Class

# ProcessInfo

A collection of information about the current process.

iOS 2.0+ | iPadOS 2.0+ | Mac Catalyst 13.0+ | macOS 10.0+ | tvOS 9.0+ | visionOS 1.0+ | watchOS 2.0+

```
class ProcessInfo
```

## Overview

Each process has a single, shared `ProcessInfo` object known as a *process information agent* that can return information such as arguments, environment variables, host name, and process name. The `processInfo` class method returns the shared agent for the current process. For example, the following line returns the `ProcessInfo` object, which then provides the name of the current process:

Swift | Objective-C

```
let processName = ProcessInfo.processInfo.processName
```

> **Note**
>
> `ProcessInfo` is thread-safe in macOS 10.7 and later.

The `ProcessInfo` class also includes the `operatingSystemVersion` property, which returns an `OperatingSystemVersion` structure identifying the operating system version on which the process is executing.

`ProcessInfo` objects attempt to interpret environment variables and command-line arguments in the user's default C string encoding if they can't convert to Unicode as UTF-8 strings. If neither the Unicode nor C string conversion works, the `ProcessInfo` object ignores these values.

## Manage Activities

The system has heuristics to improve battery life, performance, and responsiveness of applications for the benefit of the user. You can use the following methods to manage *activities* that give hints to the system that your application has special requirements:

- `beginActivity(options:reason:)`
- `endActivity(_:)`
- `performActivity(options:reason:using:)`

In response to creating an activity, the system disables some or all of the heuristics so your application can finish quickly while still providing responsive behavior if the user needs it.

You use activities when your application performs a long-running operation. If the activity can take different amounts of time (for example, calculating the next move in a chess game), it should use this API to ensure correct behavior when the amount of data or the capabilities of the user's computer varies. Activities fall into two major categories:

- *User-initiated* activities are explicitly started by the user. Examples include exporting or downloading a user-specified file.
- *Background* activities perform the normal operations of your application and aren't explicitly started by the user. Examples include autosaving, indexing, and automatic downloading of files.

In addition, if your application requires high priority input/output (I/O), you can include the `latencyCritical` flag (using a bitwise OR). You should only use this flag for activities like audio or video recording that require high priority I/O.

If your activity takes place synchronously inside an event callback on the main thread, you don't need to use this API.

Be aware that failing to end these activities for an extended period of time can have significant negative impacts on the performance of your user's computer, so be sure to use only the minimum amount of time required. User preferences may override your application's request.

You can also use this API to control automatic termination or sudden termination (see Support Sudden Termination). For example, the following code brackets the work to protect it from sudden termination:

Swift Objective-C

```
let activity = ProcessInfo.processInfo.beginActivity(
    options: .automaticTerminationDisabled,
    reason: "Good Reason")
// Perform some work.
ProcessInfo.processInfo.endActivity(activity)
```

The above example is equivalent to the following code, which uses the `disableAutomaticTermination(_:)` method:

Swift Objective-C

```
ProcessInfo.processInfo.disableAutomaticTermination("Good Reason")
// Perform some work.
ProcessInfo.processInfo.enableAutomaticTermination("Good Reason")
```

Because this API returns an object, it may be easier to pair begins and ends than when using the automatic termination API. If your app deallocates the object before the `endActivity(_:)` call, the activity ends automatically.

This API also provides a mechanism to disable system-wide idle sleep and display idle sleep. These can have a large impact on the user experience, so be careful to end activities that disable sleep (including `userInitiated`).

## Support Sudden Termination

macOS 10.6 and later includes a mechanism that allows the system to log out or shut down more quickly by, whenever possible, killing applications instead of requesting that they quit themselves.

Your application can enable this capability on a global basis and then manually override its availability during actions that could cause data corruption or a poor user experience by allowing sudden termination.

Alternatively, your application can manually enable and disable this functionality. Creating a process assigns a counter that indicates if the process is safe to terminate. You decrement and increment the counter using the methods `enableSuddenTermination()` and `disableSuddenTermination()`. A value of `0` enables the system to terminate the process without first sending a notification or event.

Your application can support sudden termination upon launch by adding a key to the application's `Info.plist` file. If the `NSSupportsSuddenTermination` key exists in the `Info.plist` file and has a value of `true`, it's the equivalent of calling `enableSuddenTermination()` during

your application launch. This allows the system to terminate the process immediately. You can still override this behavior by invoking `disableSuddenTermination()`.

Typically, you disable sudden termination whenever your app defers work that the app must complete before it terminates. If, for example, your app defers writing data to disk and enables sudden termination, you should bracket the sensitive operations with a call to `disableSuddenTermination()`, perform the necessary operations, and then send a balancing `enableSuddenTermination()` message.

In agents or daemon executables that don't depend on AppKit, you can manually invoke `enableSuddenTermination()` right away. You can then use the enable and disable methods whenever the process has work it must do before it terminates.

Some AppKit functionality automatically disables sudden termination on a temporary basis to ensure data integrity.

- `UserDefaults` temporarily disables sudden termination to prevent the process from terminating between the time at which it sets the default and the time at which it writes the preferences file — including that default — to disk.
- `NSDocument` temporarily disables sudden termination to prevent the process from terminating between the time at which the user has made a change to a document and the time at which `NSDocument` writes the user's change to disk.

> **Tip**
>
> You can determine the value of the sudden termination using the following LLDB command.
>
> ```
> print (long)[[NSClassFromString(@"NSProcessInfo") processInfo] _suddenTerminatic
> ```
>
> Don't attempt to invoke or override `suddenTerminationDisablingCount` (a private method) in your application. It's there for this debugging purpose and may disappear at any time.

## Monitor Thermal State to Adjust App Performance

*Thermal state* indicates the level of heat generated by logic components as they run apps. As the thermal state increases, the system decreases heat by reducing the speed of the processors. Optimize your app's performance by monitoring the thermal state and reducing system usage as the thermal state increases. Query the current state with `thermalState` to determine if your app needs to reduce system usage. You can register the `thermalStateDidChangeNotification` for notifications of a change in thermal state. For recommended actions, see `ProcessInfo.ThermalState`.

# Topics

## Getting the Process Information Agent

`class var processInfo: ProcessInfo`

Returns the process information agent for the process.

## Accessing Process Information

`var arguments: [String]`

Array of strings with the command-line arguments for the process.

`var environment: [String : String]`

The variable names (keys) and their values in the environment from which the process was launched.

`var globallyUniqueString: String`

Global unique identifier for the process.

`var isMacCatalystApp: Bool`

A Boolean value that indicates whether the process originated as an iOS app and runs on macOS.

`var isiOSAppOnMac: Bool`

A Boolean value that indicates whether the process is an iPhone or iPad app running on a Mac.

`var isiOSAppOnVision: Bool`

A Boolean value that indicates whether the process is an iPhone or iPad app running on visionOS.

`var processIdentifier: Int32`

The identifier of the process (often called process ID).

`var processName: String`

The name of the process.

## Accessing User Information

Each user account in macOS has a full name (e.g., “Johnny Appleseed”) and an account name (e.g., “jappleseed”). You can view these names from the *Users & Groups* pane of System Preferences, and you can use either name to log in to your Mac.

`var userName: String`

Returns the account name of the current user.

`var fullUserName: String`

Returns the full name of the current user.

## Sudden Application Termination

`func disableSuddenTermination()`

Disables the application for quickly killing using sudden termination.

`func enableSuddenTermination()`

Enables the application for quick killing using sudden termination.

## Controlling Automatic Termination

`func disableAutomaticTermination(String)`

Disables automatic termination for the application.

`func enableAutomaticTermination(String)`

Enables automatic termination for the application.

`var automaticTerminationSupportEnabled: Bool`

A Boolean value indicating whether the app supports automatic termination.

## Getting Host Information

`var hostName: String`

The name of the host computer on which the process is executing.

`var operatingSystemVersionString: String`

A string containing the version of the operating system on which the process is executing.

`var operatingSystemVersion: OperatingSystemVersion`

The version of the operating system on which the process is executing.

`func isOperatingSystemAtLeast(OperatingSystemVersion) -> Bool`

Returns a Boolean value indicating whether the version of the operating system on which the process is executing is the same or later than the given version.

~~`func operatingSystem() -> Int`~~

Returns a constant to indicate the operating system on which the process is executing.

Deprecated

~~`func operatingSystemName() -> String`~~

Returns a string containing the name of the operating system on which the process is executing.

Deprecated

## Getting Computer Information

`var processorCount: Int`

The number of processing cores available on the computer.

`var activeProcessorCount: Int`

The number of active processing cores available on the computer.

`var physicalMemory: UInt64`

The amount of physical memory on the computer in bytes.

`func isDeviceCertified(for: NSDeviceCertification) -> Bool`

Indicates whether the device supports the requested performance tier.

`func hasPerformanceProfile(NSProcessPerformanceProfile) -> Bool`

Indicates whether an app is running under a known performance profile.

`var systemUptime: TimeInterval`

The amount of time the system has been awake since the last time it was restarted.

## Managing Activities

```
func beginActivity(options: ProcessInfo.ActivityOptions, reason: String
) -> any NSObjectProtocol
```

Begin an activity using the given options and reason.

`func endActivity(any NSObjectProtocol)`

Ends the given activity.

`func performActivity(options: ProcessInfo.ActivityOptions, reason: String, using: () -> Void)`

Synchronously perform an activity defined by a given block using the given options.

`func performExpiringActivity(withReason: String, using: (Bool) -> Void)`

Performs the specified block asynchronously and notifies you if the process is about to be suspended.

## Getting the Thermal State

`var thermalState: ProcessInfo.ThermalState`

The current thermal state of the system.

## Determining Whether Low Power Mode is Enabled

`var isLowPowerModeEnabled: Bool`

A Boolean value that indicates the current state of Low Power Mode.

## Constants

`struct OperatingSystemVersion`

A structure that contains version information about the currently executing operating system, including major, minor, and patch version numbers.

`struct ActivityOptions`

Option flags used with `beginActivity(options:reason:)` and `performActivity(options:reason:using:)`.

`enum ThermalState`

Values used to indicate the system's thermal state.

Anonymous

The following constants are provided by the `NSProcessInfo` class as return values for `operatingSystem()`.

## Notifications

`class let thermalStateDidChangeNotification: NSNotification.Name`

Posts when the thermal state of the system changes.

`static let NSProcessInfoPowerStateDidChange: NSNotification.Name`

Posts when the power state of a device changes.

### Structures

`struct PowerStateDidChangeMessage`

`struct ThermalStateDidChangeMessage`

---

## Relationships

### Inherits From

`NSObject`

### Conforms To

`CVarArg`
`CustomDebugStringConvertible`
`CustomStringConvertible`
`Equatable`
`Hashable`
`NSObjectProtocol`
`Sendable`
`SendableMetatype`