

[Previous Up](#)

# Module type Set.S

```
module type S = sig .. end
```

Output signature of the functor [Set.Make](#).

---

```
type elt
```

The type of the set elements.

```
type t
```

The type of sets.

```
val empty : t
```

The empty set.

```
val is_empty : t -> bool
```

Test whether a set is empty or not.

```
val mem : elt -> t -> bool
```

`mem x s` tests whether `x` belongs to the set `s`.

```
val add : elt -> t -> t
```

`add x s` returns a set containing all elements of `s`, plus `x`. If `x` was already in `s`, `s` is returned unchanged (the result of the function is then physically equal to `s`).

**Before 4.03** Physical equality was not ensured.

```
val singleton : elt -> t
```

`singleton x` returns the one-element set containing only `x`.

```
val remove : elt -> t -> t
```

`remove x s` returns a set containing all elements of `s`, except `x`. If `x` was not in `s`, `s` is returned unchanged (the result of the function is then physically equal to `s`).

**Before 4.03** Physical equality was not ensured.

```
val union : t -> t -> t
```

Set union.

```
val inter : t -> t -> t
```

Set intersection.

```
val diff : t -> t -> t
```

Set difference.

```
val compare : t -> t -> int
```

Total ordering between sets. Can be used as the ordering function for doing sets of sets.

```
val equal : t -> t -> bool
```

`equal s1 s2` tests whether the sets `s1` and `s2` are equal, that is, contain equal elements.

```
val subset : t -> t -> bool
```

`subset s1 s2` tests whether the set `s1` is a subset of the set `s2`.

```
val iter : (elt -> unit) -> t -> unit
```

`iter f s` applies `f` in turn to all elements of `s`. The elements of `s` are presented to `f` in increasing order with respect to the ordering over the type of the elements.

**val** map : (elt -> elt) -> t -> t

map f s is the set whose elements are f a<sub>0</sub>, f a<sub>1</sub>... f a<sub>N</sub>, where a<sub>0</sub>, a<sub>1</sub>... a<sub>N</sub> are the elements of s.

The elements are passed to f in increasing order with respect to the ordering over the type of the elements.

If no element of s is changed by f, s is returned unchanged. (If each output of f is physically equal to its input, the returned set is physically equal to s.)

**Since** 4.04.0

**val** fold : (elt -> 'a -> 'a) -> t -> 'a -> 'a

fold f s a computes (f x<sub>N</sub> ... (f x<sub>2</sub> (f x<sub>1</sub> a))...), where x<sub>1</sub> ... x<sub>N</sub> are the elements of s, in increasing order.

**val** for\_all : (elt -> bool) -> t -> bool

for\_all p s checks if all elements of the set satisfy the predicate p.

**val** exists : (elt -> bool) -> t -> bool

exists p s checks if at least one element of the set satisfies the predicate p.

**val** filter : (elt -> bool) -> t -> t

filter p s returns the set of all elements in s that satisfy predicate p. If p satisfies every element in s, s is returned unchanged (the result of the function is then physically equal to s).

**Before 4.03** Physical equality was not ensured.

**val** partition : (elt -> bool) -> t -> t \* t

partition p s returns a pair of sets (s<sub>1</sub>, s<sub>2</sub>), where s<sub>1</sub> is the set of all the elements of s that satisfy the predicate p, and s<sub>2</sub> is the set of all the elements of s that do not satisfy p.

**val** cardinal : t -> int

Return the number of elements of a set.

**val** elements : t -> elt list

Return the list of all elements of the given set. The returned list is sorted in increasing order with respect to the ordering Ord.compare, where Ord is the argument given to Set.Make.

**val** min\_elt : t -> elt

Return the smallest element of the given set (with respect to the Ord.compare ordering), or raise Not\_found if the set is empty.

**val** min\_elt\_opt : t -> elt option

Return the smallest element of the given set (with respect to the Ord.compare ordering), or None if the set is empty.

**Since** 4.05

**val** max\_elt : t -> elt

Same as Set.S.min\_elt, but returns the largest element of the given set.

**val** max\_elt\_opt : t -> elt option

Same as Set.S.min\_elt\_opt, but returns the largest element of the given set.

**Since** 4.05

**val** choose : t -> elt

Return one element of the given set, or raise Not\_found if the set is empty. Which element is chosen is unspecified, but equal elements will be chosen for equal sets.

**val** choose\_opt : t -> elt option

Return one element of the given set, or `None` if the set is empty. Which element is chosen is unspecified, but equal elements will be chosen for equal sets.

**Since** 4.05

**val** split : elt -> t -> t \* bool \* t

`split x s` returns a triple `(l, present, r)`, where `l` is the set of elements of `s` that are strictly less than `x`; `r` is the set of elements of `s` that are strictly greater than `x`; `present` is `false` if `s` contains no element equal to `x`, or `true` if `s` contains an element equal to `x`.

**val** find : elt -> t -> elt

`find x s` returns the element of `s` equal to `x` (according to `Ord.compare`), or raise `Not_found` if no such element exists.

**Since** 4.01.0

**val** find\_opt : elt -> t -> elt option

`find_opt x s` returns the element of `s` equal to `x` (according to `Ord.compare`), or `None` if no such element exists.

**Since** 4.05

**val** find\_first : (elt -> bool) -> t -> elt

`find_first f s`, where `f` is a monotonically increasing function, returns the lowest element `e` of `s` such that `f e`, or raises `Not_found` if no such element exists.

For example, `find_first (fun e -> Ord.compare e x >= 0) s` will return the first element `e` of `s` where `Ord.compare e x >= 0` (intuitively: `e >= x`), or raise `Not_found` if `x` is greater than any element of `s`.

**Since** 4.05

**val** find\_first\_opt : (elt -> bool) -> t -> elt option

`find_first_opt f s`, where `f` is a monotonically increasing function, returns an option containing the lowest element `e` of `s` such that `f e`, or `None` if no such element exists.

**Since** 4.05

**val** find\_last : (elt -> bool) -> t -> elt

`find_last f s`, where `f` is a monotonically decreasing function, returns the highest element `e` of `s` such that `f e`, or raises `Not_found` if no such element exists.

**Since** 4.05

**val** find\_last\_opt : (elt -> bool) -> t -> elt option

`find_last_opt f s`, where `f` is a monotonically decreasing function, returns an option containing the highest element `e` of `s` such that `f e`, or `None` if no such element exists.

**Since** 4.05

**val** of\_list : elt list -> t

`of_list l` creates a set from a list of elements. This is usually more efficient than folding `add` over the list, except perhaps for lists with many duplicated elements.

**Since** 4.02.0

Iterators

**val** to\_seq\_from : elt -> t -> elt Seq.t

`to_seq_from x s` iterates on a subset of the elements of `s` in ascending order, from `x` or above.

**Since 4.07****val** to\_seq : t -> elt Seq.t

Iterate on the whole set, in ascending order

**Since 4.07****val** add\_seq : elt Seq.t -> t -> t

Add the given elements to the set, in order.

**Since 4.07****val** of\_seq : elt Seq.t -> t

Build a set from the given bindings

**Since 4.07**