Obtaining Directory Change Notifications

An application can monitor the contents of a directory and its subdirectories by using change notifications. Waiting for a change notification is similar to having a read operation pending against a directory and, if necessary, its subdirectories. When something changes within the directory being watched, the read operation is completed. For example, an application can use these functions to update a directory listing whenever a file name within the monitored directory changes.

An application can specify a set of conditions that trigger a change notification by using the **FindFirstChangeNotification** function. The conditions include changes to file names, directory names, attributes, file size, time of last write, and security. This function also returns a handle that can be waited on by using the wait functions. If the wait condition is satisfied, **FindNextChangeNotification** can be used to provide a notification handle to wait on subsequent changes. However, these functions do not indicate the actual change that satisfied the wait condition.

Use **FindCloseChangeNotification** to close the notification handle.

To retrieve information about the specific change as part of the notification, use the **ReadDirectoryChangesW** function. This function also enables you to provide a completion routine.

To track changes on a volume, see change journals.

The following example monitors the directory tree for directory name changes. It also monitors a directory for file name changes. The example uses the **FindFirstChangeNotification** function to create two notification handles and the **WaitForMultipleObjects** function to wait on the handles. Whenever a directory is created or deleted in the tree, the example should update the entire directory tree. Whenever a file is created or deleted in the directory, the example should refresh the directory.

Note
This simplistic example uses the **ExitProcess** function for termination and cleanup, but more complex applications should always use proper resource management such as **FindCloseChangeNotification** where appropriate.

C++

```
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <tchar.h>

void RefreshDirectory(LPTSTR);
void RefreshTree(LPTSTR);
void WatchDirectory(LPTSTR);

void _tmain(int argc, TCHAR *argv[])
{
    if(argc != 2)
    {
        _tprintf(TEXT("Usage: %s <dir>\n"), argv[0]);
        return;
    }

    WatchDirectory(argv[1]);
}

void WatchDirectory(LPTSTR lpDir)
{
   DWORD dwWaitStatus;
   HANDLE dwChangeHandles[2];
   TCHAR lpDrive[4];
   TCHAR lpFile[_MAX_FNAME];
   TCHAR lpExt[_MAX_EXT];

   _tsplitpath_s(lpDir, lpDrive, 4, NULL, 0, lpFile, _MAX_FNAME, lpExt, _MAX_

   lpDrive[2] = (TCHAR)'\\';
   lpDrive[3] = (TCHAR)'\0';

// Watch the directory for file creation and deletion.

   dwChangeHandles[0] = FindFirstChangeNotification(
      lpDir,                         // directory to watch
      FALSE,                         // do not watch subtree
      FILE_NOTIFY_CHANGE_FILE_NAME); // watch file name changes

   if (dwChangeHandles[0] == INVALID_HANDLE_VALUE)
   {
     printf("\n ERROR: FindFirstChangeNotification function failed.\n");
     ExitProcess(GetLastError());
   }

// Watch the subtree for directory creation and deletion.
```

```
   dwChangeHandles[1] = FindFirstChangeNotification(
      lpDrive,                       // directory to watch
      TRUE,                          // watch the subtree
      FILE_NOTIFY_CHANGE_DIR_NAME);  // watch dir name changes

   if (dwChangeHandles[1] == INVALID_HANDLE_VALUE)
   {
     printf("\n ERROR: FindFirstChangeNotification function failed.\n");
     ExitProcess(GetLastError());
   }


// Make a final validation check on our handles.

   if ((dwChangeHandles[0] == NULL) || (dwChangeHandles[1] == NULL))
   {
     printf("\n ERROR: Unexpected NULL from FindFirstChangeNotification.\n");
     ExitProcess(GetLastError());
   }

// Change notification is set. Now wait on both notification
// handles and refresh accordingly.

   while (TRUE)
   {
   // Wait for notification.

      printf("\nWaiting for notification...\n");

      dwWaitStatus = WaitForMultipleObjects(2, dwChangeHandles,
         FALSE, INFINITE);

      switch (dwWaitStatus)
      {
         case WAIT_OBJECT_0:

         // A file was created, renamed, or deleted in the directory.
         // Refresh this directory and restart the notification.

             RefreshDirectory(lpDir);
             if ( FindNextChangeNotification(dwChangeHandles[0]) == FALSE )
             {
               printf("\n ERROR: FindNextChangeNotification function failed.\
               ExitProcess(GetLastError());
             }
             break;

         case WAIT_OBJECT_0 + 1:
```

```
            // A directory was created, renamed, or deleted.
            // Refresh the tree and restart the notification.

                RefreshTree(lpDrive);
                if (FindNextChangeNotification(dwChangeHandles[1]) == FALSE )
                {
                  printf("\n ERROR: FindNextChangeNotification function failed.\
                  ExitProcess(GetLastError());
                }
                break;

            case WAIT_TIMEOUT:

            // A timeout occurred, this would happen if some value other
            // than INFINITE is used in the Wait call and no changes occur.
            // In a single-threaded environment you might not want an
            // INFINITE wait.

               printf("\nNo changes in the timeout period.\n");
               break;

            default:
               printf("\n ERROR: Unhandled dwWaitStatus.\n");
               ExitProcess(GetLastError());
               break;
         }
      }
   }

void RefreshDirectory(LPTSTR lpDir)
{
   // This is where you might place code to refresh your
   // directory listing, but not the subtree because it
   // would not be necessary.

   _tprintf(TEXT("Directory (%s) changed.\n"), lpDir);
}

void RefreshTree(LPTSTR lpDrive)
{
   // This is where you might place code to refresh your
   // directory listing, including the subtree.

   _tprintf(TEXT("Directory tree (%s) changed.\n"), lpDrive);
}
```

Community Additions

Beware Empty Directories

Beware empty folders getting moved or renamed. I am seeing WaitForMultipleObjects() getting stuck on file change notification handle if the directory being watched is moved or renamed, if that directory is empty.


Henri Hein
1/15/2013

Unclear comment

```
// In a single-threaded environment you might not want an
 INFINITE wait.
```

The above comment in the example is misleading.
Whether the environment is single-threaded or multi-threaded the waiting thread is suspended and does not use the processor's time. This is the whole point of waiting. Otherwise we could just use an infinite loop checking repeatedly for a condition to be met.


Franciszek Czekała
10/11/2012

Either Or

To be clear, FindFirst/NextChangeNotification and ReadDirectoryChanges appear to be two distinct ways to discover directory changes. It's easy to get the impression from this article that you could wait on the handle returned by the Find function and then get the details using the Read function synchronously. That's not the way it works.


aidtopia
8/17/2011