

BSN Spartan Network Data Center Portal

User Manual

Nov. 18, 2022

Version 1.0.0

Contents

1 OVERVIEW	1
1.1 WHAT IS BSN SPARTAN.....	1
1.1.1 Glossary	1
1.1.2 What is blockchain	1
1.1.3 What is Non-Cryptocurrency Public Chain	2
1.1.4 What is a Wallet.....	2
1.2 ROLES	2
1.2.1 Data Center Operator	2
1.2.2 Foundation Member	2
1.2.3 End-user.....	2
1.3 WHY BSN SPARTAN	2
1.3.1 Public vs. Private	3
1.3.2 Cryptocurrency vs. Non-Cryptocurrency.....	3
1.3.3 Centralized vs. Decentralized.....	3
1.3.4 Blockchain vs. Infrastructure	4
2 GETTING STARTED	4
2.1 GET A WALLET ADDRESS	4
2.1.1 An Existing secp256k1 Wallet Address	4
2.1.2 Using MetaMask	4
2.2 GET ACCESS INFORMATION	5
2.3 GATEWAY ACCESS INSTRUCTION.....	5
2.3.1 Key Parameters.....	6
2.3.2 Gateway Request Format	6
2.4 TOP UP GAS CREDIT.....	6
2.4.1 Check the email of Submitted Order.....	9

2.4.2 Check the email of Successful Payment	10
2.4.3 Check the email of Successful Top-up	10
2.4.4 Check the Currency or USDC Refund (If Top-up Failed)	10
2.5 CONTRACT MARKETPLACE	10
2.5.1 Spartan Official Smart Contract Services	10
2.5.2 Open Source Smart Contract Projects	11
2.6 SPARTAN OFFICIAL SMART CONTRACTS (OPTIONAL)	11
2.6.1 Spartan DID	11
2.6.2 SpartanUSD Stablecoin Smart Contract	13
2.6.3 Spartan Official NFT Smart Contract	20
3 INFO ON THE NON-CRYPTOCURRENCY PUBLIC CHAINS	40
3.1 SPARTAN-I CHAIN (POWERED BY NC ETHEREUM)	40
3.1.1 About Spartan-I Chain (Powered by NC Ethereum)	40
3.1.2 Ethereum and Geth Documentation	41
3.2 SPARTAN-II CHAIN (POWERED BY NC COSMOS)	42
3.2.1 About Spartan-II Chain (Powered by NC Cosmos)	42
3.2.2 Resources	42
3.3 SPARTAN-III CHAIN (POWERED BY NC POLYGONEDGE)	42
3.3.1 About Spartan-III Chain (Powered by NC PolygonEdge)	42
3.3.2 Resources	43
4 FAQS	44
4.1 FREQUENTLY ASKED QUESTIONS	44
4.2 WHAT IS A WALLET ADDRESS?	44
4.3 WHAT IS A PRIVATE KEY?	44
4.4 WHAT IS A VIRTUAL DATA CENTER?	45
4.5 WHAT IS GAS CREDIT?	45

1 Overview

1.1 What is BSN Spartan

1.1.1 Glossary

Glossary	Definition
Non-Cryptocurrency Public Chain (NC Public Chain)	A Non-Cryptocurrency Public Chain is a transformed public chain framework based on an existing public chain. Gas Credit transfers are not permitted between standard wallets. There will be no cryptocurrency incentives for mining or participating in consensus.
Full Node	A Full Node on an NC Public Chain does not participate in consensus. Upon registering a Full Node on the BSN Spartan Network, it synchronizes all data on the specific chain. The data center to which the Full Node belongs will receive relevant incentives under the Node Establishment Incentive Program.
Spartan-I Chain (Powered by NC Ethereum)	The Spartan-I Chain is an NC Public Chain version of Ethereum and serves as the default chain of the BSN Spartan Network.
Spartan-II Chain (Powered by NC Cosmos)	The Spartan-II Chain is an NC Public Chain version of Cosmos.
Spartan-III Chain (Powered by NC PolygonEdge)	The Spartan-III Chain is an NC Public Chain version of Polygon Edge.
Gas	In NC Public Chains, Gas is the amount of resources consumed during a transaction or smart contract execution.
Gas Credit	In a similar fashion to cryptocurrencies, Gas Credits are used as a means of paying the Gas fee on NC Public Chains. However, Gas Credits cannot be transferred between standard wallets. Only the Data Center Operator's NTT wallet can be used to purchase Gas Credits with NTT.
Wallet	Wallet refers to the wallet address or smart contract address of an NC Public Chain on the BSN Spartan Network, which can be generated arbitrarily by users. The wallets are used to hold non-transferable Gas Credits.

1.1.2 What is blockchain

" A blockchain is a type of Digital Ledger Technology (DLT) that consists of growing list of records, called blocks, which are securely linked together using cryptography. "

Blockchain technology was used to build digital ledgers when it was first invented, but with the continuous upgrading and iteration of the technology, various innovative applications based on blockchain continue to emerge, and NFTs are one of the most common applications. The emergence of NFTs clearly tells the world that the potential of blockchain technology is far more than a small ledger, but a new generation of data management system that could replace traditional databases. For a healthy and secure blockchain system, the data on the chain cannot be secretly tampered with, nor can it be accidentally deleted. Users can easily verify the data's authenticity

and accuracy, which greatly reduces the cost of communication, increases trust and improves the efficiency of data use. In addition, in our opinion, blockchain will also become a pivotal technology to promote the evolution of traditional private IT systems to new public IT systems.

1.1.3 What is Non-Cryptocurrency Public Chain

A Non-Cryptocurrency Public Chain is a transformed public chain framework based on an existing public chain. Gas Credit transfers are not permitted between standard wallets. There are no cryptocurrency incentives for mining or participating in consensus.

1.1.4 What is a Wallet

Wallet refers to the wallet address or smart contract address of an NC Public Chain on the BSN Spartan Network, which can be generated arbitrarily by users. The wallets are used to hold non-transferable Gas Credits.

1.2 Roles

1.2.1 Data Center Operator

A Data Center Operator is the operator of a Virtual Data Center. A Virtual Data Center is a set of locally installed software systems that contains one or more registered full nodes of different NC Public Chains. Each Virtual Data Center has one NTT wallet and is eligible to receive Node Establishment and Data Center Monthly Incentives.

1.2.2 Foundation Member

Foundation Members refers to the members of the [BSN Foundation](#). Each member must operate a Governance Data Center, which contains all NC Public Chain consensus nodes and has the right to vote on governance matters of the BSN Spartan Network.

1.2.3 End-user

An end-user refers to a person or company that deploys or calls smart contracts on the BSN Spartan Network.

1.3 Why BSN Spartan

The purpose of the BSN Spartan project is to develop, build and promote a global decentralized cloud service network that consists of non-cryptocurrency public chains for enterprise uses and

utilities without any speculative elements.

1.3.1 Public vs. Private

Traditional IT systems are built on independent and non-public databases, each company has its own unique data storage mechanism and structure, and the cost of data exchange between systems is very high. Even if it is public data, a user must go to various websites to download and collect the data. This process is destined to become unacceptable in the future, with ever-increasing demands for digitization.

The Spartan network can solve this problem to a certain extent. Due to the openness of blockchain data and the characteristics of the consensus mechanism, all data centers using the Spartan network will be able to easily share data because the data is only logically isolated; as long as permissions are provided to each other, the data can be exchanged. Complex data migration will be a thing of the past. Using the Spartan network, a user just needs to connect to the network and the data will be synchronized and accessed from anywhere with any device. On any full node a user can obtain all public data generated by different end users. This will greatly improve the efficiency of data acquisition and use.

1.3.2 Cryptocurrency vs. Non-Cryptocurrency

Cost control is a critical task for traditional industries and costs must be predictable. However, the value of cryptocurrencies that traditional public chains need to consume for normal transactions fluctuates. Yesterday, a transaction may cost 1 USD and today it costs 5 USD, and this fluctuation is unpredictable. Due to this volatility, almost no traditional industries have built their businesses using blockchain technology. The three non-cryptocurrency public chains launched by the Spartan Network fundamentally eliminate the volatility of costs to use the chains by prohibiting the transfer of Gas Credits between standard wallets. Furthermore, Gas Credit can only be purchased with NTT, which is anchored to fiat currency. The cost becomes predictable. This makes Spartan Network capable of supporting traditional industries and organizations can use Spartan Network as their underlying infrastructure with confidence they can manage costs effectively.

1.3.3 Centralized vs. Decentralized

When a user interacts with a centralized system, all the requests initiated to the system and all the information entered is owned and controlled by the system's backend. This centralized architecture allows operators to easily modify any data, and it is difficult for users to verify the authenticity

and accuracy of the data. The Spartan Network benefits from blockchain's consensus mechanism, which prevents data from being secretly tampered after being uploaded to the chain and allows users to easily verify any data. At the same time, Spartan also strictly implements decentralized governance and all new or changed governance rules are voted by all consensus parties.

1.3.4 Blockchain vs. Infrastructure

Compared with traditional public chains, the Spartan Network is more scalable and provides technology diversity. A single public chain is often subject to various performance bottlenecks, resulting in a long processing cycle for transactions on the chain. The Spartan Network is a multi-chain ecosystem and the data exchange between each chain is realized through the interchain services. At the same time, different chain frameworks have their own characteristics and often have their own advantages. Looking ahead to the future, different chains may be suitable for different industries and fields. Users can choose according to their own needs.

2 Getting Started

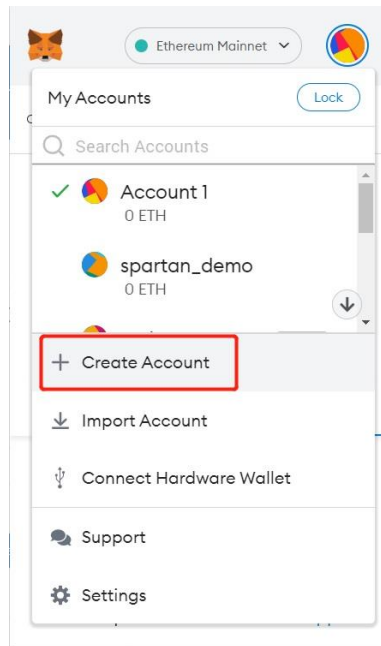
2.1 Get a Wallet Address

2.1.1 An Existing secp256k1 Wallet Address

If you already have a private key generated by the secp256k1 algorithm and its corresponding wallet address, such as an Ethereum wallet address, you can use that wallet address directly.

2.1.2 Using MetaMask

You can download [MetaMask](#) and create an account for free.



The account address in MetaMask can be used as a Wallet Address, and the corresponding private key is the same.

2.2 Get Access Information

Enter your email address and we will send you the access information of the Non-Cryptocurrency public chains.

Select a chain you want to access, then input the email address and get the verification code. Then, click the "**Confirm**" button to submit your application.

Shortly, BSN Spartan Network Data Center Portal will notify you by email and you are able to

2.3 Gateway Access Instruction

2.3.1 Key Parameters

- Access key: accessKey
- Target chain type: chainType
- Protocol: protocol

2.3.2 Gateway Request Format

2.3.2.1 HTTP Request

`https://[domain:port]/api/[accessKey]/[chainType]/rpc/[chain_path]`

Note: [chain_path] is not required, can be null

Example: `https://[domain:port]/api/015416c06ef74ac38a92521792f97e7d/spartanone/rpc`

2.3.2.2 WebSocket Request

`wss://[domain:port]/api/[accessKey]/[chainType]/ws/[chain_path]`

Note: [chain_path] is not required, can be null

Example: `wss://[domain:port]/api/015416c06ef74ac38a92521792f97e7d/spartanone/ws`

2.3.2.3 gRPC Request

[domain:port]

Request header:

x-api-key: [accessKey]

x-api-chain-type: [chainType]

Note: The access information can be found in the notification email of Network Access Information.

2.4 Top Up Gas Credit

Any Wallet in the Non-Cryptocurrency Public Chains must consume Gas Credit when initiating a transaction. For example, if a user wants to initiate a transaction, the user's Wallet needs to consume Gas Credit. At this time, users must use fiat currency or USDC to top up the Gas Credit

of the Wallet to ensure that the transaction can proceed normally.

Operation Steps

Visit Spartan Data Center Portal and click "Top-up Gas Credit":

BSN-SPARTAN Home NC Chains Documentation Contract Marketplace ▾ Return to Foundation Website

Top-up Gas Credit

* Chain Name [Get Access Information](#)

* Wallet Address [How to get a Wallet Address](#) or [Get a Wallet Address in MetaMask](#)

* Verify Wallet Address

* Gas Credit Amount GWEI

* Email Address

* Verification Code Get Verification Code

Review Order Details

Wallet Address: --
Gas Credit Amount: --
Order Total: --

Select your preferred method of payment

☒ Coinbase Pay by cryptocurrency

☐ Stripe VISA, Mastercard, American Express, Discover

☐ Remittance We will send you an email with the payment information

☐ I have read and agreed to [Terms of Service](#)

Confirm

- Select a chain, enter your wallet address and confirm it. Then, enter the amount of Gas Credit you would like to top up. Enter an email address and verify it by entering the verification code. The next step is to choose the payment method. BSN Spartan Network Data Center Portal supports 3 methods: Remittance, Stripe or Coinbase (pay in USDC);
- Click the "Confirm" button, the system will generate an order number and jump to the payment platform you selected. Complete the payment on the pop-up window;

Pay by Remittance:



Order Number: RD20221017

We will send you an email with the payment information.

[Continue to top-up](#)

[Back](#)

Pay by Stripe:

BSN-SPARTAN

BSN-SPARTAN

\$100.00

Due October 28, 2022

To

0x

0x0

From

BSN-SPARTAN

Invoice

#DABBED19-0003

View invoice details >

Choose how you'd like to pay

Card

Bank transfer

Card information

VISA

Pay \$100.00

Powered by stripe

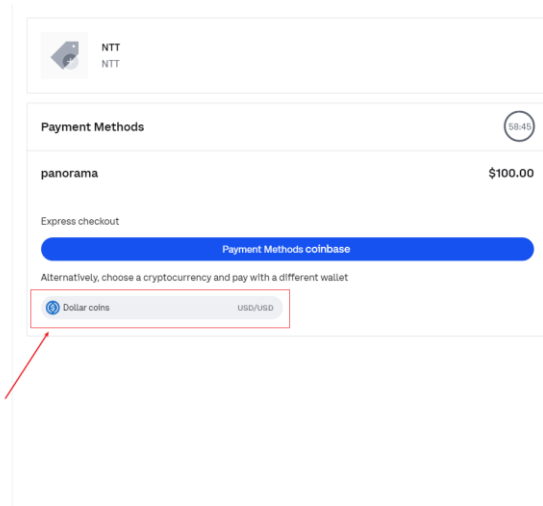
Terms

Privacy

As shown in the above figure, confirm the invoice and enter your card information. After the payment is completed, you will receive a notification by email.

Pay by Coinbase:

Red Date (Hong Kong) Technology Co., Ltd

NTT
NTT


NTT
NTT

Payment Methods 58:45

panorama \$100.00

Express checkout

Payment Methods coinbase

Alternatively, choose a cryptocurrency and pay with a different wallet

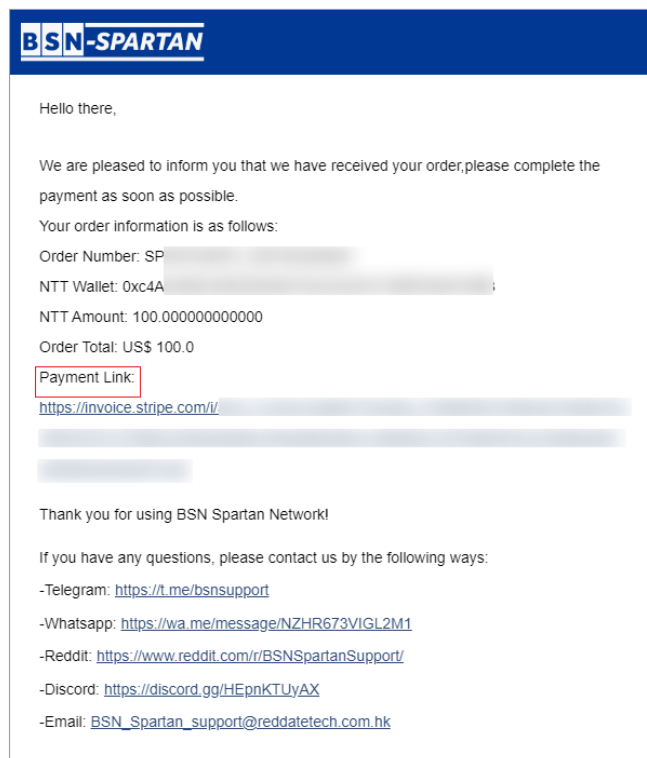
1 Dollar coins USD/USD

You can make the payment in USDC. After the payment is successful, you will receive a notification by email.

Note: All payments above are made by the third-party payment platform, and the Spartan Network Data Center Portal will never obtain your account information.

2.4.1 Check the email of Submitted Order

Users will receive an email notification when the order is submitted. You may also complete the payment via the link in the email.



BSN-SPARTAN

Hello there,

We are pleased to inform you that we have received your order, please complete the payment as soon as possible.

Your order information is as follows:

Order Number: SP [REDACTED]

NTT Wallet: 0xc4A [REDACTED]

NTT Amount: 100.000000000000

Order Total: US\$ 100.0

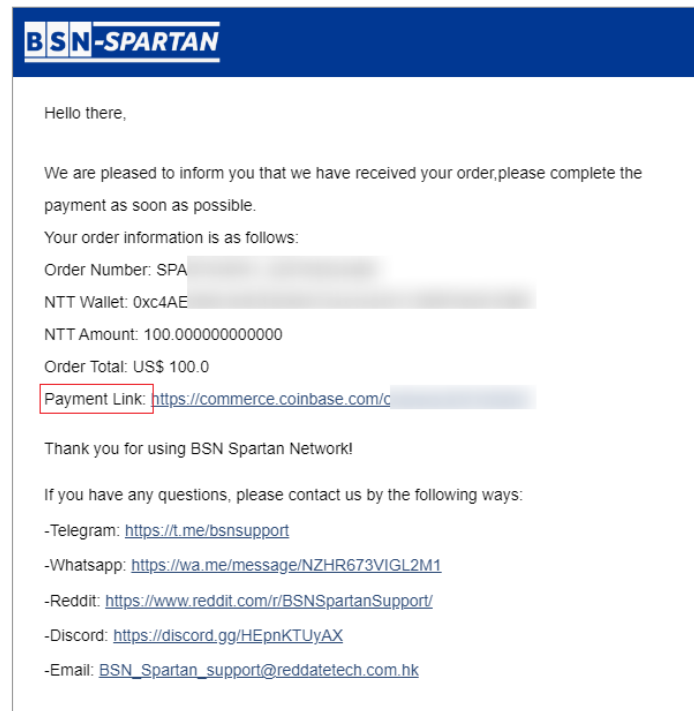
Payment Link:
<https://invoice.stripe.com/i/>

Thank you for using BSN Spartan Network!

If you have any questions, please contact us by the following ways:

- Telegram: <https://t.me/bsnsupport>
- Whatsapp: <https://wa.me/message/NZHR673VIGL2M1>
- Reddit: <https://www.reddit.com/r/BSNSpartanSupport/>
- Discord: <https://discord.gg/HEpnKTUyAX>
- Email: BSN_Spartan_support@reddatetech.com.hk

This email is automatically sent by the system, please do not reply, and if you are not using BSN-Spartan, please ignore.



This email is automatically sent by the system, please do not reply, and if you are not using BSN-Spartan, please ignore.

2.4.2 Check the email of Successful Payment

Users will receive an email notification when the payment succeeds.

2.4.3 Check the email of Successful Top-up

Users will receive an email notification when the Gas Credit top-up succeeds. Users can check the Gas Credit information through the link in the email.

2.4.4 Check the Currency or USDC Refund (If Top-up Failed)

Please make sure the Currency or USDC is correctly refunded.

2.5 Contract Marketplace

More than 30 ready-to-deploy smart contracts covering multiple industries for you to adopt to any application directly to boost your business.

2.5.1 Spartan Official Smart Contract Services

For businesses with strong universality, BSN will provide smart contract-based application services developed and deployed officially. You can fulfill your business needs by directly calling methods of contract using our official tools (SDK, API, or other tools). We will also open source

these contracts, and if you want to deploy them yourself or make some customized changes, you can access our contract source through the GitHub link. BSN will adhere to the concept of open source, easy to use, low cost, and dedicate to the construction and promotion of public IT systems. We hope that our official services can reduce your business costs. We also hope that our open source contracts can provide reference for your own business contract development.

2.5.2 Open Source Smart Contract Projects

Smart contract is an emerging technology, because of its highly transactional and transparent, which makes the application services based on smart contracts highly reliable and trustable. Mastering smart contract technology is one of the keys to making good use of Spartan network. In order to help you understand the Spartan network and smart contracts more intuitively and comprehensively, we have collected a large number of open source smart contract projects, and verified their compatibility through actual deployment in the Spartan network. You can directly deploy smart contract projects that match your business needs in the Spartan network or develop smart contracts that meet your business needs by learning and referring to the source code. At the same time, we hope to help you open your mind and better integrate the advantages of the Spartan network into your own business by showing a large number of practical projects.

2.6 Spartan Official Smart Contracts (Optional)

These Spartan Network Official Smart Contracts are pre-deployed smart contracts managed by Spartan Network operators for performing different tasks. They are open to all end-users to call and execute. These smart contracts are also open-source on Spartan Network GitHub and can be used for developers to study as use cases. We welcome interested developers to deploy more commercial smart contracts for specific business models and scenarios.

2.6.1 Spartan DID

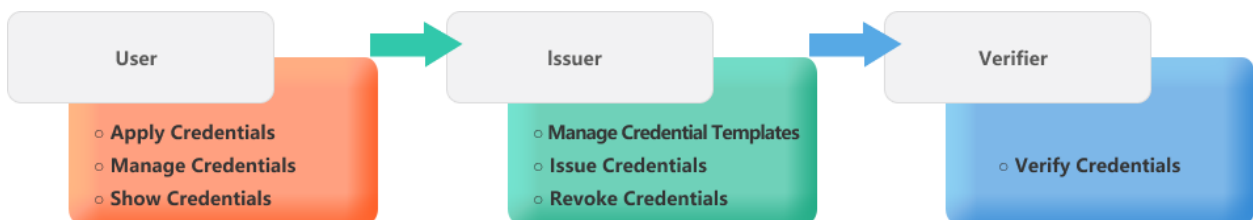
2.6.1.1 Overview

With blockchain technology as the cornerstone and W3C DID as the specification, Spartan DID Services achieve decentralized on-chain mapping of real entities, thus achieving the ability to provide digital identity and digital credential interaction for individuals/organizations.

2.6.1.2 Roles

In the DID ecosystem, there are three roles: User, Issuer, and Verifier

- **User:** Any individual/organization/entity with a digital identity on the chain. Any entity object can create and manage its DID through the developer's own project.
- **Issuer:** The individual or organization that can issue the digital credentials. For example, a university can issue a digital diploma to a student; then the university is an issuer.
- **Verifier:** Also known as a business party, is an individual or organization that uses digital credentials. After being authorized by the user, the verifier can verify the identity of the user or their digital credentials. For example, when a company hires someone, it needs to verify his college diploma, then the company is a verifier.



2.6.1.3 Components

The DID system consists of three components: SDK, Service and Smart Contract. The SDK can be integrated into the developer's own project; Service is responsible for logic processing and communication with nodes; the smart contract is deployed on the chain, and the methods in the contract are called by the DID Service.

2.6.1.4 Functions and features

- Deployed on the Spartan-I Chain (NC Ethereum), the DID Service builds a decentralized digital identity management system, which facilitates autonomous participation and affirmative collaboration among users, issuers, and verifiers.
- Provide a unified decentralized digital identity management, including identifier creation, update, and verification functions.
- Provide mechanisms for issuance, authorization, verification, and revocation of user data credentials.
- Provide the SDK that access to API services, integrate object encapsulation, signature,

verification, and other methods for easy docking by developers.

For a detailed introduction to DID, please refer to [GitHub](#).

And, BSN-Spartan has also completely open sourced IdentityHub, users can install it locally and store their own private data. For more details, please refer to [GitHub](#).

2.6.2 SpartanUSD Stablecoin Smart Contract

The SpartanUSD Stablecoin is an ERC20-based Token, issued by the Spartan Network operators officially through the stablecoin contract on Spartan-III Chain (NC PolygonEdge), which is strictly anchored to USDC in a ratio of 1:1. The basic functions in the stablecoin contract are Mint, Transfer, Withdraw and Burn. The circulation of the stablecoin will be strictly controlled within a range not greater than the amount of USDC pledged by the stablecoin users into the USDC wallet on the Polygon mainnet.

2.6.2.1 Basic Information

- **SpartanUSD Contract Address:** `0x1fD89dc1f4Ffbb797d471D6BB0dbb8EfEABdbe9c` on the Spartan-III chain
- **USDC Pledge Wallet Address:** `0x764b33c01a611597438f0286e946633685ed3d2f` on Polygon (Matic Network)
- **Maximum Counting Accuracy:** 6 Decimals (0.000001 SUSD)
- **Name:** SpartanUSD
- **Symbol:** SUSD
- **Transfer Service Fee:** 0.1% of the amount of SUSD transferred
- **Withdraw Service Fee:** 0.003 SUSD

2.6.2.2 Common Functions

2.6.2.2.1 Mint SpartanUSD

By calling the official USDC contract (**contract address:** `0x2791Bca1f2de4661ED88A30C99A7a9449Aa84174` on Polygon (Matic Network), the user uses the transfer() method to transfer USDC equal to the expected issuance amount of SUSD to the account (**accou**

nt address: `0x764b33c01a611597438f0286e946633685ed3d2f` on Polygon (Matic Network)).

Spartan obtains transaction information by listening related events, and the SpartanUSD contract will be called to mint the same amount of SUSD to the user account of the Spartan-III chain (Powered by NC PolygonEdge) through the minter account of SpartanUSD.

The Spartan-III user account address is the same as the address of the sender account of the USDC transaction on Polygon.

***Note:** Please ensure the security of the private key of the Polygon account of the USDC transfer transaction sender, which will also be used as the private key of the Spartan-III wallet receiving SUSD.*

2.6.2.2.2 Transfer SpartanUSD

The user can transfer their SpartanUSD asset to any other Spartan-III wallet by calling the SUSD contract's transfer() method (**contract address:** `0x1fD89dc1f4Fbb797d471D6B B0dbb8EfEABdbe9c` on the Spartan-III Chain).

The transfer() method will charge a service fee of 0.1% of the caller's transaction amount, and the service fee will not exceed 10 USD.

- **Input Parameters:** Receiver Wallet Address (on the Spartan-III chain), Transfer Amount (Please enter a multiple of 10000, 10000 = 0.01SUSD);

***Note:** The Transfer Amount needs to be a value which is a multiple of 10000, because SUSD's accuracy is 6 decimals, so, the value of 10000 is equivalent to 0.01 SUSD.*

- **Output Parameters:** A bool parameter shows successful or failed;
- **Method Definition:** transfer (address to, uint256 amount) returns (bool);
- **Event Parameters:** Sender Wallet Address, Receiver Wallet Address, Transfer Amount, Service Fee;
- **Event Definition:** Transfer (msg.sender, to, amount, serviceCharge);
- **Example:**

```
func TestTransfer(t *testing.T) {  
    cli, err := ethclient.Dial(NodeUrl)  
    if err != nil {
```

```

        log.Logger.Error(err)
    }
    instance, err := stablecoin.NewStablecoin(common.HexToAddress(Address), cli)
    if err != nil {
        log.Logger.Error(err)
    }
    auth, err := eth.GenAuth(cli, PrivateKey)
    if err != nil {
        log.Logger.Error(err)
    }
    tx, err := instance.Transfer(auth, common.HexToAddress(to), new(big.Int).SetUint64(amount))
    if err != nil {
        log.Logger.Error(err)
    }
    fmt.Println("tx Hash:", tx.Hash().String())
}

```

2.6.2.2.3 Withdraw SpartanUSD

The user can withdraw their SpartanUSD asset to USDC which will be transferred to a Polygon (Matic Network) wallet by calling the SUSD contract's withdraw() method (contract address: 0x1fD89dc1f4Ffbb797d471D6BB0dbb8EfEABdbe9c on the Spartan-III Chain).

The withdraw() method will charge a constant service fee of 0.003 SUSD.

- **Input Parameters:** Receiver Account Address (on Polygon (Matic Network)), Withdraw Amount;

Note: The SUSD balance of the sender account address needs to be greater than the total price of the transaction (withdraw amount + withdraw service fee).

- **Output Parameters:** None;
- **Method Definition:** withdraw (address payee, uint256 amount);
- **Event Parameter:** Sender Wallet Address, Withdraw Amount, Service Fee, Receiver Wallet Address (on Polygon (Matic Network));
- **Event Definition:** Withdraw (msg.sender, amount, _withdrawFee, payee);
- **Example:**

```
func TestWithdraw(t *testing.T) {
```

```

cli, err := ethclient.Dial(NodeUrl)
if err != nil {
    log.Logger.Error(err)
}
instance, err := stablecoin.NewStablecoin(common.HexToAddress(Address), cli)
if err != nil {
    log.Logger.Error(err)
}
auth, err := eth.GenAuth(cli, PrivateKey)
if err != nil {
    log.Logger.Error(err)
}
tx, err := instance.Withdraw(auth, common.HexToAddress(payee), new(big.Int).SetUint64(a
mount))
if err != nil {
    log.Logger.Error(err)
}
fmt.Println("tx Hash:", tx.Hash().String())
}

```

2.6.2.2.4 Check SpartanUSD Balance

Users can check their SpartanUSD balance by calling the SUSD contract's `balanceOf()` method.

- **Input Parameters:** Target Account Address (on the Spartan-III chain);
- **Output Parameters:** Balance;
- **Method Definition:** `balanceOf (address account) view returns (uint256);`
- **Example:**

```

func TestBalanceOf(t *testing.T) {
    cli, err := ethclient.Dial(NodeUrl)
    if err != nil {
        log.Logger.Error(err)
    }
    instance, err := stablecoin.NewStablecoin(common.HexToAddress(Address), cli)
    if err != nil {
        log.Logger.Error(err)
    }
    balance, err := instance.BalanceOf(nil, common.HexToAddress(Address))
    if err != nil {
        log.Logger.Error(err)
    }
}

```

```
}  
    fmt.Println("balance:", balance.String())  
}
```

2.6.2.2.5 Check the Circulation of SpartanUSD

Users can check the total circulation of SpartanUSD by calling the SUSD contract's `totalSupply()` method.

After the `mint()` method, the total circulation will be increased by the amount of mint and after the `withdraw()` or `burn()` method, the total circulation will be decreased by the amount of withdraw or burn.

- **Input Parameters:** None;
- **Output Parameters:** The Total Circulation of SUSD;
- **Method Definition:** `totalSupply()` view returns (uint256);
- **Example:**

```
func TestTotalSupply(t *testing.T) {  
    cli, err := ethclient.Dial(NodeUrl)  
    if err != nil {  
        log.Logger.Error(err)  
    }  
    instance, err := stablecoin.NewStablecoin(common.HexToAddress(Address), cli)  
    if err != nil {  
        log.Logger.Error(err)  
    }  
    totalSupply, err := instance.TotalSupply(nil)  
    if err != nil {  
        log.Logger.Error(err)  
    }  
    fmt.Println("totalSupply:", totalSupply.String())  
}
```

2.6.2.2.6 Check the Maximum Transaction Service Fee

Users can check the maximum transaction service fee by calling the SUSD contract's `TestMaximumTransferCharge()` method.

The return value is counted in units of 0.000001 SUSD (6 decimals), for example, 1000000 means 1 USUD.

- **Input Parameters:** None;
- **Output Parameters:** The maximum transaction service fee;
- **Method Definition:** maximumTransferCharge() view returns (uint256);
- **Example:**

```
func TestMaximumTransferCharge(t *testing.T) {
    cli, err := ethclient.Dial(NodeUrl)
    if err != nil {
        log.Logger.Error(err)
    }
    instance, err := stablecoin.NewStablecoin(common.HexToAddress(Address), cli)
    if err != nil {
        log.Logger.Error(err)
    }
    maximumTransferCharge, err := instance.MaximumTransferCharge(nil)
    if err != nil {
        log.Logger.Error(err)
    }
    fmt.Println("maximumTransferCharge:", maximumTransferCharge.String())
}
```

2.6.2.2.7 Check the Transaction Service Fee Ratio

Users can check the transaction service fee ratio by calling the SUSD contract's getTransferRatio() method.

The return value is counted in units of 0.0001 (4 decimals), for example, 10 means 0.001 (0.1%).

- **Input Parameters:** None;
- **Output Parameters:** the transaction service fee ratio (counted in units of 0.0001);
- **Method Definition:** getTransferRatio() view returns (uint256);
- **Example:**

```
func TestGetTransferRatio(t *testing.T) {
    cli, err := ethclient.Dial(NodeUrl)
    if err != nil {
        log.Logger.Error(err)
    }
    instance, err := stablecoin.NewStablecoin(common.HexToAddress(Address), cli)
    if err != nil {
```

```
        log.Logger.Error(err)
    }
    transferRatio, err := instance.GetTransferRatio(nil)
    if err != nil {
        log.Logger.Error(err)
    }
    fmt.Println("transferRatio:", transferRatio.String())
}
```

2.6.2.2.8 Check the Maximum Amount of SUSD for Transfer

Users can check the maximum amount of SUSD for transfer and the service fee for running a transfer of your input amount SUSD by calling the SUSD contract's `queryTransferLimit()` method.

- **Input Parameters:** SUSD Amount of a transfer;
- **Output Parameters:** Service Fee, the Maximum Amount of SUSD;
- **Method Definition:** `queryTransferLimit (uint256 amount) view returns (uint256 serviceCharge, uint256 maxTransferAmount);`
- **Example:**

```
func TestQueryTransferLimit(t *testing.T) {
    cli, err := ethclient.Dial(NodeUrl)
    if err != nil {
        log.Logger.Error(err)
    }
    instance, err := stablecoin.NewStablecoin(common.HexToAddress(Address), cli)
    if err != nil {
        log.Logger.Error(err)
    }
    outstruct, err := instance.QueryTransferLimit(nil, new(big.Int).SetUint64(amount))
    if err != nil {
        log.Logger.Error(err)
    }
    fmt.Println("ServiceCharge:", outstruct.ServiceCharge.String())
    fmt.Println("MaxTransferAmount:", outstruct.MaxTransferAmount.String())
}
```

2.6.2.2.9 Check the Maximum Amount of SUSD for Withdraw

Users can check the maximum amount of SUSD for withdraw and the service fee for running a withdraw method by calling the SUSD contract's `queryWithdrawLimit()` method.

- **Input Parameters:** None;
- **Output Parameters:** Service Fee, the Maximum Amount of SUSD for withdraw;
- **Method Definition:** queryWithdrawLimit() view returns (uint256 withdrawFee, uint256 maxWithdrawAmount);
- **Example:**

```
func TestQueryWithdrawLimit(t *testing.T) {
    cli, err := ethclient.Dial(NodeUrl)
    if err != nil {
        log.Logger.Error(err)
    }
    instance, err := stablecoin.NewStablecoin(common.HexToAddress(Address), cli)
    if err != nil {
        log.Logger.Error(err)
    }
    outstruct, err := instance.QueryWithdrawLimit(nil)
    if err != nil {
        log.Logger.Error(err)
    }
    fmt.Println("WithdrawFee:", outstruct.WithdrawFee.String())
    fmt.Println("MaxWithdrawAmount:", outstruct.MaxWithdrawAmount.String())
}
```

2.6.3 Spartan Official NFT Smart Contract

2.6.3.1 Spartan-NFT-721

2.6.3.1.1 Function Introduction

The Spartan-NFT-721 proxy contract is used to provide a complete set of APIs corresponding to ERC-721 methods. The interfaces include functions like Spartan-NFT mint, authorization, query authorization, transfer and destruction. The purpose of this set of contracts is to allow end-users to directly create and manage ERC721 NFTs under the governance of BSN Foundation.

- **Smart contract address:**

Spartan-I Chain (Powered by NC Ethereum):

0xD1A6C2dbCdbafbF0eCD033B38B83DbE0904caA4b

Spartan-II Chain (Powered by NC Cosmos):

0x8fC9EC239fe077ce57a5C5D825e47Ffc2979Fbf8

Spartan-III Chain (Powered by NC PolygonEdge):

0x55aa4279ec99E3952803b791b869B8911761f02A

- **Example:** <https://github.com/BSN-Spartan/NFT.git>

2.6.3.1.2 API Definition

2.6.3.1.2.1 Mint

Users can mint NFTs by calling this interface.

- **Input parameters:** receiver address, NFT name, NFT symbol, uri;
- **Output parameters:** none;
- **Function definition:** mint (address to, string memory name,string memory symbol, string memory tokenURI);
- **Event parameters:** 0x0 address (null address), receiver address, NFT token ID
- **Event definition:** Transfer (address (0), to, tokenID);
- **Example:**

```
func TestMint(t *testing.T) {
    cli := server.NewETHClientByURL(t, url, key)
    session, err := e.NewERC721Session(cli, common.HexToAddress(Address))
    if err != nil {
        t.Fatal(err)
    }
    tx, err := session.Mint(common.HexToAddress(owner), "sparton_nft", "sparton_nft", "sparton_nft")
    if err != nil {
        t.Fatal(err)
    }
    fmt.Println(fmt.Sprintf("tx Hash: %s", tx.Hash().String()))
}
```

2.6.3.1.2.2 Safe Mint

Users can safe mint NFTs by calling this interface.

- **Input parameters:** receiver address, NFT name, NFT symbol, uri, attached args;

- **Output parameters:** none;
- **Function definition:** safeMint (address to, string memory name, string memory symbol, string memory tokenURI, bytes memory data);
- **Event parameters:** 0x0 address (null address), receiver address, NFT token ID;
- **Event definition:** Transfer (address (0), to, tokenID);
- **Example:**

```
func TestSafeMint(t *testing.T) {
    cli := server.NewETHClientByURL(t, url, key)
    session, err := e.NewERC721Session(cli, common.HexToAddress(Address))
    if err != nil {
        t.Fatal(err)
    }
    data := []byte{0x1}
    tx, err := session.SafeMint(common.HexToAddress(owner), "sparton_nft", "sparton_nft", "sparton_nft", data)
    if err != nil {
        t.Fatal(err)
    }
    fmt.Println(fmt.Sprintf("tx Hash: %s", tx.Hash().String()))
}
```

2.6.3.1.2.3 NFT Authorization

An NFT owner can call this API to authorize the NFT, the sender of the transaction must be the NFT owner.

- **Input parameters:** authorizer's wallet address, NFT token ID;
- **Output parameters:** none;
- **Function definition:** approve (address to, uint256 tokenID);
- **Event parameters:** owner's wallet address, authorizer's wallet address, NFT token ID;
- **Event definition:** Approval (operator, to, tokenID);
- **Example:**

```
func TestApprove(t *testing.T) {
    cli := server.NewETHClientByURL(t, url, key)
    session, err := e.NewERC721Session(cli, common.HexToAddress(Address))
```

```
    if err != nil {
        t.Fatal(err)
    }
    tokenId := new(big.Int).SetUint64(1)
    tx, err := session.Approve(common.HexToAddress(account), tokenId)
    if err != nil {
        t.Fatal(err)
    }
    fmt.Println(fmt.Sprintf("tx Hash: %s", tx.Hash().String()))
}
```

2.6.3.1.2.4 Query NFT Authorization

Users can call this interface to query the NFT authorization.

- **Input parameters:** NFT token ID;
- **Output parameters:** authorizer's wallet address;
- **Function definition:** getApproved (uint256 tokenId) view returns (address);
- **Example:**

```
func TestGetApproved(t *testing.T) {
    cli := server.NewETHClientByURL(t, url, key)
    session, err := e.NewERC721Session(cli, common.HexToAddress(Address))
    if err != nil {
        t.Fatal(err)
    }
    tokenId := new(big.Int).SetUint64(1)
    tx, err := session.GetApproved(tokenId)
    if err != nil {
        t.Fatal(err)
    }
    fmt.Println(fmt.Sprintf("Account Address: %s", tx.String()))
}
```

2.6.3.1.2.5 Wallet Authorization

An NFT owner can call this interface to authorize the wallet address, the sender of the transaction must be the NFT owner.

- **Input parameters:** authorizer's wallet address, authorization ID;
- **Output parameters:** none;

- **Function definition:** setApprovalForAll (address operator, bool approved);
- **Event parameters:** owner's wallet address, authorizer's wallet address, authorization ID;
- **Event definition:** ApprovalForAll (owner, operator, approved);
- **Example:**

```
func TestSetApprovalForAll(t *testing.T) {
    cli := server.NewETHClientByURL(t, url, key)
    session, err := e.NewERC721Session(cli, common.HexToAddress(Address))
    if err != nil {
        t.Fatal(err)
    }
    tx, err := session.SetApprovalForAll(common.HexToAddress(account), true)
    if err != nil {
        t.Fatal(err)
    }
    fmt.Println(fmt.Sprintf("tx Hash: %s", tx.Hash().String()))
}
```

2.6.3.1.2.6 Verify Wallet Authorization

Users can call this interface to verify the wallet authorization.

- **Input parameters:** owner's wallet address, authorizer's wallet address;
- **Output parameters:** Boolean value;
- **Function definition:** isApprovedForAll (address owner, address operator) view returns (bool);
- **Example:**

```
func TestIsApprovedForAll(t *testing.T) {
    cli := server.NewETHClientByURL(t, url, key)
    session, err := e.NewERC721Session(cli, common.HexToAddress(Address))
    if err != nil {
        t.Fatal(err)
    }
    tx, err := session.IsApprovedForAll(common.HexToAddress(owner), common.HexToAddress(account))
    if err != nil {
        t.Fatal(err)
    }
    fmt.Println(fmt.Sprintf("Is ApprovedForAll:%t", tx))
}
```

2.6.3.1.2.7 Safe Transfer

An NFT owner or authorized wallet address can call this interface to safe transfer the NFT.

- **Input parameters:** owner's wallet address, receiver address, NFT token ID, attached args;
- **Output parameters:** none;
- **Function definition:** safeTransferFrom (address from, address to, uint256 tokenID, bytes memory data);
- **Event parameters:** owner's wallet address, receiver address, NFT token ID;
- **Event definition:** Transfer (from, to, tokenID);
- **Example:**

```
func TestSafeTransferFrom(t *testing.T) {
    cli := server.NewETHClientByURL(t, url, key)
    session, err := e.NewERC721Session(cli, common.HexToAddress(Address))
    if err != nil {
        t.Fatal(err)
    }
    data := []byte{0x1}
    tokenId := new(big.Int).SetUint64(1)
    tx, err := session.SafeTransferFrom(common.HexToAddress(owner), common.HexToAddress(account), tokenId, data)
    if err != nil {
        t.Fatal(err)
    }
    fmt.Println(fmt.Sprintf("tx Hash: %s", tx.Hash().String()))
}
```

2.6.3.1.2.8 Transfer

An NFT owner or authorized wallet address can call this interface to transfer the NFT.

- **Input parameters:** owner's wallet address, receiver address, NFT token ID;
- **Output parameters:** none;
- **Function definition:** transferFrom (address from, address to, uint256 tokenID);
- **Event parameters:** owner's wallet address, receiver address, NFT token ID;
- **Event definition:** Transfer (from, to, tokenID);

- **Example:**

```
func TestTransferFrom(t *testing.T) {
    cli := server.NewETHClientByURL(t, url, key)
    session, err := e.NewERC721Session(cli, common.HexToAddress(Address))
    if err != nil {
        t.Fatal(err)
    }
    tokenId := new(big.Int).SetUint64(2)
    tx, err := session.TransferFrom(common.HexToAddress(owner), common.HexToAddress(account), tokenId)
    if err != nil {
        t.Fatal(err)
    }
    fmt.Println(fmt.Sprintf("tx Hash: %s", tx.Hash().String()))
}
```

2.6.3.1.2.9 NFT Destruction

An NFT owner or authorized wallet address can call this interface to destroy the NFT.

- **Input parameters:** NFT token ID;
- **Output parameters:** none;
- **Function definition:** burn (uint256 tokenId);
- **Event parameters:** owner's wallet address, 0x0 address (null address), NFT token ID;
- **Event definition:** Transfer (owner, address (0), tokenId);
- **Example:**

```
func TestBurn(t *testing.T) {
    cli := server.NewETHClientByURL(t, url, key)
    session, err := e.NewERC721Session(cli, common.HexToAddress(Address))
    if err != nil {
        t.Fatal(err)
    }
    tokenId := new(big.Int).SetUint64(1)
    tx, err := session.Burn(tokenId)
    if err != nil {
        t.Fatal(err)
    }
    fmt.Println(fmt.Sprintf("tx Hash: %s", tx.Hash().String()))
}
```

2.6.3.1.2.10 Query Quantity

Users can call this interface to query the quantity of the NFTs owned by this wallet address.

- **Input parameters:** owner's wallet address;
- **Output parameters:** number of NFTs;
- **Function definition:** balanceOf (address owner) view returns (uint256);
- **Example:**

```
func TestBalanceOf(t *testing.T) {  
    cli := server.NewETHClientByURL(t, url, key)  
    session, err := e.NewERC721Session(cli, common.HexToAddress(Address))  
    if err != nil {  
        t.Fatal(err)  
    }  
    tx, err := session.BalanceOf(common.HexToAddress(owner))  
    if err != nil {  
        t.Fatal(err)  
    }  
    fmt.Println(fmt.Sprintf("nft amount: %s", tx))  
}
```

2.6.3.1.2.11 Query NFT Owner

Users can call this interface to query the owner of the NFT.

- **Input parameters:** NFT token ID;
- **Output parameters:** owner's wallet address;
- **Function definition:** ownerOf (uint256 tokenID) view returns (address);
- **Example:**

```
func TestOwnerOf(t *testing.T) {  
    cli := server.NewETHClientByURL(t, url, key)  
    session, err := e.NewERC721Session(cli, common.HexToAddress(Address))  
    if err != nil {  
        t.Fatal(err)  
    }  
    tokenId := new(big.Int).SetUint64(2)  
    tx, err := session.OwnerOf(tokenId)  
    if err != nil {
```

```
t.Fatal(err)
}
fmt.Println(fmt.Sprintf("Owner address: %s", tx.Hash().String()))
}
```

2.6.3.1.2.12 Query NFT Name

Users can call this interface to query the NFT name.

- **Input parameters:** NFT token ID;
- **Output parameters:** NFT name;
- **Function definition:** tokenName (uint256 tokenID) view returns (string memory);
- **Example:**

```
ffunc TestTokenName(t *testing.T) {
    cli := server.NewETHClientByURL(t, url, key)
    session, err := e.NewERC721Session(cli, common.HexToAddress(Address))
    if err != nil {
        t.Fatal(err)
    }
    tokenId := new(big.Int).SetUint64(2)
    tx, err := session.TokenName(tokenId)
    if err != nil {
        t.Fatal(err)
    }
    fmt.Println(fmt.Sprintf("NFT name: %s", tx))
}
```

2.6.3.1.2.13 Query NFT Symbol

Users can call this interface to query the NFT symbol.

- **Input parameters:** NFT token ID;
- **Output parameters:** NFT symbol;
- **Function definition:** tokenSymbol (uint256 tokenID) view returns (string memory);
- **Example:**

```
func TestTokenSymbol(t *testing.T) {
    cli := server.NewETHClientByURL(t, url, key)
    session, err := e.NewERC721Session(cli, common.HexToAddress(Address))
    if err != nil {
```

```
t.Fatal(err)
}
tokenId := new(big.Int).SetUint64(2)
tx, err := session.TokenSymbol(tokenId)
if err != nil {
    t.Fatal(err)
}
fmt.Println(fmt.Sprintf("NFT symbol: %s", tx))
}
```

2.6.3.1.2.14 Query NFT URI

Users can call this interface to query the NFT URI.

- **Input parameters:** NFT token ID;
- **Output parameters:** NFT URI;
- **Function definition:** tokenURI (uint256 tokenId) view returns (string memory);
- **Example:**

```
func TestTokenURI(t *testing.T) {
    cli := server.NewETHClientByURL(t, url, key)
    session, err := e.NewERC721Session(cli, common.HexToAddress(Address))
    if err != nil {
        t.Fatal(err)
    }
    tokenId := new(big.Int).SetUint64(2)
    tx, err := session.TokenURI(tokenId)
    if err != nil {
        t.Fatal(err)
    }
    fmt.Println(fmt.Sprintf("NFT URI: %s", tx))
}
```

2.6.3.1.2.15 Query Latest Token ID

Users can call this interface to query the latest NFT token ID.

- **Input parameters:** none;
- **Output parameters:** latest NFT token ID;
- **Function definition:** getLatestTokenID() view returns (uint256);

- **Example:**

```
func TestGetLatestTokenID(t *testing.T) {
    cli := server.NewETHClientByURL(t, url, key)
    session, err := e.NewERC721Session(cli, common.HexToAddress(Address))
    if err != nil {
        t.Fatal(err)
    }
    tx, err := session.GetLatestTokenID()
    if err != nil {
        t.Fatal(err)
    }
    fmt.Println(fmt.Sprintf("tokenId: %s", tx))
}
```

2.6.3.2 Spartan-NFT-1155

2.6.3.2.1 Function Introduction

Spartan-NFT-1155 proxy contract is used to provide users with a set of APIs, including mint and batch mint Spartan-NFTs under the standard of ERC1155, as well as authorization, query authorization, transfer, batch transfer and destruction. The purpose of this set of smart contracts is to allow end-users to directly mint ERC1155 NFTs under the governance of BSN Foundation.

- **Smart contract address:**

Spartan-I Chain (Powered by NC Ethereum):

0xD4366bBeF0977f278A91Ae20EfE8A035690Ac90B

Spartan-II Chain (Powered by NC Cosmos):

0xD0Bf538c75310917b2C82C0a715E126783Be030F

Spartan-III Chain (Powered by NC PolygonEdge):

0x0c0f445f359eBa39935012C0EEaFE3cA00B6BFb

- **Example:** <https://github.com/BSN-Spartan/NFT.git>

2.6.3.2.2 API Definition

2.6.3.2.2.1 Safe Mint

Users can call this interface to safe mint the NFT.

- **Input parameters:** receiver address, NFT name, NFT symbol, number of copies of the NFT, uri, attached args;
- **Output parameters:** none;
- **Function definition:** safeMint (address to, string memory name, string memory symbol, uint256 amount, string memory tokenURI, bytes memory data);
- **Event parameters:** operator, 0x0 address (null address), receiver address, NFT token ID, number of copies;
- **Event definition:** TransferSingle (operator, address (0), to, tokenID, amount);
- **Example:**

```
func TestSafeMint(t *testing.T) {
    cli := server.NewETHClientByURL(t, url, key)
    session, err := e.NewERC1155Session(cli, common.HexToAddress(Address))
    if err != nil {
        t.Fatal(err)
    }
    amount := new(big.Int).SetUint64(1)
    data := []byte{0x1}
    tx, err := session.SafeMint(common.HexToAddress(owner), "sparton_nft", "sparton_nft", amount, "sparton_nft", data)
    if err != nil {
        t.Fatal(err)
    }
    fmt.Println(fmt.Sprintf("tx Hash: %s", tx.Hash().String()))
}
```

2.6.3.2.2.2 Batch Safe Mint NFT

Users can call this interface to safer mint the NFT in batch.

- **Input parameters:** receiver address, NFT name set, NFT symbol set, number of NFT copies set, uri set, attached args;
- **Output parameters:** none;
- **Function name:** safeMintBatch;
- **Function definition:** safeMintBatch (address to, string[] memory names, string[] memory symbols, uint256[] memory amounts, string[] memory tokenURIs, bytes memory data);

- **Event parameters:** operator, 0x0 address (null address), receiver address, NFT token ID set, number of copies set;
- **Event definition:** TransferBatch (operator, address (0), to, tokenIDs, amounts);
- **Example:**

```
func TestSafeMintBatch(t *testing.T) {
    cli := server.NewETHClientByURL(t, url, key)
    session, err := e.NewERC1155Session(cli, common.HexToAddress(Address))
    if err != nil {
        t.Fatal(err)
    }
    tokenName := []string{"sparton_nft_1", "sparton_nft_2"}
    tokenSymbol := []string{"sparton_nft_1", "sparton_nft_2"}
    tokenURIs := []string{"http://sparton.json", "http://sparton.json"}
    var amount []*big.Int
    amount = append(amount, new(big.Int).SetUint64(1), new(big.Int).SetUint64(1))
    data := []byte{0x1, 0x2}
    tx, err := session.SafeMintBatch(common.HexToAddress(owner), tokenName, tokenSymbol,
amount, tokenURIs, data)
    if err != nil {
        t.Fatal(err)
    }
    fmt.Println(fmt.Sprintf("tx Hash: %s", tx.Hash().String()))
}
```

2.6.3.2.2.3 Wallet Authorization

NFT owner can call this interface to wallet authorization, the sender of the transaction must be the NFT owner.

- **Input parameters:** authorizer's wallet address, authorization ID;
- **Output parameters:** none;
- **Function definition:** setApprovalForAll (address operator, bool approved);
- **Event parameters:** NFT owner, authorizer's wallet address, authorization ID;
- **Event definition:** ApprovalForAll (owner, operator, approved);
- **Example:**

```
func TestSetApprovalForAll(t *testing.T) {
```

```
cli := server.NewETHClientByURL(t, url, key)
session, err := e.NewERC1155Session(cli, common.HexToAddress(Address))
if err != nil {
    t.Fatal(err)
}
tx, err := session.SetApprovalForAll(common.HexToAddress(account), true)
if err != nil {
    t.Fatal(err)
}
fmt.Println(fmt.Sprintf("tx Hash: %s", tx.Hash().String()))
}
```

2.6.3.2.2.4 Verify Wallet Authorization

Users can call this interface to verify the wallet authorization.

- **Input parameters:** owner's wallet address, authorizer's wallet address;
- **Output parameters:** Boolean value;
- **Function definition:** `isApprovedForAll (address owner, address operator) view returns (bool);`
- **Example:**

```
func TestIsApprovedForAll(t *testing.T) {
    cli := server.NewETHClientByURL(t, url, key)
    session, err := e.NewERC1155Session(cli, common.HexToAddress(Address))
    if err != nil {
        t.Fatal(err)
    }
    tx, err := session.IsApprovedForAll(common.HexToAddress(owner), common.HexToAddress
(account))
    if err != nil {
        t.Fatal(err)
    }
    fmt.Println(fmt.Sprintf("Is ApprovedForAll:%t", tx))
}
```

2.6.3.2.2.5 Safe Transfer

An NFT owner or authorized wallet address can call this interface to transfer the NFT.

- **Input parameters:** owner's wallet address, receiver's wallet address, NFT token ID, number of copies, attached args;
- **Output parameters:** none;

- **Function definition:** safeTransferFrom (address from, address to,uint256 tokenID,uint256 amount,bytes memory data);
- **Event parameters:** operator, owner's wallet address, receiver's wallet address, NFT token ID, number of copies;
- **Event definition:** TransferSingle (operator, from, to, tokenID, amount);
- **Example:**

```
func TestSafeTransferFrom(t *testing.T) {
    cli := server.NewETHClientByURL(t, url, key)
    session, err := e.NewERC1155Session(cli, common.HexToAddress(Address))
    if err != nil {
        t.Fatal(err)
    }
    data := []byte{0x1}
    tokenId := new(big.Int).SetUint64(1)
    amount := new(big.Int).SetUint64(1)
    tx, err := session.SafeTransferFrom(common.HexToAddress(owner), common.HexToAddress
(account), tokenId, amount, data)
    if err != nil {
        t.Fatal(err)
    }
    fmt.Println(fmt.Sprintf("tx Hash: %s", tx.Hash().String()))
}
```

2.6.3.2.2.6 Batch Safe Transfer

An NFT owner or authorized wallet address can call this interface to transfer NFTs in batch.

- **Input parameters:** owner's wallet address, receiver's wallet address, NFT token ID set, number of copies set, attached args;
- **Output parameters:** none;
- **Function definition:** safeBatchTransferFrom (address from, address to, uint256[] memory tokenIDs, uint256[] memory amounts, bytes memory data);
- **Event parameters:** operator, owner's wallet address, receiver's wallet address, NFT token ID set, number of copies set;
- **Event definition:** TransferBatch (operator, from, to, tokenIDs, amounts);

- **Example:**

```
func TestSafeBatchTransferFrom(t *testing.T) {
    cli := server.NewETHClientByURL(t, url, key)
    session, err := e.NewERC1155Session(cli, common.HexToAddress(Address))
    if err != nil {
        t.Fatal(err)
    }

    var tokenIDs []*big.Int
    tokenIDs = append(tokenIDs, new(big.Int).SetUint64(1), new(big.Int).SetUint64(2))

    var amount []*big.Int
    amount = append(amount, new(big.Int).SetUint64(1), new(big.Int).SetUint64(1))

    data := []byte{0x1, 0x2}
    tx, err := session.SafeBatchTransferFrom(common.HexToAddress(owner), common.HexToAddress(account), tokenIDs, amount, data)
    if err != nil {
        t.Fatal(err)
    }
    fmt.Println(fmt.Sprintf("tx Hash: %s", tx.Hash().String()))
}
```

2.6.3.2.2.7 NFT Destruction

An NFT owner or authorized wallet address can call this interface to destroy the NFT.

- **Input parameters:** owner's wallet address, NFT token ID;
- **Output parameters:** none;
- **Function definition:** burn (address owner, uint256 tokenID);
- **Event parameters:** operator, owner's wallet address, 0x0 address (null address), NFT token ID, number of copies;
- **Event definition:** TransferSingle (operator, owner, address (0), tokenID, amount);
- **Example:**

```
func TestBurn(t *testing.T) {
    cli := server.NewETHClientByURL(t, url, key)
    session, err := e.NewERC1155Session(cli, common.HexToAddress(Address))
    if err != nil {
```

```
t.Fatal(err)
}
tokenId := new(big.Int).SetUint64(1)
tx, err := session.Burn(common.HexToAddress(owner), tokenId)
if err != nil {
    t.Fatal(err)
}
fmt.Println(fmt.Sprintf("tx Hash: %s", tx.Hash().String()))
}
```

2.6.3.2.2.8 Batch Destruction

An NFT owner or authorized wallet address can call this interface to destroy NFTs in batch.

- **Input parameters:** owner's wallet address, NFT token ID set;
- **Output parameters:** none;
- **Function definition:** burnBatch (address owner, uint256[] memory tokenIDs);
- **Event parameters:** operator, owner's wallet address, 0x0 address (null address), NFT token ID set, number of copies set;
- **Event definition:** TransferBatch (operator, owner, address (0), tokenIDs, amounts);
- **Example:**

```
func TestBurnBatch(t *testing.T) {
    cli := server.NewETHClientByURL(t, url, key)
    session, err := e.NewERC1155Session(cli, common.HexToAddress(Address))
    if err != nil {
        t.Fatal(err)
    }
    var tokenIDs []*big.Int
    tokenIDs = append(tokenIDs, new(big.Int).SetUint64(3), new(big.Int).SetUint64(4))

    tx, err := session.BurnBatch(common.HexToAddress(owner), tokenIDs)
    if err != nil {
        t.Fatal(err)
    }
    fmt.Println(fmt.Sprintf("tx Hash: %s", tx.Hash().String()))
}
```

2.6.3.2.2.9 Query Number of NFT Copies

Users can call this interface to query the number of NFT copies held by this wallet address.

- **Input parameters:** owner's wallet address, NFT token ID;
- **Output parameters:** number of copies;
- **Function definition:** balanceOf (address owner, uint256 tokenID) view returns (uint256);
- **Example:**

```
func TestBalanceOf(t *testing.T) {
    cli := server.NewETHClientByURL(t, url, key)
    session, err := e.NewERC1155Session(cli, common.HexToAddress(Address))
    if err != nil {
        t.Fatal(err)
    }
    tokenId := new(big.Int).SetUint64(3)
    tx, err := session.BalanceOf(common.HexToAddress(owner), tokenId)
    if err != nil {
        t.Fatal(err)
    }
    fmt.Println(fmt.Sprintf("Account balance: %s", tx))
}
```

2.6.3.2.2.10 Batch Query the Number of NFT Copies

Users can call this interface to query the number of NFT copies held by this wallet address in batches.

- **Input parameters:** owner's wallet address set, NFT token ID set;
- **Output parameters:** number of copies set;
- **Function definition:** balanceOfBatch (address[] memory owners,uint256[] memory tokenIDs) view returns (uint256[] memory);
- **Example:**

```
func TestBalanceOfBatch(t *testing.T) {
    cli := server.NewETHClientByURL(t, url, key)
    session, err := e.NewERC1155Session(cli, common.HexToAddress(Address))
    if err != nil {
        t.Fatal(err)
    }
    var tokenIDs []*big.Int
    tokenIDs = append(tokenIDs, new(big.Int).SetUint64(3), new(big.Int).SetUint64(4))
}
```



```
var owners []common.Address
owners = append(owners, common.HexToAddress(owner), common.HexToAddress(account))

tx, err := session.BalanceOfBatch(owners, tokenIDs)
if err != nil {
    t.Fatal(err)
}
fmt.Println(fmt.Sprintf("owners balance: %s", tx))
}
```

2.6.3.2.2.11 Query NFT Name

Users can call this interface to query the name of the NFT.

- **Input parameters:** NFT token ID;
- **Output parameters:** NFT name;
- **Function definition:** tokenName (uint256 tokenID) view returns (string memory);
- **Example:**

```
func TestTokenName(t *testing.T) {
    cli := server.NewETHClientByURL(t, url, key)
    session, err := e.NewERC1155Session(cli, common.HexToAddress(Address))
    if err != nil {
        t.Fatal(err)
    }
    tokenId := new(big.Int).SetUint64(3)
    tx, err := session.TokenName(tokenId)
    if err != nil {
        t.Fatal(err)
    }
    fmt.Println(fmt.Sprintf("NFT name: %s", tx))
}
```

2.6.3.2.2.12 Query NFT Symbol

Users can call this interface to query the NFT symbol.

- **Input parameters:** NFT token ID;
- **Output parameters:** NFT symbol;
- **Function definition:** tokenSymbol (uint256 tokenID) view returns (string memory);

- **Example:**

```
func TestTokenSymbol(t *testing.T) {
    cli := server.NewETHClientByURL(t, url, key)
    session, err := e.NewERC1155Session(cli, common.HexToAddress(Address))
    if err != nil {
        t.Fatal(err)
    }
    tokenId := new(big.Int).SetUint64(2)
    tx, err := session.TokenSymbol(tokenId)
    if err != nil {
        t.Fatal(err)
    }
    fmt.Println(fmt.Sprintf("NFT symbol: %s", tx))
}
```

2.6.3.2.2.13 Query NFT URI

Users can call this interface to query the NFT URI.

- **Input parameters:** NFT token ID;
- **Output parameters:** NFT URI;
- **Function definition:** tokenURI (uint256 tokenId) view returns (string memory);
- **Example:**

```
func TestTokenURI(t *testing.T) {
    cli := server.NewETHClientByURL(t, url, key)
    session, err := e.NewERC1155Session(cli, common.HexToAddress(Address))
    if err != nil {
        t.Fatal(err)
    }
    tokenId := new(big.Int).SetUint64(2)
    tx, err := session.TokenURI(tokenId)
    if err != nil {
        t.Fatal(err)
    }
    fmt.Println(fmt.Sprintf("NFT URI: %s", tx))
}
```

2.6.3.2.2.14 Query the Latest NFT Token ID

Users can call this interface to query the latest NFT token ID.

- **Input parameters:** none;
- **Output parameters:** Latest NFT token ID;
- **Function definition:** getLatestTokenID() view returns (uint256);
- **Example:**

```
func TestGetLatestTokenID(t *testing.T) {  
    cli := server.NewETHClientByURL(t, url, key)  
    session, err := e.NewERC1155Session(cli, common.HexToAddress(Address))  
    if err != nil {  
        t.Fatal(err)  
    }  
    tx, err := session.GetLatestTokenID()  
    if err != nil {  
        t.Fatal(err)  
    }  
    fmt.Println(fmt.Sprintf("tokenId: %s", tx))  
}
```

3 Info on the Non-Cryptocurrency Public Chains

A Non-Cryptocurrency Public Chain is a transformed public chain framework based on an existing public chain. Gas Credit transfers are not permitted between standard wallets. There are no cryptocurrency incentives for mining or participating in consensus. On Spartan Network, there are three Non-Cryptocurrency Public Chains at launch. We expect to add more in the foreseeable future.

3.1 Spartan-I Chain (Powered by NC Ethereum)

3.1.1 About Spartan-I Chain (Powered by NC Ethereum)

The Spartan-I Chain is a blockchain compatible with Ethereum that runs independently from the public Ethereum blockchain. Full Nodes, which can freely join and exit the Spartan Network, synchronize block information of the entire chain and submit transaction requests to the network.

A Spartan-I full node runs an EVM (Ethereum Virtual Machine) that allows developers to use Solidity programming language to create smart contracts that are compatible with the Ethereum network. Also, all the different tools and wallets available for Ethereum (such as Truffle, HardHat, Metamask, etc...) can be directly used with Spartan-I Chain.

Ethereum-based networks have two identifiers, a network ID and a chain ID. Although they often have the same value, they have different uses. Peer-to-peer communication between nodes uses the network ID, while the transaction signature process uses the chain ID.

Spartan-I Chain Network Id = Chain Id = 9090

For detailed installation documentation, please refer to [GitHub](#).

3.1.2 Ethereum and Geth Documentation

Below is a list of useful online documentation about Ethereum and Geth.

How to set up Geth and execute some basic tasks using the command line tools.

<https://geth.ethereum.org/docs/getting-started>

JSON-RPC API methods Interacting with Geth requires sending requests to specific JSON-RPC API methods. Geth supports all standard JSON-RPC API endpoints. You can send RPC requests on the port 8545.

<https://geth.ethereum.org/docs/rpc/server>

Developer Documentation

This documentation is designed to help users build with Ethereum. It covers Ethereum as a concept, explains the Ethereum tech stack, and documents advanced topics for more complex applications and use cases.

<https://ethereum.org/en/developers/docs/>

Smart Contract tutorials

A list of curated Ethereum tutorials to learn about coding smart contracts and DApps.

<https://ethereum.org/en/developers/tutorials/>

Solidity Lang

<https://docs.soliditylang.org/>

Web3j Document

<https://docs.web3j.io/>

Web3.js

<https://web3js.readthedocs.io/en/v1.7.5/>

3.2 Spartan-II Chain (Powered by NC Cosmos)

3.2.1 About Spartan-II Chain (Powered by NC Cosmos)

The Spartan-II Chain is a Cosmos-based network which has two identifiers, a network ID and a chain ID. Although they often have the same value, they have different uses. Peer-to-peer communication between nodes uses the network ID, while the transaction signature process uses the chain ID.

EVM module: Network ID = Chain ID = 9003

Native module: Network ID = Chain Id = starmint

For detailed installation instructions, please refer to [Github](#).

3.2.2 Resources

To find out more about Spartan-II Chain (Powered by NC Cosmos), visit [GitHub](#).

API Introduction

<https://github.com/BSN-Spartan/NC-Cosmos/tree/main/docs/endpoints>

CLI Client Commands

<https://github.com/BSN-Spartan/NC-Cosmos/tree/main/docs/cli-client>

Solidity Lang

<https://docs.soliditylang.org/>

Web3j Document

<https://docs.web3j.io/>

Spartan-II Chain GoLang SDK

<https://github.com/BSN-Spartan/nc-cosmos-sdk-go>

3.3 Spartan-III Chain (Powered by NC PolygonEdge)

3.3.1 About Spartan-III Chain (Powered by NC PolygonEdge)

The Spartan-III Chain (Powered by NC PolygonEdge) network has two identifiers, a network ID and a chain ID. Although they often have the same value, they have different uses. Peer-to-peer communication between nodes uses the network ID, while the transaction signature process uses the chain ID.

Spartan-III Chain Network Id = Chain Id = 5566

For detailed installation instructions, please refer to [GitHub](#).

3.3.2 Resources

3.3.2.1 JSON-RPC Commands

NC PolygonEdge is compatible with ETH JSON RPC interface, please refer to the detailed interface list from below link.

<https://docs.polygon.technology/docs/edge/get-started/json-rpc-commands>

3.3.2.2 CLI Commands

NC PolygonEdge provides a wealth of CLI commands for managing your nodes. For a detailed command list, please refer to the link below.

<https://docs.polygon.technology/docs/edge/get-started/cli-commands>

3.3.2.3 Prometheus Metrics

PolygonEdge can report and serve the Prometheus metrics, which in their turn can be consumed using Prometheus collector(s).

The following is a detailed description reference.

<https://docs.polygon.technology/docs/edge/configuration/prometheus-metrics>

3.3.2.4 Backup/Restore Node Instance

This guide goes into detail on how to back up and restore a PolygonEdge node instance. It covers the base folders and what they contain, as well as which files are critical for performing a successful backup and restore.

For detailed operation, please refer to the link below.

<https://docs.polygon.technology/docs/edge/working-with-node/backup-restore>

Polygon Edge API

<https://docs.polygon.technology/docs/edge/get-started/json-rpc-commands>

How to Use Smart Contracts

<https://docs.polygon.technology/docs/category/smart-contracts>

Solidity Lang

<https://docs.soliditylang.org/>

Web3j Document

<https://docs.web3j.io/>

Web3.js

<https://web3js.readthedocs.io/en/v1.7.5/>

4 FAQs

4.1 Frequently Asked Questions

Find answers and solutions for commonly seen errors and areas of confusion within our curated series of FAQ articles.

Don't see your question answered here? Please [contact us](#).

4.2 What is a Wallet Address?

In Spartan Non-Cryptocurrency Public Chains, a Wallet Address is a Gas Credit receiving address, which is a unique sequence of hexadecimal numbers.

4.3 What is a Private Key?

A private key is a secret number that is used in cryptography, similar to a password. In cryptocurrency, private keys are also used to sign transactions and prove ownership of a blockchain address.

4.4 What is a Virtual Data Center?

A Virtual Data Center is a set of locally installed software systems that contains one or more registered full nodes of different NC Public Chains. Each Virtual Data Center has one NTT wallet and is eligible to receive Node Establishment and Data Center Monthly Incentives.

4.5 What is Gas Credit?

In a similar fashion to cryptocurrencies, Gas Credits are used as a means of paying Gas fees on NC Public Chains. However, Gas Credits cannot be transferred between standard wallets. Only the Data Center Operator's NTT wallet can be used to purchase Gas Credits with NTT.