# Cache implementation in Mozilla

- Where is the cache located?
- What gets cached?
- How is cache managed?
- Several modules that write to the cache

# Pro's and Con's of Cache

- Advantages of using cache:

1. Performance gain
2. Singal sign-up with cookies

- Disadvantages of using cache:

1. Security issues: caches are stored at known locations on the local drive, non-encrypted.
2. Data might be saved in a different location (such as the temp folder) than the one the user selected.

# Where is the cache located

- Both sensitive and non-sensitive data can be cached
- Two types of caches used by Mozilla:

1. Memory cache: mainly for docs marked explicitly not to be stored on disk

   Ex. https URLs, mail/news inlines

2. Disk cache:

   - In the user profile folder.

   Default location on Linux is HOME/.mozilla/

   ```
   [adele@localhost ka3sd1u5.slt]$ ls
   abook.mab      chrome        key3.db       News          search.rdf
   bookmarks.html  cookies.txt   localstore.rdf  panacea.dat  secmod.db
   Cache          downloads.rdf  Mail          panels.rdf    XUL.mfasl
   cert7.db       history.dat    mailViews.dat  prefs.bak
   chatzilla      history.mab    mimeTypes.rdf  prefs.js
   ```

   - OS' tmp folder. On Linux: /tmp
   - Swapfile

# Snapshot of caches

- Enter “about:cache” in the URL field

Memory cache device

Number of entries: 25
Maximum storage size: 10240 KiB
Storage in use: 138 KiB
Inactive storage: 90 KiB

List Cache Entries
Disk cache device

Number of entries: 68
Maximum storage size: 51200 KiB
Storage in use: 221 KiB
Cache Directory: /home/adele/.mozilla/sourcecode/ka3sd1u5.slt/Cache

List Cache Entries

# Snapshot of caches (cont'd)

- Click “List Cache Entries”...

Disk cache device

Number of entries: 68
Maximum storage size: 51200 KiB
Storage in use: 221 KiB
Cache Directory: /home/adele/.mozilla/sourcecode/ka3sd1u5.slt/Cache

Key:http://us.a1.yimg.com/us.yimg.com/a/qu/quinstreet/090503_25x25_cap.gif
Data size:359 bytes
Fetch count:1
Last modified:Mon 15 Mar 2004 01:37:25 PM PST
Expires:Thu 15 Apr 2010 12:59:58 PM PDT

Key:http://us.i1.yimg.com/us.yimg.com/i/us/pim/f/blkc1.gif
Data size:177 bytes
Fetch count:1
Last modified:Mon 15 Mar 2004 01:37:15 PM PST
Expires:Thu 15 Apr 2010 12:59:56 PM PDT

...

# How does the cache work

- When the user clicks “Reload”
  If the browser finds the doc in the cache, it will check with the server using If-Modified-Since request. If the server sends back 304 (use local copy), the cached doc will be displayed.

- What is the replacement algorithm
  According to “Last accessed” and “Last modified” time. The oldest will be swapped out first.

- What about the /tmp folder?
  Docs are stored in this folder in two cases:
  - Before the entire doc is downloaded.
  - Before the user specifies a destination folder.

  There is an active bug on the usage of the temp folder:

  http://bugzilla.mozilla.org/show_bug.cgi?id=69938

# Modules that execute disk cache I/Os

This is not an exhaustive list. I searched for .cpp files that utilize nsIFile objects, then removed those that do not involve disk cache operations or do not write to files. Among all these packages, I will focus mainly on profile and uriloader packages.

- profile:
  - NsProfileAccess
  - NsProfile
  - nsProfileDirServiceProvider.cpp
- uriloader/exthandler:
  - nsExternalHelperAppService.cpp
  - nsMIMEInfoImpl.cpp
  - nsExternalProtocolHandler.cpp
- caps: nsScriptSecurityManager.cpp
- content: nsContentAreaDragDrop.cpp, nsXULPrototypeCache.cpp
- editor: nsHTMLDataTransfer.cpp, nsHTMLEditorLog.cpp
- embedding: nsEmbedGlobalHistory.cpp, MozillaBrowser.cpp, EmbedComponents.cpp, EmbedProgress.cpp, PtMozilla.cpp, nsMetaModule.cpp, nsWebBrowserPersist.cpp

# Modules that execute file I/Os (cont'd)

- Extensions: nsAutoConfig.cpp, nsTridentPreferencesWin.cpp, wallet.cpp
- mailnews: nsAbLDAPReplicationData.cpp, nsAddrBookSession.cpp, nsAddressBook.cpp, nsMessenger.cpp, nsMsgDBFolder.cpp, nsMsgAccountManager.cpp, nsMsgCopyService.cpp, nsMsgFolderCompactor.cpp, nsMsgMailSession.cpp, nsMsgServiceProvider.cpp, nsSpamSettings.cpp, nsMsgIncomingServer.cpp, nsMsgMailNewsUrl.cpp, nsMsgProtocol.cpp, nsMsgAttachment.cpp, nsMsgCopy.cpp, nsMsgSend.cpp, nsMsgSendLater.cpp, nsURLFetcher.cpp, nsImapMailDatabase.cpp, nsMailDatabase.cpp, nsAbPalmSync.cpp, nsImapMailFolder.cpp, nsImapOfflineSync.cpp, nsImapOfflineSync.h, nsOEMailbox.cpp, nsOEScanBoxes.cpp, nsOESettings.cpp, nsOutlookImport.cpp, nsImportAddressBooks.cpp...
- modules: imgCache.cpp, nsPref.cpp, nsPrefBranch.cpp, nsPrefService.cpp, nsSafeSaveFile.cpp, nsMPFileLocProvider.cpp, nsPluginDirServiceProvider.cpp, nsPluginHostImpl.cpp, nsPluginInstancePeer.cpp
- netwerk: nsDirectoryIndexStream.cpp, nsDownloader.cpp, nsFileStreams.cpp, nsIOService.cpp, nsCacheService.cpp, nsDiskCacheDevice.cpp, nsDiskCacheMap.cpp, nsCookieService.cpp, nsAboutCacheEntry.cpp, nsIndexedToHTML.cpp

Let's take a closer look at some of the modules...

# nsProfileAccess

- ProfileStruct:
  Public:
  - nsresult GetResolvedProfileDir(nsILocalFile **aDirectory)
  - nsresult SetResolvedProfileDir(nsILocalFile *aDirectory):
    // update the directory in the profile and the registry
  - nsresult CopyProfileLocation(ProfileStruct *destStruct)
  - and more...
  - nsString profileName;
  - PRBool isMigrated;
  - nsCOMPtr<nsILocalFile> migratedFrom;
  - nsString NCProfileName;
  - nsString NCDeniedService;
  - nsString NCEmailAddress;
  - nsString NCHavePregInfo;
  - PRBool updateProfileEntry;
  - PRBool isImportType;
  - PRInt64 creationTime;
  - PRInt64 lastModTime;

# nsProfileAccess (cont'd)

- nsProfileAccess:
  - nsresult GetOriginalProfileDir(const PRUnichar *profileName, nsILocalFile **orginalDir);
  - void GetCurrentProfile(PRUnichar **profileName)
    - gets the current profile if set, or the first profile, or an empty string
  - nsresult SetProfileLastModTime(const PRUnichar *profileName, PRInt64 lastModTime);
  - nsresult SetStartWithLastUsedProfile(PRBool aStartWithLastUsedProfile);
  - void SetCurrentProfile(const PRUnichar *profileName);
  - nsresult FillProfileInfo(nsIFile* regName)
    - reads data from the profile registry into an array of profile structs
  - void CheckRegString(const PRUnichar *profileName, char** regString);
  - void RemoveSubTree(const PRUnichar* profileName);
  - nsresult UpdateRegistry(nsIFile* regName);
  - and more...

# nsProfile

- nsresult CopyRegKey(const PRUnichar *oldProfile, const PRUnichar *newProfile);
- nsresult CreateDefaultProfile(void);
- nsresult LoadDefaultProfileDir(nsCString & profileURLStr, PRBool canInterract);
- nsresult CopyDefaultFile(nsIFile *profDefaultsDir, nsIFile *newProfDir, const nsACString &fileName);
- nsresult LoadNewProfilePrefs();
- NS_IMETHODIMP SetCurrentProfile(const PRUnichar * aCurrentProfile);
  - used when the user switches to another profile
- nsresult SetProfileDir(const PRUnichar *profileName, nsIFile *profileDir);
- nsresult UpdateCurrentProfileModTime(PRBool updateRegistry);
- NS_IMETHODIMP CreateNewProfile(const PRUnichar* profileName, const PRUnichar* nativeProfileDir, const PRUnichar* langcode, PRBool useExistingDir)
  - renames an existing profile to a new one, copies all the keys over
- NS_IMETHODIMP RenameProfile(const PRUnichar* oldName, const PRUnichar* newName)
  - Similar to CreateNewProfile, except that deletes the old profile from the registry. However, the directory on disk still uses the old name on purpose.
- NS_IMETHODIMP nsProfile::DeleteProfile(const PRUnichar* profileName, PRBool canDeleteFiles)
  - only deletes the profile from the registry, not the real directory on disk unless the user asks for it

# nsProfileDirServiceProvider

- nsresult SetProfileDir(nsIFile* aProfileDir)
- nsresult InitProfileDir(nsIFile *profileDir)
- nsresult EnsureProfileFileExists(nsIFile *aFile, nsIFile *destDir)
- nsresult UndefineFileLocations()
- nsresult NS_NewProfileDirServiceProvider(PRBool aNotifyObservers, nsProfileDirServiceProvider** aProvider)

# uriloader/exthandler/nsExternalHelperAppService

- nsExternalHelperAppService:
  - handles content that Mozilla cannot handle by itself, makes use of the MIME info of a certain file, and calls an external app handler to operate on this file.
- nsExternalAppHandler:
  - saves the file into a temp folder, launches an app handler when receives OnStopRequest
  - nsresult SetUpTempFile(nsIChannel * aChannel);
  - nsresult MoveFile(nsIFile * aNewFileLocation);
    - moves a file from temp folder to a destination folder
  - NS_IMETHODIMP nsExternalAppHandler::OnStartRequest(nsIRequest *request, nsISupports * aCtxt)
    - sets up temp file, prompts the user for action, and opens the Save/Open dialog.
  - NS_IMETHODIMP nsExternalAppHandler::OnDataAvailable(nsIRequest *request, nsISupports * aCtxt, nsIInputStream * inStr, PRUint32 sourceOffset, PRUint32 count)
    - read from the incoming stream and output to the temp file with error checking
  - NS_IMETHODIMP nsExternalAppHandler::OnStopRequest(nsIRequest *request, nsISupports *aCtxt, nsresult aStatus)
    - checks error, closes the stream, and executes desired action

# netwerk/cache/nsCacheService

- Methods called by nsCacheSession
  - static nsresult OpenCacheEntry(nsCacheSession *session, const char *key, nsCacheAccessMode accessRequested, PRBool blockingMode, nsICacheListener *listener, nsICacheEntryDescriptor ** result);
  - static nsresult EvictEntriesForSession(nsCacheSession * session);
- Methods called by nsCacheEntryDescriptor
  - static nsresult GetFileForEntry(nsCacheEntry *entry, nsIFile **result);
  - static nsresult OpenInputStreamForEntry(nsCacheEntry *entry, nsCacheAccessMode mode, PRUint32 offset, nsIInputStream **result);
  - static nsresult OpenOutputStreamForEntry(nsCacheEntry *entry, nsCacheAccessMode mode, PRUint32 offset, nsIOutputStream **result);
  - static nsresult SetCacheElement(nsCacheEntry * entry, nsISupports * element);
- Methods called by nsCacheProfilePrefObserver
  - static void OnProfileShutdown(PRBool cleanse);
  - static void OnProfileChanged();
  - static void SetDiskCacheEnabled(PRBool enabled);
  - static void SetMemoryCacheEnabled(PRBool enabled);

# netwerk/cache/nsDiskCacheDevice

- virtual const char * GetDeviceID(void) = 0;
- virtual nsCacheEntry * FindEntry( nsCString * key ) = 0;
- virtual nsresult DeactivateEntry( nsCacheEntry * entry ) = 0;
- virtual nsresult BindEntry( nsCacheEntry * entry ) = 0;
- virtual void DoomEntry( nsCacheEntry * entry ) = 0;
- virtual nsresult OpenInputStreamForEntry(nsCacheEntry *entry, nsCacheAccessMode mode, PRUint32 offset, nsIInputStream ** result) = 0;
- virtual nsresult OpenOutputStreamForEntry(nsCacheEntry *entry, nsCacheAccessMode mode, PRUint32 offset, nsIOutputStream ** result) = 0;
- virtual nsresult GetFileForEntry( nsCacheEntry *entry, nsIFile **result ) = 0;
- virtual nsresult EvictEntries(const char * clientID) = 0;
  - evicts cache entries associated with the clientID. Active entries are doomed

# netwerk/cookie/nsCookieService

- void PrefChanged(nsIPrefBranch *aPrefBranch);
- nsresult Write();
- PRBool SetCookieInternal(nsIURI *aHostURI, nsIChannel *aChannel, nsDependentCString &aCookieHeader, nsInt64 aServerTime, nsCookieStatus aStatus, nsCookiePolicy aPolicy);
- void AddInternal(nsCookie *aCookie, nsInt64 aCurrentTime, nsIURI *aHostURI, const char *aCookieHeader);
- void RemoveCookieFromList(nsListIter &aIter);
- PRBool AddCookieToList(nsCookie *aCookie);
- void RemoveAllFromMemory();
- void RemoveExpiredCookies(nsInt64 aCurrentTime);
- void LazyWrite();
- static void DoLazyWrite(nsITimer *aTimer, void *aClosure);

# modules/libpr0n/imgCache

- static PRBool Put(nsIURI *aKey, imgRequest *request, nsICacheEntryDescriptor **aEntry);
- static PRBool Get(nsIURI *aKey, PRBool *aHasExpired, imgRequest **aRequest, nsICacheEntryDescriptor **aEntry);
- static PRBool Remove(nsIURI *aKey);
- static nsresult ClearChromeImageCache();
- static nsresult ClearImageCache();

# modules/libpref/nsSaveSafeFile

- nsresult CreateBackup(PurgeBackupType aPurgeType);
- nsresult RestoreFromBackup(void);

# modules/libpref/nsPrefService.cpp

- nsresult UseDefaultPrefFile();
- nsresult UseUserPrefFile();
- nsresult ReadAndOwnUserPrefFile(nsIFile *aFile);
- nsresult ReadAndOwnSharedUserPrefFile(nsIFile *aFile);
- nsresult SavePrefFileInternal(nsIFile* aFile);
- nsresult WritePrefFile(nsIFile* aFile);

# embedding/lite/nsEmbedGlobalHistory

- nsresult LoadData();
- nsresult FlushData(PRIntn mode = kFlushModeFullWrite);
- nsresult ResetData();
- nsresult GetHistoryFile();
- static PRIntn PR_CALLBACK enumRemoveEntryIfExpired(nsHashKey *aKey, void *aData, void* closure);

# embedding/components/webbrowserpersist /nsWebBrowserPersist

- nsresult CloneNodeWithFixedUpURIAttributes(nsIDOMNode *aNodeIn, nsIDOMNode **aNodeOut);
- nsresult SaveURIInternal(nsIURI *aURI, nsISupports *aCacheKey, nsIURI *aReferrer, nsIInputStream *aPostData, const char *aExtraHeaders, nsIURI *aFile, PRBool aCalcFileExt);
- nsresult SaveDocumentInternal( nsIDOMDocument *aDocument, nsIURI *aFile, nsIURI *aDataPath);
- nsresult SaveDocuments();

# extensions/tridentprofile/nsTridentPreferencesWin

- nsresult CopyPreferences();
- nsresult CopyCookies();
- nsresult CopyCookiesFromBuffer(char *aBuffer, PRUint32 aBufferLength, nsICookieManager2 *aCookieManager);

# References


- "FAQ of the Caching Mechanism in 331 Release", retrieved on 3/20/04, from http://www.mozilla.org/docs/netlib/cachefaq.html.
- "HTTP Caching in Mozilla", retrieved on 3/20/04, from http://www.mozilla.org/projects/netlib/http/http-caching-faq.html.
- "Bugzilla Bug 69938", retrieved on 3/20/04, from http://bugzilla.mozilla.org/show_bug.cgi?id=69938.


*End*