

A Primer on Temporary Internet Files



EricLaw [ex-MSFT] 19 Mar 2011 8:34 AM

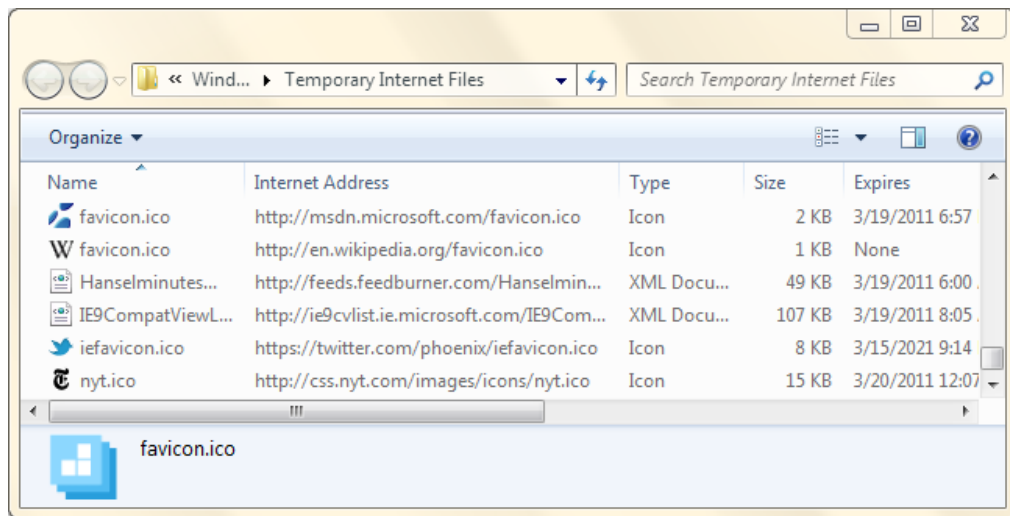
15

On Windows Vista and above, Internet Explorer's Temporary Internet Files are maintained in two isolated WinINET cache containers. One cache is used for sites loaded in Protected Mode (Internet Zone and Restricted Zone) and the other cache is used for sites loaded outside of Protected Mode (Trusted Sites, Local Intranet, and Local Machine).

Each cache container consists of two components: a memory-mapped index database (index.dat) and a nested folder structure containing the response entities that have been cached. Each index holds up to 60000 entries, and each entry maps one request URL to a set of response headers, a bit of metadata, and optionally a file path to the response entity body (if one exists). The cache is cleaned (scavenged) when either the index entry limit is reached, or the disk quota (250mb by default for IE9) is exceeded. If the user deletes their browser history (Tools > Delete Browsing History) the cache index is overwritten with zeros, and all response entities are deleted from disk. If the user closes an InPrivate Browsing session, every item in the cache which was stored during the InPrivate session is removed.

When Internet Explorer asks WinINET to make a network request on its behalf, if the request flags allow, WinINET will reuse a fresh response from the local cache, if one is available. If an expired response is available, WinINET will attempt to validate its freshness by making a conditional HTTP request in order to get back a **HTTP/304** if the response entity is still valid or a new copy of the response entity if the cached version is no longer up-to-date. If WinINET only has a partial response in the cache, it will issue a HTTP request with a **Range** header indicating the remaining part of the file which is not yet in the cache. In order to avoid corruption, the Range request will contain an **If-Range** header containing the **ETag** of the originally cached response, and/or a pre-RFC2616 **Unless-Modified-Since** header containing the Last-Modified time of the originally cached response. If the server's copy of the resource has changed, it will send the entire file again; if not, it will send a **HTTP/206** partial response containing only the requested range of the file.

Prior to IE6, Internet Explorer introduced a mechanism for viewing cached files; you can access this mechanism by clicking Tools > Internet Options > General > Browsing History > Settings. In the TIF and History Settings dialog, click the **View Files** button. Alternatively, you can simply type **shell:cache** into the Internet Explorer Address Bar or the Start > Run prompt. Doing this will open a Windows Explorer window to the **C:\Users\username\AppData\Local\Microsoft\Windows\Temporary Internet Files** folder.



It is important to understand that what you see above is **not** exactly what is stored on disk—alert readers will observe that Explorer is showing columns, like **Internet Address** and **Expires**, that are not typically seen for other folders.

This is accomplished through the magic of a [Shell Namespace Extension](#). As explained on MSDN: *With a namespace extension, you can take any body of data and have Windows Explorer present it to the user as a virtual folder. When a user browses into this folder, your data is presented as a tree-structured hierarchy of folders and files, much like the rest of the Shell namespace.*

This folder is mapped to the Namespace Extension using the **desktop.ini** file within the folder; it contains the following text:

```
[.ShellClassInfo]
UICLSID={7BD29E00-76C1-11CF-9DD0-00A0C9034933}
```







The CLSID listed refers to a COM object implemented in IEFram.dll. When the Namespace Extension is invoked, it generates the "friendly" view of the non-Protected Mode cache. It generates this view by making API calls into the WinINET cache code. The COM object enumerates the cache using [FindFirstUrlCacheEntry](#) / [FindNextUrlCacheEntry](#) without passing any filter; this means that files download by XDomainRequest, files temporarily cached while InPrivate, and [cached HTTP/3xx redirects](#), are not shown in this view.

The "Name" listed isn't actually the name of the cache file on disk, but rather a filename simplistically parsed out of the URL. The HTTP expiration information and similar columns is retrieved from the metadata stored in the index.

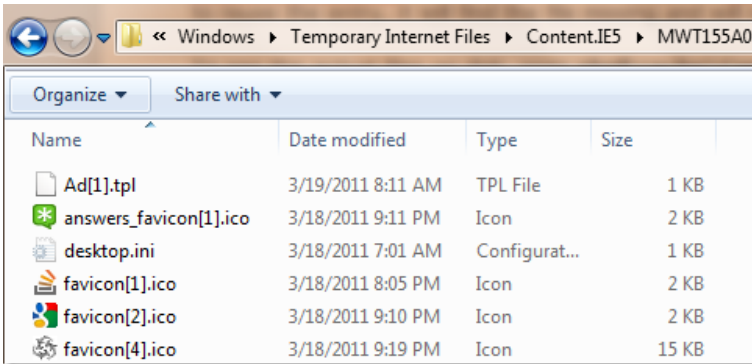
Most importantly, this view does not show the Protected Mode cache; [it only shows files that are downloaded outside of Protected Mode](#). In the screenshot, you'll see a number of Internet-Zone URLs; these are here because Internet Explorer's Medium Integrity "Frame Process" is downloading the FavIcons; the rest of these pages are downloaded and rendered by the Low Integrity Protected Mode "Tab Process."







This view may even show files that do not exist; if the backing response entity file was deleted from the disk without calling [DeleteUrlCacheEntry](#) to delete the index entry, the Namespace Extension will still show the entry. However, if WinINET ever wants to reuse the entry, it will find the file missing and will need to re-download from the server.

To see the actual files on disk, open **shell:cache\Content.IE5**. You'll see the following view:

Name	Date modified	Type	Size
 index.dat	3/19/2011 7:38 AM	DAT File	464 KB
 desktop.ini	3/18/2011 7:01 AM	Configurat...	1 KB
 C4ABNR0X	3/19/2011 7:35 AM	File folder	
 ICSUYRXA	3/19/2011 7:35 AM	File folder	
 MWT155A0	3/19/2011 7:30 AM	File folder	
 Q0Y4OM79	3/19/2011 7:30 AM	File folder	







You see the index.dat database file, and four randomly named subfolders each of which, if opened, contains cached response files:



Name	Date modified	Type	Size
 Ad[1].tpl	3/19/2011 8:11 AM	TPL File	1 KB
 answers_favicon[1].ico	3/18/2011 9:11 PM	Icon	2 KB
 desktop.ini	3/18/2011 7:01 AM	Configurat...	1 KB
 favicon[1].ico	3/18/2011 8:05 PM	Icon	2 KB
 favicon[2].ico	3/18/2011 9:10 PM	Icon	2 KB
 favicon[4].ico	3/18/2011 9:19 PM	Icon	15 KB

The folders are randomly named to mitigate certain types of attacks that involve placing malicious content at predictable file locations and then opening it using a url that uses the **file://** protocol scheme. There are four of them to similarly aid in randomness, as well as for legacy reasons (older filesystems had a limit on the number of files contained in a single folder).

The Protected Mode cache files can be viewed in a similar way, by opening **shell:cache\Low\Content.IE5** instead:

Name	Date modified	Type	Size
 index.dat	3/19/2011 7:18 AM	DAT File	2,272 KB
 desktop.ini	3/18/2011 7:01 AM	Configurat...	1 KB
 BW79U9SN	3/19/2011 7:38 AM	File folder	
 HPR49409	3/19/2011 7:38 AM	File folder	
 T3KL80CF	3/19/2011 7:38 AM	File folder	
 UFNQCAH8	3/19/2011 7:38 AM	File folder	

Again, you'll see the same layout, with four more randomly named subfolders.

Thanks for reading!

-Eric

Comments



zz 19 Mar 2011 5:48 PM

Hi Eric,

Thanks for the wonderful writeup on this topic. I find it very informative. Perhaps you can enlighten me on a particular pet peeve that might be related.

Whenever I'm in InPrivate mode, I find that the favicon never shows up on any of the tabs nor for any inprivate pinned sites.

Is this normal behavior? Is it because of the low integrity temp storage doesn't keep them around? Nevertheless, the favicons shows up in my Favorites.

So its bit strange and frustrating that favicons show up in one place but not the other.

Thanks.



EricLaw [MSFT] 19 Mar 2011 9:57 PM

@zz: When you create a Favorite, the page's FavIcon is copied from the Temporary Internet Files into an NTFS Alternate Data Stream on the .URL file that represents the favorite. That way, even if the TIF is cleared, the icon is still available in your favorites.



KS 20 Mar 2011 4:59 AM

Why is there no TIF view of the protected mode parts?



EricLaw [ex-MSFT] 20 Mar 2011 8:35 AM

@KS: The investment simply wasn't prioritized in the Windows Vista timeframe when Protected Mode was introduced; this view is a very rarely-used feature. Showing the low-integrity cache would have required a significant investment to spin up a COM object at low-integrity and enumerate the low-integrity cache at that IL.



Farhan 17 Apr 2011 4:38 AM

Thanks for this informative article. Can you please put some light over accessing Temp internet files of other user profiles created on the machine? Thanks Farhan

[EricLaw: Can you explain what you mean specifically? Temporary Internet Files are stored in a per-user location, so NTFS ACLs will generally prevent one user from accessing other user's files.]



Peter 3 Aug 2011 11:09 PM

What's the limit of number of objects in the cache for IE8 and IE9 respectively?



EricLaw [ex-MSFT] 4 Aug 2011 6:08 AM

@Peter: The scavenger will run on a cache as soon as it exceeds 60,000 objects. That number is the same between IE6-IE9.



hmdhingra 21 Jan 2013 12:41 PM

I use IE 9. If the cache limit is 60,000 why is my Norton antisoftawre on scanning counts >800,000 files in C:\Users\Harimohan\AppData\Local\Temp\Low\Temporary InternetFiles\Content.IE5\LM4P6AP1\... WITHjs. Is it Ok/possible to delete these files.

EricLaw: If there are really that many files under the Content.IE5 folder, that suggests that something has corrupted the cache index and is preventing proper cleanup. You can safely open this folder in Explorer and delete everything in it (use Shift+Delete to prevent a very slow copy to the recycle bin).



s 30 Sep 2013 9:04 PM

Still confused. I delete my temporary internet folder, however, when I check settings and files all these icons are still in temporary internet folder. Should I delete everything one at a time from there, also? What happens if I do? Thanks

[EricLaw] I'm not sure what "delete my temporary internet folder" means? If you use the Delete Browser History command, without the "preserve favorite website" option set, the folder will be cleared.



Mike Appleby 19 Mar 2014 6:09 PM

Hi, With IE10+ the wininet cache is started by a scheduled task when the user logs on and does not end till the user logs off. Is it possible to end close the wininet cache? Ideally via an API? If the scheduled task process is killed, when you run IE the wininet cache manager is ran from dllhost, again this process doesn't terminate till the user logs off. Any ideas?

[EricLaw] When one asks: "I'd like to surgically remove the brain from a patient and put it back in later, any tips?" the proper response isn't suggesting a saw to use, but rather to ask: "Why on earth would you want to do such a thing?"



Hingus 28 Apr 2014 2:58 AM

I understand that the wininet cache scavenger will be scheduled to run when the number of files cached exceeds 60000. Is it possible to initiate the cache scavenger manually, or via an API?

[EricLaw] The file count is only one trigger for the scavenger; the size of the cache is another trigger, and there's a timer (10 minutes?) for routine scavenging as well.



Hingus 28 Apr 2014 11:33 AM

In other words, there is no way for an wininet application, or from commandline, to initiate a cache scavenger run?

I ask because our application is intended to run 24/7 for weeks and it is using URLDownloadToFile with HTTP to download files, although not continuously but frequently. In a citrix environment where one or more instances of our application, from the same user account, has been running fine for 15+ hours, one of the instance would start reporting failures to open the requested files which were successfully opened just moments or minutes ago (no file changes on the webserver during the run). The only way to fix this problem is to restart the application.

I'm trying to figure out what could have cause this problem: cache scavenger, exhaustion of file handles, calling wininet API from multiple threads in our application, etc?

[EricLaw] Are we having this same conversation on StackOverflow by chance? The problem isn't likely to be the scavenger; you may want to watch a "stuck" process in Process Monitor to see if you see anything suspicious. While WinINET doesn't offer a way to invoke the scavenger, you could try clearing the cache once in a "stuck" state to see if it makes any difference. See e.g. <http://support.microsoft.com/kb/815718>



Hingus 28 Apr 2014 3:13 PM

Thanks for the feedback. I will try to use Process Monitor to watch a "stuck" process tomorrow.

I did try to purge the cache using IE but that didn't help the "stuck" process.



Hingus 30 Apr 2014 1:11 PM

Thanks to your advice, by using ProcessExplorer, I was able to see that the "stuck" process has a handle count of >9700, and a long list of opened files (same files downloaded by our application multiple times and cached at different locations in the WinINET cache). I suspect that the failure to download additional files can be attributed to the exhaustion of file handles allowed in a process.

What confuses me is that our application periodically opens these files by calling InternetOpenUrl with the INTERNET_FLAG_RESYNCHRONIZE flag, retrieves the file contents by calling InternetReadFile, and then closes them by calling InternetCloseHandle. I'd expect that calling InternetCloseHandle would take care of closing all handles relating to the file.

Furthermore, if these files have remain unchanged on the webserver, why would they be downloaded and cached again at a different location when requested repeatedly? I'd expect all subsequent open requests be satisfied by accessing the cached copy because INTERNET_FLAG_RESYNCHRONIZE stipulates that "Reloads HTTP resources if the resource has been modified since the last time it was downloaded".

What am I missing here and how can these files be closed properly?

[EricLaw] I wrote about what INTERNET_FLAG_RESYNCHRONIZE does under the covers here: <http://blogs.msdn.com/b/ieinternals/archive/2010/07/08/technical-information-about-conditional-http-requests-and-the-refresh-button.aspx>. Have you watched your traffic in Fiddler to see whether you're getting a HTTP/200 each time, or a HTTP/304? (Does the server's response have a Last-Modified and/or ETAG that would allow a 304?)

You said you're using URLDownloadToFile, but then said you're using InternetReadFile-- is there some reason you're not reading the created file directly (not using WinINET)? Do you see the same behavior if using URLDownloadToCacheFile instead?



Hingus 30 Apr 2014 3:05 PM

I should have been clearer: Our FileControl is a ActiveX control that is responsible to download files for its clients. It provide supports for redundancy via backup webservers. The FileControl basically supports two ways to return a requested file to its application clients:

- 1) the path of the cached file (client is responsible to open and close the cached file);
- 2) the content of the file in a BSTR (FileControl is expected to open and close the requested file).

The FileControl in turn calls URLDownloadToFile to facilitate for the first option and InternetOpenUrl for the second. According to ProcessExplorer, it appears that only files that were opened for option (2) were left opened.

I've not used Fiddler before, any pointers for me to get started?

[EricLaw] Install it from <http://getfiddler.com>. Open it. Watch.

