BUIP086 – bitcoinash: URI format update

Layer: Applications  
Title: BUIP086 – bitcoincash: URI format update  
Author: Brendan Lee <brendanjlee@protonmail.com>  
Comments-Summary: No comments yet.  
Comments-URI:  
Status: Draft  
Type: Standards Track  
Created: 2018-02-05

Contents

[Abstract 1](#_Toc508177338)

[Motivation 1](#_Toc508177339)

[Parameters 3](#_Toc508177340)

[Compatibility 3](#_Toc508177341)

[Capabilities 4](#_Toc508177342)

[Requirements for URL reader 4](#_Toc508177343)

[Multiscan contracts 4](#_Toc508177344)

[Bitcoinscript 5](#_Toc508177345)

[Appendix 6](#_Toc508177346)

[Examples 6](#_Toc508177347)

[References 11](#_Toc508177348)

Abstract  
  
This BUIP describes a revision of the bitcoin: URI scheme (BIP 20/BIP21) to enable the creation of Bitcoin transactions with multiple and complex outputs including scripts and data.

New features proposed to be added in version 2.0 include:

Creating a new URI format allowing multiple outputs to be defined in a single URI

Adding functionality to allow multiple URLs to be scanned together and condensed into a single transaction

Create mechanisms that allow scripts and smart contracts to be applied within the standard URI formatting guidelines

# Motivation

This revision allows merchants to create more flexible transaction URIs that can split single payments into multiple outputs. For example, a merchant may have an agreement to pay some part of all gross revenue to a landlord, some part to a franchise organisation and some part as GST/VAT/Sales tax. This BUIP allows these additional transactions to be handled instantaneously as part of the merchant’s regular commercial operations, thereby simplifying their accounting and reporting processes. It also enables the third parties to receive their payments instantaneously rather than waiting until monthly or quarterly accounting processes can be undertaken.  
This revision also allows users to condense multiple URIs into a single set of outputs by scanning them consecutively before executing the transaction. For example, a merchant could attach QR codes to each item in the store directing the wallet to pay different amounts to a single common address. The new ‘multiscanparam’ allows a compatible SPV wallet to enter multiple URLs together to create a smart contract. Smart contracts can be used to perform token passing using coloured coins, or through OP\_RETURN outputs.

The addition of the ability to create unique scripts for each output in the URI gives merchants and users a high degree of flexibility and allows new application layers to be built and executed using methods already widely used for bitcoin transactions.  
Specification  
The bitcoincash: URI scheme is updated to version 2.0. This new version allows multiple new features including:

Nesting of multiple transaction outputs within a single URI

Addition of optional transaction outputs

Processing of multiple URIs into a single transaction

Specification of unique redeem scripts for each output defined in the transaction

The URI scheme maintains compatibility with version 1.0 through the use of compatible labels, but re-structures the URI outputs into a set of transaction data arrays which contain information relating to each transaction.  
Separate arrays exist for each transaction parameter (bitcoinaddress, amountparam, messageparam, labelparam), with the elements of each array being linked to the corresponding transaction (e.g. amount[1], label[1] and message[1] would all relate to transaction[1]).  
Additional parameters are added to the scheme including:

multiscanparam – Boolean value allowing a merchant to allow payers to merge multiple URIs into a single transaction

bitcoinscript[ ] – An array containing script data for each output

txidparam[ ] and voutparam[ ] – arrays used by scripts to identify specific UTXO that the script is trying to spend

Note:

To maintain compatibility with URI standards, all reserved characters must be transmitted as percent encoded values. E.g. %5B = [ and %5D = ].

For further information on reserved characters, see the following link: <https://tools.ietf.org/html/rfc3986#section-2.2>

For the purposes of readability, this specification uses the reserved symbols, however all examples have been represented using RFC3986 compliant notation.

## Parameters

The URI and its parameters are defined as follows:  
  
Bitcoincashurn = "bitcoincash:" bitcoinaddress [ ";version=" bitcoinversion ] [ “?” bitcoinparam ] [ "?" bitcoinparam[] ] [ “?” otherparam ]  
bitcoinaddress = cashaddress | legacyaddress

cashaddress = cashadd \*cashaddr

legacyaddress = base58 \*base58  
bitcoinversion = "2.0"  
bitcoinparam = amountparam | labelparam | messageparam | sendparam | multiscanorderparam | otherparam  
amountparam = “amount=” \*amount  
amount = amountdecimal | amounthex  
amountdecimal = \*digit [ "." \*digit ] [ "X" \*digit ]  
amounthex = "x" \*hexdigit [ "." \*hexdigit ] [ "X" \*hexdigit ]  
labelparam = “label=” \*pchar  
messageparam = “message” \*pchar  
sendparam = "send=" \*pchar   
multiscanorderparam = “order[?]=” \*char  
otherparam = pchar \*pchar "=" \*pchar  
bitcoinparam[] = bitcoinaddress[] | amountparam[] | labelparam[] | messageparam[] | bitcoinscrip[]  
bitcoinaddress[] = “address[?]=” cashaddress | legacyaddress

cashaddress = cashaddr \*cashaddr

legacyaddress = base58 \*base58  
bitcoinaddress[0] = bitcoinaddresss  
amountparam[] = "amount[?]=" amount  
amountparam[?] = amountdecimal | amounthex  
amountdecimal = \*digit [ "." \*digit ] [ "X" \*digit ]  
amounthex = "x" \*hexdigit [ "." \*hexdigit ] [ "X" \*hexdigit ]  
amountparam[0] = amountparam  
labelparam[] = "label[?]=" \*pchar  
labelparam[0] = labelparam  
messageparam[] = "message[?]=" \*pchar  
messageparam[0] = messageparam

txscript[] = “txscript[?]=’ \*pchar  
bitcoinscript[] = "script[?]=" \*pchar

bitcoinscript[0] = bitcoinscript  
bitcoinscript[] = "script[?]=" \*pchar

bitcoinscript[0] = bitcoinscript

txidparam[] = “txid[?]=” \*base64

voutparam[] = “vout[?]=” \*digit

### Compressed Labels

Due to the size constraints of some URL presentation methods the following compressed labels will also be considered valid for transaction construction:

“v” = “version”

“ad” = “address”  
“am” = “amount”  
“l” = “label”  
“m” = “message”  
“o” = “order”  
“ts” = “txscript”

“bs” = “bitcoinscript”

“tx” = “txid”

“v” = “vout”

## Compatibility

The new URI format maintains compatibility with the current version 1.0 format through the inclusion of the bitcoinparam labels used in version1.0. This means that POS systems that do not need features associated with version2.0 URLs do not need to upgrade to be compatible with wallets that support version2.0

## Capabilities

The version 2.0 format allows merchants to create structured URIs that can send multiple complex outputs where each output can contain a non-default redeem script and be sent to a unique destination. This is achieved by using the bitcoinparam[ ] array structures.  
  
The minimum amount of information needed by a wallet to add an output to a transaction is a bitcoin address. For each output being requested, the URI must add a bitcoin address to a position in the bitcoinaddress[ ] array. The position of the address in the bitcoinaddress[ ] array determines the position of the ancillary data in the amountparam[ ], labelparam[ ], messageparam[ ], scriptparam[ ] and txid[ ] arrays. E.g. amount data for the output to the address located in bitcoinaddress[1], is entered into the amountparam[1], messageparam[1] and labelparam[1] array positions.  
  
A URI that defines more than one element in any bitcoinparam[N] field would be rejected as invalid. E.g. if a URI defines an amount in the amount field and then defines a conflicting amount in the amount[0] array position, the URI would be rejected as invalid. Similarly, if there are two separate values attached to the amount[2] label, the URI would be rejected as invalid.

For tokens and sidechains, a script can identify a particular UTXO which is to be spent using the txidparam[ ] and voutparam[ ] arrays. This can be used to specify a UTXO which is known to contain a coloured coins or a certain script. For the transaction to be valid, the selected UTXO

## Requirements for URL reader

The URI provides a medium through which complex transaction structures can be defined between two parties. For a transaction to be completed, the seller must meet all the requirements stipulated in the URL.

Where amount[N] is defined, the wallet can assume that the requested value is agreed upon between sender and receiver. It is up to the user’s wallet to determine how best to structure that payment using the funds has access to.

Where script[N] is the wallet can assume that any outputs sent to that address that do not contain the correct redeem script will not meet the requirements of the contract and will not complete the transaction. It is up to the sending wallet to ensure that amounts and scripting requirements imposed by the receiver are met.

### Multiscan contracts

The intent of the Multiscan function is to allow for the creation of a new type of smart contract. A customer assembles a contract by using their SPV wallet to scans URLs that include a contract order.

A multiscan contract is started when a wallet receives a URL with a non-zero value in the multiscanorderparam field. The wallet sees this and goes into multiscan mode.

In multiscan mode, the wallet should allow the user to enter multiple different URLs. URLs do not need to be scanned in order however the wallet user will not be able to terminate the contract without a URL labelled order[FF]. The order[FF] URL may contain financial transactions and parameters. If OP\_GROUP is used in this Script[0], any URLs that are part of the contract will be able to use coloured coins as well.

When the wallet assembles the transaction, the transactions in addres[X>0] of the order[FF] URL, ascending transaction numbers from zero will be set up to be the first when the contract is executed. Scripts in the remaining URLs will be appended in ascending order. Once all the URLs have been concatenated, the transactions with address[X<0] as a parameter in the order[FF] URL are appended in ascending order.

Any URLs that have the same multiscanorderparam must be concatentated in no particular order. If a zero value transaction

Where there are multiple transactions directed to a particular address in the contract, from multiple URLs with the same multiscanorderparam, those amounts can be aggregated. Where they are contained in multiple levels of the contract, the amount at each level can either be paid in a separate output or added together. – NEED FEEDBACK ON THIS IDEA

When additional URLs are scanned, the wallet will expect multiscanparam to be non-zero. If a URL without a non-zero multiscanparam is scanned, it is either rejected from the contract, or the contract is terminated.

**Example: Alice visits Bobmart to buy a few things.**

When alice arrives, she can choose three baskets. Blue, green and red. Each has a NFC chip in the handle containing an order[FF] URL.

Alice picks the blue basket and scans the handle with her phone. Its zero script sets up the transaction to support colored coins using OP\_GROUP. Her basket will now reward her with 1 Bobpoint for every bit she spends. Her phone now goes into scan mode.

The green basket’s zero script sets up the transaction to use the Bob’s charities colored coin system. Any transactions would carry discounts that apply to Bobmart Rewards members, but instead of receiving Bob’s her BobPoints, Bobmart gives 1 satoshi to a local charity.

The red basket has no points or charity, and is a basic payment script.

Alice buys some fruit, (order[12]), milk(order[2]), cheese (order[2]) and bread (order[10]), scanning the code on each as she puts them in her basket. walks to the packing area and hits ‘Pay Now’.

Her wallet sets up a contract that applies bob’s rewards to be automatically applied to her wallet. The transaction then pays 500 bits to the dairy who made the milk and cheese, 250 to the Bob’s dairy transport and storage, 1200 bits to various orchards for the fruit and 1000 bits to bob’s fruit department and 500 bits to Bob’s bakery. The wallet can send this onto the network and Bob and all of his suppliers are paid at once.

### Bitcoinscript

The version2.0 URI format allows URLs to be created which fully or partially define complex smart contracts. By using the bitcoinscript[] parameters, a URL can be constructed that defines smart contract scripts to be applied to some or all of the outputs in the transaction.

This can be used to apply non-default redeem scrips such as timelocks, tokenisation scripts or evaluation of multiple signatures to outputs containing funds, or it can be used to define the contents of an OP\_RETURN output that needs to be included in the transaction.

It is up to the sending wallet to ensure that scripting requirements specified in the URL are met.

# Appendix

Version 1.0 Syntax

The version 2.0 implementation is still fully compatible with version 1.0 wallets. POS providers should ensure to always specify version=2.0 in situation where version 2.0 URLs are being presented. Otherwise a version 1.0 compatible wallet will look only at the parameters in the bitcoinparam fields and drop all other data.

# Examples

Example 1: Bar transaction

Satoshi’s Tavern employs roving servers who take orders from patrons and retrieve drinks from an automated dispensing robot. Servers use an app to assemble orders for patrons who then pay the server directly.

Bob orders two beers and a Margarita from Alice who enters the order in her app. The app creates the following URI and presents it as a QR code (Line breaks added for clarity):

bitcoincash:qr2se9wy2egettFxPbmn99cK8v5AFq55Lx?version=2.0

&amount=1X6

&label=Satoshis%20Tavern:%20Server%20–%20Alice

&message=Order%20-%202%20x%20Sierra%20Nevada%20Pale%20Ale%2c%201%20x%20Margarita

&address%5B1%5D=qnttFxPbmn2se9wy2egAFq55Lxe99cK8v5

&amount%5B1%5D=1X5

&message%5B1%5D=Sales%20Tax

&address%5B2%5D=qn2se9nttFxPbmxwy2egFcK8v5q55Lxe99?

&label%5B2%5D=Tip

&message%5B2%5D=Tip

&receipt=true

If Bob’s wallet is compatible with the extension, Bob scans the QR code with his hot wallet and is first presented with a prompt labelled ‘Tip’ and a numerical entry field where he can enter in an amount to tip Alice. Bob enters a tip of 0.002BCH and the wallet creates a transaction with outputs as follows:

0.01BCH to qr2se9wy2egettFxPbmn99cK8v5AFq55Lx

0.001BCH to qnttFxPbmn2se9wy2egAFq55Lxe99cK8v5

0.002BCH to qn2se9nttFxPbmxwy2egFcK8v5q55Lxe99

OP\_RETURN: ’Satoshi’s Tavern: Server – Alice; Order - 2 x Sierra Nevada Pale Ale, 1 x Margarita, 0.011BCH; Sales tax, 0.01BCH; Tip, 0.002BCH’

Change output

and the following memo:

“Label: Satoshi’s Tavern <carriage return>

2 x Sierra Nevada Pale Ale, 1 x Margarita 0.01BCH <carriage return>

Sales Tax 0.001BCH <carriage return>

Tip 0.002BCH”

The memo is stored locally in Bob’s wallet and may be uploaded to a wallet provider’s systems in-case Bob’s wallet needs to be restored on a new device, however it is not recorded on-chain and is not considered immutable or appropriate for the storage of important transaction records.

The OP\_RETURN output is recorded on-chain and represents an immutable record of the transaction.

When Bob sends the transaction, the bar recognises that it has received a payment and looks for an output with information encoded using an OP\_RETURN opcode. Using this information it automatically assembles the order and places it on a tray for Alice to pick up and deliver to Bob.

The government jurisdiction in which Satoshi’s tavern operates instantaneously receives a sales tax transaction of 0.001BCH, and Alice receives a tip payment of 0.002BCH directly to her personal wallet.

Bob’s wallet shows a transaction with 5 outputs and a memo detailing his purchase. Bob can also go on-chain and see the transaction information encoded in the OP\_RETURN output.

If Bob’s wallet is not compatible with the extension, his wallet recognises that bitcoinparam version=2.0 and will reject the transaction.

Alice’s app should incorporate the ability to provide a Version1.0 compatible transaction however this would be up to the POS app provider and is not part of this specification.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Example 2: Online Content

Bchncontent.com accepts online payments for content including fiction and non-fiction articles, music and video.

Bob puts the following in his cart:

The album ‘Hex on you’ by Alice On-Chains, priced at 0.00012BCH

An e-book ‘A technical guide to URIs’ by Donald Drumpf priced at 0.00002BCH

A web comic ‘Driving with Fiat’ by Bill Burnie which is released under creative commons. Any money given to Bill Burnie would be considered a donation and is therefore tax exempt.

Bob lives in a jurisdiction that applies a 10% sales tax on top of the price of the items.

When Bob clicks ‘Purchase’, the bchncontent.com cart creates the following URI and presents it as a QR code:

bitcoincash:qtFxPbmn99cK8r2se9wy2egetv5AFq55Lx?version=2.0

&amount=14X3

&label=BCHncontent.com

&message=Album%20–%20Hex%20on%20you%2c%20Alice%20On-Chains%2c%20 0.00012BCH;%20eBook%20–%20A%20technical%20guide%20to%20URIs%2c%20Donald Drumpf%2c%202X3BCH

&address%5B1%5D=qnttFxPbmn2se9wy2egAFq55Lxe99cK8v5

&amount%5B1%5D=14X2

&message%5B1%5D=Sales%20Tax

&address%5B2%5D=qse9wy2egAFq55Lxe99cK8nttFxPbmn2v5

&label%5B2%5D=Driving%20with%20Fiat%2c%20Bill%20Burnie

&message%5B2%5D=Creative%20Commons%20Comic%20–%20Driving%20with%20Fiat%2c%20Bill%20Burnie

&receipt=false

If Bob’s wallet is compatible with the extension, Bob scans the QR code with his hot wallet. First, Bob is presented with a prompt labelled ‘Driving with Fiat, Bill Burnie’ where he can enter in an amount that he wishes to donate to Bill Burnie. Bob enters an amount of 0.0002BCH. Next, Bob is presented with a prompt asking him if he wishes to create an invoice on-chain. Bob selects ‘Yes’.

Bob’s wallet creates a transaction with outputs as follows:

0.00014BCH to qtFxPbmn99cK8r2se9wy2egetv5AFq55Lx

0.000014BCH to qnttFxPbmn2se9wy2egAFq55Lxe99cK8v5

0.0002BCH to qse9wy2egAFq55Lxe99cK8nttFxPbmn2v5

OP\_RETURN: ‘BCHncontent.com: Album – Hex On You, Alice On-Chains, 0.00012BCH; eBook – A technical guide to URIs, Donald Drumpf, 0.00002BCH; Sales tax: 0.0014BCH; Creative Commons Comic – Driving with Fiat, Bill Burnie, 0.0002BCH donation’

Change output

and the following memo:

“Label: BCHncontent.com <carriage return>

Album – Hex on you, Alice On-Chains, 0.00012BCH; eBook – A technical guide to URIs, Donald Drumpf, 0.00002BCH <carriage return>

Sales Tax 0.000014BCH <carriage return>

Driving with Fiat, Bill Burnie, 0.0002BCH”

The memo is stored locally on his wallet, and may be uploaded to a wallet provider’s systems in-case Bob’s wallet needs to be restored on a new device, however it is not recorded on-chain and is not considered immutable or appropriate for the storage of important transaction records.

The OP\_RETURN output is recorded on-chain and represents an immutable record of the transaction.

When Bob sends the transaction, the bchncontent.com SPV wallet recognises that it has received Bob’s payment and sends Bob the instructions needed to access his content.

The government jurisdiction in which Bob lives instantaneously receives a sales tax transaction of 0.000014BCH, and Bill Burnie receives a donation of 0.0002BCH.

Bob’s wallet shows a transaction with 5 outputs and a memo detailing his purchase. Bob can also go on-chain and see the transaction information encoded in the OP\_RETURN output.

If Bob’s wallet is not compatible with version 2.0 transactions, it would see parameter version = 2.0 and would reject the transaction as invalid. For Bob to continue, the online merchant would need to provide a backwards transaction method for Bob to use.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Example 3: Market Stall

Nutmeg Greg runs a market stall. He sells nutmeg and nutmeg related goods.

Bob walks into Nutmeg Greg’s stall and opens up a Version2.0 compatible app on his phone.

Bob selects a jar of premium nutmeg off the shelf and scans the QR code on the label which returns the following (line breaks added for clarity only):

bitcoincash:qt9wy2egetv5AFxPbmn99cK8r2seFq55Lx?version=2.0

&amount=25X3

&label=Nutmeg%20Greg

&message=Premium%20Nutmeg%2c%200.00025BCH

&address%5B1%5D=qnttFxPbmn2se9wy2egAFq55Lxe99cK8v5

&amount%5B1%5D=25X2

&message%5B1%5D=Market%20Fee

&Multiscan=true

&receipt=true

Bob’s wallet starts creating a transaction with two outputs, one for 0.00025BCH to qt9wy2egetv5AFxPbmn99cK8r2seFq55Lx and another to qnttFxPbmn2se9wy2egAFq55Lxe99cK8v5 for 0.000025BCH.

The wallet then asks Bob if he would like to add more items to the transaction.

Bob selects ‘YES’.

Bob selects a bag of nutmeg candy and scans the QR code on the label which returns the following (line breaks added for clarity only):

bitcoincash:qt9wy2egetv5AFxPbmn99cK8r2seFq55Lx?version=2.0

&amount=3X4

&label=Nutmeg%20Greg

&message=Nutmeg%20Candy%2c%200.0003BCH

&address%5B1%5D=qnttFxPbmn2se9wy2egAFq55Lxe99cK8v5

&amount%5B1%5D=3X3

&message%5B1%5D=Market%20Fee

&Multiscan=true

&receipt=true

Bob’s wallet looks at the transaction it has already started creating and sees common output addresses for each of the new outputs. It updates the output to qt9wy2egetv5AFxPbmn99cK8r2seFq55Lx to 0.00055BCH and the one to qnttFxPbmn2se9wy2egAFq55Lxe99cK8v5 to 0.000055BCH.

The wallet then asks Bob if he would like to add more items to the transaction.

Bob selects ‘No’.

The wallet then asks Bob if he would like a receipt to be recorded on-chain.

Bob selects ‘No’.

Bob’s wallet finalises the transaction with outputs as follows:

0.00055BCH to qt9wy2egetv5AFxPbmn99cK8r2seFq55Lx

0.000055BCH to qnttFxPbmn2se9wy2egAFq55Lxe99cK8v5

OP\_RETURN: ‘Nutmeg Greg: Premium Nutmeg, 0.00025BCH; Nutmeg Greg: Nutmeg Candy, 0.0003BCH, Market Fee, 0.000055BCH’

Change output

and the following memo:

“Label: Nutmeg Greg <carriage return>

Premium Nutmeg, 0.00025BCH

Nutmeg Candy, 0.0003BCH

Market fee, 0.000055BCH

Change output

The memo is stored locally in Bob’s wallet, and may be uploaded to a wallet provider’s systems in-case Bob’s wallet needs to be restored on a new device, however it is not recorded on-chain and is not considered immutable or appropriate for the storage of important transaction records.

When Bob sends the transaction, Nutmeg Greg’s SPV wallet sees the transaction and displays the amount on Greg’s screen. Greg’s wallet also displays the contents of the OP\_RETURN output which he uses to confirm Bob’s purchase.

The market where Greg’s stall is located instantaneously receives a fee transaction of 0.000055BCH.

Bob’s wallet shows a transaction with 4 outputs and a memo detailing his purchase. Bob can also go on-chain and see the transaction information encoded in the OP\_RETURN output.

If Bob’s wallet is not compatible with version 2.0 transactions, it would see parameter version = 2.0 and would reject the transaction as invalid. For Bob to continue, Greg would need to provide separate labels with version 1.0 compatible URI’s.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ex

# References