API documentation

blockchair.com/api/docs

Infinitable endpoints (SQL-like queries)

These endpoints allow you to filter, sort, and aggregate blockchain data. The output is database rows. Unlike dashboard and raw endpoints, all infinitable endpoints listed in this section can be considered as just one endpoint as it has the same options and the same output structure across different blockchains and entities. Here it is: `https://api.blockchair.com/{:table}{:query}`.

Just don't ask why do we call that `infinitables`… Infinite tables? Maybe.

List of tables (`{:table}`) our engine supports:

- `{:btc_chain}/blocks`
- `{:btc_chain}/transactions`
- `{:btc_chain}/mempool/transactions`
- `{:btc_chain}/outputs`
- `{:btc_chain}/mempool/outputs`
- `{:btc_chain}/addresses`
- `{:eth_chain}/blocks`
- `{:eth_chain}/uncles`
- `{:eth_chain}/transactions`
- `{:eth_chain}/mempool/transactions`
- `{:eth_chain}/calls`
- `{:xin_chain}/raw/snapshots`
- `{:xin_chain}/raw/mintings`

- `{:xin_chain}/raw/nodes`
- `{:xtz_chain}/raw/blocks`
- `bitcoin/omni/properties`
- `ethereum/erc-20/tokens`
- `ethereum/erc-20/transactions`

Where:

- `{:btc_chain}` can be one of these: `bitcoin`, `bitcoin-cash`, `litecoin`, `bitcoin-sv`, `dogecoin`, `dash`, `groestlcoin`, `zcash`, `ecash`, or `bitcoin/testnet`
- `{:eth_chain}` can be only `ethereum`
- `{:xin_chain}` can be only `mixin`
- `{:xtz_chain}` can be only `tezos`

Note on mempool tables: to speed up some requests, our architecture have separate tables (`{:chain}/mempool/{:entity}`) for unconfirmed transactions. Unlike with dashboard endpoints which search entities like transactions in both the blockchain and the mempool, infinitable endpoints don't do that.

The `{:query}` is optional; in case it's not included in the request, the default sorting applied to the table (for most of the tables it's descending by some id) and the 10 top results are returned.

Here are some example queries without using `{:query}`:

- `https://api.blockchair.com/bitcoin/blocks`
- `https://api.blockchair.com/bitcoin-cash/mempool/transactions`

The output skeleton is the following:

```
{
  "data": [
    {
      ... // row 1 data
    },
    ...
    {
      ... // row 10 data
    },
  ],
  "context": {
    "limit": 10, // the default limit of 10 is applied
    "offset": 0, // no offset has been set
    "rows": 10, // the response contains 10 rows
    "total_rows": N, // but there are N rows in the table matching {:query} (total
number of rows if it's not set)
    "state": S, // the latest block number on the blockchain
    ...
  }
}
```

Further documentation sections describe fields returned for different tables. Some of the dashboard endpoints are using the same fields as well.

How to build a query

The process is somewhat similar to constructing an SQL query, but there are less possibilities of course.

Here are the possible options:

- Setting filters — the `?q=` section — allows you to set a number of filters (SQL "`WHERE`")
- Setting sortings — the `?s=` section — allows you to sort the table (SQL "`ORDER BY `")
- Setting the limit — the `?limit=` section — limits the number of output results (SQL "`LIMIT`")
- Setting the offset — the `?offset=` section — offsets the result set (SQL "`OFFSET`")
- Aggregating data — the `?a=` sections — allows to group by some columns and calculate using function (SQL "`GROUP BY`" and functions such as `count`, `max`, etc.)
- The table (SQL "`FROM`") is set in the `{:table}` section of the API request

The order of applying various sections is irrelevant.

A quick example: `https://api.blockchair.com/bitcoin/blocks?q=time(2019-01),guessed_miner(AntPool)&s=size(desc)&limit=1`. This request:

- Makes a query to the `bitcoin/blocks` table
- Filters the table by time (finds all blocks mined in January 2019) and miner (AntPool)
- Sorts the table by block size descending
- Limits the number of results to 1

What this example does is finding the largest block mined by AntPool in January 2019.

Another example using aggregation: `https://api.blockchair.com/bitcoin/blocks?q=time(2019-01-01..2019-01-31)&a=guessed_miner,count()&s=count()(desc)`. This request:

- As the previous one, makes a query to the `bitcoin/blocks` table
- Filters the table by time (in a bit different way, but it's an invariant of `time(2019-01)`)
- Groups the table by miner, and calculating the number of rows for each miner using the `count()` function
- Sorts the result set by the number of blocks each miner has found

The `?q=` section (filters)

You can use filters as follows: `?q=field(expression)[,field(expression)]...`, where `field` is the column which is going to be filtered, and `expression` is a filtering expression. These are possilble filtering expressions:

- `equals` — equality — example: `https://api.blockchair.com/bitcoin/blocks?q=id(0)` finds information about block 0
- `left..` — non-strict inequality — example: `https://api.blockchair.com/bitcoin/blocks?q=id(1..)` finds information about block 1 and above
- `left...` — strict inequality — example: `https://api.blockchair.com/bitcoin/blocks?q=id(1...)` finds information about block 2 and above
- `..right` — non-strict inequality — example: `https://api.blockchair.com/bitcoin/blocks?q=id(..1)` finds information about blocks 0 and 1
- `...right` — strict inequality — example: `https://api.blockchair.com/bitcoin/blocks?q=id(...1)` finds information only about block 0
- `left..right` — non-strict inequality — example: `https://api.blockchair.com/bitcoin/blocks?q=id(1..3)` finds information about blocks 1, 2 and 3
- `left...right` — strict inequality — example: `https://api.blockchair.com/bitcoin/blocks?q=id(1...3)` finds information about block 2 only
- `~like` — occurrence in a string (SQL `LIKE '%str%'` operator) — example: `https://api.blockchair.com/bitcoin/blocks?q=coinbase_data_bin(~hello)` finds all blocks which contain `hello` in `coinbase_data_bin`
- `^like` — occurrence at the beginning of a string (SQL `LIKE 'str%'` operator, also further mentioned as the `STARTS WITH` operator) — example: `https://api.blockchair.com/bitcoin/blocks?q=coinbase_data_hex(^00)` finds all blocks for which `coinbase_data_hex` begins with `00`

For timestamp type fields, values can be specified in the following formats:

- `YYYY-MM-DD HH:ii:ss`
- `YYYY-MM-DD` (converted to the `YYYY-MM-DD 00:00:00..YYYY-MM-DD 23:59:59` range)
- `YYYY-MM` (converted to the `YYYY-MM-01 00:00:00..YYYY-MM-31 23:59:59` range)

Inequalities are also supported for timestamps, the left and right values must be in the same format, e.g.: `https://api.blockchair.com/bitcoin/blocks?q=time(2009-01-03..2009-01-31)`.

Ordinarilly if there's `time` column in the table, there should also be `date`, but there won't be possible to search over the `date` column directly, but you can search by date using the `time` column as follows: `?q=time(YYYY-MM-DD)`

If the left value in an inequality is larger than the right, they switch places.

If you want to list several filters, you need to separate them using commas like this: `https://api.blockchair.com/bitcoin/blocks?q=id(500000..),coinbase_data_bin(~hello)`

We're currently testing support for `NOT` and `OR` operators (this is an alpha test feature, so we don't guarantee there won't be sudden changes):

- The `NOT` operator is added before the expression for it to be inverted, e.g., `https://api.blockchair.com/bitcoin/blocks?q=not,id(1..)` returns the block `0`
- The `OR` operator can be put between two expressions and takes precedence (like it's when two expressions around `OR` are wrapped in parentheses), e.g., `https://api.blockchair.com/bitcoin/blocks?q=id(1),or,id(2)` returns information about blocks 1 and 2.

Maximum guaranteed supported number of filters in one query: 5.

The `?s=` section (sortings)

Sorting can be used as follows: `?s=field(direction)[,field(direction)]...`, where `direction` can be either `asc` for sorting in ascending order, or `desc` for sorting in descending order.

Here's a basic example: `https://api.blockchair.com/bitcoin/blocks?s=id(asc)` — sorts blocks by id ascending

If you need to apply several sortings, you can list them separating with commas. The maximum guaranteed number of sortings is 2.

The `?limit=` section (limit)

Limit is used like this: `?limit=N`, where N is a natural number from 1 to 100. The default is 10. `context.limit` takes the value of the set limit. In some cases (when using some specific "increased efficiency" filters described below) `LIMIT` may be ignored, and in such cases the API returns the entire result set, and `context.limit` will be set to `NULL`.

A basic example: `https://api.blockchair.com/bitcoin/blocks?limit=1` — returns the latest block data (as the default sorting for this table is by block height descending)

Note that increasing the limit leads to an increase in the request cost (see the formula below).

The `?offset=` section (offset)

Offset can be used as a paginator, e.g., `?offset=10` returns the next 10 rows. `context.offset` takes the value of the set offset. The maximum value is 10000. If you need just the last page, it's easier and quicker to change the direction of the sorting to the opposite.

Important: while iterating through the results, it is quite likely that the number of rows in the database will increase because new blocks had found while you were paginating. To avoid that, you may, for example, add an additional condition that limits the block id to the value obtained in `context.state` in the first query.

Here's an example. Suppose we would like to receive all the latest transactions from the Bitcoin blockchain with amount more than $1M USD. The following request should be perfomed for this:

```
https://api.blockchair.com/bitcoin/transactions?
q=output_total_usd(10000000..)&s=id(desc)
```

Now, the script with this request to the API for some reason did not work for a while, or a huge amount of transactions worth more than $1 million appeared. With the standard limit of 10 results, the script skipped some transactions. Then firstly we should make the same request once again:

```
https://api.blockchair.com/bitcoin/transactions?
q=output_total_usd(10000000..)&s=id(desc)
```

From the response we put `context.state` in a variable `{:state}`, and further to obtain next results we apply `offset` and set a filter to "fix" the blockchain state:

```
https://api.blockchair.com/bitcoin/transactions?
q=output_total_usd(10000000..),block_id(..
{:state})&s=id(desc)&offset=10
```

Next we increase the offset value until getting a data set with the transaction that we already knew about.

The `?a=` section (data aggregation)

Warning: data aggregation is currently in beta stage on our platform.

To use aggregation, put the fields by which you'd like to group by (zero, one, or several), and fields (at least one) which you'd like to calculate using some aggregate function under the `?a=` section. You can also sort the results by one of the fields included in the `?a=` section (`asc` or `desc`) using the `?s=` section, and apply additional filters using the `?q=` section.

Let's start with some examples:

- `https://api.blockchair.com/bitcoin/blocks?a=year,count()` — get the total number of Bitcoin blocks by year
- `https://api.blockchair.com/bitcoin/transactions?a=month,median(fee_usd)` — get the median Bitcoin transaction fees by month

- `https://api.blockchair.com/ethereum/blocks?a=miner,sum(generation)&s=sum(generation)(desc)` — get the list of Ethereum miners (except uncle miners) and sort it by the total amount of coins minted
- `https://api.blockchair.com/bitcoin-cash/blocks?a=sum(fee_total_usd)&q=id(478559..)` — calculate how much miners have collected in fees since the fork

In case the table you're aggregating over has a `time` column, it's always possible to group by the following virtual columns:

- `date`
- `week` (yields `YYYY-MM-DD` corresponding to Mondays)
- `month` (yields `YYYY-MM`)
- `year` (yields `YYYY`)

Supported functions:

- `avg({:field})`
- `median({:field})`
- `min({:field})`
- `max({:field})`
- `sum({:field})`
- `count()`

There are also two special functions:

- `price({:ticker1}_{:ticker2})`— yields the price; works only if you group by `date` (or one of: `week`, `month`, `year`). For example, it makes it possible to build a chart showing correlation between price and transaction count: `https://api.blockchair.com/bitcoin/blocks?a=month,sum(transaction_count),price(btc_usd)`. Supported tickers: `usd`, `btc`, `bch`, `eth`, `ltc`, `bsv`, `doge`, `dash`, `grs`
- `f({:expression})` where `{:expression}` is `{:function_1}{:operator}{:function_2}`, where `{:function_1}` and `{:function_2}` are the supported functions from the above list, and `{:operator}` is one of the following: `+`, `-`, `/`, `*` (basic math operators). It's useful to calculate percentages. Example: `https://api.blockchair.com/bitcoin/blocks?a=date,f(sum(witness_count)/sum(transaction_count))&q=time(2017-08-24..)` — calculates SegWit adoption (by dividing the SegWit transaction count by the total transaction count)

There's also a special `?aq=` section which have the following format: `?aq={:i}:{:j}` — it applies `i`th filter to `j`th function (special functions don't count); after that `i`th filter has no effect on filtering the table. It's possible to have multiple conditions by separating them with a `;`. Here's an example: `https://api.blockchair.com/bitcoin/outputs?`

`a=date,f(count()/count())&q=type(nulldata),time(2019-01)&aq=0:0` — calculates the percentage of nulldata outputs in January 2019 by day. The 0th condition (`type(nulldata)`) is applied to the 0th function (`count()`) and removed afterwards.

If you use the `?a=` section, the default limit is 10000 instead of 10.

It's possible to export aggregated data to TSV or CSV format using `&export=tsv` or `&export=csv` accordingly. Example: `https://api.blockchair.com/bitcoin/transactions?a=date,avg(fee_usd)&q=time(2019-01-01..2019-04-01)&export=tsv`. Please note that data export is only available for aggregated data. If you need to export the whole table or its part, please use Database dumps.

Warning: the `f({:expression})` special function, the `?aq=` section, and TSV/CSV export are currently in alpha stage on our platform. Special function `price({:ticker1}_{:ticker2})` can't be used within special function `f({:expression})`. There are some known issues when sorting if `f({:expression})` is present. There are some known issues when applying the `?aq=` section to inequality filters.

Fun example

The following requests return the same result:

- `https://api.blockchair.com/bitcoin/blocks?a=sum(reward)`
- `https://api.blockchair.com/bitcoin/transactions?a=sum(output_total)&q=is_coinbase(true)`
- `https://api.blockchair.com/bitcoin/outputs?a=sum(value)&q=is_from_coinbase(true)`

Export data to TSV or CSV

Please use our Database dumps feature instead of the API (see https://blockchair.com/dumps for documentation)

Front-end visualizations

- Filters and sortings: https://blockchair.com/bitcoin/blocks
- Data aggregation: https://blockchair.com/charts

Request cost formula for infinitables

Cost is calculated by summing up the following values:

- The base cost for the table (see the table below): `2`, `5`, or `10`
- Applying a filter costs `1`
- Applying a sorting costs `0`
- Applying an offset costs `0`

- Applying an aggregation costs `10`

Applying a limit over the default multiplies the summed cost by `1 + 0.01 * number_of_rows_over_the_default_limit`. If the defaut limit is 10 and the base cost is 2, requesting 100 rows will cost `2 * (1 + 0.01 * 90) = 3.8`.

Table	Base cost
`{:btc_chain}/blocks`	`2`
`{:btc_chain}/transactions`	`5`
`{:btc_chain}/mempool/transactions`	`2`
`{:btc_chain}/outputs`	`10`
`{:btc_chain}/mempool/outputs`	`2`
`{:btc_chain}/addresses`	`2`
`{:eth_chain}/blocks`	`2`
`{:eth_chain}/uncles`	`2`
`{:eth_chain}/transactions`	`5`
`{:eth_chain}/mempool/transactions`	`2`
`{:eth_chain}/calls`	`10`
`{:eth_chain}/addresses`	`2`
`{:xin_chain}/raw/snapshots`	`1`
`{:xin_chain}/raw/mintings`	`1`
`{:xin_chain}/raw/nodes`	`1`
`bitcoin/omni/properties`	`10`
`ethereum/erc-20/tokens`	`2`
`ethereum/erc-20/transactions`	`5`

Table descriptions

Further the documentation describes each of the supported tables. Each documentation section contains a general description, and a table listing the table columns (fields) in the following format:

Column	Type	Description	Q?	S?	A?	C?

Column	Type	Description	Q?	S?	A?	C?
Column name	*Column type*	*Column description*	*Is it possible to filter by this column?*	*Is it possible to sort by this column?*	*Is it possible to group by this column?*	*Is it possible to apply aggregation functions (like `sum`) to this column?*

The following marks are possible for the `Q?` column:

- `=` — possible to use equalities only
- `*` — possible to use both equalities and inequalities
- `⌘` — possible to use special format (applies to timestamp fields)
- `~` — possible to use the `LIKE` operator
- `^` — possible to use the `STARTS WITH` operator
- `*≈` — possible to use both equalities and inequalities, may return some results which are a bit out of the set range (this is used to swiftly search over the Ethereum blockchain that uses too long wei numbers for transfer amounts)

For the `S?`, `A?`, and `C?` columns it's either `+` (which means "yes") or nothing. `⌘` means some additional options may be available (in case of aggregation it may either mean additional fields like `year` are available, or in case of functions — only `min` and `max` are available).

There can also be synthetic columns which aren't shown in the response, but you can still filter or sort by them. If there are any, they will be listed in a separate table.

Inifinitable endpoints for Bitcoin-like blockchains (Bitcoin, Bitcoin Cash, Litecoin, Bitcoin SV, Dogecoin, Dash, Groestlcoin, Zcash, eCash, Bitcoin Testnet)

`blocks` table

Endpoint:

```
https://api.blockchair.com/{:btc_chain}/blocks?{:query}
```

Where:

- `{:btc_chain}` can be one of these: `bitcoin`, `bitcoin-cash`, `litecoin`, `bitcoin-sv`, `dogecoin`, `dash`, `groestlcoin`, `zcash`, `ecash`, `bitcoin/testnet`
- `{:query}` is the query against the table (how to build a query)

Output:

`data` contains an array of database rows. Each row is in the following format:

Column	Type	Description	Q?	S?	A?	C?
id	int	Block height	*	+		⌘
hash	string `[0-9a-f]{64}`	Block hash	=	+		
date	string `YYYY-MM-DD`	Block date (UTC)			⌘	
time	string `YYYY-MM-DD HH:ii:ss`	Block time (UTC)	⌘	+		
median_time	string `YYYY-MM-DD HH:ii:ss`	Block median time (UTC)		+		
size	int	Block size in bytes	*	+		+
stripped_size †	int	Block size in bytes without taking witness information into account	*	+		+
weight †	int	Block weight in weight units	*	+		+
version	int	Version field	*	+	+	
version_hex	string `[0-9a-f]*`	Version field in hex				
version_bits	string `[01]{30}`	Version field in binary format				
merkle_root	`[0-9a-f]{64}`	Merkle root hash				
final_sapling_root §	`[0-9a-f]{64}`	Sapling root hash				
nonce	int	Nonce value	*	+		
solution §	`[0-9a-f]*`	Solution value				
anchor §	`[0-9a-f]*`	Anchor value				
bits	int	Bits field	*	+		
difficulty	float	Difficulty	*	+		+

Column	Type	Description	Q?	S?	A?	C?
chainwork	string `[0-9a-f]{64}`	Chainwork field				
coinbase_data_hex	string `[0-9a-f]*`	Hex information contained in the input of the coinbase transaction	^			
transaction_count	int	Number of transactions in the block	*	+		+
witness_count †	int	Number of transactions in the block containing witness information	*	+		+
input_count	int	Number of inputs in all block transactions	*	+		+
output_count	int	Number of outputs in all block transactions	*	+		+
input_total	int	Sum of inputs in satoshi	*	+		+
input_total_usd	float	Sum of outputs in USD	*	+		+
output_total	int	Sum of outputs in satoshi	*	+		+
output_total_usd	float	Sum of outputs in USD	*	+		+
fee_total	int	Total fee in Satoshi	*	+		+
fee_total_usd	float	Total fee in USD	*	+		+
fee_per_kb	float	Fee per kilobyte (1000 bytes of data) in satoshi	*	+		+
fee_per_kb_usd	float	Fee for kilobyte of data in USD	*	+		+
fee_per_kwu †	float	Fee for 1000 weight units of data in satoshi	*	+		+
fee_per_kwu_usd †	float	Fee for 1000 weight units of data in USD	*	+		+
cdd_total	float	Number of coindays destroyed by all transactions of the block	*	+		+
generation	int	Miner reward for the block in satoshi	*	+		+

Column	Type	Description	Q?	S?	A?	C?
generation_usd	float	Miner reward for the block in USD	*	+		+
reward	int	Miner total reward (reward + total fee) in satoshi	*	+		+
reward_usd	float	Miner total reward (reward + total fee) in USD	*	+		+
guessed_miner	string `.*`	The supposed name of the miner who found the block (the heuristic is based on `coinbase_data_bin` and the addresses to which the reward goes)	=	+	+	
is_aux ‡	boolean	Whether a block was mined using AuxPoW	=		+	
cbtx ※	string `.*`	Coinbase transaction data (encoded JSON)				
shielded_value_delta_total §	int	Amount transferred into the shielded pool	*	+		+

Additional synthetic columns

Column	Type	Description	Q?	S?	A?	C?
coinbase_data_bin	string `.*`	Text (UTF-8) representation of coinbase data. Allows you to use the `LIKE` operator: `?q=coinbase_data_bin(~hello)`	~			

Notes:

- `increased efficiency` method applies if querying `id` and `hash` columns using the `equals` operator
- † — only for Bitcoin, Litecoin, Groestlcoin, and Bitcoin Testnet (SegWit data)
- ‡ — only for Dogecoin
- ※ — only for Dash
- § — only for Zcash
- The default sorting — `id DESC`

Example output:

`https://api.blockchair.com/bitcoin/blocks?limit=1`:

```
{
  "data": [
    {
      "id": 599954,
      "hash": "0000000000000000000a405e0eb599136580eed78682bfe6648c5f7b6f81a9cb",
      "date": "2019-10-18",
      "time": "2019-10-18 17:16:18",
      "median_time": "2019-10-18 16:41:08",
      "size": 1291891,
      "stripped_size": 900520,
      "weight": 3993451,
      "version": 536870912,
      "version_hex": "20000000",
      "version_bits": "100000000000000000000000000000",
      "merkle_root":
"800c37c217eb0b53f8e5144602b8605876e12939f85d350e3d677fe89b8da476",
      "nonce": 318379413,
      "bits": 387294044,
      "difficulty": 13008091666972,
      "chainwork":
"0000000000000000000000000000000000000000096007e2e467d315afd86f91",
      "coinbase_data_hex":
"039227090452f3a95d2f706f6f6c696e2e636f6d2ffabe6d6d95254907ac051f810232ebdb4865ce2
04353bc59bbd533e40fb1cd3d29b8e0670100000000000000007570ce1586aa43da2aabdab74791c8cd1
0d4473db1006158555400000000",
      "transaction_count": 2157,
      "witness_count": 1320,
      "input_count": 6564,
      "output_count": 4969,
      "input_total": 255590274198,
      "input_total_usd": 20610300,
      "output_total": 256840274198,
      "output_total_usd": 20711100,
      "fee_total": 14959404,
      "fee_total_usd": 1206.3,
      "fee_per_kb": 11583,
      "fee_per_kb_usd": 0.93403,
      "fee_per_kwu": 3744.56,
      "fee_per_kwu_usd": 0.301955,
      "cdd_total": 7884.6687017888,
      "generation": 1250000000,
      "generation_usd": 100798,
      "reward": 1264959404,
      "reward_usd": 102004,
      "guessed_miner": "Poolin"
    }
  ],
  "context": {
    "code": 200,
    "limit": 1,
    "offset": 0,
    "rows": 1,
    "total_rows": 599955,
    "state": 599954,
    ...
  }
}
```

Request cost formula:

See request costs for infinitables

Explore visualizations on our front-end:

- https://blockchair.com/bitcoin/blocks
- https://blockchair.com/bitcoin-cash/blocks
- https://blockchair.com/litecoin/blocks
- https://blockchair.com/bitcoin-sv/blocks
- https://blockchair.com/dogecoin/blocks
- https://blockchair.com/dash/blocks
- https://blockchair.com/groestlcoin/blocks
- https://blockchair.com/zcash/blocks
- https://blockchair.com/ecash/blocks
- https://blockchair.com/bitcoin/testnet/blocks

`transactions` table

Endpoints:

- `https://api.blockchair.com/{:btc_chain}/transactions?{:query}` (for blockchain transactions)
- `https://api.blockchair.com/{:btc_chain}/mempool/transactions?{:query}` (for mempool transactions)

Where:

- `{:btc_chain}` can be one of these: `bitcoin`, `bitcoin-cash`, `litecoin`, `bitcoin-sv`, `dogecoin`, `dash`, `groestlcoin`, `zcash`, `ecash`, `bitcoin/testnet`
- `{:query}` is the query against the table (how to build a query)

Output:

`data` contains an array of database rows. Each row is in the following format:

Column	Type	Description	Q?	S?	A?	C?
block_id	int	The height (id) of the block containing the transaction	*	+	+	
id	int	Internal Blockchair transaction id (not related to the blockchain, used for internal purposes)	*	+		

Column	Type	Description	Q?	S?	A?	C?
hash	string `[0-9a-f]{64}`	Transaction hash	=			
date	string `YYYY-MM-DD`	The date of the block containing the transaction (UTC)			⌘	
time	string `YYYY-MM-DD HH:ii:ss`	Timestamp of the block containing the transaction (UTC)	⌘	+		
size	int	Transaction size in bytes	*	+		+
weight †	int	Weight of transaction in weight units	*	+		+
version	int	Transaction version field	*	+	+	
lock_time	int	Lock time — can be either a block height, or a unix timestamp	*	+		
is_coinbase	boolean	Is it a coinbase (generating new coins) transaction? (For such a transaction `input_count` is equal to `1` and means there's a synthetic coinbase input)	=		+	
has_witness †	boolean	Is there a witness part in the transaction (using SegWit)?	=		+	
input_count	int	Number of inputs	*	+	+	+
output_count	int	Number of outputs	*	+	+	+
input_total	int	Input value in satoshi	*	+		+
input_total_usd	float	Input value in USD	*	+		+
output_total	int	Output value in satoshi	*	+		+
output_total_usd	float	Total output value in USD	*	+		+
fee	int	Fee in satoshi	*	+		+
fee_usd	float	Fee in USD	*	+		+
fee_per_kb	float	Fee per kilobyte (1000 bytes) of data in satoshi	*	+		+
fee_per_kb_usd	float	Fee for kilobyte of data in USD	*	+		+
fee_per_kwu †	float	Fee for 1000 weight units of data in satoshi	*	+		+

Column	Type	Description	Q?	S?	A?	C?
fee_per_kwu_usd †	float	Fee for 1000 weight units of data in USD	*	+		+
cdd_total	float	The number of destroyed coindays	*	+		+

Additional Dash-specific columns:

Column	Type	Description	Q?	S?	A?	C?
type ※	string (enum)	Transaction type, one of the following: `simple`, `proregtx`, `proupservtx`, `proupregtx`, `prouprevtx`, `cbtx`, `qctx`, `subtxregister`, `subtxtopup`, `subtxresetkey`, `subtxcloseaccount`	=		+	
is_instant_lock ※	boolean	Is instant lock?	=			
is_special ※	boolean	`true` for all transaction types except `simple`	=			
special_json ※	string `.*`	Special transaction data (encoded JSON string)				

Additional Zcash-specific columns:

Column	Type	Description	Q?	S?	A?	C?
shielded_value_delta §	int	Amount transferred into the shielded pool	*	+		+
version_group_id §	string `[0-9a-f]*`	Special version field	=		+	
is_overwintered §	boolean	Is overwintered?	=		+	
expiry_height §	int	Expiry height	*	+		
join_split_raw §	json	Raw 'v_join_split' value				
shielded_input_raw §	json	Raw 'v_shielded_spend' value				
shielded_output_raw §	json	Raw 'v_shielded_output' value				
binding_signature §	string `[0-9a-f]*`	Binding signature				

Notes:

- `increased efficiency` method applies if querying `id` and `hash` columns using the `equals` operator
- † — only for Bitcoin, Litecoin, Groestlcoin, and Bitcoin Testnet (SegWit data)
- ※ — only for Dash
- § — only for Zcash
- The default sorting — `id DESC`
- `block_id` for mempool transactions is `-1`

Example output:

`https://api.blockchair.com/bitcoin/transactions?limit=1`:

```
{
  "data": [
    {
      "block_id": 600573,
      "id": 467508697,
      "hash": "ee13104d4331cad2fff5ab6cd249a9fec940d64df442a6de5f51ea63c34ef8ff",
      "date": "2019-10-22",
      "time": "2019-10-22 19:09:34",
      "size": 250,
      "weight": 672,
      "version": 1,
      "lock_time": 0,
      "is_coinbase": false,
      "has_witness": true,
      "input_count": 1,
      "output_count": 2,
      "input_total": 29340442,
      "input_total_usd": 2408.9,
      "output_total": 29340274,
      "output_total_usd": 2408.89,
      "fee": 168,
      "fee_usd": 0.0137931,
      "fee_per_kb": 672,
      "fee_per_kb_usd": 0.0551723,
      "fee_per_kwu": 250,
      "fee_per_kwu_usd": 0.0205254,
      "cdd_total": 29.154456198211
    }
  ],
  "context": {
    "code": 200,
    "limit": 1,
    "offset": 0,
    "rows": 1,
    "total_rows": 467508698,
    "state": 600573,
    ...
  }
}
```

Request cost formula:

See request costs for infinitables

Explore visualizations on our front-end:

- https://blockchair.com/bitcoin/transactions
- https://blockchair.com/bitcoin-cash/transactions
- https://blockchair.com/litecoin/transactions
- https://blockchair.com/bitcoin-sv/transactions
- https://blockchair.com/dogecoin/transactions
- https://blockchair.com/dash/transactions
- https://blockchair.com/groestlcoin/transactions
- https://blockchair.com/zcash/transactions
- https://blockchair.com/ecash/transactions
- https://blockchair.com/bitcoin/testnet/transactions
- https://blockchair.com/bitcoin/mempool/transactions
- https://blockchair.com/bitcoin-cash/mempool/transactions
- https://blockchair.com/litecoin/mempool/transactions
- https://blockchair.com/bitcoin-sv/mempool/transactions
- https://blockchair.com/dogecoin/mempool/transactions
- https://blockchair.com/dash/mempool/transactions
- https://blockchair.com/groestlcoin/mempool/transactions
- https://blockchair.com/zcash/mempool/transactions
- https://blockchair.com/ecash/mempool/transactions
- https://blockchair.com/bitcoin/testnet/mempool/transactions

`outputs` table

Endpoints:

- `https://api.blockchair.com/{:btc_chain}/outputs?{:query}` (input and output data for blockchain transactions)
- `https://api.blockchair.com/{:btc_chain}/mempool/outputs?{:query}` (input and output data for mempool transactions)

Where:

- `{:btc_chain}` can be one of these: `bitcoin`, `bitcoin-cash`, `litecoin`, `bitcoin-sv`, `dogecoin`, `dash`, `groestlcoin`, `zcash`, `ecash`, `bitcoin/testnet`
- `{:query}` is the query against the table (how to build a query)

Output:

`data` contains an array of database rows. Rows represent transaction outputs (that also become transaction inputs when they are spent). Each row is in the following format:

Column	Type	Description	Q?	S?	A?	C?

Column	Type	Description	Q?	S?	A?	C?
block_id	int	Id of the block containing the transaction cointaining the output	*	+	+	
transaction_id	int	Internal Blockchair transaction id (not related to the blockchain, used for internal purposes)	*	+		
index	int	Output index in the transaction (from 0)	*	+		
transaction_hash	string `[0-9a-f]{64}`	Transaction hash				
date	string `YYYY-MM-DD`	Date of the block containing the output (UTC)				
time	string `YYYY-MM-DD HH:ii:ss`	Timestamp of the block containing the output (UTC)	⌘	+		
value	int	Monetary value of the output	*	+		+
value_usd	float	Monetary value of the output in USD	*	+		+
recipient	string `[0-9a-zA-Z\-]*`	Address or synthetic address of the output recipient (see address types description)	=	+	+	
type	string (enum)	Output type, one of the following: `pubkey`, `pubkeyhash`, `scripthash`, `multisig`, `nulldata`, `nonstandard`, `witness_v0_scripthash`, `witness_v0_keyhash`, `witness_unknown`	=	+	+	
script_hex	string `[0-9a-f]*`	Hex value of the output script. Filtering using the `STARTS WITH` operator is performed for `nulldata` outputs only.	^			
is_from_coinbase	boolean	Is it a coinbase transaction output?	=		+	

Column	Type	Description	Q?	S?	A?	C?
is_spendable	null or boolean	Is it theoretically possible to spend this output? For `pubkey` and `multisig` outputs, the existence of the corresponding private key is tested, in that case `true` and `false` are the possible values depending on the result of the check. For `nulldata` outputs it is always `false`. For other types it is impossible to check trivially, in that case `null` is yielded.	=		+	
is*spent | boolean | Has this output been spent? ** (`spending* fields below yield null if it is not)** | = | | +`						
spending_block_id	null or int	Id of the block containing the spending transaction	*	+	+	
spending_transaction_id	null or int	Internal Blockchair transaction id where the output was spent	*	+		
spending_index	null or int	Input index in the spending transaction (from 0)	*	+		
spending_transaction_hash	null or string `[0-9a-f]{64}`	Spending transaction hash				
spending_date	null or string `YYYY-MM-DD`	Date of the block, in which the output was spent			⌘	
spending_time	null or string `YYYY-MM-DD HH:ii:ss`	Timestamp of the block in which the output was spent	⌘	+		
spending_value_usd	null or float	Monetary value of the output in USD at the time of `spending_date`	*	+		+
spending_sequence	null or int	Sequence field	*	+		

Column	Type	Description	Q?	S?	A?	C?
spending_signature_hex	null or string `[0-9a-f]*`	Hex value of the spending script (signature)				
spending_witness †	null or string	Witness information (comma-separated, may start with a comma if the first witness element is empty)				
lifespan	null or int	The number of seconds from the time of the output creation (`time`) to its spending (`spending_time`), `null` if the output hasn't been spent	*	+		+
cdd	null or float	The number of coindays destroyed spending the output, `null` if the output hasn't been spent	*	+		+

Additional synthetic columns

Column	Type	Description	Q?	S?	A?	C?
script_bin	string `.*`	Text (UTF-8) representation of `script_hex`. Allows you to use the `LIKE` operator: `?q=script_bin(~hello)`. Filtering using the `LIKE` operator is performed for `nulldata` outputs only.	~			

Notes:

- `increased efficiency` method applies if querying `transaction_id` and `spending_transaction_id` columns using the `equals` operator
- † — only for Bitcoin, Litecoin, Groestlcoin, and Bitcoin Testnet (SegWit data)
- The default sorting — `transaction_id DESC`
- `spending_*` columns yield `null` for outputs that haven't been spent yet
- `block_id` for mempool transactions is `-1`
- `spending_block_id` is `-1` for outputs being spent by an unconfirmed transaction
- This particular table is in beta test mode on our platform. It's possible to receive duplicate rows for outputs which have just been spent. Sometimes duplicates are removed automatically, but in that case the number of rows may be less than the set limit on the number of rows. There's an additional context key `context.pre_rows` which contains the number of rows that should've been returned before the duplicate removal process.

Example outputs:

`https://api.blockchair.com/bitcoin/outputs?q=is_spent(true)&limit=1` (example of a spent output created in `transaction_hash` transaction and spent in `spending_transaction_hash` transaction :

```
{
  "data": [
    {
      "block_id": 600573,
      "transaction_id": 467508619,
      "index": 1,
      "transaction_hash":
"a3c43b4bdc245e0675812e2779703ef5cf2c0e15df8b46d99e6e085a6bbedbe7",
      "date": "2019-10-22",
      "time": "2019-10-22 19:09:34",
      "value": 14638337,
      "value_usd": 1201.83,
      "recipient": "3FdhDDr42mMXX4tpG6dPkHuoCrPTJk3yjH",
      "type": "scripthash",
      "script_hex": "a91498f0e489f60c3971fa304290257374d7ea92292b87",
      "is_from_coinbase": false,
      "is_spendable": null,
      "is_spent": true,
      "spending_block_id": 600573,
      "spending_transaction_id": 467508620,
      "spending_index": 0,
      "spending_transaction_hash":
"6350ac986bd8974fafbf3fc8c498a923dc1b8c6fa40f6569227f343aa6a50ce1",
      "spending_date": "2019-10-22",
      "spending_time": "2019-10-22 19:09:34",
      "spending_value_usd": 1201.83,
      "spending_sequence": 4294967294,
      "spending_signature_hex": "16001433f44aa318c7cac6703f0d09f2dc4314dd68d769",
      "spending_witness":
"304402204fe6a8c36d400f64975f7a08119f7e311b75d32b358a48bfe65fb355a40fd1230220122ed
99fc4024290a82efd0d94707f23eeac513978a211f6f4893e11af3b9c3301,027f502e7a018afa8d50
dd17c459d987e7754486b46f131bfe1b0e2841f3afbb64",
      "lifespan": 0,
      "cdd": 0
    }
  ],
  "context": {
    "code": 200,
    "limit": 1,
    "offset": 0,
    "rows": 1,
    "pre_rows": 1,
    "total_rows": 1150457958,
    "state": 600573,
    ...
  }
}
```

`https://api.blockchair.com/bitcoin/outputs?q=is_spent(false)&limit=1`
(example of an uspent output):

```
{
  "data": [
    {
      "block_id": 600573,
      "transaction_id": 467508697,
      "index": 1,
      "transaction_hash":
"ee13104d4331cad2fff5ab6cd249a9fec940d64df442a6de5f51ea63c34ef8ff",
      "date": "2019-10-22",
      "time": "2019-10-22 19:09:34",
      "value": 23725010,
      "value_usd": 1947.86,
      "recipient": "3P8771VCWU2tyFj7gPS1ZuV4JzJrJWjn3K",
      "type": "scripthash",
      "script_hex": "a914eb195d6b2b50fc134078f65b72741d4c37e821de87",
      "is_from_coinbase": false,
      "is_spendable": null,
      "is_spent": false,
      "spending_block_id": null,
      "spending_transaction_id": null,
      "spending_index": null,
      "spending_transaction_hash": null,
      "spending_date": null,
      "spending_time": null,
      "spending_value_usd": null,
      "spending_sequence": null,
      "spending_signature_hex": null,
      "spending_witness": null,
      "lifespan": null,
      "cdd": null
    }
  ],
  "context": {
    "code": 200,
    "limit": 1,
    "offset": 0,
    "rows": 1,
    "pre_rows": 1,
    "total_rows": 99482704,
    "state": 600573,
    ...
  }
}
```

Request cost formula:

See request costs for infinitables

Explore visualizations on our front-end:

- https://blockchair.com/bitcoin/outputs
- https://blockchair.com/bitcoin-cash/outputs

- https://blockchair.com/litecoin/outputs
- https://blockchair.com/bitcoin-sv/outputs
- https://blockchair.com/dogecoin/outputs
- https://blockchair.com/dash/outputs
- https://blockchair.com/groestlcoin/outputs
- https://blockchair.com/zcash/outputs
- https://blockchair.com/bitcoin/testnet/outputs
- https://blockchair.com/bitcoin/mempool/outputs
- https://blockchair.com/bitcoin-cash/mempool/outputs
- https://blockchair.com/litecoin/mempool/outputs
- https://blockchair.com/bitcoin-sv/mempool/outputs
- https://blockchair.com/dogecoin/mempool/outputs
- https://blockchair.com/dash/mempool/outputs
- https://blockchair.com/groestlcoin/mempool/outputs
- https://blockchair.com/zcash/mempool/outputs
- https://blockchair.com/bitcoin/testnet/mempool/outputs

`addresses` view

Endpoints:

```
https://api.blockchair.com/{:btc_chain}/addresses?{:query}
```

Where:

- `{:btc_chain}` can be one of these: `bitcoin`, `bitcoin-cash`, `litecoin`, `bitcoin-sv`, `dogecoin`, `dash`, `groestlcoin`, `zcash`, `ecash`, `bitcoin/testnet`
- `{:query}` is the query against the table (how to build a query)

Output:

The `addresses` view contains the list of all addresses and their confirmed balances. Unlike other infinitables (`blocks`, `transactions`, `outputs`) this table isn't live, it's automatically updated every 5 minutes with new data, thus we classify it as a "view". `data` contains an array of database rows. Each row is in the following format:

Column	Type	Description	Q?	S?	A?	C?
address	string `[0-9a-zA-Z\-]*`	Bitcoin address or synthetic address				
balance	int	Its confirmed balance	*	+		+

Notes:

the default sorting — `balance DESC`

Example outputs:

https://api.blockchair.com/bitcoin/addresses:

```
{
  "data": [
    {
      "address": "34xp4vRoCGJym3xR7yCVPFHoCNxv4Twseo",
      "balance": 16625913046297
    },
    {
      "address": "35hK24tcLEWcgNA4JxpvbkNkoAcDGqQPsP",
      "balance": 15100013129630
    },
    {
      "address": "385cR5DM96n1HvBDMzLHPYcw89fZAXULJP",
      "balance": 11730490887099
    },
    {
      "address": "3CgKHXR17eh2xCj2RGnHJHTDjPpqaNDgyT",
      "balance": 11185824580401
    },
    {
      "address": "37XuVSEpWW4trkfmvWzegTHQt7BdktSKUs",
      "balance": 9450576862072
    },
    {
      "address": "183hmJGRuTEi2YDCWy5iozY8rZtFwVgahM",
      "balance": 8594734898577
    },
    {
      "address": "1FeexV6bAHb8ybZjqQMjJrcCrHGW9sb6uF",
      "balance": 7995720088144
    },
    {
      "address": "3D2oetdNuZUqQHPJmcMDDHYoqkyNVsFk9r",
      "balance": 7689310178244
    },
    {
      "address": "1HQ3Go3ggs8pFnXuHVHRytPCq5fGG8Hbhx",
      "balance": 6937013094817
    },
    {
      "address": "3E35SFZkfLMGo4qX5aVs1bBDSnAuGgBH33",
      "balance": 6507708194519
    }
  ],
  "context": {
    "code": 200,
    "limit": 10,
    "offset": 0,
    "rows": 10,
    "total_rows": 27908261,
    "state": 600568,
    ...
  }
}
```

`https://api.blockchair.com/bitcoin/addresses?a=sum(balance)` (total balance of all addresses should be the same as the total number of coins minted):

```
{
  "data": [
    {
      "sum(balance)": 1800708303344571
    }
  ],
  "context": {
    "code": 200,
    "limit": 10000,
    "offset": null,
    "rows": 1,
    "total_rows": 1,
    "state": 600568,
    ...
  }
}
```

`https://api.blockchair.com/bitcoin/addresses?a=count()&q=balance(1..10)` (shows the number of addresses holding [1..10] satoshi):

```
{
  "data": [
    {
      "count()": 574591
    }
  ],
  "context": {
    "code": 200,
    "limit": 10000,
    "offset": null,
    "rows": 1,
    "total_rows": 1,
    "state": 600568,
    ...
  }
}
```

Request cost formula:

See request costs for infinitables

Explore visualizations on our front-end:

- https://blockchair.com/bitcoin/addresses
- https://blockchair.com/bitcoin-cash/addresses
- https://blockchair.com/litecoin/addresses
- https://blockchair.com/bitcoin-sv/addresses
- https://blockchair.com/dogecoin/addresses
- https://blockchair.com/dash/addresses
- https://blockchair.com/groestlcoin/addresses

- https://blockchair.com/zcash/addresses
- https://blockchair.com/bitcoin/testnet/addresses

Inifinitable endpoints for Ethereum and Ethereum Goerli Testnet

`blocks` table

Endpoint:

```
https://api.blockchair.com/{:eth_chain}/blocks?{:query}
```

Where:

- `{:eth_chain}` can only be `ethereum` or `ethereum/testnet`
- `{:query}` is the query against the table (how to build a query)

Output:

`data` contains an array of database rows. Each row is in the following format:

Column	Type	Description	Q?	S?	A?	C?
id	int	Block id	*	+		⌘
hash	string `0x[0-9a-f]{64}`	Block hash	=			
date	string `YYYY-MM-DD`	Block date (UTC)			⌘	
time	string `YYYY-MM-DD HH:ii:ss`	Block time (UTC)	⌘	+		
size	int	Block size in bytes	*	+		+
miner	string `0x[0-9a-f]{40}`	Address the miner who found the block	=		+	
extra_data_hex	string `[0-9a-f]*`	Additional data included by the miner	^			
difficulty	int	Difficulty	*	+		+
gas_used	int	Gas amount used by block transactions	*	+		+
gas_limit	int	Gas limit for the block set by the miner	*	+		+

Column	Type	Description	Q?	S?	A?	C?
logs_bloom	string `[0-9a-f]*`	Logs bloom field				
mix_hash	string `[0-9a-f]{64}`	Mix hash				
nonce	string `[0-9a-f]*`	Nonce value				
receipts_root	string `[0-9a-f]{64}`	Receipts root				
sha3_uncles	string `[0-9a-f]{64}`	SHA3 Uncles				
state_root	string `[0-9a-f]{64}`	State root				
total_difficulty	numeric string	Total difficulty at the `id` point				
transactions_root	string `[0-9a-f]{64}`	Transactions root				
uncle_count	int	Number of block uncles	*	+		+
transaction_count	int	Number of transactions in the block	*	+		+
synthetic_transaction_count	int	Number of synthetic transactions (they do not exist as separate transactions on the blockchain, but they change the state, e.g., genesis block transactions, miner rewards, DAO-fork transactions, etc.)	*	+		+
call_count	int	Total number of calls spawned by transactions	*	+		+
synthetic_call_count	int	Number of synthetic calls (same as synthetic transactions)	*	+		+

Column	Type	Description	Q?	S?	A?	C?
value_total	numeric string	Monetary value of all block transactions in wei, hereinafter `numeric string` - numeric (integer or float in some cases) value passed as a string, as values in wei do not fit into integer	*≈	+		+
value_total_usd	float	Monetary value of all block transactions in USD	*	+		+
internal_value_total	numeric string	Monetary value of all internal calls in the block in wei	*≈	+		+
internal_value_total_usd	float	Monetary value of all internal calls in a block in USD	*	+		+
generation	numeric string	The reward of a miner for the block generation in wei	*≈	+		+
generation_usd	float	The reward of a miner for the block generation in USD	*	+		+
uncle_generation	numeric string	Total reward of uncle miners in wei	*≈	+		+
uncle_generation_usd	float	Total reward of uncle miners in USD	*	+		+
fee_total	numeric string	Total fee in wei	*≈	+		+
fee_total_usd	float	Total fee in USD	*	+		+
reward	numeric string	Total reward of the miner in the wei (reward for finding the block + fees)	*≈	+		+
reward_usd	float	Total reward of the miner in USD	*	+		+

Additional synthetic columns

Column	Type	Description	Q?	S?	A?	C?

Column	Type	Description	Q?	S?	A?	C?
extra_data_bin	string `.*`	Text representation (UTF-8) of extra data. Allows you to use the `LIKE` operator: `?q=extra_data_bin(~hello)`	`~`			

Notes:

- `increased efficiency` method applies if querying `id` and `hash` columns using the `equals` operator
- Search by fields that contain values in wei (`value_total`, `internal_value_total`, `generation`, `uncle_generation`, `fee_total`, `reward`) may be with some inaccuracies
- The difference between `value_total` and `internal_value_total`: e.g., a transaction itself sends 0 eth, but this transaction is a call of a contract that sends someone, let's say, 10 eth. Then `value` will be 0 eth, and `internal_value` - 10 eth
- The default sorting is `id DESC`

Example output:

`https://api.blockchair.com/ethereum/blocks?limit=1`:

```
{
  "data": [
    {
      "id": 8766253,
      "hash":
"0xf36522b1f6ee2350c322a309ebdffe9afadc7d68713ad5b3a064657c81607ab7",
      "date": "2019-10-18",
      "time": "2019-10-18 17:39:40",
      "size": 32170,
      "miner": "0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5",
      "extra_data_hex": "50505945206e616e6f706f6f6c2e6f7267",
      "difficulty": 2408192424049377,
      "gas_used": 9895313,
      "gas_limit": 9920736,
      "logs_bloom":
"2e8e09c1046d3063207c00c2440098ac0824d0ca0818d201500a1987588a284b001315981c227c860
10880300083629c802895bb1608860a02a818a2202d405002a6140281390b00d880610822005011440
244527f24b80e3200a405848034043c3028c99218304b8040180210401c005008924d1925c11a00410
0b14e1270980d21146d4c1a1029130024a0801400350858088c03000061421007b866a8d60c0a0cb14
2100028e0c39002b010c0320082a49000040fe870022c0080024e1120a0d21ac23289060221c390080
800ab442c244130cea8102c2c20404e188468430c52aa20143110200706e642c52f4008080ac719109
32415a02108020d910780",
      "mix_hash":
"65f9fe3204d652ce2f82adface45e8c32cfacb0b80a3d1acaff8969457911342",
      "nonce": "13915815879145322367",
      "receipts_root":
"cfba6974cf3257f2c2cf674a4e2f422b9623646120364ce7be84040d7d2b9578",
      "sha3_uncles":
"1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347",
      "state_root":
"270dca9a521aa1b900cd0749a1a1c1413328cdaff1ccc7f9bcfe6e06751f0781",
      "total_difficulty": "12439420564755992111056",
      "transactions_root":
"62523508847380a506452289abe504fdef7b5e9e96cbfd166f0fd359a4837f92",
      "uncle_count": 0,
      "transaction_count": 172,
      "synthetic_transaction_count": 1,
      "call_count": 333,
      "synthetic_call_count": 1,
      "value_total": "14324135521180578322",
      "value_total_usd": 2536.74001038483,
      "internal_value_total": "15524135521180578322",
      "internal_value_total_usd": 2749.25461609772,
      "generation": "2000000000000000000",
      "generation_usd": 354.191009521484,
      "uncle_generation": "0",
      "uncle_generation_usd": 0,
      "fee_total": "29252299880000000",
      "fee_total_usd": 5.1804508126612,
      "reward": "2029252299880000000",
      "reward_usd": 359.371460334146
    }
  ],
  "context": {
    "code": 200,
    "limit": 1,
    "offset": 0,
    "rows": 1,
    "total_rows": 8766254,
    "state": 8766260,
    "state_layer_2": 8766249,
    ...
  }
}
```

Request cost formula:

See request costs for infinitables

Explore visualizations on our front-end:

https://blockchair.com/ethereum/blocks

`uncles` table

Endpoint:

`https://api.blockchair.com/{:eth_chain}/uncles?{:query}`

Where:

- `{:eth_chain}` can only be `ethereum` or `ethereum/testnet`
- `{:query}` is the query against the table (how to build a query)

Output:

Returns information about uncles. `data` contains an array of database rows. Each row is in the following format:

Column	Type	Description	Q?	S?	A?	C?
parent_block_id	int	Parent block id	*	+	+	
index	int	Uncle index in the block	*	+		
id	int	Uncle id	*	+		
hash	string `0x[0-9a-f]{64}`	Uncle hash (with 0x)	=			
date	string `YYYY-MM-DD`	Date of generation (UTC)			⌘	
time	string `YYYY-MM-DD HH:ii:ss`	Timestamp of generation (UTC)	⌘	+		
size	int	Uncle size in bytes	*	+		+

Column	Type	Description	Q?	S?	A?	C?
miner	string `0x[0-9a-f]{40}`	Address of the rewarded miner (with 0x)	=		+	
extra_data_hex	string `[0-9a-f]*`	Additional data included by the miner	^			
difficulty	int	Difficulty	*	+		+
gas_used	int	Amount of gas used by transactions	*	+		+
gas_limit	int	Gas limit for the block set up by the miner	*	+		+
logs_bloom	string `[0-9a-f]*`	Logs bloom field				
mix_hash	string `[0-9a-f]{64}`	Hash mix				
nonce	string `[0-9a-f]*`	Nonce value				
receipts_root	string `[0-9a-f]{64}`	Receipts root				
sha3_uncles	string `[0-9a-f]{64}`	Uncles hash				
state_root	string `[0-9a-f]{64}`	State root				
transactions_root	string `[0-9a-f]{64}`	Transactions root				
generation	numeric string	The reward of the miner who generated the uncle, in wei	*≈	+		+
generation_usd	float	The award of the miner who generated uncle, in USD	*	+		+

Additional synthetic columns

Column	Type	Description	Q?	S?	A?	C?
extra_data_bin	string `.*`	Text (UTF-8) representation of extra data. Allows you to use the `LIKE` operator:`?Q=extra_data_bin(~hello)`	~			

Notes:

- `increased efficiency` method applies if querying `parent_block_id` and `hash` columns using the `equals` operator

- Search by fields that contain values in wei (`generation`) may be with some inaccuracies
- The difference between `value_total` and `internal_value_total`: a transaction itself may send, say, 0 eth, but this transaction may call a contract which sends someone 10 eth. In that case `value` will be 0 eth, and `internal_value` will be 10 eth
- The default sorting is `parent_block_id DESC`

Example output:

`https://api.blockchair.com/ethereum/uncles?limit=1`:

```
{
  "data": [
    {
      "parent_block_id": 8792054,
      "index": 0,
      "id": 8792051,
      "hash":
"0x41a4d3a79644ada10207cd41f8027a3d4e506d4cbde58750a98d3ec2afce402d",
      "date": "2019-10-22",
      "time": "2019-10-22 19:10:41",
      "size": 526,
      "miner": "0xb2930b35844a230f00e51431acae96fe543a0347",
      "extra_data_hex": "73696e6733",
      "difficulty": 2374634862657186,
      "gas_used": 9979194,
      "gas_limit": 9989371,
      "logs_bloom":
"945c08608049b629008740f22070128c0602c50010d012952a08280b22022b608cc4507918e00962a
4a049440320251192429006194812fb587ad87421e4a8002a0401c405658b208898920f828646517f2
06444b10ec162024807418380a10ac51084000625802300 2c008c66c52d220e683a2400c643600101a
2720a0108446102112d41a0900105000005a212240e1012e1c17502492000c00a84823d14040308940
51690f2304e484190028201b280840044a50c083020540380183515111 0e354e2288184002073d9080
70a44e03cb809019308738c211b4100118064a080f1a60003881a880d1144c02100c00c1200488230d
91841c02e5884d4b00401",
      "mix_hash":
"3e26a6c8520bdb3afc6ff13d46f8906a508787fc3c8021656f0fe74834728538",
      "nonce": "2551618406869966062",
      "receipts_root":
"fdcb14f98b77953add5ad2115b74291c1aeeab91e5027e30a888db72ac55d2c1",
      "sha3_uncles":
"1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347",
      "state_root":
"8aab503534b41e0fa32d242829fb5ac1cae3e034db1c22a61cf15be2e2b8ca3f",
      "transactions_root":
"f47354e86bd38e6d7cbd54cd2556fce97221a0f760c518ee226f3f5472432950",
      "generation": "1250000000000000000",
      "generation_usd": 217.484378814697
    }
  ],
  "context": {
    "code": 200,
    "limit": 1,
    "offset": 0,
    "rows": 1,
    "total_rows": 944557,
    "state": 8792093,
    "state_layer_2": 8792080,
    ...
  }
}
```

Request cost formula:

See request costs for infinitables

Explore visualizations on our front-end:

https://blockchair.com/ethereum/uncles

`transactions` table

Endpoint:

- `https://api.blockchair.com/{:eth_chain}/transactions?{:query}` (for blockchain transactions)
- `https://api.blockchair.com/{:eth_chain}/mempool/transactions?{:query}` (for mempool transactions)

Where:

- `{:eth_chain}` can only be `ethereum` or `ethereum/testnet`
- `{:query}` is the query against the table (how to build a query)

Output:

`data` contains an array of database rows. Each row is in the following format:

Column	Type	Description	Q?	S?	A?	C?
block_id	int	Id of the block containing the transaction	*	+	+	
id	int	Internal Blockchair transaction id (not related to the blockchain, used for internal purposes)	*	+		
index †‡	int	The transaction index number in the block	*	+		
hash ‡	string `0x[0-9a-f]{64}`	Transaction hash	=			
date	string `YYYY-MM-DD`	Date of the block containing the transaction (UTC)			⌘	
time	string `YYYY-MM-DD HH:ii:ss`	Time of the block containing the transaction (UTC)	⌘	+		
failed †	bool	Failed transaction or not?	=		+	
type †	string (enum)	Transaction type with one of the following values: `call`, `create`, `call_tree`, `create_tree`, `synthetic_coinbase`. Description in the note below.	=	+	+	

Column	Type	Description	Q?	S?	A?	C?
sender ‡	string `0x[0-9a-f]{40}`	Address of the transaction sender	=		+	
recipient	string `0x[0-9a-f]{40}`	Address of the transaction recipient	=		+	
call_count †	int	Number of calls in the transaction	*	+		+
value	numeric string	Monetary value of transaction in wei	*≈	+		+
value_usd	float	Value of transaction in USD	*	+		+
internal_value †	numeric string	Value of all internal calls in the transaction in wei	*≈	+		+
internal_value_usd †	float	Value of all internal calls in the transaction in USD	*	+		+
fee †‡	numeric string	Fee in wei	*≈	+		+
fee_usd †‡	float	Fee in USD	*	+		+
gas_used †‡	int	Amount of gas used by a transaction	*	+		+
gas_limit ‡	int	Gas limit for transaction set by the sender	*	+		+
gas_price ‡	int	Price for gas set by the sender	*	+		+
input_hex ‡	string `[0-9a-f]*`	Transaction input data (hex)	^			
nonce ‡	int	Nonce value				
v ‡	string `[0-9a-f]*`	V value				
r ‡	string `[0-9a-f]*`	R value				
s ‡	string `[0-9a-f]*`	S value				

Additional synthetic columns

Column	Type	Description	Q?	S?	A?	C?
input_bin	string .*	Text (UTF-8) representation of input data. Allows you to use the `LIKE` operator: `?q=input_bin(~hello)`	~			

Possible types (`type`) of transactions:

- `call` — the transaction transfers the value, but there are no more calls (a simple ether sending, not in favor of a contract, or the call to a contract that does nothing)
- `create` — create a new contract
- `call_tree` — the transaction calls a contract that makes some other calls
- `create_tree` — create a new contract that create contracts or starts making calls
- `synthetic_coinbase` — a synthetic transaction for awarding a reward to the miner (block or uncle)

Notes:

- `increased efficiency` method applies if querying `id` and `hash` columns using the `equals` operator
- † — value is `null` for transactions in the mempool
- ‡ — value is `null` if `type` is `synthetic_coinbase`
- Search by fields that contain values in wei (`value_total`, `internal_value_total`, `generation`, `uncle_generation`, `fee_total`, `reward`) may be with some inaccuracies
- The difference between `value_total` and `internal_value_total`: e.g., a transaction itself sends 0 eth, but this transaction is a call of a contract that sends someone, let's say, 10 eth. Then `value` will be 0 eth, and `internal_value` - 10 eth
- The default sorting — `id DESC`
- `block_id` for mempool transactions is `-1`

Example output:

`https://api.blockchair.com/ethereum/transactions?q=block_id(46147)`:

```
{
  "data": [
    {
      "block_id": 46147,
      "id": 46147000001,
      "index": null,
      "hash": null,
      "date": "2015-08-07",
      "time": "2015-08-07 03:30:33",
      "failed": false,
      "type": "synthetic_coinbase",
      "sender": null,
      "recipient": "0xe6a7a1d47ff21b6321162aea7c6cb457d5476bca",
      "call_count": 1,
      "value": "6050000000000000000",
      "value_usd": 6.05,
      "internal_value": "6050000000000000000",
      "internal_value_usd": 6.05,
      "fee": null,
      "fee_usd": null,
      "gas_used": null,
      "gas_limit": null,
      "gas_price": null,
      "input_hex": null,
      "nonce": null,
      "v": null,
      "r": null,
      "s": null
    },
    {
      "block_id": 46147,
      "id": 46147000000,
      "index": 0,
      "hash":
"0x5c504ed432cb51138bcf09aa5e8a410dd4a1e204ef84bfed1be16dfba1b22060",
      "date": "2015-08-07",
      "time": "2015-08-07 03:30:33",
      "failed": false,
      "type": "call",
      "sender": "0xa1e4380a3b1f749673e270229993ee55f35663b4",
      "recipient": "0x5df9b87991262f6ba471f09758cde1c0fc1de734",
      "call_count": 1,
      "value": "31337",
      "value_usd": 3.1337e-14,
      "internal_value": "31337",
      "internal_value_usd": 3.1337e-14,
      "fee": "1050000000000000000",
      "fee_usd": 1.05,
      "gas_used": 21000,
      "gas_limit": 21000,
      "gas_price": 50000000000000,
      "input_hex": "",
      "nonce": "0",
      "v": "1c",
      "r": "88ff6cf0fefd94db46111149ae4bfc179e9b94721fffd821d38d16464b3f71d0",
      "s": "45e0aff800961cfce805daef7016b9b675c137a6a41a548f7b60a3484c06a33a"
    }
  ],
  "context": {
    "code": 200,
    "limit": 10,
    "offset": 0,
    "rows": 2,
    "total_rows": 2,
    "state": 8791945,
    "state_layer_2": 8791935,
    ...
  }
}
```

Request cost formula:

See request costs for infinitables

Explore visualizations on our front-end:

- https://blockchair.com/ethereum/transactions
- https://blockchair.com/ethereum/mempool/transactions

`calls` table

Endpoint:

`https://api.blockchair.com/{:eth_chain}/calls?{:query}`

Where:

- `{:eth_chain}` can only be `ethereum` or `ethereum/testnet`
- `{:query}` is the query against the table (how to build a query)

Output:

Returns information about internal transaction calls. `data` contains an array of database rows. Each row is in the following format:

Column	Type	Description	Q?	S?	A?	C?
block_id	int	Block id containing a call	*	+	+	
transaction_id	int	Transaction id containing the call	*	+		
transaction_hash †	string `0x[0-9a-f]{64}`	Transaction hash (with 0x) containing the call	=			
index	string	Call index within the transaction (tree-like, e.g., "0.8.1")	=	+		

Column	Type	Description	Q?	S?	A?	C?
depth	int	Call depth within the call tree (starting at 0)	*	+		
date	string `YYYY-MM-DD`	Date of the block that contains the call (UTC)			⌘	
time	string `YYYY-MM-DD HH:ii:ss`	Time of the block that contains the call (UTC)	⌘	+		
failed	bool	Failed call or not	=		+	
fail_reason	string `.*` or null	If failed, then the failure description, if not, then `null`	~		+	
type	string (enum)	The call type, one of the following values: `call`, `delegatecall`, `staticcall`, `callcode`, `selfdestruct`, `create`, `synthetic_coinbase`, `create2`	=	+	+	
sender †	string `0x[0-9a-f]{40}`	Sender's address (with 0x)	=		+	
recipient	string `0x[0-9a-f]{40}`	Recipient's address (with 0x)	=		+	
child_call_count	int	Number of child calls	*	+		+
value	numeric string	Call value in wei, hereinafter `numeric string` - is a numeric string passed as a string, because wei-values do not fit into uint64	*≈	+		+
value_usd	float	Call value in USD	*	+		+
transferred	bool	Has ether been transferred? (`false` if `failed`, or if the type of transaction does not change the state, e.g., `staticcall`	=		+	
input_hex †	string `[0-9a-f]*`	Input call data				
output_hex †	string `[0-9a-f]*`	Output call data				

Notes:

- `increased efficiency` method applies if querying `transction_id` column using the `equals` operator
- † — value is `null` if `type` is `synthetic_coinbase`
- Search by fields that contain values in wei (`value`) may be with some inaccuracies
- The default sorting is `transaction_id DESC`
- sorting by `index` respects the tree structure (i.e. "0.2" comes before "0.11") instead of being alphabetical

Example output:

`https://api.blockchair.com/ethereum/calls?q=not,type(synthetic_coinbase)&limit=1`:

```
{
  "data": [
    {
      "block_id": 8792132,
      "transaction_id": 8792132000050,
      "transaction_hash":
"0x9e3a13bfc5313245de7142b7ec13b80123188d9ae4cce797a44b9b426864d1ca",
      "index": "0",
      "depth": 0,
      "date": "2019-10-22",
      "time": "2019-10-22 19:30:03",
      "failed": false,
      "fail_reason": null,
      "type": "call",
      "sender": "0xe475e906b74806c333fbb1b087e523496d8c4cb7",
      "recipient": "0x3143ec5a285adfb248c9e4de934ee735d4b7d734",
      "child_call_count": 0,
      "value": "0",
      "value_usd": 0,
      "transferred": true,
      "input_hex":
"a9059cbb00000000000000000000000023ea8008420c4355570f9915b5fe39dc278540d3000000000
00000000000000000000000000000000000000000000003b9aca00",
      "output_hex":
"0000000000000000000000000000000000000000000000000000000000000001"
    }
  ],
  "context": {
    "code": 200,
    "limit": 1,
    "offset": 0,
    "rows": 1,
    "total_rows": 1422927649,
    "state": 8792138,
    "state_layer_2": 8792127,
    ...
  }
}
```

Request cost formula:

See request costs for infinitables

Explore visualizations on our front-end:

https://blockchair.com/ethereum/calls

`addresses` view

Endpoints:

`https://api.blockchair.com/{:eth_chain}/addresses?{:query}`

Where:

- `{:eth_chain}` can only be: `ethereum` or `ethereum/testnet`
- `{:query}` is the query against the table (how to build a query)

Output:

The `addresses` view contains the list of all addresses and their confirmed balances. Unlike other infinitables (`blocks`, `transactions`, `outputs`) this table isn't live, it's automatically updated **every day** with new data, thus we classify it as a "view". `data` contains an array of database rows. Each row is in the following format:

Column	Type	Description	Q?	S?	A?	C?
address	string `0x[0-9a-zA-Z\-]*`	Ethereum account or contract address				
balance	numeric string	Its balance	*	+		+
nonce	int	Its nonce value	*	+		+
is_contract	boolean	Is it a contract (`true`) or an account (`false`)?	=		+	

Notes:

the default sorting — `balance DESC`

Example outputs:

`https://api.blockchair.com/ethereum/addresses`:

```
{
  "data": [
    {
      "address": "0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2",
      "balance": "6693912559400585982377984",
      "nonce": 1,
      "is_contract": true
    },
    {
      "address": "0x00000000219ab540356cbb839cbe05303d7705fa",
      "balance": "6232610000069000205172736",
      "nonce": 1,
      "is_contract": true
    },
    {
      "address": "0xbe0eb53f46cd790cd13851d5eff43d12404d33e8",
      "balance": "2296896558056344842665984",
      "nonce": 865,
      "is_contract": false
    },
    {
      "address": "0x53d284357ec70ce289d6d64134dfac8e511c8a3d",
      "balance": "1378734066321521433903104",
      "nonce": 4,
      "is_contract": false
    },
    {
      "address": "0x61edcdf5bb737adffe5043706e7c5bb1f1a56eea",
      "balance": "1189498953581339986624512",
      "nonce": 0,
      "is_contract": true
    },
    {
      "address": "0x4ddc2d193948926d02f9b1fe9e1daa0718270ed5",
      "balance": "1146177206209739021615104",
      "nonce": 1,
      "is_contract": true
    },
    {
      "address": "0xdf9eb223bafbe5c5271415c75aecd68c21fe3d7f",
      "balance": "988648154664867412836352",
      "nonce": 1,
      "is_contract": true
    },
    {
      "address": "0xc61b9bb3a7a0767e3179713f3a5c7a9aedce193c",
      "balance": "800010760463680857440256",
      "nonce": 1,
      "is_contract": true
    },
    {
      "address": "0x8484ef722627bf18ca5ae6bcf031c23e6e922b30",
      "balance": "755009999245592554897408",
      "nonce": 1,
      "is_contract": true
    },
    {
      "address": "0x07ee55aa48bb72dcc6e9d78256648910de513eca",
      "balance": "681241111484627083591680",
      "nonce": 0,
      "is_contract": true
    }
  ],
  "context": {
    "code": 200,
    "source": "A",
    "limit": 10,
    "offset": 0,
    "rows": 10,
    "total_rows": 121050742,
    "state": 12787924,
    "state_layer_2": 12787924,
    ...
  }
}
```

`https://api.blockchair.com/ethereum/addresses?q=balance(1000000..)&a=count()` (counts the number of addresses hodling more than 1M ether):

```
{
  "data": [
    {
      "count()": 6
    }
  ],
  "context": {
    "code": 200,
    ...
  }
}
```

`https://api.blockchair.com/ethereum/addresses?a=is_contract,count()` (counts accounts and contracts):

```
{
  "data": [
    {
      "is_contract": false,
      "count()": 103337709
    },
    {
      "is_contract": true,
      "count()": 17713033
    }
  ],
  "context": {
    "code": 200,
    ...
  }
}
```

Request cost formula:

See request costs for infinitables

Explore visualizations on our front-end:

- https://blockchair.com/ethereum/addresses
- https://blockchair.com/ethereum/testnet/addresses

Inifinitable endpoints for Mixin

Please note that our Mixin API outputs raw node data for these endpoints.

`snapshots` table

Note: this particular table doesn't support advanced querying. The only query section it supports are `?offset=` and sorting/filtering by `topology`.

Endpoint:

> `https://api.blockchair.com/{:xin_chain}/raw/snapshots?{:query}`

Where:

> `{:xin_chain}` can be only `mixin`

Where:

> `{:query}` is the query against the table (how to build a query)

Output:

`data` contains an array of database rows.

Example requests:`

- `https://api.blockchair.com/mixin/raw/snapshots`
- `https://api.blockchair.com/mixin/raw/snapshots?q=topology(..18629737)&offset=10`
- `https://api.blockchair.com/mixin/raw/snapshots?s=topology(asc)`

Example output:

`https://api.blockchair.com/mixin/raw/snapshots`:

```
{
  "data": [
    {
      "hash": "a6188df5dfecef1a2650fc7efd51ad0147539182cf0459fee6986b48f83502a6",
      "node": "f7d194a68478987bc472c9f99478260dc12f4860204e0e91bee98a8b89363bc3",
      "references": {
        "self":
"c77df83dcc00afba5e8cbc34b075df975c42efe520b4e00b501289b23f9affc1",
        "external":
"4d5f06d7b8512780396c212ecf55a7bfd7c42b4d82d0bd8e7911a03cab28c8cc"
      },
      "round": 8729,
      "signature":
"652e1d783743c45aebb127a3c9a8d823d743b3dd2304f12a4dc490e104448e61de8fe14abae911528
ab9f7b845b73fc86582e53333e35ee1b78fdcb17b272e0000000000003eade5",
      "timestamp": 1587575473417249500,
      "topology": 18629830,
      "transaction": {
        "asset":
"da5f6dbd3102cd89b1b040c6b61e5f2b696bcb989dff7d8ecee8872aacf65592",
        "extra": "44876fa784bc11eabda9b827eb81dfb7",
        "hash":
"ce122ec544fc41c9cde2d350c544659ee5d4887201becf0a01eed6d238030303",
        "inputs": [
          {
            "hash":
"d3ac83d0cc8ef79bb215e6fc3326d58c6b16d2eb43fd6d6f16c18de4ddb0907a",
            "index": 0
          }
        ],
        "outputs": [
          {
            "amount": "0.01033063",
            "keys": [
              "322c48fa5b19aae518147de7223f62bcb7b444b054226d50fcfd064d0ed555c5"
            ],
            "mask":
"bc561649c4f9a36c252159717cc0deb797f1af1af1704cefd96cb467616e060e",
            "script": "fffe01",
            "type": 0
          }
        ],
        "version": 1
      },
      "version": 1
    },
    {
      "hash": "80f6199ccc5bcb2cfb484a334107a67f89dc6e4cbcbcaae341fe28c619960bd5",
      "node": "f7d194a68478987bc472c9f99478260dc12f4860204e0e91bee98a8b89363bc3",
      "references": {
        "self":
"c77df83dcc00afba5e8cbc34b075df975c42efe520b4e00b501289b23f9affc1",
        "external":
"4d5f06d7b8512780396c212ecf55a7bfd7c42b4d82d0bd8e7911a03cab28c8cc"
      },
      "round": 8729,
      "signature":
"8e18689d15e051bb484ae08fa6b9325d61d75f86cbc203e2fcb87f97f93d5906d91d8cb31036b94a4
3918fc3e007a0e82bb3acb2735d66b5a90566b68bbb130700000000001efdf0",
      "timestamp": 1587575472096503000,
      "topology": 18629829,
      "transaction": {
        "asset":
"d4c304ffc3270ee0f3468913bd8027225201f0eccd336d47062d76c6e2b6bb27",
        "extra": "c5029926c5904a4583094a9e0761c9da",
        "hash":
"a95f88e19cd5dfbb6f14dd6ea581049b065ce0065798faa3cb889995088db9c0",
        "inputs": [
          {
            "hash":
"80dac46fe23abc29d7fe74b6e3580c42e164d37c9bd50be05306ccd2c7e6c653",
            "index": 1
          }
        ],
        "outputs": [
          {
            "amount": "0.01193500",
            "keys": [
              "b8124285ceca9f5e83b2a5f0420c8483067a69719f0741550742f0ac4c38c580"
            ],
            "mask":
"0aef3fc155aa561d75490f545bb044f9a8f488060db7a6f4631d33a6d53296fd",
            "script": "fffe01",
            "type": 0
          },
          {
            "amount": "0.00944393",
            "keys": [
              "f75bfa4afe3584b2beda6998be56c93cf6cd79b5635d40f61e0a6cefdf66367b"
            ],
            "mask":
"d6bb73f16b57f7a67bb0c8bfce11b2f7ab1a1f108b9f7af242e36d448d2406e5",
            "script": "fffe01",
            "type": 0
          }
        ],
        "version": 1
      },
      "version": 1
    },
    ...
  ],
  "context": {
    "code": 200,
    "results": 10,
    "total_rows": 18629831,
    "offset": 0,
    "state": 18629830,
    ...
  }
}
```

Request cost formula:

See request costs for infinitables

Explore visualization on our front-end:

> https://blockchair.com/mixin/snapshots

`mintings` table

Note: this particular table doesn't support advanced querying. The only query section it supports are `?offset=` and sorting/filtering by `batch`.

Endpoint:

> `https://api.blockchair.com/{:xin_chain}/raw/mintings?{:query}`

Where:

> `{:xin_chain}` can be only `mixin`

Where:

> `{:query}` is the query against the table (how to build a query)

Output:

`data` contains an array of database rows.

Example requests:`

- `https://api.blockchair.com/mixin/raw/mintings`
- `https://api.blockchair.com/mixin/raw/mintings?q=batch(..400)&offset=10`
- `https://api.blockchair.com/mixin/raw/mintings?s=batch(asc)`

Example output:

`https://api.blockchair.com/mixin/raw/mintings?s=batch(asc)`:

```
{
  "data": [
    {
      "amount": "1726.02739638",
      "batch": 14,
      "group": "KERNELNODE",
      "transaction":
"20001842d6eff5129c11f7c053bf1209f0267bf223f1681c9cb9d19fc773a692",
      "snapshot": {
        "hash":
"1f408b456fe82b3e47801167649a725cb71075a58bb2568c8fe44bc223a0eece",
        "node":
"307ecfa84d100ecd6bc32743972083e5178e02db049ce16bfd743f3ae52fefc5",
        "references": {
          "self":
"31923e163f5daddcb97ef98bf3b8a76002ec007e309c209ec9a071e16f876d90",
          "external":
"0597b1772ba2a0bd814dba7f9f6010512a426eef3154d41f7e63ff1394db6ce2"
        },
        "round": 1,
        "signatures": [ ... ],
        "timestamp": 1552544417124320500,
        "topology": 116,
        "transaction": {
          "asset":
"a99c2e0e2b1da4d648755ef19bd95139acbbe6564cfb06dec7cd34931ca72cdc",
          "extra": "",
          "hash":
"20001842d6eff5129c11f7c053bf1209f0267bf223f1681c9cb9d19fc773a692",
          "inputs": [
            {
              "mint": {
                "group": "KERNELNODE",
                "batch": 14,
                "amount": "1726.02739638"
              }
            }
          ],
          "outputs": [
            {
              "amount": "115.06849309",
              "keys": [
                "5cd87b6b5a25f67445197261e1ebb5d68be598cd63b0a57eef6897f82cde5c0a"
              ],
              "mask":
"f287afceabccc3d48b52de04d0edd43b446275041b024a3b5c9517894c06f9ab",
              "script": "fffe01",
              "type": 0
            },
            ...
          ],
          "version": 1
        },
        "version": 0
      },
      "timestamp": 1552544417124320500
    },
    ...
  ],
  "context": {
    "code": 200,
    "results": 10,
    "total_rows": 404,
    "offset": 0,
    "state": 18630676,
    ...
  }
}
```

Request cost formula:

See request costs for infinitables

Explore visualization on our front-end:

> https://blockchair.com/mixin/mintings

`nodes` table

Note: this particular table doesn't support querying. It outputs all the entries (so there's no standard limit of 10 rows). Nodes are sorted by their `state`, and then by `timestamp`.

Endpoint:

> `https://api.blockchair.com/{:xin_chain}/raw/nodes`

Where:

> `{:xin_chain}` can be only `mixin`

Output:

`data` contains an array of database rows.

Example requests:`

> `https://api.blockchair.com/mixin/raw/nodes`

Example output:

`https://api.blockchair.com/mixin/raw/nodes`:

```
{
  "data": [
    {
      "id": "cbba7a5e7bae3b0cef3d6dcba7948fa03facda3be401d67aa1a38aecb1f443a0",
      "payee":
"XINCcpcWJbJRiqEoUV7pWrmAdN1AZq3wyYTxa62JojvM4UqpuQnoVX7DZ6BgJEb61pSUS4ZyZNuEbAGL5
azNyZNCbwdgqcVY",
      "signer":
"XIN3ntCzd1FqjSxrYM1f9abN3wY5DcydkDviEVgZL3paV7oYEeKnwzbMLwoRVANwyiu7w9mRrPf2eTpPa
LRgQow9rSr3hzWH",
      "state": "ACCEPTED",
      "timestamp": 1579450099118731000,
      "transaction":
"ebbbf69e9e74e4070ef0685f8d9b4d7bc443922ac93445bc9bda1567984bdda8"
    },
    {
      "id": "6985deee66ead2021925eae21737fa172d19c6efc3e53f3ca5e28ab42f7f51eb",
      "payee":
"XINYDpVHXHxkFRPbP9LZak5p7FZs3mWTeKvrAzo4g9uziTW99t7LrU7me66Xhm6oXGTbYczQLvznk3hxg
NSfNBaZveAmEeRM",
      "signer":
"XINDfgnkijCTe9ijVd9yDwQP8VY4rXwFqYczfgeKJViJqjGKmWS8MdZhJn7kPd5Hv6M8W8RobhJUAxkxg
Z6YNtdWQwefYE51",
      "state": "ACCEPTED",
      "timestamp": 1583004182403037400,
      "transaction":
"48f3d7b5ae6b03f251705cfc82c3b3c7413ec8a7e7b100de0cab4d8f3ec33bd5"
    },
    ...
  ],
  "context": {
    "code": 200,
    "results": 55,
    "state": 18630827,
    ...
  }
}
```

Request cost formula:

See request costs for infinitables

Explore visualization on our front-end:

https://blockchair.com/mixin/nodes

Inifinitable endpoints for Tezos

Please note that our Tezos API outputs raw node data for this endpoint.

`blocks` table

Note: this particular table doesn't support advanced querying. The only query section it supports are `?offset=` and sorting/filtering by `id` (height).

Endpoint:

> `https://api.blockchair.com/{:xtz_chain}/raw/blocks?{:query}`

Where:

> `{:xtz_chain}` can be only `tezos`

Where:

> `{:query}` is the query against the table (how to build a query)

Output:

`data` contains an array of database rows.

Example requests:`

- `https://api.blockchair.com/tezos/raw/blocks`
- `https://api.blockchair.com/tezos/raw/blocks?q=id(..100000)&offset=10`
- `https://api.blockchair.com/tezos/raw/blocks?s=id(asc)`

Example output:

`https://api.blockchair.com/tezos/raw/blocks?s=id(asc)`:

```
{
  "data": [
    {
      "id": 0,
      "time": "2018-06-30T16:07:32Z",
      "hash": "BLockGenesisGenesisGenesisGenesisGenesisf79b5d1CoW2",
      "priority": 0,
      "n_ops": 0,
      "volume": 0,
      "cycle": 0,
      "is_cycle_snapshot": 1,
      "version": 0,
      "n_accounts": 0,
      "n_new_accounts": 0,
      "n_new_contracts": 0,
      "gas_limit": 0,
      "gas_used": 0,
      "gas_price": 0,
      "days_destroyed": 0
    },
    {
      "id": 1,
      "time": "2018-06-30T17:39:57Z",
      "hash": "BLSqrcLvFtqVCx8WSqkVJypW2kAVRM3eEj2BHgBsB6kb24NqYev",
      "priority": 0,
      "n_ops": 0,
      "volume": 0,
      "cycle": 0,
      "is_cycle_snapshot": 0,
      "version": 1,
      "n_accounts": 31589,
      "n_new_accounts": 31589,
      "n_new_contracts": 32,
      "gas_limit": 0,
      "gas_used": 0,
      "gas_price": 0,
      "days_destroyed": 0
    },
    ...
  ],
  "context": {
    "code": 200,
    "results": 10,
    "total_rows": 1002667,
    "offset": 0,
    "state": 1002666,
    "price_usd": 2.67,
    ...
  }
}
```

Request cost formula:

See request costs for infinitables

Explore visualization on our front-end:

> https://blockchair.com/tezos/blocks

Inifinitable endpoints for second layers

`properties` table (Omni Layer)

Note: this particular table doesn't support querying. The only query section it supports is `?offset=`. Note that this endpoint is in the Alpha stage.

Endpoint:

> `https://api.blockchair.com/bitcoin/omni/properties?{:query}`

Where:

> `{:query}` is the query against the table (how to build a query), the only supported query section for this table is `?offset=`

Output:

`data` contains an array of database rows. Each row is in the format which accords with Omni Layer specification (https://github.com/OmniLayer/spec)

Example output:

`https://api.blockchair.com/bitcoin/omni/properties`:

```
{
  "data": [
    {
      "id": 412,
      "name": "ENO",
      "category": "",
      "subcategory": "",
      "description": "",
      "url": "",
      "is_divisible": false,
      "issuer": "1JcfUyi9BkXCTXHdeUusmYrsHXvnnLvTxB",
      "creation_transaction_hash":
"ea5b914ba4e80931c8d46e551f6010113ab2cba82186d2497f2b2f0c6d53953b",
      "creation_time": "2018-11-25 21:34:08",
      "creation_block_id": 551501,
      "is_issuance_fixed": false,
      "is_issuance_managed": false,
      "circulation": 222222222,
      "ecosystem": 1
    },
    ...
  ],
  "context": {
    "code": 200,
    "limit": 10,
    "offset": 0,
    "rows": 10,
    "total_rows": 412,
    "state": 599976,
    ...
  }
}
```

Request cost formula:

See request costs for infinitables

Explore visualization on our front-end:

https://blockchair.com/bitcoin/omni/properties

`tokens` table (ERC-20)

Endpoint:

- `https://api.blockchair.com/ethereum/erc-20/tokens?{:query}`
- `https://api.blockchair.com/ethereum/testnet/erc-20/tokens?{:query}` (Goerli Testnet)

Where:

`{:query}` is the query against the table (how to build a query)

Output:

Returns information about ERC-20 tokens indexed by our engine. `data` contains an array of database rows. Each row is in the following format:

Column	Type	Description	Q?	S?	A?	C?
address	string `0x[0-9a-f]{40}`	Address of the token contract	=			
id	int	Internal Blockchair id of the token	*	+		
date	string `YYYY-MM-DD`	Creation date			⌘	
time	string `YYYY-MM-DD HH:ii:ss`	Creation timestamp	⌘	+		
name	string `.*` (or an empty string)	Token name (e.g. `My New Token`)	=	+		
symbol	string `.*` (or an empty string)	Token symbol (e.g. `MNT`)	=	+		
decimals	int	Number of decimals	=	+		
creating_block_id	int	Creating block height	*	+		
creating_transaction_hash	string `0x[0-9a-f]{64}`	Creating transaction hash				

Notes:

- for the columns `address`, `id` increased efficiency when uploading one record is applied
- there is no possibility to search over `date` column, use searching `?q=time(YYYY-MM-DD)` instead
- the default sort is `id DESC`
- when using `offset`, it is reasonable to add to the filters the maximum block number (`?q=block_id(..N)`), since it is very likely that during the iteration new rows will be added to the table. For convenience, you can take the value of `context.state` from the first result of any query containing the number of the latest block at the query time and use this result later on.

Example output:

`https://api.blockchair.com/ethereum/erc-20/tokens?limit=1`:

```
{
  "data": [
    {
      "address": "0x9b460d404be254d7b2ba89336a8a41807bb1562b",
      "id": 121500,
      "date": "2019-10-22",
      "time": "2019-10-22 19:21:11",
      "name": "UGB Token",
      "symbol": "UGB",
      "decimals": 18,
      "creating_block_id": 8792093,
      "creating_transaction_hash":
"0x58e132a937c3bd60f1d113ecb14db59fd5229ae312a2afdf8f1b365bf8620e5e"
    }
  ],
  "context": {
    "code": 200,
    "limit": 1,
    "offset": 0,
    "rows": 1,
    "total_rows": 121500,
    "state": 8792147,
    "state_layer_2": 8792137,
    ...
  }
}
```

`https://api.blockchair.com/ethereum/erc-20/tokens?q=symbol(USDT)&a=count()`:

```
{
  "data": [
    {
      "count()": 72
    }
  ],
  "context": {
    "code": 200,
    "limit": 10000,
    "offset": null,
    "rows": 1,
    "total_rows": 1,
    "state": 8792205,
    "state_layer_2": 8792192,
    ...
  }
}
```

Request cost formula:

See request costs for infinitables

Explore visualization on our front-end:

https://blockchair.com/ethereum/erc-20/tokens

`transactions` table (ERC-20)

Endpoint:

- `https://api.blockchair.com/ethereum/erc-20/transactions?{:query}`
- `https://api.blockchair.com/ethereum/testnet/erc-20/transactions?{:query}` (Goerli Testnet)

Where:

`{:query}` is the query against the table (how to build a query)

Output:

Returns information about ERC-20 transfers indexed by our engine. `data` contains an array of database rows. Each row is in the following format:

Column	Type	Description	Q?	S?	A?	C?
block_id	int	Block id including the token transfer	*	+		
id	int	Internal Blockchair id of the token transfer	*	+		
transaction_hash	string `0x[0-9a-f]{64}`	Transaction hash including the token transfer				
date	string `YYYY-MM-DD`	Date of the transfer			⌘	
time	string `YYYY-MM-DD HH:ii:ss`	Timestamp of the transfer	⌘	+		
token_address	string `0x[0-9a-f]{40}`	Address of the token contract	=		+	
token_name	string `.*` (or an empty string)	Token name (e.g. `My New Token`)	=	+	+	
token_symbol	string `.*` (or an empty string)	Token symbol (e.g. `MNT`)	=	+	+	
token_decimals	int	Number of decimals	=	+		
sender	string `0x[0-9a-f]{40}`	The sender's address	=			
recipient	string `0x[0-9a-f]{40}`	The recipient's address	=			
value	numeric string	Transferred amount (in the smallest denomination)	*≈	=		

Notes:

- for the columns `id` increased efficiency when uploading one record is applied
- there is no possibility to search over `date` column, use searching `?q=time(YYYY-MM-DD)` instead
- the default sort is `id DESC`
- when using `offset`, it is reasonable to add to the filters the maximum block number (`?q=block_id(..N)`), since it is very likely that during the iteration new rows will be added to the table. For convenience, you can take the value of `context.state` from the first result of any query containing the number of the latest block at the query time and use this result later on.
- value is approximated when queried

Example output:

`https://api.blockchair.com/ethereum/erc-20/transactions?limit=1`:

```
{
  "data": [
    {
      "block_id": 8792197,
      "id": 275501753,
      "transaction_hash":
"0xec32c9b67d3e7088f14bfc17e8ccb0eb06a98eebe81224dc8703f470c62c5a2e",
      "date": "2019-10-22",
      "time": "2019-10-22 19:45:41",
      "token_address": "0xbe59434473c50021b30686b6d34cdd0b1b4f6198",
      "token_name": "Mobilio",
      "token_symbol": "MOB",
      "token_decimals": 18,
      "sender": "0x2a68bdc41e98ab0fb60c9610e62d83ab29312d06",
      "recipient": "0xfa96009f004428b85a05cfa1233c24f7afe0536a",
      "value": "12021696603378832398951"
    }
  ],
  "context": {
    "code": 200,
    "limit": 1,
    "offset": 0,
    "rows": 1,
    "total_rows": 275501753,
    "state": 8792207,
    "state_layer_2": 8792197,
    ...
  }
}
```

Request cost formula:

See request costs for infinitables

Explore visualization on our front-end:

https://blockchair.com/ethereum/erc-20/transactions