STRING. This is a combination of CHR_RTOC and CHR_PUTC.

Invocation:

```
CALL CHR_PUTR( RVALUE, STRING, IPOSN )
```

Arguments:

RVALUE = REAL (Given)
The value to be encoded into the string.

STRING = CHARACTER * (*) (Given and Returned)
The string into which DVALUE is to be copied.

IPOSN = INTEGER (Given and Returned)
The position pointer within STRING.

CHR_RJUST
Right-justify a string

Description:

The given string is right-justified by filling out the spaces between words with additional blank space. The right margin is taken as the declared length of the given string. Unprintable characters are interpreted as blanks.

Invocation:

```
CALL CHR_RJUST( STRING )
```

Arguments:

STRING = CHARACTER * (*) (Given and Returned)

The string to be right-justified and returned.

CHR_RMBLK
Remove all blanks from a string

Description:

All leading and embedded blanks in the string are removed. The remaining characters are moved to the left to eliminate the resulting empty space, and the end of the string is filled with blanks.

Invocation:

```
CALL CHR_RMBLK( STRING )
```

Arguments:

STRING = CHARACTER * (*) (Given and Returned)

The string from which all leading and embedded blanks are removed.

CHR_RMCHR
Remove all specified characters from a string

Description:

Remove a specified set of characters from a string in situ. The remaining characters are moved to the left to eliminate the resulting empty space, and the end of the string is filled with blanks.

Invocation:

```
CALL CHR_RMCHR( CHARS, STRING )
```

Arguments:

CHARS = CHARACTER * (*) (Given)

A string specifying all the characters which are to be removed.

STRING = CHARACTER * (*) (Given and Returned)

The string from which the characters are removed.

CHR_RTOAN
Write a REAL value into a string as hr/deg:min:sec

Description:

Format a REAL value as hours/degrees:minutes:seconds and write it into a character string. This routine is for writing angular measures into a character string in a format suitable for presentation to an astronomer.

If the absolute value of the number to be written exceeds a predefined maximum a conversion is not attempted, but the number is written as a real number in Fortran 'exponential' format and a couple of question marks are appended to its end. This prevents silly results when very large numbers are input. The variable UNITS controls the maximum permitted value for the conversion to be carried out.

The value is written into the part of the string beginning at position IPOSN+1 and IPOSN is returned updated to the position of the end of the encoded angle in STRING.

Invocation:

```
CALL CHR_RTOAN( RVALUE, UNITS, STRING, IPOSN )
```

Arguments:

RVALUE = REAL (Given)

The value to be encoded into the string. This value should represent an angular measure.

UNITS = CHARACTER * (*) (Given)

This string controls the maximum value which will be formatted as hr/deg:min:sec: if UNITS = 'HOURS', the maximum permitted value is 24.0; if UNITS = 'DEGREES', the maximum permitted is 360.0. In all other cases the maximum is 1000.0.

STRING = CHARACTER * (*) (Given and Returned)

The string into which RVALUE is written.

IPOSN = INTEGER (Given and Returned)

Given as the last element in STRING before the beginning of the encoded angle. Returned as the element in STRING corresponding to the end of the encoded angle.

CHR_RTOC
Encode a REAL value as a string

Description:

Encode a REAL value as a character string, using as concise a format as possible, and return the number of characters used. In the event of an error, '∗'s are written to the string.

Invocation:

```
CALL CHR_RTOC( RVALUE, STRING, NCHAR )
```

Arguments:

RVALUE = REAL (Given)
The value to be encoded.

STRING = CHARACTER ∗ (∗) (Returned)
The string into which the value is to be encoded.

NCHAR = INTEGER (Returned)
The field width used in encoding the value.

CHR_SCOMP
Compare two character strings using the ASCII character set

Description:

The first string is compared with the second using the ASCII character set, giving precedence to the left hand side of the string. If the first string is less than or equal to the second, the value .TRUE. is returned; otherwise the value .FALSE. is returned.

Invocation:

```
RESULT = CHR_SCOMP( STR1, STR2 )
```

Arguments:

STR1 = CHARACTER * (*) (Given)
The first character string.

STR2 = CHARACTER * (*) (Given)
The second character string.

Returned Value:

CHR_SCOMP = LOGICAL
Whether the first character string is less than or equal to the second, using the ASCII character set.

CHR_SIMLR
Return whether two strings are equal, apart from case

Description:

Determine whether two strings are the same, ignoring distinctions between upper and lowercase letters. Their lengths must be identical after removing trailing blanks.

Invocation:

```
RESULT = CHR_SIMLR( STR1, STR2 )
```

Arguments:

STR1 = CHARACTER * (*) (Given)

The first string.

STR2 = CHARACTER * (*) (Given)

The second string.

Returned Value:

CHR_SIMLR = LOGICAL

Returned as .TRUE. if the two strings are the same ignoring case distinctions; otherwise .FALSE.

CHR_SIZE
Return the declared size of a string

Description:

Give the declared size of a Fortran 77 character string variable, including trailing blanks.

Invocation:

```
RESULT = CHR_SIZE( STRING )
```

Arguments:

STRING = CHARACTER * (*) (Given)

The character string of whose length is determined.

Returned Value:

CHR_SIZE = INTEGER

Returns the declared size of the string.

Notes:

This routine is OBSOLETE. It exists for historical reasons. Its function is identical to the Fortran 77 intrinsic function LEN. It is recommended that the intrinsic function LEN be called directly.

CHR_SKCHR
Skip over all specified characters in a string

Description:

Increment a character pointer, IPOSN, either forward or backward through a string, until the character pointed to is not one of a specified set of characters. The direction of the search is given by the argument FORWD. If no such character position exists (i.e all remaining characters in the string are members of the specified set), the pointer is returned set to one more than the length of the string if the search is in the forward direction, or zero if the search is in the reverse direction. If the initial value of IPOSN does not point at one of the characters in the string, then the routine will return without action.

Invocation:

```
CALL CHR_SKCHR( CHARS, STRING, FORWD, IPOSN )
```

Arguments:

CHARS = CHARACTER * (*) (Given)
A string consisting of the set of characters to be skipped.

STRING = CHARACTER * (*) (Given)
The string to be searched.

FORWD = LOGICAL (Given)
The search direction: if .TRUE. then proceed through the string in a forward direction, otherwise work backwards through the string.

IPOSN = INTEGER (Given and Returned)
The character pointer.

CHR_SORT
Sort an array of character variables into alphabetical order

Description:

Sort an array of character variables into alphabetical order using the collating sequence provided by the routine CHR_SCOMP. After the sort, a search is made to remove any values which occur more than once. The total number of unique values is returned.

Invocation:

```
CALL CHR_SORT( CHR_SCOMP, MXARY, ARRAY, NSORT )
```

Arguments:

CHR_SCOMP = LOGICAL FUNCTION (Given)

An external function which compares two character strings and returns whether the first string is less than the second.

MXARY = INTEGER (Given)

The number of character values to sort.

ARRAY(MXARY) = CHARACTER * (*) (Given and Returned)

The array of character values to be sorted.

NSORT = INTEGER (Returned)

The number of unique character values returned.

Notes:

To use this subroutine it is necessary to declare the function CHR_SCOMP, or its equivalent, to be EXTERNAL in the calling routine.

CHR_SWAP
Swap two single-character variables

Description:

Exchange the values of two single-character variables.

Invocation:

```
CALL CHR_SWAP( CHAR1, CHAR2 )
```

Arguments:

CHAR1 = CHARACTER * 1 (Given and Returned)

The first character.

CHAR2 = CHARACTER * 1 (Given and Returned)

The second character.

CHR_TERM
Terminate a string by padding out with blanks

Description:

The given string, STRING, is terminated to a length of LENGTH characters by filling the remainder of its declared length with blanks.

Invocation:

```
CALL CHR_TERM( LENGTH, STRING )
```

Arguments:

LENGTH = INTEGER (Given)

The required length for the string: it must be positive and not greater than the declared length of the string.

STRING = CHARACTER * (*) (Given and Returned)

The string to be terminated.

CHR_TOCHR
Skip to the next specified character in a string

Description:

Increment a character pointer, IPOSN, either forward or backward through a string, until the character pointed to is one of a specified set of characters. The direction of the search is given by the argument FORWD. If no such character position exists (i.e. none of the remaining characters in the string are members of the specified set), the pointer is returned set to one more than the length of the string if the search is in the forward direction, or zero if the search is in the reverse direction. If the initial value of IPOSN does not point at one of the characters in the string, then the routine will return without action.

Invocation:

```
CALL CHR_TOCHR( CHARS, STRING, FORWD, IPOSN )
```

Arguments:

CHARS = CHARACTER * (*) (Given)
A string consisting of the set of characters to be searched for.

STRING = CHARACTER * (*) (Given)
The string to be searched.

FORWD = LOGICAL (Given)
The search direction: if .TRUE. then proceed through the string in a forward direction, otherwise work backwards through the string.

IPOSN = INTEGER (Given and Returned)
The character pointer.

CHR_TRCHR
Translate the specified characters in a string

Description:

Translate a specified set of characters within a string. The character translation is controlled by the translation table given by the character strings FROM and TO. Any characters not appearing in the translation table are left unchanged. If the status is set on entry, no action is taken. If the strings FROM and TO are unequal in length, STATUS is returned set to SAI__ERROR.

Invocation:

```
CALL CHR_TRCHR( FROM, TO, STRING, STATUS )
```

Arguments:

FROM = CHARACTER * (*) (Given)

A string specifying the characters to be translated.

TO = CHARACTER * (*) (Given)

A string specifying the translation values for each of the characters in the FROM argument. The lengths of the FROM and TO arguments must be the same.

STRING = CHARACTER * (*) (Given and Returned)

The string to be translated. Any character matching one of the characters specified in the FROM argument is converted to the corresponding character specified in the TO argument. All other characters are left unchanged.

STATUS = INTEGER (Given and Returned)

The global status: returned set to SAI__ERROR if FROM and TO have unequal lengths.

CHR_TRUNC
Truncate a string at a given delimiter

Description:

The given string is truncated at the first occurrence of the given delimiter character. The delimiter character and all subsequent characters are replaced by blanks. If no delimiter character is found in the string, no truncation takes place. This routine is effectively a combination of INDEX and CHR_TERM.

Invocation:

```
CALL CHR_TRUNC( DELIM, STRING )
```

Arguments:

DELIM = CHARACTER * 1 (Given)

The truncation delimiter character.

STRING = CHARACTER * (*) (Given and Returned)

The string to be truncated. All characters from, and including, the first occurrence of DELIM will be replaced with blanks.

CHR_UCASE
Convert a string to uppercase

Description:

The characters in the string are all converted to uppercase in situ.

Invocation:

```
CALL CHR_UCASE( STRING )
```

Arguments:

STRING = CHARACTER * (*) (Given and Returned)

The string to be converted to uppercase.

CHR_UPPER
Return uppercase equivalent of a character

Description:

If the given character is lowercase, the uppercase equivalent is returned, otherwise the character will be returned unchanged.

Invocation:

```
RESULT = CHR_UPPER( CVALUE )
```

Arguments:

CVALUE = CHARACTER * 1 (Given)

The character to be converted.

Returned Value:

CHR_UPPER = CHARACTER * 1

Uppercase equivalent of the given character, if the given character is a lowercase letter; otherwise the character is returned unchanged.

CHR_WILD
Return whether a string matches a wild-card pattern

Description:

A candidate string is matched with a another character string containing a pattern of characters and wild-card characters. The wild-cards used are:

% a single character wild-card; ∗ an arbitrary length string wild-card, including zero length.

There is also a literal escape character '\' for use when the characters '∗' and '%' are to be interpreted literally within the wild-card pattern.

Invocation:

```
RESULT = CHR_WILD( STRING, WILDS, MATCH )
```

Arguments:

STRING = CHARACTER ∗ (∗) (Given)
The candidate string to be matched.

WILDS = CHARACTER ∗ (∗) (Given)
The wild-card pattern to be used in the match.

MATCH = CHARACTER ∗ (∗) (Returned)
The wild-card match: this string must be the same length as STRING. All characters matched individually are returned as blanks in MATCH, and all characters matched by wild-cards are returned assigned to the particular wild-cards they matched. If the length of MATCH is less than that of STRING, then CHR_WILD returns the value .FALSE.

Returned Value:

CHR_WILD = LOGICAL
Whether the two strings match after expanding the wild-card pattern.

D C Function Descriptions

D.1 Overview

Equivalent functions written entirely in C are available for the following subset of the CHR routines. These are provided mainly for use when porting Fortran application code to C.

The C and Fortran implementations are completely independent of each other - no “wrapper” layer is involved. This is done to avoid efficiency problems caused by the continual copying of strings between Fortran and C that would be required if the C interface was provided by a wrapper layer on top of the Fortran coide (or vice versa). The behaviour of C functions is slightly different to that of the Fortran routines because the two languages index strings in different ways (C is zero-based, Fortran is one-based). Also C has no equivalent to the Fortran intrinsic function LEN, and so each C function that returns a string has an extra argument specifying the length of the buffer reserved for the output string.

chrAppnd
Copy one string into another, ignoring trailing blanks

Description:

The string "str1" (or as much of it as there is room for) is copied into the part of "str2" beginning at position "iposn". "iposn" is updated to indicate the final length of "str2" after this operation. Trailing blanks in "str1" are ignored.

Invocation:

```
void chrAppnd( const char *str1, char *str2, size_t str2_length, size_t *iposn
)
```

Notes:

If the output string is too small to append the entire input string, truncation will occur and "iposn" will be set to the total length of "str2" (excluding the trailing null).

Parameters

str1

Pointer to a null terminated string holding the string to be copied.

str2

Pointer to a null terminated string holding the string to be updated.

str2_length

The maximum length of the 'str2' string. This should include room for the terminating null.

***iposn**

The used length of the "str2". If "str2" is empty on entry, then this should be supplied as zero. On exit, the supplied value is incremented by the length of "str1" (ignoring any trailing spaces).

chrClean
Remove all unprintable characters from a string

Description:

Replace all unprintable characters in the given string with blanks.

Invocation:

```
void chrClean( char *string )
```

Parameters

string

Pointer to a null terminated string holding the string to be cleaned.

chrCtod
Read a double value from a string

Description:

Read a double value from the given character string.

Invocation:

```
void chrCtod( const char *string, double *dvalue, int *status )
```

Parameters

string

Pointer to a null terminated string holding the string from which a double value is to be read.

***dvalue**

Returned holding the resulting double value.

***status**

The status value: if this value is not SAI__OK on input, the function returns without action; if the function does not complete successfully, "status" is returned set to SAI__ERROR.

chrCtoi
Read an integer value from a string

Description:

Read an integer value from the given character string.

Invocation:

```
void chrCtoi( const char *string, int *ivalue, int *status )
```

Parameters

string

Pointer to a null terminated string holding the string from which an integer value is to be read.

***ivalue**

Returned holding the resulting integer value.

***status**

The status value. If this value is not SAI__OK on input, the function returns without action; if the function does not complete successfully, "status" is returned set to SAI__ERROR.

chrCtor
Read a float value from a string

Description:

Read a float value from the given character string.

Invocation:

```
void chrCtor( const char *string, float *rvalue, int *status )
```

Parameters

string

Pointer to a null terminated string holding the string from which a float value is to be read.

***rvalue**

Returned holding the resulting float value.

***status**

The status value: if this value is not SAI__OK on input, the function returns without action; if the function does not complete successfully, "status" is returned set to SAI__ERROR.

chrFandl
Find the first and last non-blank characters in a string

Description:

Find the indices of the first and last non-blank characters in the given string. If the string contains no non-blank characters, the first index is returned set to 1 and last index is returned set to 0, i.e. "index1" is greater than "index2".

Invocation:

```
void chrFandl( const char *string, size_t *index1, size_t *index2 )
```

Notes:

- For consistency with the Fortran routine CHR_FANDL, this function only checks for spaces. Other forms of whitespace characters such as tabs, line-feeds, etc are considered to be non-blank.

Parameters

string
Pointer to a null terminated string holding the character string.

***index1**
Returned holding the zero-based position of first non-blank character.

***index2**
Returned holding the zero-based position of last non-blank character.

chrFill
Fill a string with a given character

Description:

The given character string is filled with the specified character.

Invocation:

```
void chrFill( char cvalue, char *string, size_t string_length )
```

Parameters

cvalue

The character specified to fill the string.

string

Pointer to an array in which to return a null terminated string holding the string to be filled.

string_length

The declared length of the supplied 'string' array. This should include room for the terminating null.

chrFparx
Find a parenthesised expression in a character string

Description:

This function searches the string "str" to identify a sub-string containing a parenthesised expression and returns the character positions of the opening and closing parentheses in the "f" and "l" arguments. Allowance is made for nested parentheses. If a parenthesised expression was not found, then the returned value of "f" will be greater than the returned value of "l".

Invocation:

```
void chrFparx( const char *str, char oppar, char clpar, size_t *f, size_t *l
)
```

Parameters

str Pointer to a null terminated string holding the string to be searched.

oppar
The opening parenthesis character.

clpar
The closing parenthesis character.

***f** Returned holding the zero-based character position of the opening parenthesis.

***l** Returned holding the zero-based character position of the closing parenthesis.

chrIsalm
Return whether a character is alphanumeric

Description:

Determine whether a character is alphanumeric, i.e. A - Z, a - z, 0 - 9 or _. Note that this function treats the underscore character as an alphanumeric character.

Invocation:

```
int chrIsalm( char cvalue )
```

Returned Value:

Returns non-zero if the given character is alphanumeric,

Parameters

cvalue

The character to be tested.

chrIsnam
Return whether a string is a valid name

Description:

Determine whether the given string is a valid name: i.e. whether it starts with an alphabetic character and continues with alphanumeric or underscore characters.

Invocation:

```
int chrIsnam( const char *string )
```

Returned Value:

Returns non-zero if the given string is a valid name, returns

Parameters

string

Pointer to a null terminated string holding the string to be tested.

chrItoc
Encode an integer value as a string

Description:

Encode an integer value as a (decimal) character string, using as concise a format as possible, and return the number of characters used. In the event of an error, "∗' s will be written into to the string.

Invocation:

```
void chrItoc( int ivalue, char *string, size_t string_length, size_t *nchar )
```

Parameters

ivalue

The value to be encoded.

string

Pointer to an array in which to return a null terminated string holding the string into which the integer value is encoded.

string_length

The maximum length of the supplied ' string' array. This should include room for the terminating null.

∗**nchar**

Returned holding the field width used in encoding the value.

chrLdblk
Remove any leading blanks from a string

Description:

Remove any leading blanks from the character string. The remaining characters are moved to the left to eliminate the resulting empty space, and a terminating null is appended to the end.

Invocation:

```
void chrLdblk( char *string )
```

Parameters

string

Pointer to a null terminated string holding the string from which the leading blanks are to be removed.

chrLen
Return the length of a string, ignoring trailing spaces

Description:

Find length of string, ignoring trailing spaces.

Invocation:

```
size_t chrLen( const char *string )
```

Returned Value:

Returns the length of the string, not including the

terminating null or any trailing spaces.

Parameters

string

Pointer to a null terminated string holding the string whose length is to be determined.

chrPutc
Put a character string into another at a given position

Description:

The string "str1" (or as much of it as there is room for) is copied into the part of "str2" beginning at position "iposn+1". "iposn" is updated to indicate the end position of the copy of "str1" within "str2" after this operation. If no copying is done, "iposn" is returned unchanged.

Invocation:

```
void chrPutc( const char *str1, char *str2, size_t str2_length, size_t *iposn
)
```

Parameters

str1

Pointer to a null terminated string holding the string to be copied.

str2

Pointer to a null terminated string holding the string into which "str1" is to be copied.

str2_length

The declared length of the supplied ' str2' array. This should include room for the terminating null.

***iposn**

The zero-based position pointer within "str2".

chrPuti
Put an integer value into a string at a given position

Description:

The integer value is encoded into a concise string which is then copied into the given string beginning at position "iposn+1". "iposn" is returned updated to indicate the end position of the encoded number within "string". This is a combination of chrItoc and chrPutc.

Invocation:

```
void chrPuti( int ivalue, char *string, size_t string_length, size_t *iposn )
```

Parameters

ivalue

The value to be encoded into the string.

string

Pointer to a null terminated string holding the string into which "ivalue" is to be copied.

string_length

The declared length of the supplied ' string' array. This should include room for the terminating null.

***iposn**

The zero-based position pointer within "string".

chrRmblk
Remove all blanks from a string

Description:

All leading and embedded blanks in the string are removed. The remaining characters are moved to the left to eliminate the resulting empty space, and a terminating null is appended to the end of the string.

Invocation:

```
void chrRmblk( char *string )
```

Parameters

string

Pointer to a null terminated string holding the string from which all leading and embedded blanks are removed.

chrSimlr
Return whether two strings are equal, apart from case

Description:

Determine whether two strings are the same, ignoring distinctions between upper and lowercase letters. Their lengths must be identical after removing trailing blanks.

Invocation:

```
int chrSimlr( const char *str1, const char *str2 )
```

Returned Value:

Returned as non-zero if the two strings are the same

Parameters

str1

Pointer to a null terminated string holding the first string.

str2

Pointer to a null terminated string holding the second string.

chrSimlrN
Return whether the starts of two strings are equal, apart from case

Description:

Determine whether the first " n" characters of two strings are the same, ignoring distinctions between upper and lowercase letters.

Invocation:

```
int chrSimlrN( const char *str1, const char *str2, size_t n )
```

Returned Value:

Returned as non-zero if the two strings are the same

Parameters

str1
Pointer to a null terminated string holding the first string.

str2
Pointer to a null terminated string holding the second string.

n The number of characters that must match at the start of each string. If the length of either string is less than " n" , zero will be returned.

chrSizetoc
Encode a size_t value as a string

Description:

Encode a size_t value as a (decimal) character string, using as concise a format as possible, and return the number of characters used. In the event of an error, " ∗' s will be written into to the string.

Invocation:

```
void chrSizetoc( size_t value, char *string, size_t string_length, size_t *nchar
)
```

Parameters

value

The value to be encoded.

string

Pointer to an array in which to return a null terminated string holding the string into which the integer value is encoded.

string_length

The maximum length of the supplied ' string' array. This should include room for the terminating null.

∗**nchar**

Returned holding the field width used in encoding the value.

chrUcase
Convert a string to uppercase

Description:

The characters in the string are all converted to uppercase in situ.

Invocation:

```
void chrUcase( char *string )
```

Parameters

string

Pointer to a null terminated string holding the string to be converted to uppercase.

E Portability

E.1 Overview

This section discusses the portability of CHR, including the coding standard adopted for CHR and a list of those routines which may need to be modified when porting CHR to a new target machine.

E.2 Coding and porting prerequisites

The standard of Fortran used for the coding of CHR is fundamentally Fortran 77, using the Starlink Fortran coding conventions described in SGP/16. Several common extensions to the Fortran 77 standard are used in source code for CHR, they are as follows:

- End-of-line comments using the "!" symbol;
- Symbolic subprogram names may be longer than six characters (but are always shorter than ten characters);
- Symbolic subprogram names include the "_" symbol;
- Symbolic constant names may be longer than six characters (but are always shorter than eleven characters);
- Symbolic constant names may include the "_" symbol;
- The full ASCII character set is assumed in character constants.

The CHR library currently has no dependence upon any other package. To use CHR on any computer system porting effort is required only for those CHR routines which have operating system dependencies: these routines are listed in the following section.

E.3 Operating system specific routines

Several CHR routines make use of features specific to the operating system or language implementation. The names of these routines and their purpose are as follows:

CHR_ACHR – Return the character for a given ASCII value.

CHR_ATOK – Return the character for a given ASCII character token.

CHR_ATOM – Translate a string from ASCII to machine's character set.

CHR_BTOI – Read an INTEGER value from a binary string.

CHR_HTOI – Read an INTEGER value from a hexadecimal string.

CHR_IACHR – Return the ASCII value for a given character.

CHR_ITOB – Encode an INTEGER value as a binary string.

CHR_ITOH – Encode an INTEGER value as a hexadecimal string.

CHR_ITOO – Encode an INTEGER value as an octal string.

CHR_LOWER – Return the lower case equivalent of a character.

CHR_MTOA – Translate a string from machine's character set to ASCII.

CHR_OTOI – Read an integer from an octal string.

CHR_UPPER – Return the upper-case equivalent of a character.

These routines may have to be rewritten specifically for each new operating system and Fortran implementation.

F Changes and New Features in Version 2.0

F.1 Obsolete routines

Several routines exist in CHR purely for historical reasons. These routines are obsolete and are only provided by the CHR library because existing Starlink code may depend upon them. They should not be used in any new software.

CHR_EQUAL – Return whether two strings are equal.

CHR_INDEX – Return the index of a substring within a string.

CHR_MOVE – Move one string into another.

CHR_SIZE – Return the declared length of a string.

F.2 Changes in behaviour of existing routines

CHR_DTOC – The former algorithm has been replaced – this may result in the returned string being different from that returned from the previous version.

CHR_RTOC – The former algorithm has been replaced – this may result in the returned string being different from that returned from the previous version.

F.3 New routines

CHR_ABBRV – Return whether two strings are equal apart from case, permitting abbreviation.

CHR_ACHR – Return the character for a given ASCII value.

CHR_ATOK – Return the character for a given ASCII character token.

CHR_ATOM – Translate a string from ASCII to machine's characters set.

CHR_BTOI – Read an INTEGER value from a binary string.

CHR_DTOAN – Write a DOUBLE PRECISION value into a string as hr/deg:min:sec.

CHR_FIND – Find the next occurrence of a given substring within a string.

CHR_IACHR – Return the ASCII value for a given character.

CHR_ITOB – Encode an INTEGER value as a binary string.

CHR_ITOH – Encode an INTEGER value as a hexadecimal string.

CHR_ITOO – Encode an INTEGER value as an octal string.

CHR_LINBR – Break a line of text into a sequence of shorter lines.

CHR_MTOA – Translate a string from machine's characters set to ASCII.

CHR_NTH – Return the two-character abbreviation for a specified integer.

CHR_PFORM – Reformat a paragraph to a new width.

CHR_PREFX – Prefix a string with a substring.

CHR_RJUST – Right-justify a string.

CHR_RMCHR – Remove all specified characters from a string.

CHR_SCOMP – Compare two character strings using the ASCII character set.

CHR_SKCHR – Skip over all specified characters in a string.

CHR_SORT – Sort an array of character strings into alphabetical order.

CHR_TOCHR – Skip to the next specified character in a string.

CHR_TRCHR – Translate the specified characters in a string.

CHR_WILD – Return whether a string matches a wild-card pattern.

F.4 Other changes

CHR_DIR:LOGICAL.COM This file has been renamed CHR_DIR:CHR_DEV.COM.

CHR_DIR:CHRLINK.OPT This file has been renamed CHR_DIR:CHR_LINK.OPT.

G Changes and New Features in Version 2.2

G.1 Changes in behaviour of existing routines

The algorithm used in CHR_RTOC and CHR_DTOC has been changed – this may result in the returned string being different from that returned from the previous versions.

G.2 Documentation Changes

This document has been modified to remove references to the use of CHR on VMS.

H Changes and New Features in Version 3.0

C equivalents for a subset of the Fortran routines have been added to the CHR library to assist in porting Fortran code to C.