BRC-985 Inscription Spec

Creator: The creator of the inscription.

Create Time: The time when the inscription was created.

Owner: The owner of the inscription which is inscribed on satoshi. when the satoshi is transferred to another address via UTXO, the owner of the inscription changes.

Token

Deploy

```
    {
        "p": "brc-985-token",
        "op": "deploy",
        "tick": "memo",
        "max": "21000000",
        "lim": "1000"
    }
```

key	Required	Description
p	yes	Protocol name
op	yes	Operation
tick	yes	Ticker
max	yes	Maximum token supply
lim	no	Limit pre mint

1. Check the inscription's value according to the following rules:
    - Check if the protocol name is **"brc-985-token"**;
    - Checks if the token represented by **"tick"** exists in the token list; if it does, then the deploy inscription is invalid;
    - If **"lim"** is set, then **"lim"** should not be greater than **"max"**.
2. If the above checks pass, then the deploy inscription is valid.
3. Update the token list, add the token information to the token list.

Mint

```
{
    "p": "brc-985-token",
    "op": "mint",
    "tick": "memo",
    "amt": "1000"
}
```

key	Required	Description
p	yes	Protocol name
op	yes	Operation
tick	yes	Ticker
amt	yes	Minting amount. If "lim" is set, then "amt" must be less than "lim"

1. Check the inscription's value according to the following rules:
    - Check if the protocol name is "**brc-985-token**";
    - Check if the token represented by "**tick**" exists in the token list; if it does not, then the mint inscription is invalid;
    - Check if the Creator of the mint inscription is consistent with the Owner of the deploy inscription; if not, then the mint inscription is invalid;
    - Check if the "**amt**" is less than "**limit**";
    - Check if the minted token amount plus "**amt**" is greater than "**max**".
2. If the above checks pass, then the mint inscription is valid.
3. Update the available balance of the Creator of the mint inscription, add "**amt**";
4. Update the minted token amount, add "**amt**".

Transfer

```
{
    "p": "brc-985-token",
    "op": "transfer",
    "tick": "memo",
    "amt": "100"
}
```

key	Required	Description
p	yes	Protocol name
op	yes	Operation
tick	yes	Ticker
amt	yes	The amount of tokens to be transferred

Note: The transfer of tokens is divided into two steps, first create the transfer inscription, then send the transfer inscription to the target address, and Each transfer inscription can only be used once. And the token's balance is divided into available balance and transferable balance.

Note: The transfer of tokens is only related to the transfer inscription, the transfer of the mint inscription (sending the mint inscription to others through UTXO) does not lead to a change in balance.

1. Check the inscription's value according to the following rules:
    - Check if the protocol name is **"brc-985-token"**;
    - Check if the token represented by **"tick"** exists in the token list; if it does not, then the mint inscription is invalid;
    - Check if the available balance of the Owner of the transfer inscription is greater than **"amt"**; if it is less, then the transfer inscription is invalid.
2. If the above checks pass, then the mint inscription is valid;
3. Update the available balance of the Owner of the transfer inscription, subtract **"amt";**
4. Update the transferable balance of the Owner of the transfer inscription, add **"amt".**

DA

Deploy

```
{
    "p": "brc-985-da",
    "op": "deploy",
    "tick": "mooda",
    "storage": "",
    "foundation": "",
    "interval": "",
    "token":"",
    "price":""
}
```

key	Required	Description
p	yes	Protocol name
op	yes	Operation
tick	yes	Ticker
storage	yes	Storage address (submit proof regularly)
foundation	yes	Foundation address (receives the profit when the proof fails)
interval	yes	Proof submission cycle time (in seconds)
token	yes	Accepted token identifier
price	yes	Fee for each upload

1. Check the inscription's value according to the following rules:
   - Check if the protocol name is "**brc-985-da**";
   - Checks if the DA represented by "**tick**" exists in the DA list; if it does, then the deploy inscription is invalid;
   - Check if the token represented by "**token**" exists in the token list; if it does not, then the deploy inscription is invalid;
2. If the above checks pass, then the mint inscription is valid;
3. Use the hash value of "**p**" and "**tick**" as the default receiving address for the DA.
4. Update the DA list, add the basic information of the DA to the DA list.

Upload

```
{
    "p": "brc-985-da",
    "op": "upload",
    "tick": "mooda",
    "id": "",
    "signature": ""
}
```

key	Required	Description
p	yes	Protocol name
op	yes	Operation
tick	yes	Ticker
id	yes	Data commitment
signature	yes	The Creator of the uploaded inscription signs the id

Note 1: In addition to uploading data to the DA, the Creator of the upload inscription also needs to pay "price".

Note 2: When the Indexer synchronizes the first Upload inscription information, set the creation time of the inscription as the start time "start".

1. Check the inscription's value according to the following rules:
   - Check if the protocol name is "**brc-985-da**";
   - Checks if the DA represented by "**tick**" exists in the DA list; if it does not, then the upload inscription is invalid;
   - Check if the available balance of the Creator of the upload inscription is sufficient; if the available balance is not enough to pay "**price**", then the upload inscription is invalid;
   - Check the format of the data commitment id;
   - Verify the signature, the signature method is as follows:
     - Original message: sort the id in dictionary order and concatenate in the form of key=value. For example: id=xxx.
     - The original message is hashed twice using SHA256 to generate the message digest "**h**"
     - "**signature**" is generated by signing the message digest "**h**" with the private key of Creator using the ECDSA secp256k1 algorithm, and encoded in base64.
2. If the above checks pass, then the upload inscription is valid;
3. Update the available balance of the Creator of the upload inscription, sub "**price**";
4. Update the available balance of the default receiving address, add "**price**";

Epoch

```
{
    "p": "brc-985-da",
    "op": "epoch",
    "tick": "mooda"
}
```

Note 1：Whenever a new challenge period begins, the Epoch operation must be performed first, and then the Prove operation can be carried out. The core purpose of the Epoch operation is to establish the state of the challenge period, clarify which data will be involved in the proof generation process, and determine the verifiable random number.

Note 2: The Epoch operation will additionally check whether there is any unsubmitted situation in the previous period. If there is, the profit will be transferred to the foundation.

1. Check the inscription's value according to the following rules:
   - Check if the protocol name is "**brc-985-da**";
   - Checks if the DA represented by "**tick**" exists in the DA list; if it does not, then the upload inscription is invalid;
   - Check if the creation time of the inscription is greater than "**start**"; if it is greater, then pass; if it is not greater, then continue to check whether the current state is unsubmitted; if it is unsubmitted, then the Epoch inscription is invalid.
2. If the above checks pass, then the epoch inscription is valid;
3. Calculate the profit of the foundation address according to the following rules:
   - Use the creation time of the inscription as the current time;
   - Calculate the number of intervals and the interval time between the current time and "**start**"; for example, when the start time is 1000, the period is 200, and the current time is 1523, then the number of intervals is (1523-1000)/200=2, and the interval time is 2*200=400;
   - Calculate the profit based on the number of intervals and the balance of the default receiving address; for example, if the interval is 2 and the balance of the default receiving address is 100memo, then the profit is 100*1%*2=2memo.
   - Update the available balance of "**foundation**", add the profit value;
   - Update the available balance of the default receiving address, subtract the profit value;
   - Update "**start**", add the interval time.
4. Specify that all data specified in the data commitment list will participate in the subsequent proof generation process;
5. Specify that the hash value of the creation time of the inscription and the last random number is the latest verifiable random number;
6. Set the state of the DA to unsubmitted.

Prove

```
{
    "p": "brc-985-da",
    "op": "prove",
    "tick": "mooda",
    "proof": ""
}
```

key	Required	Description
p	yes	Protocol name
op	yes	Operation
tick	yes	Ticker
proof	yes	data access proof

Note：In addition to proving the availability of all data, the "storage" should also receive profit from the default receiving address.

1. Check the inscription's value according to the following rules:
   - Check if the protocol name is **"brc-985-da"**;
   - Checks if the DA represented by **"tick"** exists in the DA list; if it does not, then the upload inscription is invalid;
   - Check if the current cycle has already uploaded proof; if proof has been uploaded, then the proof inscription is invalid;
   - Check if the Creator of the prove inscription is consistent with **"storage"**; if not, then the prove inscription is invalid;
   - Verify the correctness of the data access proof;
2. If the above checks pass, then the mint inscription is valid;
3. Update the available balance of **"storage"**, add 1% of the default receiving address's available balance;
4. Update the available balance of the default receiving address, subtract 1% of the default receiving address's available balance;
5. Update **"start"**, add **"interval"**;
6. Change the state of the DA to submitted.

NFT

Deploy

```
{
    "p": "brc-985-nft",
    "op": "deploy",
    "tick": "mnft",
    "da": "mooda",
    "max": "1000",
    "description": "Bitcoin NFT",
    "id": "",
    "sig": ""
}
```

Key	Required	Description
p	是	Protocol name
op	是	Operation
tick	是	Ticker
da	是	Specifies the BRC-985-DA system that stores the NFT content
max	是	Maximum NFT issuance
description	是	Description of the NFT collections
id	是	The data ID that describes the NFT collections
sig	是	The signature of the Creator of the deploy inscription on id

Note: The Deploy operation will upload an image related to the NFT collections. So, the Creator should pay "price" to "storage".

1. Check the inscription's value according to the following rules:
    - Check if the protocol name is "brc-985-nft";
    - Checks if the NFT represented by "**tick**" exists in the NFT list; if it does, then the deploy inscription is invalid;
    - Checks if the DA represented by "**da**" exists in the DA list; if it does not, then the deploy inscription is invalid;
    - Verify the signature, the signature method is as follows:
        - Original message: sort the id in dictionary order and concatenate in the form of key=value. For example: id=xxx.

Mint

```
{
    "p": "brc-985-nft",
    "op": "mint",
    "tick": "mnft",
    "id": "",
}
```

Key	Required	Description
p	是	Protocol name
op	是	Operation
tick	是	Ticker
id	是	The data ID that describes the NFT
signature	是	The signature of the Creator of the mint inscription on id

Note: The Mint operation not only mints an NFT but also uploads the NFT data content to the DA. That is, the Creator should pay "price" to "storage".

1. Check the inscription's value according to the following rules:
    - Check if the protocol name is "brc-985-nft";
    - Checks if the NFT represented by "**tick**" exists in the NFT list; if it does not, then the mint inscription is invalid;
    - Check if the NFT corresponding to "**id**" already exists; if it does, then the mint inscription is invalid;
    - Check if the available balance of the Creator of the mint inscription is sufficient; if the balance is not enough to pay "**price**", then the mint inscription is invalid;
    - Verify the signature, the signature method is as follows:
        - Original message: sort the id in dictionary order and concatenate in the form of key=value. For example: id=xxx.
        - The original message is hashed twice using SHA256 to generate the message digest "**h**"
        - "**signature**" is generated by signing the message digest "**h**" with the private key of Creator using the ECDSA secp256k1 algorithm, and encoded in base64.
    - Check the format of "**id**";
2. If the check passes, then the inscription is valid;
3. Update the available balance of the Creator of mint inscription, subtract "**price**";
4. Update the available balance of the default receiving address, add "**price**";