```
//****************************************************************************
**
// FILE:                zarIO.c
// PROJECT:             zarLib  - ver 1.00 -
// COPYRIGHTS:  The famous group 4 of "Roskilde Universites Center"
//****************************************************************************
***/
// Headers:
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "zarLocal.h"


//****************************************************************************
**
// Typedefs:

typedef struct
{
        _dword   dwFileTag;             //      The tag for this file (== ZAR_FI
LE_TAG)
        _ushort  usVer;                 //  The version of the program
        _ushort  usInfo;                //  User data, can be used for any purpo
se
        _ushort  cxFileName;    //  The size of the file name in bytes:
} ZAR_FILE_HEADER;


// The file tag:
#define ZAR_FILE_TAG    0x72617A        // == 'zar';
#define BUFFER_SIZE             16384           // == 16 KB


//****************************************************************************
**
// Globals:

_dword g_dwUpdateChunck = 0xffffffff;


//****************************************************************************
**
// zarLoadFile

ZAR_FILE* zarLoadFile(_pcstr pFileName, _pcstr pReadMode)
{
        ZAR_FILE* pzarFile;

        // Open the file:
        FILE* pFile = fopen(pFileName, pReadMode);

        if (pFile == NULL)
                return NULL;


        // Allocate memory for file:
        pzarFile = (ZAR_FILE*) malloc(sizeof(ZAR_FILE));        // Cast required
 for C++
        if (pzarFile  == NULL)
        {
                fclose(pFile);
                return NULL;
        }

        // Init the data for the file:
        memset(pzarFile, 0, sizeof(ZAR_FILE));

        setvbuf(pFile, NULL, _IOFBF, BUFFER_SIZE);
```



```
        pzarFile->pFile   = pFile;
        zarFileSize(pzarFile);


        // Return the opened file:
        return pzarFile;
}


//****************************************************************************
**
// zarLoadInputFile()

ZAR_FILE* zarLoadInputFile(_pcstr pFileName)
{
        ZAR_FILE* pzFile = zarLoadFile(pFileName, "rb");

        if (pzFile == NULL)
        {
                zarError(ERR_INPUT_FILE_NOT_READABLE, pFileName);
                return NULL;
        }


        // Read the fist byte:
        pzFile->nCurByte = 0;
        pzFile->nBitMask = 0;

        return pzFile;
}


//****************************************************************************
**
// zarLoadOutputFile()

ZAR_FILE* zarLoadOutputFile(_pcstr pFileName, int nFlags)
{
        ZAR_FILE* pzFile;

        if (nFlags & ZAR_ASK_IF_EXISTS)
        {
                // See if the file already exists:
                FILE* pFile = fopen(pFileName, "rb");

                if (pFile != NULL)
                {
                        // Close the file:
                        fclose(pFile);

                        // Ask user if it´s ok to overwite the output file:
                        printf("The file \"%s\" does already exist, do you want to overwrite it (y/n)? "
, pFileName);
                        if ('y' != tolower(getc(stdin)))
                                return NULL;

                        // Print space (looks better):
                        printf("\n");
                }
        }


        // Load the file:
        pzFile = zarLoadFile(pFileName, "wb");

        if (pzFile == NULL)
        {
                zarError(ERR_OUTPUT_FILE_NOT_CREATEABLE, pFileName);
                return NULL;
        }
```

```
        pzFile->nBitMask = 0x80;

        return pzFile;
}


//***************************************************************************
**
// zarCloseFile()

void zarCloseFile(ZAR_FILE* pzarFile)
{
        if (pzarFile != NULL)
        {
                // Close file:
                if (pzarFile->pFile != NULL)    // Required, because fclose() ca
n´t handle NULL pointers.
                        fclose(pzarFile->pFile);

                // Free memory used by the structure:
                free(pzarFile);
        }
}


//***************************************************************************
**
// zarFlushData()

void zarFlushFileData(ZAR_FILE* pzFile)
{
        // Should some data be flushed:
        if (pzFile->nBitMask != 0x80)
        {
                fputc(pzFile->nCurByte, pzFile->pFile);
                pzFile->nBitMask = 0x80;
        }
}


//***************************************************************************
**
// getFileSize()

_dword zarFileSize(ZAR_FILE* pzFile)
{
        long lPos, lEof;

        // Save current position:
        lPos = ftell(pzFile->pFile);

        // Seek to the end of the file, so we can determinate the filesize:
        fseek(pzFile->pFile, 0L, SEEK_END);
        lEof = ftell(pzFile->pFile);

        // Save the size:
        pzFile->dwFileSize = (_dword) lEof;

        // Reset the file-pos, so we don´t mess things up!
        fseek(pzFile->pFile, lPos, SEEK_SET);

        return (_dword) lEof;
}


//***************************************************************************
**
// zarFileSizeFromPath()
```



```
_dword zarFileSizeFromPath(_pcstr pszFile)
{
        long  lEof;
        FILE* pFile = fopen(pszFile, "rb");

        if (pFile == NULL)
                return 0;

        // Find end of file:
        fseek(pFile, 0, SEEK_END);
        lEof = ftell(pFile);

        fclose(pFile);

        return (_dword) lEof;
}


//***************************************************************************
**
// zarGetFileNameInPath()

int zarGetFileNameInPath(_pcstr pszSrc, _pstr pszDst)
{
        _pcstr   pszFileName;

        // First try if this is a microsoft-path?
        pszFileName = strrchr(pszSrc, '\\');

        if (pszFileName == NULL)
        {
                // Then try a unix-path:
                pszFileName = strrchr(pszSrc, '/');

                if (pszFileName == NULL)
                {
                        // Then assume this is a drive:
                        pszFileName = strrchr(pszSrc, ':');

                        if (pszFileName != NULL)
                                pszFileName++;
                }
        }

        // Then it must be a filename without a path:
        if (pszFileName == NULL)
                pszFileName = pszSrc;

        // Copy the buffer:
        strcpy(pszDst, pszFileName);

        return 1;
}


//***************************************************************************
**
// zarSetFileExtension()

void zarSetFileExtension(_pstr pszSrc)
{
        _pstr pExt;

        // Find last '.':
        pExt = strrchr(pszSrc, '.');

        // Remove the old extension (if any):
        if (pExt != NULL)
                *pExt = '\0';
```

```
        // Add the new one:
        strcat(pszSrc, ZAR_FILE_EXTENSION);
}


//***************************************************************************
**
// zarSetFileHeader()

int zarSaveFileHeader(const ZAR_FILE* pzFile, _ushort usInfo, _pcstr pFileName)
{
        ZAR_FILE_HEADER zfHdr;

        // Fill in the header:
        zfHdr.dwFileTag  = ZAR_FILE_TAG;
        zfHdr.usVer              = (_ushort) ZAR_VERSION;
        zfHdr.usInfo     = usInfo;
        zfHdr.cxFileName = (_ushort) ((pFileName == NULL) ? 0 : strlen(pFileName
));

        // Find begining of file:
        if (0 != fseek(pzFile->pFile, 0, SEEK_SET))
                return zarError(ZAR_ERR_FILE_SEEK, pzFile);

        // Save header:
        if (1 != fwrite(&zfHdr, sizeof(ZAR_FILE_HEADER), 1, pzFile->pFile))
                return zarError(ERR_OUTPUT_FILE_NOT_WRITEABLE, NULL);

        // Save the file name of the source file (if any):
        if (pFileName != NULL)
                if (zfHdr.cxFileName != fwrite(pFileName, sizeof(char), zfHdr.cx
FileName, pzFile->pFile))
                        return zarError(ERR_OUTPUT_FILE_NOT_WRITEABLE, NULL);

        return 1;
}


//***************************************************************************
**
// zarLoadFileHeader()

int zarLoadFileHeader(const ZAR_FILE* pzFile, _ushort* pusInfo, _pstr pFileName)
{
        ZAR_FILE_HEADER zfHdr;

        // Read the header:
        if (1 != fread(&zfHdr, sizeof(ZAR_FILE_HEADER), 1, pzFile->pFile))
                return zarError(ERR_INPUT_FILE_NOT_READABLE, NULL);

        // Make sure this is a .zar file:
        if (zfHdr.dwFileTag != ZAR_FILE_TAG)
                return zarError(ERR_INPUT_FILE_NOT_VALID_ZAR_FILE, NULL);

        // Verify version:
        if (zfHdr.usVer > ZAR_VERSION)
                return zarError(ZAR_ERR_WRONG_ZAR_VERSION, NULL);

        // Save the info-member:
        if (pusInfo != NULL)
                *pusInfo = zfHdr.usInfo;

        // Save the filename:
        if (pFileName == NULL)
                fseek(pzFile->pFile, zfHdr.cxFileName, SEEK_CUR);       // Skip
filename
        else
        {
                // Get the fileName:
                if (zfHdr.cxFileName == 0)
```



```
                        strcpy(pFileName, "");          // No filename, se retur
n an empty string.
                else
                {
                        // Copy filename:
                        if (zfHdr.cxFileName != fread(pFileName, sizeof(char), z
fHdr.cxFileName, pzFile->pFile))
                                return zarError(ERR_INPUT_FILE_NOT_READABLE, NUL
L);

                        pFileName[zfHdr.cxFileName] = '\0';      // Append zero-t
erminator.
                }
        }

        return 1;
}

//***************************************************************************
**
// zarGetBitsF()

_dword zarGetBitsF(ZAR_FILE* pzFile, int nNumBits)
{
        _dword dwRet = 0;
        _dword dwMask = 1L << (nNumBits -1);

        while (dwMask != 0)
        {
                // Should we get a new byte?
                if (pzFile->nBitMask == 0)
                {
                        // Read a byte:
                        pzFile->nCurByte = fgetc(pzFile->pFile);
                        pzFile->dwByteCount++;

                        if ((pzFile->dwByteCount % g_dwUpdateChunck) == 0)
                                printf("*");

                        // Update mask to extract the first bit:
                        pzFile->nBitMask = 0x80;
                }

                if (pzFile->nCurByte & pzFile->nBitMask)
                        dwRet |= dwMask;

                // Update masks:
                dwMask >>= 1;
                pzFile->nBitMask >>= 1;
        }

        return dwRet;
}


//***************************************************************************
**
// zarGetBitF()

int zarGetBitF(ZAR_FILE* pzFile)
{
        int nVal;

        // Should we read a new byte?
        if (pzFile->nBitMask == 0)
        {
                // Read a byte:
                pzFile->nCurByte = fgetc(pzFile->pFile);
                pzFile->dwByteCount++;
```

```
                if ((pzFile->dwByteCount % g_dwUpdateChunck) == 0)
                        printf("*");

                // Update mask to extract the first bit:
                pzFile->nBitMask = 0x80;
        }

        // Update the mask to extract the next bit:
        nVal = pzFile->nBitMask;
        pzFile->nBitMask >>= 1;

        return (pzFile->nCurByte & nVal);
}


//***************************************************************************
**
// zarSetBits()

void zarSetBitsF(ZAR_FILE* pzFile, _dword dwBits, int nBitCount)
{
        _dword dwMask = 1L << (nBitCount -1);

        while (dwMask != 0)
        {
                // Add a one?
                if (dwMask & dwBits)
                        pzFile->nCurByte |= pzFile->nBitMask;

                // Update mask:
                pzFile->nBitMask >>= 1;

                // Should we store the data:
                if (pzFile->nBitMask == 0)
                {
                        fputc(pzFile->nCurByte, pzFile->pFile);

                        // Save the byte:
                        pzFile->dwByteCount++;

                        if ((pzFile->dwByteCount % g_dwUpdateChunck) == 0)
                                printf("*");

                        // Update:
                        pzFile->nBitMask = 0x80;
                        pzFile->nCurByte = 0;
                }

                dwMask >>= 1;
        }
}


//***************************************************************************
**
// zarSetBitF()

void zarSetBitF(ZAR_FILE* pzFile, int bit)
{
        if (bit)
                pzFile->nCurByte |= pzFile->nBitMask;


        // Update mask to next bit:
        pzFile->nBitMask >>= 1;


        // Should we store the data:
        if (pzFile->nBitMask == 0)
        {
```



```
                fputc(pzFile->nCurByte, pzFile->pFile);

                // Save the byte:
                pzFile->dwByteCount++;

                if ((pzFile->dwByteCount % g_dwUpdateChunck) == 0)
                        printf("*");

                // Update:
                pzFile->nBitMask = 0x80;
                pzFile->nCurByte = 0;
        }
}


//***************************************************************************
**
// zarGetByteF()

int zarGetByteF(const ZAR_FILE* pzFile)
{
        return fgetc(pzFile->pFile);
}


//***************************************************************************
**
// zarSetByteF()

void zarSetByteF(const ZAR_FILE* pzDst, int nByte)
{
        fputc(nByte, pzDst->pFile);
}


//***************************************************************************
**
// zarSetUpdateChunckF()

void zarSetUpdateChunckF(_dword dwChunckSize)
{
        g_dwUpdateChunck = dwChunckSize;

        if (g_dwUpdateChunck == 0)      // the library could crash, if g_dwUpdat
eChunck == 0!
                g_dwUpdateChunck = 0xffffffff;
}


//***************************************************************************
**
// zarSetBytesF()

void zarSetBytesF(const ZAR_FILE* pzDst, _pstr pBytes, int nNumBytes)
{
        fwrite(pBytes, sizeof(char), nNumBytes, pzDst->pFile);
}


//***************************************************************************
****
// zarGetBytesF()

_dword zarGetBytesF(const ZAR_FILE* pzSrc, void* pBuffer, int nNumBytes)
{
        return fread(pBuffer, sizeof(_byte), nNumBytes, pzSrc->pFile);
}
```