API Reference

Complete documentation of all Result.js methods and functions.

Creation

Result.ok()

Creates a successful Result containing a value.

```
Result.ok<T, E>(value: T): Ok<T, E>
```

Example:

```
const result = Result.ok(42)
const user = Result.ok({ id: 1, name: 'John' })
```

Result.err()

Creates a failed Result containing an error.

```
Result.err<T, E>(error: E): Err<T, E>
```

Example:

```
const result = Result.err(new Error('failed'))
const apiError = Result.err('User not found')
```

Result.fromTry()

Wraps synchronous function execution, capturing thrown exceptions.

```
Result.fromTry<T>(executor: () => T): Result<T, Error>
Result.fromTry<T, E>(executor: () => T, onError: (error: unknown) => E): Result<T, E>
```

Example:

```
const parsed = Result.fromTry(() => JSON.parse(input))

const custom = Result.fromTry(
  () => doSomething(),
  (err) => new CustomError(`Failed: ${err}`)
)
```

Result.fromPromise()

Wraps async function execution, capturing rejections.

```
async function fromPromise<T>(executor: () => Promise<T>): AsyncResult<T, Error>
async function fromPromise<T, E>(executor: () => Promise<T>, onError: (error: unknown) => E): AsyncResult<T, E>
```

Example:

```
const data = await Result.fromPromise(
  () => fetch('/api/data').then(r => r.json())
)

const withError = await Result.fromPromise(
  () => fetchUser(id),
  (err) => ({ code: 500, message: String(err) })
)
```

Result.fromNullable()

Converts null/undefined values to Err.

```
Result.fromNullable<T>(value: T | null | undefined): Result<NonNullable<T>, Error>
Result.fromNullable<T, E>(value: T | null | undefined, onError: () => E): Result<NonNullable<T>, E>
```

Example:

```
const user = Result.fromNullable(users.find(u => u.id === 1))

const config = Result.fromNullable(
  process.env.API_KEY,
  () => new Error('API_KEY not set')
)
```

Result.validate()

Validates a value with a predicate function.

```
Result.validate<T>(value: T, predicate: (value: T) => boolean): Result<T, Error>
Result.validate<T, E>(value: T, predicate: (value: T) => boolean, onError: (value: T) => E): Result<T, E>
```

Example:

```
const age = Result.validate(25, x => x >= 18)

const typed = Result.validate(
  -5,
  x => x > 0,
  x => ({ field: 'age', message: 'Must be positive' })
)
```

Validation

isOk()

Checks if Result is Ok variant.

```
isOk(): boolean
```

Example:

```
if (result.isOk()) {
  const value = result.unwrap()
}
```

isErr()

Checks if Result is Err variant.

```
isErr(): boolean
```

Example:

```
if (result.isErr()) {
  const error = result.unwrapErr()
}
```

isOkAnd()

Checks if Ok and value satisfies predicate.

```
isOkAnd(predicate: (value: T) => boolean): boolean
```

Example:

```
if (result.isOkAnd(user => user.isActive)) {
  // User exists AND is active
}
```

isErrAnd()

Checks if Err and error satisfies predicate.

```
isErrAnd(predicate: (error: E) => boolean): boolean
```

Example:

```
if (result.isErrAnd(e => e.code === 404)) {
  showNotFound()
}
```

Result.isResult()

Type guard: checks if value is a Result instance.

```
Result.isResult(value: unknown): value is Result<unknown, unknown>
```

Example:

```
if (Result.isResult(value)) {
  return value.isOk() ? value.unwrap() : null
}
```

Access & Extraction

.ok

Safe property access to success value.

```
result.ok: T | null
```

Example:

```
const value = result.ok  // null if Err
if (value !== null) {
  processValue(value)
}
```

.err

Safe property access to error.

```
result.err: E | null
```

Example:

```
const error = result.err  // null if Ok
if (error !== null) {
  logError(error)
}
```

unwrap()

Extracts value (throws if Err).

```
unwrap(): T
```

Example:

```
if (result.isOk()) {
  const value = result.unwrap()  // Safe
}
```

unwrapErr()

Extracts error (throws if Ok).

```
unwrapErr(): E
```

Example:

```
if (result.isErr()) {
  const error = result.unwrapErr()
}
```

expect()

Extracts value with custom error message (throws if Err).

```
expect(message: string): T
```

Example:

```
const value = result.expect('User must exist')
```

expectErr()

Extracts error with custom error message (throws if Ok).

```
expectErr(message: string): E
```

Example:

```
const error = result.expectErr('Operation should fail')
```

unwrapOr()

Extracts value or returns default.

```
unwrapOr(defaultValue: T): T
```

Example:

```
const port = getPort().unwrapOr(3000)
const timeout = getTimeout().unwrapOr(5000)
```

unwrapOrElse()

Extracts value or computes default from error.

```
unwrapOrElse(onError: (error: E) => T): T
```

Example:

```
const value = result.unwrapOrElse(error => {
  console.error('Error:', error)
  return defaultValue
})
```

Transformation

map()

Transforms success value.

```
map<U, F = never>(mapper: (value: T) => U | Result<U, F>): Result<U, E>
```

Example:

```
Result.ok(5)
  .map(x => x * 2)        // Ok(10)
  .map(x => x + 5)        // Ok(15)
```

mapOr()

Transforms value or returns default.

```
mapOr<U>(mapper: (value: T) => U, defaultValue: U): U
```

Example:

```
const result = value.mapOr(x => x * 2, 0)
// Returns transformed value if Ok, default if Err
```

mapOrElse()

Transforms using appropriate mapper based on state.

```
mapOrElse<U>(okMapper: (value: T) => U, errorMapper: (error: E) => U): U
```

Example:

```
result.mapOrElse(
  user => user.name,
  error => 'Anonymous'
)
```

mapErr()

Transforms error.

```
mapErr<F>(mapper: (error: E) => F): Result<T, F>
```

Example:

```
result.mapErr(e => new Error(`API Error: ${e}`))
```

filter()

Filters Ok value with predicate.

```
filter(predicate: (value: T) => boolean): Result<T, Error>
filter(predicate: (value: T) => boolean, onReject: (value: T) => E): Result<T, E>
```

Example:

```
Result.ok(10)
  .filter(x => x > 5)  // Ok(10)

Result.ok(3)
  .filter(x => x > 5, x => new Error(`${x} too small`))  // Err
```

flatten()

Flattens nested Result.

```
flatten<U, F>(this: Result<Result<U, F>, E>): Result<U, E | F>
```

Example:

```
Result.ok(Result.ok(42)).flatten()  // Ok(42)
Result.ok(Result.err('fail')).flatten()  // Err('fail')
```

Chaining

andThen()

Chains operation returning Result (synchronous).

```
andThen<U>(flatMapper: (value: T) => Result<U, E>): Result<U, E>
```

Example:

```
Result.ok(100)
  .andThen(x => divide(x, 2))
  .andThen(x => divide(x, 5))
```

orElse()

Executes recovery function on error.

```
orElse(onError: (error: E) => Result<T, E>): Result<T, E>
```

Example:

```
fetchFromCache(key)
  .orElse(() => fetchFromDatabase(key))
  .orElse(() => fetchFromAPI(key))
```

and()

Returns second Result if this is Ok.

```
and<U>(result: Result<U, E>): Result<U, E>
```

Example:

```
Result.ok(1).and(Result.ok(2))  // Ok(2)
Result.ok(1).and(Result.err('fail'))  // Err('fail')
```

or()

Returns this Result or alternative.

```
or(result: Result<T, E>): Result<T, E>
```

Example:

```
Result.err('fail').or(Result.ok(42))  // Ok(42)
Result.ok(1).or(Result.ok(2))  // Ok(1)
```

zip()

Combines two Results into tuple.

```
zip<U, F>(result: Result<U, F>): Result<[T, U], E | F>
```

Example:

```
const id = Result.ok(1)
const name = Result.ok('John')
id.zip(name)  // Ok([1, 'John'])
```

Inspection

match()

Pattern matching on Result state.

```
match<L, R>(handlers: { ok: (value: T) => L; err: (error: E) => R }): L | R
```

Example:

```
result.match({
  ok: (value) => `Success: ${value}`,
  err: (error) => `Error: ${error}`
})
```

inspect()

Performs side effect on success value.

```
inspect(visitor: (value: T) => void): Result<T, E>
```

Example:

```
result
  .inspect(x => console.log('Value:', x))
  .andThen(validate)
  .inspect(x => console.log('Valid:', x))
```

inspectErr()

Performs side effect on error.

```
inspectErr(visitor: (error: E) => void): Result<T, E>
```

Example:

```
result
  .inspectErr(e => logger.error('Failed:', e))
  .orElse(() => Result.ok(defaultValue))
```

contains()

Checks if Ok contains specific value.

```
contains(value: T, comparator?: (actual: T, expected: T) => boolean): boolean
```

Example:

```
Result.ok(42).contains(42)  // true

Result.ok({ id: 1 }).contains(
  { id: 1 },
  (a, b) => a.id === b.id
)  // true
```

containsErr()

Checks if Err contains specific error.

```
containsErr(error: E, comparator?: (actual: E, expected: E) => boolean): boolean
```

Example:

```
Result.err('fail').containsErr('fail')  // true
```

Collections

Result.all()

Combines multiple Results (fail-fast).

```
Result.all<T extends readonly Result<unknown, unknown>[]>(results: T): Result<OkTuple<T>, ErrUnion<T>>
```

Example:

```
const result = Result.all([
  Result.ok(1),
  Result.ok('two'),
  Result.ok(true)
])
result.unwrap()  // [1, "two", true]

// With error - returns first error
Result.all([Result.ok(1), Result.err('fail')])  // Err('fail')
```

Result.allSettled()

Collects all Results without failing.

```
Result.allSettled<T extends readonly Result<unknown, unknown>[]>(results: T): Ok<SettledResult<OkUnion<T>, ErrUnion<T>>[]>
```

Example:

```
const results = Result.allSettled([
  Result.ok(1),
  Result.err('failed'),
  Result.ok(3)
]).unwrap()

// [
//   { status: 'ok', value: 1 },
//   { status: 'err', reason: 'failed' },
//   { status: 'ok', value: 3 }
// ]

const successes = results.filter(r => r.status === 'ok')
const failures = results.filter(r => r.status === 'err')
```

Result.any()

Returns first Ok or all errors.

```
Result.any<T extends readonly Result<unknown, unknown>[]>(results: T): Result<OkUnion<T>, ErrTuple<T>>
```

Example:

```
// Returns first Ok
Result.any([Result.err('e1'), Result.ok(42)])  // Ok(42)

// All Err - returns array of errors
Result.any([Result.err('e1'), Result.err('e2')])  // Err(['e1', 'e2'])
```

Result.partition()

Separates Results into successes and failures.

```
Result.partition<T, E>(results: readonly Result<T, E>[]): [T[], E[]]
```

Example:

```
const operations = [
  Result.ok(1),
  Result.err('failure A'),
  Result.ok(2)
]

const [successes, errors] = Result.partition(operations)
// successes: [1, 2]
// errors: ['failure A']
```

Result.values()

Extracts only success values.

```
Result.values<T, E>(results: readonly Result<T, E>[]): T[]
```

Example:

```
const items = [Result.ok(1), Result.err('fail'), Result.ok(2)]
Result.values(items)  // [1, 2]
```

Result.errors()

Extracts only errors.

```
Result.errors<T, E>(results: readonly Result<T, E>[]): E[]
```

Example:

```
const items = [Result.ok(1), Result.err('fail A'), Result.err('fail B')]
Result.errors(items)  // ['fail A', 'fail B']
```

Conversion

toPromise()

Converts Result to Promise.

```
toPromise(): Promise<T>
```

Example:

```
const value = await Result.ok(42).toPromise()  // 42

try {
  await Result.err('fail').toPromise()
} catch (e) {
  console.log(e)  // 'fail'
}
```

toString()

Converts to string representation.

```
toString(): string
```

Example:

```
Result.ok(42).toString()     // "Ok(42)"
Result.err('fail').toString()  // "Err(fail)"
```

toJSON()

Converts to JSON object.

```
toJSON(): { type: 'ok'; value: T } | { type: 'err'; error: E }
```

Example:

```
Result.ok(42).toJSON()        // { type: 'ok', value: 42 }
Result.err('fail').toJSON()   // { type: 'err', error: 'fail' }

JSON.stringify(Result.ok(42)) // '{"type":"ok","value":42}'
```

Async Operations

All sync methods have async versions. Use `Async` suffix: `mapAsync`, `andThenAsync`, etc.

mapAsync()

Transforms value asynchronously.

```
async mapAsync<U, F = never>(mapperAsync: (value: T) => Promise<U | Result<U, F>>): Promise<Result<U, E>>
```

Example:

```
await Result.ok(userId)
  .mapAsync(async (id) => await fetchUser(id))
```

mapErrAsync()

Transforms error asynchronously.

```
async mapErrAsync<F>(mapperAsync: (error: E) => Promise<F>): Promise<Result<T, F>>
```

Example:

```
await result.mapErrAsync(async (error) => ({
  ...error,
  context: await fetchContext(),
  timestamp: Date.now()
}))
```

mapOrAsync()

Transforms value or returns default (async).

```
async mapOrAsync<U>(mapperAsync: (value: T) => Promise<U>, defaultValue: U): Promise<U>
```

mapOrElseAsync()

Transforms using appropriate async mapper.

```
async mapOrElseAsync<U>(okMapperAsync: (value: T) => Promise<U>, errMapperAsync: (error: E) => Promise<U>): Promise<U>
```

andThenAsync()

Chains async operation returning Result.

```
async andThenAsync<U>(flatMapperAsync: (value: T) => Promise<Result<U, E>>): Promise<Result<U, E>>
```

Example:

```
await Result.ok(userId)
  .andThenAsync(async (id) => {
    const user = await fetchUser(id)
    return user ? Result.ok(user) : Result.err('not found')
  })
```

andAsync()

Returns async Result if this is Ok.

```
async andAsync<U>(result: Promise<Result<U, E>>): Promise<Result<U, E>>
```

orAsync()

Returns this Result or async alternative.

```
async orAsync(result: Promise<Result<T, E>>): Promise<Result<T, E>>
```

orElseAsync()

Executes async recovery function on error.

```
async orElseAsync(onErrorAsync: (error: E) => Promise<Result<T, E>>): Promise<Result<T, E>>
```

Example:

```
await result
  .orElseAsync(async () => fetchFromCache())
  .then(r => r.orElseAsync(async () => fetchFromAPI()))
```

Type Definitions

Result<T, E>

Represents a Result that can be Ok or Err.

```
type Result<T, E> = Ok<T, E> | Err<T, E>
```

AsyncResult<T, E>

Represents a Promise that resolves to a Result.

```
type AsyncResult<T, E> = Promise<Result<T, E>>
```

Ok<T, E>

Success variant containing a value.

```
class Ok<T, E = never> { ... }
```

Err<T, E>

Error variant containing an error.

```
class Err<T = never, E = Error> { ... }
```

Quick Reference

Use Case	Method	Purpose
Creating	`Result.ok(value)`	Create success
	`Result.err(error)`	Create failure
	`Result.fromTry(fn)`	Wrap try-catch
	`Result.fromPromise(fn)`	Wrap Promise
	`Result.fromNullable(val)`	Handle null/undefined
	`Result.validate(val, pred)`	Validate with predicate
Checking	`result.isOk()`	Is Ok?
	`result.isErr()`	Is Err?
	`result.isOkAnd(pred)`	Is Ok AND predicate?
	`result.isErrAnd(pred)`	Is Err AND predicate?
	`Result.isResult(result)`	Is Result?
Extracting	`result.ok`	Safe value access (null if Err)
	`result.err`	Safe error access (null if Ok)
	`result.unwrap()`	Get value (throws if Err)
	`result.unwrapErr()`	Get error (throws if Ok)
	`result.unwrapOr(def)`	Get value or default
	`result.unwrapOrElse(onError)`	Get value or default
	`result.expect(msg)`	Get value (throws with message if Err)
	`result.expectErr(msg)`	Get error (throws with message if Ok)
Transforming	`result.map(fn)`	Transform value
	`result.mapOr(def, fn)`	Transform value or default
	`result.mapOrElse(onError, fn)`	Transform value or default
	`result.mapErr(fn)`	Transform error

Use Case	Method	Purpose
	`result.filter(pred)`	Filter value
	`result.flatten()`	Flatten nested Result
Chaining	`result.andThen(fn)`	Chain Results
	`result.orElse(fn)`	Error recovery
	`result.and(other)`	Sequence Results
	`result.or(other)`	Alternative Result
	`result.zip(other)`	Combine Results
Inspection	`result.match({ok, err})`	Pattern matching
	`result.inspect(fn)`	Inspect value
	`result.inspectErr(fn)`	Inspect error
	`result.contains(value)`	Check value
	`result.containsErr(error)`	Check error
Collections	`Result.all([...])`	Fail-fast combine
	`Result.allSettled([...])`	Combine all
	`Result.any([...])`	First Ok
	`Result.partition([...])`	Separate Ok/Err
	`Result.values([...])`	Extract Ok values
	`Result.errors([...])`	Extract errors
Conversion	`result.toPromise()`	Convert to Promise
	`result.toString()`	Convert to string
	`result.toJSON()`	Convert to JSON
Async	`result.mapAsync(fn)`	Async transform
	`result.mapErrAsync(fn)`	Async transform error
	`result.mapOrAsync(def, fn)`	Async transform or default
	`result.mapOrElseAsync(onError, fn)`	Async transform or default
	`result.andThenAsync(fn)`	Async chain
	`result.andAsync(other)`	Async sequence
	`result.orAsync(other)`	Async alternative
	`result.orElseAsync(fn)`	Async recovery