Summary

URI scheme for Tezos payment requests.

Abstract

This document describes an interface for Tezos payment requests. Tezos-based applications and delegation services should use this format to communicate with wallets.

Motivation

Smart contract applications need to tackle certain UX challenges to be as easy to use as non-blockchain applications. For example, contracts often require that users interacting with them send transactions with very specific parameters. The format of these parameters is hard to understand for users. This can be remedied by using a standardized payment request format, implemented by wallet software and applications. The user can keep using their favorite wallet that they are familiar with, while the application developer can prepare a request that the wallet will understand.

Without a standard way of communicating with wallets, applications have to ask users to copy-paste the parameters and other transaction details, which results in a bad user experience.

Specification

The payment request consists of a payload, and a URI scheme name. The payload is an array of JSON objects, containing request content and extension data. The payload is encoded with Base58Check encoding and prefixed with the URI scheme name to specify which Tezos network it's related to.

Payload

The JSON objects from Tezos RPC (called `$operation.alpha.contents`) are used as a starting point. Some fields are ommited, as they will be specified by the wallet while it's processing the operation. These are `counter` and `source`, and also `manager_pubkey` for originations. Our goal is to make it easy for a wallet that receives the data to add the missing fields and use the completed content object to interact with the RPC (or forge a transaction locally).

Individual content objects requesting a transaction, origination or delegation are wrapped in an object that lets us add extensions to the standard. Finally, the result is wrapped in an array. This allows callers to request multiple operations at once. Details about the wrapper structure are at the end of this section.

Content Objects

Transaction request fields are shown in the following table. Fields that are required in the request are in bold, while the fields that are optional are in italic.

RPC field	Treatment	Request field
kind	required	**kind**
amount	required	**amount**

RPC field	Treatment	Request field
destination	required	**destination**
source	chosen by wallet	
counter	chosen by wallet	
fee	optional	*fee*
gas_limit	optional	*gas_limit*
storage_limit	optional	*storage_limit*
parameters	optional	*parameters*

Origination request is very similar, the `manager_pubkey` will be chosen by the wallet.

RPC field	Treatment	Request field
kind	required	**kind**
balance	required	**balance**
source	chosen by wallet	
managerPubkey	chosen by wallet	
counter	chosen by wallet	
fee	optional	*fee*
gas_limit	optional	*gas_limit*
storage_limit	optional	*storage_limit*
spendable	optional	*spendable*
delegatable	optional	*delegatable*
delegate	optional	*delegate*
script	optional	*script*

Delegation request can be used to change the delegate of an account.

RPC field	Treatment	Request field
kind	required	**kind**
delegate	required	**delegate**
source	chosen by wallet	
counter	chosen by wallet	
fee	optional	*fee*
gas_limit	optional	*gas_limit*

RPC field	Treatment	Request field
storage_limit	optional	*storage_limit*

Request Objects vs RPC Objects

Here we compare the RPC objects with the objects understood by the Tezos node RPC. The RPC schemas can be seen here. We are using the JSON Schema format. Question mark denotes an optional field and dollar is a reference to another schema.

Schemas for different operations are compared below. Examples are provided in the Examples section.

Transaction Schema

The wallet has to fill in gas and storage limits and the fee if they are not specified.

RPC schema

```
{
    "kind": "transaction",
    "amount": $mutez,
    "destination": $contract_id,
    "source": $contract_id,
    "counter": $positive_bignum,
    "fee": $mutez,
    "gas_limit": $positive_bignum,
    "storage_limit":
$positive_bignum,
    "parameters"?:
$micheline.michelson_v1.expression
}
```

Request schema

```
{
    "kind": "transaction",
    "amount": $mutez,
    "destination": $contract_id,
    "fee"?: $mutez,
    "gas_limit"?:
$positive_bignum,
    "storage_limit"?:
$positive_bignum,
    "parameters"?:
$micheline.michelson_v1.expression
}
```

Origination Schema

This is similar to the transaction schema, except there are a few more optional fields in the original RPC schema.

RPC schema

```
{
    "kind": "origination",
    "balance": $mutez,
    "source": $contract_id,
    "managerPubkey":
$Signature.Public_key_hash,
    "counter": $positive_bignum,
    "fee": $mutez,
```

Request schema

```
{
    "kind": "origination",
    "balance": $mutez,
    "fee"?: $mutez,
    "gas_limit"?:
$positive_bignum,
    "storage_limit"?:
$positive_bignum,
```

```
    "gas_limit": $positive_bignum,
    "storage_limit":
$positive_bignum,
    "spendable"?: boolean,
    "delegatable"?: boolean,
    "delegate"?:
$Signature.Public_key_hash,
    "script"?: $scripted.contracts
}
```

```
    "spendable"?: boolean,
    "delegatable"?: boolean,
    "delegate"?:
$Signature.Public_key_hash,
    "script"?:
$scripted.contracts
}
```

Delegation Schema

In the delegation schema the `delegate` field is changed from optional to required. A delegation with no delegate specified is used to stop delegating without a replacement delegate, but this doesn't need a request, since there is no information to convey.

RPC schema

```
{
    "kind": "delegation",
    "source": $contract_id,
    "counter": $positive_bignum,
    "fee": $mutez,
    "gas_limit":
$positive_bignum,
    "storage_limit":
$positive_bignum,
    "delegate"?:
$Signature.Public_key_hash
}
```

Request schema

```
{
    "kind": "delegation",
    "delegate":
$Signature.Public_key_hash,
    "fee"?: $mutez,
    "gas_limit"?:
$positive_bignum,
    "storage_limit"?:
$positive_bignum
}
```

Wrapping the Payload

Individual operation objects are wrapped in an object that makes it easier to extend the standard in the future:

```
{
    "content": {
        "kind": "transaction",
        "amount": "123000",
        "destination": "tz1Ph8mdwaRp71XvixaExcNKtPvQshe5BwcR",
    }
}
```

Finally the result is added to an array:

```
[
    {
        "content": {
            "kind": "transaction",
            "amount": "123000",
            "destination": "tz1Ph8mdwaRp71XvixaExcNKtPvQshe5BwcR",
        }
    }
]
```

URI Scheme Name

Web wallets should be able to register as handlers for the URI scheme, so our URI scheme name has to start with `web+`. Firefox only allows `a-z` after `web+` so we concatenate `tezos` with the name of the network. For mainnet, we just use `tezos`.

Network	URI Scheme Name
mainnet	`web+tezos`
alphanet	`web+tezosalphanet`
zeronet	`web+tezoszeronet`

After encoding the payload with Base58Check, we prepend the URI scheme name and get a result that might look like this:

```
web+tezosalphanet:6kd7rBjLdHpk2j2G31rEFBidiMCp5qdnVarFhRiNeZ9XLNvGf6L22Fi8wEvwgzGzS
o17ns3GumtGM9FQKiNDwkUFXhditdCN6FVb2ukceZUkaJYpMQYV8ttvreHFg9475bfSSWe7528qcRaWZX7u
W9WbCzo6rh8nWw8EGypYWqrxA3DZAdVJCrSBPGnaD1E92yX9HSQktp3a3NnX6G1kH8xcCUwfwjJ89Ww2Qev
DGM2UgrkzvkgndaT1rTg7qnLsNSHJwZMNXBDxZEhtt2FHqSNERZ52BFiYcDfnVUqcMYyby4Y687KJzu5kYr
gdTYWY6w8EWVgnswHbsfLBu1wWRQj5UqCoLng6EupDGavWFM54zwu9a27EYFUKH5DumTCRFLHP2bxBSHHoo
mg9P471RVh8kKmf7iaEeQ6KBmz39QeGdWsCveSJzZkyeSYUCr3hzkDqedd3cYkLRvhtVt8reim3fVqpuYT7
mPQbnCtLb7QuxFkFVZzYEKBFd9x96jghQL7yptoQjyrCGfLMfckc8XSDV9bFuo3VT28Y7hRoMSaEYSDvShJ
cjSXr7xfWYffM8erT35iGPAJ193NNNVUYFGryH7tCUC3rzVF2wFwzd9adQbghxjk3SCpQrGbuRZ5wthm3ip
CgjRY1BAEvUaDaKmv2vPeS8LpQNzojQEetexQzhpgH5FnLhpHbxyZdJ6oCuXagn2PQwXHqYfQ7KbJizdGDC
dkKXjrrKcsVRURRBpnJy1oRgSeWo7P1BMy22mtUYR9vniH9oy73BdJi5u76aPV4xGZ7ww1QsCQo8z4T2HVC
mYcXYzgyfPNTQrpy8mV2esVrkCVEa4U
```

Overview

We can sum up request creation with these steps:

1. Application creates a JSON object with fields of interest.
2. The JSON object is wrapped and extension data is added if needed.
3. The result is added to an array, if there are more operations to be processed at once.
4. The array is encoded with Base58Check.
5. Encoded string is prefixed, depending on the relevant Tezos network.

The result can be presented as a clickable link. Wallet applications can register as handlers for the Tezos schemes and different wallets can be handlers for different networks. For example a production ready wallet can handle mainnet requests while a more developer focused wallet with diagnostic functionality can handle testnet requests.

Decoding usually involves the following:

1. Wallet receives the complete request.
2. Prefix is stripped and used to detect the network.
3. Payload is decoded and data validity is checked for each operation.
4. Wallet pre-fills a dialog.

Extensions

Some use cases would benefit from being able to transmit more information in the payment requests. This can be done by extending the standard. The extensions should be described by a separate standard and use a field based on the name of the standard to store their data. This will help avoid collisions with other extensions.

For example, an extension described in TZIP-009 that adds an information field could store its data like this:

```
[
    {
        "content": {
            "kind": "transaction",
            "amount": "123000",
            "destination": "tz1Ph8mdwaRp71XvixaExcNKtPvQshe5BwcR"
        },
        "extensions": {
            "tip9": {
                "info": "1 Blockaccino, 1 Scala Chip Frappuccino"
            }
        }
    }
]
```

Implementations that don't support the extension should ignore the unknown extension object.

Rationale

Choice of Encoding Algorithm

Base58Check is used heavily in the Tezos core code and Tezos wallets. By using Base58Check to encode the payload, implementers can avoid introducing an extra dependency to wallets and applications. Fewer depencies lead to fewer bugs and security problems.

Base58Check produces shorter output compared to hex encoding. Base64 offers negligible gains and would require an extra dependecy.

Protocol Scheme

Chrome only supports custom schemes beginning with `web+`. Firefox doesn't allow anything but lowercase ASCII letters after `web+`. Therefore we use `web+tezos` concatenated with `alphanet` or `zeronet`. For mainnet we just use `web+tezos`, so that testnet schemes look longer and can be recognized at glance. A different standard could conceivably use the same scheme in the future. This would result in a conflict, but we decided that it's a good trade-off to keep the scheme short.

Payload Format

The payload format closely mirrors the format expected by Tezos RPC, so that it can be used with minimal modifications. Using a more abstract format might better support some use cases. However it would require all wallets that implement the standard to maintain code that compiles the abstract format to the RPC format.

To make the standard as simple to implement as possible, we have decided against a more abstract payload format.

Misc. Information Field

The requests only include a subset of fields from the RPC format. To keep the standard simple, there are no extra fields such as a miscellaneous information field to describe a transaction. This could then be displayed by users' wallet, but wouldn't be sent anywhere. Extensions to the standard are possible, and a misc. information field is described by TZIP-009.

Implementation

JavaScript implementation is provided as an npm package called tezos-uri. The code can be viewed at gitlab. This section contains advice and ideas related to implementing the standard and integrating it with wallets.

Libraries

Export Minimal API

It's a good idea to make the implementation very simple to use for the common use cases. Our reference JS implementation accomplishes this by only exporting encoding and decoding functions. Internal functionality is obscured, since it is not intended to be used by the users of the package. There are different functions for handling links related to different networks, to minimize the chance that requests intended for different networks get mixed up.

Ensure Validity of Data

Implementers are advised to verify during encoding and decoding that the contents of all fields are compliant with the request schemes. This makes it more likely that errors will be caught by wallet and application developers during development and don't cause bigger problems later on.

When processing an array of requests, if one of the requests is malformed or contains invalid data, all of the requests should be treated as invalid.

Validate Michelson Data

Don't forget to validate Michelson data in `parameters` and `script` fields. The types are described in the Tezos RPC index. The script can be legal according to the RPC specification and still fail during interpretation. This

happens e.g. when an instruction has too few or too many arguments. Validating the Michelson data won't ensure that the script can be interpreted, but it will help catch mistakes sooner.

Wallets

Integrating the standard in a wallet enables UX improvements, such as pre-filling transaction dialogs for users so that they just need to confirm. This is important for application UX, as users can't be expected to copy-paste parameters or gas and storage limits.

Choosing the Source Address

All requests require wallet to add the `source` field. If the user has more than one address, the wallet can let the user select the appropriate one.

Gas and Storage

If a request doesn't specify gas and storage limits, the wallet is responsible for choosing these values. It could use a simple heuristic, but it could also simulate the transaction and set the appropriate limit or suggest it to the user.

Batch Requests

When handling an array of requests, the wallet should treat them as a batch and process them together. In case one of the requested operations fails during simulation, either all of the operations should be canceled or the user should be asked whether they want to drop the failing request and continue with the rest.

Visual Cues

It's a good idea to use visual cues to differentiate testnet operations from mainnet ones, such as with an alternate color theme when asking the user to confirm a testnet operation. Each network has its own scheme, so we can handle each in its own code path and easily adjust colors or similar configuration.

Applications

When preparing the requests, application developers should make sure to set the gas and storage limits high enough so that the operations don't run out of gas or storage space. Even though wallets can simulate the transaction to find out how much gas and storage is required, this isn't always accurate.

In the Wild

Experimental implementation of the standard can be seen at insurance.smartcontractlabs.ee & in the SCL Wallet at tw.smartcontractlabs.ee.

Examples

Transaction

Simple transaction with parameters:

```
[
    {
        "content": {
            "kind": "transaction",
            "amount": "123000",
            "destination": "tz1Ph8mdwaRp71XvixaExcNKtPvQshe5BwcR",
            "parameters": {
                "prim": "Left",
                "args": [
                    {
                        "prim": "Unit",
                        "args": []
                    }
                ]
            }
        }
    }
]
```

Simple transaction with extension containing an info field:

```
[
    {
        "content": {
            "kind": "transaction",
            "amount": "123000",
            "destination": "tz1Ph8mdwaRp71XvixaExcNKtPvQshe5BwcR",
            "parameters": {
                "prim": "Left",
                "args": [
                    {
                        "prim": "Unit",
                        "args": []
                    }
                ]
            }
        },
        "extensions": {
            "tip9": {
                "info": "1 Blockaccino, 1 Scala Chip Frappuccino"
            }
        }
    }
]
```

Request for two transactions that should be processed together:

```
[
    {
```

```
        "content": {
            "kind": "transaction",
            "amount": "123000",
            "destination": "tz1Ph8mdwaRp71XvixaExcNKtPvQshe5BwcR"
        }
    },
    {
        "content": {
            "kind": "transaction",
            "amount": "124000",
            "destination": "tz1Ph8mdwaRp71XvixaExcNKtPvQshe5BwcR"
        }
    }
]
```

Origination

Example for origination:

```
[
    {
        "content": {
            "kind": "origination",
            "balance": "123000",
            "script": {
                "code": [
                    {
                        "prim": "parameter",
                        "args": [
                            {
                                "prim": "unit"
                            }
                        ]
                    },
                    {
                        "prim": "storage",
                        "args": [
                            {
                                "prim": "unit"
                            }
                        ]
                    },
                    {
                        "prim": "code",
                        "args": [
                            [
                                {
                                    "prim": "CDR"
                                },
                                {
                                    "prim": "NIL",
                                    "args": [
```

```
                                        {
                                            "prim": "operation"
                                        }
                                    ]
                                },
                                {
                                    "prim": "PAIR"
                                }
                            ]
                        ]
                    }
                ],
                "storage": {
                    "prim": "Unit"
                }
            },
            "spendable": false,
            "delegatable": false,
            "delegate": "tz1Ph8mdwaRp71XvixaExcNKtPvQshe5BwcR"
        }
    }
]
```

Delegation

Minimal delegation request.

```
[
    {
        "content": {
            "kind": "delegation",
            "delegate": "tz1X7fu4GXBXp9A8fchu1px3zzMDKtagDBk3"
        }
    }
]
```

Delegation request with fee and limits.

```
[
    {
        "content": {
            "kind": "delegation",
            "fee": "1500",
            "gas_limit": "10100",
            "storage_limit": "0",
            "delegate": "tz1X7fu4GXBXp9A8fchu1px3zzMDKtagDBk3"
        }
    }
]
```

Appendix

Wallets Implementing the Standard

- SCL Wallet (older version of the standard)

Applications Using the Standard

- SCL Insurance (older version of the standard)

Delegation Services Using the Standard

- TBA

Copyright