Build with dotHYPE

Integrate Identity. Resolve with Confidence.

Whether you're building a wallet, dashboard, or on-chain application, integrating .hype domains helps your users move from raw addresses to structured, readable identity.

Our resolver is fully on-chain, ENS-compatible, and designed without centralized gatekeepers so you can plug-in with confidence. Everything is modular and upgrade-ready, giving us a stable foundation today and flexibility for tomorrow.

What You Can Do

- **Resolve Names to Addresses**
- **Read On-chain Text Records** (e.g. Twitter, avatar)
- **Support Reverse Resolution**
- **Query Ownership via ERC721**

Start Integrating

→ Resolver Reference
Includes ABI, contract address, and example calls (`addr()`, `text()`, `name()`).

→ Integration Examples (Coming Soon)
Code snippets for Ethers.js / Web3.js / React environments.

SDK (Coming Soon)

Our JS/TS SDK will simplify:

- Name availability + resolution
- Minting hooks

- React-ready UI components

Ready to build with dotHYPE? → Get Started with the Resolver

How to get .hype integrated in your dApp

This guide will help you integrate resolution into your HyperEVM application using our contract, with a real-world example for customizing your wallet connect UI.

Use `.hype` Names in Your App

The dotHYPE resolver is fully ENS-compatible, making integration simple for developers already familiar with Ethereum naming systems. You can use standard `addr()`, `getName()`, and `text()` calls to resolve identity information from `.hype` domains. No proprietary logic required.

Basic Usage

To resolve `.hype` names, use the d**otHYPEResolver** contract and the following standard functions:

Function	Purpose
`addr(bytes32 node)`	Returns the address for a `.hype` name
`getName(address)`	Returns the primary `.hype` name for a wallet
`text(bytes32 node, string key)`	Returns a text record (e.g. `twitter`, `avatar`)

> Note: The `node` refers to the ENS-compatible **namehash** of the `.hype` domain.

Custom Connect Button

Display .hype Names Instead of Wallet Addresses

This component enhances your RainbowKit wallet connection UX by displaying the user's primary `.hype` name (if set) instead of a truncated address. It works seamlessly across HyperEVM mainnet and testnet, supports loading states, and falls back to address when no name is set.

Step 1. Overview & Purpose

This custom component:

- Replaces wallet addresses with `.hype` domains
- Uses the `DotHypeResolver.name(address)` function
- Falls back to the address if no primary name is set
- Detects network automatically via chain ID
- Supports full styling customization

> ⚠️ **Important:**
> **Replace the placeholder contract addresses in the code with your actual deployed resolver addresses (see `/info` page for details).**

Step 2. CustomConnectButton.tsx

Full Code

```
import { useState, useEffect, useCallback } from "react";
import { ConnectButton } from "@rainbow-me/rainbowkit";
import { usePublicClient } from "wagmi";
import { Address } from "viem";
import { getCurrentNetworkAddresses } from
"@/contracts/addresses";
import { DOT_HYPE_RESOLVER_ABI } from "@/contracts/abis";

/**
 * Custom ConnectButton that shows primary domain name instead
of truncated address
 * when a primary domain is set for the connected wallet
 */
export function CustomConnectButton() {
  return (
    <ConnectButton.Custom>
      {({
        account,
        chain,
        openAccountModal,
        openChainModal,
        openConnectModal,
        authenticationStatus,
        mounted,
      }) => {
        // Note: If your app doesn't use authentication, you
        // can remove all 'authenticationStatus' checks
        const ready = mounted && authenticationStatus !==
"loading";
        const connected =
          ready &&
          account &&
          chain &&
          (!authenticationStatus || authenticationStatus ===
"authenticated");

        return (
          <div
            {...(!ready && {
              "aria-hidden": true,
              style: {
                opacity: 0,
                pointerEvents: "none",
                userSelect: "none",
              },
            })}
          >
            {() => {
```

```
              if (!connected) {
                return (
                  <button
                    onClick={openConnectModal}
                    type="button"
                    className="bg-gradient-to-r from-hype-
primary to-hype-secondary text-white px-6 py-3 rounded-lg
font-medium hover:from-hype-secondary hover:to-hype-primary
transition-all duration-200 shadow-lg hover:shadow-xl"
                  >
                    Connect Wallet
                  </button>
                );
              }

              if (chain.unsupported) {
                return (
                  <button
                    onClick={openChainModal}
                    type="button"
                    className="bg-red-500 text-white px-4 py-2
rounded-lg font-medium hover:bg-red-600 transition-colors"
                  >
                    Wrong network
                  </button>
                );
              }

              return (
                <div className="flex items-center gap-3">
                  <button
                    onClick={openChainModal}
                    className="flex items-center gap-2 bg-
gray-100 dark:bg-gray-700 hover:bg-gray-200 dark:hover:bg-
gray-600 px-3 py-3 rounded-lg transition-colors"
                    type="button"
                  >
                    {chain.hasIcon && (
                      <div
                        style={{
                          background: chain.iconBackground,
                          width: 20,
                          height: 20,
                          borderRadius: 999,
                          overflow: "hidden",
                          marginRight: 4,
                        }}
                      >
```

Step 3. CSS Styles

Styles.css

```
                        {chain.iconUrl && (
                          <img
                            alt={chain.name ?? "Chain icon"}
                            src={chain.iconUrl}
                            style={{ width: 20, height: 20 }}
                          />
                        )}
                      </div>
                    )}
                    <span className="text-sm font-medium text-
gray-700 dark:text-gray-300">
                      {chain.name}
                    </span>
                  </button>

                  <AccountButton
                    account={account}
                    openAccountModal={openAccountModal}
                  />
                </div>
              );
            })()}
          </div>
        );
      }}
    </ConnectButton.Custom>
  );
}

/**
 * Account button component that shows primary domain or
truncated address
 */
interface AccountButtonProps {
  account: {
    address: string;
    displayBalance?: string;
  };
  openAccountModal: () => void;
}

function AccountButton({ account, openAccountModal }:
AccountButtonProps) {
  const { primaryDomain, isLoading: primaryDomainLoading } =
usePrimaryDomain(
    account?.address as Address
  );
  const { avatar, isLoading: avatarLoading } = useUserAvatar(
```

```css
/* CSS Styles for Components */

/* Connect Button Styles */
.connect-button {
  background: linear-gradient(135deg, #6366f1, #8b5cf6);
  color: white;
  padding: 8px 24px;
  border-radius: 8px;
  font-weight: 500;
  border: none;
  cursor: pointer;
  transition: all 0.2s ease;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
}

.connect-button:hover {
  background: linear-gradient(135deg, #8b5cf6, #6366f1);
  box-shadow: 0 6px 12px rgba(0, 0, 0, 0.15);
}

.wrong-network-button {
  background: #ef4444;
  color: white;
  padding: 8px 16px;
  border-radius: 8px;
  font-weight: 500;
  border: none;
  cursor: pointer;
  transition: background-color 0.2s ease;
}

.wrong-network-button:hover {
  background: #dc2626;
}

.connected-container {
  display: flex;
  align-items: center;
  gap: 12px;
}

.chain-button {
  display: flex;
  align-items: center;
  gap: 8px;
  background: rgba(156, 163, 175, 0.1);
  padding: 8px 12px;
  border-radius: 8px;
```

```
  border: none;
  cursor: pointer;
  transition: background-color 0.2s ease;
  font-size: 14px;
}

.chain-button:hover {
  background: rgba(156, 163, 175, 0.2);
}

.chain-icon {
  width: 20px;
  height: 20px;
  border-radius: 50%;
}

.account-button {
  display: flex;
  align-items: center;
  gap: 8px;
  background: white;
  border: 1px solid #d1d5db;
  padding: 8px 16px;
  border-radius: 8px;
  cursor: pointer;
  transition: all 0.2s ease;
  box-shadow: 0 1px 3px rgba(0, 0, 0, 0.1);
}

.account-button:hover {
  background: #f9fafb;
}

.account-info {
  display: flex;
  align-items: center;
  gap: 8px;
}

.status-indicator {
  width: 8px;
  height: 8px;
  background: #10b981;
  border-radius: 50%;
}

.loading-spinner {
  width: 16px;
```

```
  height: 16px;
  border: 2px solid #e5e7eb;
  border-top: 2px solid #6366f1;
  border-radius: 50%;
  animation: spin 1s linear infinite;
}

@keyframes spin {
  to { transform: rotate(360deg); }
}

.input-wrapper {
  position: relative;
}

.address-input {
  width: 100%;
  padding: 12px 40px 12px 12px;
  border: 1px solid #d1d5db;
  border-radius: 8px;
  font-size: 14px;
  transition: all 0.2s ease;
  background: white;
}

.address-input:focus {
  outline: none;
  border-color: #6366f1;
  box-shadow: 0 0 0 3px rgba(99, 102, 241, 0.1);
}

.address-input.loading {
  border-color: #3b82f6;
}
```

Step 4. Usage Example

TSX

```
import { CustomConnectButton } from './CustomConnectButton';
import { useChainId } from 'wagmi';

function App() {
  const chainId = useChainId();

  return (
    <div>
      <CustomConnectButton chainId={chainId} />
    </div>
  );
}
```

⚠️ Pro Tip

Th op.
Cl s to
te

```
.address-input.valid {
  border-color: #10b981;
}

.address-input.error {
  border-color: #ef4444;
}

.input-status-icon {
  position: absolute;
  right: 12px;
  top: 50%;
  transform: translateY(-50%);
  font-size: 16px;
}

.status-message {
  margin-top: 4px;
  font-size: 14px;
  display: flex;
  align-items: center;
  gap: 4px;
}

.status-message.loading {
  color: #3b82f6;
}

.status-message.success {
  color: #10b981;
}

.status-message.error {
  color: #ef4444;
}

.resolution-display {
  margin-top: 8px;
  padding: 12px;
  border-radius: 6px;
  font-size: 14px;
}

.resolution-display.domain {
  background: rgba(59, 130, 246, 0.1);
  border: 1px solid rgba(59, 130, 246, 0.2);
}
```

```
.resolution-display.address {
  background: rgba(16, 185, 129, 0.1);
  border: 1px solid rgba(16, 185, 129, 0.2);
}

.resolution-content {
  display: flex;
  align-items: center;
  gap: 8px;
}

.avatar-thumbnail {
  width: 24px;
  height: 24px;
  border-radius: 4px;
  border: 1px solid rgba(0, 0, 0, 0.1);
  object-fit: cover;
  flex-shrink: 0;
}

.resolution-value {
  font-family: monospace;
  word-break: break-all;
  color: #1f2937;
  flex: 1;
}

.help-text {
  margin-top: 4px;
  font-size: 12px;
  color: #6b7280;
}

/* Dark mode styles */
:global(.dark) .chain-button {
  background: rgba(75, 85, 99, 0.5);
  color: #f3f4f6;
}

:global(.dark) .account-button {
  background: #1f2937;
  border-color: #374151;
  color: white;
}

:global(.dark) .account-button:hover {
  background: #111827;
}
```

```
:global(.dark) .address-input {
  background: #1f2937;
  border-color: #374151;
  color: white;
}

:global(.dark) .input-label {
  color: #d1d5db;
}

:global(.dark) .help-text {
  color: #9ca3af;
}

:global(.dark) .resolution-value {
  color: #f3f4f6;
}
```