API Reference

Complete API documentation organized by use case and functionality.

Quick Navigation

Use the sidebar to jump to specific modules, or browse by common tasks below.

Quick Reference by Use Case

🌐 Networking & Connections

Task	Module	Method
Find peers by topic	Hyperswarm	`swarm.join(topic)`
Connect to specific peer	Hyperswarm	`swarm.joinPeer(publicKey)`
Direct P2P connection	HyperDHT	`dht.connect(publicKey)`
Create a server	HyperDHT	`dht.createServer()`

📊 Data Storage

Task	Module	Method
Append-only log	Hypercore	`core.append(data)`
Key-value database	Hyperbee	`db.put(key, value)`

Task	Module	Method
File system	Hyperdrive	`drive.put(path, data)`
Multiple writers	Autobase	`base.append(data)`

🔄 Data Operations

Task	Module	Method
Read data	Hypercore/Hyperbee/Hyperdrive	`.get(index/key/path)`
Stream data	Hypercore/Hyperdrive	`.createReadStream()`
Query database	Hyperbee	`db.createReadStream({ range })`
Clear local data	Hypercore	`core.clear(start, end)`

🔐 Cryptography

Task	Module	Function
Generate keypair	hypercore-crypto	`crypto.keyPair()`
Random bytes	hypercore-crypto	`crypto.randomBytes(32)`
Derive keys	hypercore-crypto	`crypto.derive(namespace, key)`
Sign data	hypercore-crypto	`crypto.sign(message, secretKey)`

Core Modules

Hyperswarm

High-level API for peer discovery and connections.

Installation

```bash
npm install hyperswarm
```

Basic Usage

```javascript
import Hyperswarm from 'hyperswarm'

const swarm = new Hyperswarm()
const topic = Buffer.alloc(32).fill('my-topic')

swarm.join(topic, { server: true, client: true })

swarm.on('connection', (peer, info) => {
  console.log('New peer connected!')
  peer.write('Hello!')
  peer.on('data', data => console.log('Received:', data))
})
```

Constructor

`new Hyperswarm([options])`

Creates a new Hyperswarm instance.

Options:

- `keyPair` (Object): Noise keypair for the swarm. Default: auto-generated
- `seed` (Buffer): 32-byte seed to deterministically generate keypair
- `maxPeers` (Number): Maximum number of peer connections. Default: `24`
- `firewall` (Function): Function to filter connections: `(remotePublicKey) => boolean`
- `dht` (HyperDHT): Custom DHT instance

Example:

```javascript
import { randomBytes } from 'crypto'

```

```
const swarm = new Hyperswarm({
  seed: randomBytes(32),
  maxPeers: 100,
  firewall: (remotePublicKey) => {
    // Block specific peers
    return blocklist.includes(remotePublicKey.toString('hex'))
  }
})
```

Properties

`swarm.connections`

- Type: `Set<Socket>`
- All active peer connections

`swarm.connecting`

- Type: `Number`
- Count of in-progress connections

`swarm.peers`

- Type: `Map<string, PeerInfo>`
- Map of connected peers by public key

`swarm.dht`

- Type: `HyperDHT`
- Underlying DHT instance

Methods

`swarm.join(topic, [options])`

Join a topic and start discovering peers.

Parameters:

- `topic` (Buffer): 32-byte topic identifier

- `options.server` (Boolean): Accept incoming connections. Default: `true`
- `options.client` (Boolean): Search for peers. Default: `true`

Returns: `PeerDiscovery` object

Example:

```javascript
const topic = Buffer.from('my-chat-room')
const discovery = swarm.join(topic)

await discovery.flushed() // Wait for DHT announcement
console.log('Topic announced to DHT')
```

`swarm.leave(topic)`

Stop discovering peers for a topic.

Parameters:

- `topic` (Buffer): 32-byte topic identifier

Returns: `Promise<void>`

Example:

```javascript
await swarm.leave(topic)
console.log('Left topic')
```

`swarm.joinPeer(noisePublicKey)`

Connect directly to a specific peer.

Parameters:

- `noisePublicKey` (Buffer): 32-byte peer public key

Example:

```javascript
const peerKey = Buffer.from('abc123...', 'hex')
swarm.joinPeer(peerKey)
```

`swarm.leavePeer(noisePublicKey)`

Stop attempting connections to a specific peer.

Parameters:

- `noisePublicKey` (Buffer): 32-byte peer public key

`swarm.flush()`

Wait for all pending DHT operations to complete.

Returns: `Promise<void>`

Example:

```javascript
await swarm.flush()
console.log('All peers discovered')
```

`swarm.destroy()`

Close all connections and shut down the swarm.

Returns: `Promise<void>`

Events

`connection`

Emitted when a new peer connects.

```javascript
swarm.on('connection', (socket, peerInfo) => {
  console.log('Peer connected:', peerInfo.publicKey)

  socket.on('data', data => {
    console.log('Received:', data.toString())
```

```
  })

  socket.write('Hello, peer!')
})
```

Parameters:

- `socket` (NoiseSecretStream): Encrypted duplex stream
- `peerInfo` (Object): Peer information
  - `publicKey` (Buffer): Peer's public key
  - `topics` (Array): Topics this peer joined (client mode only)
  - `client` (Boolean): Whether this is a client connection

`update`

Emitted when swarm state changes (connections, peers, etc.)

```javascript
swarm.on('update', () => {
  console.log('Connections:', swarm.connections.size)
})
```

Hypercore

Distributed append-only log.

Installation

```bash
npm install hypercore
```

Basic Usage

```javascript
import Hypercore from 'hypercore'

const core = new Hypercore('./storage')

```

```
// Append data
await core.append('Hello, world!')
await core.append(['Message 1', 'Message 2'])

// Read data
const data = await core.get(0)
console.log(data.toString()) // 'Hello, world!'

// Replicate to peers
swarm.on('connection', conn => core.replicate(conn))
```

Constructor

`new Hypercore(storage, [key], [options])`

Parameters:

- `storage` (String | Function): Storage location or function
- `key` (Buffer): Public key of existing core (optional)
- `options` (Object): Configuration options

Options:

- `createIfMissing` (Boolean): Create new core if none exists. Default: `true`
- `overwrite` (Boolean): Overwrite existing core. Default: `false`
- `valueEncoding` (String | Object): Encoding for values. Options: `'json'`, `'utf-8'`, `'binary'`. Default: `'binary'`
- `encryptionKey` (Buffer): Enable block encryption
- `sparse` (Boolean): Enable sparse mode. Default: `true`

Example:

javascript

```
// In-memory storage
import RAM from 'random-access-memory'
const core = new Hypercore(RAM)

// File storage
const core = new Hypercore('./my-log')
```

```
8
9   // With encryption
10  const core = new Hypercore('./encrypted', {
11    encryptionKey: Buffer.alloc(32).fill('secret')
12  })
13
14  // JSON encoding
15  const core = new Hypercore('./json-log', {
16    valueEncoding: 'json'
17  })
18
    await core.append({ message: 'Hello', timestamp: Date.now() })
```

Properties

`core.key`

- Type: `Buffer`
- Public key (read capability)

`core.discoveryKey`

- Type: `Buffer`
- Discovery key (for finding peers without leaking read capability)

`core.length`

- Type: `Number`
- Total number of blocks

`core.byteLength`

- Type: `Number`
- Total size in bytes

`core.writable`

- Type: `Boolean`
- Whether this core is writable

`core.readable`

- Type: `Boolean`
- Whether this core is readable

Methods

`core.append(block)`

Append data to the log.

Parameters:

- `block` (Buffer | String | Array): Data to append

Returns: `Promise<{ length, byteLength }>`

Example:

```javascript
// Append single block
await core.append('Hello')

// Append multiple blocks
await core.append(['Block 1', 'Block 2', 'Block 3'])

// Check new length
const { length } = await core.append('Block 4')
console.log('Core length:', length)
```

`core.get(index, [options])`

Get a block by index.

Parameters:

- `index` (Number): Block index
- `options.wait` (Boolean): Wait for download. Default: `true`
- `options.timeout` (Number): Timeout in ms. Default: `0` (no timeout)
- `options.valueEncoding` (String): Override encoding

Returns: `Promise<Buffer>`

Example:

```javascript
const block = await core.get(0)
console.log(block.toString())

// With timeout
try {
  const block = await core.get(10, { timeout: 5000 })
} catch (err) {
  console.log('Timeout or not available')
}
```

`core.has(start, [end])`

Check if blocks are available locally.

Parameters:

- `start` (Number): Start index
- `end` (Number): End index (optional)

Returns: `Promise<Boolean>`

`core.download([range])`

Download a range of blocks.

Parameters:

- `range.start` (Number): Start index
- `range.end` (Number): End index
- `range.blocks` (Array): Specific block indices
- `range.linear` (Boolean): Download sequentially

Returns: `DownloadRange` object

Example:

```javascript
// Download blocks 0-99
const range = core.download({ start: 0, end: 100 })
await range.done()

// Download specific blocks
core.download({ blocks: [5, 10, 15] })

// Download entire core
core.download({ start: 0, end: -1 })
```

`core.createReadStream([options])`

Create a readable stream of blocks.

Parameters:

- `options.start` (Number): Start index. Default: `0`
- `options.end` (Number): End index. Default: `core.length`
- `options.live` (Boolean): Keep streaming new blocks. Default: `false`

Returns: `ReadableStream`

Example:

```javascript
const stream = core.createReadStream({ start: 0 })

for await (const block of stream) {
  console.log('Block:', block.toString())
}
```

`core.replicate(isInitiator, [options])`

Create a replication stream.

Parameters:

- `isInitiator` (Boolean | Stream): Whether this peer initiated the connection
- `options.live` (Boolean): Keep replicating. Default: `true`

Returns: `ReplicationStream`

Example:

```javascript
// With Hyperswarm
swarm.on('connection', (conn) => {
  core.replicate(conn)
})

// Manual replication
const stream1 = core1.replicate(true)
const stream2 = core2.replicate(false)
stream1.pipe(stream2).pipe(stream1)
```

`core.session([options])`

Create a new session (shared underlying core).

Returns: `Hypercore` session

Example:

```javascript
const session = core.session()
await session.append('data')
await session.close() // Doesn't close main core
```

`core.snapshot()`

Create a read-only snapshot at current length.

Returns: `Hypercore` snapshot

Example:

```javascript
const snapshot = core.snapshot()
console.log('Snapshot length:', snapshot.length)

await core.append('new data')

```

```
6  console.log('Original length:', core.length) // +1
   console.log('Snapshot length:', snapshot.length) // unchanged
```

Events

`append`

```javascript
core.on('append', () => {
  console.log('New length:', core.length)
})
```

`peer-add`

```javascript
core.on('peer-add', (peer) => {
  console.log('Peer connected')
})
```

`peer-remove`

```javascript
core.on('peer-remove', (peer) => {
  console.log('Peer disconnected')
})
```

Hyperbee

Key-value database built on Hypercore.

Installation

```bash
npm install hyperbee
```

Basic Usage

```javascript
import Hyperbee from 'hyperbee'
import Hypercore from 'hypercore'

const core = new Hypercore('./db-storage')
const db = new Hyperbee(core, {
  keyEncoding: 'utf-8',
  valueEncoding: 'json'
})

// Put data
await db.put('users/alice', { name: 'Alice', age: 30 })
await db.put('users/bob', { name: 'Bob', age: 25 })

// Get data
const node = await db.get('users/alice')
console.log(node.value) // { name: 'Alice', age: 30 }

// Range query
const stream = db.createReadStream({
  gte: 'users/',
  lte: 'users/~'
})

for await (const { key, value } of stream) {
  console.log(key, value)
}
```

Constructor

`new Hyperbee(core, [options])`

Parameters:

- `core` (Hypercore): Underlying Hypercore
- `options.keyEncoding` (String): Key encoding. Default: `'binary'`
- `options.valueEncoding` (String): Value encoding. Default: `'binary'`

Methods

`db.put(key, value)`

Insert or update a key-value pair.

`db.get(key)`

Get value by key.

Returns: `Promise<{ key, value } | null>`

`db.del(key)`

Delete a key.

`db.batch()`

Create a batch operation.

Example:

```javascript
const batch = db.batch()
await batch.put('key1', 'value1')
await batch.put('key2', 'value2')
await batch.flush()
```

`db.createReadStream([options])`

Create a range query stream.

Options:

- `gt`, `gte`, `lt`, `lte`: Range boundaries
- `reverse` (Boolean): Reverse order
- `limit` (Number): Limit results

Example:

```javascript
const stream = db.createReadStream({
  gte: 'user:a',
  lte: 'user:z',
```

```
  limit: 10
})
```

Hyperdrive

Peer-to-peer file system.

Installation

```bash
npm install hyperdrive
```

Basic Usage

```javascript
import Hyperdrive from 'hyperdrive'

const drive = new Hyperdrive('./drive-storage')

// Write file
await drive.put('/README.md', Buffer.from('# My Drive'))

// Read file
const content = await drive.get('/README.md')
console.log(content.toString())

// List files
for await (const file of drive.list('/')) {
  console.log(file.key) // file path
}
```

Methods

`drive.put(path, buffer)`

Write a file.

`drive.get(path)`

Read a file.

`drive.del(path)`

Delete a file.

`drive.list([folder])`

List files in a folder.

`drive.createReadStream(path)`

Stream a file's contents.

Example:

```javascript
const stream = drive.createReadStream('/large-file.mp4')
stream.pipe(process.stdout)
```

Helper Modules

Corestore

Manage multiple Hypercores.

```javascript
import Corestore from 'corestore'

const store = new Corestore('./storage')

const core1 = store.get({ name: 'my-core' })
const core2 = store.get({ name: 'another-core' })

// Replicate all cores over one connection
swarm.on('connection', conn => store.replicate(conn))
```

Encoding Utilities

compact-encoding

Efficient binary encoding.

```javascript
import { encode, decode } from 'compact-encoding'
import * as c from 'compact-encoding'

const state = c.state()
c.uint.encode(state, 42)
c.string.encode(state, 'hello')

const buffer = state.buffer
```

Best Practices

1. Resource Management

Always clean up resources:

```javascript
// Use Pear's teardown
Pear.teardown(() => swarm.destroy())

// Or manual cleanup
process.on('SIGINT', async () => {
  await swarm.destroy()
  await core.close()
  process.exit(0)
})
```

2. Error Handling

Handle peer errors gracefully:

```javascript
peer.on('error', (err) => {
  console.error('Peer error:', err)
  // Don't crash the app
})
```

3. Use Sessions

For multiple readers:

```javascript
const reader1 = core.session()
const reader2 = core.session()

// Both can read independently
await reader1.close() // Doesn't affect reader2
```

4. Encoding

Use appropriate encodings:

```javascript
// JSON for structured data
const core = new Hypercore('./storage', {
  valueEncoding: 'json'
})

// UTF-8 for text
const core = new Hypercore('./storage', {
  valueEncoding: 'utf-8'
})
```

Module Links

- Hypercore GitHub
- Hyperswarm GitHub
- Hyperbee GitHub
- Hyperdrive GitHub
- Autobase GitHub

Related Documentation

- Core Concepts - Understand the fundamentals
- Data Structures - Choose the right structure
- Networking - Advanced networking patterns
- Examples - See complete applications

Edit this page on GitHub

Next page
Pear Runtime