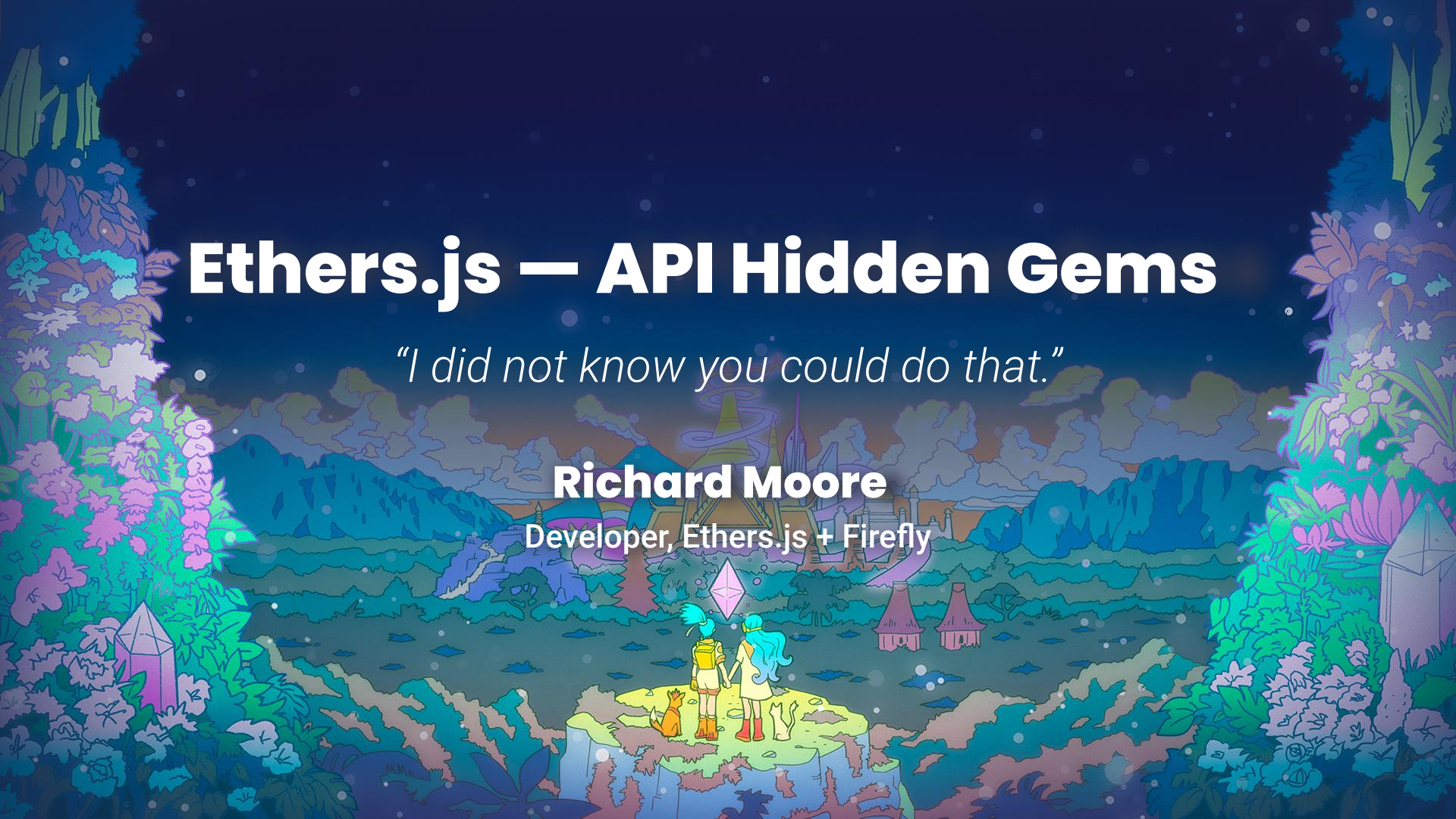
Ethers.js — API Hidden Gems

“I did not know you could do that.”

Richard Moore

Developer, Ethers.js + Firefly



Section 1

Provider

```
import { MulticallProvider } from
  “@ethers-ext/provider-multicall“

// Standard Provider
_p = getDefaultProvider()

// Mutli-call Provider
p = new MulticallProvider(_p)

c = new Contract(addr, [
 “function symbol() returns (string)”,
 “function decimals() returns (uint8)”
], p)

// Async does interfere a bit...
const [ symbol, decimals ] =
  await Promise.all([
    c.symbol(),
    c.decimals()
  ]);

// Only one call is sent
```

MulticallProvider

The MulticallProvider abuses init transactions.

No on-chain contract is necessary.

Works on any chain; nothing to deploy

Caveat: *msg.sender* **is wrong!!** *(true of all multicall)*

Super Quick *“Init Abuse”* Explainer

```
constructor(Call[] memory calls) {
  Result[] memory result = new Result[](calls.length);

  // Make each call
  for (uint256 i = 0; i < calls.length; i++) {
    (result[i].status, result[i].data) = getBytes(calls[i].target, calls[i].data);
  }

  // Override the initcode with our result
  bytes memory output = abi.encode(block.number, result);
  assembly {
    return(add(output, 0x20), mload(output))
  }
}
```

```
// Listen to "debug"
provider.on("debug", (evt) => {
  console.log(evt.action);
})

// CCIP evt.action
sendCcipReadFetchRequest
receiveCcipReadFetchError
sendCcipReadCall
receiveCcipReadCallResult
receiveCcipReadCallError

// JSON-RPC evt.action
sendRpcPayload
receiveRpcResult
receiveRpcError

// Each event has its own additional
// details. CCIP fetch includes the
// list of URLs for example.
```

Debug Events

Each provider sub-class can add its own useful events to emit on debug.

Debug CCIP contract interaction.

Debug JSON-RPC responses.

Basic accounting for JSON-RPC usage.

Any other analytics you desire.

Section 2

Fetch Hooking

```
// Data URIs
url = “data:,Hello%2C%20World%21”
req = new FetchRequest(url)
resp = await req.send()
resp.bodyText
// “Hello World!”


// Custom IPFS Gateway
myIpfsGateway = FetchRequest.
  createIpfsGatewayFunc(myUrl)
FetchRequest.registerGateway(“ipfs”,
  myIpfsGateway)


// Custom Schemes
myWeirdGateway = (req, signal) =>
  { ... }
FetchRequest.registerGateway(“weird”,
  myWeirdGateway)
```

Fetch Requests

Already supports lots of useful protocols:

Such as `data:` URLs.

Default IPFS gateways (can be reconfigured).

Additional (possibly exotic) schemes can be added. Perhaps add FTP support or new CAIP support to your app.

Hijack for User Transparency

By hijacking the FetchRequest, you can provide a pop-up whenever a Provider would make a web request allowing the user to accept.

Perhaps allow the user to select “trust this domain” and keep a Set of trusted domains.

Perhaps always allow the base URL and only ocmplain if a CCIP-read request is potentially leaking user data.

```
// Create a default getUrlFunc
guf = FetchRequest.createGetUrlFunc()

// Configure our request
url = ”https://some-host.com”
req = new FetchRequest(url)
req.getUrlFunc = (req, signal) => {
  ok = prompt(“Allow: ” + req.url)
  if (!ok) { throw new Error(“nope”) }
  return guf(req, signal)
};

// Create a specific provider
p = new JsonRpcProvider(req)

// Or make this change globally
FetchRequest.registerGetUrl(myFunc)

// Triggers a popup!
provider.getBlockNumber()
```

```
// Eg. #1: Tunnel everything over Tor
// (left as an exercise to the reader)
useTor = (req, signal) => {
  // Place magic here to send request
  // over Tor and extract the result
};

// Globally set all requests over Tor
FetchRequest.registerGetUrl(useTor)


// Eg. #2: Sign requests
guf = FetchRequest.createGetUrlFunc()
url = "https://some-host.com"
req = new FetchRequest(url)
req.getUrlFunc = (req, signal) => {
  req.headers["Authorization"] =
    doFancySign(req)
  return guf(req, signal)
};
```

Hijack for Security

You may desire other transports, such as Tor or SOCKS5.

Or truely exotic or experimental transports.

Or mutate every request to a backend, signing it in any way needed.

Hijack for Mockery

Set up an entirely fake universe by returning faux responses to requests.

Allows testing intricate or otherwise hard to reproduce circumstances.

Use the same Hijacking to generate re-playable tests.

Used within Ethers.js for testing FallbackProvider under esoteric circumstances.

```
req = new FetchRequest(url)
req.getUrlFunc = (req, signal) => {
  reqBody = toUtf8String(req.body)
  json = JSON.parse(reqBody)
  resp = { id: json.id }
  switch (json.method) {
    // Create our desired faux data
    case "eth_getBlockNumber":
      resp.result = 42
      respBody = JSON.stringify(resp)
      return {
        statusCode: 200,
        statusMessage: "OK",
        Headers: { },
        body: toUtf8Bytes(respBody)
      }
  }
};

// Create a specific provider
p = new JsonRpcProvider(req)

await p.getBlockNumber()
// 42
```

Section 3

Miscellaneous

```
// Connect to Etherscan
provider = new EtherscanProvider()

// Get the contract, connected to
// the Etherscan provider
c = await provider.getContract(addr)

c.interface
// The entire Interface for addr
```

Etherscan Contracts

Etherscan is awesome. <3

The have *verified* ABIs for many popular contracts.

If the not verified, *null* is returned.

Thank you very much!

Questions?

Richard Moore
Developer, Ethers.js + Firefly
me@ricmoo.com
@ricmoo