Technote 1121

Mac OS 8.1

Version 1.0

Mac OS 8.1 introduces a number of new and/or updated technologies. This Technote documents the changes that most likely will affect developers.

Contents

NOTE
Many of the changes mentioned here are accompanied by corresponding changes to interface files. These updated interface files will be available in the next release (after 3.0.1) of Universal Interfaces.

Finder 8.1

Finder 8.1 changes fall into three categories:

- New features
- Enhancements to existing features
- Bug fixes

Finder 8.1 also defines two new bits in its Gestalt selector.

New features:

- Get Info windows for disk volumes show the volume format.
- Folder windows have a "sort order" button which toggles the order of list views. Sort order is also accessible through the AppleScript property "sort direction" of class "container window".
- A key sequence of 'command-shift-W' will close any window, including pop-up windows. A key sequence of 'command-shift-option-W' will close all windows, including pop-up windows. Either of these key sequences will return a pop-up window to its "normal" state before closing it.
- Support for "unconventional" desktop printing via LaserWriter 8. See Technote 1112, "Introducing the LaserWriter Driver Version 8.5.1" .
- The `bSupportsAsyncRequests` bit of the `vMAttrib` field of the `GetVolParmsInfoBuffer` structure is honored. File systems that set this bit are guaranteeing that they can correctly handle asynchronous File Manager requests.
- The `vmVolumeGrade` field of the `GetVolParmsInfoBuffer` structure is honored. File system developers should fill in this field according to the following formula: `-1 * (transfer rate in bytes per second)`. Since Finder checks the `vmVolumeGrade` field before initiating each copy operation, file systems should feel free to produce different values for this field as often as they like. A file system might choose to change this field based on such metrics as network reliability or the number of users logged in to a server.

Enhancements to existing features:

- File copying operations have been substantially accelerated in many cases.
- Window opening (both drawing and redrawing) has been substantially accelerated.

Bug fixes:

Dozens of bugs have been fixed, including many in AppleScript support. Some internal subsystems have been redesigned and re-implemented for substantially better stability.

Finder/AppleScript bug fixes:

- The `typeIconFamily` data type is once again supported.
- Creating folders and renaming objects will now record properly.
- A copy operation with "replacing conflicts" no longer fails.
- The support for "entire contents" has been rewritten completely and should be more reliable in general.
- Setting a view which has not been enabled in the View Options dialog no longer causes unpredictable behavior.
- Support for the kAEFinderSuite/kAESync ('fndr'/'fupd') event now conforms more closely to that of Finder 7. Because of fundamental differences between Finder 7 and 8, there is no way to make the behavior absolutely identical.
- File type comparisons are now case-sensitive.
- Duplicate and move operations "with routing suppressed" now works correctly.
- Several memory leaks were eliminated, some of which existed prior to Finder 8.0. Several involved "whose" clause resolution and disposing of the descriptors used to represent the search specification. Another leak had to do with token handles in token descriptors not getting disposed when the token did.
- Attempts to resolve a broken alias will now reliably produce an error -5018 (`afpObjectNotFound`).
- Support for "whose" clauses now works properly with respect to processes.

Other specific bug fixes:

- Finder 8.0 disables the Put Away item in the Special menu for all RAM disks which (truthfully, per DTS sample code) identify themselves as such. It was thought this would provide better user experience for users of the Mac OS RAM disk, but it had unintended effects on third-party RAM disks. This menu item is no longer disabled under these circumstances.
- For any drag flavor whose data was larger than 1024 bytes, Finder 8.0 was reliably storing only the very first chunk of flavor data in the `DragReference`.

- Finder 8.0 created incorrect clipping files when the drag contained flavors with the `flavorNotSaved` flag. The item count stored in the clipping was too high. Finder 8.1 handles `flavorNotSaved` correctly and tolerates bogus clipping files created by Finder 8.0.
- Finder 8.0 occasionally created custom icon files where they were not needed. Finder 8.1 does not do this.
- Finder 8.1 honors the order in which a group of icons are selected. The order can be important for subsequent commands to that group, like a Print or Open command.
- Get Info on a CFM-68K application now includes the "Memory requirements will increase/decrease" message, which only used to appear for PowerPC-native applications.
- Finder 8.0 did not post the watch cursor every time it became busy. Finder 8.1 posts it in more of these cases.

Gestalt:

As a result of the above changes and changes in Finder 8, Finder's Gestalt selector response value has acquired two new defined bits under Mac OS 8.1:

```
enum
{
    gestaltFinderFloppyRootComments = 8,
    gestaltFinderLargeAndNotSavedFlavorsOK = 9
};
```

`gestaltFinderFloppyRootComments` describes whether desktop database comments for floppy disks contain data the user typed in (notes, etc.) or are used internally by Finder. Under Mac OS 8 and later, this field is used by Finder; if Finder ever relinquishes the use of the field, it will set this bit.

`gestaltFinderLargeAndNotSavedFlavorsOK` describes whether two bugs (described above) in Finder's handling of clipping files have been fixed. Under Mac OS 8.1, this bit is set.

Before testing for either of these bits, you should first test for the presence of Mac OS 8 or later. If Mac OS 8 is present, neither of these bits will be set.

HFS Plus Volume Format

Mac OS 8.1 introduces support for Apple's new volume format, HFS Plus (also known as Mac OS Extended). This format is available for use with any storage device larger than 32MB that support the HFS volume format.

Benefits Provided in Mac OS 8.1

The main benefit in this release of the Mac OS Extended volume format is the use of smaller allocation blocks. The size of allocation blocks used for a particular volume depend on its size.

Volume Size	Default allocation block size
<= 256 MB	512
<= 512 MB	1K
<= 1GB	2K
> 1GB	4K

The default allocation block size of 4K was chosen for volumes larger than 1GB. For a number of reasons:

- The more allocation blocks you have (because of smaller size), the greater the probability of fragmentation.

- The VM page size is 4K. If your allocation blocks are a multiple of 4K, then your Virtual Memory pages won't be fragmented (though the entire backing store file might be).
- A survey of actual fork sizes on actual HFS disks revealed that reducing the allocation block size to 4K gave proportional reduction in the wasted space on a volume. Reducing the allocation block size further did not reduce the wasted space nearly as much. There was a distinct "knee" in the curve between 2K and 8K.
- Doing I/O in larger chunks (4K-16K or more, per I/O) dramatically improves performance compared to small (512 byte) I/O.

Future Benefits

The Mac OS Extended volume format also provides support for long Unicode file names, large files, and extended attributes. However, Mac OS 8.1 does not introduce the new APIs required to access these advanced features. The support for Mac OS Extended volumes was added in such a way that use of these volumes should be transparent to developers who use only the documented File Manager API.

Note:
Even though there are no APIs to directly manipulate Unicode file names, the File Manager uses Unicode to store file names on HFS Plus volumes. These file names are stored in the catalog in an different order than the `RelString` order for HFS volumes. If an application depends on the way files are ordered on disk, it will probably behave differently with the different formats.

Mac OS 8.1 introduces a few changes to the existing File Manager API that developers should be aware of. These changes include routines for getting extended volume information and for controlling the formatting process. These changes are described in a separate HFS Plus Technote which will be published in the near future along with another document, "Mac OS Extended Volume Format", that describes the format of an HFS Plus volume.

Note:
If your application makes use of any undocumented File Manager low memory vectors, it will probably not work with Mac OS 8.1. Most of these private vectors were either changed or removed when support for HFS Plus volumes was added.

File Manager Disk Cache

The File Manager provides services for storage and retrieval of disk based information.

The File Manager disk cache performance on multi-block I/O operations has been improved. Cached multi-block I/O operations should be noticeably faster.

Start Manager Changes

The Start Manager was changed in Mac OS 8.1 to add a mechanism for controlling the loading of extensions at system startup. During the boot process, an extension table is created before any extensions are loaded. This table is then used by the boot code to determine which extensions should be loaded and in which order.

Prior to the introduction of the Extension Table Manager in Mac OS 8.1, extensions were loaded from three folders, Extensions, Control Panels and the System Folder, in the order they were found on the disk. On an HFS volume, items are stored in the catalog in `RelString` order, i.e., in the order the names would be in if sorted by the `RelString` function. This is the same order items are returned in by the File Manager's `GetFInfo` routine.

With the introduction of Mac OS 8.1 a new bootable disk format was introduced, HFS Plus. On an HFS Plus volume, items are stored in the catalog in a different order than a Mac OS Standard volume because of the internal use of Unicode for item names. Since extensions were loaded in the order they were found in the catalog, extensions would load in a different order when booted from an HFS vs. an HFS Plus volume.

Note:
Apple has always maintained that extensions cannot depend on extension loading order; however, many extensions require specific loading order.

To prevent problems for our users, and to ensure that extensions load in the same order regardless of the volume format used, an extension table mechanism was added to the Start Manager. The default behavior of the extension table handler is to sort the extensions into `RelString` order for each folder extensions are loaded from. The result is that extensions are loaded in the same order regardless of which type of volume the system is booting from.

The Extension Table Manager also provides a mechanism that third-party products can use to watch, and optionally control, the extension loading process. A Technote covering this new mechanism is in progress and will be released in the near future.

Sound for PCI Systems

There have been a number of additions and changes to the sound software for PCI-based Macintoshes under Mac OS 8.1 that will affect developers.

Additions

A programmatic method for selecting sound sources via the Sound Input Manager using `OSType` selectors has been added. Both input and playthrough sources are selectable this way.

- The input source via the Sound Input Manager
- A playthrough source.

Four new selectors have been added to the built-in sound input driver:

```
enum {
    siMonitorAvailable      = 'mnav',
    siMonitorSource         = 'mons',
    siOSTypeInputSource     = 'inpt',
    siOSTypeInputAvailable  = 'inav'
};
```

These selectors allow the use of several new constants to set a particular input source or monitor source programmatically. For instance, to have the user record the CD, an application could use code like this:

```
inline OSErr SetInputSource (long soundRefNum, OSType inputSource) {
        return SPBSetDeviceInfo (soundRefNum, siOSTypeInputSource, &inputSource);
}
```

where `inputSource` was set to `kCDSource`.

The `OSType` sources were required to guarantee input source selection without user intervention, since the old sound input numbers were not consistent across machines.

The currently defined input source selectors are:

```
enum {
    kNoSource         = 'none',     /*no source selection*/
    kCDSource         = 'cd  ',     /*internal CD player input*/
    kExtMicSource     = 'emic',     /*external mic input*/
    kRCAInSource      = 'irca',     /*RCA jack input*/
    kTVFMTunerSource  = 'tvfm',
    kDAVInSource      = 'idav',     /*DAV analog input*/
    kIntMicSource     = 'imic',     /*internal mic input*/
    kMediaBaySource   = 'mbay',     /*media bay input*/
    kModemSource      = 'modm',     /*modem input*/
    kZoomVideoSource  = 'zvpc'      /*zoom video input*/
};
```

Changes

- Reduced interrupt buffer sizes, which allows for decreased latency when playing and recording. The new buffer sizes, with VM off, are 512 samples per interrupt, which decreases the latency of sound at 44.1 kHz to approximately 11.6 ms. With VM on, the buffer sizes are 4 times larger.
- The sound buffers are again a fixed size which do not change with sample rate.
- The built-in sound input driver no longer supports the `siOptionsDialog` selector. This support was removed for a number of reasons.
    - It is felt that drivers should not present a user interface.
    - The existing interface was rather clunky and inconsistent between Mac OS computers.
    - QuickTime's Sequence Grabber offers much of the same support, and looks and works better.

End User Experience

There have also been some minor changes, which make the sound experience more enjoyable and predictable for the user. They are:

- User interface changes to support monitor sources instead of "input" sources for the user to play back analog audio devices such as CDs. There is a new Control Strip module and Monitors & Sound has changed to provide the user a choice of "monitor source."
- The internal speaker and headphone port(s) act as a single output port once again. Thus, if a headphone

is inserted, the internal speaker will always be inactive. This simplifies the UI for output controls to a single set of controls (e.g. for volume, mute, etc.).

MathLib v3

MathLib is a collection of numerical functions designed to facilitate a wide range of numerical programming in preparation for C9X. All of its routines are IEEE-754 aware and treat exceptions, NaNs, positive and negative zero and infinity consistent with the floating- point standard. The interface to MathLib is defined in the fp.h and fenv.h interface files.

MathLib v3 is a major rewrite of MathLib that boosts performance and increases accuracy.

Performance

MathLib v3 boosts the performance of many of its functions by an average of 30 % when measured with a uniform distribution over a subset of the domain of each function.

Accuracy

The accuracy of routines in MathLib v3 have been improved, most notably in the trigonometric functions (e.g, sin and cos). While most numerical functions have been improved, the double data type trigonometric functions are now more in line with their long double counterpart due to a change in their argument reduction methodology.

Argument reduction in MathLib v2 was 53-bit Pi based. To improve coherency and uniformity between double and long double, the trigonometric functions in MathLib v3 reduce their arguments with a 107-bit Pi. This may introduce a difference in the computed results of circular functions in MathLib v2 vs MathLib v3.

The reason for this difference is that MathLib v2 and MathLib v3 carry out a machine Pi argument reduction as opposed to an infinite Pi reduction. Any change in the precision of Pi will affect the final result. Arguments passed to MathLib v3 are no longer able to match the argument reduction seed Pi, (which now has 107-bits of precision) hence, exact edge case results are no longer produced. For example, cos(Pi/2) is no longer exactly 0, but about 6E-17. This result is still well below the roundoff threshold of the IEEE double data type.

All circular inequalities and equalities are maintained under MathLib v3, but their precision has improved. For example, sin(x)^2+cos(x)^2 = 1 more often in an exact sense.

To get cos(±Pi/2) = 0.0, instead of 6E-17, developers have two choices:

- Continue using MathLib v2
- special case cos(±nPi/2) using the IEEE remainder function before calling MathLib.

Using MathLib v2 will forfeit the performance and added precision of MathLib v3.

ADB Manager

A change was made to the ADB Manager to fix a problem on Power Macintosh 4400 and 4400-based Mac OS clones where PS/2 input devices were reported as being available, when in fact none were connected. The ADB Manager now properly checks for PS/2 input devices and removes entries for nonexistent devices from its device table. This problem affected Game Sprockets and any applications which used it.

Appearance Manager

Mac OS 8.1 includes Appearance Manager 1.0.1. This version of Appearance did not ship as a separate SDK. You should use the Appearance Manager 1.0.2 SDK. While the SDK contains a newer version of the Appearance Manager, there were no API changes.

PC Exchange

PC Exchange permits Macintosh users to mount MS-DOS and Windows disks on their desktops.

Mac OS 8.1 installs Version 2.2 of PC Exchange, which adds:

- Support for Win95 long filenames (LFN)
- Support for the FAT32 disk format (in addition to FAT12 and FAT16)

VFAT Long Filename Support

PC Exchange 2.2 supports Win95 long filenames and can read Unicode filenames of up to 31 characters. Limitations with the Finder and file system require truncation of filenames greater than 31 characters. If the truncated names are not subsequently edited by the user, they will remain >31 character names on disk. PC Exchange 2.2 can also create Win95 long filenames of up to 31 characters in length under Mac OS 8.1.

Language Kit Users

Users of Apple's Language Kits should be aware that PC Exchange allows file names with characters which are technically illegal under Windows 95, but are used to represent non-Roman characters by the Language Kits. Such users may encounter "illegal character" errors when running Windows 95 disk utilities such as ScanDisk. Users are advised to select "ignore" for such errors to avoid unwanted changes to their file names.

Different PC Filenames

Previous versions of PC Exchange use a different method to construct PC filenames. PC Exchange 2.2 uses the same method as Microsoft. For example, the file "helloworld.doc" would have a PC filename of "!HELLOWO.DOC" under previous versions of PC Exchange. Under PC Exchange 2.2, the file "helloworld.doc" would have a long filename of "helloworld.doc" and a PC filename of "HELLOW~1.DOC". The PC names of files which have been created under previous versions of PC Exchange will not change unless the user modifies the filename.

FAT32 Support

The FAT32 disk format introduced with Windows 95 OSR 2 is supported by PC Exchange 2.2.

Formatting PC Disks

Users can format floppies for the Mac OS, PC, or ProDOS. Users may format PC volumes on their Macintosh, but they cannot change the format of a hard disk or removable medium from Mac OS to PC or vice versa. Allowing this might result in the loss of user data without warning.

Bug fixes

There have been a number of bugs fixes, including proper string truncation on two-byte systems and some low-level bugs.

Apple Location Manager 2.0.1

Apple Location Manager is a toolbox extension that allows users to save and restore sets of configurations

("locations") under a single name. For example, a user can define a group of printers, network settings, and extension sets for multiple locations.

Apple Location Manager 2.0.1 includes the following changes:

- No longer restricted to PowerBooks
- New UI
- Additional API calls
- New notifications
- New module types
- CFM-68K support
- Reboot level escalation
- Bug fixes
- Name
- SDK

Additional API Calls

If `gestaltALMHasSFLocation` is returned from a gestalt call with `gestaltALMAttr`, then the following calls are supported:

```
extern pascal OSErr
ALMPutLocation (ConstStr255Param            prompt,
                ALMLocationName             name,
                SInt16                      numTypes,
                ConstALMModuleTypeListPtr   typeList,
                ModalFilterYDUPP            filter,
                void*                       yourDataPtr)
```

This function allows the developer to create a location using a standardized interface. Typically, the developer will pass `kALMAddAllOff` or `kALMAddAllOnSimple` for `numTypes`, and NULL for `typeList`, but it is possible to pass an array of module signatures. The filter and `yourDataPtr` parameters behave as in Standard File.

```
extern pascal OSErr
ALMMergeLocation (ConstStr255Param            prompt,
                  ALMLocationName             name,
                  SInt16                      numTypes,
                  ConstALMModuleTypeListPtr   typeList,
                  ModalFilterYDUPP            filter,
                  void*                       yourDataPtr);
```

This function allow the developer to merge settings into an existing location using a standardized interface; the parameters are as with `ALMPutLocation`, though typically a module signature array is used in this case.

```
extern pascal OSErr
ALMGetLocation (ConstStr255Param    prompt,
                ALMLocationName     name,
                ModalFilterYDUPP    filter,
                void*               yourDataPtr);
```

To ask the user to choose a location by name, call this.

This function allows a user to choose a location by name.

New Notifications

Previously, an interested program could receive notification that the current location had changed. In ALM 2.0.1, if bit `gestaltALMHasRescanNotifiers` returned from `gestaltALMAttr` is set, the client code will be notified when the locations list is changed (a location gets deleted, renamed, or added). In this way, interested programs will know if location data they retrieved earlier (such as through a call to `ALMGetIndLocation`) might be out of date because the user has edited the locations list.

New Module Types

Under ALM 1.0.x, all modules had file types of 'thng'. Under ALM 2.0.1, 'thng' files are still supported, but the preferred file types are 'almn' and 'almb'. An 'almn' file is a preference-swapping module, such as "Extension Set". An 'almb' file is an action module, such as "Auto-Open Items". The distinction is that the "value" of an action module cannot be directly determined from the system--it requires user intervention (for example, in the case of Auto-Open Items, the user is asked what files to open).

Developers are encouraged to adopt these new file types so that modules can be auto-routed to the Module folder in a later version of the system software.

CFM-68K Support

All calls in ALM 2.0.1 are available under CFM-68K.

Reboot Level Escalation

When a module was called with the `kALMSetCurrentSelect` selector, the value of the `*flags` parameter was `kALMNoChange` under 1.0.x. Under 2.0 this value is the current "escalation" of the setting; that is, the level to which all modules prior to the developer's module have raised the setting. For example, if a location contains an Extension Set that requires a restart, and subsequent modules do not want to act if the system is restarting, the module can use the input value to decide how to interpret the `SetCurrent` call.

Bug Fixes

- Calling `ALMSwitchToLocation` from an INIT did not work prior to ALM 2.0.1.
- Under `ALMConfirmName`, the users filterproc was not called under both dialogs presented prior to 2.0; now it is, and the window refCons can be used to determine which dialog is up. All dialog elements are documented constants.
- It was possible to call `ALMSwitchToLocation` from within a module during a call to `ALMSwitchToLocation`. This tail-biting now generates an error.
- Modules could not return an error on their Open calls; they can now (for example, if they do not wish to display because a necessary piece of hardware is not available on a given machine).

Name

Apple may, for business reasons, change the name of "Apple Location Manager". Developers are cautioned not to rely on the names of items such as the modules folder, and should use `FindFolder`.

SDK

The Apple Location Manager 2.0.1 SDK is available on the Jan 98 Mac OS SDK CDs. The necessary interface files are included in Universal Interfaces 3.0.1.

Virtual Memory

The Virtual Memory Manager provides virtual memory services for the Mac OS.

There have been a number of changes to the Virtual Memory Manager aimed at providing improved performance for applications. Some of these changes are transparent to applications, while others allow applications to actively control Virtual Memory.

System Level Changes

- Apple Virtual Memory's file mapping code (used by the Code Fragment Manager) was rewritten to

provide much better performance. Preparation of code fragments with Virtual Memory on should be noticeably faster.
- Apple Virtual Memory supports the `kdgVMOptions` `DriverGestalt` selector, which allows disk drivers to tell Virtual Memory and the Memory Control panel what Virtual Memory operations are supported by a disk drive.

Virtual Memory Paging Control Functions

The Virtual Memory Paging Control functions allow applications to help the Virtual Memory Manager optimize system performance. This is accomplished by giving the Virtual Memory Manager hints about:

- How memory pages are likely to be used in the near future
- What pages will not be used in the near future
- What pages are not likely to be changed again in the near future
- What pages contain data that will not ever be needed again.

Determining When Virtual Memory Paging Control Functions Are Available

You can determine when the four Virtual Memory Paging Control functions (`MakeMemoryResident`, `MakeMemoryNonResident`, `FlushMemory`, and `ReleaseMemoryData`) are available by calling Gestalt with the `gestaltVMAttr` selector and checking if the `gestaltVMHasPagingControl` bit (bit 4) is set in the response.

```
Boolean VMHasPagingControl(void)
{
    long response;

    if ( (Gestalt( gestaltVMAttr, &response ) == noErr) &&
         ((response & ( 1L << gestaltVMHasPagingControl )) != 0) )
    {
        return ( true );
    }
    else
    {
        return ( false );
    }
}
```

Virtual Memory Page States

Mac OS virtual memory pages can be in several states. A page's state determines what happens when it's not resident in physical memory, and is made resident, and what happens when it's resident in physical memory and is made non-resident. Understanding how the state of a page affects the Mac OS virtual memory system will help you understand the purpose of the Virtual Memory Paging Control functions.

A page can be either resident or non-resident. A page that is currently in physical memory is resident. A page that is currently not in physical memory is non-resident.

Note:
Pages held by `HoldMemory` or locked by `LockMemory`, `LockMemoryContiguous`, and `LockMemoryForOutput` are always resident.

A resident page has two states: clean and dirty. A clean page's data does not need to written from memory to the backing store file before the page is made non-resident. A dirty page's data must be written from memory to the backing store file before the page is made non-resident.

A non-resident page also has two states: valid on disk and invalid on disk. A non-resident page which is valid on disk must be read from the backing store file into memory when it is made resident. A non-resident page which is invalid on disk does not need to be read from the backing store file into memory when it is made resident.

Note:
Memory pages are used for file mapping CFM containers stored in file data forks are always clean when resident (because they are read-only) and are always valid on disk when non-resident.

MakeMemoryResident

Use the `MakeMemoryResident` function to make a portion of the address space resident in physical memory. `MakeMemoryResident` can be used to optimize system performance by causing the Virtual Memory Manager to read in pages that will be needed in the future.

```
pascal OSErr MakeMemoryResident (void *address,
                                 unsigned long count);

address   The starting address of the range of memory to be made resident.
count     The size, in bytes, of the range of memory to be made resident.
```

Description

The `MakeMemoryResident` function makes a portion of the address space beginning at `address` and having a size of `count` bytes resident in physical memory.

If the `address` parameter supplied to the `MakeMemoryResident` function is not on a page boundary, then it is rounded down to the nearest page boundary. Similarly, if the specified range does not end on a page boundary, the `count` parameter is rounded up so that the entire range of memory is made resident.

The `MakeMemoryResident` function makes the range of memory resident as efficiently as possible. Non-resident pages that are made resident are always marked clean.

Special Considerations

Even though `MakeMemoryResident` does not move or purge memory, you must not call it at interrupt time.

The specified range must be entirely in allocated space in main memory (the System and Process Manager memory partitions), or entirely in a single file-mapped space.

Assembly-Language Information

The trap macro and routine selector for the `MakeMemoryResident` function are:

Trap macro Selector

```
_MemoryDispatch $000B
```

The registers on entry and exit for this routine are

Registers on entry

```
D0      Selector code
A0      Starting address
A1      Number of bytes to make resident
```

Registers on exit

```
D0      Result code
```

Result Codes

```
noErr                   0  No error
paramErr             -50  The address range specified is invalid
notEnoughMemoryErr   -620  There is insufficient physical memory to make entire range r
```

MakeMemoryNonResident

Use the `MakeMemoryNonResident` function to make a portion of the address space non-resident. `MakeMemoryNonResident` can be used to optimize system performance by giving the Virtual Memory Manager pages which can be used to handle future page faults.

```
pascal OSErr MakeMemoryNonResident (void *address,
                                    unsigned long count);

address   The starting address of the range of memory to be made non-resident.
count     The size, in bytes, of the range of memory to be made non-resident.
```

Description

The `MakeMemoryNonResident` function makes a portion of the address space beginning at `address` and having a size of `count` bytes non-resident.

If the `address` parameter supplied to the `MakeMemoryResident` function is not on a page boundary, it is rounded up to the nearest page boundary. Similarly, if the specified range does not end on a page boundary, the `count` parameter is rounded down so that only pages completely contained in the range of memory are made non-resident (although all pages of the specified range are flushed). This means that less memory than was specified may be made non-resident.

All dirty pages of the specified range are first written to the backing store file and then marked valid on disk.

Any pages of the specified range that are held (with `HoldMemory`) or locked (with `LockMemory` or `LockMemoryForOutput`) will not be made non-resident, but are flushed.

Special Considerations

Even though `MakeMemoryNonResident` does not move or purge memory, you must not call it at interrupt time.

The specified range must be entirely in allocated space in main memory (the System and Process Manager memory partitions), or entirely in a single file mapped space.

Assembly-Language Information

The trap macro and routine selector for the `MakeMemoryNonResident` function are:

Trap macro Selector

```
_MemoryDispatch $000D
```

The registers on entry and exit for this routine are

Registers on entry

```
D0      Selector code
A0      Starting address
A1      Number of bytes to make non-resident
```

Registers on exit

```
D0      Result code
```

Result Codes

```
noErr            0   No error
paramErr       -50   The address range specified is invalid
```

FlushMemory

Use the `FlushMemory` function to make a portion of the address space clean. This can be useful when you want to ensure the backing store file's data matches what is in a resident page. `FlushMemory` can also be used to optimize system performance by letting the Virtual Memory Manager know what pages can be made clean because they are unlikely to change again in the near future (making those pages non-resident will not require I/O in the future).

```
pascal OSErr FlushMemory (void *address,
                          unsigned long count);

address   The starting address of the range of memory to flush.
count     The size, in bytes, of the range of memory to flush.
```

Description

The `FlushMemory` function makes a portion of the address space, beginning at `address` and having a size of `count` bytes, clean. All dirty pages found in the specified range are written to the backing store file. Resident pages are left resident by `FlushMemory`.

If the `address` parameter supplied to the `FlushMemory` function is not on a page boundary, then it is rounded down to the nearest page boundary. Similarly, if the specified range does not end on a page boundary, the `count` parameter is rounded up so that the entire range of memory is flushed.

Special Considerations

Even though `FlushMemory` does not move or purge memory, you must not call it at interrupt time.

The specified range must be entirely in allocated space in main memory (the System and Process Manager memory partitions), or entirely in a single file mapped space.

Assembly-Language Information

The trap macro and routine selector for the `FlushMemory` function are:

Trap macro Selector

```
_MemoryDispatch $000E
```

The registers on entry and exit for this routine are

Registers on entry

```
D0      Selector code
A0      Starting address
A1      Number of bytes to flush
```

Registers on exit

```
D0      Result code
```

Result Codes

```
noErr           0   No error
paramErr      -50   The address range specified is invalid
```

ReleaseMemoryData

Use the `ReleaseMemoryData` function to release the data of a portion of the address space. `ReleaseMemoryData` can be used to optimize system performance by preventing unnecessary writing to and reading from the backing store file.

```
pascal OSErr ReleaseMemoryData (void *address,
                                unsigned long count);

address   The starting address of the range of memory to release.
count     The size, in bytes, of the range of memory to release.
```

Description

The `ReleaseMemoryData` function informs the Virtual Memory Manager that a portion of the address space, beginning at `address` and having a size of `count` bytes, is no longer needed.

If the `address` parameter supplied to the `ReleaseMemoryData` function is not on a page boundary, then it is rounded up to the nearest page boundary. Similarly, if the specified range does not end on a page boundary, the `count` parameter is rounded down so that only pages completely contained in the range of memory are made invalidated. This means that less memory than was specified may be released.

All resident pages in the specified range are marked clean without writing their contents to the backing store file; all non-resident pages in read/write areas of the specified range are marked invalid on disk (Note: read-only file mapped areas are not invalidated on disk). Following a `ReleaseMemoryData` request with a `MakeMemoryNonResident` request makes pages in the specified range immediately available for reuse by the Virtual Memory Manager.

Special Considerations

If the released range is subsequently accessed, the values in memory will be unpredictable.

Even though `ReleaseMemoryData` does not move or purge memory, you should not call it at interrupt time.

The specified range must be entirely in allocated space in main memory (the System and Process Manager memory partitions), or entirely in a single file mapped space.

The Memory Manager calls `ReleaseMemoryData` for `NewPtr`, `NewHandle`, `TempNewHandle`, and `InitZone` requests. There's no need to duplicate the work the Memory Manager has already done.

Assembly-Language Information

The trap macro and routine selector for the `ReleaseMemoryData` function are:

Trap macro Selector

```
_MemoryDispatch $000C
```

The registers on entry and exit for this routine are

Registers on entry

```
D0      Selector code
A0      Starting address
A1      Number of bytes to release
```

Registers on exit

```
D0      Result code
```

Result Codes

```
noErr            0   No error
paramErr       -50   The address range specified is invalid
```

File System Manager

The File System Manager provides a general means by which foreign file systems can be installed, identified, and interfaced to the Operating System. Mac OS 8.1 includes version 2.0 of the File System Manager, which includes a number of changes:

- `gestaltFSMVersion` will return version 2.0.
- All multi-block I/O requests through `UTCacheReadIP`, `UTCacheWriteIP`, `UTVolCacheReadIP`, and `UTVolCacheWriteIP` now use an XIOParam block, so foreign file systems that use 4GB block storage or larger can be accessed.
- Fixed `FSM's _Control` patch that asks foreign file systems for icons (via `ffsGetIconMessage`) so that it only looks for synchronous requests -- `ffsGetIconMessage` can't be made at interrupt time, so asynchronous requests were not safe to handle.
- Fixed a bug in `ffsGetIconMessage` introduced in Mac OS 8 that made it fail.
- Foreign file systems are now called with the `ffsUnloadMessage` at a time that's safe to make other File Manager requests. This lets you call the Code Fragment Manager to unload code fragments. Previously, foreign file systems got the `ffsUnloadMessage` while the File Manager was busy and any direct or indirect File Manager requests would deadlock the system.
- In some rare cases, a foreign file system's `HFSCIProc` would be called with a `MountVol` request outside of the context of a File Manager request. This could cause a system crash if the foreign file system made a cache I/O request while the File Manager was busy with another request. Now all requests to foreign file systems come from within the context of a File Manager request.
- FSM now supports `idSectors` greater than or equal to `kMaximumBlocksIn4GB` on foreign file system volumes.
- FSM would sometimes call a foreign file system's `HFSCIProc` with a `MountVol` or a `VolumeMount` request while running on **another** foreign file system's stack. If that stack wasn't large enough, it could cause a crash. This was fixed.
- `UTVolCacheWriteIP` never worked correctly in Mac OS 8 due to a disk cache bug. That bug has been fixed.
- Fixed a possible memory leak in `fsmDrvQElChangedMessage` code.
- Fixed a rare crashing bug in `fsmGetFSIconMessage` code.

An updated version of the Guide to the File System Manager will be available in the near future.

`DriverGestalt` Additions

The `DriverGestalt` driver status and control calls allow drivers to provide information about the services they provide to clients of the driver.

Two new `DriverGestalt` selectors have been added. These new selectors allow drivers to provide information to the system about the support they provide for VM backing store and about the physical characteristics of the media they support.

The selectors are:

`kdgVMOptions` Returns information about VM support

`kdgMediaInfo` Returns information about supported media

kdgVMOptions

The `kdgVMOptions` `DriverGestalt` selector allows a disk driver to tell the Virtual Memory Manager about its support for Virtual Memory operations. The disk drive for which the driver should return information for is indicated by the value in the `ioVRefNum` field of the `DriverGestaltParam`. This is a per-drive call, not a per-driver call.

The response to this selector is a `DriverGestaltVMOptionsResponse` structure. The `vmOptions` field contain flags that, when combined and tested against the possible masks, indicate a drive's suitability for various combinations of Virtual Memory operations.

The only three valid responses to `kdgVMOptions` at this time are `kAllowVMNoneMask`, `kAllowVMReadOnlyMask`, and `kAllowVMReadWriteMask` (i.e., setting only `kAllowVMWriteBit` is not valid).

- `kAllowVMNoneMask`: Indicates that a drive should never be in the page fault path. Examples of this are drives that have manual eject buttons that are not disabled by software, drives with very slow throughput, or drives that depend on a network connection.
- `kAllowVMReadOnlyMask`: Indicates that a drive should never be written to but is safe for read-only file mapping. Examples of this are WORM drives, where each write eats write-once space on the disks, and CD-ROM drives which are read-only media.
- `kAllowVMReadWriteMask`: Indicates that a drive can be used by Virtual Memory to create its main backing store file. Examples of this are fast read/write drives that don't allow manual eject and don't use a network connection.

Important: All bits not defined here are reserved and should be set to zero until they are defined for a specific purpose.

The `kdcVMOptions` Driver Configure `_Control` call provides the ability to change a driver's response to `kdgVMOptions` Driver Gestalt requests. A driver should return `controlErr` if it doesn't want to provide the ability to change the `kdgVMOptions` response. If a driver supports the `kdcVMOptions` Driver Configure `_Control` call, but is asked to set an option bit that it doesn't support (for example, if a read-only device is asked to set the `kAllowVMWriteBit`), it should return `paramErr`.

```
struct DriverGestaltVMOptionsResponse {
    UInt32      vmOptions;
};
typedef struct DriverGestaltVMOptionsResponse DriverGestaltVMOptionsResponse;

/* Bits and masks for DriverGestaltVMOptionsResponse.vmOptions field */
enum {
    kAllowVMReadBit         = 0,        /* Allow VM to use this drive for read access */
    kAllowVMWriteBit        = 1,        /* Allow VM to use this drive for write access */
    kAllowVMNoneMask        = 0,
    kAllowVMReadOnlyMask    = 1 << kAllowVMReadBit,
    kAllowVMReadWriteMask   = (1 << kAllowVMReadBit) + (1 << kAllowVMWriteBit)
};
```

kdgMediaInfo

The `kdgMediaInfo` `DriverGestalt` selector allows a disk driver to tell the caller about physical characteristics of the device it supports. The disk drive for which the driver should return information for is indicated by the value in the `ioVRefNum` field of the `DriverGestaltParam`. This is a per-drive call, not a per-driver call.

The response to this selector is a `DriverGestaltMediaInfoResponse` structure. The fields in this structure contain the physical block size, the number of blocks that are of that size, and the media type for a given device.

On drives that support ejectable media, the response can change depending on what media is currently in the drive.

Note:
The only current defined media types are CD-ROM and DVD-ROM. This is because an application that makes this call currently only cares if it's one of these two types. Until this changes, all other media types should return `kMediaTypeUnknown`.

The File Manager calls this selector and uses the values returned to help determine the value it will use for the allocation block size when formatting a volume as Mac OS Extended.

```
struct DriverGestaltMediaInfoResponse {
    UInt32          numberBlocks;   /* number of blocks */
    UInt32          blockSize;      /* physical size of blocks */
    SInt16          mediaType;      /* media type identifier */
};
typedef struct DriverGestaltMediaInfoResponse DriverGestaltMediaInfoResponse;

/* DriverGestaltMediaInfoResponse.mediaType constants */
enum {
    kMediaTypeUnknown    = 128,   /* media type is unknown */
    kMediaTypeCDROM      = 129,   /* media type is a CD-ROM */
    kMediaTypeDVDROM     = 130,   /* media type is a DVD-ROM */
    kMediaTypeNoMedia    = -1     /* no media is present */
};
```

Mac OS Runtime for Java 2.0

MRJ 2.0 supports Sun's Java version 1.1.3 specification, which has added or improved support in the following areas:

- Internationalization
- Security and Signed Applets
- AWT Enhancements
- JavaBeans™
- JAR File Format
- Networking Enhancements
- I/O Enhancements
- Math Package
- Remote Method Invocation
- Object Serialization
- Reflection
- JDBC™ - Connecting Java and Databases
- Inner Classes
- Java Native Interface
- Performance Enhancements
- The demonstration applets have been updated and/or replaced with new examples

For further information, see the Mac OS Runtime for Java site.

Open Transport 1.3

Mac OS 8.1 installs Open Transport v1.3, which contains a number of new features and bug fixes.

An updated **OT 1.3 SDK** and **OT 1.3 Release Notes** are available.

Bug fixes:

General

- Plugged a 256-byte memory leak which could occur each time ARA 3.0 made a PPP connection. This leak could possibly have also affected other protocols.
- Added and corrected `'CCI™'` resources to the Open Transport components to support Extension Manager 4.0 and higher.
- Removed many unnecessary debugger breaks from the debug version of OpenTransport. This includes the debugger breaks you get on boot on every non-PCI Power Macintosh and on all PCI PowerBooks.
- Fixed a bug in the OT patch on `CRMInstall` that caused serial ports to sometimes fail to register properly. This was most notably a problem with the Global Village Platinum Pro PC Card. The symptom was getting a "Serial port in use" error when try to connect with the PPP control panel.
- Fixed a bug in option management handling that prevented the setting of DDP options when using an ADSP or ATP endpoints.
- Fixed a crash that occurred when a 68K client running in emulation on a Power PC system called `OTUseSyncIdleEvents`. This same crash also occurred with CFM-68K clients.
- Fixed a bug that caused `OTOpenEndpoint` and `OTAsyncOpenEndpoint` to sometimes return garbage for the `EndpointRef`, if there was an error. These functions now return a NULL result if an error occurs.
- Fixed a memory leak in `InitOpenTransport` if it was called from an ASLM shared library.
- Fixed problems with the "tilisten" module that caused listening endpoints to go deaf. The symptom was that a listening endpoint would return the `kOTStateChangeErr` on various Open Transport calls to process incoming events.
- Fixed a memory leak in the "tilisten" module.
- Fixed synchronous `OTConnect` to call a notifier with `kOTSyncIdleEvent` while waiting to complete the call.
- Fixed a bug in `OTSndDisconnect` that could cause corruption of the cookie parameter that gets passed with a `T_MEMORYRELEASED` event if a send (i.e. `OTSnd`), with AckSends enabled, interrupts `OTSndDisconnect` at just the right time.

Open Transport Debugger Preferences (for debug version only)

- The OT Debugger Preferences file is now installed into the MacsBug Preferences folder.
- Fixed up `OTErrors` text.
- Added `modname` and `modnext` macros which can be used to display the name of the module calling `putnext`, and the module name that will next receive the message block. To be used, register R3 must contain the pointer to a module's queue element such as when you break on entry to the `putnext` call.
- Added `qname` and `dmsg` macros.
- Fixed value for `M_CTL` in the `MSGTypes` text template.
- Fixed `module_info` so it is correct for Power PC.
- Added `module_info68k` for displaying `module_info` on 68K machines.
- Made `msgb::fNext` of type `msgb` so you can quickly see all message blocks in the message chain.

AppleTalk

- Fixed a bug that sometimes prevented ARA 3.0 from properly setting the default zone (the zone that

is automatically selected in the Chooser).
- Fixed a crash that could occur when using LocalTalk in a low memory situation.
- Fixed a bug in the Chooser that caused it to add duplicate `'STR ' -4090` resources to the selected Chooser device file (i.e. LaserWriter 8) each time you selected the device. This only happened if you had no zones. This bug could eventually cause the device file to become corrupt.
- Integrated newer `'ltlk'` resources from IRTalk to fix problems with using IRTalk on certain desktop machines.
- Fixed PAP so that SendData requests are no longer sent from a listening endpoint after the PAP connection has been closed.
- Fixed problems with the PAP server handling multiple sessions. Although more than one session could be opened to the server at a time, the server would only service one of the sessions. It will now service all of the open sessions in parallel.
- Fixed PAP so it properly handles dealing with more then 64K transactions (files > 240M). It had a rollover problem when the sequence number reached 64K.
- Fixed D3 register corruption on Power Macintoshes that occurred when calling any OT routine that resulted in data being sent out over a LocalTalk connection. This bug is most apparent if you have 68K code that calls Power PC code that then calls an OT send routine (i.e. `OTSnd`). Upon return to the 68K code, the D3 register would have been corrupted.
- Fixed the `ATALK_IOC_FULLSELFSEND` macro.
- Fixed an ARA 3.0 problem on ARAP connections, where the server was shut down by manual restart and the client reported that the connection would disconnect in one minute, when in fact the connection had already been broken.

TCP/IP

- Improved HTTP performance by not delaying before sending out the second packet from the server.

Links

- Fixed a bug in the "tpi8022x" module when processing an option management call to place an Ethernet endpoint into promiscuous mode, such that the code entered an infinite loop.
- Fixed a bug in the "tpi8022x" module which limited a rawmode Ethernet endpoint to sending a maximum data packet of 1486 bytes instead of the full 1500 bytes.

API Changes

- Bumped the `kInetInterfaceInfoVersion` up to 3 to indicate support for single link multi-homing. See section on Single Link Multi-homing.
- Changed `OTCreatePCMCIAPortRef` to `OTCreatePCCardPortRef` and fixed the bus reference it uses to `kOTPCCardBus`.
- Added `kOTNetbufIsRawMode` constant for specifying `TNetbufs` that contain raw mode data.
- Added `mi_open_detached`. It was already in "`OpenTptModule.h`", but never properly exported by the OT libraries. It is similar to `mi_open_comm`, but is useful when you need to make the call before the module is actually open, such as when you maintain separate `q_ptr` data for upper and lower queues, and you want to allocate the `q_ptr` for the lower queue before the `I_LINK` is received.
- Support the new 8.0.1 `DriverLoaderLib` which allows multiple `'ndrv'`s in a CFM code fragment.
- Really added support for CFM port scanners and configurators. On Power PC, you no longer need to write your port scanners and resident configurators using ASLM. This support was actually in OT 1.1.1, but only worked if you explicitly registered your shared library with CFM (which required use of a private CFM `SPI`). OT will now scan the Extensions folder on boot, looking for all `'shlb'` and `'libr'` files with an extended `'cfrg'` resources. The extended `'cfrg'` is then used to locate port scanner and resident configurator shared libraries within the file. Your extended `'cfrg'` resource for a resident configurator should look as follows:

```
#define UseExtendedCFRGTemplate 1
#include "OpenTransport.r"
#include "CodeFragmentTypes.r"
resource 'cfrg' (0)
{
  {
```

```
        extendedEntry {
        kPowerPC,
        kFullLib,
        kNoVersionNum,      /* Current Version Number */
        kNoVersionNum,      /* Old Def Version Number */
        kDefaultStackSize,
        kNoAppSubFolder,
        kIsLib,
        kOnDiskFlat,
        kZeroOffset,
        kWholeFork,
        "XYZProtocolRS_ConfiguratorLib",
        /* start of extended info */
        kOTCFMClass,
        kOTConfiguratorCFMTag,
        "",
        "",
        "XYZProtocol"       /* external name: may be seen by user */
        }
    }
};
```

Use `kOTPortScannerCFMTag` for port scanners instead of `kOTPortConfiguratorCFMTag`.

CFM-68K Support

- CFM-68K is now officially supported. See the "Open Transport CFM-68K Developer Note" (part of the OT SDK) for details. The OT Installer has been modified to automatically install CFM-68K support.

Single Link Multi-homing

Open Transport 1.3 introduces **single link multi-homing**, a mechanism by which Open Transport can support multiple IP addresses on the same hardware interface. Synonyms for this feature include IP Aliasing, Secondary IP address support, IP Masquerading, "Multihoming", and IP Multinode support. This is useful for sites like Internet Service Providers (ISPs), that want to give each of their clients a distinct IP address, without requiring separate computers for each address. Web server software packages or server plug-ins that utilize this feature can offer virtual domain support that supports all web browsers.

This functionality is transparent to Open Transport clients who are not specifically interested in single link multi-homing. The following information will help TCP/IP server developers implement single link multi-homing support into their products.

Important: As described below, in a multi-homed environment, if you bind to a specific IP address, you will only received connections targeted at that IP address. While this is useful if you want to support different operations on different IP addresses, most programs do not want this. If your program does not take advantage of single link multi-homing, it's important that it bind listening endpoints to the address `kOTAnyInetAddress`, not to a specific IP address (such as the address returned by `OTInetGetInterfaceInfo`). The use of `kOTAnyInetAddress` has always been the recommended way of binding listening endpoints, and single link multi-homing makes it important that you follow this recommendation.

Single Link Multi-homing System Setup

Single link multi-homing support is only available with Open Transport 1.3 or higher. Your product will need to check that this version of Open Transport is present. See the section Checking the Open Transport Version for details on how to check for the presence of Open Transport 1.3 or higher.

You configure a system to use multiple IP addresses as follows:

- The TCP/IP Control Panel must be set for manual addressing.
- You create a text file with the required name, "IP Secondary Addresses", and put it into the Preferences folder in the System Folder.

Each line of the IP Secondary Addresses file contains a secondary IP address to be used by the system, and an optional subnet mask and router address for the secondary IP address. If there is no subnet mask entry, then a default subnet mask for the IP address class will be used. If there is no router address entry, then the default router associated with the primary address will be used.

Each secondary address entry must be prefixed by "ip=". Each subnet mask entry must be prefixed by "sm=". Each router address entry must be prefixed by "rt=". An example of the contents of the IP Secondary Addresses file follows.

```
; 'ip=' for ip address, 'sm=' subnet mask, 'rt=' router address
;  Note: nspace in 'ip=192.168.22.200'
;
; IP address             Subnet Mask          router addresses
;-----------             -----------          ----------------
ip=192.168.22.200        sm=255.255.255.0     rt=192.168.20.1
ip=192.168.22.201                             rt=192.168.20.1
ip=192.168.22.202
```

The order of the entries is important. The "rt=" entry must follow the "sm=" entry if used.

When Open Transport 1.3 activates TCP/IP, the primary address will be obtained from the TCP/IP Control Panel setting. Open Transport then looks for the IP Secondary Addresses file in the Preferences folder, to determine if additional addresses should also be configured. If there are duplicate IP address entries in the IP Secondary Addresses file, the duplicated addresses will be ignored. When Open Transport binds a TCP/IP connection, if there is an address conflict of the primary or any secondary addresses with another host, Open Transport will present an error message using a dialog box and will unload Open Transport TCP/IP from memory. The error dialog will display the conflicting IP address, the hardware address of the conflicting machine and note that your TCP/IP network interface has been shut down.

Checking the Open Transport Version

To check that Open Transport version 1.3 is present, use the Gestalt function with the `gestaltOpenTptVersions 'otvr'` selector. Check that the result is greater or equal to `kOTIPSingleLinkMultihomingVersion`.

```
enum
{
        kOTIPSingleLinkMultihomingVersion = 0x01300000  // OT 1.3
};
```

InetInterfaceInfo Structure Change

The `OTInetGetInterfaceInfo` returns information about the local host. Under Open Transport 1.3, the `InetInterfaceInfo` structure has changed to allow the function to return additional information on supported secondary addresses. The new version of the structure is as follows

```
struct InetInterfaceInfo
{
    InetHost          fAddress;
    InetHost          fNetmask;
    InetHost          fBroadcastAddr;
    InetHost          fDefaultGatewayAddr;
    InetHost          fDNSAddr;
    UInt16            fVersion;
    UInt16            fHWAddrLen;
    UInt8*            fHWAddr;
```

```
    UInt32            fIfMTU;
    UInt8*            fReservedPtrs[2];
    InetDomainName    fDomainName;
    UInt32            fIPSecondaryCount;  // returns number of IP secondary addresses
    UInt8             fReserved[252];
};
```

You can pass this new structure to the `OTInetGetInterfaceInfo`. On return, the `fIPSecondaryCount` field returns the number of secondary IP addresses configured for the system. If there are secondary addresses, use the `OTInetGetSecondaryAddresses` function to obtain the additional addresses.

To distinguish this structure from earlier variants, the `OTInetGetInterfaceInfo` sets the `fVersion` field to 3.

`OTInetGetSecondaryAddresses`

FUNCTION

```
OTInetGetSecondaryAddresses return active secondary IP addresses
```

C INTERFACE

```
OSStatus OTInetGetSecondaryAddresses(InetHost* addr, UInt32* count, SInt32 index);
```

C++ INTERFACE

None. C++ clients use the C interface to this function.

DESCRIPTION

Parameters	Before Call	After Call
addr	x	(x)
count	(x)	(x)
index	x	/

`OTInetGetSecondaryAddress` is used to copy the supported secondary addresses associated with an IP interface. `OTInetGetSecondaryAddress` uses the index parameter to specify which IP interface to obtain secondary addresses for. For the primary IP interface, set index to `kDefaultInetInterface` which is -1. `OTInetGetSecondaryAddress` uses the count parameter to know how many secondary addresses to return in the buffer pointed to by `addr`. The `addr` buffer must be of size `count * sizeof(InetAddr)` to hold all of the desired addresses. Use the `fIPSecondaryCount` field of the `InetInterfaceInfstructure` returned by calling `OTInetGetInterfaceInfo` to determine the required size of the buffer. `OTInetGetSecondaryAddress` also modifies count to indicate the number of secondary addresses actually returned if less than the specified number of secondary addresses are returned.

Text Encoding Converter Manager 1.3

The Text Encoding Conversion Manager provides two facilities--the Text Encoding Converter and the Unicode Converter--that your application can use to handle text encoding conversion on the Mac OS.

For further information, see *Inside Macintosh:* *Programming With the Text Encoding Conversion Manager* .

Version 1.3 of the Text Encoding Converter Manager (TEC) is included with Mac OS 8.1. The HFS Plus volume

format introduced with Mac OS 8.1 stores filenames using the canonical decomposition form of Unicode 2.0; several of the TEC 1.3 changes are to support HFS Plus.

Interface File Changes

- Moved contents of old `Unicode.h` into new file `UnicodeConverter.h` to avoid confusion as other Unicode-related functionality is added over the next few months. `Unicode.h` includes `UnicodeConverter.h`, but is otherwise currently empty.
- Added constant `kUnicodeCanonicalDecompVariant` (`TextCommon.h`) to specify a variant of Unicode using canonical decomposition (maximal decomposition with characters in canonical order). This constant has the same value as the constant `kUnicodeMaxDecomposedVariant` (unsupported in earlier versions of TEC). (The only other Unicode variant currently supported allows all defined Unicode characters, and is specified by the constant `kUnicodeNoSubset`).
- Added constant `kUnicodeUseHFSPlusMapping` (`UnicodeConverter.h`), which can be used for the `mappingVersion` field of a `UnicodeMapping` structure to specify the mapping version used by HFS Plus. (The only other constant specifying a value for this field is `kUnicodeUseLatestMapping`).
- Defined new feature/fix bits for the `tecUnicodeConverterFeatures` field of the `TECInfo` structure returned by `TECGetInfo`, to indicate new bug fixes/enhancements in TEC 1.3. These bits are:

  `kTECTextRunBitClearFixBit` `ConvertFromUnicodeToTextRun` & `ConvertFromUnicodeToScriptCodeRun` now function correctly if the `kUnicodeTextRunBit` is clear (previously their determination of best target encoding was incorrect).

  `kTECTextToUnicodeScanFixBit` `ConvertFromTextToUnicode` mappings can now depend on context and saved state. There are several related changes:

  - Malformed input produces `kTextMalformedInputErr`.
  - `ConvertFromTextToUnicode` accepts the control flags `kUnicodeLooseMappingsMask`, `kUnicodeKeepInfoMask`, `kUnicodeStringUnterminatedMask`.
  - No redundant direction overrides when converting Mac OS Arabic & Hebrew to Unicode
  - Improved mapping of 0x30-0x39 digits in Mac OS Arabic when loose mappings are used
  - Better context-dependent mapping of certain characters in Mac OS Indic encodings.

- Renamed the static library initialization and termination functions from `InitializeUnicode` and `TerminateUnicode` to `InitializeUnicodeConverter` and `TerminateUnicodeConverter`, for the same reasons as in (a) above. This should not be a problem, since we have not previously released a version of the static library. However, the old names are still exported by the static library, just in case.

Implementation Bug Fixes

- Fixed a crashing bug in `TECGetAvailableSniffers` which occurred in low-memory situations.
- `TECGetTextEncodingFromInternetName` was sensitive to the case of the Internet name passed in, even though Internet names are supposed to be case-insensitive.
- `TECCreateOneToManyConverter` and `TECCreateOneToManyConverterFromPath` should return `paramError` if the `numOutputEncodings` is 0.
- Actually allowed the `kUnicodeStringUnterminatedBit` control flag to be used with `ConvertFromUnicodeToText` (...ToTextRun, ...ToScriptCodeRun). This control was previously documented for use with these APIs (and the supporting code was implemented). However, if it was used, the functions returned `paramErr`, due to an error in parameter validity checking.
- Fixed problems with `ConvertFromUnicodeToTextRun` (...ToScriptCodeRun) when the `kUnicodeTextRun` control flag is clear; in this case these functions were making bad guesses about the best target encoding.
- Fixed problems with handle locking and unlocking that showed up when multiple threads were calling the Unicode Converter.

- `ConvertFromUnicodeToTextRun` (...ToScriptCodeRun) could enter infinite loop when running out of buffer space after executing a fallback handler.
- Fixed several problems in `ConvertFromUnicodeToText` (...ToTextRun, ...ToScriptCodeRun) for Unicode in UTF-8 format: Errors in direction resolution (and a crash when handling bidirectional text), and errors recovering from scanning ahead too far for text element boundaries.
- In `ConvertFromUnicodeToText` (...ToTextRun, ...ToScriptCodeRun), the `kUnicodeVerticalFormBit` was ignored when converting to any Japanese variant except `kJapaneseStandardVariant` and `kJapanesePostScriptScreenVariant`.
- Fixed `CreateUnicodeToTextRunInfo` (and the ...ByEncoding and ...ByScriptCode forms) so that if 0 is passed for number of mappings/encodings/scripts or if NULL is passed for the array, it only creates entries for the script variants that are installed, instead of for all of the possible variants for each installed script.

Implementation Enhancements

- Allowed the following control flags to be used with `ConvertFromTextToUnicode` (previously, these were only intended for use with `ConvertFromUnicodeToText`, ...ToTextRun, ...ToScriptCodeRun): `kUnicodeLooseMappingsMask`, `kUnicodeKeepInfoMask`, `kUnicodeStringUnterminatedMask`.
- Improved the `ConvertFromTextToUnicode` scanner to emit context-dependent information that can affect the mappings, and provided for saving scanner state in the `TextToUnicodeInfo` structure if `kUnicodeKeepInfoMask` is set. Enhanced the mapping table formats to permit mappings that depend on context information from the scanner and on tolerance information (i.e. whether loose mappings are requested).
- When `ConvertFromTextToUnicode` encounters an invalid sequence of bytes in a particular encoding--such as 0x8120 in Shift-JIS--it now reports `kTextMalformedInputErr` (previously, it attempted to look up the invalid combination and returned `kTECUnmappableElementErr`).
- The 68K static library version of the Unicode Converter & Text Common functionality is available with this release. In order to use it, the TEC 1.3 extension and the TEC 1.3 Text Encodings folder and files must be present.

Mapping Changes

Among other things, the changes listed below ensure 100% round trip fidelity for strict mapping in either direction (non-Unicode to Unicode and back or vice versa) for both Mac OS encodings and other non-Unicode encodings.

- Added tables to support mapping between Mac OS encodings and the `kUnicodeCanonicalDecompVariant` variant of Unicode; the mapping version for these tables is `kUnicodeUseHFSPlusMapping`. These tables are located in the Text Encoding Converter extension itself, rather than in files in the Text Encodings folder (note that some of these tables also support other mappings, such as Shift-JIS, EUC-CN, Big-5, and EUC-KR). Note: This Unicode variant is currently not supported by the high-level Text Encoding Converter.
- Eliminated redundant direction overrides when converting from Mac OS Arabic, Farsi or Hebrew to Unicode.
- If `ConvertFromTextToUnicode` is called with the `kUnicodeLooseMappings` bit set, then the mapping of 0x30-0x39 digits in MacArabic and MacFarsi depends on the context, in a manner similar to how the WorldScript I display of these digits depends on context. If the 0x30-0x90 digits are preceded by Latin letters or other "strong European" characters, they display as "Western" digits in WorldScript I, and they are mapped to Unicode characters 0030-0039. Otherwise, they are displayed with the Arabic digits forms in WorldScript I, and they are mapped to the Unicode characters 0660-0669.
- Fixed the scanners for EUC-CN and Big-5 to use the correct high-byte range.
- In cases where we mapped to a Unicode character that has a one-character canonical decomposition, change the mapping to use the canonical decomposition. This affected the following mappings for Mac OS encodings:

Roman, Croatian, Icelandic, Turkish	0xBD
Greek	0xAF
Symbol	0xE1 and 0xF1

- Changed mapping of several characters in Mac OS Romanian (0xAF, 0xBF, 0xDE, 0xDF) to use COMBINING COMMA BELOW.
- Mapped the remainder of the user-defined range in MacJapanese and Shift-JIS.
- Defined several new corporate characters to be "grouping transcoding hints": These precede a group of 2,3 or 4 standard Unicode characters that are to be treated as a group for transcoding. This way we can map additional characters that are in Apple character sets--but not in Unicode--using mostly standard Unicodes plus one of

these transcoding hints. Using these, we changed the mapping for several characters in Mac OS encodings that previously just mapped to single corporate characters:

Japanese	0x8591, 0x85AB-AD, 0x85BF-C1, 0x865D, 0x869E, 0x86CE, 0x86D3-D6, 0x87FB-FC
Hebrew	0xC0
Farsi TrueType variant	0xA4
Symbol	0xE6-EE, 0xF4, 0xF6-FE

We also changed the mapping for several Mac OS Korean characters that were using two "combining disambiguation tag transcoding hints" to just use one. This affected 0xA14F-50, 0xA16A, 0xA170, 0xA198, 0xA19F, 0xA245-46, 0xA64E, 0xA78A, 0xA78E, 0xA792.

- Other mapping fixes for Mac OS encodings:

Devanagari	Deleted obsolete mappings for byte pairs 0xF0B5, 0xF0B8, 0xF0BF
Gurmukhi	Deleted mappings for byte pairs xB4E9, xB5E9, xBAE9, xBFE9, xC0E9, xC9E9. Changed the mapping for 0x91
Arabic AlBayan variant	0x81 should be unmappable
Mac OS VT100 font encoding	Added mappings for 0xE2, 0xE3, 0xF5, 0xF6
Korean	Added mappings for many additional characters (these are all in the Apple extensions area)

- Changed loose mapping of Unicode LINE SEPARATOR to be RETURN (instead of LINE FEED) for Mac OS encodings.

User Interface Changes

- In Mac OS 8.1, the Text Encoding Converter extension is a required system component; the Extensions Manager will not allow it to be disabled, Finder will issue a warning if a user attempts to remove it, and a boot warning will be issued if it is not present.
- Changed the Japanese name of Shift-JIS (returned by `GetTextEncodingName`) to have "Shift-JIS" in romaji instead of katakana.

Downloadables


Acrobat version of this Note (K)
