Search...

Links

License
Release Notes
Source Repository

Namespaces

FSharp.Collections
FSharp.Control
FSharp.Control.TaskBuilderExtension
FSharp.Core
FSharp.Core.CompilerServices
FSharp.Linq
FSharp.Linq.RuntimeHelpers
FSharp.NativeInterop
FSharp.Quotations
FSharp.Reflection

Option Module

Namespace: FSharp.Core
Assembly: FSharp.Core.dll

Contains operations for working with options.

Functions and values

Function or value | Description

Option.bind binder option

▼ `bind f inp` evaluates to `match inp with None -> None | Some x -> f x`

binder : 'T -> 'U option
A function that takes the value of type T from an option and transforms it into an option containing a value of type U.

option : 'T option
The input option.

Returns: 'U option
An option of the output type of the binder.

Example

```
let tryParse (input: string) =
    match System.Int32.TryParse input with
    | true, v -> Some v
    | false, _ -> None
None |> Option.bind tryParse // evaluates to None
Some "42" |> Option.bind tryParse // evaluates to Some 42
Some "Forty-two" |> Option.bind tryParse // evaluates to None
```

Option.contains value option

▼ Evaluates to true if **option** is Some and its value is equal to **value**.

value : 'T
The value to test for equality.

option : 'T option
The input option.

Returns: bool
True if the option is Some and contains a value equal to **value**, otherwise false.

Example

```
(99, None) ||> Option.contains // evaluates to false
(99, Some 99) ||> Option.contains // evaluates to true
(99, Some 100) ||> Option.contains // evaluates to false
```

Option.count option

▼ `count inp` evaluates to `match inp with None -> 0 | Some _ -> 1`.

option : 'T option
The input option.

Returns: int
A zero if the option is None, a one otherwise.

Example

```
None |> Option.count // evaluates to 0
Some 99 |> Option.count // evaluates to 1
```

Option.defaultValue value option

▼ Gets the value of the option if the option is Some, otherwise returns the specified default value.

Identical to the built-in defaultArg operator, except with the arguments swapped.

value : 'T
The specified default value.

option : 'T option
The input option.

Returns: 'T
The option if the option is Some, else the default value.

Example

```
(99, None) ||> Option.defaultValue // evaluates to 99
(99, Some 42) ||> Option.defaultValue // evaluates to 42
```

Option.defaultWith defThunk option

▼ Gets the value of the option if the option is Some, otherwise evaluates **defThunk** and returns the result.

defThunk is not evaluated unless **option** is None.

defThunk : unit -> 'T
A thunk that provides a default value when evaluated.

option : 'T option
The input option.

Returns: 'T
The option if the option is Some, else the result of evaluating **defThunk**.

Example

```
None |> Option.defaultWith (fun () -> 99) // evaluates to 99
Some 42 |> Option.defaultWith (fun () -> 99) // evaluates to 42
```

Option.exists predicate option

▼ `exists p inp` evaluates to `match inp with None -> false | Some x -> p x`.

predicate : 'T -> bool
A function that evaluates to a boolean when given a value from the option type.

option : 'T option
The input option.

Returns: bool
False if the option is None, otherwise it returns the result of applying the predicate to the option value.

Example

```
None |> Option.exists (fun x -> x >= 5) // evaluates to false
Some 42 |> Option.exists (fun x -> x >= 5) // evaluates to true
Some 4 |> Option.exists (fun x -> x >= 5) // evaluates to false
```

Option.filter predicate option

▼ `filter f inp` evaluates to `match inp with None -> None | Some x -> if f x then Some x else None`.

predicate : 'T -> bool
A function that evaluates whether the value contained in the option should remain, or be filtered out.

option : 'T option
The input option.

Returns: 'T option
The input if the predicate evaluates to true; otherwise, None.

Example

```
None |> Option.filter (fun x -> x >= 5) // evaluates to None
Some 42 |> Option.filter (fun x -> x >= 5) // evaluates to Some 42
Some 4 |> Option.filter (fun x -> x >= 5) // evaluates to None
```

Option.flatten option

▼ `flatten inp` evaluates to `match inp with None -> None | Some x -> x`

`flatten` is equivalent to `bind id`.

option : 'T option option
The input option.

Returns: 'T option
The input value if the value is Some; otherwise, None.

Example

```
(None: int option option) |> Option.flatten // evaluates to None
(Some ((None: int option))) |> Option.flatten // evaluates to None
(Some (Some 42)) |> Option.flatten // evaluates to Some 42
```

Option.fold folder state option

▼ `fold f s inp` evaluates to `match inp with None -> s | Some x -> f s x`.

folder : 'State -> 'T -> 'State
A function to update the state data when given a value from an option.

state : 'State
The initial state.

option : 'T option
The input option.

Returns: 'State
The original state if the option is None, otherwise it returns the updated state with the folder and the option value.

Example

```
(0, None) ||> Option.fold (fun accum x -> accum + x * 2) // evaluates to 0
(0, Some 1) ||> Option.fold (fun accum x -> accum + x * 2) // evaluates to 2
(10, Some 1) ||> Option.fold (fun accum x -> accum + x * 2) // evaluates to 12
```

Option.foldBack folder option state

▼ `fold f inp s` evaluates to `match inp with None -> s | Some x -> f x s`.

folder : 'T -> 'State -> 'State
A function to update the state data when given a value from an option.

option : 'T option
The input option.

state : 'State
The initial state.

Returns: 'State
The original state if the option is None, otherwise it returns the updated state with the folder and the option value.

Example

```
(None, 0) ||> Option.foldBack (fun x accum -> accum + x * 2) // evaluates to 0
(Some 1, 0) ||> Option.foldBack (fun x accum -> accum + x * 2) // evaluates to 2
(Some 1, 10) ||> Option.foldBack (fun x accum -> accum + x * 2) // evaluates to 12
```

Option.forall predicate option

▼ `forall p inp` evaluates to `match inp with None -> true | Some x -> p x`.

predicate : 'T -> bool
A function that evaluates to a boolean when given a value from the option type.

option : 'T option
The input option.

Returns: bool
True if the option is None, otherwise it returns the result of applying the predicate to the option value.

Example

```
None |> Option.forall (fun x -> x >= 5) // evaluates to true
Some 42 |> Option.forall (fun x -> x >= 5) // evaluates to true
Some 4 |> Option.forall (fun x -> x >= 5) // evaluates to false
```

Option.get option

▼ Gets the value associated with the option.

option : 'T option
The input option.

Returns: 'T
The value within the option.

Example

```
Some 42 |> Option.get // evaluates to 42
(None: int option) |> Option.get // throws exception!
```

Option.isNone option

▼ Returns true if the option is None.

option : 'T option
The input option.

Returns: bool
True if the option is None.

Example

```
None |> Option.isNone // evaluates to true
Some 42 |> Option.isNone // evaluates to false
```

Option.isSome option

▼ Returns true if the option is not None.

option : 'T option
The input option.

Returns: bool
True if the option is not None.

Example

```
None |> Option.isSome // evaluates to false
Some 42 |> Option.isSome // evaluates to true
```

Option.iter action option

▼ `iter f inp` executes `match inp with None -> () | Some x -> f x`.

action : 'T -> unit
A function to apply to the option value.

option : 'T option
The input option.

Example

```
None |> Option.iter (printfn "%s") // does nothing
Some "Hello world" |> Option.iter (printfn "%s") // prints "Hello world"
```

Option.map mapping option

▼ `map f inp` evaluates to `match inp with None -> None | Some x -> Some (f x)`.

mapping : 'T -> 'U
A function to apply to the option value.

option : 'T option
The input option.

Returns: 'U option
An option of the input value after applying the mapping function, or None if the input is None.

Example

```
None |> Option.map (fun x -> x * 2) // evaluates to None
Some 42 |> Option.map (fun x -> x * 2) // evaluates to Some 84
```

Option.map2 mapping option1 option2

▼ `map f option1 option2` evaluates to `match option1, option2 with Some x, Some y -> Some (f x y) | _ -> None`.

mapping : 'T1 -> 'T2 -> 'U
A function to apply to the option values.

option1 : 'T1 option
The first option.

option2 : 'T2 option
The second option.

Returns: 'U option
An option of the input values after applying the mapping function, or None if either input is None.

Example

```
(None, None) ||> Option.map2 (fun x y -> x + y) // evaluates to None
(Some 5, None) ||> Option.map2 (fun x y -> x + y) // evaluates to None
(None, Some 10) ||> Option.map2 (fun x y -> x + y) // evaluates to None
(Some 5, Some 10) ||> Option.map2 (fun x y -> x + y) // evaluates to Some 15
```

Option.map3 mapping option1 option2 option3

▼ `map f option1 option2 option3` evaluates to `match option1, option2, option3 with Some x, Some y, Some z -> Some (f x y z) | _ -> None`.

mapping : 'T1 -> 'T2 -> 'T3 -> 'U
A function to apply to the option values.

option1 : 'T1 option
The first option.

option2 : 'T2 option
The second option.

option3 : 'T3 option
The third option.

Returns: 'U option
An option of the input values after applying the mapping function, or None if any input is None.

Example

```
(None, None, None) |||> Option.map3 (fun x y z -> x + y + z) // evaluates to None
(Some 100, None, None) |||> Option.map3 (fun x y z -> x + y + z) // evaluates to None
(None, Some 100, None) |||> Option.map3 (fun x y z -> x + y + z) // evaluates to None
(None, None, Some 100) |||> Option.map3 (fun x y z -> x + y + z) // evaluates to None
(Some 5, Some 100, Some 10) |||> Option.map3 (fun x y z -> x + y + z) // evaluates to Some 115
```

Option.ofNullable value

▼ Convert a Nullable value to an option.

value : Nullable<'T>
The input nullable value.

Returns: 'T option
The result option.

Example

```
System.Nullable<int>() |> Option.ofNullable // evaluates to None
System.Nullable(42) |> Option.ofNullable // evaluates to Some 42
```

Option.ofObj value

▼ Convert a potentially null value to an option.

value : 'T
The input value.

Returns: 'T option
The result option.

Example

```
(null: string) |> Option.ofObj // evaluates to None
"not a null string" |> Option.ofObj // evaluates to (Some "not a null string")
```

Option.orElse ifNone option

▼ Returns **option** if it is Some, otherwise returns **ifNone**.

ifNone : 'T option
The value to use if **option** is None.

option : 'T option
The input option.

Returns: 'T option
The option if the option is Some, else the alternate option.

Example

```
((None: int Option), None) ||> Option.orElse // evaluates to None
(Some 99, None) ||> Option.orElse // evaluates to Some 99
(None, Some 42) ||> Option.orElse // evaluates to Some 42
(Some 99, Some 42) ||> Option.orElse // evaluates to Some 42
```

Option.orElseWith ifNoneThunk option

▼ Returns **option** if it is Some, otherwise evaluates **ifNoneThunk** and returns the result.

ifNoneThunk is not evaluated unless **option** is None.

ifNoneThunk : unit -> 'T option
A thunk that provides an alternate option when evaluated.

option : 'T option
The input option.

Returns: 'T option
The option if the option is Some, else the result of evaluating **ifNoneThunk**.

Example

```
(None: int Option) |> Option.orElseWith (fun () -> None) // evaluates to None
None |> Option.orElseWith (fun () -> (Some 99)) // evaluates to Some 99
Some 42 |> Option.orElseWith (fun () -> None) // evaluates to Some 42
Some 42 |> Option.orElseWith (fun () -> (Some 99)) // evaluates to Some 42
```

Option.toArray option

▼ Convert the option to an array of length 0 or 1.

option : 'T option
The input option.

Returns: 'T[]
The result array.

Example

```
(None: int option) |> Option.toArray // evaluates to [||]
Some 42 |> Option.toArray // evaluates to [|42|]
```

Option.toList option

▼ Convert the option to a list of length 0 or 1.

option : 'T option
The input option.

Returns: 'T list
The result list.

Example

```
(None: int option) |> Option.toList // evaluates to []
Some 42 |> Option.toList // evaluates to [42]
```

Option.toNullable option

▼ Convert the option to a Nullable value.

option : 'T option
The input option.

Returns: Nullable<'T>
The result value.

Example

```
(None: int option) |> Option.toNullable // evaluates to new System.Nullable<int>()
Some 42 |> Option.toNullable // evaluates to new System.Nullable(42)
```

Option.toObj value

▼ Convert an option to a potentially null value.

value : 'T option
The input value.

Returns: 'T
The result value, which is null if the input was None.

Example

```
(None: string option) |> Option.toObj // evaluates to null
Some "not a null string" |> Option.toObj // evaluates to "not a null string"
```