

```

/*****
/*
/*          E N U M F O L D . C
/*
/*-----
/* Task      : Functions for enumerating Shell folders and
/*            Shell objects.
/*-----
/* Authors    : Michael Tischer and Bruno Jennrich
/* developed on : 09/15/1995
/* last update  : 09/22/1995
/*****
#include <windows.h>
#include <shlobj.h>
#include "EnumFold.h"

/*****
/* pidlNext : Function for getting the next PIDL from a
/*            given PIDL list
/*-----
/* Parameters:  pidl      - Address of PIDL list
/* Return value : Address of next PIDL and its successors
/*              or NULL, in case no other PIDLs
/*****
LPITEMIDLIST pidlNext(LPCITEMIDLIST pidl)
{
    LPSTR lpMem=(LPSTR)pidl;

    // Get position of next PIDL. With the last PIDL,
    // mkid.cb equals 0
    if( pidl->mkid.cb )
        lpMem += pidl->mkid.cb;
    else return NULL;

    return (LPITEMIDLIST)lpMem;
}

/*****
/* pidlGetSize : Function for getting the size of a PIDL
/*              list
/*-----
/* Parameters:  pidl      - Address of PIDL list
/* Return value : Size of PIDL list in bytes
/*****
UINT pidlGetSize(LPCITEMIDLIST pidl)
{
    UINT cbTotal = 0;
    if( pidl )
        // Was a PIDL passed?
        {
            cbTotal += sizeof( pidl->mkid.cb );
            // With the last PIDL of the list, pidl->mkid.cb equals 0 -----
            while (pidl->mkid.cb)
            {
                cbTotal += pidl->mkid.cb;
                pidl = pidlNext(pidl);
                // Add up PIDL sizes
                // Get next PIDL of list
            }
        }
    return cbTotal;
}

/*****
/* pidlGetNumOfItems : Function for getting number of PIDLs of
/*                    a PIDL list
/*-----
/* Parameters:  pidl      - Address of PIDL list
/* Return value : Number of items in the list
/*****
UINT pidlGetNumOfItems(LPCITEMIDLIST pidl)
{
    UINT cbTotal = 0;
    if( pidl )
        // Was a PIDL passed?
        {
            cbTotal = 1;
            // At least one "blank" item
            // With the last PIDL of the list, pidl->mkid.cb equals 0 -----
            while (pidl->mkid.cb)
            {
                cbTotal++;
                pidl = pidlNext(pidl);
                // Get next PIDL of list
            }
        }
    return cbTotal;
}

```

```

    }
}
return cbTotal;
}

/*****
/* pidlCreate : Function for allocation of a PIDL list with
/*
/* specification of size
/*-----
/* Parameters:      cbSize - Size of desired PIDL list
/* Return value : Address of new PIDL list
/*-----
*****/
LPITEMIDLIST pidlCreate(UINT cbSize)
{
    LPITEMIDLIST pidl=NULL;

    pidl=(LPITEMIDLIST)CoTaskMemAlloc(cbSize);          // Allocate memory...
    if (pidl) memset(pidl, 0, cbSize);                  // ...and null out
    return pidl;
}

/*****
/* pidlConcat : Function for merging two PIDL lists into
/*
/* a third one
/*-----
/* Parameters:      pidl1 - First PIDL list
/*                  pidl2 - Second PIDL list
/* Return value : Address of new PIDL list
/*-----
*****/
LPITEMIDLIST pidlConcat(LPCITEMIDLIST pidl1, LPCITEMIDLIST pidl2)
{
    LPITEMIDLIST pidlNew;
    UINT cb1;
    UINT cb2;

    // Get size of first PIDL list (pidl1 can also be NULL)
    cb1 = pidl1 ? (pidlGetSize(pidl1) - sizeof( pidl1->mkid.cb)): 0;

    // Get size of second PIDL list (pidl2 always != NULL )
    cb2 = pidlGetSize(pidl2);

    // Create new PIDL list of desired size -----
    pidlNew = pidlCreate(cb1 + cb2);

    if (pidlNew)                                     // Able to create new PIDL list?
    {
        // Copy first PIDL to new PIDL list -----
        if (pidl1) memcpy(pidlNew, pidl1, cb1);
        memcpy( (LPVOID)(( DWORD)pidlNew+(DWORD)cb1), pidl2, cb2 );
    }
    return pidlNew;
}

/*****
/* pidlClone : Function for creating a clone PIDL
/*-----
/* Parameters:      pidl - PIDL you are going to make a copy of
/* Return value : Address of new PIDL
/*-----
*****/
LPITEMIDLIST pidlClone( LPITEMIDLIST pidl)
{
    LPITEMIDLIST pidlTemp;

    pidlTemp=(LPITEMIDLIST)CoTaskMemAlloc( pidlGetSize( pidl ) );
    if( pidlTemp )
        CopyMemory((PVOID)pidlTemp,
            (CONST VOID *)pidl,
            pidlGetSize( pidl ));
    return pidlTemp;
}

/*****
/* ExtractStrRet : Gets ANSI string contained in a
/*
/* STRRET
/*-----
/* Parameters:      pIDL - Address of Pidl belonging to
*****/

```

```

/*          STRRET string          */
/*          lpStr    - Address of STRRET string          */
/* Return value : Address of ANSI string or NULL, if error          */
/*-----*/
/* Info : The returned memory must be deallocated again          */
/*          by the caller via CoTaskMemFree().          */
/*-----*/
LPSTR ExtractStrRet( LPITEMIDLIST pIDL,
                    LPSTRRET      lpStr )
{
    LPSTR lpsz = NULL;
    int cch;

    switch( lpStr->uType )                // Get type of STRRET string
    {
        case STRRET_WSTR:                // Wide characters
            // Get number of characters -----
            cch = WideCharToMultiByte( CP_OEMCP,
                                      0,
                                      lpStr->pOleStr,
                                      -1,
                                      NULL,
                                      0,
                                      NULL,
                                      NULL ) + 1;

            // Allocate memory -----
            lpsz = CoTaskMemAlloc( cch * 2 );                // WORD character
            if( lpsz ) // Copy wide characters to lpsz -----
                WideCharToMultiByte( CP_OEMCP,
                                    0,
                                    lpStr->pOleStr,
                                    -1,
                                    lpsz,
                                    cch,
                                    NULL,
                                    NULL );

            break;
        case STRRET_OFFSET:                // Text contained in PIDL
            // The beginning of the first character within PIDL is in lpStr
            cch = lstrlen( ( (char*) pIDL ) + lpStr->uOffset ) + 1;
            // Allocate memory -----
            lpsz = CoTaskMemAlloc( cch );
            if( lpsz )
                strcpy( lpsz, ((char*)pIDL) + lpStr->uOffset);        // Copy text
            break;
        case STRRET_CSTR:                // ordinary C string (how boring!)
            cch = lstrlen( lpStr->cStr ) + 1;                // Get length
            // Allocate memory -----
            lpsz = CoTaskMemAlloc( cch );
            if( lpsz )
                strcpy( lpsz, lpStr->cStr );                // Copy text
            break;
    }
    return lpsz;                // Address of ANSI string
}

/*-----*/
/* CallCallback : Helper function for simplified call of actual          */
/*          callback.          */
/*-----*/
/* Parameters:      pESC      - Address of callback          */
/*          lpFolder    - Address of parent folder          */
/*          pidlComplete - Address of PIDL (starting from          */
/*          Desktop)          */
/*          pidl        - Address of PIDL for folder          */
/*          dwUser      - User data          */
/*          iLevel      - Nesting level          */
/* Return value : TRUE  - continue enumerating          */
/*          FALSE  - end enumeration          */
/*-----*/
/* Info : This function assumes the "burdensome" tasks of getting          */
/*          attributes, putting together PIDL path, getting the path          */
/*-----*/
BOOL CallCallback(ENUMSHELLCALLBACK pESC,
                  LPSHELLFOLDER      lpFolder,

```

```

        LPITEMIDLIST    pidlComplete,
        LPITEMIDLIST    pidl,
        DWORD           dwUser,
        int             iLevel )
{
    DWORD           ulAttrs;                // Attributes of object
    STRRET          sName;                  // for GetDisplayNameOf
    char            szPath[ MAX_PATH ];     // Path, if FS object

    ulAttrs = 0xFFFFFFFF;                  // Get all attributes of object
    if( lpFolder->lpVtbl->GetAttributesOf( lpFolder,
                                           1, // Only 1 object
                                           (const struct _ITEMIDLIST **)&pidl,
                                           &ulAttrs) != NOERROR )
        return FALSE;

    // Get path name of object. If FileSystem object, use -----
    // Desktop path, otherwise use folder relatives path -----
    szPath[ 0 ] = '\\0';
    if( ulAttrs & SFGAO_FILESYSTEM )
        SHGetPathFromIDList( pidlComplete, szPath );
    else
        SHGetPathFromIDList( pidl, szPath );

    // Get DisplayName of object -----
    if( lpFolder->lpVtbl->GetDisplayNameOf( lpFolder,
                                           pidl,
                                           SHGDN_INFOLDER,
                                           &sName ) == NOERROR )
    {
        BOOL bRetVal;
        // Convert DisplayName to ANSI -----
        LPSTR lpszDisplayName = ExtractStrRet( pidl, &sName );

        // Call user-defined Callback -----
        bRetVal = pESC( lpFolder,
                        szPath,
                        lpszDisplayName,
                        ulAttrs,
                        dwUser,
                        pidlComplete,
                        pidl,
                        iLevel );

        if( lpszDisplayName )                // Release DisplayName
            CoTaskMemFree( lpszDisplayName );

        return bRetVal;
    }
    else
        return pESC( lpFolder,
                      szPath,                // No DisplayName available,
                      NULL,                  // call callback anyway
                      ulAttrs,
                      dwUser,
                      pidlComplete,
                      pidl,
                      iLevel );
}

/*****
/* EnumFolderEngine : Starting from a folder, enumerates all the
/* objects contained within and calls user-
/* defined callback function
/*-----
/* Parameters:    lpFolder - IShellFolder whose items are to be
/* enumerated.
/* pidlPath - Address of PIDL path for folder,
/* starting from Desktop
/* pESC - Address of EnumShellCallback
/* dwUser - User data
/* iLevel - Current nesting level
*****/
void WINAPI EnumFolderEngine( LPSHELLFOLDER lpFolder,
                             LPITEMIDLIST pidlPath,
                             ENUMSHELLCALLBACK pESC,

```

```

        DWORD          dwUser,
        int            iLevel )
{
    LPENUMIDLIST lpEnumerator;          // IEnumIDList interface pointer

    // Get enumerator for folder to be enumerated -----
    if( lpFolder->lpVtbl->EnumObjects( lpFolder,
        NULL,
        SHCONTF_FOLDERS|
        SHCONTF_NONFOLDERS|
        SHCONTF_INCLUDEHIDDEN,
        &lpEnumerator)==ERROR_SUCCESS )
    {
        LPITEMIDLIST pidl;
        DWORD          dwFetched;
        BOOL           bContinue;

        bContinue = TRUE;                // Keep on enumerating
        while( ( bContinue ) &&
            // Get next PIDL contained in folder -----
            ( lpEnumerator->lpVtbl->Next( lpEnumerator,
                1,
                &pidl,
                &dwFetched ) == NOERROR ) )
        {
            DWORD          ulAttrs;                // Attributes of object
            LPITEMIDLIST pidlComplete;    // PIDL list, starting from Desktop
            LPSHELLFOLDER lpSubFolder;      // Interface for subfolder

            // Add current PIDL to PIDL list of all predecessors -----
            pidlComplete = pidlConcat( pidlPath, pidl );

            ulAttrs = 0xFFFFFFFF;            // Get all attributes of object
            if( lpFolder->lpVtbl->GetAttributesOf( lpFolder,
                1,                // Only 1 object
                (const struct _ITEMIDLIST **)&pidl,
                &ulAttrs) == NOERROR )
            {
                // Call callback -----
                bContinue = CallCallback(pESC,
                    lpFolder,
                    pidlComplete,
                    pidl,
                    dwUser,
                    iLevel );

                // Continue enumerating and is current -
                // object a folder?
                if( ( bContinue ) && ( ulAttrs & SFGAO_FOLDER ) )
                {
                    // Get interface for SubFolder -----
                    if( lpFolder->lpVtbl->BindToObject( lpFolder,
                        pidl,
                        NULL,
                        &IID_IShellFolder,
                        &lpSubFolder ) == NOERROR )
                    {
                        EnumFolderEngine( lpSubFolder,        // Enumerate objects of
                            pidlComplete,        // SubFolder
                            pESC,
                            dwUser,
                            iLevel + 1 );

                        // Release interface of SubFolder -----
                        lpSubFolder->lpVtbl->Release( lpSubFolder );
                    }
                }
            }
        }
        else bContinue = FALSE;

        CoTaskMemFree( pidl );            // Release memory of current PIDL
        CoTaskMemFree( pidlComplete );    // and PIDL path to current
        // object
        // Release interface of enumerator -----
        lpEnumerator->lpVtbl->Release( lpEnumerator );
    }
}

```

```

}

/*****
/* EnumFolders : Starting from a folder, enumerates all objects
/*               contained within and calls user-defined callback
/*               function
/*-----*/
/* Parameters:   dwFolderID - CSIDL of folder to be enumerated
/*               pESC       - Address of EnumShellCallback
/*               dwUser      - User data
*****/
void WINAPI EnumFolders( DWORD      dwFolderID,
                        ENUMSHELLCALLBACK pESC,
                        DWORD      dwUser )
{
    LPSHELLFOLDER lpDesktop;

    // First get ShellFolder interface of Desktop -----
    if( SHGetDesktopFolder( &lpDesktop ) == ERROR_SUCCESS )
    {
        LPSHELLFOLDER lpSubFolder;
        LPITEMIDLIST pidl;

        // Get PIDL of desired special folder -----
        if( SHGetSpecialFolderLocation( NULL, dwFolderID, &pidl ) == NOERROR )
        {
            // First call of callback returns start object -----
            CallCallback(pESC,
                        lpDesktop,
                        pidl,
                        pidl,
                        dwUser,
                        0 );

            // Get ShellFolder interface for SubFolder determined by
            // PIDL -----
            if( lpDesktop->lpVtbl->BindToObject( lpDesktop,
                                                pidl,
                                                NULL,
                                                &IID_IShellFolder,
                                                &lpSubFolder ) == NOERROR )
            {
                // Enumerate objects of folder -----
                EnumFolderEngine( lpSubFolder, pidl, pESC, dwUser, 1 );
                // Release interface of SubFolder -----
                lpSubFolder->lpVtbl->Release( lpSubFolder );
            }
            else
            {
                // If BindToObject doesn't function properly, -----
                // we assume that the objects of the Desktop -----
                // need to be enumerated, since -----
                // Desktop->BindToObject(..., DESKTOP,...) doesn't work. -----
                EnumFolderEngine( lpDesktop, pidl, pESC, dwUser, 1 );
            }

            CoTaskMemFree( pidl ); // Release memory of pidl
        }
        // Release interface pointer of Desktop -----
        lpDesktop->lpVtbl->Release( lpDesktop );
    }
}

```