# Documentation Updates for APAR PH54481

# Updates for z/OS C/C++ Runtime Library Reference

This document contains updates to the information in z/OS C/C++ Runtime Library Reference (SC14-7314-XX).

# Chapter 2. Header files

# C/C++ header files

## dirent.h — POSIX directory access

The dirent.h header file contains constants, prototypes, and typedef definitions for POSIX directory access functions. It declares the following functions.

…

_POSIX_C_SOURCE 200809L

dirfd() fdopendir()

_XPLATFORM_SOURCE
fdclosedir()

## unistd.h — Implementation-specific functions

The unistd.h header file declares a number of implementation-specific functions:

…

_XPLATFORM_SOURCE
dup3() getentropy() pipe2() pivot_root() syncfs() syscall()

# Chapter 3. Library functions

## fdopen() — Associate a stream with an open file descriptor

…

**fdopendir – open directory associated with file descriptor**

**Standards**

| Standards / Extensions | C or C++ | Dependencies |
|---|---|---|
| POSIX.1 | both | |

**Format**

```
#define _POSIX_C_SOURCE 200809L
#include <dirent.h>

DIR *fdopendir(int fd);
```

**General description**

The fdopendir() function shall be equivalent to the opendir() function except that the directory is specified by a file descriptor rather than by a name. The file offset associated with the file descriptor at the time of the call determines which entries are returned.

Upon successful return from fdopendir(), the file descriptor is under the control of the system, and if any attempt is made to close the file descriptor, or to modify the state of the associated description, other than by means of closedir(), readdir(), readdir_r(), rewinddir(), or seekdir(),the behavior is undefined. Upon calling closedir() the file descriptor shall be closed.

**Returned value**

Upon successful completion, this function shall return a pointer to an object of type DIR. Otherwise, this function shall return a null pointer and set errno to indicate the error.

**EBADF**

The fd argument is not a valid file descriptor open for reading.

**ENOTDIR**

Not a directory.

**ENOMEM**

Insufficient memory to complete the operation.

**Example**

```
#define _POSIX_SOURCE
#define  _POSIX_C_SOURCE 200809L
#include <dirent.h>
#include <stdio.h>

int main() {

  int fd = 0;

  DIR *dir;
  DIR *newdirp;
  struct dirent *entry;
```

```
    dir = opendir("/u/userexample/fdopendir");
    fd = dirfd(dir);
    newdirp = fdopendir(fd);
    while ((entry = readdir(newdirp))!= NULL)
      printf("  %s\n", entry->d_name);

    return 0;
  }
```

Related information

- "dirent.h — POSIX directory access" on page XX
- "stdio.h — Standard input and output" on page XX
- "sys/types.h — typedef symbols and structures" on page XX
- "closedir() — Close a directory" on page XX
- "opendir() — Open a directory" on page XX
- "readdir() — Read an entry from a directory" on page XX
- "rewinddir() — Reposition a directory stream to the beginning" on page XX
- "seekdir() — Set position of directory stream" on page XX
- "telldir() — Current location of directory stream" on page XX
- "fdclosedir() - close directory associated with file descriptor" on page XX

# fdatasync() — Write changes to direct-access storage

...

**fdclosedir - close directory associated with file descriptor**

**Standards**

| Standards / Extensions | C or C++ | Dependencies |
|---|---|---|
| z/OS UNIX, __XPLAT | both | |

**Format**

```
#define _XPLATFORM_SOURCE
#include <dirent.h>
```

int **fdclosedir**(DIR *dirp);

**General description**

The fdclosedir() function is equivalent to the closedir() function except that this function returns directory file descriptor instead of closing it.

**Returned value**

Upon successful completion, this function shall return a file descriptor. Otherwise, this function shall return -1 and set errno to indicate the error.

**ENOTDIR**

Not a directory.

**Example**

```
#define _POSIX_SOURCE
#define  _POSIX_C_SOURCE 200809L
#define _XPLATFORM_SOURCE
#include <dirent.h>
#include <stdio.h>

int main() {

  int fd = 0;

  DIR *dir;
  DIR *newdirp;
  struct dirent *entry;

  dir = opendir("/u/userexample/fdopendir");
  fd = dirfd(dir);
  newdirp = fdopendir(fd);
  while ((entry = readdir(newdirp))!= NULL)
    printf("  %s\n", entry->d_name);

  fd = fdclosedir(newdirp);

  return 0;
}
```

Related information

- “dirent.h — POSIX directory access” on page XX
- “stdio.h — Standard input and output” on page XX

- "sys/types.h — typedef symbols and structures" on page XX
- "closedir() — Close a directory" on page XX
- "opendir() — Open a directory" on page XX
- "readdir() — Read an entry from a directory" on page XX
- "rewinddir() — Reposition a directory stream to the beginning" on page XX
- "seekdir() — Set position of directory stream" on page XX
- "telldir() — Current location of directory stream" on page XX
- "fdopendir() - open directory associated with file descriptor" on page XX

Also add the below part into related information of closedir(), opendir(), readdir(), rewinddir(), seekdir(), telldir():

- "fdopendir() - open directory associated with file descriptor" on page XX
- "fdclosedir() - close directory associated with file descriptor" on page XX

## syncfs() — Schedule specific file system updates

...

**syscall - indirect system call**

**Standards**

| Standards / Extensions | C or C++ | Dependencies |
|---|---|---|
| z/OS UNIX, __XPLAT | both | |

**Format**

```
#define _XPLATFORM_SOURCE
#include <unistd.h>

long syscall(long number, ...);
```

**General description**

syscall() is a library function that invokes the system call whose assembly language interface has the specified number with the specified arguments. Employing syscall() is useful, for example, when invoking a system call that has no wrapper function in the C library.

Symbolic constants for system call numbers can be found in the header file </usr/include/zos/bpxysysc.h>.

**Returned value**

The return value is defined by the system call being invoked.

**Example**

```
#define _XPLATFORM_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>


int main (void)
{
   int i;
   int dirlen = 10;
   char dirname[10]= "/u/userexample";
   int buflen = 1023;
   char *bufferc;
   char bufferout[1024];
   int retval = 0;
   int retcod = 0;
   int reascod = 0;

   int fd;
   int alet = 0;

   bufferc = (char *)malloc(1024);
   memset(bufferc,0x00,1024);

   /* opendir */
   i = syscall(37,&dirlen,dirname,&retval,&retcod,&reascod);

   if (retval > 0) {
     fd = retval;
     retval = 0;
     /* readdir */
     i = syscall(41,&fd,&bufferc,&alet,
                 &buflen,&retval,&retcod,&reascod);
   }

   printf("fd = %d\n", fd);
   printf("errno = %d, errno2 = %x\n", errno, __errno2());
   memcpy(bufferout,bufferc,1024);
   printf("***bufferc start***\n");
   for (i=0; i<1024; i++) {
     printf("%c", bufferout[i]);
   }
   printf("\n***bufferc end***\n");

   return 0;
}
```

Related information

- “unistd.h — Implementation-specific functions” on page XX