

Technote 1123

Start Manager Extension Table Mechanism

Version 1.0

CONTENTS

[Introduction](#)

[Monitoring System Extension
Loading and Execution](#)

[Controlling System Extension
Loading and Execution](#)

[Extension Table Manager Reference](#)

[Summary of the Extension Table
Manager](#)

The Start Manager was revised in Mac OS 8.1 to add a mechanism for monitoring and controlling the loading of system extensions during system startup.

This Technote describes how the Start Manager was changed, and shows how a program can monitor or take control of the system extension loading process.

Introduction

Prior to Mac OS 8.1, system extensions were loaded and executed from three folders in order: the Extensions folder, Control Panels folder, and the System Folder. The system extensions in each folder were loaded and executed in the order they were found on the disk. On an HFS volume (also known as Mac OS Standard) item names are stored using ASCII characters. Items are stored in the catalog file in `RelString` order, i.e., in the order the names would be in if sorted by the `RelString` function. Because of this, the File Manager's `GetFInfo` routine returns files in `RelString` order.

With the introduction of Mac OS 8.1, a new bootable disk format, HFS Plus (also known as Mac OS Extended), was introduced. On HFS Plus volumes item names are stored using Unicode, rather than ASCII characters. Items are stored in the catalog file in a different order than they would be on an HFS volume. Details on the sorting order for HFS Plus volumes can be found in the HFS Plus Volume Format documentation.

Note:

Applications should never rely on the order in which files are stored on a volume. Each volume format is free to sort item in any order, or to not sort items at all.

On systems using the Roman script system, files on HFS Plus volumes are returned by `GetFInfo` in a similar, but not exactly the same, order as `RelString`. However, on systems running non-Roman script systems, `GetFInfo` may return files in a radically different order.

Note:

Apple has always maintained that system extensions cannot depend on a particular load order; however, many system extensions do require specific loading orders.

To prevent problems for our users, and to ensure that system extensions load and execute in the same order regardless of the volume format and script system used, the system extension mechanism was revised to be table driven. The Start Manager builds a single table of system extensions, sorted using `RelString`, from the system extensions found in each folder. Then, each system extension is loaded and executed, in order, from the table. The

result is that system extensions are loaded and executed in the same order regardless of which type of volume the system is booting from and what script system the system is running.

The Start Manager was revised to allow other programs access to the system extension loading mechanism. Programs can either monitor or control the system extension loading and execution process.

Monitor and Control

A program can install an `ExtensionNotificationProc` to monitor the system extension loading process. Each `ExtensionNotificationProc` is called before the first system extension is loaded and executed, both before and after each system extension is loaded and executed, and after the last system extension is loaded and executed. When called, an extension notification routine can perform some action such as drawing an icon, playing a sound, displaying the system extension's name, or saving the system extension's name in a file to help debug system extension crashes or document system extension loading order. An `ExtensionNotificationProc` cannot change the order that system extensions load and execute. Any number of `ExtensionNotificationProcs` can be installed.

A program can install an `ExtensionTableHandlerProc` to control the system extension loading process. The `ExtensionTableHandlerProc` is called before the first system extension is loaded and executed, both before and after each system extension is loaded and executed, and after the last system extension is loaded and executed. Unlike an `ExtensionNotificationProc`, the `ExtensionTableHandlerProc` owns the extension table and has complete control over the order that system extensions load and execute. A program that installs an `ExtensionTableHandlerProc` can also prevent some system extensions from loading and load system extensions from other than the default folders. Only one `ExtensionTableHandlerProc` can be installed.

If no `ExtensionTableHandlerProc` is installed, the Start Manager uses its own default extension table handler. This default extension table handler mimics the behavior of the File Manager under previous versions of Mac OS (i.e., if the contents of a folder used to build the extension table changes, then the extension table is rebuilt and execution is resumed where it would have if the Start Manager were calling `GetFInfo`).

Note:

An `ExtensionNotificationProc` or `ExtensionTableHandlerProc` can be installed at any point in the startup process and will begin receiving messages from that point forward. An `ExtensionTableHandlerProc` can control the loading and execution of system extensions that have not already been installed.

The Extension Table

The extension table is a relocatable block in the system heap containing an `ExtensionTable` structure. An `ExtensionTable` consists of an `ExtensionTableHeader` followed by an array of `ExtensionElements`. The `ExtensionTableHeader` contains the version field which indicates the version of both the `ExtensionTable` and the `ExtensionElement` records, an index into the `ExtensionElements` which indicates which system extension is currently loading and executing, the size of an `ExtensionElement`, and the number of `ExtensionElements` in the table. Each `ExtensionElement` in the `ExtensionTable` contains information used to identify which system extensions will be loaded and executed by the Start Manager.

Each installed `ExtensionNotificationProc` will be passed a copy of the `ExtensionElement` for the system extension which is currently being loaded and executed.

The `ExtensionTableHandlerProc` controlling the system extension loading process will be passed the `ExtensionTable` each time it is called. This handler can control the loading of system extensions by modifying the `ExtensionTable`.

The Boot Process

The Start Manager installs the APIs for installing and removing `ExtensionTableHandlerProc` and `ExtensionNotificationProcs` early in the boot process (after `MacBug` has loaded but before `MacBug` `dcmdSecondaryINIT` time).

Before system extensions are to be loaded and executed, the Start Manager creates an extension table.

Before the first system extension is loaded and executed, the `ExtensionTableHandlerProc` and each `ExtensionNotificationProc` will be called with the `extNotificationBeforeFirst` message indicating that the system extension loading process is about to begin. The `ExtensionTableHandlerProc` is always called with the `extNotificationBeforeFirst` message before any `ExtensionNotificationProc`.

The Start Manager then loads and executes each system extension in order from the extension table. Before each system extension is loaded and executed, the `ExtensionTableHandlerProc` and all `ExtensionNotificationProcs` are called with the `extNotificationBeforeCurrent` message. After each system extension is loaded and executed, all `ExtensionNotificationProcs` and the `ExtensionTableHandlerProc` are called with the `extNotificationAfterCurrent` message. The `ExtensionTableHandlerProc` is always called with the `extNotificationBeforeCurrent` message before any `ExtensionNotificationProcs`, and the `ExtensionTableHandlerProc` is always called with the `extNotificationAfterCurrent` message after any `ExtensionNotificationProcs`.

After all system extensions have been loaded all `ExtensionNotificationProcs` and the `ExtensionTableHandlerProc` are called with the `extNotificationAfterLast` message indicating that the system extension loading process is complete. The `ExtensionTableHandlerProc` is always called with the `extNotificationAfterLast` message after the last `ExtensionNotificationProc`.

In all cases, the relative order in which `ExtensionNotificationProcs` are called is not defined.

Monitoring System Extension Loading

An INIT, or a program executed before system extensions are loaded, can monitor the system extension loading process by calling `InstallExtensionNotificationProc` to install an `ExtensionNotificationProc`. Once installed, this handler will be called:

- Before the first system extension is loaded and executed (if installed before the system extension load and execute process).
- Before each system extension is loaded and executed.
- After each system extension has been loaded and executed.
- After the last system extension has been loaded and executed.

An `ExtensionNotificationProc` receives an `ExtensionElementPtr` which points to a copy of the `ExtensionElement` for the system extension currently being loaded.

Note:

The data in the `ExtensionElement` is read-only. While you can change it, you'll only be changing a copy of the `ExtensionElement`. Any changes will be discarded when the `ExtensionNotificationProc` returns.

There can be any number of `ExtensionNotificationProcs` installed.

Controlling System Extension Loading

An INIT, or a program executed before system extensions are loaded, can take control of the system extension loading process by calling `InstallExtensionTableHandlerProc` to install a `ExtensionTableHandlerProc`. `InstallExtensionTableHandlerProc` returns the default `ExtensionTable` created by the Start Manager.

There can only be one `ExtensionTableHandlerProc` installed.

While it is installed, the `ExtensionTableHandlerProc` is responsible for all changes to the `ExtensionTable`,

except for incrementing `extElementIndex` field between system extensions. After an `ExtensionTableHandlerProc` is installed, the system's default extension table handler no longer manages the `ExtensionTable`.

Once installed, the `ExtensionTableHandlerProc` will be called:

- Before the first system extension is loaded and executed (if installed before the system extension load and execute process).
- Before each system extension is loaded and executed.
- After each system extension has been loaded and executed.
- After the last system extension has been loaded and executed.

Unlike `ExtensionNotificationProcs`, the `ExtensionTableHandlerProc` receives a handle to the `ExtensionTable` used by the Start Manager -- not a copy.

Note:

Once an installed `ExtensionTableHandlerProc` modifies the `ExtensionTable`, or the contents of the folders controlled in the `ExtensionTable` are changed, the handler cannot be removed. Calling `RemoveExtensionTableHandlerProc` after these changes will return a `paramErr` error.

The `extElementIndex` field in the `ExtensionTable` is always incremented to the next `ExtensionElement` immediately after the `ExtensionTableHandlerProc` is called with the `extNotificationAfterCurrent` message.

WARNING:

When controlling the loading of system extensions, the only safe time to change which `ExtensionElement` is at `ExtensionTable.extElements[extElementIndex]` is when your `ExtensionTableHandlerProc` is called with the `extNotificationAfterCurrent` message. You may change the `ExtensionTable` or the `extElementIndex` at other times, but you must ensure that the `ExtensionElement` at `ExtensionTable.extElements[extElementIndex]` stays the same.

Extension Table Manager Reference

This section discusses the techniques you can use to monitor or control the system extension loading and execution process.

Extension Table Version (*Gestalt*)

You should call `Gestalt` with the `gestaltExtensionTableVersion` selector to determine the version of `ExtensionTable` currently installed before installing any handlers for the extension table mechanism.

```
enum {
    gestaltExtensionTableVersion = FOUR_CHAR_CODE('etbl') /* ExtensionTable version */
};
```

The current (Mac OS 8.1) `ExtensionTable` version is 1.0.0.

```
enum {
    kExtensionTableVersion = 0x00000100 /* current ExtensionTable version (1.0.0) */
};
```

If `gestaltExtensionTableVersion` is not defined, it indicates that there is no extension table mechanism present and that the associated extension table routines will be undefined.

When the major version number of `kExtensionTableVersion` and the value returned by `gestaltExtensionTableVersion` are different, it indicates that the extension table mechanism has radically changed and code that doesn't know about the new major version must not attempt to use the extension table

mechanism.

Changes to the minor version number of `kExtensionTableVersion` indicate that the definition of the `ExtensionElement` structure has been extended, but the fields defined for previous minor versions of `kExtensionTableVersion` have not changed.

ExtensionTable Structure

An `ExtensionTable` is an `ExtensionTableHeader` followed by an array of `ExtensionElements`. This structure is built by the extension table mechanism when it scans for system extensions to be loaded and executed at boot time.

```
struct ExtensionTable {
    ExtensionTableHeader    extTableHeader; /* the ExtensionTableHeader */
    ExtensionElement        extElements[1]; /* one element for each extension to load */
};
typedef struct ExtensionTable    ExtensionTable;
typedef ExtensionTable          * ExtensionTablePtr;
typedef ExtensionTablePtr       * ExtensionTableHandle;
```

An `ExtensionTableHandle` containing the current `ExtensionTable` will be passed as a parameter to the installed `ExtensionTableHandlerProc` when it is called.

ExtensionTableHeader Structure

An `ExtensionTable` begins with an `ExtensionTableHeader`.

```
struct ExtensionTableHeader {
    UInt32    extTableHeaderSize;
    UInt32    extTableVersion;
    UInt32    extElementIndex;
    UInt32    extElementSize;
    UInt32    extElementCount;
};
typedef struct ExtensionTableHeader ExtensionTableHeader;
```

<code>extTableHeaderSize</code>	The size of <code>ExtensionTableHeader</code> . Equal to <code>offsetof(ExtensionTable, extElements[0])</code>
<code>extTableVersion</code>	The current <code>ExtensionTable</code> version. The same as the value returned by <code>gestaltExtensionTableVersion</code> Gestalt selector
<code>extElementIndex</code>	The current index into <code>ExtensionElement[]</code> (zero-based). Incremented by the Start Manager after each system extension is loaded and executed
<code>extElementSize</code>	The size of the <code>ExtensionElements</code> in this version of the <code>ExtensionTable</code>
<code>extElementCount</code>	The number of <code>ExtensionElement</code> records in the <code>ExtensionTable</code>

The `ExtensionTableHandlerProc` should check the minor version of the `extTableVersion` field. If it has changed since the `ExtensionTableHandlerProc` was written, it indicates that the value of `extElementSize` will be larger. If the `ExtensionTableHandlerProc` is only moving and deleting elements in the table it could continue to run. However, if it wants to create new elements, it should remove itself because it will not know how to create the new fields in the `ExtensionElements`.

This structure is used internally by the Start Manager, and is passed to the installed `ExtensionTableHandlerProc` each time it is called.

[illegible]

message	input	One of the 4 defined <code>ExtensionNotification</code> message codes
reserved		Reserved for future use
extElement	input	Copy of the <code>ExtensionElement</code> for the system extension currently being loaded and executed

Note:

`extElement` is not valid for the `extNotificationBeforeFirst` and `extNotificationAfterLast` messages.

An `ExtensionNotificationProc` receives an `ExtensionElementPtr` which points to a copy of the `ExtensionElement` for the system extension currently being loaded.

Note:

The data in the `ExtensionElement` is read-only. While you can change it, you'll only be changing a copy of the `ExtensionElement`. Any changes will be discarded when the `ExtensionNotificationProc` returns.

There can be any number of `ExtensionNotificationProcs` installed.

Note:

An `ExtensionNotificationProc` cannot call `RemoveExtensionNotificationProc`. If there is a need need for an `ExtensionNotificationProc` to remove itself, the removal must be deferred by, for example, installing a Notification Manager task and using the notification task to remove the `ExtensionNotificationProc`.

ExtensionTableHandlerProc

```
pascal void MyExtensionTableHandlerProc( UInt32 message, void *reserved,
                                         ExtensionTableHandle extTableHandle );
```

message	input	One of the 4 defined <code>ExtensionNotification</code> message codes
reserved		Reserved for future use
extTableHandle	input	<code>ExtensionTableHandle</code> containing all system extensions to be loaded

An `ExtensionTableHandlerProc` receives a handle to the `ExtensionTable` created at startup by the Start Manager. When an `ExtensionTableHandlerProc` is installed, the system's default handler no longer manages the `ExtensionTable`. While it is installed, the `ExtensionTableHandlerProc` is responsible for all changes to the `ExtensionTable`, except for incrementing `extElementIndex` between system extensions.

Note:

If the `ExtensionTable` or the contents of the folders included in the `ExtensionTable` are changed after installing an `ExtensionTableHandlerProc`, `RemoveExtensionTableHandlerProc` cannot be called.

The `extElementIndex` field is always incremented to point to the next system extension to be loaded immediately after the `ExtensionTableHandlerProc` is called with the `extNotificationAfterCurrent` message.

WARNING:

When controlling the loading of system extensions, the only safe time to change which `ExtensionElement` is at `ExtensionTable.extElements[extElementIndex]` is when your `ExtensionTableHandlerProc` is called with the `extNotificationAfterCurrent` message. You may change the `ExtensionTable` or the `extElementIndex` at other times, but you must ensure that the `ExtensionElement` at `ExtensionTable.extElements[extElementIndex]` stays the same.

The installed `ExtensionTableHandlerProc` is always the first handler called with `extNotificationBeforeFirst` and `extNotificationBeforeCurrent` messages and the last handler called with `extNotificationAfterLast` and `extNotificationAfterCurrent` messages.

There can only be one `ExtensionTableHandlerProc` installed.

InstallExtensionNotificationProc

```
OSErr InstallExtensionNotificationProc( ExtensionNotificationUPP extNotificationProc )
```

Parameters:

<code>extNotificationProc</code>	input	The <code>ExtensionNotificationUPP</code> to install
----------------------------------	-------	--

Results:

<code>noErr</code>	0	The <code>ExtensionNotificationUPP</code> was installed
<code>paramErr</code>	-50	This <code>ExtensionNotificationUPP</code> has already been installed
<code>memFullErr</code>	-108	Not enough memory to install the <code>ExtensionNotificationUPP</code>

Installs an `ExtensionNotificationUPP`.

Multiple `ExtensionNotificationProcs` may be installed.

RemoveExtensionNotificationProc

```
OSErr RemoveExtensionNotificationProc (ExtensionNotificationUPP  
extNotificationProc)
```

Parameters:

<code>extNotificationProc</code>	input	The <code>ExtensionNotificationUPP</code> to remove
----------------------------------	-------	---

Results:

<code>noErr</code>	0	The <code>ExtensionNotificationUPP</code> was installed
<code>paramErr</code>	-50	This <code>ExtensionNotificationUPP</code> was not found, or RemoveExtensionNotificationProc was called from within a ExtensionNotificationProc

Removes an `ExtensionNotificationUPP`.

Note:

`ExtensionNotificationProcs` cannot call `RemoveExtensionNotificationProc`.

InstallExtensionTableHandlerProc

```
OSErr InstallExtensionTableHandlerProc( ExtensionTableHandlerUPP extMgrProc,  
ExtensionTableHandle * extTable)
```

Parameters:

extMgrProc	input	The ExtensionTableHandlerUPP to install
extTable	input	A pointer to an ExtensionTableHandle where InstallExtensionTableHandlerProc will return the current ExtensionTableHandle. You don't own the handle itself and must not dispose of it, but you can change the extElementIndex, the extElementCount, and the ExtensionElements in the table.

Results:

noErr	0	The ExtensionTableHandlerUPP was installed
paramErr	-50	Another ExtensionTableHandlerUPP has already been installed
memFullErr	-108	Not enough memory to install the ExtensionTableHandlerUPP

Installs an ExtensionTableHandlerUPP.

There can only be one ExtensionTableHandlerProc installed.

RemoveExtensionTableHandlerProc

```
OSErr RemoveExtensionTableHandlerProc( ExtensionTableHandlerUPP extMgrProc )
```

Parameters:

extMgrProc	input	The ExtensionTableHandlerUPP to remove
------------	-------	--

Results:

noErr	0	The ExtensionTableHandlerUPP was removed
paramErr	-50	This ExtensionTableHandlerUPP was not installed, or the ExtensionTable no longer matches the original ExtensionTable.

Remove an ExtensionTableHandlerUPP. Control is passed back to the default handler.

Note:

If the ExtensionTable or the contents of the folders included in the ExtensionTable are changed after installing an ExtensionTableHandlerProc, RemoveExtensionTableHandlerProc cannot be called.

Summary of the Extension Table Manager

Note:

All definitions in this section are available in Universal Interfaces 3.1.

Constants

```
enum {
    gestaltExtensionTableVersion = FOUR_CHAR_CODE('etbl') /* ExtensionTable version */
};

enum {
    kExtensionTableVersion      = 0x00000100 /* current ExtensionTable version (1.0.0) */
};

/* ExtensionNotification message codes */
```

```
enum {
    extNotificationBeforeFirst = 0, /* Before any system extensions have loaded */
    extNotificationAfterLast   = 1, /* After all system extensions have loaded */
    extNotificationBeforeCurrent = 2, /* Before system extension at */
                                     /* extElementIndex is loaded */
    extNotificationAfterCurrent = 3  /* After system extension at */
                                     /* extElementIndex is loaded */
};
```

Data Types

```
struct ExtensionElement {
    Str31      fileName;           /* The file name */
    long       parentDirID;        /* the file's parent directory ID */
                                     /* and everything after ioNamePtr */
                                     /* in the HParamBlockRec.fileParam variant */
    short      ioVRefNum;          /* always the real volume reference number */
                                     /* (not a drive, default, or working dirID) */

    short      ioFRefNum;
    SInt8      ioFVersNum;
    SInt8      filler1;
    short      ioFDirIndex;        /* always 0 in table */
    SInt8      ioFlAttrib;
    SInt8      ioFlVersNum;
    FInfo      ioFlFndrInfo;
    long       ioDirID;
    unsigned short ioFlStBlk;
    long       ioFlLgLen;
    long       ioFlPyLen;
    unsigned short ioFlRStBlk;
    long       ioFlRLgLen;
    long       ioFlRPyLen;
    unsigned long ioFlCrDat;
    unsigned long ioFlMdDat;
};

typedef struct ExtensionElement      ExtensionElement;
typedef ExtensionElement *          ExtensionElementPtr;

struct ExtensionTableHeader {
    UInt32      extTableHeaderSize; /* size of ExtensionTable header ( equal to */
                                     /* offsetof(ExtensionTable, extElements[0])) */
    UInt32      extTableVersion;    /* current ExtensionTable version (same as */
                                     /* returned by gestaltExtensionTableVersion) */
    UInt32      extElementIndex;    /* current index into ExtensionElement records */
                                     /* (zero-based) */
    UInt32      extElementSize;     /* size of ExtensionElement */
    UInt32      extElementCount;    /* number of ExtensionElement records */
                                     /* in table (1-based) */
};

typedef struct ExtensionTableHeader  ExtensionTableHeader;

struct ExtensionTable {
    ExtensionTableHeader extTableHeader; /* the ExtensionTableHeader */
    ExtensionElement     extElements[1]; /* 1 element per system ext. to load */
};

typedef struct ExtensionTable      ExtensionTable;
typedef ExtensionTable *          ExtensionTablePtr;
typedef ExtensionTablePtr *       ExtensionTableHandle;
```

Routines

```
OSErr  InstallExtensionNotificationProc( ExtensionNotificationUPP  extNotificationProc)
OSErr  RemoveExtensionNotificationProc( ExtensionNotificationUPP  extNotificationProc)
OSErr  InstallExtensionTableHandlerProc( ExtensionTableHandlerUPP  extMgrProc,
                                         ExtensionTableHandle *    extTable)
OSErr  RemoveExtensionTableHandlerProc( ExtensionTableHandlerUPP  extMgrProc )
```

Result Codes

noErr	0	The ExtensionNotificationUPP was removed.
paramErr	-50	An error in the parameters prevented the normal execution of the routine.
memFullErr	-108	Not enough memory to install the ExtensionTableHandlerUPP.

Downloadables



[Acrobat version of this Note \(K\)](#)

To contact us, please use the [Contact Us](#) page.
Updated: 15-April-98

[Technotes](#)
[Previous Technote](#) | [Contents](#) | [Next Technote](#)