Contents

GraphQL Token API Programming Guide

Table of Contents

---

1. Introduction

This guide provides comprehensive documentation for the GraphQL Token API, a blockchain-based token management system built on Ethereum. The API enables you to:

- Deploy and manage ERC-20 compatible tokens
- Implement multi-approval task workflows
- Control access through role-based permissions
- Manage token fees and VIP accounts
- Handle KYC and compliance requirements
- Execute secure token operations

The API is built on GraphQL, providing a flexible and type-safe interface for all operations.

---

2. Getting Started

API Endpoint

The default API endpoint is:

```
http://localhost:4000/graphql
```

GraphQL Basics

GraphQL uses two main operation types:

- **Queries**: Read-only operations that fetch data
- **Mutations**: Operations that modify data

All requests are POST requests to the `/graphql` endpoint with a JSON payload containing:

- `query`: The GraphQL query or mutation string
- `variables`: (Optional) Variables used in the query

Making Requests with curl

Basic curl request format:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { version }"
  }'
```

With variables:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query($name: String!) { token { balanceOf(name: $name, account: \"0x123...\")
    "variables": {"name": "MyToken"}
  }'
```

---

3. Core Concepts

Task Lifecycle

Many operations create **tasks** that require approval before execution:

1. **Pending**: Task created, waiting for approvals
2. **Ready**: Minimum approvals met, ready to execute
3. **Executing**: Task is being executed on blockchain
4. **Executed**: Task completed successfully
5. **Aborted**: Task was cancelled
6. **Expired**: Task exceeded expiration time

Multi-Approval System

Tasks can require multiple approvals:

- **minApprovals**: Minimum number of approvals needed

- **Signatures**: Off-chain approval signatures from authorized accounts
- **Auto-execute**: Tasks can auto-execute when approvals are met

Role-Based Access Control

Eight built-in roles control access:

- `DEFAULT_ADMIN_ROLE`: Overall admin
- `UPGRADER_ROLE`: Can upgrade contracts
- `MINT_ADMIN_ROLE`: Can mint tokens
- `BURN_ADMIN_ROLE`: Can burn tokens
- `COMPLIANCE_ADMIN_ROLE`: Can manage permissions
- `FEE_ADMIN_ROLE`: Can configure fees
- `FEE_RECIPIENT_ROLE`: Can withdraw fees
- `PAUSER_ROLE`: Can pause operations

Result Interface

All operations return a result following this pattern:

```
interface Result {
  success: Boolean!      # true if operation succeeded
  message: String        # Human-readable message
  code: String          # Error code (on failure)
  category: String      # Error category
  location: String      # Error location
  stack: [String]       # Error stack trace
  trace: [String]       # Function call trace
  details: Details      # Decoded error details
}
```

---

4. Token Management

Upload Token Code

Upload bytecode for a new token implementation.

Required Role: `UPGRADER_ROLE`

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation($name: String!, $bytecode: String!, $minApprovals: Int!, $opts: TaskI
    "variables": {
      "name": "MyToken",
      "bytecode": "0x6080604052348015610010576000080fd5b50...",
      "minApprovals": 2,
```

```
      "opts": {
        "expiry": 3600,
        "collectDelay": 300,
        "autoExec": true
      }
    }
  }'
```

Response:

```
{
  "data": {
    "token": {
      "uploadTokenCode": {
        "success": true,
        "result": {
          "key": "12345678901234567890",
          "hash": "0xabc..."
        },
        "message": "Task created successfully"
      }
    }
  }
}
```

TaskLifeCycleOptions:

- `expiry`: Seconds until task expires (default: system setting)
- `collectDelay`: Seconds to wait before task can be collected (default: 0)
- `autoExec`: Auto-execute when approvals met (default: false)

Deploy Token

Deploy a new token with specified settings.

Required Role: `DEFAULT_ADMIN_ROLE`

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation($name: String!, $settings: TokenSettingsInput!, $minApprovals: Int!)
    "variables": {
      "name": "MyToken",
      "settings": {
        "symbol": "MTK",
        "decimals": 18,
        "totalSupply": "1000000000000000000000000",
        "owner": "0x742d35Cc6634C0532925a3b844Bc9e7595f0bEb",
        "privatePermission": true,
```

```
      "tokenPermission": "0x0000000000000000000000000000000000000000",
      "withMask": true,
      "withKYC": false,
      "accountMan": "0x0000000000000000000000000000000000000000",
      "enableFee": false,
      "feeSettings": {
        "enabled": false,
        "vipRate": 0,
        "defaultRate": 0,
        "minFee": "0",
        "maxFee": "0"
      },
      "pausable": true,
      "burnable": true,
      "mintable": true
    },
    "minApprovals": 1
  }
}'
```

TokenSettingsInput Fields:

Token Info:

- `symbol`: Token symbol (e.g., "ETH", "BTC")
- `decimals`: Number of decimals (typically 18)
- `totalSupply`: Initial supply in atomic units (BigInt)
- `owner`: Token owner address

Permission:

- `privatePermission`: Use private (token-specific) permission contract
- `tokenPermission`: Address of existing permission contract (use `0x0` to auto-create)

Permission Mode:

- `withMask`: Enable permission mask mode (send/receive flags)
- `withKYC`: Enable KYC level mode (0-4 levels)

Account Management:

- `accountMan`: Address of AccountMan contract (use `0x0` to auto-create)

Token Fee:

- `enableFee`: Enable fee collection on transfers
- `feeSettings`: Fee configuration (if enableFee is true)

Safety Flags:

- `pausable`: Allow pausing token operations

- `burnable`: Allow burning tokens
- `mintable`: Allow minting new tokens

Migrate Token

Migrate token to a new implementation version.

Required Role: `UPGRADER_ROLE`

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { token { migrate(name: \"MyToken\", minApprovals: 2) { success resul
  }'
```

Query Token Information

Get token balance:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { token { balanceOf(name: \"MyToken\", account: \"0x742d35Cc6634C053292
  }'
```

Get total supply:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { token { totalSupply(name: \"MyToken\") { success result } } }"
  }'
```

Get token version:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { token { version(name: \"MyToken\") { success result { major minor pato
  }'
```

Get token symbol:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { token { symbol(name: \"MyToken\") { success result } } }"
  }'
```

Get token decimals:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { token { decimals(name: \"MyToken\") { success result } } }"
  }'
```

Get token settings:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { token { settings(name: \"MyToken\") { success result { symbol decimals
  }'
```

Get token address:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { token { getTokenAddress(name: \"MyToken\") { success result } } }"
  }'
```

Get total tokens count:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { token { totalTokens { success result } } }"
  }'
```

Get token info by index:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { token { getTokenInfoByIndex(index: 0) { success result { name address
  }'
```

Set Token Symbol

Change the token symbol.

Required Role: `DEFAULT_ADMIN_ROLE`

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { token { setSymbol(name: \"MyToken\", symbol: \"NEWMTK\", minApprova
  }'
```

Set Token Decimals

Change the token decimals.

Required Role: `DEFAULT_ADMIN_ROLE`

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { token { setDecimals(name: \"MyToken\", decimals: 6, minApprovals: 1
  }'
```

Mint Tokens

Mint new tokens to an address.

Required Role: `MINT_ADMIN_ROLE`

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { token { mint(name: \"MyToken\", to: \"0x742d35Cc6634C0532925a3b844E
  }'
```

Burn Tokens

Burn tokens from an address.

Required Role: `BURN_ADMIN_ROLE`

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { token { burn(name: \"MyToken\", from: \"0x742d35Cc6634C0532925a3b84
  }'
```

Transfer Operations

Send tokens from owner:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { token { sendFromOwner(name: \"MyToken\", to: \"0x123...\", amount:
  }'
```

Send tokens (requires private key):

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
```

```
    "query": "mutation { token { send(name: \"MyToken\", privateKey: \"0xabc...\", to: \"0x1
  }'
```

---

5. Task System

Task Status

Task status values:

- `Pending`: Waiting for approvals
- `Ready`: Has minimum approvals, ready to execute
- `Executing`: Being executed on blockchain
- `Executed`: Completed successfully
- `Aborted`: Cancelled
- `Expired`: Exceeded expiration time

Get Task Info

Get detailed task information:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { task { info(key: \"12345678901234567890\") { success result { taskId s
  }'
```

Get task status:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { task { status(address: \"0x123...\") { success result } } }"
  }'
```

Get multi-approval task status:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { task { multiApprovalTaskStatus(address: \"0x123...\") { success result
  }'
```

Sign Task

Generate signature for task approval:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
```

```
    "query": "query { task { sign(taskId: \"0xabc...\", privateKey: \"0x123...\") { success
  }'
```

Approve Task

Submit approvals for a task:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation($address: String!, $sigs: [String!]!) { task { approve(address: $addr
    "variables": {
      "address": "0x123...",
      "signatures": ["0xsig1...", "0xsig2..."]
    }
  }'
```

Execute Task

Execute a ready task:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { task { execute(address: \"0x123...\") { success result message } }
  }'
```

Collect Task

Collect a completed task (returns it to the pool for reuse):

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { task { collect(address: \"0x123...\") { success result message } }
  }'
```

Abort Task

Abort a pending task:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { task { abort(keyToAbort: \"12345678901234567890\", minApprovals: 1
  }'
```

---

6. Role Management

Understanding Roles

Eight built-in roles:

```
enum Role {
  DEFAULT_ADMIN_ROLE
  UPGRADER_ROLE
  MINT_ADMIN_ROLE
  BURN_ADMIN_ROLE
  COMPLIANCE_ADMIN_ROLE
  FEE_ADMIN_ROLE
  FEE_RECIPIENT_ROLE
  PAUSER_ROLE
}
```

Check Root

Check if address is root:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { role { isRoot(name: \"MyToken\", account: \"0x123...\") { success resu
  }'
```

Get root address:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { role { getRoot(name: \"MyToken\") { success result } } }"
  }'
```

Get Role Members

Check if account has role:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { role { isRole(name: \"MyToken\", account: \"0x123...\", role: MINT_ADM
  }'
```

Get all members of a role:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
```

```
    "query": "query { role { getRoleMembers(name: \"MyToken\", role: MINT_ADMIN_ROLE) { succ
  }'
```

Change Root

Transfer root to new address:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { role { changeRoot(name: \"MyToken\", newRoot: \"0x123...\", minAppr
  }'
```

Set Role

Grant or revoke role:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { role { setRole(name: \"MyToken\", account: \"0x123...\", role: MINT
  }'
```

---

7. Permission Control

Permission Modes

Two permission modes can be enabled:

Permission Mask Mode (`withMask`):

- Controls send/receive permissions per account
- `canSend`: Account can send tokens
- `canReceive`: Account can receive tokens

KYC Mode (`withKYC`):

- Assigns KYC levels to accounts (0-4)
- Controls transfers based on compliance levels

```
enum KYCLevel {
  KYC0  # Blocked - cannot send or receive
  KYC1  # View only - can hold but not transfer
  KYC2  # Standard - normal operations allowed
  KYC3  # Accredited - enhanced privileges
  KYC4  # Institutional - highest level
}
```

Create Permission

Create a permission contract:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { permission { createPermission(name: \"MyToken\", withKYC: true, wit
  }'
```

Enable/Disable Permission

Enable permission for token:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { permission { enablePermission(name: \"MyToken\", enable: true, minA
  }'
```

Check if permission is enabled:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { permission { isPermissionEnabled(name: \"MyToken\") { success result }
  }'
```

Set Permission Mode

Set permission work mode:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { permission { setPermission(name: \"MyToken\", withKYC: true, withMa
  }'
```

Get permission mode:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { permission { getMode(name: \"MyToken\") { success result { withKYC wit
  }'
```

Check if token has private permission:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
```

```
    "query": "query { permission { hasPrivatePermission(name: \"MyToken\") { success result
  }'
```

Get default permission contract address:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { permission { getDefaultPermission { success result } } }"
  }'
```

Get token permission contract address:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { permission { getTokenPermission(name: \"MyToken\") { success result }
  }'
```

Lock/Unlock Accounts

Lock account in default permission:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { permission { lockAccount(account: \"0x123...\", lock: true, minAppr
  }'
```

Lock account in token permission:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { permission { lockTokenAccount(name: \"MyToken\", account: \"0x123.
  }'
```

Check if account is locked (default permission):

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { permission { isAccountLocked(account: \"0x123...\") { success result
  }'
```

Check if account is locked (token permission):

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
```

```
    "query": "query { permission { isTokenAccountLocked(name: \"MyToken\", account: \"0x123
  }'
```

Set KYC Level

Set KYC level for account (default permission):

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { permission { setKYCLevel(account: \"0x123...\", level: KYC2, minApp
  }'
```

Set KYC level for account (token permission):

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { permission { setTokenKYCLevel(name: \"MyToken\", account: \"0x123.
  }'
```

Get KYC level:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { permission { getKYCLevel(name: \"MyToken\", account: \"0x123...\") { s
  }'
```

Set Permission Mask

Set permission mask (default permission):

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation($account: String!, $mask: PermissionMaskInput!, $minApprovals: Int!)
    "variables": {
      "account": "0x123...",
      "mask": {
        "canSend": true,
        "canReceive": true
      },
      "minApprovals": 1
    }
  }'
```

Set permission mask (token permission):

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation($name: String!, $account: String!, $mask: PermissionMaskInput!, $minA
    "variables": {
      "name": "MyToken",
      "account": "0x123...",
      "mask": {
        "canSend": true,
        "canReceive": true
      },
      "minApprovals": 1
    }
  }'
```

Get permission mask:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { permission { getMask(name: \"MyToken\", account: \"0x123...\") { succe
  }'
```

Check Transfer Permission

Check if transfer is allowed:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { permission { canTransfer(name: \"MyToken\", from: \"0x123...\", to: \"
  }'
```

---

8. Fee Management

Get Fee Settings

Get complete fee settings:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { fee { settings(name: \"MyToken\") { success result { enabled vipRate c
  }'
```

Check if fee is enabled:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { fee { isFeeEnabled(name: \"MyToken\") { success result } } }"
  }'
```

Get VIP rate (basis points):

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { fee { getVIPRate(name: \"MyToken\") { success result } } }"
  }'
```

Get default rate (basis points):

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { fee { getDefaultRate(name: \"MyToken\") { success result } } }"
  }'
```

Get account-specific rate:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { fee { getAccountRate(name: \"MyToken\", account: \"0x123...\") { succe
  }'
```

Get minimum fee:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { fee { getMinFee(name: \"MyToken\") { success result } } }"
  }'
```

Get maximum fee:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { fee { getMaxFee(name: \"MyToken\") { success result } } }"
  }'
```

Get current fee pool balance:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
```

```
    "query": "query { fee { getCurrentTotalFeeInPool(name: \"MyToken\") { success result } }
  }'
```

Get total collected fees:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { fee { getTotalCollectedFee(name: \"MyToken\") { success result } } }"
  }'
```

Set Fee Settings

Configure fee settings via task.

Required Role: `FEE_ADMIN_ROLE`

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation($name: String!, $settings: TokenFeeSettingsInput!, $minApprovals: Int
    "variables": {
      "name": "MyToken",
      "settings": {
        "enabled": true,
        "vipRate": 10,
        "defaultRate": 50,
        "minFee": "1000000000000000",
        "maxFee": "10000000000000000000"
      },
      "minApprovals": 1
    }
  }'
```

Fee Rate in Basis Points:

- 1 basis point = 0.01%
- 10 basis points = 0.1%
- 100 basis points = 1%
- Example: `vipRate: 10` = 0.1% fee for VIPs

Enable/Disable Fee

Enable or disable fee collection.

Required Role: `FEE_ADMIN_ROLE`

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
```

```
    "query": "mutation { fee { enableFee(name: \"MyToken\", enable: true, minApprovals: 1) {
  }'
```

Set Account Rate

Set custom fee rate for specific account.

Required Role: `FEE_ADMIN_ROLE`

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { fee { setAccountRate(name: \"MyToken\", account: \"0x123...\", rate
  }'
```

Manage VIP Accounts

Set VIP status for account.

Required Role: `FEE_ADMIN_ROLE`

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { fee { setVIPAccount(name: \"MyToken\", account: \"0x123...\", isVIP
  }'
```

Check VIP status:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { fee { isVIPAccount(name: \"MyToken\", account: \"0x123...\") { success
  }'
```

Calculate Fee

Calculate fee for a transfer:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { fee { calculateFee(name: \"MyToken\", from: \"0x123...\", to: \"0x456.
  }'
```

Withdraw Fee

Withdraw collected fees to recipient.

Required Role: `FEE_RECIPIENT_ROLE`

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { fee { withdrawFee(name: \"MyToken\", to: \"0x123...\", amount: \"10
  }'
```

---

9. Account Operations

Create Account

Create a new Ethereum account:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { account { createAccount { success result { address privateKey } } } }"
  }'
```

Security Warning: Store the private key securely. It cannot be recovered if lost.

Calculate account address from private key:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { account { calculateAddress(privateKey: \"0xabc...\") { success result
  }'
```

Validate address format:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { account { isAddressValid(address: \"0x123...\") { success result } } }
  }'
```

Sign Message

Generate message digest:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { account { digest(message: \"Hello World\") { success result } } }"
  }'
```

Sign a message:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { account { signMessage(message: \"Hello World\", privateKey: \"0xabc...
  }'
```

Sign a digest directly:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { account { signDigest(digest: \"0x123...\", privateKey: \"0xabc...\") {
  }'
```

Recover address from signature (digest):

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { account { recoverAddressFromDigestSignature(digest: \"0x123...\", sign
  }'
```

Recover address from signature (message):

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { account { recoverAddressFromMessageSignature(message: \"Hello World\",
  }'
```

Encrypt/Decrypt Data

Encrypt data with password:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { account { encrypt(privateKey: \"0xabc...\", password: \"mypassword\")
  }'
```

Decrypt data with password:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { account { decrypt(json: \"{...encrypted...}\", password: \"mypassword
  }'
```

Check Balance

Get ETH balance:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { account { balanceOf(account: \"0x123...\") { success result } } }"
  }'
```

Send ETH

Send ETH to address:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { account { send(from: \"0x123...\", privateKey: \"0xabc...\", to: \'
  }'
```

---

10. Maintenance Operations

Freeze/Unfreeze Token

Freeze token (pause all operations):

Required Role: `PAUSER_ROLE`

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { maintenance { freeze(name: \"MyToken\") { success result message }
  }'
```

Unfreeze token:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { maintenance { unfreeze(name: \"MyToken\") { success result message
  }'
```

Check if frozen:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { maintenance { isFrozen(name: \"MyToken\") { success result } } }"
  }'
```

Enable/Disable Transfer

Disable transfers:

Required Role: `PAUSER_ROLE`

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { maintenance { disableTransfer(name: \"MyToken\") { success result m
  }'
```

Enable transfers:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { maintenance { enableTransfer(name: \"MyToken\") { success result me
  }'
```

Check if transfer is enabled:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { maintenance { isTransferEnabled(name: \"MyToken\") { success result } }
  }'
```

---

11. System Information

API Version

Get API version:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { version }"
  }'
```

System Version

Get smart contract system version:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { systemVersion { success result { major minor patch build } } }"
  }'
```

Network Info

Get blockchain network information:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { networkInfo { success result { name chainId blockNumber } } }"
  }'
```

Health Check

Check API health status:

```
curl -X POST http://localhost:4000/graphql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { ping }"
  }'
```

Response values:

- `OK`: Blockchain fully synchronized
- `SYNCING`: Synchronizing with peers
- `INDEXING`: Indexing blockchain data

---

12. Error Handling

All API responses follow the `Result` interface pattern:

Success Response

```
{
  "data": {
    "token": {
      "balanceOf": {
        "success": true,
        "result": "1000000000000000000",
        "message": null
      }
    }
  }
}
```

Error Response

```
{
  "data": {
    "token": {
      "deploy": {
        "success": false,
```

```
        "message": "Insufficient approvals",
        "code": "TASK_NOT_READY",
        "category": "VALIDATION_ERROR",
        "location": "TaskFactory.sol:123",
        "stack": [
          "at TaskFactory.executeTask",
          "at TokenFactory.deploy"
        ],
        "trace": [
          "deploy",
          "executeTask",
          "validateApprovals"
        ],
        "details": {
          "type": "TaskNotReady",
          "name": "InsufficientApprovals",
          "reason": "Required 2 approvals, received 1",
          "selector": "0x12345678",
          "signature": "InsufficientApprovals(uint256,uint256)",
          "fragment": "error InsufficientApprovals(uint256 required, uint256 received)"
        }
      }
    }
  }
}
```


Common Error Categories

- `VALIDATION_ERROR`: Invalid input parameters
- `PERMISSION_ERROR`: Insufficient permissions or role
- `STATE_ERROR`: Invalid contract state
- `TASK_ERROR`: Task-related errors
- `BLOCKCHAIN_ERROR`: Blockchain transaction errors

Error Handling Best Practices

1. **Always Check `success` Field**:

   ```
   const response = await fetch(...);
   const data = await response.json();
   if (!data.data.token.deploy.success) {
     console.error(data.data.token.deploy.message);
     return;
   }
   ```

2. **Use Error Code for Logic**:

   ```
   if (result.code === 'TASK_NOT_READY') {
   ```

```
    // Wait and retry
  } else if (result.code === 'INSUFFICIENT_ROLE') {
    // Request admin approval
  }
```

3. **Log Details for Debugging**:

```
if (!result.success) {
  console.error('Error:', result.message);
  console.error('Location:', result.location);
  console.error('Stack:', result.stack);
  console.error('Details:', result.details);
}
```

---

13. Complete API Reference

Query Operations

Token Queries

- `balanceOf(name, account)` - Get token balance
- `totalSupply(name)` - Get total token supply
- `version(name)` - Get token implementation version
- `symbol(name)` - Get token symbol
- `decimals(name)` - Get token decimals
- `settings(name)` - Get complete token settings
- `getTokenAddress(name)` - Get token contract address
- `totalTokens()` - Get count of deployed tokens
- `getTokenInfoByIndex(index)` - Get token info by index
- `getCurrentTokenVersion(name)` - Get current token version

Task Queries

- `info(key)` - Get task information
- `status(address)` - Get task status
- `multiApprovalTaskStatus(address)` - Get approval count
- `sign(taskId, privateKey)` - Generate task signature

Role Queries

- `isRoot(name, account)` - Check if account is root
- `getRoot(name)` - Get root address
- `isRole(name, account, role)` - Check role membership
- `getRoleMembers(name, role)` - Get all role members

Permission Queries

- `isPermissionEnabled(name)` - Check if permission enabled
- `getMode(name)` - Get permission mode settings
- `hasPrivatePermission(name)` - Check if private permission
- `getDefaultPermission()` - Get default permission address
- `getTokenPermission(name)` - Get token permission address
- `isAccountLocked(account)` - Check if account locked (default)
- `isTokenAccountLocked(name, account)` - Check if account locked (token)
- `getMask(name, account)` - Get permission mask
- `getKYCLevel(name, account)` - Get KYC level
- `canTransfer(name, from, to, amount)` - Check transfer permission

Fee Queries

- `settings(name)` - Get fee settings
- `isFeeEnabled(name)` - Check if fee enabled
- `getVIPRate(name)` - Get VIP fee rate
- `getDefaultRate(name)` - Get default fee rate
- `getAccountRate(name, account)` - Get account fee rate
- `calculateFee(name, from, to, amount)` - Calculate transfer fee
- `isVIPAccount(name, account)` - Check VIP status
- `getMinFee(name)` - Get minimum fee
- `getMaxFee(name)` - Get maximum fee
- `getCurrentTotalFeeInPool(name)` - Get fee pool balance
- `getTotalCollectedFee(name)` - Get total fees collected

Account Queries

- `createAccount()` - Create new account
- `calculateAddress(privateKey)` - Get address from private key
- `isAddressValid(address)` - Validate address format
- `digest(message)` - Generate message digest
- `signMessage(message, privateKey)` - Sign message
- `signDigest(digest, privateKey)` - Sign digest
- `recoverAddressFromDigestSignature(digest, signature)` - Recover signer
- `recoverAddressFromMessageSignature(message, signature)` - Recover signer
- `encrypt(privateKey, password)` - Encrypt private key
- `decrypt(json, password)` - Decrypt private key
- `balanceOf(account)` - Get ETH balance

Maintenance Queries

- `isFrozen(name)` - Check if token frozen
- `isTransferEnabled(name)` - Check if transfer enabled

System Queries

- `version` - Get API version
- `systemVersion` - Get smart contract version
- `networkInfo` - Get blockchain network info
- `ping` - Health check

Mutation Operations

Token Mutations

- `uploadTokenCode(name, code, minApprovals, lifeCycleOpts)` - Upload bytecode
- `deploy(name, settings, minApprovals, lifeCycleOpts)` - Deploy token
- `migrate(name, minApprovals, lifeCycleOpts)` - Migrate to new version
- `setAccountMan(name, accountMan, minApprovals, lifeCycleOpts)` - Set AccountMan
- `sendFromOwner(name, to, amount, minApprovals, lifeCycleOpts)` - Owner transfer
- `send(name, privateKey, to, amount)` - Direct transfer
- `setSymbol(name, symbol, minApprovals, lifeCycleOpts)` - Change symbol
- `setDecimals(name, decimals, minApprovals, lifeCycleOpts)` - Change decimals
- `mint(name, to, amount, minApprovals, lifeCycleOpts)` - Mint tokens
- `burn(name, from, amount, minApprovals, lifeCycleOpts)` - Burn tokens

Task Mutations

- `approve(address, signatures)` - Submit approvals
- `execute(address)` - Execute ready task
- `collect(address)` - Collect completed task
- `abort(keyToAbort, minApprovals, lifeCycleOpts)` - Abort pending task

Role Mutations

- `changeRoot(name, newRoot, minApprovals, lifeCycleOpts)` - Change root
- `setRole(name, account, role, grant, minApprovals, lifeCycleOpts)` - Grant/revoke role

Permission Mutations

- `createPermission(name, withKYC, withMask, minApprovals, lifeCycleOpts)` - Create permission
- `enablePermission(name, enable, minApprovals, lifeCycleOpts)` - Enable/disable
- `setPermission(name, withKYC, withMask, minApprovals, lifeCycleOpts)` - Set mode
- `lockAccount(account, lock, minApprovals, lifeCycleOpts)` - Lock account (default)
- `lockTokenAccount(name, account, lock, minApprovals, lifeCycleOpts)` - Lock account (token)
- `setKYCLevel(account, level, minApprovals, lifeCycleOpts)` - Set KYC (default)
- `setTokenKYCLevel(name, account, level, minApprovals, lifeCycleOpts)` - Set KYC (token)
- `setMask(account, mask, minApprovals, lifeCycleOpts)` - Set mask (default)
- `setTokenMask(name, account, mask, minApprovals, lifeCycleOpts)` - Set mask (token)

Fee Mutations

- `setFeeSettings(name, settings, minApprovals, lifeCycleOpts)` - Configure fees
- `enableFee(name, enable, minApprovals, lifeCycleOpts)` - Enable/disable fee
- `setAccountRate(name, account, rate, minApprovals, lifeCycleOpts)` - Set account rate
- `setVIPAccount(name, account, isVIP, minApprovals, lifeCycleOpts)` - Set VIP status
- `withdrawFee(name, to, amount)` - Withdraw fees

Account Mutations

- `send(from, privateKey, to, amount)` - Send ETH

Maintenance Mutations

- `freeze(name)` - Freeze token
- `unfreeze(name)` - Unfreeze token
- `disableTransfer(name)` - Disable transfers
- `enableTransfer(name)` - Enable transfers

Appendix: Type Definitions

TaskLifeCycleOptions

```
input TaskLifeCycleOptions {
  expiry: Int           # Seconds until expiration
  collectDelay: Int     # Seconds before collection allowed
  autoExec: Boolean     # Auto-execute when ready
}
```

TokenSettingsInput

```
input TokenSettingsInput {
  symbol: String!
  decimals: Int!
  totalSupply: BigInt!
  owner: String!
  privatePermission: Boolean!
  tokenPermission: String!
  withMask: Boolean!
  withKYC: Boolean!
  accountMan: String!
  enableFee: Boolean!
  feeSettings: TokenFeeSettingsInput!
  pausable: Boolean!
  burnable: Boolean!
  mintable: Boolean!
}
```

TokenFeeSettingsInput

```
input TokenFeeSettingsInput {
  enabled: Boolean!
  vipRate: Int!         # Basis points (100 = 1%)
  defaultRate: Int!     # Basis points
  minFee: BigInt!       # Atomic units
  maxFee: BigInt!       # Atomic units
}
```

PermissionMaskInput

```
input PermissionMaskInput {
  canSend: Boolean!
  canReceive: Boolean!
}
```

Role Enum

```
enum Role {
  DEFAULT_ADMIN_ROLE
  UPGRADER_ROLE
  MINT_ADMIN_ROLE
  BURN_ADMIN_ROLE
  COMPLIANCE_ADMIN_ROLE
  FEE_ADMIN_ROLE
  FEE_RECIPIENT_ROLE
  PAUSER_ROLE
}
```

KYCLevel Enum

```
enum KYCLevel {
  KYC0  # Blocked
  KYC1  # View only
  KYC2  # Standard
  KYC3  # Accredited
  KYC4  # Institutional
}
```

TaskStatus Enum

```
enum TaskStatus {
  Pending
  Ready
  Executing
  Executed
  Aborted
  Expired
}
```

End of Programming Guide