PBFileSys

P. Baillehache

February 10, 2019

Contents

Introduction

PBFileSys library is a C library to interact with the file system.

It provide functions to:

- Join paths together taking care of adding the folder separator character when needed
- Get the root of a path (e.g. "/A/B" -¿ "/A")

It uses the `PBErr` and `PBCExtension` libraries.

1 Interface

```
// ============ PBFILESYS.H ================

#ifndef PBFILESYS_H
#define PBFILESYS_H

// ================= Include =================

#include <stdlib.h>
#include <stdio.h>
#include "pberr.h"
#include "pbcextension.h"

// ================= Define ==================

#define PBFILESYS_FOLDERSEP '/'

// ================ Functions declaration ====================

// Join the paths in arguments
// Ignore the empty paths
// Return the result path as a new string
// Take care of adding the folder separator where needed
char* _PBFSJoinPath(const int nbPath, ...);
#define PBFSJoinPath(...) \
  _PBFSJoinPath(__VA_NB_ARGS__(const char*, __VA_ARGS__), __VA_ARGS__)

// Get the root folder of the path 'path'
// Examples:
// PBFSGetRootPath("A/B") -> "A"
// PBFSGetRootPath("/A/B") -> "/A"
// PBFSGetRootPath("/A/B/") -> "/A/B"
// PBFSGetRootPath("A") -> ""
// PBFSGetRootPath("/A") -> "/"
// PBFSGetRootPath("/") -> "/"
// PBFSGetRootPath("") -> ""
// PBFSGetRootPath(NULL) -> NULL
char* PBFSGetRootPath(const char* path);

// ================ Inliner ====================

#if BUILDMODE != 0
#include "pbfilesys-inline.c"
#endif

#endif
```

2 Code

2.1 pbfilesys.c

```
// ============ PBFILESYS.C ================

// ================= Include =================

#include "pbfilesys.h"
#if BUILDMODE == 0
```

```
#include "pbfilesys-inline.c"
#endif

// ================ Functions implementation ====================

// Join the paths in arguments
// Ignore the empty paths
// Return the result path as a new string
// Take care of adding the folder separator where needed
char* _PBFSJoinPath(const int nbPath, ...) {
  // Declare a variable to calculate the length of the final string
  int len = 0;
  // Loop on the arguments
  va_list paths;
  va_start(paths, nbPath);
  for (int iPath = 0; iPath < nbPath; ++iPath) {
    char* path = va_arg(paths, char*);
    int l = strlen(path);
    len += l;
    if (l > 0 && path[l - 1] != PBFILESYS_FOLDERSEP)
      ++len;
  }
  // Allocate memory for the final string
  char* joinedPath = PBErrMalloc(PBFileSysErr, (len + 1) * sizeof(char));
  // Loop again on the arguments
  va_start(paths, nbPath);
  int startPos = 0;
  for (int iPath = 0; iPath < nbPath; ++iPath) {
    char* path = va_arg(paths, char*);
    int l = strlen(path);
    if (l > 0) {
      strcpy(joinedPath + startPos, path);
      startPos += l;
      if (iPath < nbPath - 1 && path[l - 1] != PBFILESYS_FOLDERSEP) {
        joinedPath[startPos] = PBFILESYS_FOLDERSEP;
        ++startPos;
      }
    }
  }
  va_end(paths);
  // Return the final string
  return joinedPath;
}

// Get the root folder of the path 'path'
// Examples:
// PBFSGetRootPath("A/B") -> "A"
// PBFSGetRootPath("/A/B") -> "/A"
// PBFSGetRootPath("/A/B/") -> "/A/B"
// PBFSGetRootPath("A") -> ""
// PBFSGetRootPath("/A") -> "/"
// PBFSGetRootPath("/") -> "/"
// PBFSGetRootPath("") -> ""
// PBFSGetRootPath(NULL) -> NULL
char* PBFSGetRootPath(const char* path) {
  // If the path is null
  if (path == NULL)
    return NULL;
  // Declare a variable for the result
  char* res = NULL;
  // Declare a pointer to search the last separator
  const char* ptr = path + strlen(path);
```

```
  // Look for the last separator
  while (*ptr != PBFILESYS_FOLDERSEP && ptr != path)
    ptr--;
  // If we went down to the beginning of the path
  if (ptr == path) {
    // if the first char of the path is a separator
    if (*ptr == PBFILESYS_FOLDERSEP) {
      // The result is the separtor only
      res = PBErrMalloc(PBFileSysErr, sizeof(char) * 2);
      res[0] = PBFILESYS_FOLDERSEP;
      res[1] = '\0';
    // Else the the first char is not a separator
    } else {
      // The result is the empty string
      res = PBErrMalloc(PBFileSysErr, sizeof(char));
      res[0] = '\0';
    }
  // Else, we have found the last separator in the middle of the path
  } else {
    // If we have stopped on a separator
    if (*ptr == PBFILESYS_FOLDERSEP)
      // Skip it
      ptr--;
    // Copy the root path in the result
    int l = (1 + ptr - path);
    res = PBErrMalloc(PBFileSysErr, sizeof(char) * (1 + l));
    memcpy(res, path, l);
    res[l] = '\0';
  }
  // Return the root path
  return res;
}
```

2.2 pbfilesys-inline.c

```
// ============ PBFILESYS_INLINE.C ================

// ================ Functions implementation ====================
```

3 Makefile

```
# Build mode
# 0: development (max safety, no optimisation)
# 1: release (min safety, optimisation)
# 2: fast and furious (no safety, optimisation)
BUILD_MODE?=0

all: pbmake_wget main

# Automatic installation of the repository PBMake in the parent folder
pbmake_wget:
if [ ! -d ../PBMake ]; then wget https://github.com/BayashiPascal/PBMake/archive/master.zip; unzip master.zip; rm -f

# Makefile definitions
MAKEFILE_INC=../PBMake/Makefile.inc
include $(MAKEFILE_INC)
```

```
# Rules to make the executable
repo=pbfilesys
$($(repo)_EXENAME): \
$($(repo)_EXENAME).o \
$($(repo)_EXE_DEP) \
$($(repo)_DEP)
$(COMPILER) `echo "$($(repo)_EXE_DEP) $($(repo)_EXENAME).o" | tr ' ' '\n' | sort -u` $(LINK_ARG) $($(repo)_LINK_ARG)

$($(repo)_EXENAME).o: \
$($(repo)_DIR)/$($(repo)_EXENAME).c \
$($(repo)_INC_H_EXE) \
$($(repo)_EXE_DEP)
$(COMPILER) $(BUILD_ARG) $($(repo)_BUILD_ARG) `echo "$($(repo)_INC_DIR)" | tr ' ' '\n' | sort -u` -c $($(repo)_DIR)/$
```

4 Unit tests

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <math.h>
#include "pbfilesys.h"

void UnitTestJoinPath() {
  char pathA[4] = ". A\0";
  pathA[1] = PBFILESYS_FOLDERSEP;
  char pathB[2] = "B\0";
  char pathC[6] = " A B \0";
  pathC[0] = PBFILESYS_FOLDERSEP;
  pathC[2] = PBFILESYS_FOLDERSEP;
  pathC[4] = PBFILESYS_FOLDERSEP;
  char pathD[3] = "C \0";
  pathD[1] = PBFILESYS_FOLDERSEP;
  char pathE[6] = "D.txt\0";
  char checkA[6] = ". A B\0";
  checkA[1] = PBFILESYS_FOLDERSEP;
  checkA[3] = PBFILESYS_FOLDERSEP;
  char checkB[13] = " A B C D.txt\0";
  checkB[0] = PBFILESYS_FOLDERSEP;
  checkB[2] = PBFILESYS_FOLDERSEP;
  checkB[4] = PBFILESYS_FOLDERSEP;
  checkB[6] = PBFILESYS_FOLDERSEP;
  char* path = PBFSJoinPath(pathA, pathB);
  printf("%s + %s -> %s\n", pathA, pathB, checkA);
  if (path == NULL || strcmp(path, checkA) != 0) {
    PBFileSysErr->_type = PBErrTypeUnitTestFailed;
    sprintf(PBFileSysErr->_msg, "PBFSJoinPath failed");
    PBErrCatch(PBFileSysErr);
  }
  free(path);
  path = PBFSJoinPath(pathC, pathD, pathE);
  printf("%s + %s + %s -> %s\n", pathC, pathD, pathE, checkB);
  if (path == NULL || strcmp(path, checkB) != 0) {
    PBFileSysErr->_type = PBErrTypeUnitTestFailed;
    sprintf(PBFileSysErr->_msg, "PBFSJoinPath failed");
    PBErrCatch(PBFileSysErr);
  }
  free(path);
```

```
  path = PBFSJoinPath("", pathD);
  printf("'' + %s -> %s\n", pathD, pathD);
  if (path == NULL || strcmp(path, pathD) != 0) {
    PBFileSysErr->_type = PBErrTypeUnitTestFailed;
    sprintf(PBFileSysErr->_msg, "PBFSJoinPath failed");
    PBErrCatch(PBFileSysErr);
  }
  free(path);
  path = PBFSJoinPath(pathD, "");
  printf("%s + '' -> %s\n", pathD, pathD);
  if (path == NULL || strcmp(path, pathD) != 0) {
    PBFileSysErr->_type = PBErrTypeUnitTestFailed;
    sprintf(PBFileSysErr->_msg, "PBFSJoinPath failed");
    PBErrCatch(PBFileSysErr);
  }
  free(path);
  printf("UnitTestJoinPath OK\n");
}

void UnitTestRootPath() {
  char* res = NULL;
  res = PBFSGetRootPath("A/B");
  printf("A/B -> %s\n", res);
  if (strcmp(res, "A") != 0) {
    PBFileSysErr->_type = PBErrTypeUnitTestFailed;
    sprintf(PBFileSysErr->_msg, "PBFSGetRootPath failed");
    PBErrCatch(PBFileSysErr);
  }
  free(res);
  res = PBFSGetRootPath("/A/B");
  printf("/A/B -> %s\n", res);
  if (strcmp(res, "/A") != 0) {
    PBFileSysErr->_type = PBErrTypeUnitTestFailed;
    sprintf(PBFileSysErr->_msg, "PBFSGetRootPath failed");
    PBErrCatch(PBFileSysErr);
  }
  free(res);
  res = PBFSGetRootPath("/A/B/");
  printf("/A/B/ -> %s\n", res);
  if (strcmp(res, "/A/B") != 0) {
    PBFileSysErr->_type = PBErrTypeUnitTestFailed;
    sprintf(PBFileSysErr->_msg, "PBFSGetRootPath failed");
    PBErrCatch(PBFileSysErr);
  }
  free(res);
  res = PBFSGetRootPath("A");
  printf("A -> %s\n", res);
  if (strcmp(res, "") != 0) {
    PBFileSysErr->_type = PBErrTypeUnitTestFailed;
    sprintf(PBFileSysErr->_msg, "PBFSGetRootPath failed");
    PBErrCatch(PBFileSysErr);
  }
  free(res);
  res = PBFSGetRootPath("/A");
  printf("/A -> %s\n", res);
  if (strcmp(res, "/") != 0) {
    PBFileSysErr->_type = PBErrTypeUnitTestFailed;
    sprintf(PBFileSysErr->_msg, "PBFSGetRootPath failed");
    PBErrCatch(PBFileSysErr);
  }
  free(res);
  res = PBFSGetRootPath("/");
```

```
  printf("/ -> %s\n", res);
  if (strcmp(res, "/") != 0) {
    PBFileSysErr->_type = PBErrTypeUnitTestFailed;
    sprintf(PBFileSysErr->_msg, "PBFSGetRootPath failed");
    PBErrCatch(PBFileSysErr);
  }
  free(res);
  res = PBFSGetRootPath(NULL);
  printf("NULL -> NULL\n");
  if (res != NULL) {
    PBFileSysErr->_type = PBErrTypeUnitTestFailed;
    sprintf(PBFileSysErr->_msg, "PBFSGetRootPath failed");
    PBErrCatch(PBFileSysErr);
  }
  free(res);

  printf("UnitTestRootPath OK\n");
}

void UnitTestAll() {
  UnitTestJoinPath();
  UnitTestRootPath();
}

int main(void) {
  UnitTestAll();
  return 0;
}
```

5 Unit tests output

```
./A + B -> ./A/B
/A/B/ + C/ + D.txt -> /A/B/C/D.txt
'' + C/ -> C/
C/ + '' -> C/
UnitTestJoinPath OK
A/B -> A
/A/B -> /A
/A/B/ -> /A/B
A ->
/A -> /
/ -> /
NULL -> NULL
UnitTestRootPath OK
```