SUN/121.6

Starlink Project
Starlink User Note 121.6

P M Allan
A J Chipperfield
T Jenness
David S Berry

26th September 2019

PSX
POSIX interface routines
Version 0.6-0
Programmer's Manual

Abstract

PSX is a FORTRAN subroutine library that allows programmers to use the functionality provided by the POSIX and X/OPEN libraries. The use of this library will enable programmers to make use of operating system facilities in a machine independent way.

Contents

1 Introduction

1.1 Who should read this document?

It is frequently the case that a FORTRAN programmer will need to use facilities that the operating system of the host computer provides and which pure FORTRAN 77 does not. This document is intended for programmers who need to use such facilities, and to do so in a portable way.

1.2 Overview

When writing programs that interact with the real world, as opposed to those that do some purely mathematical calculation, it is often necessary to make use of features of the operating system. A simple example of this is getting the current date and time. Most FORTRAN systems provide subroutines to let the programmer get this information, but the syntax of these routines differ from one computer to another, making the resulting code non portable. What is required is a portable operating system interface, and that is exactly what POSIX is. It is a set of routines that let an application program interact with the host operating system of the computer in a standard manner.

POSIX will eventually define a whole set of standards, but the one of interest here is that known as IEEE 1003.1-1988. This standard is published by the Institute of Electrical and Electronic Engineers (IEEE) and is recognized by the American National Standards Institute (ANSI). To quote from the standard [1], 'It defines a standard operating system interface and environment to support application portability at the source code level.' What this means is that if you include a POSIX call in your program, then your program will work on any system that is POSIX compliant. Since POSIX has its origins in Unix, it is not surprising that several of the many different Unix systems are already fully or nearly, POSIX compliant.

On account of the Unix origin of POSIX, some of the terminology used in describing the routines has a Unix accent. Any confusion caused by this should be resolved by the notes for each routine.

1.3 What exactly *is* POSIX?

If you really mean that, you had better go and read the standard[1]. However, to a reasonable approximation, POSIX (1003.1) is just the C run time library and indeed the POSIX standard makes reference to the ANSI C standard [2]. Unfortunately, different computers have somewhat different C run time libraries. The point of POSIX is to define a common run time library that will be available on all computers.

2 The PSX library

The 1003.1 standard actually defines a C interface to POSIX. This is the obvious thing to do since POSIX is derived from the C run time library. However, that is not much comfort if you actually want to call the POSIX routines from FORTRAN. There is a draft standard (1003.9) that defines how the routines should be called from FORTRAN (referred to as the FORTRAN binding), but that is still a draft and commercial products based on that standard are unlikely to be available for some while. Furthermore, in order to cope with the C concepts that are inherent in POSIX, the draft standard for the FORTRAN binding has defined things in such a way that it makes using POSIX directly from FORTRAN somewhat painful.

To make life easier for the FORTRAN programmer, a package of routines called PSX has been written to which let FORTRAN programs make calls to POSIX functions in a manner consistent with other Starlink subroutine libraries. This means that FORTRAN programs that need to make use of functions provided by the operating system can be written in a portable manner. Another advantage of using the PSX routines instead of using raw POSIX (even if we could) is that it allows us to use inherited status that is a common feature of Starlink subroutine libraries. If a PSX routine detects an error, it sets the STATUS argument of the routine to one of the values given in appendix B and reports an error via EMS (see SSN/4).

Note that the names of the PSX routines are often longer than the Starlink recommendation (SGP/16) of the PSX_ prefix plus five more characters. This has been done so that the name of the PSX routine corresponds directly with the name of the 'real' POSIX routine. This creates a potential problem with porting the routines to other computers that might not accept names longer than six characters. However, the draft standard for the FORTRAN binding to POSIX assumes that compilers will accept names up to 31 characters (this is the *only*, extension to ANSI standard FORTRAN 77 that it assumes), so this is no worse than basing routines directly on the specification of the draft standard.

The PSX routines are just wrap around routines for POSIX routines. For further details on these routines, you should consult the relevant standards documents [1] and [2]. If you do not have copies of these, the documentation for the C run time library on your computer may be helpful.

3 The Level of the Implementation

So far, not all of the POSIX 1003.1 routines have been provided with a PSX equivalent. Indeed this may never be achieved. The purpose of the PSX routines is to enable the programmer to use functionality of POSIX, so only those routines that are actually thought to be needed will be provided. If you have a need for a routine that has not been provided, please mail the author, who will do his best to provide the routine. Each routine is fairly simple to write; it is only the sheer number of routines that prevents a complete set being provided to date.

The POSIX 1003.1 standard refers to the ANSI C standard for the description of some of the routines. These are listed in the POSIX standard under 'language specific services for the C programming language'. There are no corresponding routines in the draft FORTRAN binding. Nevertheless, some of these routines are so useful that a PSX implementation of them has been provided. Examples are allocating virtual memory (using malloc) and getting the current date and time.

Descriptions of the routines that are currently available in the PSX library are given in appendix E. Those routines that have been considered for inclusion in the library and have been rejected are listed in appendix F.

3.1 Future Extensions

While the PSX routines are clearly very useful as they stand, they do not always present the information in the manner that you may want. An example of this is the subroutine PSX_CTIME, which returns the data and time in a particularly perverse format. The temptation to 'improve' some routines has been resisted to ensure that there is consistency between the PSX routines and the corresponding true POSIX or C run-time library routines. However, there is clearly a need for routines that pull together some of the PSX routines in a more user friendly way and to provide information in a different manner. These routines will be provided in a separate package.

4 X/OPEN

Another attempt to achieve portable programs is that of the X/OPEN group. The X/OPEN portability guide is a similar document to the POSIX standard and is also based on the C run time library. The routines defined in the X/OPEN portability guide tend to be of a higher level than those in the POSIX standard and where appropriate, PSX equivalents for X/OPEN routines will also be provided.

5 Compiling and Linking

To use the PSX routines, you first need to 'log on' for development by using the `psx_dev` command to create the necessary soft links in your directory so that you can use the short form of the include file names.

```
% psx_dev
```

If a FORTRAN program wishes to check for a particular error status returned by a PSX routine, then it should contain the line

```
INCLUDE 'PSX_ERR'
```

The use of upper-case in the file name is important. This will define the symbolic constants listed in appendix B.

To compile and link a program that uses the PSX library, type

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

6 References

References

[1] IEEE Standard Portable Operating System Interface for Computer Environments (IEEE Std 1003.1-1988). Publ, Institute of Electrical and Electronic Engineers, Inc.

[2] American National Standard for Information Systems – Programming Language – C (ANSI X3.159-1989). Publ, American National Standards Institute.

[3] Portable Operating System Interface for Computer Environments, FORTRAN 77 Bindings (P1003.9 / Draft 5.0).

1.2, 1.3, 2

1.3, 2

A Examples

So much for theory, here are some examples of the use of PSX routines. Each POSIX routine tends to stand on its own, so the examples are fairly simple.

A.1 Create a file with name that is specific to a user

The requirement is to create a file that will be used to hold the output of several programs, but this file must be in a directory that is used by several people. Clearly the file name must be related to the username. Also it is necessary to take into account the difference in the syntax of directory names on VMS and Unix systems.

```
      PROGRAM NEWFIL

      IMPLICIT NONE
      INCLUDE 'SAE_PAR'
      INTEGER STATUS                  ! The global status value
      CHARACTER * ( 32 ) NAME         ! The name of the current user
      CHARACTER * ( 80 ) FILNAM       ! The name of the file to be created
      CHARACTER * ( 15 ) SYSNAME      ! The name of the operating system
      CHARACTER * ( 1 ) DUMMY1
      CHARACTER * ( 1 ) DUMMY2
      CHARACTER * ( 1 ) DUMMY3
      CHARACTER * ( 1 ) DUMMY4

* Set STATUS since this is is not an ADAM program.
      STATUS = SAI__OK

* Get the username.
      CALL PSX_GETENV( 'USER', NAME, STATUS )

* Get the system name.
      CALL PSX_UNAME( SYSNAME, DUMMY1, DUMMY2, DUMMY3, DUMMY4, STATUS )

* Create the file.
      IF( STATUS .EQ. SAI__OK ) THEN
         IF( SYSNAME .EQ. 'VMS' ) THEN
            FILNAM = 'COMMON_AREA:' // NAME // '.DAT'
         ELSE
            FILNAM = '/usr/common/' // NAME // '.DAT'
         END IF
         OPEN( UNIT=1, FILE=FILNAM, STATUS='NEW' )
         CLOSE( UNIT=1 )
      ELSE
         PRINT *,'Could not get username'
      END IF

      END
```

Although the PSX routines are designed to be used with other Starlink routines in the ADAM environment and use the concept of inherited status, they can just as easily be used in a stand alone program like the one above provided that the status is set correctly before calling the first routine. This is important since the PSX routine will exit immediately if STATUS is not set to the value of the symbolic constant SAI__OK.

A.2 Get some virtual memory

One of the annoying features of FORTRAN 77 is that all storage space must be allocated at compile time, *i.e.* there are no dynamic arrays. Here is an example of using PSX routines to dynamically allocate an array.

```
      PROGRAM MAIN
      IMPLICIT NONE
      INCLUDE 'SAE_PAR'
      INTEGER STATUS

* Set the STATUS to OK.
      STATUS = SAI__OK

* Call the subroutine.
      CALL GETVM( STATUS )

      END


      SUBROUTINE GETVM( STATUS )
      IMPLICIT NONE
      INCLUDE 'SAE_PAR'
      INCLUDE 'CNF_PAR'
      INTEGER STATUS, PNTR

* Check global status.
      IF( STATUS .NE. SAI__OK ) RETURN

* Create a ten element integer array and return a pointer to it.
      CALL PSX_CALLOC( 10, '_INTEGER', PNTR, STATUS )

* If all is well, operate on the array.
      IF (STATUS .EQ. SAI__OK) THEN
         CALL FILL( %VAL(PNTR), 10 )
         CALL PRNT( %VAL(PNTR), 10 )
         CALL PSX_FREE( PNTR, STATUS )
      END IF

      END


      SUBROUTINE FILL( ARRAY, N )
* Put some numbers in the array.
      INTEGER N, ARRAY( N )
      INTEGER I
      DO I = 1, N
         ARRAY( I ) = I
      END DO

      END


      SUBROUTINE PRNT( ARRAY, N )
* Print the elements of ARRAY.
      INTEGER N, ARRAY( N )
      INTEGER I

      DO I = 1, N
         PRINT *,ARRAY( I )
      END DO

      END
```

In this case the main program merely sets the value of STATUS and calls the subroutine GETVM to do the work. Although this is more typing, it does have the advantage that this could be made into an ADAM task simply by deleting the main program. GETVM tests that the value of STATUS returned from PSX_CALLOC is OK, but does not print any error message. This is not necessary, as the PSX routines all report their own errors via the EMS routines. The reporting of error messages may be deferred if required, as described in SUN/104 and SSN/4.

If the code is required to work where pointers may be longer than INTEGERs, The construct `%VAL(CNF_PVAL(PNTR))`, rather than simply `%VAL(PNTR)`, should be used in passing the pointer to FILL and PRNT. Function CNF_PVAL is defined in the CNF_PAR include file and described in SUN/209 (section 'Pointers').

B Include Files

The symbolic constants that define the error codes returned by PSX routines are defined in the file `/star/include/psx_err`. The meaning of these constants is given below:

PSX__INTYP	Invalid argument TYPE given in call to PSX_CALLOC
PSX__NOALL	Null pointer returned on memory allocation
PSX__NOENV	No translation of an environment variable
PSX__NOGMT	Could not get GMT with gmtime()
PSX__NOTIM	Could not get current time with time()
PSX__NOMEM	Could not get required memory
PSX__ERRNO	System error during POSIX call

C Alphabetical list of routines

PSX_ACCESS (NAME, MODE, ACCESSIBLE, REASON, STATUS)
Check file accessibility

PSX_ASCTIME (TSTRCT, STRING, STATUS)
Convert a time structure to a character string

PSX_CALLOC (NMEMB, TYPE, PNTR, STATUS)
Allocate space for several objects of specified type

PSX_CHDIR (DIR, STATUS)
Change working directory

PSX_CTIME (NTICKS, STRING, STATUS)
Convert the calendar time to a character string

PSX_CUSERID (USER, STATUS)
Get the username

PSX_FREE (PNTR, STATUS)
Free virtual memory

PSX_GETCWD (CWD, STATUS)
Get the current working directory

PSX_GETEGID (GID, STATUS)
Gets the effective group ID

PSX_GETENV (NAME, TRANS, STATUS)
Translate an environment variable

PSX_GETEUID (UID, STATUS)
Gets the effective user ID

PSX_GETGID (GID, STATUS)
Gets the real group ID

PSX_GETPID (PID, STATUS)
Gets the process ID

PSX_GETPPID (PID, STATUS)
Gets the process ID of the parent process

PSX_GETUID (UID, STATUS)
Gets the real user ID

PSX_GMTIME
(NTICKS, SECS, MINS, HOURS, DAY, MONTH, YEAR, WDAY, YDAY, TSTRCT, STATUS)
Convert the value returned by PSX_TIME to individual GMT values

PSX_ISATTY (FILDSC, ISTTY, STATUS)
Determine if a file is a terminal

PSX_LOCALTIME
(NTICKS, SECS, MINS, HOURS, DAY, MONTH, YEAR, WDAY, YDAY, ISDST, TSTRCT, STATUS)
Convert the value returned by PSX_TIME to individual local time values

PSX_MALLOC (SIZE, PNTR, STATUS)
Allocate virtual memory

PSX_PUTENV (NAME, VALUE, STATUS)
Set a new environment variable value

PSX_RAND (INUM, MAXNUM, FNUM, STATUS)
Generate a random number

PSX_REALLOC (SIZE, PNTR, STATUS)
Change the size of an allocated region of virtual memory

PSX_REMOVE (PATH, STATUS)
Remove a file or empty directory

PSX_RENAME (INFIL, OUTFIL, STATUS)
Rename a file

PSX_SRAND (SEED, STATUS)
Set the seed for the random number generator

PSX_STAT
(PATH, ITEM, VALUE, STATUS)
Return an item of information about a file

PSX_TIME (NTICKS, STATUS)
Get the current calendar time

PSX_TTYNAME (FILDSC, TNAME, STATUS)
Get the name of the terminal

PSX_UNAME
(SYSNAME, NODENAME, RELEASE, VERSION, MACHINE, STATUS)
Gets information about the host computer system

D Classified list of routines

D.1 Process Environment

PSX_CUSERID (USER, STATUS)
Get the username

PSX_GETEGID (GID, STATUS)
Gets the effective group ID

PSX_GETENV (NAME, TRANS, STATUS)
Translate an environment variable

PSX_GETEUID (UID, STATUS)
Gets the effective user ID

PSX_GETGID (GID, STATUS)
Gets the real group ID

PSX_GETPID (PID, STATUS)
Gets the process ID

PSX_GETPPID (PID, STATUS)
Gets the process ID of the parent process

PSX_GETUID (UID, STATUS)
Gets the real user ID

PSX_ISATTY (FILDSC, ISTTY, STATUS)
Determine if a file is a terminal

PSX_PUTENV (NAME, VALUE, STATUS)
Set a new environment variable value

PSX_TTYNAME (FILDSC, TNAME, STATUS)
Get the name of the terminal

PSX_UNAME (SYSNAME, NODENAME, RELEASE, VERSION, MACHINE, STATUS)
Gets information about the host computer system

D.2 File System Support

PSX_ACCESS (NAME, MODE, ACCESSIBLE, REASON, STATUS)
Check file accessibility

PSX_CHDIR (DIR, STATUS)
Change working directory

PSX_GETCWD (CWD, STATUS)
Get the current working directory

PSX_REMOVE (PATH, STATUS)
Remove a file or empty directory

PSX_RENAME (INFIL, OUTFIL, STATUS)
Rename a file

PSX_STAT (PATH, ITEM, VALUE, STATUS)
Return an item of information about a file

D.3 Language Specific Services for C (FORTRAN versions)

D.3.1 Pseudo-Random Numbers

PSX_RAND (INUM, MAXNUM, FNUM, STATUS)
Generate a random number

PSX_SRAND (SEED, STATUS)
Set the seed for the random number generator

D.3.2 Memory Management

PSX_CALLOC (NMEMB, TYPE, PNTR, STATUS)
Allocate space for several objects of specified type

PSX_FREE (PNTR, STATUS)
Free virtual memory

PSX_MALLOC (SIZE, PNTR, STATUS)
Allocate virtual memory

PSX_REALLOC (SIZE, PNTR, STATUS)
Change the size of an allocated region of virtual memory

D.3.3 Date and Time

PSX_ASCTIME (TSTRCT, STRING, STATUS)
Convert a time structure to a character string

PSX_CTIME (NTICKS, STRING, STATUS)
Convert the calendar time to a character string

PSX_GMTIME
(NTICKS, SECS, MINS, HOURS, DAY, MONTH, YEAR, WDAY, YDAY, TSTRCT, STATUS)
Convert the value returned by PSX_TIME to individual GMT values

PSX_LOCALTIME
(NTICKS, SECS, MINS, HOURS, DAY, MONTH, YEAR, WDAY, YDAY, ISDST, TSTRCT, STATUS)
Convert the value returned by PSX_TIME to individual local time values

PSX_TIME (NTICKS, STATUS)
Get the current calendar time

E Routine Descriptions

PSX_ACCESS
Check file accessibility

Description:

Provides a FORTRAN interface to the C library function access() to determine the existence of a file for a specified type of access.

Invocation:

```
CALL PSX_ACCESS( NAME, MODE, ACCESSIBLE, REASON, STATUS )
```

Arguments:

NAME = CHARACTER∗(∗) (Given)
The name of the file to test.

MODE = CHARACTER∗(∗) (Given)
The access mode - either a `' '` (space) to merely check for file existence or one or more of the letters R,W,X.

ACCESSIBLE = LOGICAL (Returned)
.TRUE. if the access mode is allowed. .FALSE. otherwise.

REASON = INTEGER (Returned)
Error code (errno) describing the reason for failure. Can be passed to EMS_SYSER for translation if required. Will be zero if ACCESSIBLE is true or if status was set.

STATUS = INTEGER (Given & Returned)
Inherited status.

Notes:

Some FORTRAN compilers have an ACCESS intrinsic, but not all. This PSX routine is provided for portability.

References :

- POSIX Standard ISO/IEC 9945-1:1990

Copyright :

PSX_ASCTIME
Convert a time structure to a character string

Description:

Convert the information in the structure pointed to by TSTRCT to a character string. TSTRCT should have been set by a call to PSX_LOCALTIME or PSX_GMTIME.

Invocation:

```
CALL PSX_ASCTIME( TSTRCT, STRING, STATUS )
```

Arguments:

TSTRCT = POINTER (Given)
The pointer to the time structure.

STRING = CHARACTER $*$ ($*$) (Returned)
The character string representation of the time.

STATUS = INTEGER (Given)
The global status.

Examples:

```
      CALL PSX_TIME( NTICKS, STATUS )
      CALL PSX_LOCALTIME( NTICKS, SEC, MINS, HOUR, DAY, MONTH, YEAR,
     :  WDAY, YDAY, ISDST, TSTRCT, STATUS )
      CALL PSX_ASCTIME( TSTRCT, STRING, STATUS )
      PRINT *,'The time is ', STRING
```

Prints the current local time as something like:
`"Wed Apr 17 09:01:04 1991"` (without the quotes).

Notes:

- TSTRCT is declared to be of type POINTER. This is usually represented in FORTRAN as an INTEGER, although any type that uses the same amount of storage would be just as good.
- The C string returned by the function localtime contains a new line character. This is removed before being passed back to the calling FORTRAN routine.
- The actual argument corresponding to STRING should be at least 24 characters long.

External Routines Used :

cnf: cnfCptr, cnfExprt

References :

- POSIX standard (1988), section 8.1
- ANSI C standard (1989), section 4.12.3.1

Copyright :

PSX_CALLOC
Allocate space for several objects of specified type

Description:

The routine allocates an amount of virtual memory specified by NMEMB and TYPE. The number of bytes allocated is equal to the number of bytes required to store a single variable of type TYPE, multiplied by NMEMB. A pointer to the allocated storage is returned in PNTR. This pointer can be passed on to other subroutines using the %VAL construct. If the storage cannot be allocated, then PNTR is set to zero, STATUS is set to PSX__NOALL and an error is reported.

Invocation:

```
CALL PSX_CALLOC( NMEMB, TYPE, PNTR, STATUS )
```

Arguments:

NMEMB = INTEGER (Given)

The number of locations of TYPE required. If the number required exceeds the maximum that can be stored in an INTEGER (about 2.1E9), them routine PSX_CALLOC8 should be used in place of PSX_CALLOC. PSX_CALLOC8 has an identical interface except that the NMEMB argument is an INTEGER∗8.

TYPE = CHARACTER ∗ (∗) (Given)

The type of each location

PNTR = POINTER (Returned)

A pointer to the allocated storage.

STATUS = INTEGER (Given and Returned)

The global status.

Examples:

```
CALL PSX_CALLOC( 20, '_INTEGER', PNTR, STATUS )
CALL SUB1( %VAL(PNTR), 20, STATUS )
...
SUBROUTINE SUB1( ARRAY, N, STATUS )
INTEGER N
INTEGER ARRAY( N )
...
```

Generates storage for an array.

The call to PSX_CALLOC allocates storage for a 20 element array of type INTEGER. The pointer to this storage is then passed to subroutine SUB1, where it is accessed as an array of INTEGERs. We assume SUB1 returns without action if STATUS is bad.

Notes:

- Storage allocated by PSX_CALLOC should be returned by a call to PSX_FREE when it is no longer needed.
- PNTR is declared to be of type POINTER. This is usually represented in FORTRAN as an INTEGER, although any type that uses the same amount of storage would be just as good. The pointer will have been registered for C and FORTRAN use according to the scheme described in SUN/209, allowing its use where pointers are longer than INTEGERs. For portability, the construct %VAL(CNF_PVAL(PNTR)), rather than simply %VAL(PNTR), should be used to pass the pointer to the subroutine. Function CNF_PVAL is described in SUN/209 Section 'Pointers'.
- If several calls to PSX_CALLOC are made, the space returned by each call is completely separate from that made by any other call. In particular, the program should not assume that the space returned by successive calls is contiguous.
- PSX_CALLOC differs from the POSIX function calloc in that the size of each member to be allocated is specified by a character string (TYPE) rather than as a numerical value. This has been done to increase the portability of the routine.
- The allowed values of TYPE are _INTEGER, _REAL, _DOUBLE, _LOGICAL, _CHAR, _BYTE, _UBYTE, _WORD and _UWORD. The number of bytes allocated for each is as defined in the f77.h header file.

External Routines Used :

cnf: cnf Calloc, cnfFptr, cnfImpn

References :

- POSIX standard (1988), section 8.1
- ANSI C standard (1989), section 4.10.3.1

Copyright :

Copyright (C) 1991 Science & Engineering Research Council

PSX_CHDIR
Change current working directory

Description:

Provides a Fortran interface to change working directory.

Invocation:

```
CALL PSX_CHDIR( DIR, STATUS )
```

Arguments:

DIR = CHARACTER $*$ ($*$) (Given)

On exit contains the name of the current directory. Status will be set to PSX__ERRNO on error.

STATUS = INTEGER (Given & Returned)

The global status. No action takes place if status is bad on entry.

References :

- POSIX Standard IEEE Std 1003.1-1988

Copyright :

PSX_CTIME
Convert the calendar time to a character string

Description:

Convert the number of ticks since the beginning of the calendar (the value returned by PSX_TIME) to a character string.

Invocation:

```
CALL PSX_CTIME( NTICKS, STRING, STATUS )
```

Arguments:

NTICKS = INTEGER (Given)

The number of ticks since the start of the calendar.

STRING = CHARACTER * (*) (Returned)

The character string representation of the time.

STATUS = INTEGER (Given)

The global status.

Examples:

```
CALL PSX_TIME( NTICKS, STATUS )
CALL PSX_CTIME( NTICKS, STRING, STATUS )
PRINT *,'The time is ',STRING
```

Prints the current time as something like:
`"Wed Apr 17 09:01:04 1991"` (without the quotes).

Notes:

- The C string returned by the POSIX function ctime contains a new line character. This is removed before being passed back to the FORTRAN routine.
- The actual argument corresponding to STRING should be at least 24 characters long.

External Routines Used :

cnf: cnfExprt

References :

- POSIX standard (1988), section 8.1
- ANSI C standard (1989), section 4.12.3.2

Copyright :

Copyright (C) 1991 Science & Engineering Research Council

PSX_CUSERID
Get the username

Description:

This routine will get a username associated with the effective user ID of the current process. If the username cannot be found, a blank string is returned.

Invocation:

```
CALL PSX_CUSERID( USER, STATUS )
```

Arguments:

USER = CHARACTER $*$ ($*$) (Returned)
The username

STATUS = INTEGER (Given)
The global status.

Notes:

- On a Unix system the translation from effective user ID to username is performed. Since there can be several usernames associated with a user ID, there is no guarantee that the value returned will be unique.
- The Unix function cuserid is no longer in the IEEE 1003.1-1990 standard, so an alternative to this routine should be used.
- If the first attempt to get the username fails, one more attempt is made. This overcomes an occasional (timing?) problem on Linux.

External Routines Used :

cnf: cnfExprt

References :

- POSIX standard (1988), section 4.2.4

Copyright :

PSX_FREE
Free virtual memory

Description:

The routine frees the virtual memory pointed to by PNTR that was previously allocated by a call to PSX_CALLOC or PSX_MALLOC.

Invocation:

```
CALL PSX_FREE( PNTR, STATUS )
```

Arguments:

PNTR = POINTER (Given and Returned)

A pointer to the allocated storage.

STATUS = INTEGER (Given)

The global status.

Notes:

- PNTR is declared to be of type POINTER. This is usually represented in FORTRAN as an INTEGER, although any type that uses the same amount of storage would be just as good.

External Routines Used :

cnf: cnfCptr, cnfFree

References :

- POSIX standard (1988), section 8.1
- ANSI C standard (1989), section 4.10.3.2

Copyright :

Copyright (C) 1991 Science & Engineering Research Council

PSX_GETCWD
Rename a file

Description:

Provides a Fortran interface to obtain the current working directory. Some Fortran implementations provide a GETCWD builtin but this is provided for compatibility.

Invocation:

```
CALL PSX_GETWCD( CWD, STATUS )
```

Arguments:

CWD = CHARACTER ∗ (∗) (Returned)

On exit contains the name of the current directory. Status will be set to PSX__TRUNC if the string is too short to hold the directory.

STATUS = INTEGER (Given & Returned)

The global status.

Notes:

Internally, may use getwd or getcwd.

References :

- POSIX Standard ISO/IEC 9945-1:1990

Copyright :

PSX_GETEGID
Gets the effective group ID

Description:

The routine obtains the effective group identification number of the calling process and returns the value in GID.

Invocation:

```
CALL PSX_GETEGID( GID, STATUS )
```

Arguments:

GID = INTEGER (Returned)
The value of the effective group ID.

STATUS = INTEGER (Given)
The global status.

References :

- POSIX standard (1988), section 4.2.1

Copyright :

PSX_GETENV
Translate an environment variable

Description:

The routine tries to get the translation of the environment variable NAME. If it succeeds, it returns the translation in TRANS. If it fails, it sets STATUS to PSX__NOENV and reports an error.

Invocation:

```
CALL PSX_GETENV( NAME, TRANS, STATUS )
```

Arguments:

NAME = CHARACTER $*$ ($*$) (Given)
Name of the environment variable to be translated.

TRANS = CHARACTER $*$ ($*$) (Returned)
The translation of the environment variable.

STATUS = INTEGER (Given and Returned)
The global status.

Examples:

```
CALL PSX_GETENV( 'USER', TRANS, STATUS )
```

This will return the value of the environment variable USER, i.e. the username of the current process.

External Routines Used :

cnf: cnfCreim, cnfExprt, cnfFree

References :

- POSIX standard (1988), section 4.6.1
- ANSI C standard (1989), section 4.10.4.4

Copyright :

Copyright (C) 1991 Science & Engineering Research Council

PSX_GETEUID
Gets the effective user ID

Description:

The routine obtains the effective user identification number of the calling process and returns the value in UID.

Invocation:

```
CALL PSX_GETEUID( UID, STATUS )
```

Arguments:

UID = INTEGER (Returned)
The value of the effective user ID.

STATUS = INTEGER (Given)
The global status.

References :

- POSIX standard (1988), section 4.2.1

Copyright :

PSX_GETGID
Gets the real group ID

Description:

The routine obtains the real group identification number of the calling process and returns the value in GID.

Invocation:

```
CALL PSX_GETGID( GID, STATUS )
```

Arguments:

GID = INTEGER (Returned)
The value of the real group ID.

STATUS = INTEGER (Given)
The global status.

References :

- POSIX standard (1988), section 4.2.1

Copyright :

Copyright (C) 1991 Science & Engineering Research Council

PSX_GETPID
Gets the process ID

Description:

The routine obtains the process identification number of the current process and returns the value in PID.

Invocation:

```
CALL PSX_GETPID( PID, STATUS )
```

Arguments:

PID = INTEGER (Returned)
The value of the process ID.

STATUS = INTEGER (Given)
The global status.

Notes:

- When the same program is run several times on a Unix system, a different PID is returned every time.

References :

- POSIX standard (1988), section 4.1.1

Copyright :

PSX_GETPPID
Gets the process ID of the parent process

Description:

The routine obtains the process identification number of the parent process and returns the value in PID.

Invocation:

```
CALL PSX_GETPPID( PID, STATUS )
```

Arguments:

PID = INTEGER (Returned)

The value of the process ID of the parent process.

STATUS = INTEGER (Given)

The global status.

Notes:

- If a program that calls this routine is run several times, then unlike GETPID, it will always return the same process ID as all the processes will have the same parent.

References :

- POSIX standard (1988), section 4.1.1

Copyright :

Copyright (C) 1991 Science & Engineering Research Council

PSX_GETUID
Gets the real user ID

Description:

The routine obtains the real user identification number of the calling process and returns the value in UID.

Invocation:

```
CALL PSX_GETUID( UID, STATUS )
```

Arguments:

UID = INTEGER (Returned)

The value of the real user ID.

STATUS = INTEGER (Given)

The global status.

References :

- POSIX standard (1988), section 4.2.1

Copyright :

PSX_GMTIME
Convert the value returned by PSX_TIME to individual GMT values

Description:

Convert the value returned by PSX_TIME into a set of usable numbers expressed in GMT, and a pointer to the corresponding C structure. If GMT is not available, STATUS will be set to PSX__NOGMT and an error is reported.

Invocation:

```
CALL PSX_GMTIME( NTICKS, SECS, MINS, HOURS, DAY, MONTH, YEAR,
:  WDAY, YDAY, TSTRCT, STATUS )
```

Arguments:

NTICKS = INTEGER (Given)
The number of ticks since the start of the calendar.

SECS = INTEGER (Returned)
The number of seconds in the current time.

MINS = INTEGER (Returned)
The number of minutes in the current time.

HOURS = INTEGER (Returned)
The number of hours in the current time.

DAY = INTEGER (Returned)
The number of the day of the month.

MONTH = INTEGER (Returned)
The number of the month in the year.

YEAR = INTEGER (Returned)
The number of the years since 1900.

WDAY = INTEGER (Returned)
The number of the day in the week.

YDAY = INTEGER (Returned)
The number of the day in the year.

TSTRCT = POINTER (Returned)
A pointer to the C time structure.

STATUS = INTEGER (Given)
The global status.

Notes:

- The value of MONTH is 0 for January, 1 for February, etc. This is to maintain compatibility with the C run time library.
- The value of YEAR is 0 for 1900, 100 for 2000, 101 for 2001, etc.
- The value of YDAY is 0 for the first of January, 1 for the second of January, etc. This is to maintain compatibility with the C run time library.
- The value of WDAY is 0 for Sunday, 1 for Monday, etc.
- The pointer TSTRCT points to the C structure that contains the information about the time. This pointer is needed as it may be passed on to PSX_ASCTIME. The structure will be overwritten by any future call to PSX_GMTIME or PSX_LOCALTIME.
- TSTRCT is declared to be of type POINTER. This is usually represented in FORTRAN as an INTEGER, although any type that uses the same amount of storage would be just as good.

External Routines Used :

cnf: cnfFptr, cnfMalloc

References :

- POSIX standard (1988), section 4.2.1
- ANSI C standard (1989), section 4.12.3.3

Copyright :

Copyright (C) 2000 Council for the Central Laboratory of the Research Councils

PSX_ISATTY
Determine if a file is a terminal

Description:

Determine if FILDSC is a valid file descriptor associated with a terminal. ISTTY is set to TRUE if the file descriptor is associated with a terminal and FALSE otherwise.

Invocation:

```
CALL PSX_ISATTY( FILDSC, ISTTY, STATUS )
```

Arguments:

FILDSC = INTEGER (Given)
The file descriptor, which is just an integer.

ISTTY = LOGICAL (Returned)
Is the file descriptor associated with a terminal?

STATUS = INTEGER (Given)
The global status.

Examples:

```
CALL PSX_ISATTY( 0, ISTTY, STATUS )
```

Is the standard input channel a terminal?

Notes:

- On Unix the standard file descriptors are 0,1,2, for stdin, stdout and stderr, respectively.

References :

- POSIX standard (1988), section 4.7.2

Copyright :

PSX_LOCALTIME
Convert the value returned by PSX_TIME to individual local time values

Description:

Convert the value returned by PSX_TIME into a set of usable numbers expressed in local time, and a pointer to the corresponding C structure.

Invocation:

```
CALL PSX_LOCALTIME( NTICKS, SECS, MINS, HOURS, DAY, MONTH, YEAR,
:  WDAY, YDAY, ISDST, TSTRCT, STATUS )
```

Arguments:

NTICKS = INTEGER (Given)
The number of ticks since the start of the calendar.

SECS = INTEGER (Returned)
The number of seconds in the current time.

MINS = INTEGER (Returned)
The number of minutes in the current time.

HOURS = INTEGER (Returned)
The number of hours in the current time.

DAY = INTEGER (Returned)
The number of the day of the month.

MONTH = INTEGER (Returned)
The number of the month in the year.

YEAR = INTEGER (Returned)
The number of the years since 1900.

WDAY = INTEGER (Returned)
The number of the day in the week.

YDAY = INTEGER (Returned)
The number of the day in the year.

ISDST = INTEGER (Returned)
Daylight savings time flag.

TSTRCT = POINTER (Returned)
A pointer to the C time structure.

STATUS = INTEGER (Given)
The global status.

Notes:

- The value of MONTH is 0 for January, 1 for February, etc. This is to maintain compatibility with the C run time library.
- The value of YEAR is 0 for 1900, 100 for 2000, 101 for 2001, etc.
- The value of YDAY is 0 for the first of January, 1 for the second of January, etc. This is to maintain compatibility with the C run time library.
- The value of WDAY is 0 for Sunday, 1 for Monday, etc.
- The value of ISDST is 1 when daylight saving time is in effect, 0 when it is not and -1 when the information is not available.
- The pointer TSTRCT points to the C structure that contains the information about the time. This pointer is needed as it may be passed on to PSX_ASCTIME. The structure will be overwritten by any future call to PSX_LOCALTIME or PSX_GMTIME.
- TSTRCT is declared to be of type POINTER. This is usually represented in FORTRAN as an INTEGER, although any type that uses the same amount of storage would be just as good.

External Routines Used :

cnf: cnfFptr, cnfMalloc

References :

- POSIX standard (1988), section 4.2.1
- ANSI C standard (1989), section 4.12.3.4

Copyright :

PSX_MALLOC
Allocate virtual memory

Description:

The routine allocates an amount of virtual memory specified by SIZE. The unit of SIZE is the amount of storage required to store a single character. A pointer to the allocated storage is returned in PNTR. This pointer can be passed on to other subroutines using the %VAL construct. If the storage cannot be allocated, then PNTR is set to zero, STATUS is set to PSX__NOALL and an error is reported.

Invocation:

```
CALL PSX_MALLOC( SIZE, PNTR, STATUS )
```

Arguments:

SIZE = INTEGER (Given)

The amount of virtual memory to be allocated. If the number required exceeds the maximum that can be stored in an INTEGER (about 2.1E9), them routine PSX_MALLOC8 should be used in place of PSX_MALLOC. PSX_MALLOC8 has an identical interface except that the SIZE argument is an INTEGER∗8.

PNTR = POINTER (Returned)

A pointer to the allocated storage.

STATUS = INTEGER (Given and Returned)

The global status.

Examples:

```
CALL PSX_MALLOC( 40, PNTR, STATUS )
CALL SUB1( %VAL(PNTR), 10, STATUS )
...
SUBROUTINE SUB1( ARRAY, N, STATUS )
INTEGER N
INTEGER ARRAY( N )
...
```

Allocates 40 bytes and uses this as a 10 element INTEGER array.

The call to PSX_MALLOC allocates forty bytes of storage. The pointer to this storage is then passed to subroutine SUB1, where it is accessed as an array of INTEGERs. We assume SUB1 returns without action if STATUS is bad.

Note that in this case the program needs to know that an INTEGER variable is stored in four bytes. *This is not portable*. In such a case it is better to use PSX_CALLOC or to use the symbolic constants NUM_NB<T> defined in the file PRM_PAR to determine the number of bytes per unit of storage. (See SUN/39 for a description of these constants).

Notes:

- Storage allocated by PSX_MALLOC should be returned by a call to PSX_FREE when it is no longer needed.
- PNTR is declared to be of type POINTER. This is usually represented in FORTRAN as an INTEGER, although any type that uses the same amount of storage would be just as good. The pointer will have been registered for C and FORTRAN use according to the scheme described in SUN/209, allowing its use where pointers are longer than INTEGERs. For portability, the construct %VAL(CNF_PVAL(PNTR)), rather than simply %VAL(PNTR), should be used to pass the pointer to the subroutine. Function CNF_PVAL is described in SUN/209 Section 'Pointers'.
- If several calls to PSX_MALLOC are made, the space returned by each call is completely separate from that made by any other call. In particular, the program should not assume that the space returned by successive calls is contiguous.

External Routines Used :

cnf: cnfFptr, cnfMalloc

References :

- POSIX standard (1988), section 8.1
- ANSI C standard (1989), section 4.10.3.3

Copyright :

PSX_PUTENV
Set a new environment variable value

Description:

The routine sets the specified environment variable to the supplied value. If it fails it sets STATUS to PSX__NOMEM.

Invocation:

```
CALL PSX_PUTENV( NAME, VALUE, STATUS )
```

Arguments:

NAME = CHARACTER * (*) (Given)
Name of the environment variable to be set.

VALUE = CHARACTER * (*) (Given)
The new value of the environment variable.

STATUS = INTEGER (Given and Returned)
The global status.

Examples:

```
CALL PSX_PUTENV( 'DATADIR', DIR, STATUS )
```

Set the DATADIR environment variable to the value stored in the DIR character string.

External Routines Used :

cnf: cnfCreim, cnfExprt, cnfFree

References :

- POSIX standard (1988), section 4.6.1
- ANSI C standard (1989), section 4.10.4.4

Copyright :

Copyright (C) 2003 Particle Physics and Astronomy Research Council

PSX_RAND
Generate a random number

Description:

Generate a random number. The number is generated as the integer INUM. The maximum value that this may have is returned as MAXNUM. Also the value of INUM divided by MAXNUM is returned as FNUM.

Invocation:

```
CALL PSX_RAND( INUM, MAXNUM, FNUM, STATUS )
```

Arguments:

INUM = INTEGER (Returned)
The random (integer) number.

MAXNUM = INTEGER (Returned)
The maximum value that INUM may have.

FNUM = REAL (Returned)
The value INUM/MAXNUM.

STATUS = INTEGER (Given)
The global status.

Notes:

- A seed for the random number generator may be set with PSX_SRAND.
- The sequence of numbers generated by the operating system service that is called by this routine is not always as random as it should be. It is probably better to use a different routine such as one of the NAG routines. This routine is included here for completeness, though.

References :

- POSIX standard (1988), section 8.1
- ANSI C standard (1989), section 4.10.2.1

Copyright :

PSX_REALLOC
Change the size of an allocated region of virtual memory

Description:

The routine changes the size of the region of virtual memory pointed to by PNTR. The new size may be larger or smaller than the old size. The contents of the object pointed to by PNTR shall be unchanged up to the lesser of the old and new sizes.

Invocation:

```
CALL PSX_REALLOC( SIZE, PNTR, STATUS )
```

Arguments:

SIZE = INTEGER (Given)

The new amount of virtual memory required. If the number required exceeds the maximum that can be stored in an INTEGER (about 2.1E9), them routine PSX_REALLOC8 should be used in place of PSX_REALLOC. PSX_REALLOC8 has an identical interface except that the SIZE argument is an INTEGER∗8.

PNTR = POINTER (Given and Returned)

A pointer to the allocated storage

STATUS = INTEGER (Given)

The global status

Examples:

```
CALL PSX_MALLOC( 20, PNTR, STATUS )
...
CALL PSX_REALLOC( 40, PNTR, STATUS )
CALL SUB1( %VAL(PNTR), 10, STATUS )
...
SUBROUTINE SUB1( ARRAY, N, STATUS )
INTEGER N
INTEGER ARRAY( N )
...
```

Allocate 20 bytes of storage, then extend it to 40 bytes.

The call to PSX_MALLOC allocates twenty bytes of storage. The subsequent call to PSX_REALLOC extends this area to forty bytes. The pointer to this storage is then passed to subroutine SUB1, where it is accessed as an array of INTEGERs. We assume SUB1 returns without action if STATUS is bad.

Note that in this case the program needs to know that an INTEGER variable is stored in four bytes. *This is not portable*. In such a case it is better to use the symbolic constants NUM_NB<T> defined in the file PRM_PAR to determine the number of bytes per unit of storage. (See SUN/39 for a description of these constants).

Notes:

- Storage allocated by PSX_REALLOC should be returned by a call to PSX_FREE when it is no longer needed.
- PNTR is declared to be of type POINTER. This is usually represented in FORTRAN as an INTEGER, although any type that uses the same amount of storage would be just as good. The pointer will have been registered for C and FORTRAN use according to the scheme described in SUN/209, allowing its use where pointers are longer than INTEGERs. For portability, the construct %VAL(CNF_PVAL(PNTR)), rather than simply %VAL(PNTR), should be used to pass the pointer to the subroutine. Function CNF_PVAL is described in SUN/209 Section 'Pointers'.
- If SIZE is zero, then the space pointed to by PNTR is freed.
- If the space that PNTR pointed to has been deallocated by a call to PSX_FREE (or to PSX_REALLOC with SIZE = 0), then it is undefined whether the pointer can subsequently be used by PSX_REALLOC. Consequently this should not be attempted, even though it will work on some machines.

External Routines Used :

cnf: cnfCptr, cnfFptr, cnfMalloc, cnfRegp, cnfUregp

References :

- POSIX standard (1988), section 8.1
- ANSI C standard (1989), section 4.10.3.4

Copyright :

PSX_REMOVE
Remove a file or directory

Description:

This C function calls the "remove" RTL function to remove a specified file or directory. It is equivalent to "unlink" for files and "rmdir" for directories. A directory must be empty. On error, STATUS is set to PSX__ERRNO and the error message will contain the system error message.

Invocation:

```
CALL PSX_REMOVE( PATHNAME, STATUS )
```

Arguments:

PATHNAME = CHARACTER $*$ ($*$) (Given)
The path to the file.

STATUS = INTEGER (Given and Returned)
The inherited global status.

Examples:

```
CALL PSX_REMOVE( 'tmp.dat', STATUS )
```

This will remove the file tmp.dat

External Routines Used :

cnf: cnfImprt ems: emsSyser

References :

- POSIX standard, IEEE Std 1003.1

Copyright :

Copyright (C) 1999-2004 CLRC

PSX_RENAME
Rename a file

Description:

Provides a Fortran interface to rename files. The file with the name specified by the first argument is renamed to the second name.

Invocation:

```
CALL PSX_RENAME( INFIL, OUTFIL, STATUS )
```

Arguments:

INFIL = CHARACTER∗(∗) (Given)
The name of the file to rename

OUTFIL = CHARACTER∗(∗) (Given)
The new name of the file

STATUS = INTEGER (Given & Returned)
The global status.

References :

- POSIX Standard

Copyright :

PSX_SRAND
Set the seed for the random number generator

Description:

The argument SEED is used to set a new seed for the sequence of random numbers returned by the subroutine PSX_RAND. If PSX_SRAND is called with the same value of SEED, then the values returned by subsequent calls to PSX_RAND will be the same. If PSX_RAND is called before calling PSX_SRAND, then the sequence of random number returned by PSX_RAND will be the same as if PSX_SRAND had been called with SEED set to one.

Invocation:

```
CALL PSX_SRAND( SEED, STATUS )
```

Arguments:

SEED = INTEGER (Given)
The seed for the random number generator.

STATUS = INTEGER (Given and Returned)
The global status.

Notes:

- The range of values allowed for SEED is not specified. It is unlikely that values between 1 and the maximum integral value that PSX_RAND can return will cause problems.

References :

- POSIX standard (1988), section 8.1
- ANSI C standard (1989), section 4.10.2.2

Copyright :

Copyright (C) 1991 Science & Engineering Research Council

PSX_STAT
Obtain information about a file

Description:

The routine tries to get information about a specified file. If it succeeds, it returns the information in either IVAL or CVAL. If it fails, it sets STATUS to PSX__ERRNO and reports an error.

Invocation:

```
CALL PSX_STAT( PATH, ITEM, VALUE, STATUS )
```

Arguments:

PATH = CHARACTER $*$ ($*$) (Given)
The full path to the file.

ITEM = CHARACTER $*$ ($*$) (Given)
The item of information required about the file. See "Items" below.

VALUE = INTEGER (Returned)
The value for the requested item of information.

STATUS = INTEGER (Given and Returned)
The global status.

Items :

The ITEM argument can take any of the following values:

- "UID" - user ID of owner.
- "GID" - group ID of owner.
- "SIZE" - total file size, in bytes.
- "ATIME" - time of last access.
- "CTIME" - time of last status change (e.g. file creation).
- "MTIME" - time of last modification.

The time values are returned as the number of ticks since an arbitrary point in the past. See PSX_TIME.

PSX_TIME
Get the current calendar time

Description:

Determine the current calendar time. The encoding of the value is unspecified, but is the number of ticks since some date in the past. If it is not possible to get the value of NTICKS, STATUS is set to PSX__NOTIM and an error is reported.

Invocation:

```
CALL PSX_TIME( NTICKS, STATUS )
```

Arguments:

NTICKS = INTEGER (Returned)
The current time.

STATUS = INTEGER (Given and Returned)
The global status.

Notes:

- This routine is not directly useful in itself, but the value returned in NTICKS can be passed to other routines that process it further.

References :

- POSIX standard (1988), section 4.5.1
- ANSI C standard (1989), section 4.12.2.4

Copyright :

PSX_TTYNAME
Get the name of the terminal

Description:

Get the name of the terminal attached to the given file descriptor.

Invocation:

```
CALL PSX_TTYNAME( FILDSC, TNAME, STATUS )
```

Arguments:

FILDSC = INTEGER (Given)
The file descriptor.

TNAME = CHARACTER * (*) (Returned)
The name of the terminal attached to FILDSC.

STATUS = INTEGER (Given)
The global status.

Examples:

```
CALL PSX_TTYNAME( 0, TNAME, STATUS )
```

When run on a Unix system, this will return something like "/dev/ttyp2" (without the quotes).

Notes:

- If a terminal name is not found, then a blank string is returned in TNAME.

External Routines Used :

cnf: cnfExprt

References :

- POSIX standard (1988), section 4.7.2

Copyright :

PSX_UNAME
Gets information about the host computer system

Description:

The routine inquires about the operating system, the name of the computer and the type of the hardware. If an error is detected then STATUS is set to SAI__ERROR and an error is reported, although this should not happen.

Invocation:

```
CALL PSX_UNAME( SYSNAME, NODENAME, RELEASE, VERSION, MACHINE,
STATUS )
```

Arguments:

SYSNAME = CHARACTER * (*) (Returned)
Name of the operating system.

NODENAME = CHARACTER * (*) (Returned)
Node name of the computer.

RELEASE = CHARACTER * (*) (Returned)
Version of the operating system.

VERSION = CHARACTER * (*) (Returned)
Sub-version of the operating system.

MACHINE = CHARACTER * (*) (Returned)
Name of the hardware of the computer.

STATUS = INTEGER (Given and Returned)
The global status.

Examples:

```
CALL PSX_UNAME( SYSNAME, NODENAME, RELEASE, VERSION, MACHINE, STATUS)
```

When run on a SUN workstation, this will return values something like:

SYSNAME =	SunOS
NODENAME =	rlssp1
RELEASE =	4.1.1
VERSION =	1
MACHINE =	sun4c

```
CALL PSX_UNAME( SYSNAME, NODENAME, RELEASE, VERSION, MACHINE, STATUS)
```

When run on a DECstation, this will return values something like:

SYSNAME =	ULTRIX
NODENAME =	rlsux1
RELEASE =	4.0
VERSION =	0
MACHINE =	RISC

External Routines Used :

cnf: cnfCopyf, cnfExprt

References :

- POSIX standard (1988), section 4.4.1

Copyright :

F Routines that have not been implemented

This is an alphabetical list of POSIX routines that have been considered for implementation as PSX routines and have been rejected as unnecessary. Any routine that has not been implemented, yet is not in the following list has simply not yet been looked at and may well be done in the future.

If you have a need for any routine that has not been implemented, for whatever the reason, then please contact the author of this document, who will endeavour to change this.

GETGROUPS
Get the supplementary group IDs of the calling process

SETGID
Sets the real and effective group ID

SETUID
Sets the real and effective user ID

G Notes for System Programmers

Although the PSX routines appear to the user as FORTRAN subroutines, they are actually written in C. The FORTRAN – C interface is handled by the macros and functions of Starlink's CNF package (see SUN/209). The use of these macros makes the source code sufficiently portable that it runs on all Starlink supported hardware platforms.

Despite the fact that the routines are written in C, all character strings are returned as normal FORTRAN strings with trailing blanks.

The PSX routines call the C run time library.

The PSX routines report errors via EMS (see SSN/4). While this is a standard feature of Starlink subroutine libraries, occasionally it will be necessary *not* to report errors via EMS. For example, if a PSX routine is used within EMS, and that PSX routine were to report an error then there is the potential for recursive error reports being generated. Also, if the PSX routines were needed on a non-Starlink system, then EMS would not necessarily be present. To try to take account of these situations, the PSX routines actually call the internal routine `psx1_rep_c` to report an error. This normally calls `emsRep` (the C interface to `EMS_REP`), but could be re-coded where required.