Technote 1134

The Preferences Problem

By Mark Cookson
Apple Worldwide Developer Technical Support

CONTENTS

Preferences: nearly every application has them, but there is no Mac OS API specifically designed to help you deal with them. This requires you to write a fair amount of code to deal correctly with preferences.

This Technote will attempt to outline the problems with preferences files and the solutions to those problems so that you can code a robust solution.

Often a problem has multiple solutions, and none of them is the obvious best-choice solution for all possible cases. This Note lists the top options, and discusses each solution so that you can make an informed decision about which to implement.

This Note is directed at developers who have a preferences file, or files, and who want to make sure that they are working correctly with those files by making sure they are covering all reasonable possibilities.

The Preferences Problem

Preferences files: nearly every application of substance has them, but they are often the cause of great pain and frustration when they should make your life simpler.

What are the major problems relating to preference files?

In the real world it is impossible to solve all of these problems with one solution that will always be the correct one. With that in mind, we will attempt to give you the information that you need to come as close as possible to an ideal solution for your application.

Issue 18 of develop Magazine included an article called "The Write Way to Implement Preferences Files." DTS does not believe you should follow the recommendations in that article. This Technote supersedes that article. If, for historical interest, you wish to see what it is we are recommending you avoid, you can download the article.

Discussion

How you store your preferences inside a file is up to you. You can store your preferences as a flattened structure that your application will read directly into a handle when it starts, or you can store it in a much more complicated form.

What should I name my preferences file(s)?

The name of your preferences file is completely up to you. The only suggestion that DTS has is that it **not** end with "Prefs": use "Preferences" instead. This way the user has a clear idea that the file contains SurfWriter preferences and not some mysteriously essential data which might be dangerous to delete.

This seems like a trivial point, but unless your application's name is so long that "Preferences" can't be tacked onto the end of it, you should avoid the contraction. You only have to type it once (presumably into a `'STR#'` resource), so making it verbose shouldn't be a real problem.

Where do preferences file(s) go?

The easy answer is that you should put them in a file or files in the Preferences folder, as identified by a call to `FindFolder`.

If you have only one preferences file, you should put that file into the Preferences folder (returned from `FindFolder`). However, if you have multiple preferences files, or other files, such as log files, that you want to put into the Preferences folder, you should create a folder in the Preferences folder, and store all of your files in that folder. This reduces clutter in the Preferences folder.

When your application needs more than one preferences file, consider creating a folder within the Preferences folder and putting your preferences files in this additional folder. However, storing everything in the Preferences folder hierarchy is not best for all applications. For example, in a multiple-user situation, you may want to have a preferences file stored in each user's personal folder rather than have a large number of files kept in the Preferences folder.

Another option that some programs already implement is the concept of "local preferences". A local preference is one that is stored in the same folder as the application. The application first looks for a local preferences file, and if it does not find a local one, it then looks for a preferences file in the Preferences folder. This add complexity to the preferences search code, but for that added complexity your users gain the ability to easily:

- Run two (or more) copies of the application simultaneously with different preferences.
- Allow applications to load, save and store preferences even if the volume which contains the Preferences folder is locked (booting off a CD for example).
- Use a set of preferences quickly and revert to the original preferences (by copying and deleting local preferences).

What file type and creator should I use for my preferences file(s)?

What file type?

You should give your preferences file the type `'pref'`, which is preferred. Using this type will allow for future versions of the Finder to auto-route your preferences file to the Preferences folder. The only reason we can think of for not using a file type of `'pref'` is that files with that file type get the Finder's Balloon Help message:

> Finder Preferences
> This file stores preferences settings for the Finder.

This might be confusing for the user, since nearly every preferences file has a type of `'pref'`. It is the opinion of DTS that Engineering should fix the Finder instead. We've filed a bug report; its ID is 2241857.

If you have files other than preferences files that you are storing in the Preferences folder (or a folder in the Preferences folder), feel free to use any type that you wish. For instance, if you have a log file in the Preferences folder, it is perfectly reasonable to make it of type `'TEXT'`, since it is obviously not a preferences file and opening it might be a reasonable thing to do.

In summary, if it's a preferences file, it should have a type of `'pref'`; if it isn't a preferences file, it shouldn't, even if it is kept in the Preferences folder.

What creator code?

There are three schools of thought on this issue. The first claims you should give your preferences file a creator code that matches your application signature. The second claims that you should give it a creator of `'????'`. The third claims you should give it a (registered) creator code which is neither `'????'` nor the same as your application signature.

Using your application's creator code

There are good reasons why you would want to give your preference file the same creator as your application signature. They are (in no particular order):

- It is reasonably easy to find a preference file with a specific creator code, even if the user changes its name.
- Utility applications, such as ones which delete "orphaned" preferences files, know that the file belongs to your application.
- Localization is easy because the creator code does not change when your application is localized. Your code doesn't need to know the name of its preferences file, which simplifies localization.

However, there is also a reason why you would not want to use your application signature as the creator code of your preferences file:

- Your application will be launched by a user opening your preference file, and if this happens, you should probably do something in response. See the discussion below for details.

Using a creator code of '????'

Of course, there are also some benefits to not using your creator code, and using a creator code of '????' instead. They are:

- It is very easy to find a file by name (click here for a snippet showing how).
- You don't have to register a '????' as a creator code. (In fact, you can't; it's explicitly defined as the signature of no application.)
- Your application will not be launched by a user attempting to open your preference file.
- You can have a application-missing message string to tell the user why they can't open your preference file.

There are, however, some drawbacks to using a creator code of '????':

- You have to find your preference file by name, which could mean localization issues.
- Utility applications could flag your preference file as belonging to an application no longer on the user's system, and the user might delete it. The utility could also be fooled and not clean up your preferences file when your application is deleted.
- When you use a creator code of '????', you won't be able to locate your preferences file by creator code, because there may be multiple files with that creator code.

Using a different creator code

By using a creator code that is different from your application's signature, but not '????', you get these benefits:

- It is reasonably easy to find a preference file with a specific, non-generic, creator code.
- There are no localization issues with respect to a creator code.
- Your application will not be launched by a user attempting to open your preference file.
- You can have a application-missing message string to tell the user why they can't open your preference file.

However, this approach has a slightly different set of disadvantages. They are:

- You must register a second application signature for use as your preference file's creator code to avoid collisions with other applications.
- Utility applications will still identify your preferences file as an orphaned preferences file.

Note: There is code at the end of this Note that shows how you might want to locate a preferences file by creator code and/or file name. You may find this a good start for your own search code.

A word on localized preferences

There is the possibility that the user of your application will use a preferences file that was not created with the same language code as the currently running system. For instance, a U.S. user may have sent their preferences to a Japanese colleague.

If your application finds its preferences based on a creator code and not a name, then this combination of Japanese application and U.S.-named preferences file will work as the user expected. If your application always opens the preferences file by name, or requires that the name be a specific string, then you prevent the easy duplication of preferences files, and make it difficult for users to make copies of their preferences files for rapid switching between different preference sets.

A word on application-missing messages

The application-missing message string is a `'STR '` resource ID -16397 put in a file by the creating application, which the Finder uses to tell the user why a file could not be opened. The message should tell the user which application created the file and the purpose of the file. This string should be used if you give your preferences file (or any other file) a creator code that is not the same as your application signature. This way, when the user attempts to open your preferences file they will receive a message such as:

> "This document describes user preferences for the application SurfWriter. You cannot open or print this document. To be effective, this document must be stored in the Preferences folder in the System Folder."

There is also a missing-application name string which is a `'STR '` resource ID -16396. This resource is simply the name of the application that created the file ("SurfWriter") which the Finder will use to tell the user which application created a specific file when that application cannot be found. Normally this string is used only for documents that are meant to be opened by the user.

> Important: Do not use both the application-missing message string and the missing-application name string; use one or the other, but not both. For more information on application-missing messages, see Inside Macintosh: Macintosh Toolbox Essentials, Chapter 7 - Finder Interface, using the Finder Interface Displaying Messages When the Finder Can't Find Your Application.

How do I find my preferences file(s)?

Finding your preferences file is simple if you find it by name; it's either there or it isn't. If you find your preferences file by creator code, you will have to search the entire Preferences folder for a file of the correct type. You may want to also match it by name in case multiple files have the same creator code, which is especially likely to happen if the creator code you use is `'????'`.

To search for files with a specific creator code, use indexed calls to `PBHGetFInfo` to have it return you the Finder information about each file in a given directory. Remember that after each call to `PBHGetFInfo`, you have to reset `ioDirID` field of the parameter block because `PBHGetFInfo` changes the value of this field.

Should I have a `'vers'` resource in my preferences file(s)?

Yes, you should. Every file that is part of your application should have a `'vers'` resource ID 1 and 2 in it as a matter of course, and preferences files are not an exception.

The immediate benefit of `'vers'` resources is that a user will see a more complete Get Info dialog in the Finder: one showing the application's name and version that belongs to the file. A future benefit of the

'vers' resource is that it can come in handy when a new version of your program has to identify and convert an older version's preferences settings to a new format.

How do I give my preferences file(s) the right icon?

This is easy; if you give your preferences file a type of 'pref', it will automatically get the correct icon (the default preferences file icon). You should not have an entry of 'pref' in your application's bundle resource. Furthermore, having files with generic icons makes opening the Preferences folder a lot faster.

We discourage developers from trying to have preferences files with non-generic (custom) icons. However, if you want a custom icon, use a custom Finder icon so that your preferences file will have an icon even if your application is deleted. This may help the user determine that they can now delete your preferences file. Another reason for using a Finder custom icon is that the Finder only has to open one file (the preferences file) whose location it knows, rather than searching the desktop database on every volume, including volumes like AppleShare volumes mounted over Apple Remote Access, which may be very slow.

For information on how to create a custom Finder icon, see Inside Macintosh: Macintosh Toolbox Essentials, Chapter 7 - Finder Interface, using the Finder Interface, Creating Customized Document Icons.

What should I do if the user double-clicks a preferences file?

This is a problem only if you give your preferences file the same creator code as your application signature. There are several thoughts on what to do if your application is asked to open a preferences file:

1. You could put up a dialog saying that you don't open preferences files:
    - By putting up a dialog saying that you don't open preferences files, the user has been able to launch your application, and now they know what file they just "opened."
    - Some users may find such a dialog distracting.
2. You could open your preferences dialog:
    - Opening your preferences dialog is a cool idea and seems very Mac-like, but it also has some issues which need to be worked out, the scope of which is beyond a general purpose document such as this Technote.
        - From which file do you extract and display preferences values: the current file or the newly opened file?
        - When the user clicks OK, where do you save any changes? Do you put them in the existing preferences file, or the newly opened preferences file, or both files?
        - What do you do if the newly opened preferences file is on a server and you can't write to it?
        - If some or all of the settings you store in a preferences file are not set via an explicit preferences dialog, how do you inform the user those preferences have changed?
3. You could do nothing:
    - Doing nothing is certainly the simplest thing that you can do.
    - Make sure that you don't crash (you will probably need to filter out files of type 'pref' from your application's "open documents" Apple Event handler).
    - However, we believe this is actually a bad user experience -- you should provide the user with some evidence that you recognize that they have double-clicked on a file.

There are no real disadvantages to any of these methods other than the user experience that they offer,

and only you are able to determine what user experience to deliver (since Apple has no official position on this subject).

What should I do if a preferences file doesn't exist?

This would seem to be an easy question to answer; however, there are two different opinions on this subject.

The opinions are:

1. Do not create one, yet set all of your preferences in your application as if one existed and every setting was set to your default value. If the user changes the preferences, then create a preferences file with these values.
2. Create a preferences file and fill it in with default values.

The obvious advantage to the first approach is that if the user never changes the preferences from their default values, they never have a preferences file wasting space on their hard drive. This also makes it easy for someone to install your application, try it, and then delete it without having to search their system for orphaned files. This point is especially directed at developers whose software is distributed on a trial-use basis.

Always creating a preferences file makes for slightly simpler code, but since you have to be prepared to run even if you cannot create a preferences file, you might as well wait until you actually need to create a preferences file before dealing with those possible problems.

One of the problems of creating a preferences file is that you may not be able to create it because the volume or system folder is locked. For ideas of what to do in this situation, see the following section on what to do if the file is on a locked volume or the preferences file is locked.

There is also the problem of what to do if another file or folder exists where you would want to save your preferences file or folder. For ideas when in this situation, see the section on what to do if your preferences file is usurped by another application.

What should I do if the preferences file is not writable?

If your application is not able to save or update a preferences file, what do you do?

- You could refuse to run, but that's a bad solution, and should be your last resort, rather than your first option.
- You could run and just not allow the user to save their preferences.

However, what if the user wants to use your program with new preferences?

- You should probably allow the user to change all possible preference settings, but warn them that they will not be able to save them for use at a later time. Your program should run with the new preferences settings.
- If you allow your user to have their preferences file stored in any arbitrary folder, you may prompt the user for an alternate location to store the preferences file rather than telling them that the preferences will not be saved.

What should I do if a preferences file(s) are corrupt?

Many applications have preferences files that are merely a resource fork that contains one or more custom resource data structures that the program reads in during its launch. If you have a preferences file based on the Resource Manager, you are subject to its limitations and you should be aware of them.

One of the most important things to know when relying on the Resource Manager is that the current Resource Manager doesn't perform the most robust sanity checking before opening a resource file. If your resource file has become corrupted in such a way as to cause the Resource Manager to crash or produce an error, the user will not have any way of discovering the problem. Deleting the offending preferences file may be one of the last things that they try.

For this reason, you may want to perform sanity checking on the resource map of your preferences file before you attempt to open it through the Resource Manager. Code for this is provided in the public domain as part of Internet Config; look for the file called "ICResourceForkSanity.c" in the IC Programmers Kit.

If your preferences file is corrupted, you should alert the user to that fact, and you should not take any immediate action that isn't easily reversible. It may be that your preferences have been usurped by another application and merely *appear* to be corrupt from the perspective of your application. In that case, you don't want to delete the file. Allow the user to have the final decision as to what action should be taken to correct the problem, though you are free to suggest the course of action you believe to the be the best.

If your preferences file has not been usurped and is indeed corrupt, you should attempt to continue running, either with default settings, or with backed-up settings. Forcing the user to quit and delete your preferences file is frustrating for the user. Your application may be perfectly usable with the default settings, and the user may not want to spend the time fixing the problem immediately if they are in a hurry to perform a simple task.

As an example of a good thing to do, Internet Config makes a copy of its preferences file's resource fork and stores it in the data fork of the preferences file. If the resource fork turns out to be corrupt, it deletes the resource fork, copies the saved preferences from the data into the resource fork, and continues. This is a complicated solution and it doubles the size of your preferences file, but usually that is not a significant amount of disk space.

If you would like to attempt such a solution in your application, the code for Internet Config is in the public domain; you are free to use it in your application.

What should I do if my preferences file(s) are usurped by another application?

If another file or folder exists where you would want to put your preferences, you should alert the user to this fact, but you should also be able to continue gracefully.

If possible, attempt to save your preferences file in some alternate location. If there is a permanent conflict, the user doesn't have to choose between your application and some other application, or use one of the applications without their preferred preferences.

One of the easiest workarounds to this problem is to locate your preferences file by a unique creator code (your application's signature springs to mind), which allows the user to rename your preferences file and not affect your application at all.

If you choose to locate your preferences file by name, putting a registered string (such as "Apple") in it may help to reduce the likelihood that it will conflict.

Overwhelmed?

If you are overwhelmed by all these choices, DTS prefers that:

- Your application should have a single preferences file which has the same creator code as your application's signature, because every file that belongs to an application should have a creator code matching the application's signature. (If your application creates files intended for use by another application, you should of course use that application's signature as appropriate.)
- Your preferences file should be stored in the Preferences folder (as returned by `FindFolder`).
- Your preferences should be of type `'pref'`.
- Your preferences file should have a generic icon.
- If the user attempts to open your preferences file, your application should present its preferences dialog populated with the current settings, not those of the file being double-clicked.
- Your application should not create a preferences file until a preference is changed.
- If your preferences file is not writable, you should use any settings you can read. Allow the user to modify the current preferences, but alert them that they will not be able to save these new settings.
- If your preferences file is corrupt, use your default values and continue running. If possible, attempt to recover your preferences from another location.
- If your preferences file is usurped by another application, use your default values and continue running. Allow the user to change their preferences. If possible, save your preferences in another location or in the Preferences folder with another name.
- You should give your preferences file an application-missing message string.

Summary

Maintaining a preferences file isn't as simple as it seems at first glance. There are options to consider and trade-offs to be weighed, but because nearly every application must have a preferences file, correct maintenance is vitally important.

In this Note, you have learned of all the possibilities we could think of and been shown their merits and flaws. Now it is up to you to decide what is right for you and your users.

Appendix A: Some Sample Code

Here is some code that shows how to find a preference file by file name and/or creator code:

`FindFolder` returns `fnfErr` (-43) if it can't find the specified folder, or `dupFNErr` (-48) if a file exists where the folder should. `FindPrefsFile` takes as long as the last parameter, which is an index into an `enum` which will tell you more about what happened if an error is returned (sometimes even if `noErr` is returned). You can use this value to better explain the error that is returned to the user (if you choose to tell the user anything). For instance, you could use this value as an index into a `'STR#'` resource which explains the error to the user.

```
/*
**  Apple Macintosh Developer Technical Support
**
**  Routine demonstrating how to find a preferences file by creator code
**  and/or file name.
**
**  by Mark Cookson, Apple Developer Technical Support
**
**  File:   FindPrefsFile.c
```

```
**
**  Copyright ©1996 Apple Computer, Inc.
**  All rights reserved.
**
**  You may incorporate this sample code into your applications without
**  restriction, though the sample code has been provided "AS IS" and the
**  responsibility for its operation is 100% yours.  However, what you are
**  not permitted to do is to redistribute the source as "Apple Sample
**  Code" after having made changes. If you're going to re-distribute the
**  source, we require that you make it clear in the source that the code
**  was descended from Apple Sample Code, and that you've made changes.
*/

#include <Types.h>
#include <Folders.h>
#include <Files.h>
#include <Errors.h>
#include <TextUtils.h>

#include <assert.h>

enum {
    noError                 = 0,
    noFileExists            = 1,
    notRightName            = 2,
    fileWithNameNotCreator  = 3,
    folderInsteadOfFile     = 4,
    tooManyFiles            = 5,
    findFolderErr           = 6
};

OSErr   FindPrefsFile (OSType creatorCode, Str63 prefsName,
  FSSpec * prefsFSSpec, long *result);

/*
    This function attempts to find a file in the Preferences folder by creator
    and file name.

    * Overview:

    The idea is to find the preference file by creator code, using the name only
    as a fallback search method.

    If only one file with the right creator code is found, then return the FSSpec
    to that file, regardless of its file name.

    If multiple files with the same creator code are found, then see if one of
    those files has the right name, if it does then return an FSSpec to that file.

    If no files with the correct creator code are found, then check to see if a
    file with the correct name exists, if it does return a dupFNErr and a FSSpec
    to the offending file, or if it doesn't then return a fnfErr and a FSSpec to
    where the file should be created.

    The function also returns an enum that tells more about what happened.  Use
    this value to better inform the user (if needed) about what went wrong.  You
    could use this value as an index into a STR# resource, for instance.

    * Implementation:

    There is a short-circuit test that looks for a file with the correct file
```

```
    name first, and if it finds that, it checks that its file type is the correct
    file type.  If the two match, then no other searching is done and the FSSpec
    to this file is returned.

    If no file with that name and creator is found, then every file in the
    Preferences folder is searched to get its creator code.  If a file with a
    matching creator code is found, it is copied into a temporary FSSpec and the
    search continues.  If another file with the correct creator code is found
    then the search is aborted and a fnfErr is returned.  The FSSpec that is
    returned is set to be where the correctly named file would go.

    If only one file is found with the correct creator code, it is
    returned in the FSSpec and the function returns noErr, regardless of
    its name.
*/

OSErr   FindPrefsFile (OSType creatorCode, Str63 prefsName, FSSpec *
   prefsFSSpec, long *result) {
    HParamBlockRec      hpb;
    FSSpec              matchFSSpec;
    SInt32              foundPrefDirID      = 0;
    UInt32              numMatches          = 0;
    SInt16              foundPrefVRefNum    = 0;
    OSErr               err                 = noErr;
    Str63               fileName            = "\p";
    Boolean             shortCircuit        = false,
                        foundFileName       = false,
                        foundDirectory      = false;

    assert (prefsName != nil);
    assert (prefsName[0] <= 63);
    assert (prefsFSSpec != nil);

    // Find the Preferences folder to begin our search.
    err = FindFolder (kOnSystemDisk, kPreferencesFolderType,
    kDontCreateFolder, &foundPrefVRefNum, &foundPrefDirID);

    // Short circuit the search and see if a file with the right name
       and creator code exists.
    if (err == noErr) {
        BlockMoveData (prefsName, fileName, prefsName[0] + 1);
        hpb.fileParam.ioCompletion = nil;
        hpb.fileParam.ioNamePtr = fileName;
        hpb.fileParam.ioVRefNum = foundPrefVRefNum;
        hpb.fileParam.ioDirID = foundPrefDirID;
        hpb.fileParam.ioFDirIndex = 0;

        // This allows us to return a FSSpec to where the file would be
        // if we end up not finding one.
        err = FSMakeFSSpec (foundPrefVRefNum, foundPrefDirID, prefsName, prefsFSSpec);
        if (err == noErr) {
            foundFileName = true;
            err = PBHGetFInfoSync (&hpb);

            if (err == noErr) {
                if (hpb.fileParam.ioFlFndrInfo.fdCreator == creatorCode) {
                    // We found the file we were looking for.
                    shortCircuit = true;
                    numMatches = 1;
                    if (result != nil) {
                        *result = noError;
```

```
                }
            }
        } else {
            // If PBHGetFInfoSync returns an error, it's probably because we asked
            // about a directory.
            foundDirectory = true;
            err = noErr;    // Continue the search by creator type.
            if (result != nil) {
                *result = folderInsteadOfFile;
            }
        }
    } else {
        err = noErr;    // Continue the search by creator type.
    }
} else {
    if (result != nil) {
        *result = findFolderErr;
    }
}

// The exact file didn't exist so try to find it by creator type.
if (err == noErr && shortCircuit == false) {
    // Find all files in the preferences folder and see if some have
    // the specified creator code.
    do {
        // Check the file's creator code to see if it is the right one.
        hpb.fileParam.ioFDirIndex += 1;
        hpb.fileParam.ioDirID = foundPrefDirID;
        err = PBHGetFInfoSync (&hpb);

        // Assume the file doesn't exist.
        if (result != nil) {
            *result = noFileExists;
        }

        if (err == noErr && hpb.fileParam.ioFlFndrInfo.fdCreator == creatorCode) {
            // This file has the right creator code, so we want to remember it.
            if (numMatches == 0) {
                matchFSSpec.vRefNum = foundPrefVRefNum;
                matchFSSpec.parID = foundPrefDirID;
                BlockMoveData (fileName, matchFSSpec.name, fileName[0] + 1);
                numMatches = 1;
            } else {
                // We found another file with the same creator code but wrong
                // file name.
                // We won't know which file to return a reference to, so stop
                // the search.
                numMatches = 2;
            }
        }
    } while (err != fnfErr && numMatches < 2);
}

if (numMatches == 0) {
    if (foundDirectory == true) {
        err = notAFileErr;
        if (result != nil) {
            *result = folderInsteadOfFile;
        }
    } else if (foundFileName == true) {
        err = dupFNErr;     // A file with the right name, but the wrong creator code
```

```
                                // exists!
            if (result != nil) {
                *result = fileWithNameNotCreator;
            }
        } else {
            err = fnfErr;
        }
    } else if (numMatches == 1 && shortCircuit == false) {
        // We found exactly one match, so this is the right prefs file.
        BlockMoveData (&matchFSSpec, prefsFSSpec, sizeof (FSSpec));
        err = noErr;
        if (result != nil) {
            *result = notRightName;
        }
    } else if (numMatches > 1) {
        // There were multiple files with the same creator found,
        // which means none had the right name
        // (the correct one would have been caught by the short-circuit
        // test), so we return fnfErr.
        err = fnfErr;
        if (result != nil) {
            *result = tooManyFiles;
        }
    }

    return err;
}
```

This snippet shows how to find a file by name in the Preferences Folder.

`FindFolder` will return `fnfErr` (-43) if it can't find the System Folder or the Preferences folder (and you are using `kDontCreateFolder`, as this code does). `FSMakeFSSpec` will return `fnfErr` if a file or folder doesn't exist with the specified name. If no file exists, you will have to create your preferences file before you can open it.

Note: `FSMakeFSSpec` will return a valid `FSSpec` if a folder with the specified name exists. When you attempt to open the returned `FSSpec`, make sure that you can deal with the open call failing.

```
#include <Types.h>
#include <Folders.h>
#include <Files.h>
#include <Errors.h>
#include <TextUtils.h>

#include <assert.h>

enum {
    // Choose an unused error number that the calling code will know how to handle.
    findFolderErr   = 128
};

OSErr FindPrefsByName (FSSpec * prefsFSSpec) {
    OSErr               err                 = noErr;
    SInt32              foundPrefDirID      = 0;
    SInt16              foundPrefVRefNum    = 0;
    Str255              prefsName;

    assert (prefsFSSpec != nil);

    // Get the preferences file name.
    GetIndString (prefsName, kPreferencesSTRRes, kPrefsNameSTRIndex);

    if (prefsName == nil) {
        // If the prefs name's string is nil then the resource probably doesn't exist.
        err = resNotFound;
    }

    if (err == noErr) {
        // Find the Preferences folder to begin our search.
        err = FindFolder (kOnSystemDisk, kPreferencesFolderType,
        kDontCreateFolder, &foundPrefVRefNum, &foundPrefDirID);
    }

    if (err == noErr) {
        // Make an FSSpec to where the preferences file should be.
        err = FSMakeFSSpec (foundPrefVRefNum, foundPrefDirID, prefsName, prefsFSSpec);
    } else {
        err = findFolderErr;
    }

    return err;
}
```

Further References

- Inside Macintosh: Macintosh Toolbox Essentials, Chapter 7 - Finder Interface
- The Creator code registration web page

Downloadables

 Acrobat version of this Note (51K).

 Internet Config source

Acknowledgments

Thanks to Quinn, Pete Gontier, Ingrid Kelly, and Rich Kubota.

To contact us, please use the Contact Us page.
Updated: 19-Oct-98