Proposal for AFS Windows Shell Namespace Extension

As things currently stand, users interact with AFS in the same manner they interact with normal CIFS shares. AFS specific functionality is exposed via a set of command line tools and a shell extension that adds AFS specific menu items to the context menus in the Windows Explorer shell. Many institutions that use AFS have found that this arrangement is not sufficiently user friendly due to a number of reasons including the difficulty of navigating the AFS hierarchy. This proposal is for a Windows Explorer Shell namespace extension and the associated AFS cache manager modifications that aim to improve the accessibility of the AFS folder hierarchy.

Note that this proposal does not cover other aspects of the AFS shell extensions such as the addition of AFS specific property sheets and meta-data handlers.

Contents

Windows Shell Namespaces 3

AFS Namespaces 4

"Recent" Namespace 4

Criteria for Adding a Folder 5

Naming Shortcuts 5

Accessing the Namespace 6

Persistence 6

Communication with the Cache Manager 6

Custom Namespaces 7

Accessing Custom Namespaces 7

Custom Namespace Mount Points 8

User Experience 9

Mounting AFS namespaces 9

Interaction with the "Recent" Namespace 9

Interaction with Custom Namespaces 10

Mapping Drives 10

Intended Usage 11

Specifications 12

New PIOCTL Calls 12

SetMRU 12

GetMRU..... 14
Implementation Estimates..... 16

Windows Shell Namespaces

Windows Explorer provides a graphical interface to browse, navigate and manage the Shell namespace. This namespace consists of the file system hierarchy and several other Shell namespaces that don't correspond to the file system directly or at all. The latter types of namespaces are called "virtual namespaces". Examples of these are the Control Panel (Figure 1) and My Network Places.

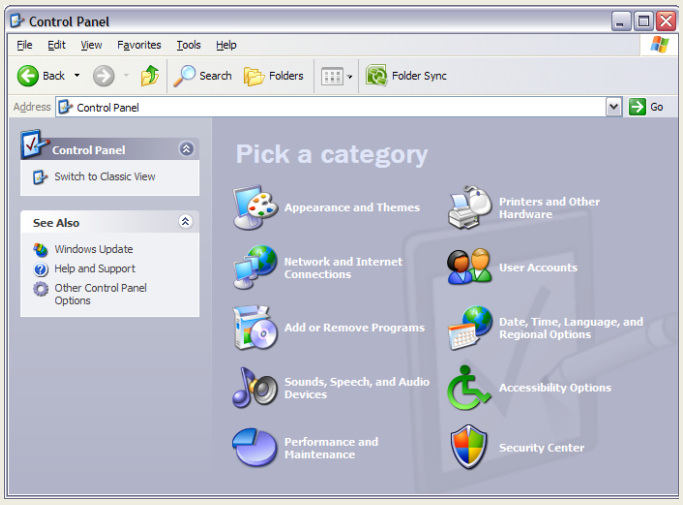

Figure 1: Windows Explorer displaying the Control Panel namespace

Third party applications can extend the Shell namespace by providing a plug-in to Windows Explorer called a Shell Namespace Extension. These extensions provide information about the namespace to Windows Explorer and can have full control over how the namespace is rendered to the user. A new namespace can be attached to the current Shell namespace as a child of an existing namespace, including file system folders, allowing the user to navigate to the new namespace.

AFS Namespaces

While the entire AFS file system hierarchy is huge, the subset of locations in AFS that a typical user will routinely access tends to be quite small. The goal of the proposed AFS namespaces is to make these locations easier to access.

To this end, the AFS namespace extension will add two types of namespaces to the Windows Explorer namespace hierarchy. One namespace, tentatively called "Recent", will contain a list of folders in AFS that have been accessed recently. The other will be a set of namespaces that are available for exposing organization specific locations in AFS, such as user and group home folders.

"Recent" Namespace

Windows maintains a list of network shares that have been recently accessed and makes those locations visible via the "My Network Places" namespaces. These shortcuts are stored in the *NetHood* folder in the user's home folder. The *NetHood* folder contains a set of subfolders which are junction points for the individual network locations. If a user accesses a document in a network share, a shortcut will be placed in this folder. However, these shortcuts only reference the network share, and not the folder within the share that was accessed. If a document is accessed in the network path `\\server\share\path\document.doc`, then the shortcut will target `\\server\share`. In the context of AFS, this will usually result in a shortcut to the root of the cell unless the network path referenced a Freelance submount or symbolic link. As such, the "My Network Places" namespace does not provide a sufficiently convenient way to access locations in AFS.

To remedy this, the proposed "Recent" namespace will contain a list of shortcuts to the folders in AFS that have been accessed most recently by the user.

The number of shortcuts that is maintained will be configurable. The list will not follow strict MRU semantics, as the user will be able to "pin" a shortcut to prevent it from getting discarded if the list of accessed locations exceed the limit. In addition, users would be able to remove items from the list, or clear the list entirely.

The cache manager will be augmented to keep track of recently accessed folders on a per-user basis. This information will then be used by the namespace extension to maintain the list of shortcuts. Using the cache manager directly ensures that the data is consistent regardless of the method used to access resources in AFS.

When the cache manager receives a file system operation, it will conditionally add the referenced folder to an MRU list that is maintained per logged in user (The conditions under which a folder is added to the list is discussed in subsection Criteria for Adding a Folder below). The cache manager will then notify the shell extension that the user's MRU list has changed. These notifications are done via a change notification for the mount point for the namespace. The shell extension can merge the cache manager MRU list with the pinned folder list to display the user's "Recent" folders list. This process is illustrated in Figure 2.

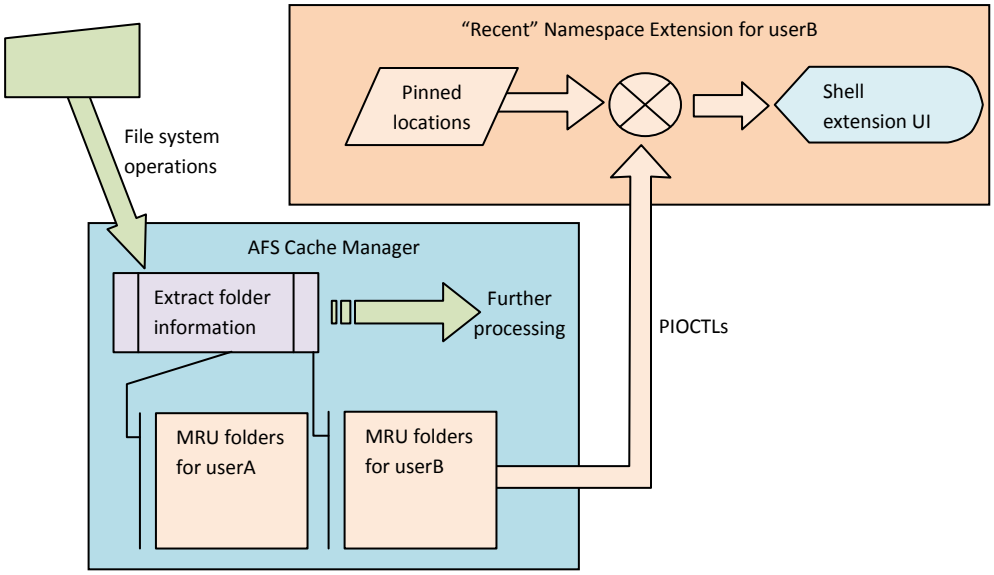

Figure 2 : Operation of the "Recent" namespace extension

Criteria for Adding a Folder

Currently, it is proposed that a folder be added to the "Recent" list if a file within that folder is read from or written to.

Simply adding a folder because a file operation was received for any resource in the folder can generate too much noise due to directory listings. Adding a folder when a file in the folder was read or written to can still generate false positives since Windows Explorer opens and reads many of the files in a folder when it is displaying a folder view (reading the "desktop.ini" file for the folder, reading the icon resources off of binaries, generating thumbnails of images, reading author/subject/title information from documents etc…). But these false positives are not expected to be numerous enough to adversely affect usability.

Naming Shortcuts

The shortcuts that are automatically generated need to be named properly for them to be exposed to the shell extension and to the user. This is proposed as follows:

- If the last component of the target folder path is unique among the set of shortcuts, use the last component.
- Otherwise, traverse up the target folder paths of the subset of shortcuts that have the same last component until a unique component is found, and then concatenate the unique component and the last component using a space as a delimiter.
- Failing all else, concatenate the last component and a serial number using a space as a delimiter.

For example, the following set of folders:

1. `/afs/athena.mit.edu/user/j/a/jaltman/Public/OpenAFS`

2. /afs/athena.mit.edu/user/a/s/asanka/Public/ja
3. /afs/openafs.org/software/openafs/1.5.19/winxp
4. /afs/openafs.org/software/openafs/1.5.20/winxp

Will be given the following set of shortcuts:

1. OpenAFS → /afs/athena.mit.edu/user/j/a/jaltman/Public/OpenAFS
2. ja → /afs/athena.mit.edu/user/a/s/asanka/Public/ja
3. "1.5.19 winxp" → /afs/openafs.org/software/openafs/1.5.19/winxp
4. "1.5.20 winxp" → /afs/openafs.org/software/openafs/1.5.20/winxp

Accessing the Namespace

The "Recent" namespace will be exposed as a junction point at the UNC path `\\afs\recent`. The AFS cache manager will implement junction point semantics to associate `\\afs\recent` with the "Recent" namespace extension, thereby allowing the shell extension to control the appearance and interaction of the namespace.

In addition, since Shell extension namespaces can be mounted in multiple locations, the "Recent" namespace can be exposed in other locations as well. The User Experience section discusses other options for mounting AFS namespaces.

Persistence

Recent and pinned folder lists will be stored in the user's registry hive in a persistent manner. Users with roaming profiles will then see a consistent list of folders regardless of where they are logging in from. After the user's profile is loaded, the shell extension will read the list of folders from the user's registry hive and make PIOCTL call to the AFS cache manager to set the starting list of recent folders. Periodically during the course of operation and during logout, the shell extension will update the recent folder list in the registry. Pinned folders will be written to the registry when the user makes a change to the pinned state of a folder.

Communication with the Cache Manager

The shell extension will utilize new PIOCTL calls for getting and setting the list of recently accessed folders for the user. In addition, the shell extension will request a change notification for the mount point of the "Recent" namespace (at the UNC path `\\afs\recent`). The cache manager will signal a file change notification for that folder whenever the list of recently accessed folders changes. This allows the shell extension to keep the list of recently accessed folders current.

The proposed PIOCTL calls are documented in the Specifications section.

Custom Namespaces

The AFS namespaces will allow additional extensions to support custom namespaces. Typically, an organization might want to expose several large namespaces corresponding to locations in AFS that users will need to access. These can be home folders of other users, groups or projects.

Custom AFS namespaces, as proposed, will exist at a path relative to the AFS namespace root and perform the following:

- Map a given string to one or more shortcuts in the namespace (Search or lookup)

At the current time, custom AFS namespaces will not be providing a separate user interface. Instead, the AFS Shell Namespace Extension will provide the necessary user interface.

In the general case, a custom namespace extension would be necessary to provide a convenient method for users to locate a resource in AFS by name (or a suitable search phrase). The search space (or the namespace) is assumed to not be convenient for the user to navigate manually. Therefore custom namespaces are not expected to expose the entire namespace to the user. Instead, they will be used as a lookup or search mechanism.

It is not a requirement that the name entered by the user directly correspond to the target of the returned shortcut when performing a lookup or search. However, the name of a shortcut that is returned as a unique result for a lookup has to be the same as the term that was looked up. Names of the shortcuts that are returned must also be unique and static. I.e. if a shortcut named "`foo`" with target `\\afs\some\path\foo` was returned, then a shortcut should not be returned that is named "`foo`" with any other target.

The custom namespace is expected to support wildcard and prefix based searching.

Accessing Custom Namespaces

Custom namespaces exist as children of the AFS namespace. They will be designated as a junction points that are handled by the AFS Shell Namespace extension.

For the purpose of illustration, assume that a custom namespace called "users" exists at `\\afs\users`. The user can open a view of Explorer at `\\afs\users`, in which case she will be presented with a user interface provided by the AFS Shell Namespace extension that exposes the "users" custom namespace. This UI is discussed in the User Experience section below.

If the user attempts to visit `\\afs\users\joeuser` using Windows Explorer, then the AFS Namespace extension will invoke the custom namespace to map the string "`joeuser`" to a file system object. If the custom namespace uniquely maps "`joeuser`" to a directory in AFS, then the result would be equivalent to opening a shortcut to that directory. However, if there are multiple results to the query, such as `\\afs\some\path\joeuser1` and `\\afs\some\other\path\joeuser2`, then "`joeuser1`" and "`joeuser2`" will be displayed as auto completion possibilities. Note that pathnames containing wildcards cannot be used as a target when browsing.

Custom Namespace Mount Points

More than one organization may wish to deploy custom namespaces. However, each mount point can only host one custom namespace. Therefore, organizations deploying custom namespaces are encouraged to use mount points that are named after the organization. For example, Stanford University might deploy a custom namespace including all the home directories of their users that is mounted at "`\\afs\Stanford Users`." Doing so reduces the possibility of namespace collisions.

User Experience

The primary user interface for the AFS Shell namespaces will be Windows Explorer. The AFS Shell extension will extend the user interface functionality provided by Windows Explorer by providing task items, search boxes, meta-data and context menus.

Figure 3 shows a mockup of what the "Recent" namespace might look like when a user browses to `\\afs\recent`, which is the default mount point for the namespace.

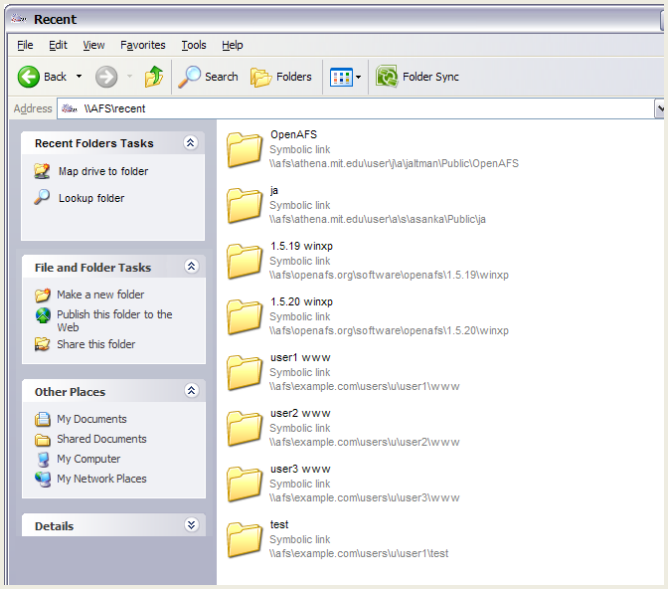

Figure 3 : Recent Namespace

Mounting AFS namespaces

An extension namespace can be mounted in multiple locations in the Windows Shell namespace. Therefore, for convenience, the AFS namespace root can be mounted under the user's Desktop. This is illustrated in Figure 4, which shows a mockup of the Folders view in Windows Explorer. Mounting a namespace under the user's Desktop namespace also has the side-effect of creating an icon on the user's desktop, which provides a convenient method for the user to access the namespace.

Convenience shortcuts can also be placed on the Start Menu. On Windows XP and Vista, the top left area of the Start Menu hosts "pinned" items. Although this area normally contains shortcuts to programs, it can also host shortcuts that point to shell namespaces.

The right side of the Start Menu is reserved for use by Windows Explorer. While it can be customized to include or exclude predefined shortcuts, it cannot be used to host arbitrary shortcuts. One exception to this is the slot reserved for OEM use, which is typically used by hardware manufacturers to link to support documentation.

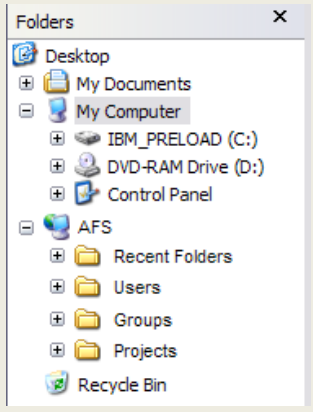

Figure 4 : Folder hierarchy including AFS namespaces

Interaction with the "Recent" Namespace

The user will see the "Recent" namespace as a regular folder that contains a list of shortcuts. The extension will provide the list of shortcuts that target the recently used directories when Windows Explorer requests an enumeration of the contents of the namespace. This same list will be used to display the contents to the user.

User interaction will resemble that of a regular folder, with the exception that new items cannot be created in the “recent” namespace, nor will it be a target for drag and drop operations.

Interaction with Custom Namespaces

Custom namespaces are a special case. Since these namespaces are assumed to be large, the contents that are exposed to Windows Explorer and the user will be different. For the purpose of handling automatic completion, the extension will provide an exhaustive list to Windows Explorer. However, when a user visits a custom namespace, the contents will only consist of items that have been added as a result of the most recent search operation.

Since the contents as seen by Windows Explorer includes the entire namespace, if a user were to manually type a path such as `\\afs\users\joeu` in the Explorer location bar, Windows Explorer will be able to aid the user by providing a list of completions for the last component consisting of partial matches to what has already been typed (assuming, of course, that `\\afs\users` is a custom namespace consisting of all the user home directories).

At the same time, if the user visits `\\afs\users`, she will not be presented by the entire contents of the namespace. Instead the view would resemble the mockup in Figure 5. The mockup demonstrates the outcome of the user searching for the prefix “joe” in the search bar at the top of the folder view area.

The search bar will be provided by the AFS Shell Extension for custom namespaces.

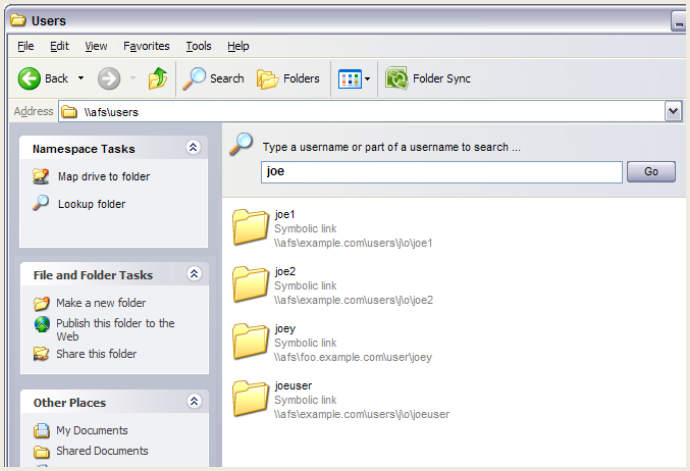

Figure 5 : The "users" namespace

The contents of the remainder of the folder view area consist of a list of shortcuts that target the resulting locations in AFS. The behavior of this namespace resembles that of the “Search Results” folder in Windows.

Mapping Drives

Users can map a drive to any folder they see in any of the namespace views either by right clicking the folder or by clicking the “Map drive to folder” item from the “Namespace Tasks” list on the left of the folder view. Either of these actions will invoke the “map drive” dialog provided by the AFS shell extension that is automatically populated with the selected folder as the target. Only

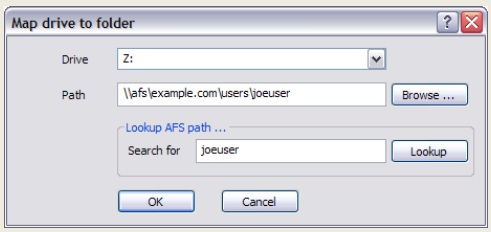

Figure 6 : Map Drive Dialog

one folder can be mapped at a time.

Figure 6 shows a mockup of the “map drive” dialog. It will resemble a simplified version of the “Map Network Drive” dialog in Windows Explorer, but includes an additional set of controls for looking up folders in custom namespaces.

Intended Usage

Note that the AFS namespaces are only meant to be accessed via Windows Explorer. Some of the objects in these namespaces, such as symbolic links in the \\afs\recent namespace, are visible from the file system. However, the exposed objects are not meant to be accessed directly. As mentioned above, the lifetimes of these objects will not be managed in a way that facilitates direct usage.

Specifications

New PIOCTL Calls

Two new PIOCTL calls are proposed to facilitate communication between the AFS shell extension and the AFS cache manager for the purpose of keeping track of recently accessed folders.

These folders cannot be exposed as regular symbolic links within AFS. The list of folders is expected to change often and abruptly. Since versions of Windows prior to Windows Vista are not aware of symbolic links, the cache manager has to expose them as regular files and folders. Applications that are accessing folders and files through symbolic links expect them to behave the same as regular folders and files. Abruptly removing a symbolic link can result in an application not being able to access the target file system object. Therefore, the proposed PIOCTL calls are used to obtain and set the MRU list.

SetMRU

This call is used to set the MRU and other options for the current user. When a user logs in, the AFS shell extension reads the persistent list of recent folders from the user's registry hive and makes this call to initialize the cache manager's MRU list for the user.

The proposed format of the SetMRU PIOCTL is as follows:

```
CSETMRU {
WORD            SubCode
  union {
  CSETMRUOPT    MRUOpt
  CSETMRUDATA0  MRUData0
  CSETMRUDATA1  MRUData1
  }
}
```

The SubCode is a value that identifies the purpose of the SetMRU call. It can be either SETMRUOPT, SETMRUDATA0 or SETMRUDATA1. The SubCode also identifies which member of the union is present in the data packet.

Shell extensions call SetMRU with SubCode set to SETMRUOPT in order to set the options for the per-user MRU list maintained by the cache manager. In this case, the CSETMRU packet contains a CSETMRUOPT structure, which is defined below. This structure defines the number of paths that should be kept in the cache manager MRU for this user in nMRUSize and a Boolean flag in nReset that determines whether the cache manager should clear the contents of the existing MRU.

```
CSETMRUOPT {
WORD            nMRUSize
WORD            bReset
}
```

In response to this packet, the cache manager should resize the MRU list for the user. If bReset is non-zero, then the resulting MRU list should be empty, but capable of holding at least nMRUSize paths. If the bReset value is zero, then the cache manager is expected to set the size of the MRU list to nMRUSize

by removing the oldest entries if necessary. Setting nMRUSize to zero disables MRU folder tracking for the user.

The cache manager does not need to keep track of MRU folders for the user until it receives a CSETMRUOPT packet. In addition, the cache manager can reject any other SetMRU call if it has not received a CSETMRUOPT packet for the user or the last call disabled MRU folder tracking.

In response, the server should send an error code indicating whether the operation was successful.

The value SETMRUDATA0 in the SubCode field identifies the initial packet in a series of packets that define the contents of the MRU. In this case, the CSETMRU packet will contain a CSETMRUDATA0 structure in the union.

```
CSETMRUDATA0 {
DWORD           Cookie
WORD            nTotalEntries
WORD            nEntries
MRUENTRY        entries[nEntries]
}
```

This structure contains a Cookie field which contains a random number. Subsequent calls to SetMRU with SubCode set to SETMRUDATA1 for the remainder of the data will also contain a Cookie field whose value is computed by incrementing the Cookie value that was passed in the previous CSETMRUDATA0 or CSETMRUDATA1 packet.

The nTotalEntries field contains the total number of MRU entries that will be sent in this sequence of calls. This value cannot exceed the nMRUSize that was specified in the last SETMRUOPT packet. The remaining fields specify the actual MRU entries.

```
CSETMRUDATA1 {
DWORD           Cookie
WORD            nEntries
MRUENTRY        entries[nEntries]
}
```

Once the CSETMRUDATA0 packet has initiated the sequence of calls for setting the MRU entries in the cache manager, the caller should send CSETMRUDATA1 packets until all the MRU entries have been sent. The SubCode of the SetMRU call should be set to CSETMRUDATA1 and the Cookie field of CSETMRUDATA1 should contain the value of the previous Cookie incremented by one. CSETMRUDATA1 is similar to CSETMRUDATA0, except there is no nTotalEntries field. The sum of all the nEntries fields in the CSETMRUDATA0 and CSETMRUDATA1 packets should equal the nTotalEntries value specified in the CSETMRUDATA0 call.

```
MRUENTRY {
TIME_T          aTime
DWORD           AccessCount
WORD            CCh
CHAR            path[CCh]
}
```

MRUENTRY structures define an MRU entry. The aTime field contains the last known access time for the folder represented as a time_t value. The AccessCount field contains the number of times this folder has been accessed. This value is only used for statistical purposes and as a hint for determining folders that the user frequently accesses.

Once a CSETMRUDATA0 packet is received, the cache manager should queue the data from that and subsequent CSETMRUDATA1 packets until the sum of nEntries of the packets equal the nTotalEntries value of the CSETMRUDATA0 packet. At that point, the cache manager should replace the contents of the MRU list for the user with the entries that were specified in the SetMRU calls.

The cache manager can abandon a CSETMRUDATA sequence under any of the following conditions:

- A CSETMRUOPT request is received for the user
- A new CSETMRUDATA0 request is received for the user
- The sum of nEntries for the sequence exceeds the nTotalEntries value
- nTotalEntries is not reached and no CSETMRUDATA1 packets were received in the past 10 seconds
- A GetMRU call was received for the user

If the Cookie field does not match up, the cache manager should discard that packet.

The cache manager should respond to each packet with a return code indicating whether the packet was accepted. The only exception is the last packet in the sequence, whose return value should indicate whether the entire sequence was accepted.

GetMRU

This call is used to obtain the list of MRU folders for the current user. The caller passes in the following parameters:

```
CGETMRU {
DWORD   Cookie
}
```

The server responds with the following message:

```
SGETMRU {
WORD    SubCode
union {
SGETMRUDATA0    MRUData0
SGETMRUDATA1    MRUData1
}
}
```

The semantics are similar to the CSETMRU call. The SubCode identifies the purpose of the response and the effective member of the union.

Initially, the caller sets the Cookie field of the CGETMRU structure to zero to indicate that this is a new call. The cache manager responds with a SGETMRU structure with SubCode set to GETMRUDATA0 and the MRUData0 member of the union. The SGETMRUDATA0 structure is as follows:

```
SGETMRUDATA0 {
DWORD           Cookie
WORD            nTotalEntries
WORD            nEntries
MRUENTRY        entries[nEntries]
}
```

The Cookie field contains a random number that the caller should use in a subsequent GetMRU call to retrieve the next set of entries. The nTotalEntries item specifies the total number of MRU entries that are available and the nEntries field specifies the number of entries that are included in this packet.

When the caller makes a subsequent GetMRU call, the server should respond with a SGETMRU structure that has SubCode set to GETMRUDATA1 and a populated MRUData1 structure.

```
SGETMRUDATA1 {
DWORD           Cookie
WORD            nEntries
MRUENTRY        entries[nEntries]
}
```

The sequence ends when the server has sent all the MRU entries. I.e. the sum of nEntries reaches nTotalEntries. The final packet may pass in a zero for the Cookie field since no further data is available.

The server can reject a GetMRU call if:

- The cookie does not match
- A SetMRU call was issued before this GetMRU call and the Cookie for this call is non-zero

The SGETMRU structure should only be included in the response packet if the return code is zero. The server should abandon a GetMRU sequence if the next packet was not fetched by the caller for 10 seconds or if the last GetMRU call for the user was rejected.

Implementation Estimates

The implementation tasks are listed below. Each task item is followed by a time estimate in hours. Three times are listed in the format: [expected time, worst case time].

- Junction point support in the cache manager [12, 24]
- Per-user MRU folder lists in the cache manager [15, 24]
- New PIOCTLs [20, 30]
- Extensible API and registration for custom namespaces [15, 30]
    - Documentation for the API
- Shell extension
    - User interface for “Recent” Namespace [45, 72]
        - Management of pinned and cache manager supplied folder lists
        - Persistence of folder lists
        - Documentation
    - User interface for custom namespaces [45, 72]
        - Instance support for custom namespaces
        - Search history persistence
        - Result set management
        - Documentation
    - Common UI elements [60, 90]
        - Explorer side bar extensions for task lists
        - Context menus
        - Map Drive dialog
        - Documentation
    - Custom namespaces for Stanford [70, 100]
        - Custom installers for deploying custom namespaces
        - Deployment guide

Total time all task items: [282, 442]