Inscription

Creator：铭文的创建者；

Create Time：铭文的创建时间；

Owner：铭文的所有者，铭文所在聪的拥有者，当该聪通过 UTXO 转移到其他地址后，铭文的所有者发生变化；

Token

Deploy

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

key	Required	Description
p	yes	协议名
op	yes	操作
tick	yes	标识符
max	yes	代币最大供应量
lim	no	单次铸币限制

1. 按照如下规则检查铭文的值：
    - 检查协议名是否为 **brc-985-token**；
    - 在代币列表中查看 **tick** 代表的代币是否存在；若存在，则该部署铭文无效；
    - 若设置铸币限制 **lim**，则铸币限制 **lim** 不应大于代币最大供应量 **max**。
2. 若上述检查通过，则部署铭文有效。
3. 更新代币列表，将该代币信息添加到代币列表中。

Mint

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

key	Required	Description
p	yes	协议名
op	yes	操作
tick	yes	标识符
amt	yes	铸币数量，如果 deploy 有 lim，则每次铸币必须小于 lim

1. 按照如下规则检查铭文的值：
    - 检查协议名是否为 **brc-985-token**；
    - 在代币列表中查看 **tick** 代表的代币是否存在；若不存在，则该铸造铭文无效；
    - 检查铸造铭文的 **Creator** 地址和部署铭文的 **Owner** 地址是否一致；若不一致，则该铸造铭文无效；
    - 若在部署时设置了铸币限制 **lim**，则每次铸币值 **amt** 必须小于铸币限制 **lim；**
    - 检查铸币数量是否已超过代币的最大供应量；即验证已铸造代币量加上本次铸造的代币量是否大于代币的最大供应量。
2. 若上述检查通过，则铸造铭文有效；
3. 更新铸造铭文的 **Creator** 地址的可用余额，加上铸币数量 **amt**；
4. 更新已铸造代币量，加上铸币数量 **amt**。

Transfer

key	Required	Description
p	yes	协议名

op	yes	操作
tick	yes	标识符
amt	yes	转移代币的数量

注：token 的转移分为两步，首先创建转移铭文，随后将转移铭文发送到目标地址，且转移铭文仅第一次转让有效。并将 token 的余额分为可用余额以及可转让余额。

注：token 的转移仅与转移铭文相关，铸造铭文的转移（通过 UTXO 将铸造铭文发送给其他人）并不会导致余额的变化。

1. 按照如下规则检查铭文的值：
    - 检查协议名是否为 **brc-985-token**；
    - 在代币列表中查看 **tick** 代表的代币是否存在；若不存在，则该转移铭文无效；
    - 检查转移铭文的接收地址的可用余额是否大于转移代币的数量 **amt；**若小于，则该转移铭文无效。
2. 若上述检查通过，则转移铭文有效；
3. 更新转移铭文的接收地址的可用余额，减去转移代币的数量 **amt**；
4. 更新目标地址的可转移余额，加上转移代币的数量 **amt**。

DA

Deploy

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

key	Required	Description
p	yes	协议名
op	yes	操作
tick	yes	标识符
storage	yes	存储地址（定期提交证明）
foundation	yes	基金会地址（收取证明失败时收益）
interval	yes	证明提交周期时间（单位为秒）
token	yes	接受的代币
price	yes	每次上传时所需要支付的费用

1. 按照如下规则检查铭文的值：
   - 检查协议名是否为 **brc-985-da**；
   - 在 DA 列表中查看 **tick** 代表的 DA 是否存在；若存在，则该部署铭文无效；
   - 在代币列表中查看 **token** 表示的代币是否存在；若不存在，则该部署铭文无效；
2. 若上述检查通过，则部署铭文有效；
3. 将协议名 **p** 以及标识符 **tick** 的哈希值，作为 DA 的默认收款地址。
4. 更新 DA 列表，将该 DA 的基本信息添加到 DA 列表中。

Upload

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

key	Required	Description
p	yes	协议名
op	yes	操作
tick	yes	标识符
id	yes	数据承诺
signature	yes	上传铭文的 Creeator 对 id 的签名

注 1：上传操作除了将数据上传至 DA 外，发起地址还需要支付一定的费用，即上传操作包含隐示的转账，将发起地址的部分 token 转移到 DA 的收款地址。

1. 按照如下规则检查铭文的值：
   - 检查协议名是否为 **brc-985-da**；
   - 在 DA 列表中查看 **tick** 代表的 DA 是否存在；若不存在同名的 DA，则该上传铭文无效；
   - 检查上传铭文的 **Creator** 地址的余额是否足够；若余额不足以支付上传费用 **price**，则该上传铭文无效；
   - 检查数据承诺 **id** 的格式；
   - 验证签名，签名方式如下：
     - 构建签名原始信息 **message：**按照字典序对 id 排序，并按照 **key=value** 的方式拼接。例如：**id=xxx**。
     - 对原始信息 **message** 进行两次 SHA256 哈希，得到哈希值 **h**；
     - 使用上传铭文的 **Creator** 对应的私钥对哈希值 **h** 使用 ECDSA secp256k1 签名，得到签名 **s**，将签名 **s** 进行 base64 编码得到 **signature**。

Epoch

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

注 1：Epoch 操作是 Prove 的前置操作，每当进入一个新的挑战周期，必须首先执行 Epoch 操作，之后才能进行 Prove 操作。这一步骤是为了防止在证明生成过程中，由于数据上传操作导致证明验证失败。Epoch 操作的核心目的是确立挑战周期的状态，明确哪些数据将参与到证明生成的过程中，并且确定在后续 Prove 操作中的可验证随机数。

注 2：Epoch 操作会额外检查在之前的周期内，是否存在未提交的情况，若存在，则原本属于存储地址的收益将转给基金会地址。

1. 按照如下规则检查铭文的值：
   - 检查协议名是否为 **brc-985-da**；
   - 在 DA 列表中查看 **tick** 代表的 DA 是否存在；若不存在同名的 DA，则该 Epoch 铭文无效；
   - 检查铭文的创建时间是否大于起始时间 **start**；若大于则通过；若不大于则继续检查当前状态是否为未提交；若为未提交，则该 Epoch 铭文无效。
2. 若上述检查通过，则 Epoch 铭文有效；
3. 按照如下规则计算基金会地址的收益：
   - 将铭文的创建时间作为当前时间；
   - 计算当前时间与起始时间 **start** 的间隔周期数以及间隔周期时间；例如当起始时间为 1000，周期为 200，当前时间为 1523 时，则间隔周期数为(1523-1000)/200=2，间隔周期时间为 2*200=400；
   - 根据间隔周期数以及收款地址的余额，计算罚款；例如间隔 2 个周期，收款地址的余额为 100memo，则罚款总额为 100*1%*2=2memo。
   - 更新基金会地址 **foundation** 的余额，加上罚款值；
   - 更新收款地址的余额，减去罚款值；
   - 更新起始时间 **start**，加上间隔周期时间，以 2 中例子为例，会将起始时间更新到 1400。
4. 规定所有在数据承诺列表中指定的数据都将参与后续证明生成过程；

Prove

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

key	Required	Description
p	yes	协议名
op	yes	操作
tick	yes	标识符
proof	yes	数据证明

注 1：证明操作除了证明所有数据的可用性之外，存储地址应收取部分费用，该功能包含隐示的转账，将收款地址的部分 token 转移到存储地址。

1. 按照如下规则检查铭文的值：
   - 检查协议名是否为 **brc-985-da**；
   - 在 DA 列表中查看 **tick** 代表的 DA 是否存在；若不存在，则该证明铭文无效；
   - 检查当前周期是否已上传证明；若上传过证明，则该证明铭文无效；
   - 检查创造证明铭文的地址和存储地址是否一致；若不一致，则证明铭文无效；
   - 验证数据证明的正确性；
2. 若检查通过，则证明铭文有效；
3. 更新存储地址 **storage** 的余额，加上收款地址余额的 1%；
4. 更新收款地址的余额，减去收款地址余额的 1%；
5. 更新起始时间 **start**，加上证明提交周期 **interval**；
6. 将 DA 的状态改为已提交。

NFT

Deploy

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

Key	Required	Description
p	是	协议标识符：帮助其他系统识别和处理 brc-985 的 NFT 事件
op	是	操作：事件类型（deploy, mint）
tick	是	brc-985-nft 的唯一标识符
da	是	指定存储 NFT 数据内容的 BRC-985-DA 系统
max	是	NFT 最大发行量
description	是	NFT 的描述
id	是	描述 NFT 的图片数据 id
signature	是	部署铭文的 Creator 对 id 的签名

注：Deploy 操作除了部署 NFT 集合外，还会将 NFT 集合的描述图片上传至 DA。即 Deploy 操作包含隐式的上传操作，同样需要支付给 DA 中 storage 一定的费用。

1. 按照如下规则检查铭文的值：
    - 检查协议名是否为 **brc-985-nft**；
    - 在 NFT 列表查看 **tick** 代表的 NFT 是否存在；若存在同名的 NFT，则部署铭文无效；
    - 在 DA 列表中查看 **da** 代表的 DA 是否存在；若不存在同名的 DA，则部署铭文无效；
    - 检查部署铭文的 **Creator** 地址的余额是否足够；若余额不足以支付上传费用 **price**，则该铸造铭文无效；
    - 验证签名，签名方式如下：

Mint

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

Key	Required	Description
p	是	协议标识符：帮助其他系统识别和处理 brc-985 的 NFT 事件
op	是	操作：事件类型（deploy, mint）
tick	是	brc-985-nft 的唯一标识符
id	是	NFT 内容的数据承诺值
signature	是	铸造铭文的 Creeator 对 id 的签名

注：Mint 操作除了铸造一枚 NFT 外，还会将 NFT 的数据内容上传至 DA。即 Mint 操作包含隐式的上传操作，同样需要支付给 DA 中 storage 一定的费用。

1. 按照如下规则检查铭文的值：
    - 检查协议名是否为 **brc-985-nft**；
    - 在 NFT 列表查看 **tick** 代表的 NFT 是否存在；若不存在同名的 NFT，则铸造铭文无效；
    - 检查 **id** 对应的 NFT 是否已存在；若存在，则铸造铭文无效；
    - 检查铸造铭文的 **Creator** 地址的余额是否足够；若余额不足以支付上传费用 **price**，则该铸造铭文无效；
    - 验证签名，签名方式如下：
        - 构建签名原始信息 **message：**按照字典序对 id 排序，并按照 **key=value** 的方式拼接。例如：**id=xxx**。
        - 对原始信息 **message** 进行两次 SHA256 哈希，得到哈希值 **h**；
        - 使用上传铭文的 **Creator** 对应的私钥对哈希值 **h** 使用 ECDSA secp256k1 签名，得到签名 **s**，将签名 **s** 进行 base64 编码得到 **signature**。
    - 检查数据承诺 **id** 的格式；
2. 若检查通过，则铭文有效；

3. 更新创造上传铭文的地址的余额，减去上传费用 **price**；
4. 更新收款地址的余额，加上上传费用 **price**；
5. 将该 NFT 添加到 NFT 列表中；
6. 将该 NFT 的所有人设置为该铭文的 Creator。

Transfer

NFT 的转移很简单，只需要将 Mint 铭文发送给指定地址即可。