A Cleanup API for Windows

By Marcel Lambert, 24 Aug 2006

- **Download demo project - 177 KB**

Introduction

When you opt to keep in private your working and internet browsing habits, you can install software tools that cleanup typical files reflecting these habits such as temporary internet files, cookies, internet history, recent document traces etc. In contrast, you can write own clean up program in a snap, using some general cleanup API, which hides details in the background. This article provides a simple cleanup API ported into a Win32 DLL and into an equivalent COM DLL. One practical example of this cleanup interface implementation is the program of automating computer cleanup tasks, which never forgets to destroy private information upon you leaving a computer.

If you are interested in the how-to and testing aspects beforehand, I suggest you go to the How to Use section at once.

Contents

Internet Cache Cleanup

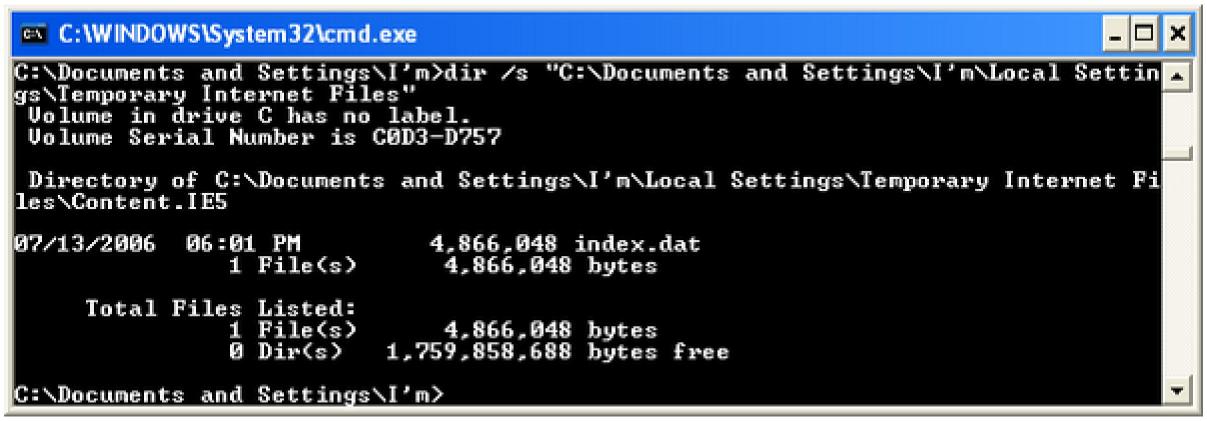
Typically, all internet files browsed on the internet with IE are stored (cached) in the *Temporary Internet Files* folder. This speeds up future browsing. The internet cache is located at *%**userprofile**%\Local Settings\Temporary Internet Files*. If not deleted, these files provide information about our working and browsing habits. At first glance, the cleanup of internet cache, in particular (and computer cleanup, in general), might seem straightforward because, indeed, we only need to empty (or delete) some specific folders or files. Let's recap, however, some typical pitfalls.

Typical Pitfalls

Pitfall #1: Use of Internet Explorer's own tools. Why cleanup programmatically if we can choose Delete Files/Cookies in Internet Options of IE? First, the Internet Options utility is misleading you a little bit. In fact, it does not cleanup all the files and folders in the *Temporary Internet Files* folder. Although after running cleanup, Windows Explorer shows an empty list view control, we'll see different results when trying the command prompt:

⊟Collapse | Copy Code

```
dir /s “C:\Documents and Settings\I'm\Local Settings\Temporary Internet Files”
```



```
C:\Documents and Settings\I'm>dir /s "C:\Documents and Settings\I'm\Local Settin
gs\Temporary Internet Files"
 Volume in drive C has no label.
 Volume Serial Number is C0D3-D757

 Directory of C:\Documents and Settings\I'm\Local Settings\Temporary Internet Fi
les\Content.IE5

07/13/2006  06:01 PM         4,866,048 index.dat
               1 File(s)      4,866,048 bytes

     Total Files Listed:
               1 File(s)      4,866,048 bytes
               0 Dir(s)   1,759,858,688 bytes free

C:\Documents and Settings\I'm>
```

The picture shows that after the IE cleanup, the *Content.IE5* folder and the *index.dat* file still exist. Why is it that the Windows Explorer shows an empty list in opposition to this? Because, *Temporary Internet Files*, as you are likely aware, is a *special folder* (I'll touch this subject later) and Windows Explorer provides a special UI for *special folders*.

By the way, you can see the full content of the internet cache using the following trick.

- Open Windows Explorer.
- Copy-and-paste into the address area: *C:\Documents and Settings\I'm\Local Settings\Temporary Internet Files\Content.IE5*, and press Enter (or place the shortcut on the desktop and provide this quoted path as the target for the shortcut).

The second reason to implement a programmatic cleanup is that Windows does not provide an interface to automate cleanup tasks.

Pitfall #2: Manual deletion with Windows Explorer. Trying to delete the internet cache folder in Windows Explorer with keyboard's Delete (with an idea to re-create it later on). In response, Windows prompts a message saying that:

⊟Collapse | Copy Code

```
Temporary Internet Files is a Windows system folder and
it is required for Windows to run properly. It cannot be deleted.
```

This is again because the *Temporary Internet Files* folder is a Windows *special folder* (identified by `CSIDL_INTERNET_CACHE` and `PIDL`), and belongs to the `shell` namespace (managed by the COM-style API) rather than to the file system. In contrast to the file system folders, *special folders* cannot be deleted/created manually or programmatically (in general).

The `shell` namespace is more inclusive than the file system, i.e., it contains not only the NTFS folders, and folders like *My Documents*, *History* etc., which represent physical storage, but also virtual folders such as the *Printers* folder, which represents links to printers which are not physical storage. All file system objects can be managed by the `shell` namespace, but not vice versa, because the file system, as Microsoft says, is a subset of a broader `shell` namespace.

Pitfall #3: Command line deletion. Trying to delete the contents of the internet cache folder in the command prompt with **del** (or the core Win32 API). Seemingly, we can try the most direct programmatic method like this:

⊟Collapse | Copy Code

```
del /s /q “C:\Documents and
          Settings\I'm\Local Settings\Temporary Internet Files”
```

In the result, we'll see a lot of messages like:

⊟Collapse | Copy Code

```
The process cannot access the file because it is being used by another process.
```

As we'll see later, this is because some of internet cache files/folders are locked by another processes.

Pitfall #4: Choice of windows API. As described in the MSDN documentation, the Shell API can be used to manage (including deletion) the content of *special folders*. In general, you translate a *special folder*'s COM-style ID (`CSIDL_INTERNET_CACHE`, `CSIDL_HISTORY` etc.) to a folder's PIDL (shell namespace ID) with `SHGetSpecialFolderLocation`. Next, using PIDL, you get `IShellFolder`'s pointer to this folder. With `IShellFolder`'s pointer, you can un-scroll interfaces, further to enumerate entries and make deletions finally. However, when deleting entries in the *History* folder (`IShellFolder`'s pointer already obtained), MSDN recommends (**Q327569**: "`DeleteUrl()` Does Not Delete the Internet Explorer History Folder Entry.") to use the `IContextMenu::InvokeCommand` method to delete *History* items, which has a disadvantage because "you cannot disable the confirmation dialog box that appears." (same article), which is unacceptable for UI-less cleanup API.

Would we be more successful if we try to delete the internet cache files/folders with Win32's `DeleteFile`/`DeleteDirectory`? This works perfectly for ordinary folders: we need to enumerate files/subdirectories in the directory with `FindFirstFile`/`FindNextFile`, delete them one by one, and use recursion for deletion if a subdirectory found. In my experiments, however, running cleanup with Win32 functions corrupts the view of Windows Explorer for the internet cache folder, although the view is auto-restored (the subfolder *Content.IE5* and its file *index.dat* are not deleted).

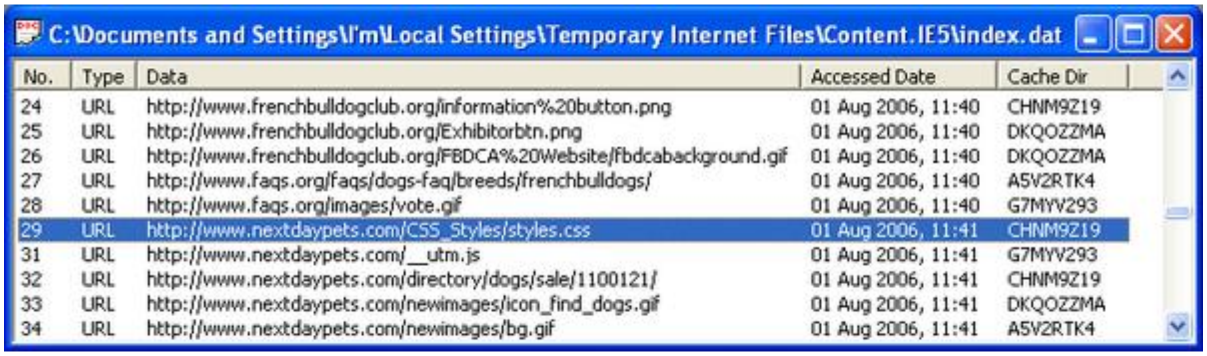
When doing internet cache cleanup, it is important to notice that it is managed by the WinInet library, and therefore other APIs (like the Win32 core API) have no guarantee to succeed. In contrast to other libraries, WinInet provides a straightforward subset, which is designed exactly for managing the content of internet cache, which includes all logical level files stored in the *Temporary Internet Files*, *Cookies*, and *History* folders.

Having said that, I am using for cleanup, in most cases, the WinInet API (instead of the Shell API or Win32 API). In the following sections, I'll discuss briefly every function in this API.

Internet Cache's index.dat Removal

Windows employs *index.dat* files for indexing the content of some folders (internet folders for the most part) to make subsequent accesses faster. As noticed earlier, cleanup with IE's Internet Options utility does not make full cleanup of the internet cache. In opposite to the empty content shown by Windows Explorer, the ***dir*** */s* command shows that the *Temporary Internet Files* folder content is not actually empty, and contains the *Content.IE5* folder (plus subfolders) and the *index.dat* file. This is because folders like Recent Documents, *History*, *Temporary Internet Files*, and some others are managed in Windows by Windows Shell, they are *special folders*.

Why bother about *index.dat* files at all? In fact, unhandled *index.dat* files represent a personal security breach. These files contain a list of all internet places visited, and can be viewed, copied etc. For example, suppose we don't want that private information about our favorite puppies (found out with IE on the internet) be recorded permanently on our computer. We empty the internet cache (i.e., remove physically cached web pages), but leave *index.dat* untouched. At first glance, our privacy has been protected. In contract to this conjecture, here is the screenshot of a typical *index.dat* viewer program output:


C:\Documents and Settings\I'm\Local Settings\Temporary Internet Files\Content.IE5\index.dat

No.	Type	Data	Accessed Date	Cache Dir
24	URL	http://www.frenchbulldogclub.org/information%20button.png	01 Aug 2006, 11:40	CHNM9Z19
25	URL	http://www.frenchbulldogclub.org/Exhibitorbtn.png	01 Aug 2006, 11:40	DKQOZZMA
26	URL	http://www.frenchbulldogclub.org/FBDCA%20Website/fbdcabackground.gif	01 Aug 2006, 11:40	DKQOZZMA
27	URL	http://www.faqs.org/faqs/dogs-faq/breeds/frenchbulldogs/	01 Aug 2006, 11:40	A5V2RTK4
28	URL	http://www.faqs.org/images/vote.gif	01 Aug 2006, 11:40	G7MYV293
29	URL	http://www.nextdaypets.com/CSS_Styles/styles.css	01 Aug 2006, 11:41	CHNM9Z19
31	URL	http://www.nextdaypets.com/__utm.js	01 Aug 2006, 11:41	G7MYV293
32	URL	http://www.nextdaypets.com/directory/dogs/sale/1100121/	01 Aug 2006, 11:41	CHNM9Z19
33	URL	http://www.nextdaypets.com/newimages/icon_find_dogs.gif	01 Aug 2006, 11:41	DKQOZZMA
34	URL	http://www.nextdaypets.com/newimages/bg.gif	01 Aug 2006, 11:41	A5V2RTK4

Indeed, *index.dat* provides an outlook of visited internet sites. Further, for the sake of experiment, you can try to copy an old *index.dat* onto a clean system, by replacing the existing *index.dat*, to see that the *Temporary Internet Files* folder is, in fact, a viewer of *index.dat*: no web content cached (internet browser has never been used), but the *TIF* folder shows many URLs (i.e., *index.dat*'s content). Obviously, this is a loophole for malicious users who can copy the single file to see the browsing habits of a targeted person. Generally, there are three major *index.dat* files that keep a track of visited sites:

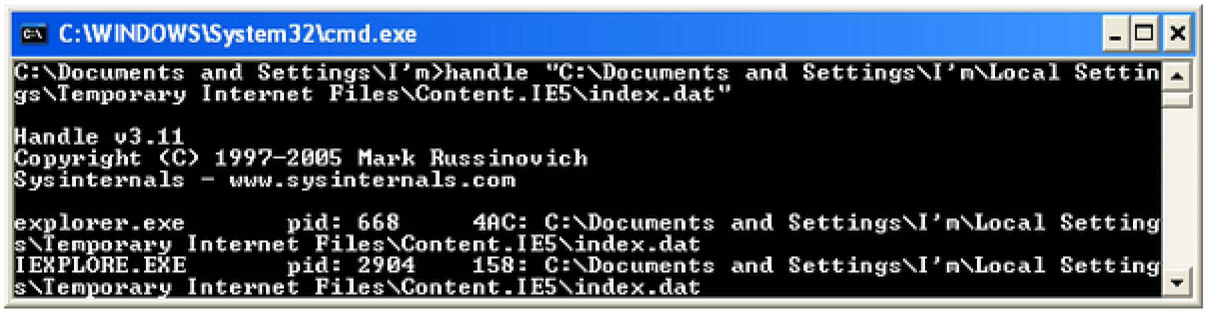
- Internet Cache's *index.dat* located at *%**userprofile**%\Local Settings\Temporary Internet Files\Content.IE5*.
- Internet History's *index.dat* located at *%**userprofile**%\Local Settings\History\History.IE5*.
- Internet Cookies' *index.dat* located at *%**userprofile**%\Cookies*.

Because the deletion of any *index.dat* file goes in the same manner, I'd focus on the details on the subject of deleting cache's *index.dat*. Apparently, Win32's `DeleteFile` can be used to delete *index.dat* programmatically. Let's try, however, to delete *index.dat* manually with the equivalent **del** command. We'd see the infamous problem that index.dat cannot be deleted because:

⊟Collapse | Copy Code

```
The process cannot access the file because it is being used by another process.
```

I've seen a lot of questions in newsgroups regarding the problem of how to delete this file, and why it cannot be deleted in a regular way. In fact, this is a typical situation when file/folder handles are used by another process. Commonly, we need to release these file handles by killing the relevant process. What processes obstruct us to delete? There is Mark Russinovich's **handle** utility, which shows what running Windows processes have open handles on a particular file or folder. The following picture shows the output of this utility (when IE is running):


C:\WINDOWS\System32\cmd.exe

```
C:\Documents and Settings\I'm>handle "C:\Documents and Settings\I'm\Local Settin
gs\Temporary Internet Files\Content.IE5\index.dat"

Handle v3.11
Copyright (C) 1997-2005 Mark Russinovich
Sysinternals - www.sysinternals.com

explorer.exe       pid: 668      4AC: C:\Documents and Settings\I'm\Local Setting
s\Temporary Internet Files\Content.IE5\index.dat
IEXPLORE.EXE       pid: 2904     158: C:\Documents and Settings\I'm\Local Setting
s\Temporary Internet Files\Content.IE5\index.dat
```

Notably, it is better to provide a full quoted path for the `handle` argument, otherwise the utility is prone to output irrelevant processes.

It is immediately clear that Windows Explorer (*explorer.exe*) and Internet Explorer (*iexplore.exe*) have open handles on *index.dat*, obstructing us to delete this file manually with **del**. Obviously, one should kill the *explorer.exe* and *iexplore.exe* processes and then use *del* again. Notice that other processes might have open handles on *index.dat*, but if this were the case, the *handle* utility would show them on your computer. In particular, if Windows Messenger, which comes with SP2, was ever used on Windows XP, the Messenger's process *msmsgs.exe* is auto-started, by default, on every computer startup. The *msmsgs.exe* also locks *index.dat* files. Of course, the list of processes that might lock *index.dat* files is not exhaustive. For example, Sony VAIO notebooks, which come with pre-installed Sony software, run additional process(es) that lock *index.dat* files by default.

The resulting manual solution to delete *index.dat* is the following:

- Run Windows Task Manager and kill *explorer.exe* and *iexplore.exe*.

- In Windows Task Manager, launch the command prompt, navigate to the *index.dat* location, and delete it with the **del** command.
- Run in the command prompt *explorer.exe* to restore shell.

In the same manner, *index.dat* can be deleted manually, starting Windows in safe mode, because in safe mode, processes and services, which might lock this file, are not yet loaded.

Delete_IECache Function

So far, I've concentrated on some pitfalls that occur while emptying the IE cache, and, particularly, described manual steps to delete *index.dat*. What about a programmatic solution? Leave the *index.dat* alone, the deletion of internet cache is pretty easy and can be programmed with WinInet functions. How can we delete *index.dat* programmatically? Ostensibly, Win32's `DeleteFile` can be used in a proper time frame during Windows startup when processes/services have not yet opened handles on this file, or the open handles should be released forcefully, and `DeleteFile` used in the same way. For this matter, the cleanup API provides the function `Delete_IECache` (see also the next section) to delete *index.dat* that you can use by setting the `bDeleteCacheIndex` parameter to `TRUE` (`FALSE`, by default) if *index.dat* is not locked:

⊟Collapse | Copy Code

```
BOOL Delete_IECache(BOOL bDeleteCache = TRUE, BOOL bDeleteCacheIndex = FALSE);
```

More importantly, the `bDeleteCacheIndex = TRUE` switch can be used if you are sure that other processes do not have open handles on that file. Otherwise, the *index.dat* deletion silently fails.

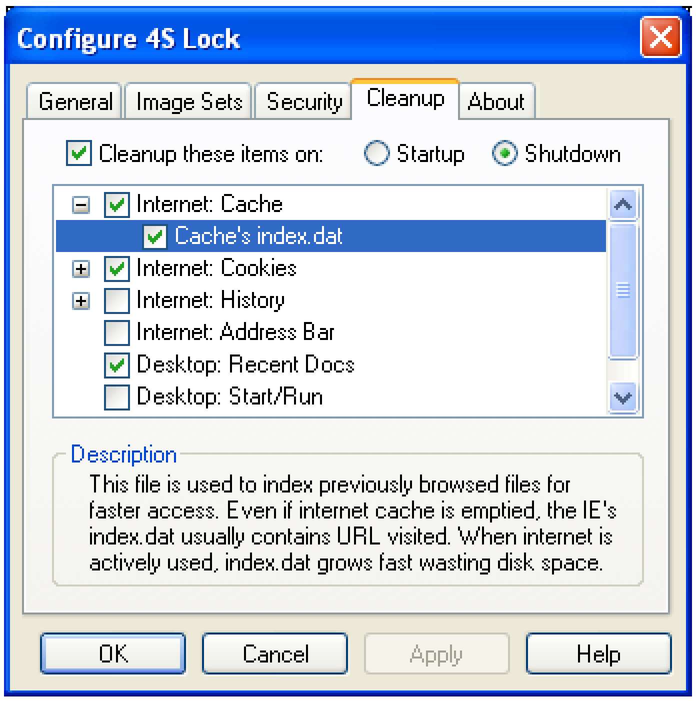
The code below represents the `Delete_IECache`'s function body implemented with WinInet functions. The code is the programmatic equivalent of Delete Files in the Internet Options of IE, and enables you to delete the cache's physical content (HTML pages, pictures, scripts etc.) and its *index.dat* file. Basically, all the work is done by three functions `FindFirstUrlCacheEntry`, `FindNextUrlCacheEntry`, and `DeleteUrlCacheEntry`. We obtain a handle to the internet cache `hCacheEnumHandle` with `FindFirstUrlCacheEntry`, and having this handle obtains pointers to the next cache entries with `FindNextUrlCacheEntry`, deleting them with `DeleteUrlCacheEntry`. One thing to note is that we should adjust the size of the `LPINTERNET_CACHE_ENTRY_INFO` structure before making a call to `DeleteUrlCacheEntry`. This is simple because the correct size is returned in the last parameter (in/out parameter) of `FindNextUrlCacheEntry`.

⊟Collapse | Copy Code

```
__declspec( dllexport ) BOOL Delete_IECache(bDeleteCache =
                        TRUE, BOOL bDeleteCacheIndex = FALSE)
{
    TCHAR szUserProfile[200];
    TCHAR szFilePath[200];
    HANDLE hCacheEnumHandle  = NULL;
    LPINTERNET_CACHE_ENTRY_INFO lpCacheEntry = NULL;
    DWORD dwSize = 4096; // initial buffer size

    // Delete index.dat if requested.
    // Be sure that index.dat is not locked.
    if (bDeleteCacheIndex)
    {
        // Retrieve from environment user profile path.
        ExpandEnvironmentStrings("%userprofile%", szUserProfile,
                                 sizeof(szUserProfile));
        wsprintf(szFilePath, "%s%s", szUserProfile,
                 "\\Local Settings\\Temporary Internet"
                 " Files\\Content.IE5\\index.dat");

        DeleteFile(szFilePath);

        if (!bDeleteCache) return TRUE;
    }

    // Enable initial buffer size for cache entry structure.
    lpCacheEntry = (LPINTERNET_CACHE_ENTRY_INFO) new char[dwSize];
    lpCacheEntry->dwStructSize = dwSize;

    // URL search pattern (1st parameter)
    // options are:  NULL ("*.*"), "cookie:"
    // or "visited:".
    hCacheEnumHandle = FindFirstUrlCacheEntry(NULL /* in */ ,
                       lpCacheEntry /* out */,
                       &dwSize /* in, out */);

    // First, obtain handle to internet cache
    // with FindFirstUrlCacheEntry
    // for later use with FindNextUrlCacheEntry.
    if (hCacheEnumHandle != NULL)
    {
        // When cache entry is not a cookie, delete entry.
        if (!(lpCacheEntry->CacheEntryType & COOKIE_CACHE_ENTRY))
        {
            DeleteUrlCacheEntry(lpCacheEntry->lpszSourceUrlName);
        }
    }
    else
    {
        switch (GetLastError())
        {
            case ERROR_INSUFFICIENT_BUFFER:
            lpCacheEntry = (LPINTERNET_CACHE_ENTRY_INFO) new char[dwSize];
            lpCacheEntry->dwStructSize = dwSize;

            // Repeat first step search with adjusted buffer, exit if not
            // found again (in practice one buffer's size adustment is
            // always OK).
            hCacheEnumHandle = FindFirstUrlCacheEntry(NULL, lpCacheEntry,
                                                      &dwSize);
            if (hCacheEnumHandle != NULL)
            {
                // When cache entry is not a cookie, delete entry.
                if (!(lpCacheEntry->CacheEntryType & COOKIE_CACHE_ENTRY))
                {
                         DeleteUrlCacheEntry(lpCacheEntry->lpszSourceUrlName);
                     }
           break;
            }
            else
            {
                // FindFirstUrlCacheEntry fails again, return.
                return FALSE;
            }
        default:
            FindCloseUrlCache(hCacheEnumHandle);
            return FALSE;
    }
    }

    // Next, use hCacheEnumHandle obtained
    // from the previous step to delete
    // subsequent items of cache.
    do
    {
         // Notice that return values of FindNextUrlCacheEntry (BOOL) and
         // FindFirstUrlCacheEntry (HANDLE) are different.
         if (FindNextUrlCacheEntry(hCacheEnumHandle, lpCacheEntry, &dwSize))
         {
             // When cache entry is not a cookie, delete entry.
             if (!(lpCacheEntry->CacheEntryType & COOKIE_CACHE_ENTRY))
             {
                 DeleteUrlCacheEntry(lpCacheEntry->lpszSourceUrlName);
             }
         }
         else
         {
            switch(GetLastError())
            {
                case ERROR_INSUFFICIENT_BUFFER:
                lpCacheEntry =
                  (LPINTERNET_CACHE_ENTRY_INFO) new char[dwSize];
                lpCacheEntry->dwStructSize = dwSize;

                // Repeat next step search with adjusted buffer, exit if
                // error comes up again ((in practice one buffer's size
                // adustment is always OK).
                if (FindNextUrlCacheEntry(hCacheEnumHandle, lpCacheEntry,
                                           &dwSize))
                {
                     // When cache entry is not a cookie, delete entry.
                     if (!(lpCacheEntry->CacheEntryType &
                           COOKIE_CACHE_ENTRY))
                     {
                         DeleteUrlCacheEntry(
                           lpCacheEntry->lpszSourceUrlName);
                     }
                     break;
                }
                else
                {
                    // FindFirstUrlCacheEntry fails again, return.
                    FindCloseUrlCache(hCacheEnumHandle);
                    return FALSE;
                }
                break;
                case ERROR_NO_MORE_ITEMS:
                FindCloseUrlCache(hCacheEnumHandle);
                return TRUE;
                default:
```

```
                FindCloseUrlCache(hCacheEnumHandle);
                return FALSE;
            }
        }
    } while (TRUE);
}
```

An example of the cleanup API implementation is the Synaptex 4S Lock 1.08 program, which uses the `Delete_IECache` function in its cleanup module to delete *index.dat* in a startup time frame or during the shutdown to automate cleanup. The following picture shows a cleanup interface of 4S Lock 1.08, which can be downloaded (4.7 MB) from here:



[✓] Cleanup these items on: ( ) Startup [✓ Shutdown]

- [✓] Internet: Cache
  - [✓] Cache's index.dat
- [✓] Internet: Cookies
- [ ] Internet: History
- [ ] Internet: Address Bar
- [✓] Desktop: Recent Docs
- [ ] Desktop: Start/Run

Description

This file is used to index previously browsed files for faster access. Even if internet cache is emptied, the IE's index.dat usually contains URL visited. When internet is actively used, index.dat grows fast wasting disk space.

Internet Cookies Cleanup

Cookies store user-specific information such as names and passwords used to access web pages, and, in more general terms, are frequently used to store web page customization information. Though they enable us not to re-enter our own credentials, a private data stored in them can represent a security loophole.

In fact, cookies are part of internet cache located at the *%**userprofile**%\Local Settings\Temporary Internet Files* folder, although Windows stores them physically in a separate location at *%**userprofile**%\Cookies*. Because the internet cache is managed by the WinInet library, `FindFirstUrlCacheEntry`/`FindNextUrlCacheEntry` locates cookies as well, and they can be deleted with `DeleteUrlCacheEntry` as other cache entries. To access previously stored cookies faster, Windows indexes cookie files in Cookies's *index.dat* file located at *%**userprofile**%\Cookies*. Even if cookies are deleted, the *index.dat* usually contains a list of cookies.

Accordingly, the cleanup API provides the `Delete_IECookies` function, where `bDeleteCookies` and `bDeleteCookiesIndex` should be set depending on the user's choice. Be sure that Cookies' *index.dat* is not locked before setting `bDeleteCookiesIndex` to `TRUE`.

⊟Collapse | Copy Code

```
BOOL Delete_IECookies(BOOL bDeleteCookies = TRUE,
                      BOOL bDeleteCookiesIndex = FALSE);
```

The code for the `Delete_IECookies` function is almost identical to `Delete_IECache`, but while enumerating cookies, I use "cookie:" (instead of `NULL`) as the first parameter for the `FindFirstUrlCacheEntry` function.

Internet History Cleanup

The Internet History is the list of previously visited internet sites reflecting your personal internet browsing history, which you can see by clicking the "History" button in the Internet Explorer. Internet History is located at *%**userprofile**%\Local Settings\History* and has the hidden *History.IE5* subfolder containing History's *index.dat*.

Like Internet Cache's content, the full content of the History folder is better seen under the command prompt, which shows hidden folders and files (*History.IE5* and *index.dat*). Trying to empty the History folder, including *History.IE5* and *index.dat*, causes the problems we've seen when playing with the deletion of TIF (*Temporary Internet Files*) in the previous sections. In general, hidden History content is locked by running processes, and cannot be deleted until the locks are released (manually or programmatically) or deletion occurs in a proper time frame when History's objects are not yet locked. As before, trying to delete Internet History with Internet Options/Clear History of IE, **del** /s, or Win32's `DeleteFile`/`DeleteDirectory` (see **pitfalls** #1-4 of the previous section) brings up incomplete results.

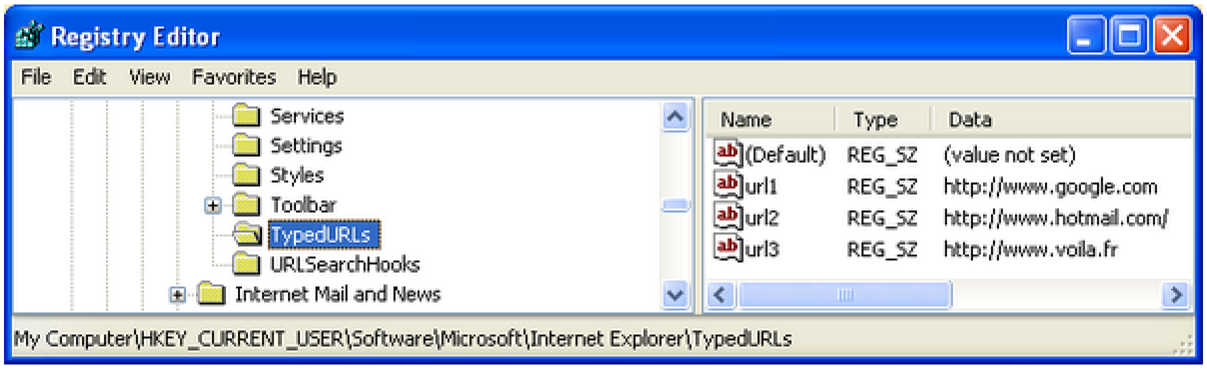
Here is the code to cleanup internet history, including History's *index.dat* removal. Notice that you should be sure that *index.dat* is not locked before setting `bDeleteIndex` to `TRUE`.

⊟Collapse | Copy Code

```
__declspec( dllexport ) HRESULT Delete_IEHistory(BOOL bDeleteHistory = TRUE,
                                     BOOL bDeleteHistoryIndex = FALSE)
{
    TCHAR szUserProfile[200];
    TCHAR szFilePath[200];
    HRESULT hr;

    // Delete index.dat if requested.
    // Be sure that index.dat is not locked.
    if (bDeleteHistoryIndex)
    {
        // Retrieve from environment user profile path.
        ExpandEnvironmentStrings("%userprofile%", szUserProfile,
                                 sizeof(szUserProfile));
        wsprintf(szFilePath, "%s%s", szUserProfile,
                              "\\Local Settings\\History"
                              "\\History.IE5\\index.dat");
        DeleteFile(szFilePath);

        if (!bDeleteHistoryIndex) return S_OK;
    }

    CoInitialize(NULL);

    IUrlHistoryStg2* pUrlHistoryStg2 = NULL;
    hr = CoCreateInstance(CLSID_CUrlHistory, NULL, CLSCTX_INPROC,
                          IID_IUrlHistoryStg2,
                          (void**)&pUrlHistoryStg2);
    if (SUCCEEDED(hr))
    {
        hr = pUrlHistoryStg2->ClearHistory();
        pUrlHistoryStg2->Release();
    }

    CoUninitialize();

    return hr;
}
```

IE's Address Bar History Cleanup

URLs previously typed in the address bar of IE are known as typed URLs, and are visible by expanding the address bar's drop down list. They are recorded to registry as url1, url2, url3 etc., at *HKCU\Software\Microsoft\Internet Explorer\TypedURLs*:



Name	Type	Data
(Default)	REG_SZ	(value not set)
url1	REG_SZ	http://www.google.com
url2	REG_SZ	http://www.hotmail.com/
url3	REG_SZ	http://www.voila.fr

My Computer\HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\TypedURLs

Their deletion is a simple matter of enumerating the entries and removing them one by one:

⊟Collapse | Copy Code

```
__declspec( dllexport ) void Delete_IEAddressBarHistory()
{
    HKEY hKey;
    DWORD dwResult;
    TCHAR szValueName[10];
    int i;

    // Open TypedURLs key.
    dwResult = RegOpenKey(HKEY_CURRENT_USER,
                   "Software\\Microsoft\\Internet Explorer\\TypedURLs", &hKey );

    i = 1; wsprintf(szValueName, "url%d", i);
    while (RegDeleteValue(hKey, szValueName) == ERROR_SUCCESS)
    {
        i++; wsprintf(szValueName, "url%d", i);
    }

    RegCloseKey(hKey);
}
```

Desktop's Recent Documents Cleanup

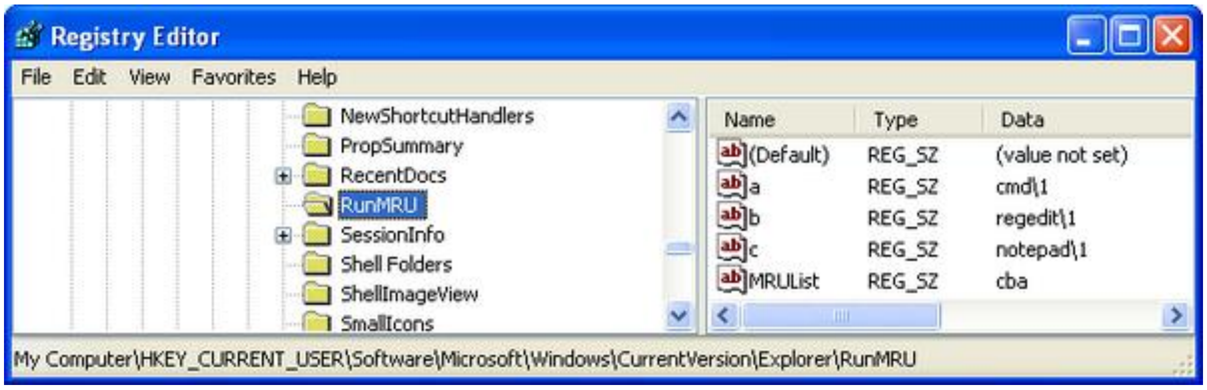
This list tracks the recently opened documents, such as Word, Excel, Notepad documents etc., and is accessible from the Start/My Recent Documents menu. All cleanup work is done by the single Shell command:

⊟Collapse | Copy Code

```
__declspec( dllexport ) void Delete_DesktopRecentDocsHistory()
{
    SHAddToRecentDocs(SHARD_PATH, NULL /* NULL clears history */);
}
```

Desktop's Run History Cleanup

The commands previously typed at Start/Run ... are visible by expanding its dropdown control. They are known as RunMRU list (MRU stands for the Most Recently Used), and is recorded using consecutive alphabets for the registry's value name as a, b, c etc., at *HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\RunMRU*:



Name	Type	Data
(Default)	REG_SZ	(value not set)
a	REG_SZ	cmd\1
b	REG_SZ	regedit\1
c	REG_SZ	notepad\1
MRUList	REG_SZ	cba

My Computer\HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\RunMRU

The command that is typed wrongly is not recorded. The RunMRU list is limited to 26 entries. If the 27th command name is going to be recorded, it replaces the value data for the top value name ("a") and so on.

In contrast to the method of recording typed URL in the IE's address bar, the RunMRU list has a MRUList value name, which is formed by concatenating value names of the RunMRU list and can be used for enumerating and cleaning the list like this:

⊟Collapse | Copy Code

```
__declspec( dllexport ) void Delete_DesktopRunHistory()
{
    HKEY hKey;
    DWORD dwResult;
    TCHAR szValueName[10];

    string c, s;
    s = "abcdefghijklmnopqrstuvwxyz";

    // Open RunMRU key.
    dwResult = RegOpenKey(HKEY_CURRENT_USER,
               "Software\\Microsoft\\Windows\\"
               "CurrentVersion\\Explorer\\RunMRU", &hKey );

    for (int i = 0; i < 26 /* z */; i++)
    {
        c = s.at(i); wsprintf(szValueName, "%s", c.c_str());
        RegDeleteValue(hKey, szValueName);
    }

    RegDeleteValue(hKey, _T("MRUList"));


    RegCloseKey(hKey);
}
```

Desktop's Recycle Bin

Unless you hold SHIFT key while deleting files, they are placed into the Recycle Bin to provide easy restore option. You might want to automate emptying the Recycle Bin with a single Shell cleanup function like this:

⊟Collapse | Copy Code

```
__declspec( dllexport ) void Delete_DesktopRecycleBinContents()
{
    SHEmptyRecycleBin(NULL, NULL,
             SHERB_NOCONFIRMATION | SHERB_NOPROGRESSUI | SHERB_NOSOUND);
}
```

How to Use

The cleanup API for Windows represents a set of functions ported into a Win32 DLL and a COM DLL, named *cleanup.dll* for both DLL types, that are provided as downloadable projects (I used VS.NET 2003 as the IDE). DLL projects should be built as release builds prior to building test clients. Though the COM cleanup API interface is slightly different from Win32 interface due to COM-style requirements, both APIs are identical inside and use only standard Win32 functions (no C run-time libraries). Because of this, *cleanup.dll* relies on the libraries available not only on WindowsXP or Windows 2003, but also on the systems dating back to Windows 95 with IE installed. In particular, Win32's DLL employs standard *kernel32.dll*, *user32.dll*, *shell32.dll*, *ole32.dll*, *advapi32.dll*, and *wininet.dll*.

Which one of the DLL to use, therefore, is a matter of personal preference and style. In a C# or COM application, the COM's *cleanup.dll* would be preferable; in MFC or a compact Win32 application, you might prefer Win32's *cleanup.dll*.

The following table provides the cleanup API function prototypes for the Win32 DLL and the descriptions. The important point is that when deleting *index.dat* files (TIF's, Cookies's, and History's), it is the responsibility of your application to make sure that these files are not locked. If the *index.dat* file is locked, the corresponding functions would silently and harmlessly fail. An example of an application that checks the lock status of *index.dat* files before using cleanup API functions is 4S Lock 1.08's cleanup module mentioned before.

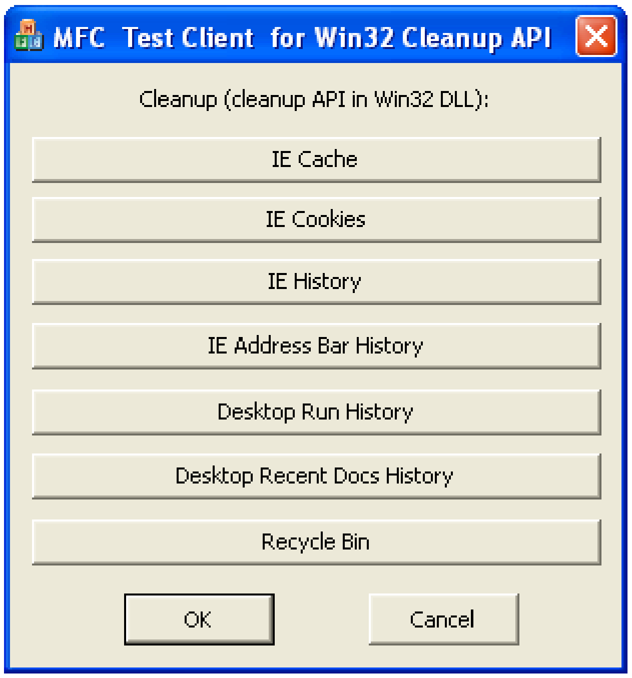
	Function Prototype	**Description**
#1	`BOOL Delete_IECache(BOOL bDeleteCache = TRUE, BOOL bDeleteCacheIndex = FALSE)`	All internet files browsed on the internet with IE are stored (cached) in the *Temporary Internet Files* folder. This speeds up future browsing. To empty the internet cache, but leave its *index.dat* intact, use: ⊟Collapse | Copy Code `Delete_IECache();` `Delete_IECache` can also be used to delete cache's *index.dat* (set `bDeleteCacheIndex = TRUE`) if you are sure this file is not locked.
#2	`BOOL Delete_IECookies(BOOL bDeleteCookies = TRUE, BOOL bDeleteCookiesIndex = FALSE)`	Cookies store user-specific web page customization information. They enable a user not to re-enter his credentials such such as names and passwords. So, private data stored in them can represent a security loophole. To empty internet cookies, but leave its *index.dat* intact, use: ⊟Collapse | Copy Code `Delete_IECookies();` `Delete_IECookies` can be used to delete cookies's *index.dat* (set `bDeleteCookiesIndex = TRUE`) if you are sure this file is not locked.
#3	`HRESULT Delete_IEHistory(BOOL bDeleteHistory = TRUE, BOOL`	Internet browsing history reflects a user's browsing habits, and is visible by clicking IE's History button.

bDeleteHistoryIndex = FALSE)	To empty internet history, but leave its *index.dat* intact, use: ⊟Collapse | Copy Code `Delete_IEHistory();` Delete_IEHistory can be used to delete history's *index.dat* (set bDeleteCookiesIndex = TRUE) if you are sure this file is not locked.
#4 void Delete_IEAddressBarHistory()	Delete_IEAddressBarHistory() can be used to empty the list of previously entered internet addresses reflecting a user's browsing habits, which is visible by expanding the address bar's drop-down control in your Internet Explorer.
#5 void Delete_DesktopRecentDocsHistory()	Delete_DesktopRecentDocsHistory() function can be used to empty the list of recently opened documents, such as Word, Excel, Notepad documents etc., which is accessible from Start/My Recent Documents menu.
#6 void Delete_DesktopRunHistory()	Delete_DesktopRunHistory() function can be used to empty personal command history, which reflects commands entered at Start/Run. The list is visible by expanding a drop-down control at Start/Run.
#7 void Delete_DesktopRecycleBinContents()	Unless the SHIFT key pressed while deleting files, they are placed into the Recycle Bin to provide easy restore option. The Delete_DesktopRecycleBinContents() function can be used to empty Recycle Bin contents.

Finally, you would find in the article's downloads, three test applications (besides two projects for Win32 and COM *cleanup.dll*):

- MFC Test Client for Win32 Cleanup API, which tests Win32's *cleanup.dll*.
- C# Test Client for COM Cleanup API, which tests COM's *cleanup.dll*.
- MFC Test Client for COM Cleanup API, which tests COM's *cleanup.dll*.

All these testing applications have identical interfaces. For example, the interface for MFC application for testing Win32's *cleanup.dll* is the following:


MFC Test Client for Win32 Cleanup API
Cleanup (cleanup API in Win32 DLL):
IE Cache
IE Cookies
IE History
IE Address Bar History
Desktop Run History
Desktop Recent Docs History
Recycle Bin
OK
Cancel

Conclusion

This article provides a general cleanup API for Windows, test applications, and complete source code. The cleanup API can be used on any Windows system, dating back to Windows 95, provided that IE is installed (or *wininet.dll* is available). The cleanup API is packaged into Win32 and COM DLLs, which are identical internally, and can be easily used in your custom application (C++, MFC, Win32, C#, ASP.NET etc.) provided that you mention this article in accordance with regular CodeProject terms of use. Hopefully, information and tools provided with this article make one feel safer in any environment.

Contact me

You can reach Marcel Lambert at mlambert@synaptex.biz.

License

About the Auth