Search - Articles

DevASP.Net

Home
Dev Articles
Sample Chapters
Add a Listing
Contact

Dev Articles

ASP.NET (528)
C# (302)
SQL Server (62)
VB.NET (195)
Web Services (6)

Search Directory

ASP.Net AJAX
AJAX ToolKit
.NET Namespaces
Applications
Articles & Samples
Assembly & Controls
Community
Developer Sites
Downloads
Hosting Services
Introduction
Knowledge Base
Sample Chapters
WebCasts

ASP.NET

Applications
Articles & Samples
ASP.Net Sites
Assembly & Controls
Downloads
Errors & Bugs
Introduction
Knowledge Base
Sample Chapters
WebCasts

VB.Net

Articles & Samples
Downloads
Errors, Bugs & Fixes
Introduction
Knowledge Base
Sample Chapters
WebCasts

C-Sharp

Applications
Articles & Samples
C-Sharp Sites
Errors, Bugs & Fixes
Introduction
Knowledge Base
Sample Chapters
WebCasts

SQL Server

Applications
Articles & Samples
SQL Sites
Downloads
Errors, Bugs & Fixes
Introduction
Knowledge Base
Sample Chapters
WebCasts

Traversing Directory Trees Using Win32 System Services

Author: DevASP

Directory traversal is a very common activity in many different types of applications, and there are a number of good reasons to use this capability.

For example, you might need to accomplish any of the following tasks:

1. Delete a directory and any subdirectories within it
2. Search a drive for the biggest file or 100 biggest files
3. Get all of the file names held in a directory so that you can back them up to tape
4. Search for a file or group of files anywhere on a drive
5. Search through directories to find .obj or .tmp files to delete, to free disk space
6. Find the total disk space consumed by a directory tree
7. Search for a specific word in any .txt file you find on a drive
8. Display the directory tree for a drive as a graphic
9. Check a path to ensure that each directory it contains is valid
10. Count the total number of files on a drive

The 32-bit APIs provides a set of three functions that let you easily traverse a directory. Using these same functions recursively, you can traverse entire directory trees. The sample code below demonstrates the use of the directory-walking functions in their simplest form. This code lists all of the files and directorynames found in a single directory.This code performs listing the current directories contents. But before this I have write two functions which display some useful information and get the files information respectively.

```
void PrintFindData(WIN32_FIND_DATA *findData)
{
      // If it's a directory, print the name
      if( findData->dwFileAttributes &FILE_ATTRIBUTE_DIRECTORY )
      {
            CString strMessage;
            strMessage.Format("Directory: %s",findData->cFileName);
            AfxMessageBox(strMessage);
      }
      // else if it's a file, print name and size
      else
      {
            CString strMessage;
            strMessage.Format("This Is a File\n%s \n Size is:%
d",findData->cFileName,findData->nFileSizeLow);
            AfxMessageBox(strMessage);
      }
}
```

The above function will display weather the given parameter is directory or a file id it's a directory then it will display its name and if it's a file then it will display its name and its size in bytes.

```
void ListDirectoryContents(char *fileMask)

{
      HANDLE fileHandle;
      WIN32_FIND_DATA findData;

      // get first file
      fileHandle = FindFirstFile( fileMask,&findData );
      if( fileHandle != INVALID_HANDLE_VALUE )
      {
            PrintFindData( &findData );

            // loop on all remeaining entries in dir
```

DevASP

```
            while( FindNextFile( fileHandle,&findData ) )
            {
                  PrintFindData( &findData );
            }
      }

      FindClose( fileHandle );
}
```

The above function's code will iterate in the current directory and finds all files and directories and send their information to the "PrintFindData" function. So in this way all the files and directories information will be obtained in the current directory. You can use the WIN32_FIND_DATA to find a lot of information about the current founded file or directory. No I will just call the ListDirectoryContents function against a button control.

```
ListDirectoryContents( "*.*" );
```

The **ListDirectoryContents** function starts by calling the API's **FindFirstFile** function.The signature of **FindFirstFile** looks like this.

```
HANDLE FindFirstFile(LPTSTR searchFile,LPWIN32_FIND_DATA findData);
```

searchFile	The file to search for (wild cards are OK)
findData	Information about the file it finds

Returns a search handle to the first matching file found or INVALID_HANDLE_VALUE on failure

The **FindFirstFile** function accepts the name of the file to be found, and returns both a HANDLE to the file if it is found, and a structure describing the file. The file handle is not a normal file handle like the ones produced by **CreateFile** .It is specific to the **Find** functions described in this section. The WIN32_FIND_DATA structure returns the following information (from the API documentation):

```
typedef struct _WIN32_FIND_DATA {
    DWORD dwFileAttributes;
    FILETIME ftCreationTime;
    FILETIME ftLastAccessTime;
    FILETIME ftLastWriteTime;
    DWORD    nFileSizeHigh;
    DWORD    nFileSizeLow;
    DWORD    dwReserved0;
    DWORD    dwReserved1;
    TCHAR    cFileName[ MAX_PATH ];
    TCHAR    cAlternateFileName[ 14 ];
} WIN32_FIND_DATA;
```

In our above code the program is searching for every file in the current directory. It passes the structure returned by **FindFileFirst** to **PrintFindData,** which decides whether or not it is a directory name and also prints out some of the information. The program then continues looking for other files in the directory using the **FindNextFile** function.
The signature of this function looks like this.

```
BOOL FindNextFile(HANDLE findFile,LPWIN32_FIND_DATA finData);
```

findFile	File handle returned by FindFileFirst or Next
findData	Information about the file it finds Returns TRUE on success

FindNextFile accepts a handle produced either by **FindFirstFile** or by a previous call to **FindNextFile.** It finds the next file in the directory that matches the file name description first passed to **FindFirstFile,** and returns a HANDLE and file information on the match. If no match is found, the returned Boolean value will be false, and the **GetLastError** function will contain the error code. Once no match is found, it means that the code has reached the end of the directory. At this point, the program calls **FindClose** to clean up the file handle used by the previous **Find** functions.
The signature of **FindClose** looks like this.

```
BOOL FindClose(HANDLE findFile);
```

findFile	File handle returned by FindFileFirst or Next

Returns TRUE on success
The above code in is suitable for examining all of the files in any single directory. If you want to look at entire directory trees, then you can make slight modifications to the code to make it recursive. In the recursive version, any directory found must in turn be traversed. The code showed below

Recursively traverse a directory tree starting at the current directory. For this I have also write two functions.

```
void PrintFindData1(WIN32_FIND_DATA *findData)
{
      /*cout << "\t";
      cout << findData->cFileName;
      cout << "\t(" << findData->nFileSizeLow << ")";
      cout << endl;*/
      CString strMessage;
      strMessage.Format("%s\nFile size: %d",findData->cFileName,
                   findData->nFileSizeLow);
      AfxMessageBox(strMessage);
}
```

The above function will print the file name and its size in bytes, which is provided it by the poarameters.this function will be called by the ListDirectoryContents1() function listed below.

```
void ListDirectoryContents1(char *dirName,char *fileMask)
{
      char *fileName;
      char curDir[ 256 ];
      char fullName[ 256 ];
      HANDLE fileHandle;
      WIN32_FIND_DATA findData;

      // save current dir so it can restore it
      if( !GetCurrentDirectory( 256, curDir) )
            return;

      // if the directory name is neither . or .. then
      // change to it, otherwise ignore it
      if( strcmp( dirName, "." ) &&
            strcmp( dirName, ".." ) )
      {
            if( !SetCurrentDirectory( dirName ) )
                  return;
      }
      else
            return;

      // print out the current directory name
      if( !GetFullPathName( fileMask, 256, fullName,&fileName ) )
            return;
      CString strInfo;
      strInfo.Format("Directory - %s",fullName);
      AfxMessageBox(strInfo);

      // Loop through all files in the directory
      fileHandle = FindFirstFile( fileMask,
            &findData );
      while ( fileHandle != INVALID_HANDLE_VALUE )
      {
            // If the name is a directory,
            // recursively walk it. Otherwise
            // print the file's data
            if( findData.dwFileAttributes &
                  FILE_ATTRIBUTE_DIRECTORY )
            {
                  ListDirectoryContents1( findData.cFileName,
                        fileMask );
            }
            else
                  PrintFindData1( &findData );

            // loop thru remaining entries in the dir
            if (!FindNextFile( fileHandle, &findData ))
                  break;
      }

      // clean up and restore directory
      FindClose( fileHandle );
      SetCurrentDirectory( curDir );
}
```

This function will recursively search each directory in the current directory. Now i have called these functions against the button click like this.

```
char curDir[ 256 ];

      if( !GetCurrentDirectory( 256, curDir ) )
```

```
        {
                AfxMessageBox("Could not Get the Current Directory");
                return;
        }

        // List all files, starting with the
        // current directory
        ListDirectoryContents1( curDir, "*.*" );
```

The main differences between the listing in the start of the article and the above lies in the code's treatment of directories. The **ListDirectoryContents1** function accepts two parameters, one for the file name and one for the current directory. It changes into the specified directory and begins looking for files using **FindFileFirst** and **FindNextFile.** If it encounters any directories during its search, it recursively calls itself to probe the new directory. This process continues until all of the files in the subtree have been examined.

Add Article Comment:

Name :	
Email Address :	
Comments :	

☐ Send me an email about this article when other users post a Comment

Submit Clear All

<< Searching for a file Using Win32 System Services

Monitoring Window services in local and remote machine >>