Protocol

AsyncSequence

A type that provides asynchronous, sequential, iterated access to its elements.

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

```
protocol AsyncSequence<Element, Failure>
```

Overview

An `AsyncSequence` resembles the `Sequence` type — offering a list of values you can step through one at a time — and adds asynchronicity. An `AsyncSequence` may have all, some, or none of its values available when you first use it. Instead, you use `await` to receive values as they become available.

As with `Sequence`, you typically iterate through an `AsyncSequence` with a `for await-in` loop. However, because the caller must potentially wait for values, you use the `await` keyword. The following example shows how to iterate over `Counter`, a custom `AsyncSequence` that produces `Int` values from 1 up to a `howHigh` value:

```
for await number in Counter(howHigh: 10) {
    print(number, terminator: " ")
}
// Prints "1 2 3 4 5 6 7 8 9 10 "
```

An `AsyncSequence` doesn't generate or contain the values; it just defines how you access them. Along with defining the type of values as an associated type called `Element`, the `AsyncSequence` defines a `makeAsyncIterator()` method. This returns an instance of type `AsyncIterator`. Like the standard `IteratorProtocol`, the `AsyncIteratorProtocol` defines a single `next()` method to produce elements. The difference is that the `AsyncIterator` defines

its `next()` method as `async`, which requires a caller to wait for the next value with the `await` keyword.

`AsyncSequence` also defines methods for processing the elements you receive, modeled on the operations provided by the basic `Sequence` in the standard library. There are two categories of methods: those that return a single value, and those that return another `AsyncSequence`.

Single-value methods eliminate the need for a `for await-in` loop, and instead let you make a single `await` call. For example, the `contains(_:)` method returns a Boolean value that indicates if a given value exists in the `AsyncSequence`. Given the `Counter` sequence from the previous example, you can test for the existence of a sequence member with a one-line call:

```
let found = await Counter(howHigh: 10).contains(5) // true
```

Methods that return another `AsyncSequence` return a type specific to the method's semantics. For example, the `.map(_:)` method returns a `AsyncMapSequence` (or a `AsyncThrowingMapSequence`, if the closure you provide to the `map(_:)` method can throw an error). These returned sequences don't eagerly await the next member of the sequence, which allows the caller to decide when to start work. Typically, you'll iterate over these sequences with `for await-in`, like the base `AsyncSequence` you started with. In the following example, the `map(_:)` method transforms each `Int` received from a `Counter` sequence into a `String`:

```
let stream = Counter(howHigh: 10)
    .map { $0 % 2 == 0 ? "Even" : "Odd" }
for await s in stream {
    print(s, terminator: " ")
}
// Prints "Odd Even Odd Even Odd Even Odd Even Odd Even "
```

Topics

Creating an Iterator

`func makeAsyncIterator() -> Self.AsyncIterator`

Creates the asynchronous iterator that produces elements of this asynchronous sequence.

Required

`associatedtype AsyncIterator : AsyncIteratorProtocol`

The type of asynchronous iterator that produces elements of this asynchronous sequence.

Required

```
protocol AsyncIteratorProtocol
```

A type that asynchronously supplies the values of a sequence one at a time.

```
associatedtype Element
```

The type of element produced by this asynchronous sequence.

Required

Finding Elements

```
func contains(Self.Element) async rethrows -> Bool
```

Returns a Boolean value that indicates whether the asynchronous sequence contains the given element.

```
func contains(where: (Self.Element) async throws -> Bool) async rethrows -> Bool
```

Returns a Boolean value that indicates whether the asynchronous sequence contains an element that satisfies the given predicate.

```
func allSatisfy((Self.Element) async throws -> Bool) async rethrows -> Bool
```

Returns a Boolean value that indicates whether all elements produced by the asynchronous sequence satisfy the given predicate.

```
func first(where: (Self.Element) async throws -> Bool) async rethrows -> Self.Element?
```

Returns the first element of the sequence that satisfies the given predicate.

```
func min() async rethrows -> Self.Element?
```

Returns the minimum element in an asynchronous sequence of comparable elements.

```
func min(by: (Self.Element, Self.Element) async throws -> Bool) async rethrows -> Self.Element?
```

Returns the minimum element in the asynchronous sequence, using the given predicate as the comparison between elements.

```
func max() async rethrows -> Self.Element?
```

Returns the maximum element in an asynchronous sequence of comparable elements.

```
func max(by: (Self.Element, Self.Element) async throws -> Bool) async rethrows -> Self.Element?
```

Returns the maximum element in the asynchronous sequence, using the given predicate as the comparison between elements.

Selecting Elements

`func prefix(Int) -> AsyncPrefixSequence<Self>`

Returns an asynchronous sequence, up to the specified maximum length, containing the initial elements of the base asynchronous sequence.

`struct AsyncPrefixSequence`

An asynchronous sequence, up to a specified maximum length, containing the initial elements of a base asynchronous sequence.

`func prefix(while: (Self.Element) async -> Bool) rethrows -> AsyncPrefixWhileSequence<Self>`

Returns an asynchronous sequence, containing the initial, consecutive elements of the base sequence that satisfy the given predicate.

`struct AsyncPrefixWhileSequence`

An asynchronous sequence, containing the initial, consecutive elements of the base sequence that satisfy a given predicate.

`func prefix(while: (Self.Element) async throws -> Bool) rethrows -> AsyncThrowingPrefixWhileSequence<Self>`

Returns an asynchronous sequence, containing the initial, consecutive elements of the base sequence that satisfy the given error-throwing predicate.

`struct AsyncThrowingPrefixWhileSequence`

An asynchronous sequence, containing the initial, consecutive elements of the base sequence that satisfy the given error-throwing predicate.

Excluding Elements

`func dropFirst(Int) -> AsyncDropFirstSequence<Self>`

Omits a specified number of elements from the base asynchronous sequence, then passes through all remaining elements.

`struct AsyncDropFirstSequence`

An asynchronous sequence which omits a specified number of elements from the base asynchronous sequence, then passes through all remaining elements.

`func drop(while: (Self.Element) async -> Bool) -> AsyncDropWhileSequence<Self>`

Omits elements from the base asynchronous sequence until a given closure returns false, after which it passes through all remaining elements.

`struct AsyncDropWhileSequence`

An asynchronous sequence which omits elements from the base sequence until a given closure returns false, after which it passes through all remaining elements.

`func drop(while: (Self.Element) async throws -> Bool) -> AsyncThrowingDropWhileSequence<Self>`

Omits elements from the base sequence until a given error-throwing closure returns false, after which it passes through all remaining elements.

`struct AsyncThrowingDropWhileSequence`

An asynchronous sequence which omits elements from the base sequence until a given error-throwing closure returns false, after which it passes through all remaining elements.

`func filter((Self.Element) async -> Bool) -> AsyncFilterSequence<Self>`

Creates an asynchronous sequence that contains, in order, the elements of the base sequence that satisfy the given predicate.

`struct AsyncFilterSequence`

An asynchronous sequence that contains, in order, the elements of the base sequence that satisfy a given predicate.

`func filter((Self.Element) async throws -> Bool) -> AsyncThrowingFilterSequence<Self>`

Creates an asynchronous sequence that contains, in order, the elements of the base sequence that satisfy the given error-throwing predicate.

`struct AsyncThrowingFilterSequence`

An asynchronous sequence that contains, in order, the elements of the base sequence that satisfy the given error-throwing predicate.

Transforming a Sequence

`func map<Transformed>((Self.Element) async -> Transformed) -> AsyncMapSequence<Self, Transformed>`

Creates an asynchronous sequence that maps the given closure over the asynchronous sequence's elements.

`struct AsyncMapSequence`

An asynchronous sequence that maps the given closure over the asynchronous sequence's elements.

```
func map<Transformed>((Self.Element) async throws -> Transformed) -> AsyncThrowingMapSequence<Self, Transformed>
```

Creates an asynchronous sequence that maps the given error-throwing closure over the asynchronous sequence's elements.

```
struct AsyncThrowingMapSequence
```

An asynchronous sequence that maps the given error-throwing closure over the asynchronous sequence's elements.

```
func compactMap<ElementOfResult>((Self.Element) async -> ElementOfResult?) -> AsyncCompactMapSequence<Self, ElementOfResult>
```

Creates an asynchronous sequence that maps the given closure over the asynchronous sequence's elements, omitting results that don't return a value.

```
struct AsyncCompactMapSequence
```

An asynchronous sequence that maps a given closure over the asynchronous sequence's elements, omitting results that don't return a value.

```
func compactMap<ElementOfResult>((Self.Element) async throws -> ElementOfResult?) -> AsyncThrowingCompactMapSequence<Self, ElementOfResult>
```

Creates an asynchronous sequence that maps an error-throwing closure over the base sequence's elements, omitting results that don't return a value.

```
struct AsyncThrowingCompactMapSequence
```

An asynchronous sequence that maps an error-throwing closure over the base sequence's elements, omitting results that don't return a value.

```
struct AsyncFlatMapSequence
```

An asynchronous sequence that concatenates the results of calling a given transformation with each element of this sequence.

```
struct AsyncThrowingFlatMapSequence
```

An asynchronous sequence that concatenates the results of calling a given error-throwing transformation with each element of this sequence.

```
func reduce<Result>(Result, (Result, Self.Element) async throws -> Result) async rethrows -> Result
```

Returns the result of combining the elements of the asynchronous sequence using the given closure.

```
func reduce<Result>(into: Result, (inout Result, Self.Element) async throws -> Void) async rethrows -> Result
```

Returns the result of combining the elements of the asynchronous sequence using the given closure, given a mutable initial value.

Adapting Textual Sequences

```
var characters: AsyncCharacterSequence<Self>
```

A non-blocking sequence of `Characters` created by decoding the elements of `self` as UTF8.

```
struct AsyncCharacterSequence<Base> where Base : AsyncSequence, Base.Element == UInt8
```

An asynchronous sequence of characters.

```
var unicodeScalars: AsyncUnicodeScalarSequence<Self>
```

A non-blocking sequence of `UnicodeScalars` created by decoding the elements of `self` as UTF8.

```
struct AsyncUnicodeScalarSequence<Base> where Base : AsyncSequence, Base.Element == UInt8
```

An asychronous sequence of Unicode scalar values.

```
var lines: AsyncLineSequence<Self>
```

A non-blocking sequence of newline-separated `Strings` created by decoding the elements of `self` as UTF8.

```
struct AsyncLineSequence<Base> where Base : AsyncSequence, Base.Element == UInt8
```

An asynchronous sequence of lines of text.

Associated Types

```
associatedtype Failure = any Error
```

The type of errors produced when iteration over the sequence fails.

Required

Instance Methods

`func flatMap<SegmentOfResult>((Self.Element) async -> SegmentOfResult) -> AsyncFlatMapSequence<Self, SegmentOfResult>`

Creates an asynchronous sequence that concatenates the results of calling the given transformation with each element of this sequence.

`func flatMap<SegmentOfResult>((Self.Element) async -> SegmentOfResult) -> AsyncFlatMapSequence<Self, SegmentOfResult>`

Creates an asynchronous sequence that concatenates the results of calling the given transformation with each element of this sequence.

`func flatMap<SegmentOfResult>((Self.Element) async throws -> SegmentOfResult) -> AsyncThrowingFlatMapSequence<Self, SegmentOfResult>`

Creates an asynchronous sequence that concatenates the results of calling the given error-throwing transformation with each element of this sequence.

`func flatMap<SegmentOfResult>((Self.Element) async -> SegmentOfResult) -> AsyncFlatMapSequence<Self, SegmentOfResult>`

Creates an asynchronous sequence that concatenates the results of calling the given transformation with each element of this sequence.

Relationships

Conforming Types

`AsyncCompactMapSequence`

Conforms when `Base` conforms to `AsyncSequence`, `ElementOfResult` conforms to `Copyable`, and `ElementOfResult` conforms to `Escapable`.

`AsyncDropFirstSequence`

Conforms when `Base` conforms to `AsyncSequence`.

`AsyncDropWhileSequence`

Conforms when `Base` conforms to `AsyncSequence`.

`AsyncFilterSequence`

Conforms when `Base` conforms to `AsyncSequence`.

`AsyncFlatMapSequence`

Conforms when `Base` conforms to `AsyncSequence` and `SegmentOfResult` conforms to `AsyncSequence`.

`AsyncMapSequence`

Conforms when `Base` conforms to `AsyncSequence`, `Transformed` conforms to `Copyable`, and `Transformed` conforms to `Escapable`.

AsyncPrefixSequence

Conforms when `Base` conforms to `AsyncSequence`.

AsyncPrefixWhileSequence

Conforms when `Base` conforms to `AsyncSequence`.

AsyncStream

Conforms when `Element` conforms to `Copyable` and `Escapable`.

AsyncThrowingCompactMapSequence

Conforms when `Base` conforms to `AsyncSequence`, `ElementOfResult` conforms to `Copyable`, and `ElementOfResult` conforms to `Escapable`.

AsyncThrowingDropWhileSequence

Conforms when `Base` conforms to `AsyncSequence`.

AsyncThrowingFilterSequence

Conforms when `Base` conforms to `AsyncSequence`.

AsyncThrowingFlatMapSequence

Conforms when `Base` conforms to `AsyncSequence` and `SegmentOfResult` conforms to `AsyncSequence`.

AsyncThrowingMapSequence

Conforms when `Base` conforms to `AsyncSequence`, `Transformed` conforms to `Copyable`, and `Transformed` conforms to `Escapable`.

AsyncThrowingPrefixWhileSequence

Conforms when `Base` conforms to `AsyncSequence`.

AsyncThrowingStream

Conforms when `Element` conforms to `Copyable`, `Element` conforms to `Escapable`, and `Failure` conforms to `Error`.

Observations

TaskGroup

Conforms when `ChildTaskResult` conforms to `Copyable`, `Escapable`, and `Sendable`.

ThrowingTaskGroup

Conforms when `ChildTaskResult` conforms to `Copyable`, `ChildTaskResult` conforms to `Escapable`, `ChildTaskResult` conforms to `Sendable`, and `Failure` conforms to `Error`.

See Also

Asynchronous Sequences

`struct AsyncStream`

An asynchronous sequence generated from a closure that calls a continuation to produce new elements.

`struct` `AsyncThrowingStream`

An asynchronous sequence generated from an error-throwing closure that calls a continuation to produce new elements.