Author Profile

Haider Khan *is a blockchain researcher/ writer specializing in interoperable blockchain ecosystems. His research focuses on translating complex blockchain concepts into accessible insights for diverse audiences, from cryptocurrency newcomers to seasoned investors and technology enthusiasts. With a background in traditional finance, distributed systems and tokenomics, Haider brings analytical depth to his exploration of emerging blockchain technologies and their practical applications.*

Contact Information

Available for research collaborations, consulting, and speaking engagements on interoperability, Cosmos ecosystem, and cross-chain DeFi.

Email(s) : alik.defianalyst@gmail.com, khan.haiderali87@gmail.com

Website : https://blockchaindefispecialist.com/

LinkedIN *:* https://www.linkedin.com/in/haider-a-khan/

Nexus DeFi Lending Protocol API Documentation

Overview

Nexus is a decentralized lending protocol built on Ethereum that enables users to lend and borrow digital assets without traditional intermediaries. This documentation provides comprehensive information for developers looking to integrate the Nexus protocol into their applications.

Key Features

- Permissionless lending and borrowing of supported assets
- Variable and fixed interest rate options
- Over-collateralized loan positions
- Liquidation protection mechanisms
- Flash loan functionality
- Governance participation

Prerequisites

Before integrating with the Nexus protocol, ensure you have:

- A development environment with Node.js (v16+)
- Knowledge of JavaScript/TypeScript and Web3 concepts
- An Ethereum wallet (MetaMask, WalletConnect, etc.)
- Test ETH and tokens on supported networks (Ethereum Mainnet, Goerli Testnet, etc.)
- Ethers.js or Web3.js library
- A provider URL (Infura, Alchemy, etc.)

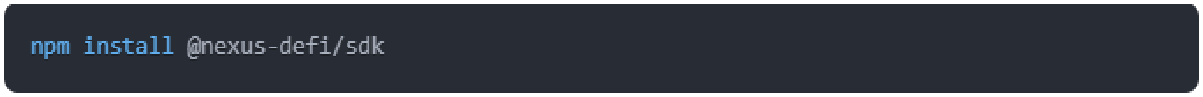
Supported Networks

Network	Chain ID	Contract Address
Ethereum Mainnet	1	0x7Fc66500c84A76Ad7e9c93437bFc5Ac33E2DDaE9
Goerli Testnet	5	0x4da27a545c0c5B758a6BA100e3a049001de870f5
Arbitrum	42161	0xBA5DdD1f9d7F570dc94a51479a000E3BCE967196
Optimism	10	0x76FB31fb4af56892A25e32cFC43De717950c9278

Quick Start Guide

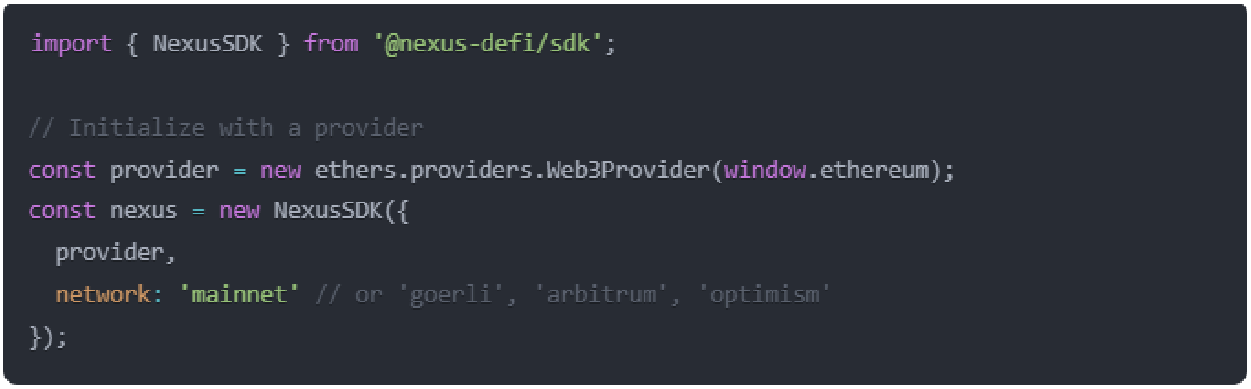
This section provides basic integration steps to quickly implement Nexus lending functionality.

1. Install the SDK



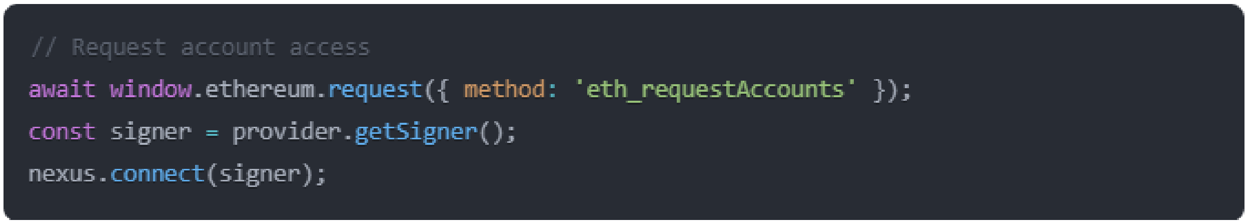
```
npm install @nexus-defi/sdk
```

2. Initialize the SDK



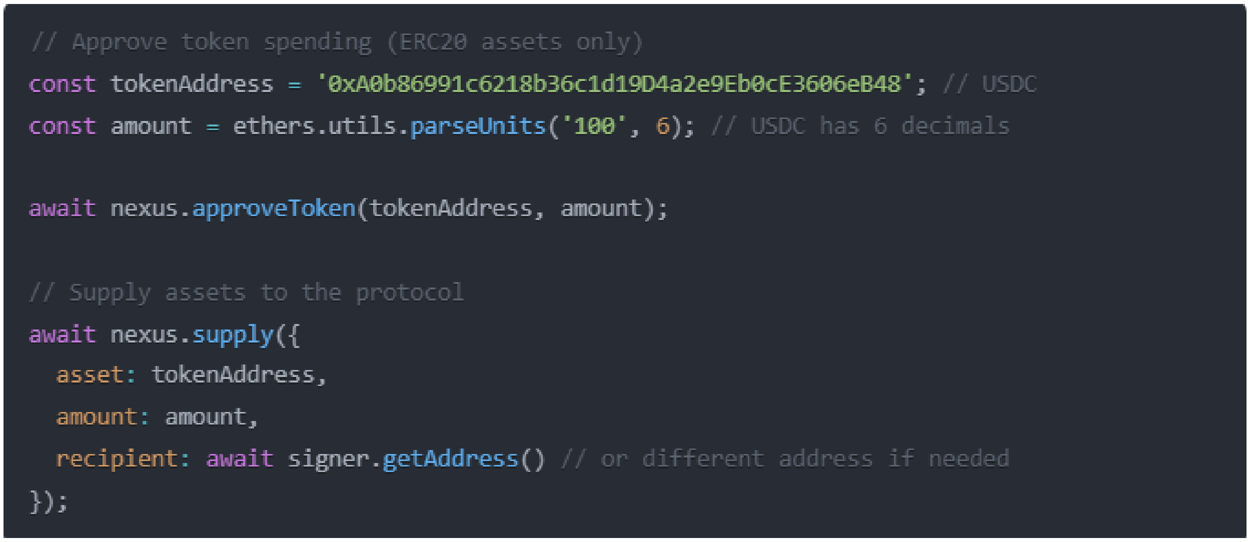
```
import { NexusSDK } from '@nexus-defi/sdk';

// Initialize with a provider
const provider = new ethers.providers.Web3Provider(window.ethereum);
const nexus = new NexusSDK({
  provider,
  network: 'mainnet' // or 'goerli', 'arbitrum', 'optimism'
});
```

3. Connect User Wallet



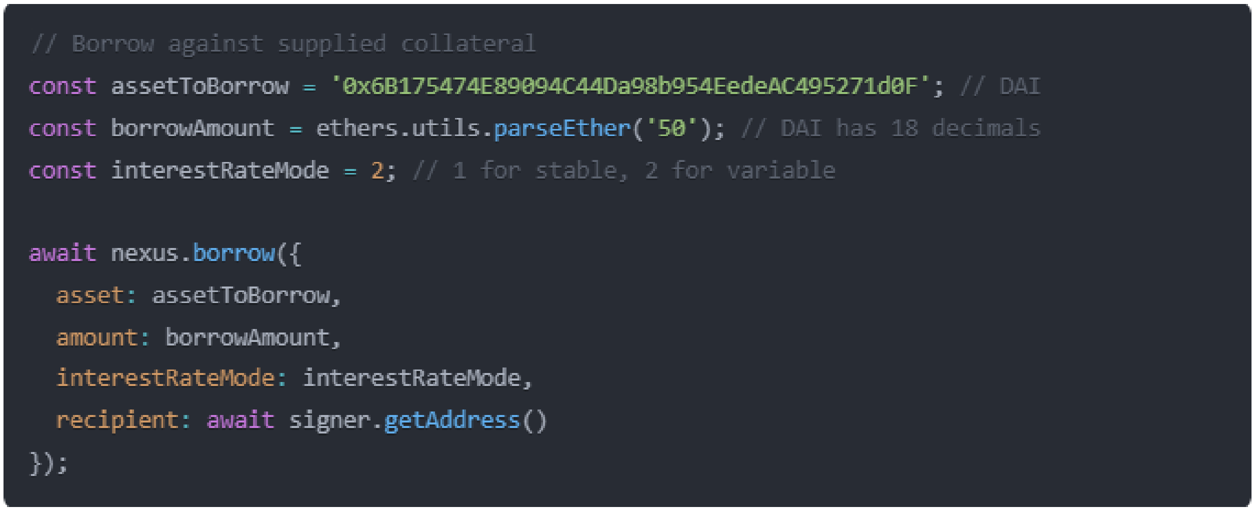
```
// Request account access
await window.ethereum.request({ method: 'eth_requestAccounts' });
const signer = provider.getSigner();
nexus.connect(signer);
```

4. Supply Assets



```
// Approve token spending (ERC20 assets only)
const tokenAddress = '0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48'; // USDC
const amount = ethers.utils.parseUnits('100', 6); // USDC has 6 decimals

await nexus.approveToken(tokenAddress, amount);

// Supply assets to the protocol
await nexus.supply({
  asset: tokenAddress,
  amount: amount,
  recipient: await signer.getAddress() // or different address if needed
});
```

5. Borrow Assets



```
// Borrow against supplied collateral
const assetToBorrow = '0x6B175474E89094C44Da98b954EedeAC495271d0F'; // DAI
const borrowAmount = ethers.utils.parseEther('50'); // DAI has 18 decimals
const interestRateMode = 2; // 1 for stable, 2 for variable

await nexus.borrow({
  asset: assetToBorrow,
  amount: borrowAmount,
  interestRateMode: interestRateMode,
  recipient: await signer.getAddress()
});
```

Core Concepts

Understanding these key concepts is essential for successful integration with the Nexus protocol.

Lending Pools

Lending pools are the core component of the Nexus protocol. Each supported asset has a corresponding lending pool where users can supply assets and borrow against their collateral.

Supply and Borrow Mechanisms

When users supply assets to the protocol:

- They receive nTokens representing their share of the lending pool
- Their supplied assets begin earning interest immediately
- Assets become available as collateral (if enabled)

When users borrow from the protocol:

- They must have sufficient collateral value
- They can choose between variable and stable interest rates
- Health factor must remain above 1.0 to avoid liquidation

Interest Rates

Nexus uses a dynamic interest rate model:

- **Utilization Rate**: The percentage of pool funds currently borrowed
- **Variable Interest Rate**: Adjusts based on utilization, rises as utilization increases
- **Stable Interest Rate**: Fixed at the time of borrowing but can be rebalanced under certain conditions

Collateral and Loan-to-Value (LTV)

Each asset has specific risk parameters:

- **Loan-to-Value (LTV)**: Maximum percentage of collateral value that can be borrowed
- **Liquidation Threshold**: When health factor falls below 1.0, position can be liquidated
- **Liquidation Penalty**: Fee paid by borrower during liquidation

Health Factor

The health factor is a numeric representation of loan safety:

- Health Factor = Total Collateral Value in ETH × Liquidation Threshold ÷ Total Borrows in ETH
- Must remain > 1.0 to avoid liquidation
- Higher health factor indicates safer position

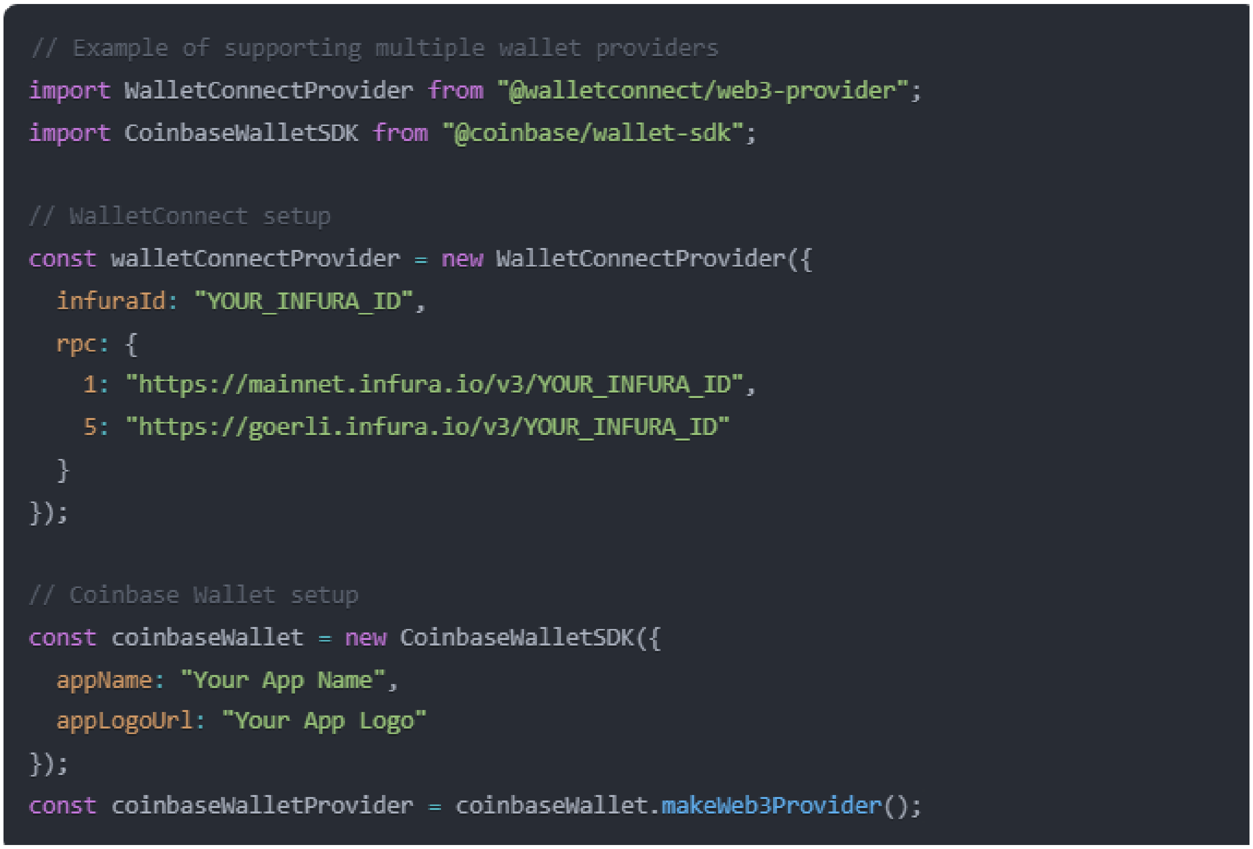
Authentication & Security

Nexus uses standard Web3 authentication methods for secure interaction with the protocol.

Connecting Wallets

Users must connect their Ethereum wallet to interact with Nexus. Supported connection methods include:

- MetaMask
- WalletConnect
- Coinbase Wallet
- Fortmatic
- Portis

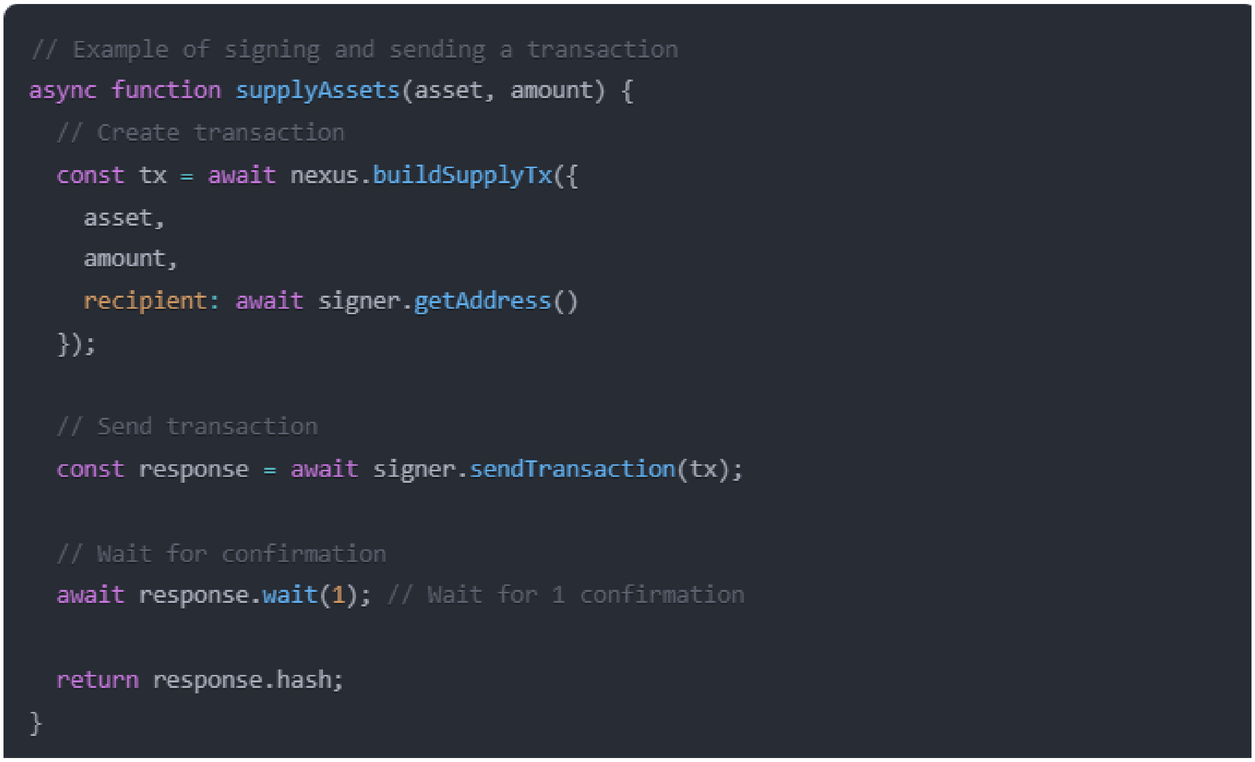


```javascript
// Example of supporting multiple wallet providers
import WalletConnectProvider from "@walletconnect/web3-provider";
import CoinbaseWalletSDK from "@coinbase/wallet-sdk";

// WalletConnect setup
const walletConnectProvider = new WalletConnectProvider({
  infuraId: "YOUR_INFURA_ID",
  rpc: {
    1: "https://mainnet.infura.io/v3/YOUR_INFURA_ID",
    5: "https://goerli.infura.io/v3/YOUR_INFURA_ID"
  }
});

// Coinbase Wallet setup
const coinbaseWallet = new CoinbaseWalletSDK({
  appName: "Your App Name",
  appLogoUrl: "Your App Logo"
});
const coinbaseWalletProvider = coinbaseWallet.makeWeb3Provider();
```

Transaction Signing

All interactions with the protocol require signed transactions:

1. Transaction is created (e.g., supply assets, borrow)
2. User signs the transaction with their private key
3. Signed transaction is submitted to the blockchain
4. Transaction is processed, and the operation completes



```
// Example of signing and sending a transaction
async function supplyAssets(asset, amount) {
  // Create transaction
  const tx = await nexus.buildSupplyTx({
    asset,
    amount,
    recipient: await signer.getAddress()
  });

  // Send transaction
  const response = await signer.sendTransaction(tx);

  // Wait for confirmation
  await response.wait(1); // Wait for 1 confirmation

  return response.hash;
}
```

Permission Management

Different actions in the protocol require specific permissions:

- **Token Approvals**: Users must approve the protocol to spend their ERC20 tokens
- **Delegation**: Users can delegate borrowing power to other addresses
- **Contract Interactions**: Smart contract allowances for flash loans and other advanced operations

Always use the minimum required permissions for security best practices.

API Reference

The Nexus API provides methods to interact with all aspects of the protocol.

Protocol Data Methods

getReserveData(asset)

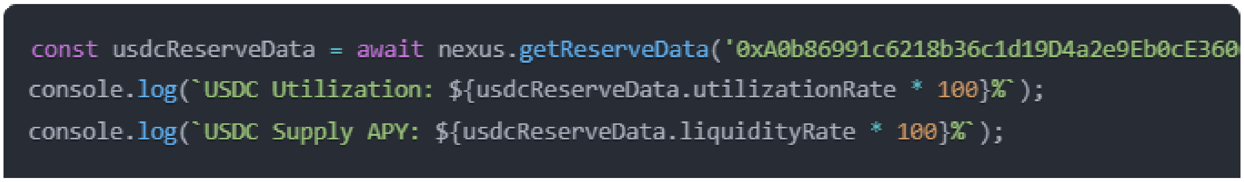
Retrieves detailed information about a specific asset reserve.

Parameters:

- asset (string): The address of the asset

Returns: Object containing:

- utilizationRate: Current utilization of the reserve
- availableLiquidity: Amount available for borrowing
- totalStableDebt: Total debt borrowed at stable rate
- totalVariableDebt: Total debt borrowed at variable rate
- liquidityRate: Current supply APY
- variableBorrowRate: Current variable borrow APY
- stableBorrowRate: Current stable borrow APY
- averageStableBorrowRate: Weighted average of stable rates
- ltv: Maximum loan-to-value ratio
- liquidationThreshold: Threshold for liquidation
- liquidationBonus: Bonus for liquidators
- isActive: Whether reserve is active
- isFrozen: Whether reserve is frozen



```
const usdcReserveData = await nexus.getReserveData('0xA0b86991c6218b36c1d19D4a2e9Eb0cE360
console.log(`USDC Utilization: ${usdcReserveData.utilizationRate * 100}%`);
console.log(`USDC Supply APY: ${usdcReserveData.liquidityRate * 100}%`);
```

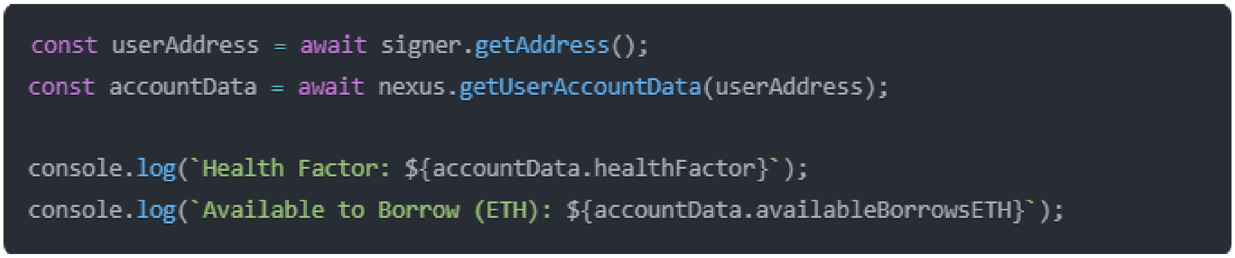
Parameters:

- user (string): The address of the user

Returns: Object containing:

- totalCollateralETH: Total collateral in ETH
- totalDebtETH: Total debt in ETH
- availableBorrowsETH: Available borrowing power in ETH
- currentLiquidationThreshold: Current liquidation threshold
- ltv: Current loan to value

- healthFactor: Current health factor



```
const userAddress = await signer.getAddress();
const accountData = await nexus.getUserAccountData(userAddress);

console.log(`Health Factor: ${accountData.healthFactor}`);
console.log(`Available to Borrow (ETH): ${accountData.availableBorrowsETH}`);
```

User Action Methods

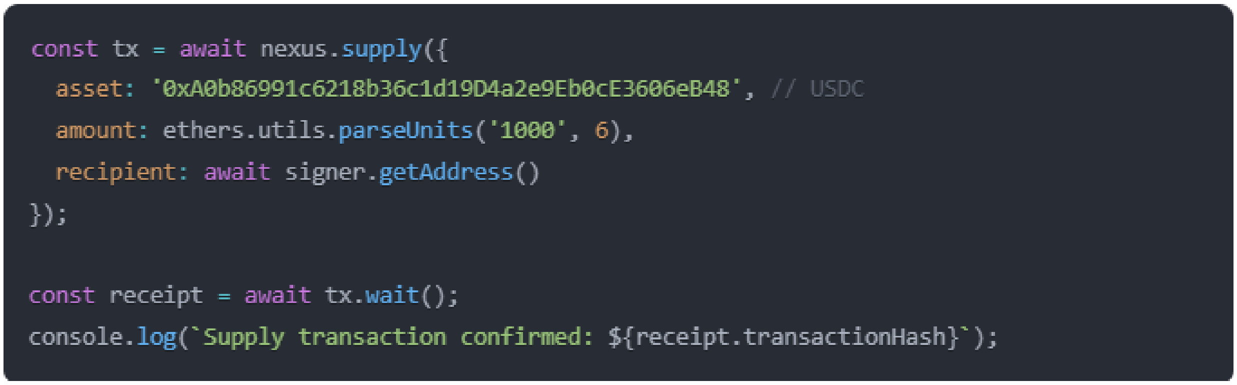
supply(params)

Supplies an asset to the Nexus protocol.

Parameters:

- params (object):
  - asset (string): The address of the asset
  - amount (BigNumber): The amount to supply
  - recipient (string, optional): The address that will receive the nTokens
  - referralCode (number, optional): Referral code

Returns: Transaction response object



```
const tx = await nexus.supply({
  asset: '0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48', // USDC
  amount: ethers.utils.parseUnits('1000', 6),
  recipient: await signer.getAddress()
});

const receipt = await tx.wait();
console.log(`Supply transaction confirmed: ${receipt.transactionHash}`);
```

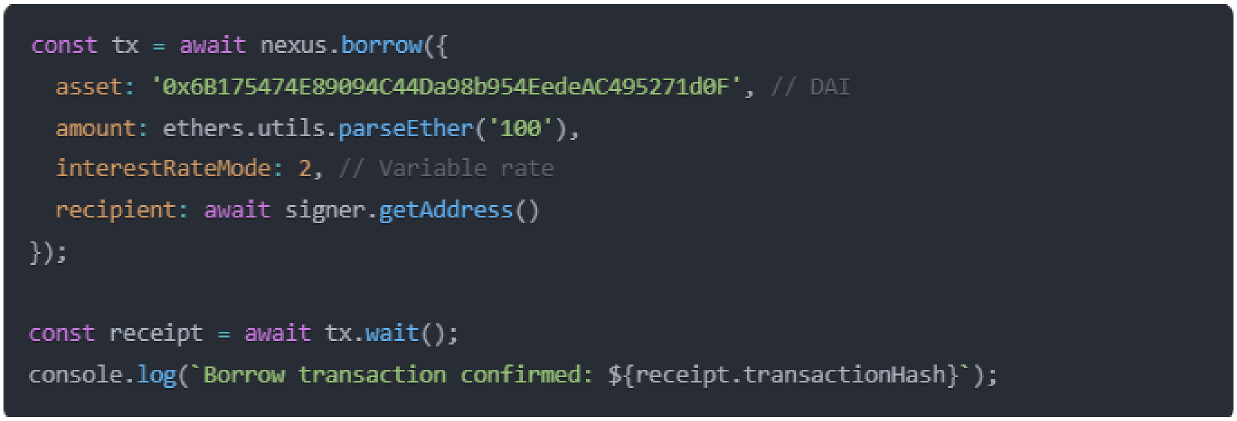
Withdraws supplied assets from the protocol.

Parameters:

- params (object):
  - asset (string): The address of the asset

    - amount (BigNumber): The amount to withdraw (use MAX_UINT256 for maximum)
    - recipient (string, optional): The address that will receive the withdrawn assets

Returns: Transaction response object



```
const tx = await nexus.borrow({
  asset: '0x6B175474E89094C44Da98b954EedeAC495271d0F', // DAI
  amount: ethers.utils.parseEther('100'),
  interestRateMode: 2, // Variable rate
  recipient: await signer.getAddress()
});

const receipt = await tx.wait();
console.log(`Borrow transaction confirmed: ${receipt.transactionHash}`);
```

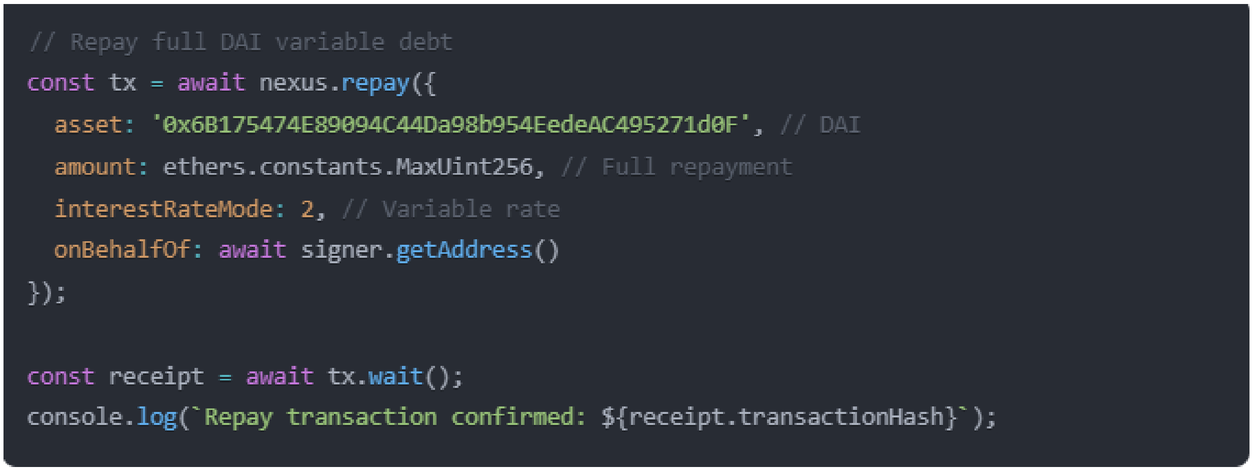
repay(params)

Repays a borrowed asset.

Parameters:

- params (object):
    - asset (string): The address of the asset
    - amount (BigNumber): The amount to borrow
    - interestRateMode (number): 1 for stable, 2 for variable
    - referralCode (number, optional): Referral code
    - recipient (string, optional): The address that will receive the borrowed assets

Returns: Transaction response object



```
// Repay full DAI variable debt
const tx = await nexus.repay({
  asset: '0x6B175474E89094C44Da98b954EedeAC495271d0F', // DAI
  amount: ethers.constants.MaxUint256, // Full repayment
  interestRateMode: 2, // Variable rate
  onBehalfOf: await signer.getAddress()
});

const receipt = await tx.wait();
console.log(`Repay transaction confirmed: ${receipt.transactionHash}`);
```

const receipt = await tx.wait();

console.log(` Borrow transaction confirmed: ${receipt.transactionHash}`);

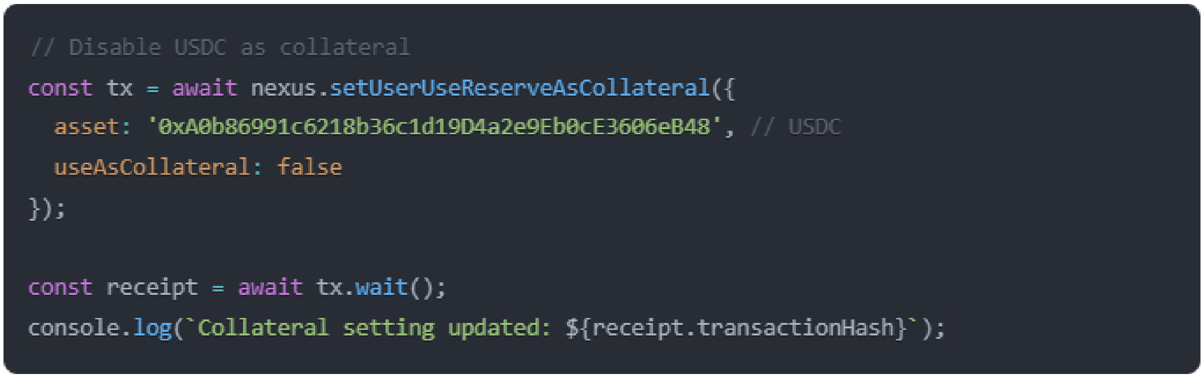
repay(params)

Repays a borrowed asset.

Parameters:

- params (object):
  - asset (string): The address of the asset
  - amount (BigNumber): The amount to repay (use MAX_UINT256 for full repayment)
  - interestRateMode (number): 1 for stable, 2 for variable
  - onBehalfOf (string, optional): The address of the borrower (if repaying on behalf)

Returns: Transaction response object



```javascript
// Disable USDC as collateral
const tx = await nexus.setUserUseReserveAsCollateral({
  asset: '0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48', // USDC
  useAsCollateral: false
});

const receipt = await tx.wait();
console.log(`Collateral setting updated: ${receipt.transactionHash}`);
```

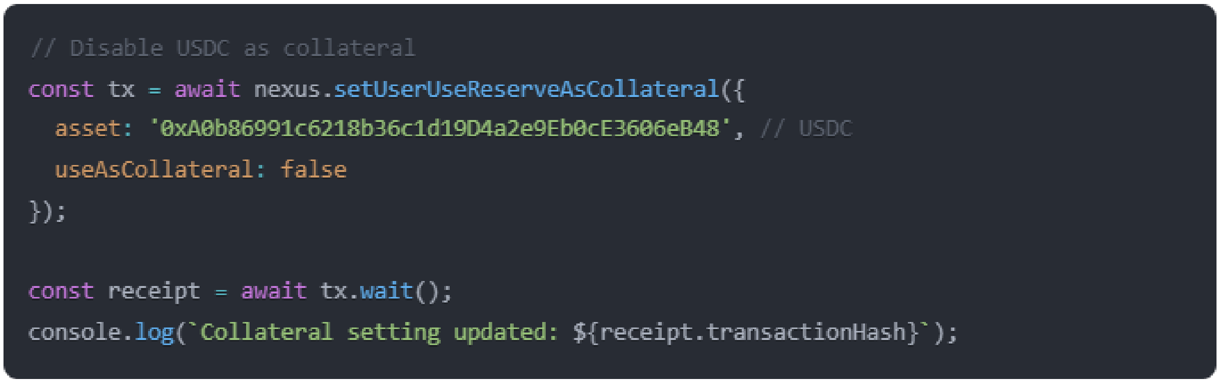
setUserUseReserveAsCollateral(params)

Enables or disables an asset as collateral.

Parameters:

- params (object):
  - asset (string): The address of the asset
  - useAsCollateral (boolean): True to use as collateral, false otherwise

Returns: Transaction response object



```javascript
// Disable USDC as collateral
const tx = await nexus.setUserUseReserveAsCollateral({
  asset: '0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48', // USDC
  useAsCollateral: false
});

const receipt = await tx.wait();
console.log(`Collateral setting updated: ${receipt.transactionHash}`);
```

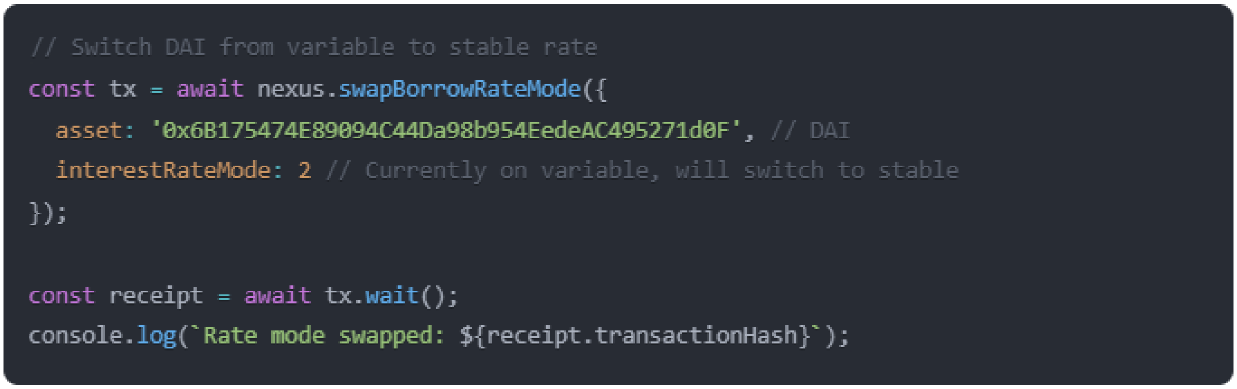
swapBorrowRateMode(params)

Switches between stable and variable borrow rate modes.

Parameters:

- params (object):
  - asset (string): The address of the asset
  - interestRateMode (number): The current interest rate mode (1 for stable, 2 for variable)

Returns: Transaction response object



```javascript
// Switch DAI from variable to stable rate
const tx = await nexus.swapBorrowRateMode({
  asset: '0x6B175474E89094C44Da98b954EedeAC495271d0F', // DAI
  interestRateMode: 2 // Currently on variable, will switch to stable
});

const receipt = await tx.wait();
console.log(`Rate mode swapped: ${receipt.transactionHash}`);
```

Advanced Methods

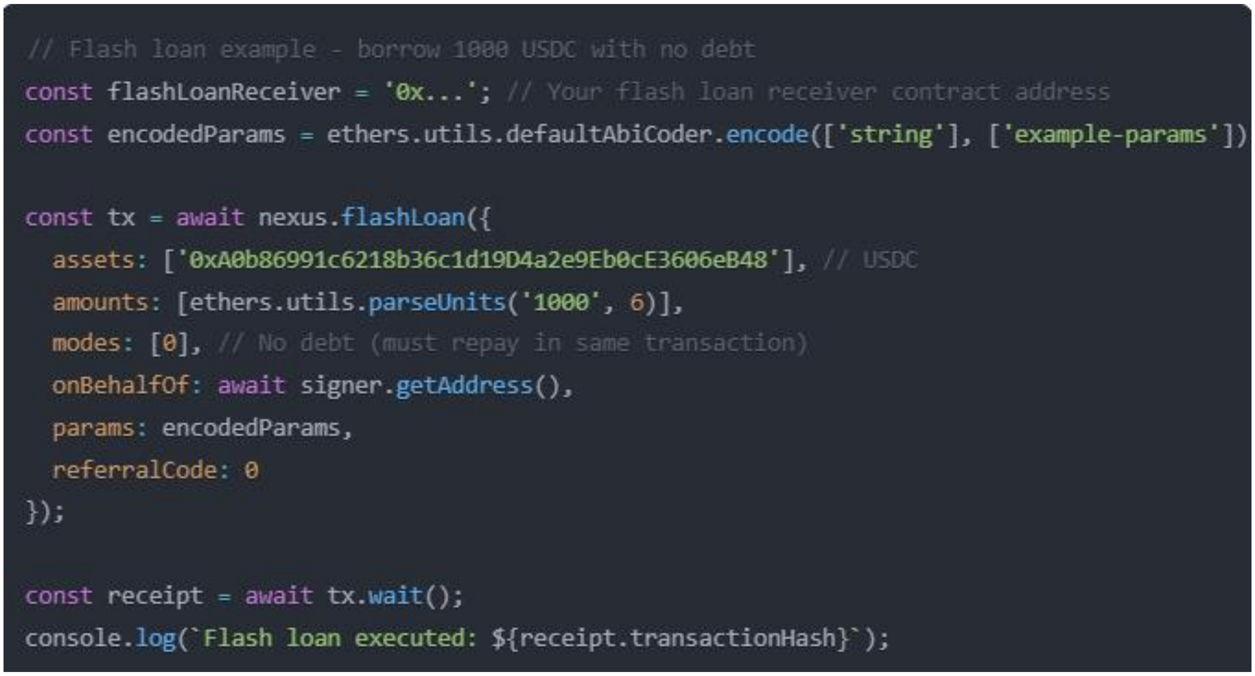
flashLoan(params)

Executes a flash loan.

Parameters:

- params (object):
  - assets (string[]): Array of asset addresses
  - amounts (BigNumber[]): Array of amounts to borrow
  - modes (number[]): Array of interest rate modes (0 for no debt, 1 for stable, 2 for variable)
  - onBehalfOf (string): The address that will incur debt if modes[i] > 0
  - params (string): Encoded parameters for the receiver
  - referralCode (number, optional): Referral code

Returns: Transaction response object



```
// Flash loan example - borrow 1000 USDC with no debt
const flashLoanReceiver = '0x...'; // Your flash loan receiver contract address
const encodedParams = ethers.utils.defaultAbiCoder.encode(['string'], ['example-params'])

const tx = await nexus.flashLoan({
  assets: ['0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48'], // USDC
  amounts: [ethers.utils.parseUnits('1000', 6)],
  modes: [0], // No debt (must repay in same transaction)
  onBehalfOf: await signer.getAddress(),
  params: encodedParams,
  referralCode: 0
});

const receipt = await tx.wait();
console.log(`Flash loan executed: ${receipt.transactionHash}`);
```

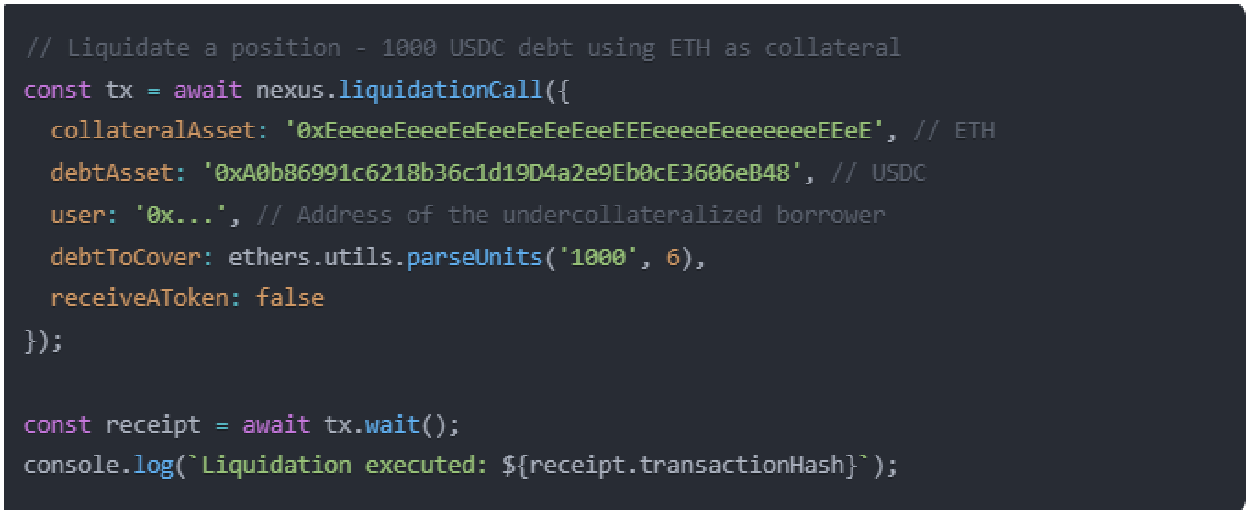
liquidationCall(params)

Liquidates an undercollateralized position.

Parameters:

- params (object):
  - collateralAsset (string): The address of the collateral asset
  - debtAsset (string): The address of the debt asset
  - user (string): The address of the borrower
  - debtToCover (BigNumber): The amount of debt to cover
  - receiveAToken (boolean): Whether to receive the collateral as aToken or the underlying asset

Returns: Transaction response object



```javascript
// Liquidate a position - 1000 USDC debt using ETH as collateral
const tx = await nexus.liquidationCall({
  collateralAsset: '0xEeeeeEeeeEeEeeEeEeeEEEeeeeEeeeeeeeEEeE', // ETH
  debtAsset: '0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48', // USDC
  user: '0x...', // Address of the undercollateralized borrower
  debtToCover: ethers.utils.parseUnits('1000', 6),
  receiveAToken: false
});

const receipt = await tx.wait();
console.log(`Liquidation executed: ${receipt.transactionHash}`);
```

Sample Implementations

Basic Lending dApp Implementation

This example demonstrates a simple React component for supplying assets to the protocol:

javascript

Copy

```
import React, { useState, useEffect } from 'react';

import { ethers } from 'ethers';

import { NexusSDK } from '@nexus-defi/sdk';


function SupplyForm() {

  const [asset, setAsset] = useState('');

  const [amount, setAmount] = useState('');

  const [nexus, setNexus] = useState(null);

  const [assets, setAssets] = useState([]);

  const [loading, setLoading] = useState(false);

  const [error, setError] = useState('');

  const [success, setSuccess] = useState('');
```

```
// Initialize SDK on component mount
useEffect(() => {
  async function initialize() {
    try {
      // Request access to user's wallet
      await window.ethereum.request({ method: 'eth_requestAccounts' });

      // Create provider and SDK instance
      const provider = new ethers.providers.Web3Provider(window.ethereum);
      const signer = provider.getSigner();
      const nexusInstance = new NexusSDK({
        provider,
        network: 'mainnet'
      });
      nexusInstance.connect(signer);

      // Get available assets
      const availableAssets = await nexusInstance.getReservesList();
      const assetDetails = await Promise.all(
        availableAssets.map(async (assetAddress) => {
          const data = await nexusInstance.getReserveData(assetAddress);
          const tokenMetadata = await nexusInstance.getTokenMetadata(assetAddress);
          return {
            address: assetAddress,
            symbol: tokenMetadata.symbol,
      decimals: tokenMetadata.decimals,
      liquidityRate: data.liquidityRate
     };
    })
   );

   setNexus(nexusInstance);
   setAssets(assetDetails);
  } catch (err) {
   setError('Failed to initialize: ' + err.message);
  }
 }

 initialize();
}, []);

// Handle asset supply
const handleSupply = async (e) => {
 e.preventDefault();
 setLoading(true);
 setError('');
 setSuccess('');

 try {
  const selectedAsset = assets.find(a => a.address === asset);
  const amountInWei = ethers.utils.parseUnits(amount, selectedAsset.decimals);
```

```
      // First approve token spending
      await nexus.approveToken(asset, amountInWei);

      // Then supply to protocol
      const tx = await nexus.supply({
        asset: asset,
        amount: amountInWei,
        recipient: await nexus.signer.getAddress()
      });

      const receipt = await tx.wait();
      setSuccess(`Successfully supplied ${amount} ${selectedAsset.symbol}! Transaction: ${receipt.transactionHash}`);
      setAmount('');
    } catch (err) {
      setError('Transaction failed: ' + err.message);
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="supply-form">
      <h2>Supply Assets</h2>
```

```
{error && <div className="error">{error}</div>}
{success && <div className="success">{success}</div>}

<form onSubmit={handleSupply}>
  <div className="form-group">
    <label>Asset:</label>
    <select
      value={asset}
      onChange={(e) => setAsset(e.target.value)}
      required
    >
      <option value="">Select an asset</option>
      {assets.map((a) => (
        <option key={a.address} value={a.address}>
          {a.symbol} - APY: {(a.liquidityRate * 100).toFixed(2)}%
        </option>
      ))}
    </select>
  </div>

  <div className="form-group">
    <label>Amount:</label>
    <input
      type="number"
      value={amount}
      onChange={(e) => setAmount(e.target.value)}
```

```
            min="0"
            step="any"
            required
          />
        </div>

        <button type="submit" disabled={loading}>
          {loading ? 'Processing...' : 'Supply'}
        </button>
      </form>
    </div>
  );
}

export default SupplyForm;
```

Python Integration Example

python

Copy

```
from web3 import Web3
from eth_account import Account
import json
import os

# Load ABI
with open('nexus_lending_pool_abi.json', 'r') as f:
    LENDING_POOL_ABI = json.load(f)
```

```
with open('erc20_abi.json', 'r') as f:
    ERC20_ABI = json.load(f)

# Connect to Ethereum node
web3 = Web3(Web3.HTTPProvider(os.environ.get('INFURA_URL')))

# Contract addresses
LENDING_POOL_ADDRESS = '0x7Fc66500c84A76Ad7e9c93437bFc5Ac33E2DDaE9'
USDC_ADDRESS = '0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48'

# Initialize contracts
lending_pool = web3.eth.contract(address=LENDING_POOL_ADDRESS, abi=LENDING_POOL_ABI)
usdc = web3.eth.contract(address=USDC_ADDRESS, abi=ERC20_ABI)

# Load account (never hardcode private keys in production)
private_key = os.environ.get('PRIVATE_KEY')
account = Account.from_key(private_key)
wallet_address = account.address

def supply_usdc(amount_decimal):
    """
    Supply USDC to the Nexus protocol

    Args:
        amount_decimal: Amount in USDC (e.g., 100.0 for 100 USDC)
    """
    # Convert to wei (USDC has 6 decimals)
    amount = int(amount_decimal * 10**6)

    # Check balance
    balance = usdc.functions.balanceOf(wallet_address).call()
    if balance < amount:
        raise ValueError(f"Insufficient USDC balance. Have {balance/10**6}, need {amount_decimal}")

    # 1. Approve token spending
    approve_tx = usdc.functions.approve(LENDING_POOL_ADDRESS, amount).build_transaction({
        'from': wallet_address,
        'nonce': web3.eth.get_transaction_count(wallet_address),
        'gas': 150000,
        'gasPrice': web3.to_wei('50', 'gwei')
    })

    signed_approve_tx = web3.eth.account.sign_transaction(approve_tx, private_key)
    approve_tx_hash = web3.eth.send_raw_transaction(signed_approve_tx.rawTransaction)
    approve_receipt = web3.eth.wait_for_transaction_receipt(approve_tx_hash)

    print(f"Approval successful: {approve_receipt.transactionHash.hex()}")

    # 2. Supply to protocol
    supply_tx = lending_pool.functions.deposit(
        USDC_ADDRESS,
        amount,
        wallet_address,
        0  # referral code
    ).build_transaction({
        'from': wallet_address,
        'nonce': web3.eth.get_transaction_count(wallet_address),
        'gas': 250000,
        'gasPrice': web3.to_wei('50', 'gwei')
    })

    signed_supply_tx = web3.eth.account.sign_transaction(supply_tx, private_key)
    supply_tx_hash = web3.eth.send_raw_transaction(signed_supply_tx.rawTransaction)
    supply_receipt = web3.eth.wait_for_transaction_receipt(supply_tx_hash)

    print(f"Supply successful: {supply_receipt.transactionHash.hex()}")
    return supply_receipt.transactionHash.hex()

def get_user_data():
    """Get user account data from the protocol"""
    data = lending_pool.functions.getUserAccountData(wallet_address).call()

    return {
        'totalCollateralETH': web3.from_wei(data[0], 'ether'),
        'totalDebtETH': web3.from_wei(data[1], 'ether'),
        'availableBorrowsETH': web3.from_wei(data[2], 'ether'),
        'currentLiquidationThreshold': data[3] / 10000,  # basis points to percentage
        'ltv': data[4] / 10000,  # basis points to percentage
        'healthFactor': data[5] / 1e18
    }

if __name__ == "__main__":
    # Example: Supply 100 USDC
    try:
        tx_hash = supply_usdc(100.0)
        print(f"Supply transaction: https://etherscan.io/tx/{tx_hash}")

        # Display user data after supply
        user_data = get_user_data()
        print("\nUser Account Data:")
        for key, value in user_data.items():
            print(f"{key}: {value}")

    except Exception as e:
        print(f"Error: {str(e)}")
```

Ruby Integration Example

ruby

Copy

```
require 'eth'
require 'json'
require 'httparty'
```

```
class NexusClient
  LENDING_POOL_ADDRESS = '0x7Fc66500c84A76Ad7e9c93437bFc5Ac33E2DDaE9'

  def initialize(provider_url, private_key)
    @client = Eth::Client.new(provider_url)
    @key = Eth::Key.new(priv: private_key)
    @address = @key.address

    # Load ABIs
    lending_pool_abi = JSON.parse(File.read('nexus_lending_pool_abi.json'))
    @lending_pool = Eth::Contract.from_abi(name: 'LendingPool', address: LENDING_POOL_ADDRESS, abi: lending_pool_abi)

    puts "Initialized with address: #{@address}"
  end

  def get_reserve_data(asset_address)
    data = @client.call(@lending_pool, 'getReserveData', asset_address)

    {
      utilization_rate: data[0] / 1e27,
      availability_liquidity: data[1],
      total_stable_debt: data[2],
      total_variable_debt: data[3],
      liquidity_rate: data[4] / 1e27,
      variable_borrow_rate: data[5] / 1e27,
      stable_borrow_rate: data[6] / 1e27,
      average_stable_rate: data[7] / 1e27,
      liquidity_index: data[8] / 1e27,
      variable_borrow_index: data[9] / 1e27,
      last_update_timestamp: data[10]
    }
  end

  def get_user_account_data
    data = @client.call(@lending_pool, 'getUserAccountData', @address)

    {
      total_collateral_eth: data[0] / 1e18,
      total_debt_eth: data[1] / 1e18,
      available_borrows_eth: data[2] / 1e18,
      current_liquidation_threshold: data[3] / 10000,
      ltv: data[4] / 10000,
      health_factor: data[5] / 1e18
    }
  end

  def supply_asset(asset_address, amount, token_decimals = 18)
    # First approve token spending
    erc20_abi = JSON.parse(File.read('erc20_abi.json'))
    token = Eth::Contract.from_abi(name: 'ERC20', address: asset_address, abi: erc20_abi)
```

```ruby
    # Convert amount to wei equivalent
    amount_in_wei = (amount * 10**token_decimals).to_i

    # Approve
    approve_data = token.approve.encode_data(LENDING_POOL_ADDRESS, amount_in_wei)
    approve_tx = build_transaction(asset_address, approve_data)
    approve_receipt = send_transaction(approve_tx)

    puts "Approval transaction sent: #{approve_receipt['transactionHash']}"

    # Supply
    supply_data = @lending_pool.deposit.encode_data(asset_address, amount_in_wei, @address, 0)
    supply_tx = build_transaction(LENDING_POOL_ADDRESS, supply_data)
    supply_receipt = send_transaction(supply_tx)

    puts "Supply transaction sent: #{supply_receipt['transactionHash']}"
    return supply_receipt
  end

  private

  def build_transaction(to, data)
    nonce = @client.get_nonce(@address)
  tx = {
    from: @address,
    to: to,
    value: 0,
    gas: 250000,
    gas_price: @client.gas_price,
    data: data,
    nonce: nonce
  }

  return tx
end

def send_transaction(tx)
  signed_tx = @key.sign_transaction(tx)
  tx_hash = @client.send_transaction(signed_tx)

  puts "Waiting for transaction to be mined..."
  receipt = nil
  30.times do
    sleep 2
    receipt = @client.get_transaction_receipt(tx_hash)
    break if receipt
  end

  raise "Transaction not mined after 60 seconds" unless receipt
```

```
    return receipt
  end
end

# Usage example
if __FILE__ == $0
  infura_key = ENV['INFURA_KEY']
  private_key = ENV['PRIVATE_KEY']
  provider_url = "https://mainnet.infura.io/v3/#{infura_key}"

  nexus = NexusClient.new(provider_url, private_key)

  # Get account data
  account_data = nexus.get_user_account_data
  puts "Account data:"
  puts account_data.inspect

  # Get reserve data for USDC
  usdc_address = '0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48'
  reserve_data = nexus.get_reserve_data(usdc_address)
  puts "USDC reserve data:"
  puts reserve_data.inspect

  # Supply 100 USDC (USDC has 6 decimals)
  if ARGV[0] == 'supply'
    receipt = nexus.supply_asset(usdc_address, 100, 6)
  puts "Supply completed: #{receipt['transactionHash']}"
 end
end
```

Troubleshooting

This section covers common issues encountered when integrating with the Nexus protocol.

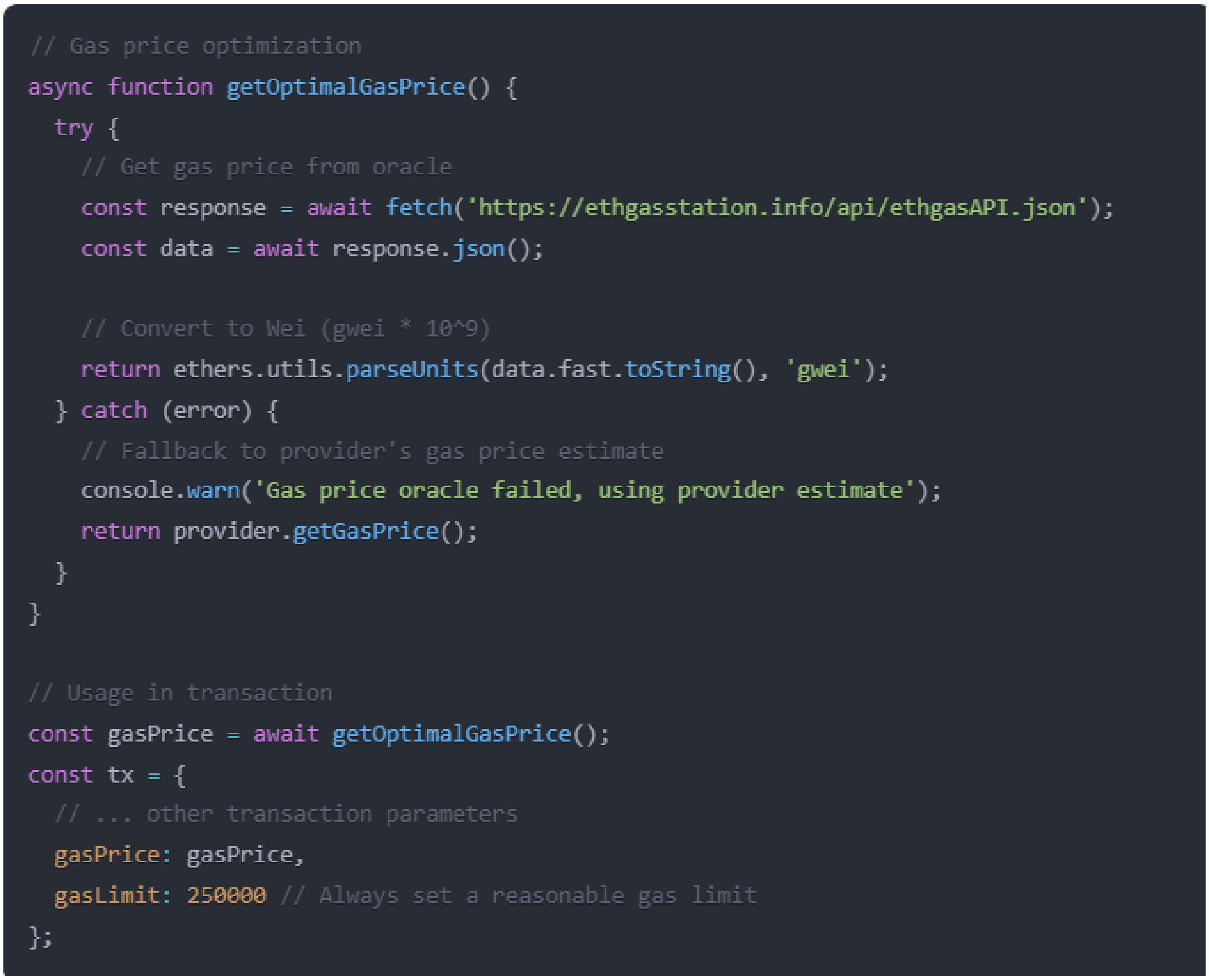
Transaction Errors

Error	Possible Cause	Solution
Insufficient allowance	Token approval transaction failed or is too low	Check token approval status and increase allowance if needed
Health factor below threshold	Attempting to borrow too much against collateral	Supply more collateral or borrow less
Not enough liquidity	Trying to borrow more than available in the pool	Borrow less or try a different asset
Slippage exceeded	Price movement during transaction	Increase slippage tolerance or try again
Reverted	General transaction failure	Check parameters and transaction data

Gas Optimization

To optimize gas usage when interacting with the protocol:

1. **Batch operations**: Use the batch function when performing multiple operations
2. **Gas price strategy**: Use a gas price oracle to determine optimal gas price
3. **Gas estimation**: Always estimate gas before sending transactions
4. **Nonce management**: Properly manage nonces to avoid stuck transactions

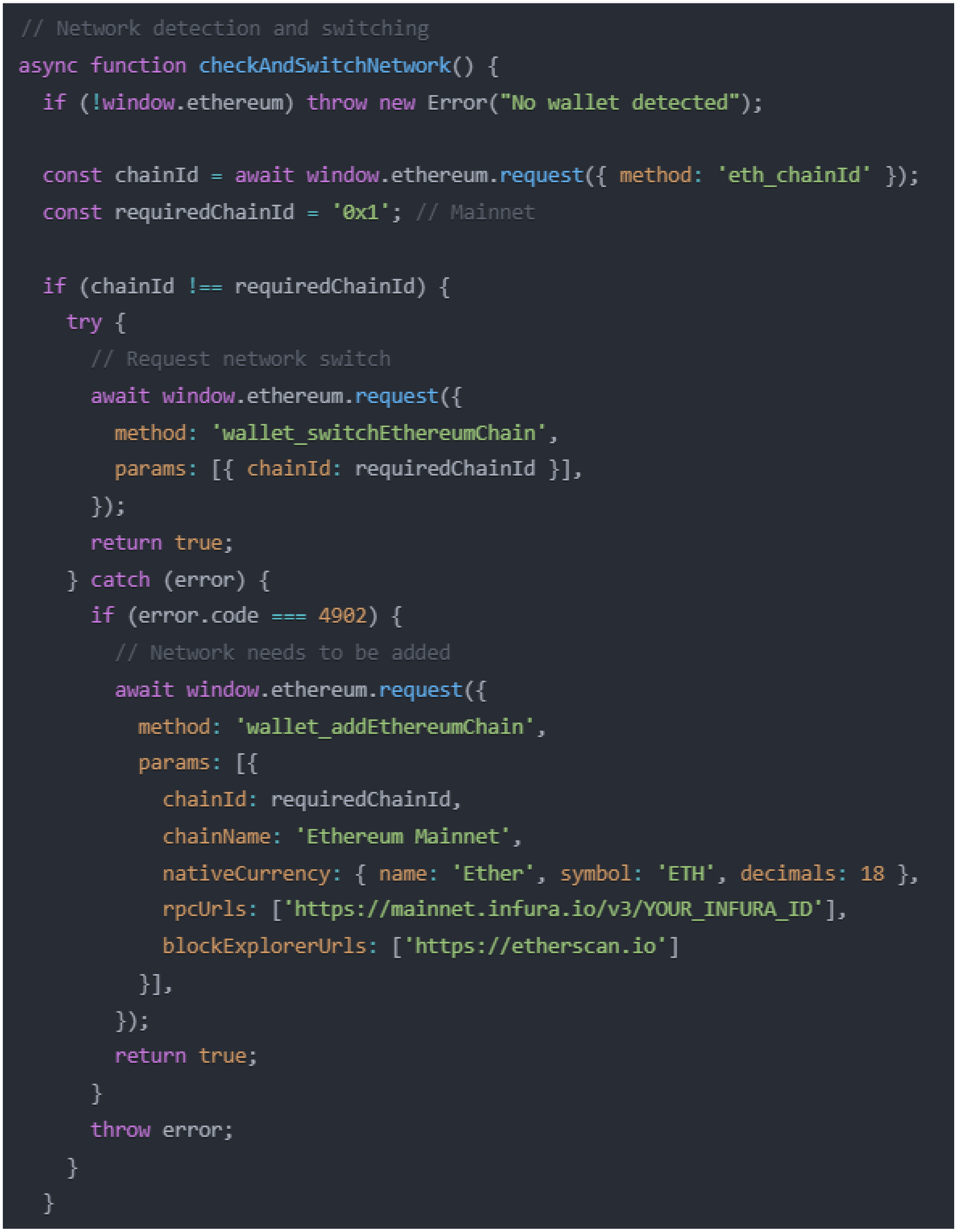


```javascript
// Gas price optimization
async function getOptimalGasPrice() {
  try {
    // Get gas price from oracle
    const response = await fetch('https://ethgasstation.info/api/ethgasAPI.json');
    const data = await response.json();

    // Convert to Wei (gwei * 10^9)
    return ethers.utils.parseUnits(data.fast.toString(), 'gwei');
  } catch (error) {
    // Fallback to provider's gas price estimate
    console.warn('Gas price oracle failed, using provider estimate');
    return provider.getGasPrice();
  }
}

// Usage in transaction
const gasPrice = await getOptimalGasPrice();
const tx = {
  // ... other transaction parameters
  gasPrice: gasPrice,
  gasLimit: 250000 // Always set a reasonable gas limit
};
```

Wallet Connection Issues

Common wallet connection issues and solutions:

1. **MetaMask not detected**:
   - Ensure MetaMask extension is installed and unlocked
   - Add a detection loop that checks for window.ethereum periodically
2. **Wrong network**:
   - Detect network mismatch and prompt user to switch
   - Implement automatic network switching (requires user permission)



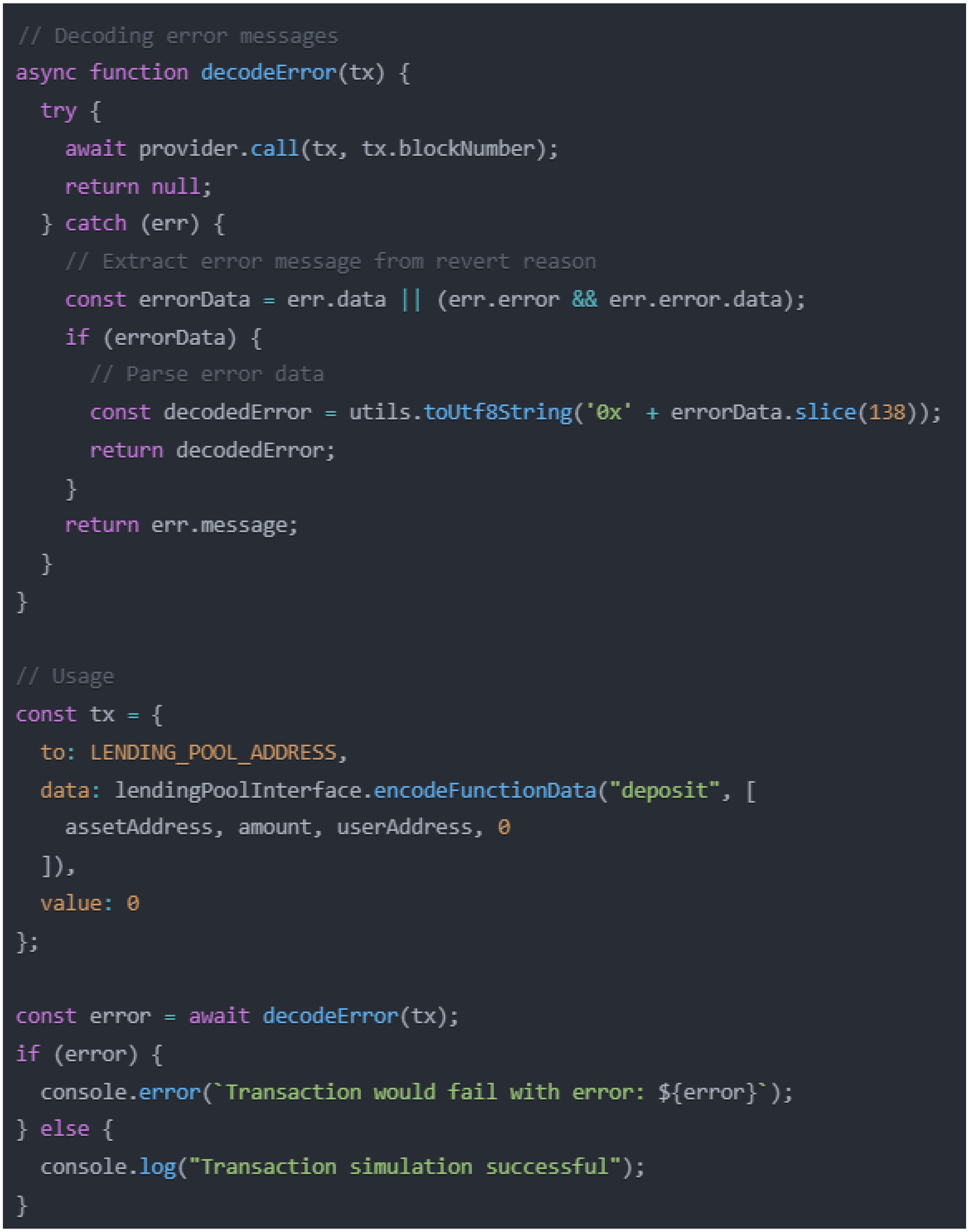
```
// Network detection and switching
async function checkAndSwitchNetwork() {
  if (!window.ethereum) throw new Error("No wallet detected");

  const chainId = await window.ethereum.request({ method: 'eth_chainId' });
  const requiredChainId = '0x1'; // Mainnet

  if (chainId !== requiredChainId) {
    try {
      // Request network switch
      await window.ethereum.request({
        method: 'wallet_switchEthereumChain',
        params: [{ chainId: requiredChainId }],
      });
      return true;
    } catch (error) {
      if (error.code === 4902) {
        // Network needs to be added
        await window.ethereum.request({
          method: 'wallet_addEthereumChain',
          params: [{
            chainId: requiredChainId,
            chainName: 'Ethereum Mainnet',
            nativeCurrency: { name: 'Ether', symbol: 'ETH', decimals: 18 },
            rpcUrls: ['https://mainnet.infura.io/v3/YOUR_INFURA_ID'],
            blockExplorerUrls: ['https://etherscan.io']
          }],
        });
        return true;
      }
      throw error;
    }
  }
```

return true;

Debugging Smart Contract Interactions

For debugging contract interactions:

1. **Event logging**: Monitor emitted events for transaction status
2. **Transaction simulation**: Use services like Tenderly to simulate transactions before sending
3. **Error decoding**: Decode revert reasons for better error messages



```
// Decoding error messages
async function decodeError(tx) {
  try {
    await provider.call(tx, tx.blockNumber);
    return null;
  } catch (err) {
    // Extract error message from revert reason
    const errorData = err.data || (err.error && err.error.data);
    if (errorData) {
      // Parse error data
      const decodedError = utils.toUtf8String('0x' + errorData.slice(138));
      return decodedError;
    }
    return err.message;
  }
}

// Usage
const tx = {
  to: LENDING_POOL_ADDRESS,
  data: lendingPoolInterface.encodeFunctionData("deposit", [
    assetAddress, amount, userAddress, 0
  ]),
  value: 0
};

const error = await decodeError(tx);
if (error) {
  console.error(`Transaction would fail with error: ${error}`);
} else {
  console.log("Transaction simulation successful");
}
```

Versioning and Updates

The Nexus protocol follows semantic versioning (SemVer) for its smart contracts and SDK, with version numbers in the format of MAJOR.MINOR.PATCH.

Contract Versioning

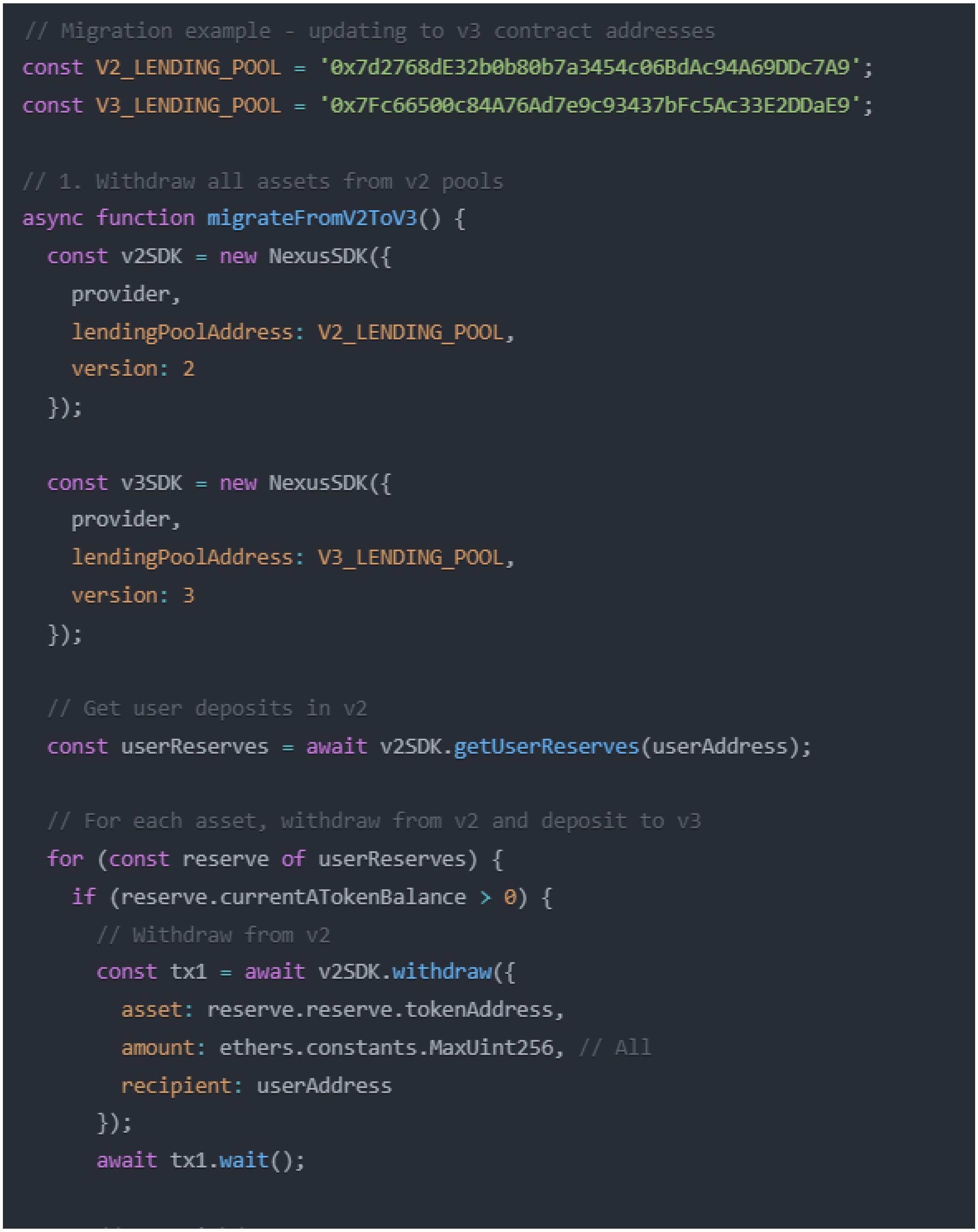
Version	Release Date	Key Changes	Status
3.0.0	2025-01-15	Governance token integration, yield strategies	Current
2.1.0	2024-09-10	Flash loan fee adjustment, improved interest rate model	Supported
2.0.0	2024-05-22	Multi-chain support, gas optimizations	Supported
1.0.0	2023-11-08	Initial mainnet release	Deprecated

Migration Guides

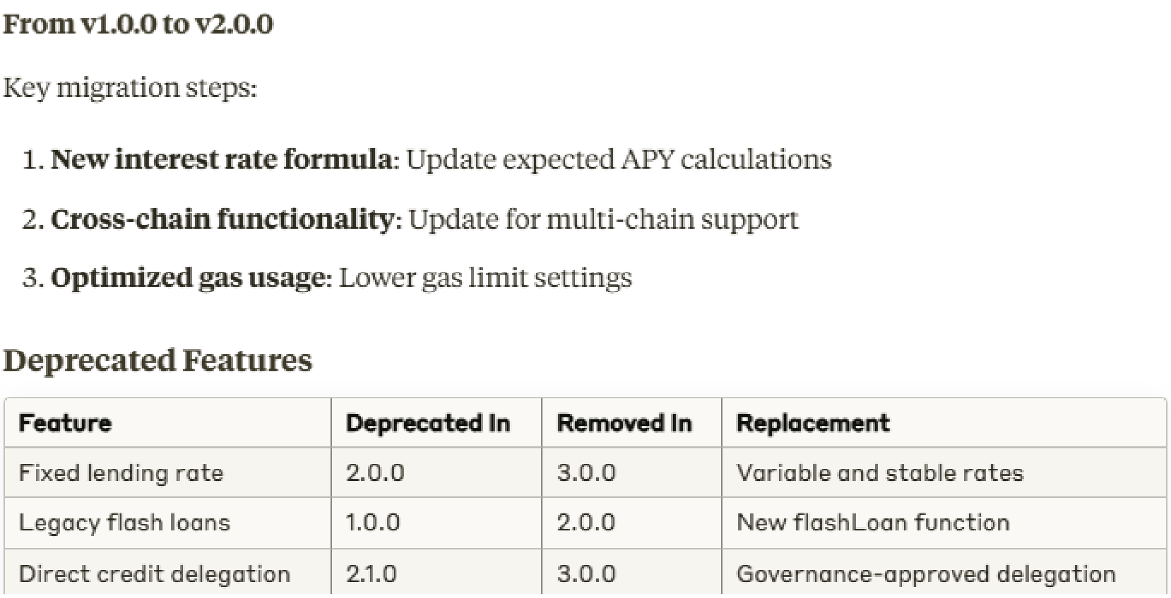
From v2.x to v3.0.0

The v3.0.0 release includes breaking changes to the lending pool interface. Key migration steps:

1. **Updated contract addresses**: All lending pools have new addresses
2. **Interest rate model changes**: Recalculate expected rates with the new model
3. **New governance features**: Integrate with governance token if needed



```
// Migration example - updating to v3 contract addresses
const V2_LENDING_POOL = '0x7d2768dE32b0b80b7a3454c06BdAc94A69DDc7A9';
const V3_LENDING_POOL = '0x7Fc66500c84A76Ad7e9c93437bFc5Ac33E2DDaE9';

// 1. Withdraw all assets from v2 pools
async function migrateFromV2ToV3() {
  const v2SDK = new NexusSDK({
    provider,
    lendingPoolAddress: V2_LENDING_POOL,
    version: 2
  });

  const v3SDK = new NexusSDK({
    provider,
    lendingPoolAddress: V3_LENDING_POOL,
    version: 3
  });

  // Get user deposits in v2
  const userReserves = await v2SDK.getUserReserves(userAddress);

  // For each asset, withdraw from v2 and deposit to v3
  for (const reserve of userReserves) {
    if (reserve.currentATokenBalance > 0) {
      // Withdraw from v2
      const tx1 = await v2SDK.withdraw({
        asset: reserve.reserve.tokenAddress,
        amount: ethers.constants.MaxUint256, // All
        recipient: userAddress
      });
      await tx1.wait();
```



From v1.0.0 to v2.0.0

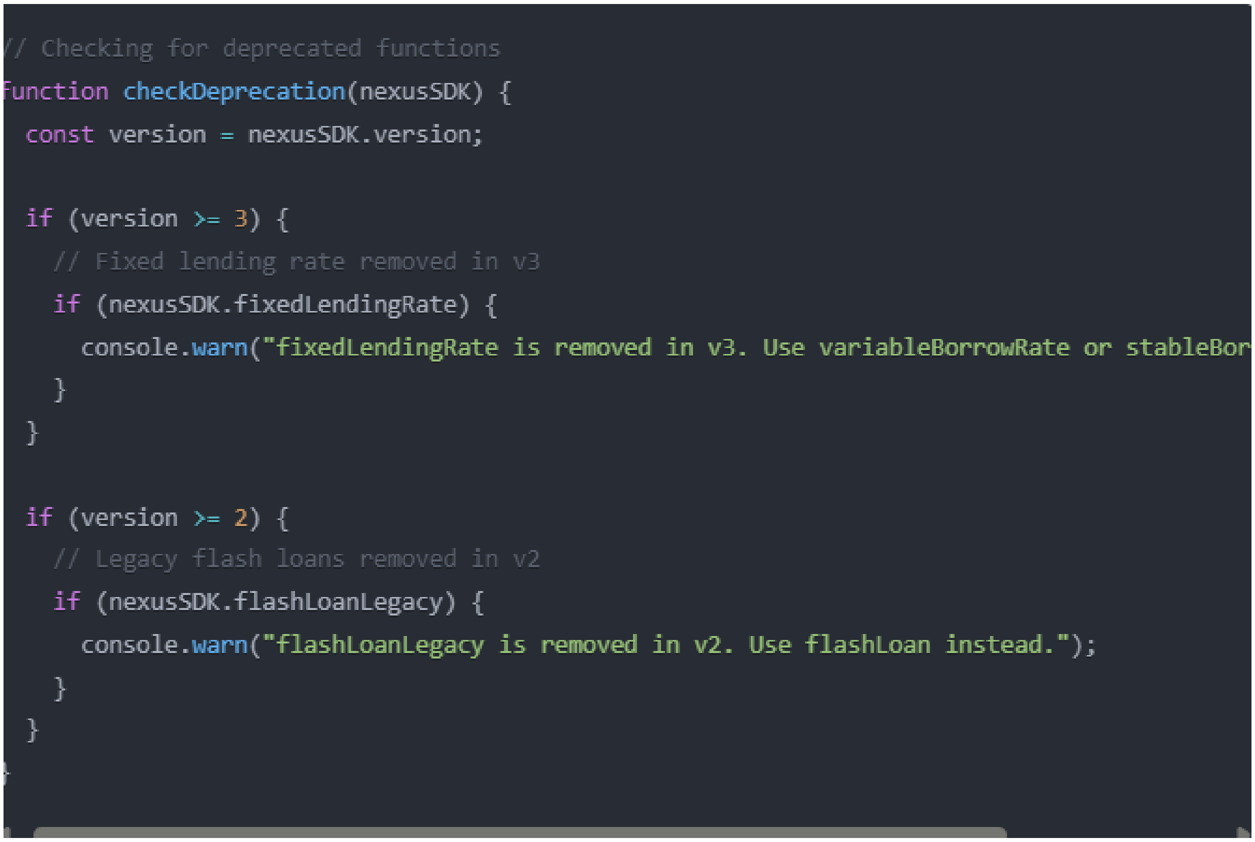
Key migration steps:

1. **New interest rate formula**: Update expected APY calculations
2. **Cross-chain functionality**: Update for multi-chain support
3. **Optimized gas usage**: Lower gas limit settings

Deprecated Features

Feature	Deprecated In	Removed In	Replacement
Fixed lending rate	2.0.0	3.0.0	Variable and stable rates
Legacy flash loans	1.0.0	2.0.0	New flashLoan function
Direct credit delegation	2.1.0	3.0.0	Governance-approved delegation

Example of handling deprecated functions:



```
// Checking for deprecated functions
function checkDeprecation(nexusSDK) {
  const version = nexusSDK.version;

  if (version >= 3) {
    // Fixed lending rate removed in v3
    if (nexusSDK.fixedLendingRate) {
      console.warn("fixedLendingRate is removed in v3. Use variableBorrowRate or stableBor
    }
  }

  if (version >= 2) {
    // Legacy flash loans removed in v2
    if (nexusSDK.flashLoanLegacy) {
      console.warn("flashLoanLegacy is removed in v2. Use flashLoan instead.");
    }
  }
}
```

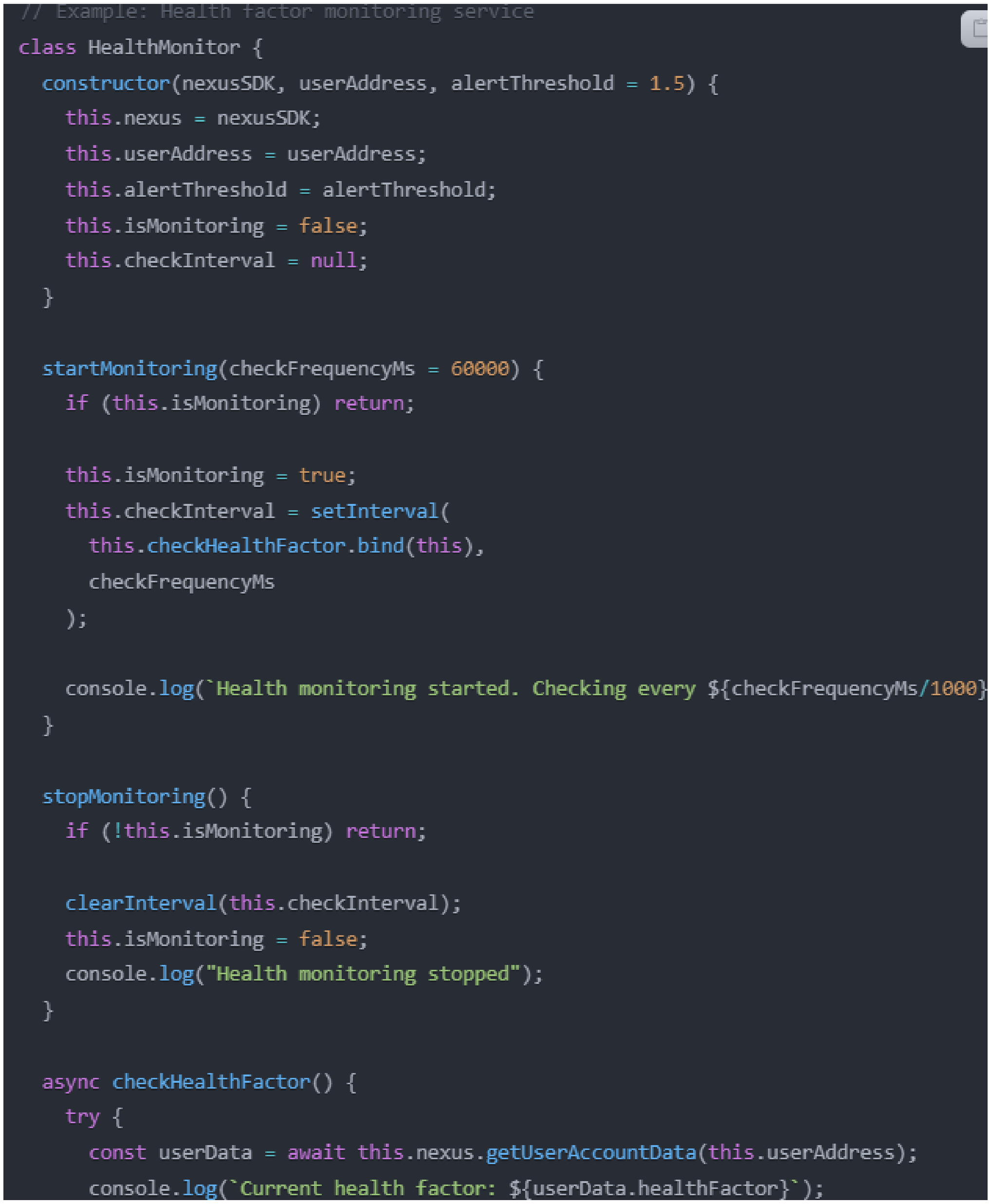
Glossary

Term	Definition
APY	Annual Percentage Yield. The effective annual rate of return taking into account compounding interest.
Collateral	Assets deposited by users that secure borrowed positions.
Flashloan	A type of uncollateralized loan that must be borrowed and repaid within a single transaction.
Health Factor	A numeric representation of the safety of a borrowed position relative to the collateral provided.
Liquidation	The process of selling a borrower's collateral to repay their debt when their health factor falls below 1.
Liquidation Threshold	The percentage of collateral value at which a position is considered undercollateralized and can be liquidated.
LTV (Loan-to-Value)	The ratio of borrowed amount to collateral value, expressed as a percentage.
nToken	Interest-bearing tokens representing deposits in the Nexus protocol.
Stable Rate	A fixed interest rate that can still be rebalanced under certain conditions.
Utilization Rate	The percentage of deposited funds currently borrowed by users.
Variable Rate	An interest rate that changes based on the utilization rate of the pool.
Gas	The computational cost of executing transactions on the Ethereum network.
Smart Contract	Self-executing code deployed on a blockchain that automatically implements the terms of an agreement.
Wallet	Software that stores private keys and allows interaction with the blockchain.

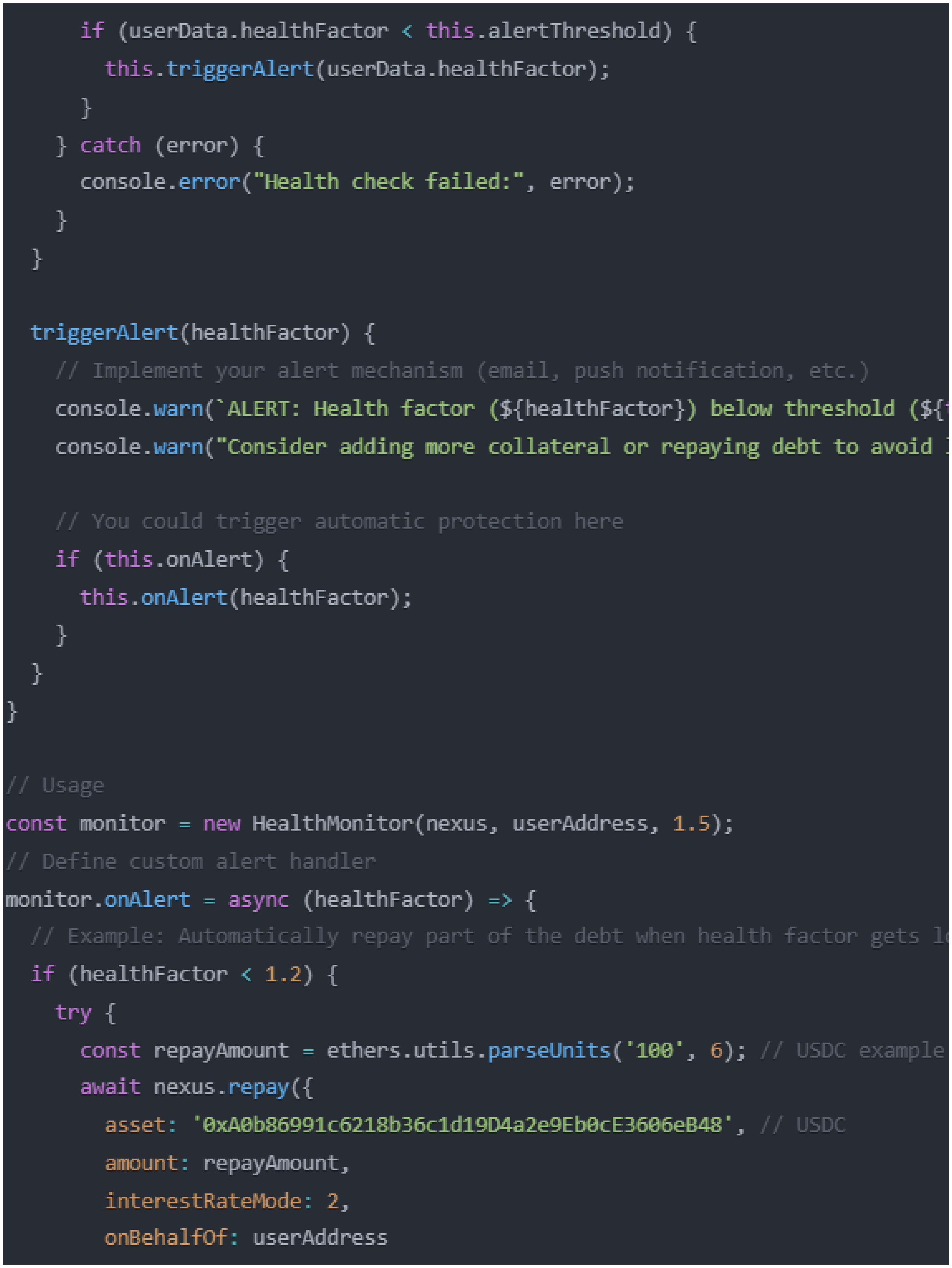
Important Security Considerations

When integrating with the Nexus protocol, keep these security best practices in mind:

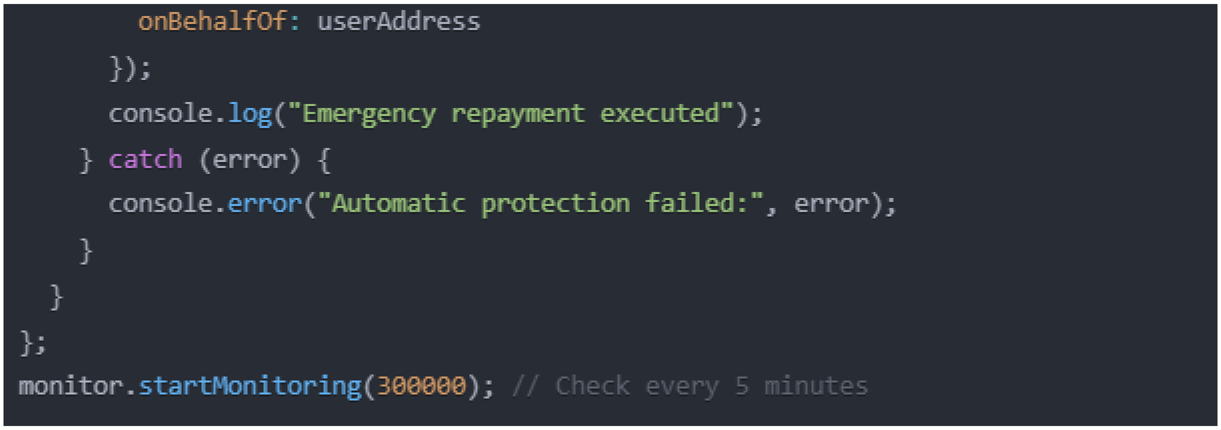
1. **Always verify transactions**: Confirm transaction details before signing
2. **Monitor health factor**: Regularly check positions to avoid liquidation
3. **Set sensible gas limits**: Prevent out-of-gas errors
4. **Implement reentrancy protection**: Guard against reentrancy attacks in smart contracts
5. **Audit integrations**: Have third-party security audits for production implementations
6. **Keep private keys secure**: Never expose private keys in client-side code
7. **Stay updated**: Always use the latest SDK version with security patches



```javascript
// Example: Health factor monitoring service
class HealthMonitor {
  constructor(nexusSDK, userAddress, alertThreshold = 1.5) {
    this.nexus = nexusSDK;
    this.userAddress = userAddress;
    this.alertThreshold = alertThreshold;
    this.isMonitoring = false;
    this.checkInterval = null;
  }

  startMonitoring(checkFrequencyMs = 60000) {
    if (this.isMonitoring) return;

    this.isMonitoring = true;
    this.checkInterval = setInterval(
      this.checkHealthFactor.bind(this),
      checkFrequencyMs
    );

    console.log(`Health monitoring started. Checking every ${checkFrequencyMs/1000}
  }

  stopMonitoring() {
    if (!this.isMonitoring) return;

    clearInterval(this.checkInterval);
    this.isMonitoring = false;
    console.log("Health monitoring stopped");
  }

  async checkHealthFactor() {
    try {
      const userData = await this.nexus.getUserAccountData(this.userAddress);
      console.log(`Current health factor: ${userData.healthFactor}`);
```



```
      if (userData.healthFactor < this.alertThreshold) {
        this.triggerAlert(userData.healthFactor);
      }
    } catch (error) {
      console.error("Health check failed:", error);
    }
  }

  triggerAlert(healthFactor) {
    // Implement your alert mechanism (email, push notification, etc.)
    console.warn(`ALERT: Health factor (${healthFactor}) below threshold (${
    console.warn("Consider adding more collateral or repaying debt to avoid 

    // You could trigger automatic protection here
    if (this.onAlert) {
      this.onAlert(healthFactor);
    }
  }
}

// Usage
const monitor = new HealthMonitor(nexus, userAddress, 1.5);
// Define custom alert handler
monitor.onAlert = async (healthFactor) => {
  // Example: Automatically repay part of the debt when health factor gets l
  if (healthFactor < 1.2) {
    try {
      const repayAmount = ethers.utils.parseUnits('100', 6); // USDC example
      await nexus.repay({
        asset: '0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48', // USDC
        amount: repayAmount,
        interestRateMode: 2,
        onBehalfOf: userAddress
```



```
        onBehalfOf: userAddress
      });
      console.log("Emergency repayment executed");
    } catch (error) {
      console.error("Automatic protection failed:", error);
    }
  }
};
monitor.startMonitoring(300000); // Check every 5 minutes
```