Process Manager Reference

Carbon > Process Management

2007-12-04

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Contents

Tables

Process Manager Reference

Framework:	Carbon/Carbon.h
Declared in	MacTypes.h Processes.h

Overview

The Process Manager provides the cooperative multitasking environment for versions of Mac OS that preceded Mac OS X. The Process Manager controls access to shared resources and manages the scheduling and execution of applications.

You can use the Process Manager to control the execution of processes and to get information about processes, including your own. You can use the Process Manager routines to

- control the execution of your application
- get information about processes
- launch other applications

Some Process Manager functions access a `ProcessInfoRec` data structure, which contains fields that are no longer applicable in a preemptively scheduled environment (for example, the `processLocation`, `processFreeMem`, and `processActiveTime` fields). Your application should avoid accessing such fields. Changes to the memory model may also affect certain fields.

Carbon does not support Process Manager functions that deal with control panels or desk accessories.

Functions by Task

Getting Process Information

`CopyProcessName` (page 9)
Gets a copy of the name of a process.

`GetCurrentProcess` (page 10)
Gets information about the current process, if any.

`GetFrontProcess` (page 10)
Gets the process serial number of the front process.

`GetNextProcess` (page 11)
Gets information about the next process, if any, in the Process Manager's internal list of open processes.

`GetProcessBundleLocation` (page 11)
Retrieves the file system location of the application bundle (or executable file) associated with a process.

`GetProcessInformation` (page 12)
Get information about a specific process.

`ProcessInformationCopyDictionary` (page 16)
Obtains a superset of `GetProcessInformation` in modern data types.

`GetProcessPID` (page 13)
Obtains the Unix PID from a process serial number.

`GetProcessForPID` (page 12)
Obtains the process serial number from a Unix PID.

`IsProcessVisible` (page 14)
Determines the visiblility of the user interface for a process.

`SameProcess` (page 17)
Determines whether two process serial numbers specify the same process.

Starting and Terminating Processes

`LaunchApplication` (page 15)
Launches an application.

`ExitToShell` (page 9)
Terminates an application.

`KillProcess` (page 14)
Terminates a process with the specified ID.

Modifying Processes

`SetFrontProcess` (page 18)
Moves a process to the foreground.

`SetFrontProcessWithOptions` (page 19)
Brings a process to the front of the process list, and activates it.

`ShowHideProcess` (page 19)
Shows or hides a given process.

`TransformProcessType` (page 20)
Changes the type of the specified process.

`WakeUpProcess` (page 20)
Makes a suspended process eligible for CPU time.

Functions

CopyProcessName

Gets a copy of the name of a process.

```
OSStatus CopyProcessName (
   const ProcessSerialNumber *psn,
   CFStringRef *name
);
```

Parameters

PSN

A pointer to a valid process serial number. See `ProcessSerialNumber` (page 25) for more information.

name

A Core Foundation string that contains the name of the specified process.

Return Value

A result code. See "Process Manager Result Codes" (page 31).

Discussion

Because the string returned is a Core Foundation string, it can represent a multilingual name, unlike the `processName` field value you obtain using `GetProcessInformation` (page 12).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Processes.h`

ExitToShell

Terminates an application.

```
void ExitToShell (
   void
);
```

Discussion

In general, you need to call `ExitToShell` only if you want your application to terminate without reaching the end of its main function.

The `ExitToShell` function terminates the calling process. The Process Manager removes your application from the list of open processes and performs any other necessary cleanup operations. In particular, all memory in your application partition and any temporary memory still allocated to your application is released. If necessary, the Application Died Apple event is sent to the process that launched your application.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BSDLLCTest

HideMenuBar

ictbSample
QTCarbonShell
QTMetaData

Declared In
`Processes.h`

GetCurrentProcess

Gets information about the current process, if any.

```
OSErr GetCurrentProcess (
    ProcessSerialNumber * PSN
);
```

Parameters

PSN

On output, a pointer to the process serial number of the current process, that is, the one currently accessing the CPU. This application can be running in either the foreground or the background.

Return Value
A result code. See "Process Manager Result Codes" (page 31).

Discussion
Applications can use this function to find their own process serial number. Drivers can use this function to find the process serial number of the current process. You can use the returned process serial number in other Process Manager functions.

This function is named `MacGetCurrentProcess` on non Macintosh platforms and `GetCurrentProcess` on Macintosh computers. However, even Macintosh code can use the `MacGetCurrentProcess` name because a macro exists that automatically maps that call to `GetCurrentProcess`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

GetFrontProcess

Gets the process serial number of the front process.

```
OSErr GetFrontProcess (
   ProcessSerialNumber *PSN
);
```

Parameters

PSN

On return, a pointer to the process serial number of the process running in the foreground.

Return Value
A result code. See "Process Manager Result Codes" (page 31). If no process is running in the foreground, returns `procNotFound`.

Discussion
You can use this function to determine if your process or some other process is in the foreground. You can use the process serial number returned in the `PSN` parameter in other Process Manager functions.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

GetNextProcess

Gets information about the next process, if any, in the Process Manager's internal list of open processes.

```
OSErr GetNextProcess (
   ProcessSerialNumber *PSN
);
```

Parameters

PSN

On input, a pointer to the process serial number of a process. This number should be a valid process serial number returned from `LaunchApplication` (page 15), `GetFrontProcess` (page 10), or `GetCurrentProcess` (page 10), or a process serial number structure containing `kNoProcess`. For details about this structure, see `ProcessSerialNumber` (page 25). On return, a pointer to the process serial number of the next process, or else `kNoProcess`.

Return Value
A result code. See "Process Manager Result Codes" (page 31).

Discussion
The Process Manager maintains a list of all open processes. You can derive this list by using repetitive calls to `GetNextProcess`. Begin generating the list by calling `GetNextProcess` and specifying the constant `kNoProcess` in the `PSN` parameter. You can then use the returned process serial number to get the process serial number of the next process. Note that the order of the list of processes is internal to the Process Manager. When the end of the list is reached, `GetNextProcess` returns the constant `kNoProcess` in the `PSN` parameter and the result code `procNotFound`.

You can use the returned process serial number in other Process Manager functions. You can also use this process serial number to specify a target application when your application sends a high-level event.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

GetProcessBundleLocation

Retrieves the file system location of the application bundle (or executable file) associated with a process.

```
OSStatus GetProcessBundleLocation (
   const ProcessSerialNumber *psn,
   FSRef *location
);
```

Parameters

PSN

A pointer to a valid process serial number. See `ProcessSerialNumber` (page 25) for more information.

location

Return Value
A result code. See "Process Manager Result Codes" (page 31).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTCarbonShell

Declared In
`Processes.h`

GetProcessForPID

Obtains the process serial number from a Unix PID.

```
OSStatus GetProcessForPID (
   pid_t pid,
   ProcessSerialNumber *psn
);
```

Parameters

pid

The Unix process ID (PID).

psn

On return, `psn` points to the process serial number.

Return Value
A result code. See "Process Manager Result Codes" (page 31).

Discussion
Note that this call does not make sense for Classic applications, since they all share a single UNIX process ID.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

GetProcessInformation

Get information about a specific process.

```
OSErr GetProcessInformation (
   const ProcessSerialNumber *PSN,
   ProcessInfoRec *info
);
```

Parameters

PSN

A pointer to a valid process serial number. You can pass a process serial number structure containing the constant `kCurrentProcess` to get information about the current process. See `ProcessSerialNumber` (page 25) for more information.

info

On return, a pointer to a structure containing information about the specified process.

Return Value

A result code. See "Process Manager Result Codes" (page 31).

Discussion

The information returned in the `info` parameter includes the application's name as it appears in the Application menu, the type and signature of the application, the address of the application partition, the number of bytes in the application partition, the number of free bytes in the application heap, the application that launched the application, the time at which the application was launched, and the location of the application file.

The `GetProcessInformation` function also returns information about the application's `'SIZE'` resource and indicates whether the process is an application or a desk accessory.

You need to specify values for the `processInfoLength`, `processName`, and `processAppSpec` fields of the process information structure. Specify the length of the process information structure in the `processInfoLength` field. If you do not want information returned in the `processName` and `processAppSpec` fields, specify `NULL` for these fields. Otherwise, allocate at least 32 bytes of storage for the string pointed to by the `processName` field and, in the `processAppSpec` field, specify a pointer to an `FSSpec` structure.

The `processName` field may not be what you expect, especially if an application has a localized name. The `processName` field, if not `NULL`, on return will contain the filename part of the executable file of the application. If you want the localized, user-displayable name for an application, call `CopyProcessName` (page 9).

In Mac OS X, the `processActiveTime` field of the returned structure is always 0, and the `modeCanBackground`, `mode32BitCompatible`, and `modeHighLevelEventAware` fields are always set.

Special Considerations

In most cases, Mac OS X applications should use `ProcessInformationCopyDictionary` (page 16) instead of this function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Processes.h`

GetProcessPID

Obtains the Unix PID from a process serial number.

```
OSStatus GetProcessPID (
   const ProcessSerialNumber *psn,
   pid_t *pid
);
```

Parameters

psn

A pointer to a valid process serial number. See `ProcessSerialNumber` (page 25) for more information.

pid

On return, `pid` points to a Unix PID.

Return Value
A result code. See "Process Manager Result Codes" (page 31).

Discussion
Note that this call does not make sense for Classic applications, since they all share a single UNIX process ID.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

IsProcessVisible

Determines the visiblility of the user interface for a process.

```
Boolean IsProcessVisible (
   const ProcessSerialNumber *psn
);
```

Parameters

PSN

A pointer to a valid process serial number. See `ProcessSerialNumber` (page 25) for more information.

Return Value
Returns `true` if the user interface is currently visible. Otherwise, returns `false`.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`Processes.h`

KillProcess

Terminates a process with the specified ID.

```
OSErr KillProcess (
   const ProcessSerialNumber *inProcess
);
```

Parameters

inProcess

The serial number of the process you want to terminate. You can also pass a process serial number structure containing the constant `kCurrentProcess` to refer to the current process. See `ProcessSerialNumber` (page 25) for more information.

Return Value
A result code. See "Process Manager Result Codes" (page 31).

Discussion
`KillProcess` terminates an process without sending a "quit" Apple event or allowing it any time to save user data or perform cleanup. You should use this function only as a last resort when all other attempts have failed. Even then, there is no guarantee that this call will succeed in killing the application, even if it returns with `noErr`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Processes.h`

LaunchApplication

Launches an application.

```
OSErr LaunchApplication (
   LaunchPBPtr LaunchParams
);
```

Parameters

LaunchParams

A pointer to a `LaunchParamBlockRec` (page 21) specifying information about the application to launch.

Return Value
A result code. See "Process Manager Result Codes" (page 31).

Discussion
The `LaunchApplication` function launches the application from the specified file and returns the process serial number, preferred partition size, and minimum partition size if the application is successfully launched.

Note that if you launch another application without terminating your application, the launched application is not actually executed until you make a subsequent call to `WaitNextEvent` or `EventAvail`.

Set the `launchContinue` flag in the `launchControlFlags` field of the launch parameter block if you want your application to continue after the specified application is launched. If you do not set this flag, `LaunchApplication` terminates your application after launching the specified application, even if the launch fails.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

ProcessInformationCopyDictionary

Obtains a superset of `GetProcessInformation` in modern data types.

```
CFDictionaryRef ProcessInformationCopyDictionary (
   const ProcessSerialNumber *PSN,
   UInt32 infoToReturn
);
```

Parameters

PSN

A pointer to a valid process serial number. See `ProcessSerialNumber` (page 25) for more information.

infoToReturn

A bitmask indicating the information to obtain. Pass `kProcessDictionaryIncludeAllInformationMask` for this parameter.

Return Value

An immutable Core Foundation dictionary containing the system information in key-value pairs.

Discussion

You should use this function instead of `GetProcessInformation` (page 12). Table 1 and Table 2 show keys you can use to get process attributes in the returned Core Foundation dictionary. All keys in the dictionary are Core Foundation strings. (Note that additional keys exist, but these are for internal use only.) Keys marked with an asterisk (*) may not appear in the dictionary, depending on the application.

Table 1 Process information keys

Key	Type	Summary
PSN	`CFNumberRef`	The process serial number. See `ProcessSerialNumber` (page 25).
Flavor	`CFNumberRef`	A hint as to the type of the application. You shouldn't need to use this key.
Attributes	`CFNumberRef`	Attributes for the process. Useful attributes generally have their own keys.
ParentPSN *	`CFNumberRef`	The process serial number of the application that launched this process.
FileType *	`CFStringRef`	The file type (if any) of the executable.
FileCreator *	`CFStringRef`	The creator type (if any) of the executable.
pid *	`CFNumberRef`	The UNIX PID for this process.
LSBackgroundOnly	`CFBooleanRef`	`kCFBooleanTrue` if the application is a background-only application.

Key	Type	Summary
LSUIElement	`CFBooleanRef`	`kCFBooleanTrue` if the application is an accessibility UIElement.
IsHiddenAttr	`CFBooleanRef`	`kCFBooleanTrue` if the application is currently hidden.
RequiresCarbon	`CFBooleanRef`	`kCFBooleanTrue` if the application's Info.plist file indicates that it is a Carbon application.
LSUIPresentationMode	`CFNumberRef`	The initial user-interface mode for the application. See *Runtime Configuration Guidelines* for a list of possible values.
BundlePath *	`CFStringRef`	The path to the application bundle (if the application is bundled).

Table 2 lists additional keys that you should reference by their predefined constants, rather than the actual string names.

Table 2 Process information key constants

Key	Type	Summary
`kCFBundleExecutableKey` *	`CFStringRef`	The path to the actual executable file.
`kCFBundleNameKey` *	`CFStringRef`	The application's display name.
`kCFBundleIdentifierKey` *	`CFStringRef`	The application's bundle identifier (if the application is bundled). For example, "com.apple.TextEdit".

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Processes.h`

SameProcess

Determines whether two process serial numbers specify the same process.

```
OSErr SameProcess (
   const ProcessSerialNumber *PSN1,
   const ProcessSerialNumber *PSN2,
   Boolean *result
);
```

Parameters

PSN1

A process serial number.

PSN2

A process serial number.

result

On return, a pointer to a Boolean value which is `TRUE` if the process serial numbers passed in `PSN1` and `PSN2` refer to the same process; otherwise `FALSE`.

Return Value
A result code. See "Process Manager Result Codes" (page 31).

Discussion
Do not attempt to compare two process serial numbers by any means other than the `SameProcess` function, because the interpretation of the bits in a process serial number is internal to the Process Manager.

The values of `PSN1` and `PSN2` must be valid process serial numbers returned from `LaunchApplication` (page 15) , `GetNextProcess` (page 11) , `GetFrontProcess` (page 10) , `GetCurrentProcess` (page 10) , or a high-level event. You can also pass a process serial number structure containing the constant `kCurrentProcess` to refer to the current process.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

SetFrontProcess

Moves a process to the foreground.

```
OSErr SetFrontProcess (
   const ProcessSerialNumber *PSN
);
```

Parameters

PSN

A pointer to a valid process serial number. You can also pass a process serial number structure containing the constant `kCurrentProcess` to refer to the current process. See `ProcessSerialNumber` (page 25) for more information.

Return Value
A result code. See "Process Manager Result Codes" (page 31).

Discussion
The `SetFrontProcess` function moves the specified process to the foreground immediately.

If the specified process serial number is invalid or if the specified process is a background-only application, `SetFrontProcess` returns a nonzero result code and does not change the current foreground process.

Special Considerations

Do not call `SetFrontProcess` at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

SetFrontProcessWithOptions

Brings a process to the front of the process list, and activates it.

```
OSStatus SetFrontProcessWithOptions (
   const ProcessSerialNumber *inProcess,
   OptionBits inOptions
);
```

Parameters

PSN

A pointer to a valid process serial number. You can also pass a process serial number structure containing the constant `kCurrentProcess` to refer to the current process. See `ProcessSerialNumber` (page 25) for more information.

inOptions

A flag that indicates how process windows should be brought forward—see the discussion below.

Return Value

A result code. See "Process Manager Result Codes" (page 31).

Discussion

If you pass 0 in the `inOptions` parameter, the process is activated and all process windows are brought forward. This is equivalent to calling `SetFrontProcess` (page 18). If you pass `kSetFrontProcessFrontWindowOnly`, the process is activated and the frontmost nonfloating window is brought forward.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Processes.h`

ShowHideProcess

Shows or hides a given process.

```
OSErr ShowHideProcess (
   const ProcessSerialNumber *psn,
   Boolean visible
);
```

Parameters

PSN

A pointer to a valid process serial number. See `ProcessSerialNumber` (page 25) for more information.

visible

A Boolean value that specifies whether you want to show (`true`) or hide (`false`) the process.

Return Value

A result code. See "Process Manager Result Codes" (page 31).

Availability

Available in Mac OS X v10.1 and later.

Declared In

`Processes.h`

TransformProcessType

Changes the type of the specified process.

```
OSStatus TransformProcessType (
   const ProcessSerialNumber *psn,
   ProcessApplicationTransformState transformState
);
```

Parameters

PSN

The serial number of the process you want to transform. You can also use the constant `kCurrentProcess` to refer to the current process. See `ProcessSerialNumber` (page 25) for more information.

transformState

A constant indicating the type of transformation you want. See "Process Transformation Constant" (page 31). Currently you can pass only `kProcessTransformToForegroundApplication`.

Return Value

A result code. See "Process Manager Result Codes" (page 31).

Discussion

You can use this call to transform a background-only application into a foreground application. A foreground application appears in the Dock (and in the Force Quit dialog) and contains a menu bar. This function does not cause the application to be brought to the front; you must call `SetFrontProcess` (page 18) to do so.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`Processes.h`

WakeUpProcess

Makes a suspended process eligible for CPU time.

```
OSErr WakeUpProcess (
   const ProcessSerialNumber *PSN
);
```

Parameters

PSN

The serial number of the process you want to wake up. You can also pass a process serial number structure containing the constant `kCurrentProcess` to refer to the current process. See `ProcessSerialNumber` (page 25) for more information.

Return Value

A result code. See "Process Manager Result Codes" (page 31).

Discussion

The `WakeUpProcess` function makes a process suspended by `WaitNextEvent` eligible to receive CPU time. A process is suspended when the value of the `sleep` parameter in the `WaitNextEvent` function is not 0 and no events for that process are pending in the event queue. This process remains suspended until the time specified in the sleep parameter expires or an event becomes available for that process. You can use

`WakeUpProcess` to make the process eligible for execution before the time specified in the `sleep` parameter expires. This function does not change the order of the processes scheduled for execution; it only makes the specified process eligible for execution.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

Data Types

AppParameters

Defines the first high-level event sent to a newly-launched application.

```
struct AppParameters {
    struct {
        UInt16 what;
        UInt32 message;
        UInt32 when;
        Point where;
        UInt16 modifiers;
    } theMsgEvent;
    unsigned long eventRefCon
    unsigned long messageLength
};
typedef struct AppParameters AppParameters;
typedef AppParameters * AppParametersPtr;
```

Discussion
The application parameters structure is used in the `launchAppParameters` field of the launch parameter block, `LaunchParamBlockRec` (page 21) , whose address is passed to the `LaunchApplication` (page 15) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

LaunchParamBlockRec

Defines the required parameters when launching an application.

```
struct LaunchParamBlockRec {
    unsigned long reserved1;
    unsigned short reserved2;
    unsigned short launchBlockID;
    unsigned long launchEPBLength;
    unsigned short launchFileFlags;
    LaunchFlags launchControlFlags;
    FSSpecPtr launchAppSpec;
    ProcessSerialNumber launchProcessSN;
    unsigned long launchPreferredSize;
    unsigned long launchMinimumSize;
    unsigned long launchAvailableSize;
    AppParametersPtr launchAppParameters;
};
typedef struct LaunchParamBlockRec LaunchParamBlockRec;
typedef LaunchParamBlockRec * LaunchPBPtr;
```

Fields

`reserved1`
: Reserved.

`reserved2`
: Reserved.

`launchBlockID`
: A value that indicates whether you are using the fields following it in the launch parameter block. Specify the constant `extendedBlock` if you use the fields that follow it.

`launchEPBLength`
: The length of the fields following this field in the launch parameter block. Use the constant `extendedBlockLen` to specify this value.

`launchFileFlags`
: The Finder flags for the application file. Set the `launchNoFileFlags` constant in the `launchControlFlags` field if you want the `LaunchApplication` function to extract the Finder flags from the application file and to set the `launchFileFlags` field for you.

`launchControlFlags`
: See "Launch Options" (page 28) for a complete description of these flags.

`launchAppSpec`
: A pointer to a file specification structure that gives the location of the application file to launch.

`launchProcessSN`
: The process serial number returned to your application if the launch is successful. You can use this process serial number in other Process Manager functions to refer to the launched application.

`launchPreferredSize`
: The preferred partition size for the launched application as specified in the launched application's `'SIZE'` resource. `LaunchApplication` sets this field to 0 if an error occurred or if the application is already open.

`launchMinimumSize`
: The minimum partition size for the launched application as specified in the launched application's `'SIZE'` resource. `LaunchApplication` sets this field to 0 if an error occurred or if the application is already open.

`launchAvailableSize`

The maximum partition size that is available for allocation. This value is returned to your application only if the `memFullErr` result code is returned. If the application launch fails because of insufficient memory, you can use this value to determine if there is enough memory available to launch in the minimum size.

`launchAppParameters`

The first high-level event to send to the launched application. If you set this field to `NULL`, `LaunchApplication` creates and sends the Open Application Apple event to the launched application.

Discussion

You specify a launch parameter block as a parameter to the `LaunchApplication` (page 15) function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Processes.h`

ProcessInfoRec

Defines the structure of a process information record.

```
struct ProcessInfoRec {
    unsigned long processInfoLength;
    StringPtr processName;
    ProcessSerialNumber processNumber;
    unsigned long processType;
    OSType processSignature;
    unsigned long processMode;
    Ptr processLocation;
    unsigned long processSize;
    unsigned long processFreeMem;
    ProcessSerialNumber processLauncher;
    unsigned long processLaunchDate;
    unsigned long processActiveTime;
    FSSpecPtr processAppSpec;
};
typedef struct ProcessInfoRec ProcessInfoRec;
typedef ProcessInfoRec * ProcessInfoRecPtr;
```

Fields

`processInfoLength`

The number of bytes in the process information structure. For compatibility, you should specify the length of the structure in this field.

`processName`

The name of the application. This field contains the name of the application as designated by the user at the time the application was opened. For example, for foreground applications, the `processName` field contains the name as it appears in the Application menu. You must specify `NULL` in the `processName` field if you do not want the application name returned. Otherwise, you should allocate at least 32 bytes of storage for the string pointed to by the `processName` field. Note that the `processName` field specifies the name of the application, whereas the `processAppSpec` field specifies the location of the file.

`processNumber`

The process serial number.

`processType`
: The file type of the application, generally `'APPL'` for applications and `'appe'` for background-only applications launched at startup.

`processSignature`
: The signature (or creator) of the file containing the application.

`processMode`
: Process mode flags. These flags indicate whether the process is an application or desk accessory. For applications, this field also returns information specified in the application's `'SIZE'` resource. This information is returned as flags.

  On Mac OS X, some flags in `processMode` will not be set as they were on Mac OS 9, even for Classic applications. Mac OS X doesn't support applications which can't be sent into the background, so `modeCanBackground` will always be set. Similarly, Mac OS X applications will always have `mode32BitCompatible` and `modeHighLevelEventAware` set

`processLocation`
: The beginning address of the application partition.

`processSize`
: The number of bytes in the application partition (including the heap, stack, and A5 world).

`processFreeMem`
: The number of free bytes in the application heap.

`processLauncher`
: The process serial number of the process that launched the application or desk accessory. If the original launcher of the process is no longer open, the `lowLongOfPSN` field of the process serial number structure contains the constant `kNoProcess`.

`processLaunchDate`
: The value of the `Ticks` global variable at the time that the process was launched.

`processActiveTime`
: The accumulated time, in ticks, during which the process has used the CPU, including both foreground and background processing time.

`processAppSpec`
: The address of a file specification structure that stores the location of the file containing the application or `'DRVR'` resource. You should specify `NULL` in the `processAppSpec` field if you do not want the `FSSpec` structure of the file returned.

Discussion
A process information record is returned by the `GetProcessInformation` (page 12) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

ProcessInfoExtendedRec

Defines an extended version of the process information record.

```
struct ProcessInfoExtendedRec {
    unsigned long processInfoLength;
    StringPtr processName;
    ProcessSerialNumber processNumber;
    unsigned long processType;
    OSType processSignature;
    unsigned long processMode;
    Ptr processLocation;
    unsigned long processSize;
    unsigned long processFreeMem;
    ProcessSerialNumber processLauncher;
    unsigned long processLaunchDate;
    unsigned long processActiveTime;
    FSSpecPtr processAppSpec;
    unsigned long processTempMemTotal;
    unsigned long processPurgeableTempMemTotal;
};
typedef struct ProcessInfoExtendedRec ProcessInfoExtendedRec;
typedef ProcessInfoExtendedRec * ProcessInfoExtendedRecPtr;
```

Discussion
See `ProcessInfoRec` (page 23) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

ProcessSerialNumber

Defines the unique identifier for an open process.

```
struct ProcessSerialNumber {
    unsigned long highLongOfPSN;
    unsigned long lowLongOfPSN;
};
typedef struct ProcessSerialNumber ProcessSerialNumber;
typedef ProcessSerialNumber * ProcessSerialNumberPtr;
```

Fields

`highLongOfPSN`
The high-order long integer of the process serial number.

`lowLongOfPSN`
The low-order long integer of the process serial number.

Discussion
All applications (defined as things which can appear in the Dock that are not documents and are launched by the Finder or Dock) on Mac OS X have a unique process serial number. This number is created when the application launches, and remains until the application quits. Other system services, like Apple events, use the `ProcessSerialNumber` structure to specify an application.

During launch, every application "checks in" with the Process Manager. Before this checkin, the application can not receive events or draw to the screen. Prior to Mac OS 10.2, this check in occurred before the applications's `main` function was entered. In Mac OS 10.2 and later, this check in does not occur until the first time the application calls a Process Manager function, or until it enters `CFRunLoopRun` for the main

event loop. This allows tools and other executables which do not need to receive events to link against more of the higher level toolbox frameworks, but may cause a problem if the application expects to be able to retrieve events or use CoreGraphics services before this checkin has occurred. An application can force the connection to the Process Manager to be set up by calling any Process Manager routine, but the recommended way to do this is to call `GetCurrentProcess` (page 10) to ask for the current application's PSN. Doing so initializes the connection to the Process Manager if it has not already been set up and "check in" the application with the system.

You should not make any assumptions about the meaning of the bits in a process serial number. To compare two process serial numbers, you should use the function `SameProcess` (page 17).

You can obtain a process serial number in one of the following ways:

- Process serial numbers are returned by the functions `LaunchApplication` (page 15), `GetCurrentProcess` (page 10), and `GetFrontProcess` (page 10).
- Some high-level events return process serial numbers.

If you want to specify a process using the "Process Identification Constants" (page 30), you must populate a process serial number structure, passing 0 in `highLongOfPSN` and the appropriate constant (such as `kCurrentProcess`) in `lowLongOfPSN`. For example, to bring the current process forward, you can use the following code:

```
ProcessSerialNumber psn = { 0, kCurrentProcess };
SetFrontProcess( &psn );
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTypes.h`

SizeResourceRec

Defines a representation of the `SIZE` resource.

```
struct SizeResourceRec {
    unsigned short flags;
    unsigned long preferredHeapSize;
    unsigned long minimumHeapSize;
};
typedef struct SizeResourceRec SizeResourceRec;
typedef SizeResourceRec * SizeResourceRecPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

Constants

Control Panel Result Codes

Specifies the values that a control panel can return.

Unsupported

```
enum {
    cdevGenErr = -1,
    cdevMemErr = 0,
    cdevResErr = 1,
    cdevUnset = 3
};
```

Extension Launch Codes

Specifies the values used when launching extensions.

```
enum {
    extendedBlock = 0x4C43,
    extendedBlockLen = sizeof(LaunchParamBlockRec) - 12
};
```

Control Panel Message Codes

Specifies the values for messages to a control panel.

```
enum {
    initDev = 0,
    hitDev = 1,
    closeDev = 2,
    nulDev = 3,
    updateDev = 4,
    activDev = 5,
    deactivDev = 6,
    keyEvtDev = 7,
    macDev = 8,
    undoDev = 9,
    cutDev = 10,
    copyDev = 11,
    pasteDev = 12,
    clearDev = 13,
    cursorDev = 14
};
```

Termination Options

Specifies masks to control the timing of application termination during system shutdown or restart.

```
enum {
    kQuitBeforeNormalTimeMask = 1,
    kQuitAtNormalTimeMask = 2,
    kQuitBeforeFBAsQuitMask = 4,
    kQuitBeforeShellQuitsMask = 8,
    kQuitBeforeTerminatorAppQuitsMask = 16,
    kQuitNeverMask = 32,
    kQuitOptionsMask = 0x7F,
    kQuitNotQuitDuringInstallMask = 0x0100,
    kQuitNotQuitDuringLogoutMask = 0x0200
};
```

Discussion
Applications and background applications can control when they are asked to quit by the system at restart and shutdown by setting these bits in a `'quit'(0)` resource located in the resource fork.

Applications without this resource are terminated at `kQuitAtNormalTime`.

Availability
Available in CarbonLib 1.0 and later. Not available in Mac OS X version 10.0 and later.

Front Process Options

Specifies options for bringing windows forward when a process is activated.

```
enum {
    kSetFrontProcessFrontWindowOnly = (1 << 0)
};
```

Constants

`kSetFrontProcessFrontWindowOnly`
Activate the process, but bring only the frontmost non-floating window forward.

Available in Mac OS X version 10.2 and later.

Declared in `Processes.h`.

Launch Options

Specifies the valid launch options in the `launchControlFlags` field of the launch parameter block.

```
typedef unsigned short LaunchFlags;
enum {
  launchContinue                = 0x4000,
  launchNoFileFlags             = 0x0800,
  launchUseMinimum              = 0x0400,
  launchDontSwitch              = 0x0200,
  launchAllow24Bit              = 0x0100,
  launchInhibitDaemon           = 0x0080
};
```

Constants

`launchContinue`

Set this flag if you want your application to continue after the specified application is launched. If you do not set this flag, `LaunchApplication` terminates your application after launching the specified application, even if the launch fails.

Available in Mac OS X v10.0 and later.

Declared in `Processes.h`.

`launchNoFileFlags`

Set this flag if you want the `LaunchApplication` function to ignore any value specified in the `launchFileFlags` field. If you set the `launchNoFileFlags` flag, the `LaunchApplication` function extracts the Finder flags from the application file for you. If you want to supply the file flags, clear the `launchNoFileFlags` flag and specify the Finder flags in the `launchFileFlags` field of the launch parameter block.

Available in Mac OS X v10.0 and later.

Declared in `Processes.h`.

`launchUseMinimum`

Clear this flag if you want the `LaunchApplication` function to attempt to launch the application in the preferred size (as specified in the application's `'SIZE'` resource). If you set the `launchUseMinimum` flag, the `LaunchApplication` function attempts to launch the application using the largest available size greater than or equal to the minimum size but less than the preferred size. If the `LaunchApplication` function returns the result code `memFullErr` or `memFragErr`, the application cannot be launched under the current memory conditions.

Available in Mac OS X v10.0 and later.

Declared in `Processes.h`.

`launchDontSwitch`

Set this flag if you do not want the launched application brought to the front. If you set this flag, the launched application runs in the background until the user brings the application to the front—for example, by clicking in one of the application's windows. Note that most applications expect to be launched in the foreground. If you clear the `launchDontSwitch` flag, the launched application is brought to the front, and your application is sent to the background.

Available in Mac OS X v10.0 and later.

Declared in `Processes.h`.

`launchAllow24Bit`

Available in Mac OS X v10.0 and later.

Declared in `Processes.h`.

`launchInhibitDaemon`
Set this flag if you do not want `LaunchApplication` to launch a background-only application. (A background-only application has the `onlyBackground` flag set in its `'SIZE'` resource.)

Available in Mac OS X v10.0 and later.

Declared in `Processes.h`.

Discussion
For more information, see `LaunchApplication` (page 15) and `LaunchParamBlockRec` (page 21).

Process Mode Flags

Specifies the type of information returned in a process information record.

```
enum {
    modeReserved = 0x01000000,
    modeControlPanel = 0x00080000,
    modeLaunchDontSwitch = 0x00040000,
    modeDeskAccessory = 0x00020000,
    modeMultiLaunch = 0x00010000,
    modeNeedSuspendResume = 0x00004000,
    modeCanBackground = 0x00001000,
    modeDoesActivateOnFGSwitch = 0x00000800,
    modeOnlyBackground = 0x00000400,
    modeGetFrontClicks = 0x00000200,
    modeGetAppDiedMsg = 0x00000100,
    mode32BitCompatible = 0x00000080,
    modeHighLevelEventAware = 0x00000040,
    modeLocalAndRemoteHLEvents = 0x00000020,
    modeStationeryAware = 0x00000010,
    modeUseTextEditServices = 0x00000008,
    modeDisplayManagerAware = 0x00000004
};
```

Discussion
These constants indicate, in the `processMode` field of the `ProcessInfoRec` (page 23) structure, whether the process is an application or a desk accessory. If the process is an application, these flags return information about the application's `'SIZE'` resource.

Process Identification Constants

Specifies constants used instead of a process serial number to identify a process.

```
enum {
    kNoProcess = 0,
    kSystemProcess = 1,
    kCurrentProcess = 2
};
```

Constants
`kNoProcess`
Identifies a process that doesn't exist.

Available in Mac OS X v10.0 and later.

Declared in `Processes.h`.

`kSystemProcess`
Identifies a process that belongs to the Operating System.

Available in Mac OS X v10.0 and later.

Declared in `Processes.h`.

`kCurrentProcess`
Identifies the current process.

Available in Mac OS X v10.0 and later.

Declared in `Processes.h`.

Discussion
If you want to use these constants to specify a process, you must populate a process serial number structure (`ProcessSerialNumber` (page 25)), passing 0 in the `highLongOfPSN` field and the appropriate constant (such as `kCurrentProcess`) in the `lowLongOfPSN`. For example, to bring the current process forward, you can use the following code:

```
ProcessSerialNumber psn = { 0, kCurrentProcess };
SetFrontProcess( &psn );
```

Process Transformation Constant

Specify tranformation types to be applied when calling `TransformProcessType` (page 20).

```
enum {
  kProcessTransformToForegroundApplication = 1L
};
typedef UInt32 ProcessApplicationTransformState;
```

Constants
`kProcessTransformToForegroundApplication`
Use to convert a background-only application to a foreground application.

Available in Mac OS X v10.3 and later.

Declared in `Processes.h`.

Result Codes

The table below lists the most common result codes returned by the Process Manager.

Result Code	Value	Description
`procNotFound`	-600	No eligible process with specified process serial number. Available in Mac OS X v10.0 and later.
`memFragErr`	-601	Not enough room to launch application with special requirements. Available in Mac OS X v10.0 and later.
`appModeErr`	-602	Addressing mode is 32-bit, but application is not 32-bit clean. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
protocolErr	-603	app made module calls in improper order Available in Mac OS X v10.0 and later.
hardwareConfigErr	-604	Hardware configuration not supported. Available in Mac OS X v10.0 and later.
appMemFullErr	-605	Partition size specified in SIZE resource is not big enough for launch. Available in Mac OS X v10.0 and later.
appIsDaemon	-606	Application runs in background only. Available in Mac OS X v10.0 and later.
wrongApplicationPlatform	-875	The application could not launch because the required platform is not available. Available in Mac OS X v10.0 and later.
appVersionTooOld	-876	The application's creator and version are incompatible with the current version of Mac OS. Available in Mac OS X v10.0 and later.
notAppropriateForClassic	-877	This application will not (or should not) run in Classic. Available in Mac OS X v10.0 and later.

Document Revision History

This table describes the changes to *Process Manager Reference*.

Date	Notes
2007-12-04	Updated description of `ProcessInformationCopyDictionary` (page 16) function.
2005-12-06	Clarified how to use process identification constants (such as kCurrentProcess). Removed outdated material pertaining to Mac OS 9 or earlier.
2005-07-07	Updated GetProcessInformationCopyDictionary keys. Added documentation for KillProcess and TransformProcessType.
2003-05-19	Added documentation for functions `ProcessInformationCopyDictionary` (page 16), `GetProcessPID` (page 13), and `GetProcessForPID` (page 12).
2003-04-01	Added documentation for the function `ShowHideProcess` (page 19).
2003-02-01	Updated to include Mac OS X availability information.

Document Revision History

Index

N

P

S

T

W