Structure

TaskPriority

The priority of a task.

iOS 13.0+ | iPadOS 13.0+ | Mac Catalyst 13.0+ | macOS 10.15+ | tvOS 13.0+ | visionOS 1.0+ | watchOS 6.0+

```
struct TaskPriority
```

Overview

The executor determines how priority information affects the way tasks are scheduled. The behavior varies depending on the executor currently being used. Typically, executors attempt to run tasks with a higher priority before tasks with a lower priority. However, the semantics of how priority is treated are left up to each platform and `Executor` implementation.

Child tasks automatically inherit their parent task's priority. Detached tasks created by `detach(priority:operation:)` don't inherit task priority because they aren't attached to the current task.

In some situations the priority of a task is elevated — that is, the task is treated as it if had a higher priority, without actually changing the priority of the task:

- If a task runs on behalf of an actor, and a new higher-priority task is enqueued to the actor, then the actor's current task is temporarily elevated to the priority of the enqueued task. This priority elevation allows the new task to be processed at the priority it was enqueued with.
- If a higher-priority task calls the `get()` method, then the priority of this task increases until the task completes.

In both cases, priority elevation helps you prevent a low-priority task from blocking the execution of a high priority task, which is also known as *priority inversion*.

Topics

Operators

```
static func != (TaskPriority, TaskPriority) -> Bool
```

Initializers

```
init?(JobPriority)
```

Convert this UnownedJob/Priority to a TaskPriority.

```
init(rawValue: UInt8)
```

Creates a new instance with the specified raw value.

Instance Properties

```
var rawValue: UInt8
```

The corresponding value of the raw type.

Type Aliases

```
typealias RawValue
```

The raw type that can be used to represent all values of the conforming type.

Type Properties

```
static let background: TaskPriority
```

~~`static let `default`: TaskPriority`~~ Deprecated

```
static let high: TaskPriority
```

```
static let low: TaskPriority
```

```
static var medium: TaskPriority
```

~~`static var unspecified: TaskPriority`~~ Deprecated

```
static let userInitiated: TaskPriority
```

~~`static var userInteractive: TaskPriority`~~ Deprecated

`static let utility: TaskPriority`

Default Implementations

- Comparable Implementations
- CustomStringConvertible Implementations
- Equatable Implementations
- RawRepresentable Implementations

Relationships

Conforms To

`Comparable`
`Copyable`
`CustomStringConvertible`
`Decodable`
`Encodable`
`Equatable`
`RawRepresentable`
`Sendable`
`SendableMetatype`

See Also

Tasks

`struct Task`

A unit of asynchronous work.

`struct TaskGroup`

A group that contains dynamically created child tasks.

```
func withTaskGroup<ChildTaskResult, GroupResult>(of: ChildTaskResult
.Type, returning: GroupResult.Type, isolation: isolated (any Actor)?,
body: (inout TaskGroup<ChildTaskResult>) async -> GroupResult) async ->
GroupResult
```

Starts a new scope that can contain a dynamic number of child tasks.

```
struct ThrowingTaskGroup
```

A group that contains throwing, dynamically created child tasks.

```
func withThrowingTaskGroup<ChildTaskResult, GroupResult>(of: ChildTask
Result.Type, returning: GroupResult.Type, isolation: isolated (any
Actor)?, body: (inout ThrowingTaskGroup<ChildTaskResult, any Error>)
async throws -> GroupResult) async rethrows -> GroupResult
```

Starts a new scope that can contain a dynamic number of throwing child tasks.

```
struct DiscardingTaskGroup
```

A discarding group that contains dynamically created child tasks.

```
func withDiscardingTaskGroup<GroupResult>(returning: GroupResult.Type,
isolation: isolated (any Actor)?, body: (inout DiscardingTaskGroup)
async -> GroupResult) async -> GroupResult
```

Starts a new scope that can contain a dynamic number of child tasks.

```
struct ThrowingDiscardingTaskGroup
```

A throwing discarding group that contains dynamically created child tasks.

```
func withThrowingDiscardingTaskGroup<GroupResult>(returning: Group
Result.Type, isolation: isolated (any Actor)?, body: (inout Throwing
DiscardingTaskGroup<any Error>) async throws -> GroupResult) async
throws -> GroupResult
```

Starts a new scope that can contain a dynamic number of child tasks.

```
struct UnsafeCurrentTask
```

An unsafe reference to the current task.