MetaID v0.1协议说明

brfc	title	authors	version
d2c53c87b1f5	MetaID Protocol	撰写：ShowPayTeam 校对：MetaSVTeam	V0.1

1. 简介

MetaID是Metanet[1]的二级协议，旨在解决BSV上应用之间用户信息的无法连通而提出的协议。

MetaID希望越来越多钱包方和应用方都支持MetaID协议，从而达成以下的MetaID目标：

- 用户在BSV网络上只需一个主私钥对就可以使用所有支持MetaID的应用；
- 用户基本信息和应用交易数据记录在自己掌握的Metanet节点上，做到用户数据和钱包方和应用方无关，用户彻底掌握自己数据。
- 不同应用间的数据可以相互联通，消除应用间的信息孤岛状况；
- 不同协议的数据可以在MetaID关联下能相互组合，BSV应用开发工作大为减少；
- MetaID具有灵活的扩展性，应用/协议制定方能方便地加入或修改其需要的协议，从而支持各种各样的应用；

2. MetaID协议

MetaID整体协议格式为：

```
<Metanet Flag><P(node)><TxID(parent)><MetaID Flag><node_name><data><encrypt>
<version><data_type><encoding>
```

其中前三个元素为Metanet协议标准部分，后七个元素为MetaID二级协议的扩展部分。MetaID协议遵循Metanet协议，通过包含 的OP_0 OP_RETURN操作码来创建。

2.1 Metanet标准部分

其中前面三个元素即：

```
<Metanet Flag><P(node)><TxID(parent)>
```

为Metanet的标准格式：

- < Metanet Flag >固定为“meta”；
- < P(node) >为节点的公钥；
- < TxID(parent) >为父节点的交易ID

更详细的解析Metanet构造协议，请参考Metanet协议白皮书。

2.2 MetaID二级协议部分

后七部分，即：

```
<MetaID Flag><node_name><data><encrypt><version><data_type><encoding>
```

为MetaID作为Metanet二级协议所增加的内容，具体含义如下：

key	value
MetaID Flag	固定为"MetaID"
node_name	节点标识名字，必须字段。
data	存储节点所对应的数据内容
encrypt	标识该节点内容是否加密。本协议版本支持两种方式：0为不加密；1为ECIES加密，即加密key为对应节点的公钥，采用对应节点路径的私钥解密。默认为0不加密。
version	节点类型的版本号，不同版本号意味着data内容的格式不相同。
data_type	可选项目。data对应的数据类型，可用数据类型请参考：https://www.iana.org/assignments/media-types/media-types.xhtmls。默认为text/plain
encoding	可选项目。data对应的编码格式，可用编码类型请参考：https://www.iana.org/assignments/character-sets/character-sets.xhtml。默认为UTF-8，

一条标准的MetaID交易格式应为如下：

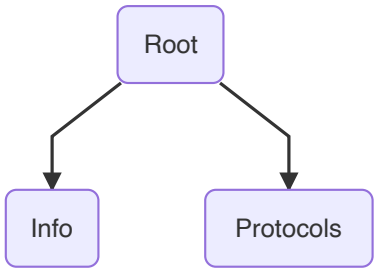
$TxID_{node}$

Inputs	Outputs
< Sig P_{parent} > <P_{parent} >	OP_0 OP_RETURN < Metanet Flag > < P_{node} > < $TxID_{parent}$ > < MetaID Flag > < node_ame > < data >< encrypt >< version >< data_type >< encoding >

3. MetaID结构

本版本协议中，MetaID的根节点下只有两个子节点，分别是"Info"和"Protocols"。

- Info 为用户基本信息节点。记录用户姓名、头像之类的基本信息。
- Protocols 为协议节点。记录用户使用相关协议所产生的交易。



MetaID固定必须有的节点为Root和Info和Protocols。此三个节点约定不能更改名称，不接受Metanet的版本更新操作。

3.1 Root节点

Root节点是MetaID的顶点，根据Metanet协议，构建Metanet交易时$TxID_{parent}$为空即为顶点。

构造Root节点时，node_ame固定为“Root”。以下为一个构建顶点的例子：

```
OP_0 OP_RETURN meta <P(node)> NULL MetaID Root NULL NULL NULL NULL NULL NULL
```

由于是顶点节点，所以$TxID_{parent}$需为NULL。然后设置 Root节点名称为Root，其后的部分均不需要，所以都设置为NULL即可。

MetaID相关P_{node}节点的public key生成约定采用HD方案[2]生成，Root 节点地址约定使用hd方案 0/0 路径。

3.2 Info节点

Info节点为存储用户基本信息的节点 ， node_name固定为"Info"。

Info节点的构造和Root相似，只是$TxID_{parent}$需指向Root节点并将node_name设置为”Info“即可，以下为构建Info节点例子：

```
OP_0 OP_0 OP_RETURN meta <P(node)> <Root TxID> MetaID Info NULL NULL NULL NULL
NULL NULL
```

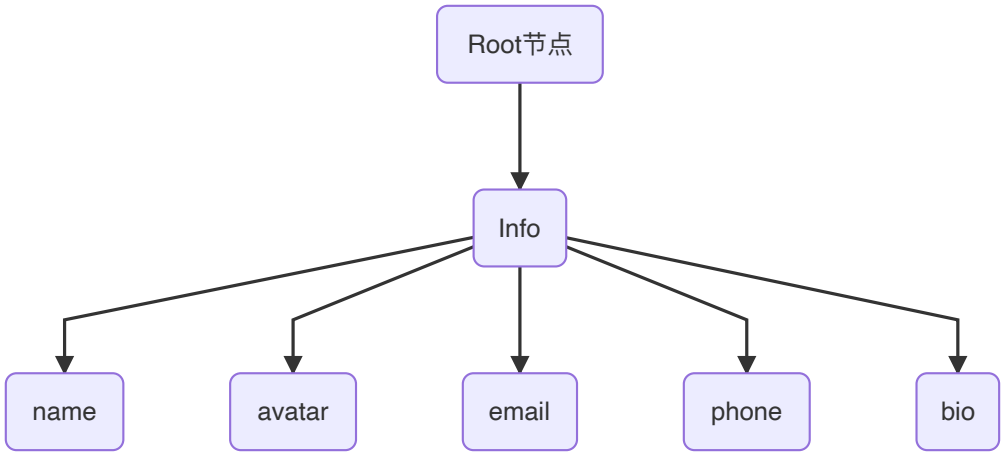
关于用户基本信息

本版本协议中，用户基本信息放在Info节点下。Info节点固定包含以下5个子节点：

- name 用户名，建议不加密。格式固定为text/plain
- avatar 用户图像，建议不加密。格式为binary，data部分存放图片的二进制流

- email 用户email，建议不加密。格式固定为text/plain
- phone 用户手机号码，建议将数据加密。格式固定为text/plain
- bio 用户个人简介，建议不加密。格式固定为text/plain

结构图如下：



以上五个节点内容均应用可根据需要自行设置。节点不创建或节点内容为空均是可以接受的。但建议一般应用新建MetaID用户时，最少要设置name节点，以确定用户名。

以下为构建“name”节点的一个例子，我们想要将用户的名字设为"Alice"。只需构建MetaID交易，将其父节点指向Info节点，将node_name设为"name", data设为“Alice”，如下表示：

```
OP_0 OP_RETURN meta <P(node)> <TxID(Info)> MetaID name Alice 0 1 NULL NULL
```

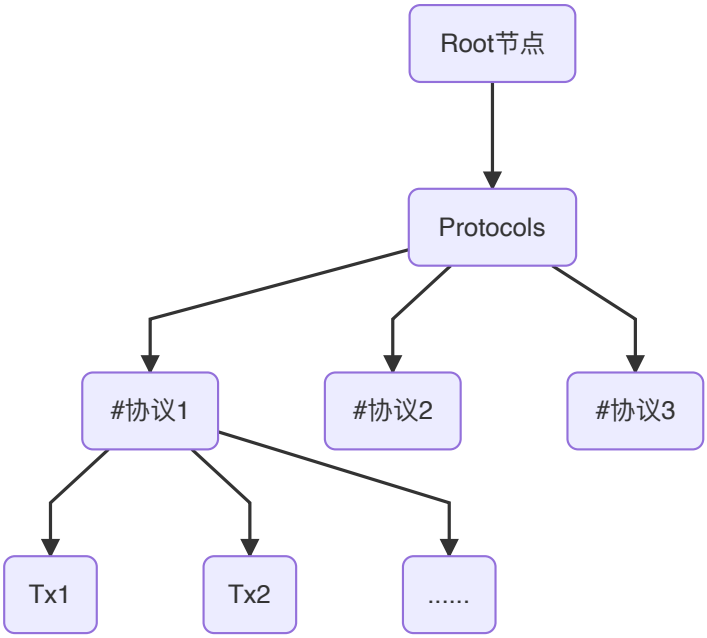
avatar节点是存储用户头像信息，数据格式固定为binary。构建可参考如下：

```
OP_0 OP_RETURN meta <P(node)> <TxID(Info)> MetaID avatar <IMAGE BUFFER> 0 1
image/png binary
```

其他节点均采用类似方式构建即可。需注意的是，如果某些信息比如phone节点信息希望加密，只需将encrypt设置为1，然后采用ECIES方式，用公钥对相关字符加密即可，这样加密的信息只有用户自己用对应私钥即可解密。

3.3 Protocols节点

Protocols节点为记录用户使用各种第三方协议的交易情况。Protocols节点下的子节点为第三方协议节点，其node_name应为协议名称。协议节点下的子节点为用户使用该协议所产生的具体交易。结构如下：



由于Protocols的协议是开放的，所有应用方都可以构建自己的协议。故此Protocols下的子节点数是无限制的，每个协议下的结构由协议制定方/应用方自己决定。但需保证协议节点的标识具有唯一性。

3.3.1 协议节点

关于协议名字的约定

构建一个新协议节点时，node_name为协议名称。协议名称没有限制，应用方可根据自己需要构成一个方便阅读和理解的协议名称。但为解决命名冲突的问题，需data部分则需为一个12字节长的哈希值，称为HashID。由于没有一个中心化机构来审核协议名称是否重复，所以data名分保留协议对应的HashID是有必要的。data部分的哈希值生成方式此版本协议采用BRFC ID[3]生成协议，最后生成的结果需是一个12字节长度哈希值。关于BRFC ID如何生成，请参考BRFC ID文档。

即Protocols下的第三方协议节点唯一标识方案为：

```
节点名+HashID
```

例如，某一个协议的BRFCID为3065510ee0dd，构建参考如下：

```
OP_0 OP_RETURN meta <P(node)> <Txid(Protocols)> MetaID SampleProtocol
3065510ee0dd 0 1 text/plain UTF-8
```

在实际数据中处理重名问题时，采用node_name+data即可构成唯一的节点识别ID，从而解决可能的协议重名问题。如上例子中，当需要唯一判断时，比如用URI定位时，可采node_name+data，即SampleProtocol3065510ee0dd来定位该节点。

关于协议的结构和约定

每个协议的结构由协议制定方/应用方约定，每个协议下可以是扁平的一层结构，也可以是复杂的多层结构。如果业务允许的话，该协议的的结构和详细说明，协议制定方应公开出来，以便其他应用方/数据服务方可以调用和解析。

需注意的是，这些节点的结构虽然为协议制定方/应用方决定，但相关的节点创建还是由用户创建，为用户所掌握，用户只记录和自己相关的协议数据。

3.3.2 协议交易节点

每个协议下的子节点均为用户使用该协议下的产生的具体交易，每条具体的交易称为协议交易节点。

data 为协议数据的存储处。协议数据由应用方根据约定的协议自行解析。

data_type 为数据类型，encoding为编码方式。应用方需根据data_type和encoding来读取data数据。

协议交易节点的node_name需为域树中同一层级内是唯一的，以方便以后用URI方式查找。例如可以采用public key作为node_name，也可以自行设定，只需确定与public key映射关系即可。

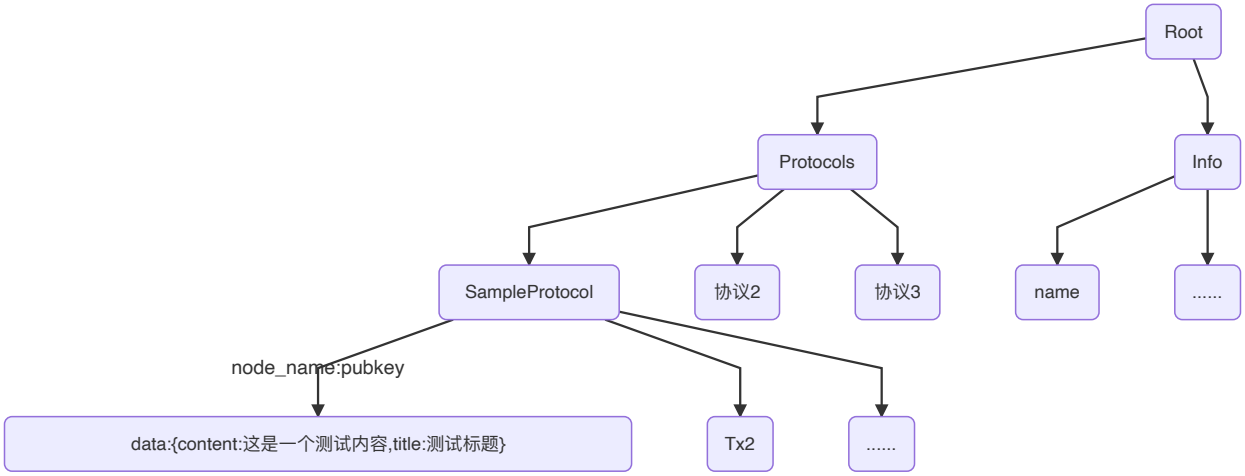
假设某一MetaID交易是属于SampleProtocol协议的，其数据格式采用Json，数据内容如下：

```
{"content":"这是一个测试内容","title":"测试标题"}
```

那么，该metaID交易的OP_RETURN构建参考如下：

```
OP_0 OP_RETURN meta <P(node)> <Txid(SampleProtocol)> MetaID <node_name>
{"content":"这是一个测试内容","title":"测试标题"} 0 1 applicaiton/json UTF-8
```

需注意的是，协议交易节点中的version值代表了其所遵循的协议版本，不同版本号代表其data内容有可能不一样。数据解析时，需根据不同的version号做不同的解析。上面例子中，生成的Metanet树结构参考如下：



4. 总结

用以上方法构建的MetaID，可让用户和应用方得到下的便利：

- 造构了一个由真正由用户掌握的Metanet树，该Metanet树记录了用户的基本信息和使用过的协议的所有交易记录。
- 用户只需保管一个主私钥（或助记词），即可导入不同的钱包，钱包方可还原用户链上的所有应用

交易记录。用户链上信息做到和钱包方无关。

- 应用方只需扫描其链上所有用户关于其协议的交易，记录下相关合格交易，并聚合整理即可呈现整个应用数据。
- 多个应用可使用同一个协议，一个应用也可以聚合多个协议，应用的开发更为灵活，开发工作也大为减少。

5. 参考文档

编号	参考
[1]	Metanet白皮书 https://nchain.com/app/uploads/2019/06/The-Metanet-Technical-Summary-v1.0.pdf
[2]	BIP32https://github.com/moneybutton/bips/blob/master/bip-0032.mediawiki
[3]	BRFC ID 分配协议 http://bsvalias.org/01-02-brfc-id-assignment.html