Technote 1186

How To Be a Good Multiple Users Citizen

CONTENTS

This Technote describes the new APIs provided by the Multiple Users technology introduced in Mac OS 9.0. It also answers the most frequent questions from developers and provides a few workarounds for commonly encountered problems.

This Note is directed at application developers who are accessing folders within the System Folder or using the `FindFolder` API and need to pay extra attention to the access privileges of those folders.

At Ease and Macintosh Manager were previous similar products and all the information provided in this Note is also relevant for those technologies.

Where is that Folder?

The `FindFolder` routine allows callers to determine the location of folders that have a special purpose. For example, `FindFolder` can be used to locate the Preferences folder&emdash;the appropriate location for applications to store their preference files. In the beginning, `FindFolder`'s main utility was the fact that it would always find the appropriate folder even when a different language system was in effect: in these cases, folder names were often spelled differently or may have been named using non-Roman character encodings. Now, with the advent of Multiple Users, not only may the folders returned by `FindFolder` be named in many different ways according to the language in effect, but their location may be different than what is expected.

The Multiple Users technology manages a group of redirected folders associated with each User who may be logged in. If the Owner of the computer is logged in, then no redirection occurs and all folders are accessed as if the Multiple Users technology was not there. However, if any other User is logged in then those redirected (one of each per User) folders are accessible with the `FindFolder` API using the constants:

- `kDesktopFolderType`,
- `kTrashFolderType`,
- `kPrintMonitorDocsFolderType`,
- `kStartupFolderType`,
- `kAppleMenuFolderType`,
- `kPreferencesFolderType`,

- kLauncherItemsFolderType,
- kShutdownItemsFolderType,
- kStartupItemsDisabledFolderType,
- kShutdownItemsDisabledFolderType,
- kInternetSearchSitesFolder,
- kControlStripModulesFolder,
- kDesktopPicturesFolderType,
- kFavoritesFolderType,
- kALMLocationsFolderType,
- kSpeakableItemsFolderType,
- kKeychainFolderType,

Note:
In the case of remote access (i.e., when using Macintosh Manager), the desktop folder (kDesktopFolderType) and the PrintMonitor folder (kPrintMonitorDocsFolderType) will be redirected to the local startup volume instead of the remote volume where user preferences would normally be. Thus, when logged into Macintosh Manager, the preference structure would look like this:

Startup Volume
  Users
    <username>
      Desktop Folder
      PrintMonitor Documents
Remote Volume
  Users
    <username>
      Preferences Folder
      Startup Items
      etc

However, developers should never assume that the location of the desktop folder is at a certain location as this may change in the future. Instead use the vRefNum and dirID from the FindFolder call to locate all redirected folders.

If you need to access the non-redirected folders (not all of them are accessible), you may do so using the following constants:

- kSystemDesktopFolderType
- kSystemTrashFolderType
- kSystemPreferencesFolderType

Additionally, there are new constants defined to locate the User's folders:

- kUsersFolderType: "Users" folder, contains one folder for each user.
- kCurrentUserFolderType: The folder for the currently logged on user.
- kCurrentUserRemoteFolderLocation: The remote folder for the currently logged on user
- kCurrentUserRemoteFolderType: The remote folder location for the currently logged on user
- kSharedUserDataFolderType: A Shared "Documents" folder, readable & writable by all users
- kVolumeSettingsFolderType: Volume specific user information goes here

However, location of those special folders is unfortunately not the last word of the story...

Back to top

Not All Users are Equal

After the boot process completes, Multiple Users displays a login dialog. To login, a user must have an account (usually set up by the owner of the computer). The user may have to enter a password to login. While the login dialog is displayed, background-only applications (i.e., type `'appe'`, or any application with the background-only bit set in the `SIZE` resource) are allowed to run, but normal applications are not.

When the user is finished, they choose **Logout** from the **Special** menu. This begins the logout process. First, all normal applications are sent quit Apple Events until they quit. (Note that once the user has confirmed the logout, there is no way to abort it&emdash;an application will be sent a quit event every few seconds until it actually quits.) Next, a logout notification is sent to all registered programs. Finally, the Folder Manager is told to stop redirecting the per-user folders. Once logout is complete, the login dialog is displayed.

Note:
If the user chooses **Restart** or **Shutdown** from the **Special** menu, the logout notifications will also occur in the same manner.

Applications usually need to write two different kinds of data: the user data, such as documents, or user's preferences, and application data, such as hardware configuration, or dictionaries. The former typically needs to be stored in one of the redirected folder, the latter needs a non-redirected folder. The problem comes from the fact that some environments (Limited and Panels) will forbid write-access to all non-redirected folders but one: the Application Support folder that you can locate through the constant: `kApplicationSupportFolderType`.

Therefore, if your application is currently writing data in any other folder, you will have to make that change in order to be compatible with the Multiple Users technology. In particular, as with Netboot, you can't even be sure that your application's folder is going to be writable.

To determine if you have write access to a given folder, check the `ioACUser` field in the `CInfoPBRec` struct after a call to `PBGetCatInfo`. There is one exception to that rule, though. To provide some level of security for the Application Support folder and its subfolders and to prevent the Finder, Panels, Standard File Package, and Navigation Services to write to the Application Support folder, and since those processes check for it, `PBGetCatInfo` will always return that you don't have write access to those folders. But, in fact, just for those folders, and since most of the current applications never bother to check the permissions first, you can write to them. That means that you should always check for write permission for a folder except for the Application Support folder and its subfolders.

Or, if you prefer, do not check in advance if you can write to a folder, but be ready to handle `afpAccessDenied` (`-5000`) errors when you attempt to read from or write to any folder.

You also have to pay attention to the `foundVRefNum` value returned by `FindFolder` because it may not be the same as the `vRefNum` value you passed in. Some of the redirected folders may be on another volume and even on a remote volume. So always use both the `foundVRefNum` and `foundDirID` values returned.

Back to top

Not All Applications are Equal

If you're writing a classic Macintosh application, aside from using `FindFolder` to locate the folders you need and checking the write access of those folders, it's business as usual. The Multiple Users technology redirects the folders and that's fairly transparent. These applications never "see" the login, and

they receive the `'quit'` Apple Event whenever there is a logout.

But if you're writing a background-only application (type `'appe'`) or some kind of system extension which is always loaded, then you have to be aware of the login/logouts as they occur.

All the following APIs are found in Folders.h.

You can register for the following notifications:

```
kFolderManagerNotificationMessageUserLogIn
```

Sent when a user has logged in. When you receive this message `FindFolder` will return the `vRefNum` and `dirID` of the user's redirected folders (see list of redirected folders, above) until the user logs out. This message can be used to load the new user's preferences.

```
kFolderManagerNotificationMessagePreUserLogIn
```

Sent just prior to redirecting `FindFolder` to the user's folders. Calling `FindFolder` when receiving this notification will return the `vRefNum` and `dirID` of the system folders. This message can be used to update the owner's preference files prior to `FindFolder` being redirected.

```
kFolderManagerNotificationMessageUserLogOut
```

Sent when a user has logged out. This is the last time `FindFolder` will return user's folders&emdash;after this notification `FindFolder` will return the `vRefNum` and `dirID` of system folders. This message can be used to update a user's preference files during logout.

```
kFolderManagerNotificationMessagePostUserLogOut
```

Sent just after `FindFolder` has been restored to return the `vRefNum` and `dirID` of system folders. This message can be used to load the owner's preferences.

Use the following APIs to register and unregister your notifications:

```
OSErr FolderManagerRegisterNotificationProc(
            FolderManagerNotificationUPP notifyProc,
            void * refCon,
            UInt32 options);

OSErr FolderManagerUnregisterNotificationProc(
            FolderManagerNotificationUPP notifyProc,
            void * refCon);
```

Use `FolderManagerRegisterNotificationProc` to register your notification function with the Folder Manager. The `notifyProc` parameter is the pointer or UPP to your notification function. The `refCon`

parameter is for your own use&emdash;this value will be passed to your notification function each time it is called. The options parameter specifies registration options.

Currently, the only option is specified by the `kDoNotRemoveWhenCurrentApplicationQuitsBit` constant. By setting this bit, you tell the Folder Manager to not remove your notification function when the current application quits. Otherwise, a notification function registered within an application's context will be automatically removed when that application quits. Programs that register notifications at system startup should set this bit. Most likely you will need to set this bit, since applications won't normally need to receive Folder Manager notifications.

Use `FolderManagerUnregisterNotificationProc` to remove your notification function from the Folder Manager's queue. The `notifyProc` parameter is the pointer or UPP to your notification function that you passed to the `FolderManagerRegisterNotificationProc` function. The `refCon` parameter should be the same value that you passed to the `FolderManagerRegisterNotificationProc` function.

The prototype of your notification function should be:

```
typedef OSStatus (*FolderManagerNotificationProcPtr)(
            OSType message,
            void * arg,
            void * userRefCon);
```

This function will be called for all Folder Manager notifications. The message parameter will contain the type of notification (user login, user logout, etc.), the `arg` parameter will contain a pointer to additional information (if any), and the `userRefCon` parameter is a value provided when the function was registered. The `userRefCon` is for your own use and can be any value you want (such as a pointer to your globals or other state information).

In the case of the four notifications cited above, the arg parameter is a pointer to a `FindFolderUserRedirectionGlobals structure`:

```
struct FindFolderUserRedirectionGlobals {
        UInt32 version;
        UInt32 flags;

        Str31  userName;
        short  userNameScript;

        short  currentUserFolderVRefNum;
        long   currentUserFolderDirID;

        short  remoteUserFolderVRefNum;
        long   remoteUserFolderDirID;
};
```

There is an additional fifth notification which is very different from the first four and will be of use only to a few developers:

`kFolderManagerNotificationDiscardCachedData`

Sent by third-party software when the entire Folder Manager cache should be flushed

In that case, `arg` is undefined.

Back to top

Is Multiple Users Installed?

If you wish to be more than just compatible with the Multiple Users technology then it is crucial to know whether it is installed on your users system of not. In order to check, use the `Gestalt` API and the defined constant `gestaltMultipleUsersState` (`'mfdr'`):

```
GestaltRecHand result;
if ( Gestalt(gestaltMultipleUsersState, (long*) &result) == noErr &&
     (*result)->giVersion > 0)
      {
      // do stuff
      }
```

Note that the result of the `Gestalt` call is a handle to the following structure. Do **not** dispose of this handle.

The current version of the `GestaltRecHand` structure for Multiple Users is 6 (`gestaltMultipleUsersStateVers`) and the structure itself is defined as:

```
#pragma options align=mac68k

typedef struct { // gestalt info structure

 // Version 1 fields.

 short         giVersion;              // structure version;
                                       // (0 = invalid)

 short         giReserved0;            // [OBSOLETE]
 short         giReserved1;            // [OBSOLETE]
 short         giReserved2;            // [OBSOLETE]
 short         giReserved3;            // [OBSOLETE]
 FSSpec        giReserved4;            // [OBSOLETE]

 // Version 2 fields.

 short         giDocsVRefNum;          // vrefnum of user's documents
                                       // location (only valid if not
                                       // on floppy)

 long          giDocsDirID;            // directory id of user's
                                       // documents folder (only valid
                                       // if not on floppy)

 short         giForceSaves;           // true if user is forced to
                                       // save to their documents
```

```
                                    // folder

short         giForceOpens;         // true if user is forced to
                                    // open from their documents
                                    // folder

Str31         giWorkgroupName;      // name of current workgroup
                                    // (for Macintosh Manager)

Str31         giUserName;           // name of current user

Str31         giFrontAppName;       // name of the frontmost
                                    // application

short         giReserved5;          // [OBSOLETE]

short         giIsOn;               // true if Multiple Users or
                                    // Macintosh Manager is on
                                    // right now

// Version 3 fields. - There were no additional fields for version 3
// Version 4 fields.

short         giUserLoggedInType;   // logged in user type

char          giUserEncryptPwd[16]; // encrypted user password
                                    // (our digest form)

short         giUserEnvironment;    // environment that user has
                                    // logged into

long          giReserved6;          // [OBSOLETE]
long          giReserved7;          // [OBSOLETE]

Boolean       giDisableScrnShots;   // true if screen shots are not
                                    // allowed

// Version 5 fields.

Boolean       giSupportsAsyncFSCalls; // Finder uses this to tell if
                                    // our patches support async
                                    // trap patches

short         giPrefsVRefNum;       // vrefnum of preferences

long          giPrefsDirID;         // dirID of the At Ease Items
                                    // folder on preferences volume

unsigned long giUserLogInTime;      // time in seconds when user
                                    // logged in (0 or 1 mean not
                                    // logged in)

Boolean       giUsingPrintQuotas;   // true if logged in user is
                                    // using printer quotas

Boolean       giUsingDiskQuotas;    // true if logged in user has
                                    // disk quotas active

// Version 6 fields

Boolean       giInSystemAccess;     // true if system is in System
                                    // Access (i.e., owner logged
```

```
                                        // in)

 Boolean       giUserFolderEnabled;     // true if FindFolder is
                                        // redirecting folders (uses
                                        // giUserName for user)

 short         giReserved8;             // [OBSOLETE]
 long          giReserved9;             // [OBSOLETE]

 Boolean       giInLoginScreen;         // true if no user has logged
                                        // in (including owner)

 //
 // May have more fields added in future,
 // so never check for sizeof(GestaltRec)
 //
} GestaltRec, *GestaltRecPtr, **GestaltRecHand;

#pragma options align=reset

// Possible values for giUserLoggedInType
// (only applicable for AEFW/MM, not Multiple Users)
enum {
 kMUUserTypeNormal      = 0, // default is a normal user
 kMUUserTypeWGAdmin     = 1, // Workgroup Administrator type
 kMUUserTypeGlobalAdmin = 2  // Global Administrator type (superuser)
};

// Possible values for giUserEnvironment
enum {
 kMUEnvironmentPanels       = 0, // Panels environment
 kMUEnvironmentFinder       = 1, // Normal (or Unrestricted Finder)
                                 // environment
 kMUEnvironmentFinderSecure = 2  // Limited (or Restricted Finder)
                                 // environment
};
```

Note:
At Ease and the Macintosh Manager also register the `'mfdr' Gestalt` constant but use earlier versions of the `GestaltRec` structure. Be sure to always check its version (`giVersion`) field before accessing the other fields.

Back to top

Is Multiple Users Running?

In order to determine whether Multiple Users is active on your users machine, check for the existence of the Multiple Users `Gestalt` and look at the `giIsOn` field to see if Multiple Users is active.

```
Boolean IsMultipleUsersOn (void)
{
 GestaltRecHand result;

 if (Gestalt (gestaltMultipleUsersState, (long *) &result) == noErr &&
     (*result)->giVersion >= 2 && (*result)->giIsOn)
     return true;
 else
     return false;
}
```

Back to top

Who is Logged In?

How can I tell if no user is currently logged in?

You should make sure Multiple Users is active and then look at the `giInLoginScreen` field to see if the machine is currently in the login screen. Or register your notification function, and in between a login notification and a logout notification, you know that a user is logged in.

How can I tell if a User other than the Owner is logged in?

Use the `Gestalt` API as described above. If `giUserLogInTime` is > 1 then a User other than the Owner is logged in. If the Owner is logged in, then `giUserLogInTime` will be 0 and `giInSystemAccess` will be true.

```
Boolean IsUserLoggedIn (void)
{
 GestaltRecHand result;

 if (Gestalt (gestaltMultipleUsersState, (long *) &result) == noErr &&
     (*result)->giVersion >= 5 && (*result)->giIsOn &&
     (*result)->giUserLogInTime > 1)
     return true;
 else
     return false;
}
```

Back to top

User Names and Passwords

Many developers may be interested in knowing what their user's name and password are, and how secure the password is.

In order to get the user name, check the `giUserLoginTime` field to make sure it's greater than 1. If so, then check the `giUserName` field to obtain the user's name. Alternatively, you can register your notification function and catch the login notification, which will indicate the user name in the `FindFolderUserRedirectionGlobals`. However, you should note that when the owner logs in, the globals structure will indicate an empty string for the user name in this case.

```
void GetUserName (Str255 userName)
{
 GestaltRecHand result;

 userName [0] = 0;
 if (Gestalt (gestaltMultipleUsersState, (long *) &result) == noErr &&
     (*result)->giVersion >= 5 && (*result)->giIsOn &&
     (*result)->giUserLogInTime > 1)
     BlockMoveData ((*result)->giUserName, userName,
                    (*result)->giUserName [0] + 1);
}
```

With regards to the security of the password, it is stored hashed in the Multiple User database and is never in a form that can be decrypted.

NOTE: The user's password is not available.

Back to top

Additional Notes & Warnings

- If you are a good NetBoot citizen (see Technote 1151) then you have a good chance to be a good Multiple Users citizen.

- The following is not a Multiple Users problem per se but is important to all applications. The very first application launched by the system after the boot can be either the Finder, At Ease, Login, or Panels and is referred hereby as the shell:
    - Some processes are allocating memory to store data or code in the shell's heap, or in some cases, the heap of the first process to launch, or making a copy of state variables, such as the `A5` value of this other process.
    - Usually, this particular process is the Finder and it does not go away until the User chooses **Restart** or **Shut Down** which is why this practice hasn't been a huge problem in the past.
    - When Multiple Users is running, the shell or the first process to launch is going to go away as soon as the User logs in.
    - So any memory allocated in that heap is going to be purged. If data is stored there, it's going to be lost, if code is stored there, it's going to cause a crash as soon as the process who installed it is going to try to execute it.
    - Therefore, allocate memory only in your heap if it's going to be around long enough, or in the System heap if you can't help it but do NOT allocate memory in the shell or in the first process to launch any more and do not assume that you can keep a valid copy of state variables of this process.
- When using Macintosh Manager, the Preferences folder is redirected to a server volume. All connected machines may not be using the same version of a particular application. The different versions of that application may be using a different preferences file format. That file is going to be rewritten each time by the different versions. That file is also going to be read by the different versions. That means that the application has to be ready to deal gracefully, not only with previous formats but also with newer formats.

Back to top

Summary

To be compatible with the Multiple Users technology, always use the `FindFolder` API (using all returned values) and either be ready to handle `afpAccessDenied` (`-5000`) errors when you attempt I/O calls, or test beforehand the read/write permission of the folder. Additionally, do not assume that the Finder is always running, or that any Finder information you got at one time is still valid later on.

To take advantage of the Multiple Users technology, register the new notification functions, and use the Gestalt call with the `gestaltMultipleUsersState` parameter.

Further References

- Technote 1151: Creating NetBoot Server-Friendly Applications
- Technote 1134: The Preferences Problem

Back to top

Downloadables

 Acrobat version of this Note (32K).

Back to top

To contact us, please use the Contact Us page.
Updated: 01-June-2000

Technotes | Contents
Previous Technote | Next Technote