

Node.js Driver API reference

Connection

TypeDB

Namespace variables

Name

`DEFAULT_ADDRESS`

coreDriver

`coreDriver(address?): Promise<TypeDBDriver>`

Creates a connection to TypeDB.

Input parameters

Name	Description	Type
<code>address</code>	Address of the TypeDB server. Examples <code>const driver = TypeDB.coreDriver("127.0.0.1:11729");</code> Copy	<code>string = DEFAULT_ADDRESS</code>

Returns

`Promise<TypeDBDriver>`

enterpriseDriver

```
enterpriseDriver(addresses, credential): Promise<TypeDBDriver>
```

Creates a connection to TypeDB Enterprise, authenticating with the provided credentials.

Input parameters

Name	Description	Type
<code>addresses</code>	List of addresses of the individual TypeDB Enterprise servers. As long one specified address is provided, the driver will discover the other addresses from that server.	<code>string</code> <code>string[]</code>
<code>credential</code>	The credentials to log in, and encryption settings. See <code>TypeDBCredential</code> Examples <pre>const driver = TypeDB.enterpriseDriver(["127.0.0.1:11729"], new TypeDBCredential(username, password));</pre> Copy	<code>TypeDBCredential</code>

Returns

```
Promise<TypeDBDriver>
```

TypeDBDriver

Fields

Name	Type	Description
<code>databases</code>	<code>DatabaseManager</code>	The DatabaseManager for this connection, providing access to database management methods.

Name	Type	Description
<code>users</code>	<code>UserManager</code>	The UserManager instance for this connection, providing access to user management methods. Only for TypeDB Enterprise.

close

```
close(): Promise<void>
```

Closes the driver. Before instantiating a new driver, the driver that's currently open should first be closed.

Returns

```
Promise<void>
```

Code examples

```
driver.close()
```

isOpen

```
isOpen(): boolean
```

Checks whether this connection is presently open.

Returns

```
boolean
```

Code examples

```
driver.isOpen()
```

session

```
session(database, type, options?): Promise<TypeDBSession>
```

Input parameters

Name	Description	Type
<code>database</code>		<code>string</code>
<code>type</code>		<code>SessionType</code>
<code>options</code>		<code>TypeDBOptions</code>

Returns

`Promise<TypeDBSession>`

user

```
user(): Promise<User>
```

Returns the logged-in user for the connection. Only for TypeDB Enterprise.

Returns

`Promise<User>`

Code examples

```
driver.user()
```

TypeDBCredential

User credentials and TLS encryption settings for connecting to TypeDB Enterprise.

password

```
get password(): string
```

Returns

`string`

tlsRootCAPath

```
get tlsRootCAPath(): string
```

Returns

```
string
```

username

```
get username(): string
```

Returns

```
string
```

new TypeDBCredential

```
new TypeDBCredential(username, password, tlsRootCAPath?): TypeDBCredential
```

Input parameters

Name	Description	Type
<code>username</code>	The name of the user to connect as	<code>string</code>
<code>password</code>	The password for the user	<code>string</code>
<code>tlsRootCAPath</code>	Path to the CA certificate to use for authenticating server certificates.	<code>string</code>

Returns

```
TypeDBCredential
```

DatabaseManager

Provides access to all database management methods.

all

```
all(): Promise<Database[]>
```

Retrieves all databases present on the TypeDB server

Returns

```
Promise<Database[]>
```

Code examples

```
driver.databases().all()
```

contains

```
contains(name): Promise<boolean>
```

Checks if a database with the given name exists

Input parameters

Name	Description	Type
<code>name</code>	The database name to be checked	<code>string</code>

Returns

```
Promise<boolean>
```

Code examples

```
driver.databases().contains(name)
```

create

```
create(name): Promise<void>
```

Create a database with the given name

Input parameters

Name	Description	Type
<code>name</code>	The name of the database to be created	<code>string</code>

Returns

`Promise<void>`

Code examples

```
driver.databases().create(name)
```

get

```
get(name): Promise<Database>
```

Retrieve the database with the given name.

Input parameters

Name	Description	Type
<code>name</code>	The name of the database to retrieve	<code>string</code>

Returns

`Promise<Database>`

Code examples

```
driver.databases().get(name)
```

Database

Fields

Name	Type	Description
------	------	-------------

Name	Type	Description
<code>name</code>	<code>string</code>	The database name as a string.
<code>preferredReplica</code>	<code>Replica</code>	The preferred replica for this database. Operations which can be run on any replica will prefer to use this replica. Only works in TypeDB Enterprise
<code>primaryReplica</code>	<code>Replica</code>	The primary replica for this database. Only works in TypeDB Enterprise
<code>replicas</code>	<code>Replica</code>	The Replica instances for this database. Only works in TypeDB Enterprise

delete

```
delete(): Promise<void>
```

Deletes this database.

Returns

```
Promise<void>
```

Code examples

```
database.delete()
```

schema

```
schema(): Promise<string>
```

Returns a full schema text as a valid TypeQL define query string.

Returns

```
Promise<string>
```


Code examples

```
database.schema()
```

Replica

The metadata and state of an individual raft replica of a database.

Fields

Name	Type	Description
<code>address</code>	<code>string</code>	The address of the server hosting this replica
<code>databaseName</code>	<code>string</code>	The database for which this is a replica.
<code>preferred</code>	<code>boolean</code>	Checks whether this is the preferred replica of the raft cluster. If true, Operations which can be run on any replica will prefer to use this replica.
<code>primary</code>	<code>boolean</code>	Checks whether this is the primary replica of the raft cluster.
<code>term</code>	<code>number</code>	The raft protocol 'term' of this replica.

UserManager

Provides access to all user management methods.

all

```
all(): Promise<User[]>
```

Retrieves all users which exist on the TypeDB server.

Returns

```
Promise<User[]>
```

Code examples

```
driver.users.all()
```

contains

```
contains(name): Promise<boolean>
```

Checks if a user with the given name exists.

Input parameters

Name	Description	Type
<code>name</code>		<code>string</code>

Returns

```
Promise<boolean>
```

Code examples

```
driver.users.contains(username)
```

create

```
create(name, password): Promise<void>
```

Create a user with the given name & password.

Input parameters

Name	Description	Type
<code>name</code>		<code>string</code>
<code>password</code>	The password of the user to be created	<code>string</code>

Returns

Promise<void>

Code examples

```
driver.users.create(username, password)
```

delete

```
delete(name): Promise<void>
```

Deletes a user with the given name.

Input parameters

Name	Description	Type
<code>name</code>		<code>string</code>

Returns

Promise<void>

Code examples

```
driver.users.delete(username)
```

get

```
get(name): Promise<User>
```

Retrieve a user with the given name.

Input parameters

Name	Description	Type
<code>name</code>		<code>string</code>

Returns

Promise<User>

Code examples

```
driver.users.get(username)
```

passwordSet

```
passwordSet(name, password): Promise<void>
```

Sets a new password for a user. This operation can only be performed by administrators.

Input parameters

Name	Description	Type
<code>name</code>		<code>string</code>
<code>password</code>	The new password	<code>string</code>

Returns

Promise<void>

Code examples

```
driver.users.passwordSet(username, password)
```

User

`User` class

Fields

Name	Type	Description
<code>passwordExpirySeconds</code>	<code>number</code>	The number of seconds remaining till this user's current password expires.

Name	Type	Description
<code>username</code>	<code>string</code>	The name of this user.

passwordUpdate

```
passwordUpdate(oldPassword, newPassword): Promise<void>
```

Updates the user's password.

Input parameters

Name	Description	Type
<code>oldPassword</code>	Old password	<code>string</code>
<code>newPassword</code>	New password	<code>string</code>

Returns

`Promise<void>`

Code examples

```
user.passwordUpdate("oldpassword", "nEwp@ssw0rd");
```

Session

TypeDBSession

Fields

Name	Type	Description
<code>database</code>	<code>Database</code>	The database of the session. Examples session.database() Copy

Name	Type	Description
<code>options</code>	<code>TypeDBOptions</code>	Gets the options for the session
<code>type</code>	<code>SessionType</code>	The current session's type (SCHEMA or DATA)

close

```
close(): Promise<void>
```

Closes the session. Before opening a new session, the session currently open should first be closed.

Returns

```
Promise<void>
```

Code examples

```
session.close()
```

isOpen

```
isOpen(): boolean
```

Checks whether this session is open.

Returns

```
boolean
```

Code examples

```
session.isOpen()
```

transaction

```
transaction(type, options?): Promise<TypeDBTransaction>
```

Opens a transaction to perform read or write queries on the database connected to the session.

Input parameters

Name	Description	Type
<code>type</code>		<code>TransactionType</code>
<code>options</code>	Options for the session	<code>TypeDBOptions</code>

Returns

`Promise<TypeDBTransaction>`

Code examples

```
session.transaction(transactionType, options)
```

SessionType

This class is used to specify the type of the session.

Namespace variables

Name
<code>DATA</code>
<code>SCHEMA</code>

isData

```
isData(): boolean
```

Returns

`boolean`

isSchema

```
isSchema(): boolean
```

Returns

```
boolean
```

TypeDBOptions

TypeDB session and transaction options. `TypeDBOptions` object can be used to override the default server behaviour. Options could be specified either as constructor arguments or using setters.

explain

```
get explain(): boolean
```

If set to `True`, enables explanations for queries. Only affects read transactions.

Returns

```
boolean
```

explain

```
set explain(value): void
```

If set to `True`, enables explanations for queries. Only affects read transactions.

Returns

```
void
```

infer

```
get infer(): boolean
```

If set to `True`, enables inference for queries. Only settable at transaction level and above. Only affects read transactions.

Returns

```
boolean
```


infer

```
set infer(value): void
```

If set to `True`, enables inference for queries. Only settable at transaction level and above. Only affects read transactions.

Returns

```
void
```

parallel

```
get parallel(): boolean
```

If set to `True`, the server uses parallel instead of single-threaded execution.

Returns

```
boolean
```

parallel

```
set parallel(value): void
```

If set to `True`, the server uses parallel instead of single-threaded execution.

Returns

```
void
```

prefetch

```
get prefetch(): boolean
```

If set to `True`, the first batch of answers is streamed to the driver even without an explicit request for it.

Returns

```
boolean
```

prefetch

```
set prefetch(value): void
```

If set to `True`, the first batch of answers is streamed to the driver even without an explicit request for it.

Returns

`void`

prefetchSize

```
get prefetchSize(): number
```

If set, specifies a guideline number of answers that the server should send before the driver issues a fresh request.

Returns

`number`

prefetchSize

```
set prefetchSize(value): void
```

If set, specifies a guideline number of answers that the server should send before the driver issues a fresh request.

Returns

`void`

readAnyReplica

```
get readAnyReplica(): boolean
```

If set to `True`, enables reading data from any replica, potentially boosting read throughput. Only settable in TypeDB Enterprise.

Returns

`boolean`

readAnyReplica

```
set readAnyReplica(value): void
```

If set to `True`, enables reading data from any replica, potentially boosting read throughput. Only settable in TypeDB Enterprise.

Returns

```
void
```

schemaLockAcquireTimeoutMillis

```
get schemaLockAcquireTimeoutMillis(): number
```

If set, specifies how long the driver should wait if opening a session or transaction is blocked by a schema write lock.

Returns

```
number
```

schemaLockAcquireTimeoutMillis

```
set schemaLockAcquireTimeoutMillis(value): void
```

If set, specifies how long the driver should wait if opening a session or transaction is blocked by a schema write lock.

Returns

```
void
```

sessionIdleTimeoutMillis

```
get sessionIdleTimeoutMillis(): number
```

If set, specifies a timeout that allows the server to close sessions if the driver terminates or becomes unresponsive.

Returns

```
number
```

sessionIdleTimeoutMillis

```
set sessionIdleTimeoutMillis(millis): void
```

If set, specifies a timeout that allows the server to close sessions if the driver terminates or becomes unresponsive.

Returns

`void`

traceInference

```
get traceInference(): boolean
```

If set to `True`, reasoning tracing graphs are output in the logging directory. Should be used with `parallel = False`.

Returns

`boolean`

traceInference

```
set traceInference(value): void
```

If set to `True`, reasoning tracing graphs are output in the logging directory. Should be used with `parallel = False`.

Returns

`void`

transactionTimeoutMillis

```
get transactionTimeoutMillis(): number
```

If set, specifies a timeout for killing transactions automatically, preventing memory leaks in unclosed transactions.

Returns

number

transactionTimeoutMillis

```
set transactionTimeoutMillis(millis): void
```

If set, specifies a timeout for killing transactions automatically, preventing memory leaks in unclosed transactions.

Returns

void

new TypeDBOptions

```
new TypeDBOptions(obj?): TypeDBOptions
```

Input parameters

Name	Description	Type
obj	If set to <code>True</code> , enables explanations for queries. Only affects read transactions.	<pre>{ explain?: boolean; infer?: boolean; parallel?: boolean; prefetch?: boolean; prefetchSize?: number; readAnyReplica?: boolean; schemaLockAcquireTimeoutMillis?: number; sessionIdleTimeoutMillis?: number; traceInference?: boolean; transactionTimeoutMillis?: number; } = {}</pre>

Returns

TypeDBOptions

Opts

Interface for TypeDBOptions. Use TypeDBOptions instead.

Fields

Name	Type	Description
------	------	-------------

Name	Type	Description
<code>explain</code>	<code>boolean</code>	If set to True, enables explanations for queries. Only affects read transactions.
<code>infer</code>	<code>boolean</code>	If set to True, enables inference for queries. Only settable at transaction level and above. Only affects read transactions.
<code>parallel</code>	<code>boolean</code>	If set to True, the server uses parallel instead of single-threaded execution.
<code>prefetch</code>	<code>boolean</code>	If set to True, the first batch of answers is streamed to the driver even without an explicit request for it.
<code>prefetchSize</code>	<code>number</code>	If set, specifies a guideline number of answers that the server should send before the driver issues a fresh request.
<code>readAnyReplica</code>	<code>boolean</code>	If set to True, enables reading data from any replica, potentially boosting read throughput. Only settable in TypeDB Enterprise.

Name	Type	Description
<code>schemaLockAcquireTimeoutMillis</code>	<code>number</code>	If set, specifies how long the driver should wait if opening a session or transaction is blocked by a schema write lock.
<code>sessionIdleTimeoutMillis</code>	<code>number</code>	If set, specifies a timeout that allows the server to close sessions if the driver terminates or becomes unresponsive.
<code>traceInference</code>	<code>boolean</code>	If set to True, reasoning tracing graphs are output in the logging directory. Should be used with <code>parallel = False</code> .
<code>transactionTimeoutMillis</code>	<code>number</code>	If set, specifies a timeout for killing transactions automatically, preventing memory leaks in unclosed transactions.

Transaction

TypeDBTransaction

Fields

Name	Type	Description
------	------	-------------

Name	Type	Description
<code>concepts</code>	<code>ConceptManager</code>	The ConceptManager for this transaction, providing access to all Concept API methods.
<code>logic</code>	<code>LogicManager</code>	The LogicManager for this Transaction, providing access to all Concept API - Logic methods.
<code>options</code>	<code>TypeDBOptions</code>	The options for the transaction.
<code>query</code>	<code>QueryManager</code>	TheQueryManager for this Transaction, from which any TypeQL query can be executed.
<code>type</code>	<code>TransactionType</code>	The transaction's type (READ or WRITE)

close

```
close(): Promise<void>
```

Closes the transaction.

Returns

```
Promise<void>
```

Code examples

```
transaction.close()
```

commit

```
commit(): Promise<void>
```

Commits the changes made via this transaction to the TypeDB database. Whether or not the transaction is committed successfully, it gets closed after the commit call.

Returns

`Promise<void>`

Code examples

```
transaction.commit()
```

isOpen

```
isOpen(): boolean
```

Checks whether this transaction is open.

Returns

`boolean`

Code examples

```
transaction.isOpen()
```

rollback

```
rollback(): Promise<void>
```

Rolls back the uncommitted changes made via this transaction.

Returns

`Promise<void>`

Code examples

```
transaction.rollback()
```

TransactionType

This class is used to specify the type of transaction.

Namespace variables

Name

Name

READ

WRITE

isRead

```
isRead(): boolean
```

Checks whether this is the READ TransactionType

Returns

`boolean`

isWrite

```
isWrite(): boolean
```

Checks whether this is the WRITE TransactionType

Returns

`boolean`

QueryManager

Provides methods for executing TypeQL queries in the transaction.

define

```
define(query, options?): Promise<void>
```

Performs a TypeQL Define query in the transaction.

Input parameters

Name	Description	Type
------	-------------	------

Name	Description	Type
<code>query</code>	The TypeQL Define query to be executed	<code>string</code>
<code>options</code>	Specify query options	<code>TypeDBOptions</code>

Returns

`Promise<void>`

Code examples

```
transaction.query.define(query, options)
```

delete

```
delete(query, options?): Promise<void>
```

Performs a TypeQL Delete query in the transaction.

Input parameters

Name	Description	Type
<code>query</code>	The TypeQL Delete query to be executed	<code>string</code>
<code>options</code>	Specify query options	<code>TypeDBOptions</code>

Returns

`Promise<void>`

Code examples

```
transaction.query.delete(query, options)
```

explain

```
explain(explainable, options?): Stream<Explanation>
```

Performs a TypeQL Explain query in the transaction.

Input parameters

Name	Description	Type
<code>explainable</code>	The Explainable to be explained	<code>Explainable</code>
<code>options</code>	Specify query options	<code>TypeDBOptions</code>

Returns

```
Stream<Explanation>
```

Code examples

```
transaction.query.explain(explainable, options)
```

insert

```
insert(query, options?): Stream<ConceptMap>
```

Performs a TypeQL Insert query in the transaction.

Input parameters

Name	Description	Type
<code>query</code>	The TypeQL Insert query to be executed	<code>string</code>
<code>options</code>	Specify query options	<code>TypeDBOptions</code>

Returns

```
Stream<ConceptMap>
```

Code examples

```
transaction.query.insert(query, options)
```

match

```
match(query, options?): Stream<ConceptMap>
```

Performs a TypeQL Match (Get) query in the transaction.

Input parameters

Name	Description	Type
<code>query</code>	The TypeQL Match (Get) query to be executed	<code>string</code>
<code>options</code>	Specify query options	<code>TypeDBOptions</code>

Returns

```
Stream<ConceptMap>
```

Code examples

```
transaction.query.match(query, options)
```

matchAggregate

```
matchAggregate(query, options?): Promise<Numeric>
```

Performs a TypeQL Match Aggregate query in the transaction.

Input parameters

Name	Description	Type
<code>query</code>	The TypeQL Match Aggregate query to be executed	<code>string</code>

Name	Description	Type
<code>options</code>	Specify query options	<code>TypeDBOptions</code>

Returns

`Promise<Numeric>`

Code examples

```
transaction.query.matchAggregate(query, options)
```

matchGroup

```
matchGroup(query, options?): Stream<ConceptMapGroup>
```

Performs a TypeQL Match Group query in the transaction.

Input parameters

Name	Description	Type
<code>query</code>	The TypeQL Match Group query to be executed	<code>string</code>
<code>options</code>	Specify query options	<code>TypeDBOptions</code>

Returns

`Stream<ConceptMapGroup>`

Code examples

```
transaction.query.matchGroup(query, options)
```

matchGroupAggregate

```
matchGroupAggregate(query, options?): Stream<NumericGroup>
```

Performs a TypeQL Match Group Aggregate query in the transaction.

Input parameters

Name	Description	Type
<code>query</code>	The TypeQL Match Group Aggregate query to be executed	<code>string</code>
<code>options</code>	Specify query options	<code>TypeDBOptions</code>

Returns

`Stream<NumericGroup>`

Code examples

```
transaction.query.matchGroupAggregate(query, options)
```

undefine

```
undefine(query, options?): Promise<void>
```

Performs a TypeQL Undefine query in the transaction.

Input parameters

Name	Description	Type
<code>query</code>	The TypeQL Undefine query to be executed	<code>string</code>
<code>options</code>	Specify query options	<code>TypeDBOptions</code>

Returns

`Promise<void>`

Code examples

```
transaction.query.undefine(query, options)
```

update

```
update(query, options?): Stream<ConceptMap>
```

Performs a TypeQL Update query in the transaction.

Input parameters

Name	Description	Type
<code>query</code>	The TypeQL Update query to be executed	<code>string</code>
<code>options</code>	Specify query options	<code>TypeDBOptions</code>

Returns

```
Stream<ConceptMap>
```

Code examples

```
transaction.query.update(query, options)
```

Answer

ConceptMapGroup

Contains an element of the group query result.

Fields

Name	Type	Description
<code>conceptMaps</code>	<code>ConceptMap</code>	The ConceptMaps of the group.
<code>owner</code>	<code>Concept</code>	The concept that is the group owner.

ConceptMap

Contains a mapping of variables to concepts.

Fields

Name	Type	Description
<code>explainables</code>	<code>Explainables</code>	The Explainables object for this ConceptMap, exposing which of the concepts in this ConceptMap are explainable.

concepts

```
concepts(): IterableIterator<Concept>
```

Produces an iterator over all concepts in this `ConceptMap`.

Returns

```
IterableIterator<Concept>
```

Code examples

```
conceptMap.concepts()
```

get

```
get(variable): Concept
```

Retrieves a concept for a given variable name.

Input parameters

Name	Description	Type
<code>variable</code>	The string representation of a variable	<code>string</code>

Returns

```
Concept
```

Code examples

```
conceptMap.get(variable)
```

toJSONRecord

```
toJSONRecord(): Record<string, Record<string, string | number | boolean>>
```

Retrieves this `ConceptMap` as JSON.

Returns

```
Record<string, Record<string, string | number | boolean>>
```

Code examples

```
conceptMap.toJSONRecord()
```

variables

```
variables(): IterableIterator<string>
```

Produces an iterator over all variables in this `ConceptMap`.

Returns

```
IterableIterator<string>
```

Code examples

```
conceptMap.variables()
```

Stream<T>

A stream of elements offering a functional interface to manipulate elements. Typically the elements are generated/retrieved lazily.

[asyncIterator]

```
[asyncIterator](): AsyncIterator<T, any, undefined>
```

Returns

```
AsyncIterator<T, any, undefined>
```

collect

```
collect(): Promise<T[]>
```

Collects all the answers from this stream into an array

Returns

```
Promise<T[]>
```

Code examples

```
results = transaction.query.get(query).collect().await;
```

every

```
every(callbackFn): Promise<boolean>
```

Checks whether a condition is satisfied by ALL answers in this stream.

Input parameters

Name	Description	Type
<code>callbackFn</code>		<code>((value) => unknown)</code>

Returns

```
Promise<boolean>
```

filter

```
filter(filter): Stream<T>
```

Returns a new stream from this stream consisting only of elements which satisfy a given condition.

Input parameters

Name	Description	Type
<code>filter</code>	The condition to evaluate. Examples <code>// For a query "match \$p isa person, has age \$a; get;", only retrieve results having \$a >= 60.results = transaction.query.match(query).filter(cm => cm.get("a").value > 60).collect();</code> Copy	<code>((value) => boolean)</code>

Returns

`Stream<T>`

first

```
first(): Promise<T>
```

Returns the first element in the stream.

Returns

`Promise<T>`

flatMap

```
flatMap<U>(mapper): Stream<U>
```

Given a function which accepts a single element of this stream and returns a new stream, This function returns a new stream obtained by applying the function to each element in the stream, and concatenating each result thus obtained

Input parameters

Name	Description	Type
<code>mapper</code>	The mapping function to apply. Must return a stream.	<code>((value) => Stream<U>)</code>

Returns

Stream<U>

forEach

```
forEach(fn): Promise<void>
```

Executes the given function for each element in the stream.

Input parameters

Name	Description	Type
<code>fn</code>	The function to evaluate for each element.	<code>((value) ⇒ void)</code>

Returns

`Promise<void>`

iterator

```
iterator(): AsyncIterator<T, any, undefined>
```

Returns

`AsyncIterator<T, any, undefined>`

map

```
map<U>(mapper): Stream<U>
```

Input parameters

Name	Description	Type
<code>mapper</code>	The mapping function to apply. Returns a new stream from this stream by applying the <code>mapper</code> function to each element.	<code>((value) ⇒ U)</code>

Returns

Stream<U>

new Stream

```
new Stream<T>(): Stream<T>
```

Returns

Stream<T>

some

```
some(callbackFn): Promise<boolean>
```

Checks whether a condition is satisfied by ANY answer in this stream.

Input parameters

Name	Description	Type
callbackFn		((value) ⇒ unknown)

Returns

Promise<boolean>

NumericGroup

Contains an element of the group aggregate query result.

Fields

Name	Type	Description
numeric	Numeric	Retrieves the Numeric answer of the group. Examples numericGroup.numeric Copy
owner	Concept	Retrieves the concept that is the group owner. Examples numericGroup.owner Copy

Numeric

Stores an aggregate query answer.

asNumber

```
asNumber(): number
```

Retrieves numeric value of an aggregate answer as a number.

Returns

`number`

Code examples

```
numeric.asNumber()
```

isNaN

```
isNaN(): boolean
```

Checks if the aggregate answer is not a number.

Returns

`boolean`

Code examples

```
numeric.isNan()
```

isNumber

```
isNumber(): boolean
```

Checks if the type of an aggregate answer is a number.

Returns

`boolean`

Code examples

```
numeric.isNumber()
```

Explainables

Contains explainable objects.

Fields

Name	Type	Description
<code>attributes</code>	<code>Map</code>	All of this ConceptMap's explainable attributes.
<code>ownerships</code>	<code>Map</code>	All of this ConceptMap's explainable ownerships.
<code>relations</code>	<code>Map</code>	All of this ConceptMap's explainable relations.

attribute

```
attribute(variable): Explainable
```

Retrieves the explainable attribute with the given variable name.

Input parameters

Name	Description	Type
<code>variable</code>	The string representation of a variable	<code>string</code>

Returns

`Explainable`

Code examples

```
conceptMap.explainables.attribute(variable)
```

ownership


```
ownership(owner, attribute): Explainable
```

Retrieves the explainable attribute ownership with the pair of (owner, attribute) variable names.

Input parameters

Name	Description	Type
<code>owner</code>	The string representation of the owner variable	<code>string</code>
<code>attribute</code>	The string representation of the attribute variable	<code>string</code>

Returns

```
Explainable
```

Code examples

```
conceptMap.explainables.ownership(owner, attribute)
```

relation

```
relation(variable): Explainable
```

Retrieves the explainable relation with the given variable name.

Input parameters

Name	Description	Type
<code>variable</code>	The string representation of a variable	<code>string</code>

Returns

```
Explainable
```

Code examples

```
conceptMap.explainables.relation(variable)
```

Explainable

Contains an explainable object.

Fields

Name	Type	Description
<code>conjunction</code>	<code>string</code>	The subquery of the original query that is actually being explained.
<code>id</code>	<code>number</code>	A unique ID that identifies this Explainable.

Explanation

An explanation of which rule was used for inferring the explained concept, the condition of the rule, the conclusion of the rule, and the mapping of variables between the query and the rule's conclusion.

Fields

Name	Type	Description
<code>conclusion</code>	<code>ConceptMap</code>	The Conclusion for this Explanation.
<code>condition</code>	<code>ConceptMap</code>	The Condition for this Explanation.
<code>rule</code>	<code>Rule</code>	Retrieves the Rule for this Explanation.
<code>variableMapping</code>	<code>Map</code>	Retrieves the query variables for this Explanation.

Concept

ConceptManager

Provides access for all Concept API methods.

getAttribute

```
getAttribute(iid): Promise<Attribute>
```

Retrieves an `Attribute` by its iid.

Input parameters

Name	Description	Type
<code>iid</code>	The iid of the <code>Attribute</code> to retrieve	<code>string</code>

Returns

```
Promise<Attribute>
```

Code examples

```
transaction.concepts().getAttribute(iid)
```

getAttributeType

```
getAttributeType(label): Promise<AttributeType>
```

Retrieves an `AttributeType` by its label.

Input parameters

Name	Description	Type
<code>label</code>	The label of the <code>AttributeType</code> to retrieve	<code>string</code>

Returns

```
Promise<AttributeType>
```

Code examples

```
transaction.concepts().getAttributeType(label)
```

getEntity

```
getEntity(iid): Promise<Entity>
```

Retrieves an `Entity` by its iid.

Input parameters

Name	Description	Type
<code>iid</code>	The iid of the <code>Entity</code> to retrieve	<code>string</code>

Returns

```
Promise<Entity>
```

Code examples

```
transaction.concepts().getEntity(iid)
```

getEntityType

```
getEntityType(label): Promise<EntityType>
```

Retrieves an `EntityType` by its label.

Input parameters

Name	Description	Type
<code>label</code>	The label of the <code>EntityType</code> to retrieve	<code>string</code>

Returns

```
Promise<EntityType>
```

Code examples

```
transaction.concepts().getEntityType(label)
```

getRelation

```
getRelation(iid): Promise<Relation>
```

Retrieves a `Relation` by its iid.

Input parameters

Name	Description	Type
<code>iid</code>	The iid of the <code>Relation</code> to retrieve	<code>string</code>

Returns

```
Promise<Relation>
```

Code examples

```
transaction.concepts().getRelation(iid)
```

getRelationType

```
getRelationType(label): Promise<RelationType>
```

Retrieves a `RelationType` by its label.

Input parameters

Name	Description	Type
<code>label</code>	The label of the <code>RelationType</code> to retrieve	<code>string</code>

Returns

```
Promise<RelationType>
```

Code examples

```
transaction.concepts().getRelationType(label)
```

getRootAttributeType

```
getRootAttributeType(): Promise<AttributeType>
```

Retrieve the root `AttributeType`, "attribute".

Returns

```
Promise<AttributeType>
```

Code examples

```
transaction.concepts().getRootAttributeType()
```

getRootEntityType

```
getRootEntityType(): Promise<EntityType>
```

Retrieves the root `EntityType`, "entity".

Returns

```
Promise<EntityType>
```

Code examples

```
transaction.concepts().getRootEntityType()
```

getRootRelationType

```
getRootRelationType(): Promise<RelationType>
```

Retrieve the root `RelationType`, "relation".

Returns

```
Promise<RelationType>
```

Code examples

```
transaction.concepts().getRootRelationType()
```

getRootThingType

```
getRootThingType(): Promise<ThingType>
```

Retrieves the root `ThingType`, "thing".

Returns

```
Promise<ThingType>
```

Code examples

```
transaction.concepts().getRootThingType()
```

getSchemaExceptions

```
getSchemaExceptions(): Promise<TypeDBDriverError[]>
```

Retrieves a list of all schema exceptions for the current transaction.

Returns

```
Promise<TypeDBDriverError[]>
```

Code examples

```
transaction.concepts().getSchemaException()
```

putAttributeType

```
putAttributeType(label, valueType): Promise<AttributeType>
```

Creates a new `AttributeType` if none exists with the given label, or retrieves the existing one. or retrieve. :return:

Input parameters

Name	Description	Type
------	-------------	------

Name	Description	Type
<code>label</code>	The label of the <code>AttributeType</code> to create or retrieve	<code>string</code>
<code>valueType</code>	The value type of the <code>AttributeType</code> to create	<code>ValueType</code>

Returns

`Promise<AttributeType>`

Code examples

```
await transaction.concepts().putAttributeType(label, valueType)
```

putEntityType

```
putEntityType(label): Promise<EntityType>
```

Creates a new `EntityType` if none exists with the given label, otherwise retrieves the existing one.

Input parameters

Name	Description	Type
<code>label</code>	The label of the <code>EntityType</code> to create or retrieve	<code>string</code>

Returns

`Promise<EntityType>`

Code examples

```
transaction.concepts().putEntityType(label)
```

putRelationType


```
putRelationType(label): Promise<RelationType>
```

Creates a new `RelationType` if none exists with the given label, otherwise retrieves the existing one.

Input parameters

Name	Description	Type
<code>label</code>	The label of the <code>RelationType</code> to create or retrieve	<code>string</code>

Returns

```
Promise<RelationType>
```

Code examples

```
transaction.concepts().putRelationType(label)
```

Concept

asAttribute

```
asAttribute(): Attribute
```

Casts the concept to `Attribute`.

Returns

```
Attribute
```

Code examples

```
concept.asAttribute()
```

asAttributeType

```
asAttributeType(): AttributeType
```

Casts the concept to `AttributeType`.

Returns

`AttributeType`

Code examples

```
concept.asAttributeType()
```

asEntity

```
asEntity(): Entity
```

Casts the concept to `Entity`.

Returns

`Entity`

Code examples

```
concept.asEntity()
```

asEntityType

```
asEntityType(): EntityType
```

Casts the concept to `EntityType`.

Returns

`EntityType`

Code examples

```
concept.asEntityType()
```

asRelation

```
asRelation(): Relation
```

Casts the concept to `Relation`.

Returns

`Relation`

Code examples

```
concept.asRelation()
```

asRelationType

```
asRelationType(): RelationType
```

Casts the concept to `RelationType`.

Returns

`RelationType`

Code examples

```
concept.asRelationType()
```

asRoleType

```
asRoleType(): RoleType
```

Casts the concept to `RoleType`.

Returns

`RoleType`

Code examples

```
concept.asRoleType()
```

asThing

```
asThing(): Thing
```

Casts the concept to `Thing`.

Returns

`Thing`

Code examples

```
concept.asThing()
```

asThingType

```
asThingType(): ThingType
```

Casts the concept to `ThingType`.

Returns

`ThingType`

Code examples

```
concept.asThingType()
```

asType

```
asType(): Type
```

Casts the concept to `Type`.

Returns

`Type`

Code examples

```
concept.asType()
```

asValue

```
asValue(): Value
```

Casts the concept to `Value`.

Returns

`Value`

Code examples

```
concept.asValue()
```

equals

```
equals(concept): boolean
```

Checks if this concept is equal to the argument `concept`.

Input parameters

Name	Description	Type
<code>concept</code>	The concept to compare to.	<code>Concept</code>

Returns

`boolean`

isAttribute

```
isAttribute(): boolean
```

Checks if the concept is an `Attribute`.

Returns

`boolean`

Code examples

```
concept.isAttribute()
```

isAttributeType

```
isAttributeType(): boolean
```

Checks if the concept is an `AttributeType`.

Returns

```
boolean
```

Code examples

```
concept.isAttributeType()
```

isEntity

```
isEntity(): boolean
```

Checks if the concept is an `Entity`.

Returns

```
boolean
```

Code examples

```
concept.isEntity()
```

isEntityType

```
isEntityType(): boolean
```

Checks if the concept is an `EntityType`.

Returns

```
boolean
```

Code examples

```
concept.isEntityType()
```

isRelation

```
isRelation(): boolean
```

Checks if the concept is a `Relation`.

Returns

```
boolean
```

Code examples

```
concept.isRelation()
```

isRelationType

```
isRelationType(): boolean
```

Checks if the concept is a `RelationType`.

Returns

```
boolean
```

Code examples

```
concept.isRelationType()
```

isRoleType

```
isRoleType(): boolean
```

Checks if the concept is a `RoleType`.

Returns

```
boolean
```

Code examples

```
concept.isRoleType()
```

isThing

```
isThing(): boolean
```

Checks if the concept is a `Thing`.

Returns

```
boolean
```

Code examples

```
concept.isThing()
```

isThingType

```
isThingType(): boolean
```

Checks if the concept is a `ThingType`.

Returns

```
boolean
```

Code examples

```
concept.isThingType()
```

isType

```
isType(): boolean
```

Checks if the concept is a `Type`.

Returns

```
boolean
```

Code examples

```
concept.isType()
```

isValue


```
isValue(): boolean
```

Checks if the concept is a `Value`.

Returns

```
boolean
```

Code examples

```
concept.isValue()
```

toJsonRecord

```
toJsonRecord(): Record<string, string | number | boolean>
```

Retrieves the concept as JSON.

Returns

```
Record<string, string | number | boolean>
```

Code examples

```
concept.toJsonRecord()
```

Schema

Type

Supertypes:

- `Concept`

Fields

Name	Type	Description
------	------	-------------

Name	Type	Description
<code>abstract</code>	<code>boolean</code>	Whether the type is prevented from having data instances (i.e., abstract).
<code>label</code>	<code>Label</code>	The unique label of the type.
<code>root</code>	<code>boolean</code>	Whether the type is a root type.

asAttribute

```
asAttribute(): Attribute
```

Casts the concept to `Attribute`.

Returns

`Attribute`

Code examples

```
concept.asAttribute()
```

asAttributeType

```
asAttributeType(): AttributeType
```

Casts the concept to `AttributeType`.

Returns

`AttributeType`

Code examples

```
concept.asAttributeType()
```

asEntity

```
asEntity(): Entity
```

Casts the concept to `Entity`.

Returns

`Entity`

Code examples

```
concept.asEntity()
```

asEntityType

```
asEntityType(): EntityType
```

Casts the concept to `EntityType`.

Returns

`EntityType`

Code examples

```
concept.asEntityType()
```

asRelation

```
asRelation(): Relation
```

Casts the concept to `Relation`.

Returns

`Relation`

Code examples

```
concept.asRelation()
```

asRelationType

```
asRelationType(): RelationType
```

Casts the concept to `RelationType`.

Returns

```
RelationType
```

Code examples

```
concept.asRelationType()
```

asRoleType

```
asRoleType(): RoleType
```

Casts the concept to `RoleType`.

Returns

```
RoleType
```

Code examples

```
concept.asRoleType()
```

asThing

```
asThing(): Thing
```

Casts the concept to `Thing`.

Returns

```
Thing
```

Code examples

```
concept.asThing()
```

asThingType

```
asThingType(): ThingType
```

Casts the concept to `ThingType`.

Returns

```
ThingType
```

Code examples

```
concept.asThingType()
```

asType

```
asType(): Type
```

Casts the concept to `Type`.

Returns

```
Type
```

Code examples

```
concept.asType()
```

asValue

```
asValue(): Value
```

Casts the concept to `Value`.

Returns

```
Value
```

Code examples

```
concept.asValue()
```

delete

```
delete(transaction): Promise<void>
```

Deletes this type from the database.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

```
Promise<void>
```

Code examples

```
type.delete(transaction)
```

equals

```
equals(concept): boolean
```

Checks if this concept is equal to the argument `concept`.

Input parameters

Name	Description	Type
<code>concept</code>	The concept to compare to.	<code>Concept</code>

Returns

```
boolean
```

getSubtypes

```
getSubtypes(transaction): Stream<Type>
```

Retrieves all direct and indirect (or direct only) subtypes of the type.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<Type>`

Code examples

```
type.getSubtypes(transaction) type.getSubtypes(transaction,
Transitivity.EXPLICIT)
```

getSubtypes

```
getSubtypes(transaction, transitivity): Stream<Type>
```

Retrieves all direct and indirect (or direct only) subtypes of the type.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>transitivity</code>	<code>Transitivity.TRANSITIVE</code> for direct and indirect subtypes, <code>Transitivity.EXPLICIT</code> for direct subtypes only	<code>Transitivity</code>

Returns

`Stream<Type>`

Code examples

```
type.getSubtypes(transaction) type.getSubtypes(transaction,
Transitivity.EXPLICIT)
```

getSupertype

```
getSupertype(transaction): Promise<Type>
```

Retrieves the most immediate supertype of the type.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

```
Promise<Type>
```

Code examples

```
type.getSupertype(transaction)
```

getSupertypes

```
getSupertypes(transaction): Stream<Type>
```

Retrieves all supertypes of the type.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

```
Stream<Type>
```

Code examples

```
type.getSupertypes(transaction)
```


isAttribute

```
isAttribute(): boolean
```

Checks if the concept is an `Attribute`.

Returns

`boolean`

Code examples

```
concept.isAttribute()
```

isAttributeType

```
isAttributeType(): boolean
```

Checks if the concept is an `AttributeType`.

Returns

`boolean`

Code examples

```
concept.isAttributeType()
```

isDeleted

```
isDeleted(transaction): Promise<boolean>
```

Check if the concept has been deleted

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<boolean>`

isEntity

```
isEntity(): boolean
```

Checks if the concept is an `Entity`.

Returns

`boolean`

Code examples

```
concept.isEntity()
```

isEntityType

```
isEntityType(): boolean
```

Checks if the concept is an `EntityType`.

Returns

`boolean`

Code examples

```
concept.isEntityType()
```

isRelation

```
isRelation(): boolean
```

Checks if the concept is a `Relation`.

Returns

`boolean`

Code examples

```
concept.isRelation()
```

isRelationType

```
isRelationType(): boolean
```

Checks if the concept is a `RelationType`.

Returns

`boolean`

Code examples

```
concept.isRelationType()
```

isRoleType

```
isRoleType(): boolean
```

Checks if the concept is a `RoleType`.

Returns

`boolean`

Code examples

```
concept.isRoleType()
```

isThing

```
isThing(): boolean
```

Checks if the concept is a `Thing`.

Returns

`boolean`

Code examples

```
concept.isThing()
```

isThingType

```
isThingType(): boolean
```

Checks if the concept is a `ThingType`.

Returns

```
boolean
```

Code examples

```
concept.isThingType()
```

isType

```
isType(): boolean
```

Checks if the concept is a `Type`.

Returns

```
boolean
```

Code examples

```
concept.isType()
```

isValue

```
isValue(): boolean
```

Checks if the concept is a `Value`.

Returns

```
boolean
```

Code examples

```
concept.isValue()
```

setLabel

```
setLabel(transaction, label): Promise<void>
```

Renames the label of the type. The new label must remain unique.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>label</code>	The new <code>Label</code> to be given to the type.	<code>string</code>

Returns

```
Promise<void>
```

Code examples

```
type.setLabel(transaction, label)
```

toJSONRecord

```
toJSONRecord(): Record<string, string | number | boolean>
```

Retrieves the concept as JSON.

Returns

```
Record<string, string | number | boolean>
```

Code examples

```
concept.toJSONRecord()
```

Label

A `Label` holds the uniquely identifying name of a type.

It consists of an optional 'scope', and a 'name', represented "scope:name". The scope is used only used to distinguish between role-types of the same name declared in different relation types.

name

```
get name(): string
```

Returns the name part of the label.

Returns

```
string
```

scope

```
get scope(): string
```

Returns the (possibly null) scope part of the label.

Returns

```
string
```

scopedName

```
get scopedName(): string
```

Returns the string representation of the scoped name.

Returns

```
string
```

equals

```
equals(that): boolean
```

Compares this label to `that` label.

Input parameters

Name	Description	Type
<code>that</code>	The label to compare to.	<code>Label</code>

Returns

`boolean`

toString

```
toString(): string
```

Printable string

Returns

`string`

ThingType

Supertypes:

- `Type`

Fields

Name	Type	Description
<code>abstract</code>	<code>boolean</code>	Whether the type is prevented from having data instances (i.e., abstract).
<code>label</code>	<code>Label</code>	The unique label of the type.
<code>root</code>	<code>boolean</code>	Whether the type is a root type.

asAttribute

```
asAttribute(): Attribute
```

Casts the concept to `Attribute`.

Returns

`Attribute`

Code examples

```
concept.asAttribute()
```

asAttributeType

```
asAttributeType(): AttributeType
```

Casts the concept to `AttributeType`.

Returns

`AttributeType`

Code examples

```
concept.asAttributeType()
```

asEntity

```
asEntity(): Entity
```

Casts the concept to `Entity`.

Returns

`Entity`

Code examples

```
concept.asEntity()
```

asEntityType

```
asEntityType(): EntityType
```


Casts the concept to `EntityType`.

Returns

`EntityType`

Code examples

```
concept.asEntityType()
```

asRelation

```
asRelation(): Relation
```

Casts the concept to `Relation`.

Returns

`Relation`

Code examples

```
concept.asRelation()
```

asRelationType

```
asRelationType(): RelationType
```

Casts the concept to `RelationType`.

Returns

`RelationType`

Code examples

```
concept.asRelationType()
```

asRoleType

```
asRoleType(): RoleType
```

Casts the concept to `RoleType`.

Returns

`RoleType`

Code examples

```
concept.asRoleType()
```

asThing

```
asThing(): Thing
```

Casts the concept to `Thing`.

Returns

`Thing`

Code examples

```
concept.asThing()
```

asThingType

```
asThingType(): ThingType
```

Casts the concept to `ThingType`.

Returns

`ThingType`

Code examples

```
concept.asThingType()
```

asType

```
asType(): Type
```

Casts the concept to `Type`.

Returns

`Type`

Code examples

```
concept.asType()
```

asValue

```
asValue(): Value
```

Casts the concept to `Value`.

Returns

`Value`

Code examples

```
concept.asValue()
```

delete

```
delete(transaction): Promise<void>
```

Deletes this type from the database.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<void>`

Code examples

```
type.delete(transaction)
```

equals

```
equals(concept): boolean
```

Checks if this concept is equal to the argument `concept`.

Input parameters

Name	Description	Type
<code>concept</code>	The concept to compare to.	<code>Concept</code>

Returns

`boolean`

getInstances

```
getInstances(transaction): Stream<Thing>
```

Retrieves all direct and indirect `Thing` objects that are instances of this `ThingType`.
Equivalent to `getInstances(transaction, Transitivity.TRANSITIVE)`

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<Thing>`

Code examples

```
thingType.getInstances(transaction)
```

getInstances

```
getInstances(transaction, transitivity): Stream<Thing>
```

Retrieves all direct and indirect (or direct only) `Thing` objects that are instances of this `ThingType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>transitivity</code>	<code>Transitivity.TRANSITIVE</code> for direct and indirect instances, <code>Transitivity.EXPLICIT</code> for direct instances only	<code>Transitivity</code>

Returns

`Stream<Thing>`

Code examples

```
thingType.getInstances(transaction, Transitivity.EXPLICIT)
```

getOwns

```
getOwns(transaction): Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<AttributeType>`

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT,[Annotation.KEY])
```

getOwns

```
getOwns(transaction, valueType): Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>valueType</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>ValueType</code>

Returns

`Stream<AttributeType>`

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT,[Annotation.KEY])
```

getOwns

```
getOwns(transaction, annotations): Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>annotations</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>Annotation[]</code>

Returns

`Stream<AttributeType>`

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT,[Annotation.KEY])
```

getOwns

```
getOwns(transaction, valueType, annotations): Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>valueType</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>ValueType</code>
<code>annotations</code>	Only retrieve attribute types owned with annotations.	<code>Annotation[]</code>

Returns

`Stream<AttributeType>`

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT,[Annotation.KEY])
```

getOwns

```
getOwns(transaction, transitivity): Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>transitivity</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>Transitivity</code>

Returns

```
Stream<AttributeType>
```

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT,[Annotation.KEY])
```

getOwns

```
getOwns(transaction, valueType, transitivity): Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>valueType</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>ValueType</code>
<code>transitivity</code>	Only retrieve attribute types owned with annotations.	<code>Transitivity</code>

Returns

`Stream<AttributeType>`

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT, [Annotation.KEY])
```

getOwns

```
getOwns(transaction, annotations, transitivity): Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>annotations</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>Annotation[]</code>
<code>transitivity</code>	Only retrieve attribute types owned with annotations.	<code>Transitivity</code>

Returns

`Stream<AttributeType>`

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT,[Annotation.KEY])
```

getOwns

```
getOwns(transaction, valueType, annotations, transitivity):
Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>valueType</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>ValueType</code>
<code>annotations</code>	Only retrieve attribute types owned with annotations.	<code>Annotation[]</code>
<code>transitivity</code>	<code>Transitivity.TRANSITIVE</code> for direct and inherited ownership, <code>Transitivity.EXPLICIT</code> for direct ownership only	<code>Transitivity</code>

Returns

`Stream<AttributeType>`

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT,[Annotation.KEY])
```

getOwnsOverridden

```
getOwnsOverridden(transaction, attributeType): Promise<AttributeType>
```

Retrieves an `AttributeType`, ownership of which is overridden for this `ThingType` by a given `attribute_type`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeType</code>	The <code>AttributeType</code> that overrides requested <code>AttributeType</code>	<code>AttributeType</code>

Returns

```
Promise<AttributeType>
```

Code examples

```
thingType.getOwnsOverridden(transaction, attributeType)
```

getPlays

```
getPlays(transaction): Stream<RoleType>
```

Retrieves all direct and inherited (or direct only) roles that are allowed to be played by the instances of this `ThingType`.

Input parameters

Name	Description	Type
------	-------------	------

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<RoleType>`

Code examples

```
thingType.getPlays(transaction) thingType.getPlays(transaction,
Transitivity.EXPLICIT)
```

getPlays

```
getPlays(transaction, transitivity): Stream<RoleType>
```

Retrieves all direct and inherited (or direct only) roles that are allowed to be played by the instances of this `ThingType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>transitivity</code>	<code>Transitivity.TRANSITIVE</code> for direct and indirect playing, <code>Transitivity.EXPLICIT</code> for direct playing only	<code>Transitivity</code>

Returns

`Stream<RoleType>`

Code examples

```
thingType.getPlays(transaction) thingType.getPlays(transaction,
Transitivity.EXPLICIT)
```

getPlaysOverridden

```
getPlaysOverridden(transaction, role): Promise<RoleType>
```

Retrieves a `RoleType` that is overridden by the given `role_type` for this `ThingType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>role</code>	The <code>RoleType</code> that overrides an inherited role	<code>RoleType</code>

Returns

```
Promise<RoleType>
```

Code examples

```
thingType.getPlaysOverridden(transaction, role)
```

getSubtypes

```
getSubtypes(transaction): Stream<ThingType>
```

Retrieves all direct and indirect subtypes of the `ThingType`. Equivalent to `getSubtypes(transaction, Transitivity.TRANSITIVE)`

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

```
Stream<ThingType>
```

Code examples

```
thingType.getSubtypes(transaction)
```

getSubtypes

```
getSubtypes(transaction, transitivity): Stream<ThingType>
```

Retrieves all direct and indirect (or direct only) subtypes of the `ThingType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>transitivity</code>	<code>Transitivity.TRANSITIVE</code> for direct and indirect subtypes, <code>Transitivity.EXPLICIT</code> for direct subtypes only	<code>Transitivity</code>

Returns

```
Stream<ThingType>
```

Code examples

```
thingType.getSubtypes(transaction, Transitivity.EXPLICIT)
```

getSupertype

```
getSupertype(transaction): Promise<ThingType>
```

Retrieves the most immediate supertype of the `ThingType`.

Input parameters

Name	Description	Type
------	-------------	------

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<ThingType>`

Code examples

```
thingType.getSupertype(transaction)
```

getSupertypes

```
getSupertypes(transaction): Stream<ThingType>
```

Retrieves all supertypes of the `ThingType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<ThingType>`

Code examples

```
thingType.getSupertypes(transaction)
```

getSyntax

```
getSyntax(transaction): Promise<string>
```

Produces a pattern for creating this `ThingType` in a `define` query.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<string>`

Code examples

```
thingType.getSyntax(transaction)
```

isAttribute

```
isAttribute(): boolean
```

Checks if the concept is an `Attribute`.

Returns

`boolean`

Code examples

```
concept.isAttribute()
```

isAttributeType

```
isAttributeType(): boolean
```

Checks if the concept is an `AttributeType`.

Returns

`boolean`

Code examples

```
concept.isAttributeType()
```


isDeleted

```
isDeleted(transaction): Promise<boolean>
```

Check if the concept has been deleted

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

```
Promise<boolean>
```

isEntity

```
isEntity(): boolean
```

Checks if the concept is an `Entity`.

Returns

```
boolean
```

Code examples

```
concept.isEntity()
```

isEntityType

```
isEntityType(): boolean
```

Checks if the concept is an `EntityType`.

Returns

```
boolean
```

Code examples

```
concept.isEntityType()
```

isRelation

```
isRelation(): boolean
```

Checks if the concept is a **Relation**.

Returns

boolean

Code examples

```
concept.isRelation()
```

isRelationType

```
isRelationType(): boolean
```

Checks if the concept is a **RelationType**.

Returns

boolean

Code examples

```
concept.isRelationType()
```

isRoleType

```
isRoleType(): boolean
```

Checks if the concept is a **RoleType**.

Returns

boolean

Code examples

```
concept.isRoleType()
```

isThing

```
isThing(): boolean
```

Checks if the concept is a **Thing**.

Returns

boolean

Code examples

```
concept.isThing()
```

isThingType

```
isThingType(): boolean
```

Checks if the concept is a **ThingType**.

Returns

boolean

Code examples

```
concept.isThingType()
```

isType

```
isType(): boolean
```

Checks if the concept is a **Type**.

Returns

boolean

Code examples

```
concept.isType()
```

isValue

```
isValue(): boolean
```

Checks if the concept is a `Value`.

Returns

```
boolean
```

Code examples

```
concept.isValue()
```

setAbstract

```
setAbstract(transaction): Promise<void>
```

Set a `ThingType` to be abstract, meaning it cannot have instances.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

```
Promise<void>
```

Code examples

```
thingType.setAbstract(transaction)
```

setLabel

```
setLabel(transaction, label): Promise<void>
```

Renames the label of the type. The new label must remain unique.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>label</code>	The new <code>Label</code> to be given to the type.	<code>string</code>

Returns

```
Promise<void>
```

Code examples

```
type.setLabel(transaction, label)
```

setOwns

```
setOwns(transaction, attributeType): Promise<void>
```

Allows the instances of this `ThingType` to own the given `AttributeType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeType</code>	The <code>AttributeType</code> to be owned by the instances of this type.	<code>AttributeType</code>

Returns

```
Promise<void>
```

Code examples

```
thingType.setOwns(transaction, attributeType)
thingType.setOwns(transaction, attributeType, overriddenType,
[Annotation.KEY])
```

setOwns

```
setOwns(transaction, attributeType, annotations): Promise<void>
```

Allows the instances of this `ThingType` to own the given `AttributeType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeType</code>	The <code>AttributeType</code> to be owned by the instances of this type.	<code>AttributeType</code>
<code>annotations</code>	The <code>AttributeType</code> that this attribute ownership overrides, if applicable.	<code>Annotation[]</code>

Returns

`Promise<void>`

Code examples

```
thingType.setOwns(transaction, attributeType)
thingType.setOwns(transaction, attributeType, overriddenType,
[Annotation.KEY])
```

setOwns

```
setOwns(transaction, attributeType, overriddenType): Promise<void>
```

Allows the instances of this `ThingType` to own the given `AttributeType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeType</code>	The <code>AttributeType</code> to be owned by the instances of this type.	<code>AttributeType</code>
<code>overriddenType</code>	The <code>AttributeType</code> that this attribute ownership overrides, if applicable.	<code>AttributeType</code>

Returns

`Promise<void>`

Code examples

```
thingType.setOwns(transaction, attributeType)
thingType.setOwns(transaction, attributeType, overriddenType,
[Annotation.KEY])
```

setOwns

```
setOwns(transaction, attributeType, overriddenType, annotations):
Promise<void>
```

Allows the instances of this `ThingType` to own the given `AttributeType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeType</code>	The <code>AttributeType</code> to be owned by the instances of this type.	<code>AttributeType</code>

Name	Description	Type
<code>overriddenType</code>	The <code>AttributeType</code> that this attribute ownership overrides, if applicable.	<code>AttributeType</code>
<code>annotations</code>	Adds annotations to the ownership.	<code>Annotation[]</code>

Returns

`Promise<void>`

Code examples

```
thingType.setOwns(transaction, attributeType)
thingType.setOwns(transaction, attributeType, overriddenType,
[Annotation.KEY])
```

setPlays

```
setPlays(transaction, role): Promise<void>
```

Allows the instances of this `ThingType` to play the given role.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>role</code>	The role to be played by the instances of this type	<code>RoleType</code>

Returns

`Promise<void>`

Code examples


```
thingType.setPlays(transaction, role) thingType.setPlays(transaction, role,
overriddenType)
```

setPlays

```
setPlays(transaction, role, overriddenType): Promise<void>
```

Allows the instances of this `ThingType` to play the given role.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>role</code>	The role to be played by the instances of this type	<code>RoleType</code>
<code>overriddenType</code>	The role type that this role overrides, if applicable	<code>RoleType</code>

Returns

```
Promise<void>
```

Code examples

```
thingType.setPlays(transaction, role) thingType.setPlays(transaction, role,
overriddenType)
```

toJSONRecord

```
toJSONRecord(): Record<string, string | number | boolean>
```

Retrieves the concept as JSON.

Returns

```
Record<string, string | number | boolean>
```

Code examples

```
concept.toJSONRecord()
```

unsetAbstract

```
unsetAbstract(transaction): Promise<void>
```

Set a `ThingType` to be non-abstract, meaning it can have instances.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<void>`

Code examples

```
thingType.unsetAbstract(transaction)
```

unsetOwns

```
unsetOwns(transaction, attributeType): Promise<void>
```

Disallows the instances of this `ThingType` from owning the given `AttributeType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeType</code>	The <code>AttributeType</code> to not be owned by the type.	<code>AttributeType</code>

Returns

`Promise<void>`

Code examples

```
thingType.unsetOwns(transaction, attributeType)
```

unsetPlays

```
unsetPlays(transaction, role): Promise<void>
```

Disallows the instances of this `ThingType` from playing the given role.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>role</code>	The role to not be played by the instances of this type.	<code>RoleType</code>

Returns

`Promise<void>`

Code examples

```
thingType.unsetPlays(transaction, role)
```

EntityType

Supertypes:

- `ThingType`

Fields

Name	Type	Description
------	------	-------------

Name	Type	Description
<code>abstract</code>	<code>boolean</code>	Whether the type is prevented from having data instances (i.e., abstract).
<code>label</code>	<code>Label</code>	The unique label of the type.
<code>root</code>	<code>boolean</code>	Whether the type is a root type.

asAttribute

```
asAttribute(): Attribute
```

Casts the concept to `Attribute`.

Returns

`Attribute`

Code examples

```
concept.asAttribute()
```

asAttributeType

```
asAttributeType(): AttributeType
```

Casts the concept to `AttributeType`.

Returns

`AttributeType`

Code examples

```
concept.asAttributeType()
```

asEntity

```
asEntity(): Entity
```

Casts the concept to `Entity`.

Returns

`Entity`

Code examples

```
concept.asEntity()
```

asEntityType

```
asEntityType(): EntityType
```

Casts the concept to `EntityType`.

Returns

`EntityType`

Code examples

```
concept.asEntityType()
```

asRelation

```
asRelation(): Relation
```

Casts the concept to `Relation`.

Returns

`Relation`

Code examples

```
concept.asRelation()
```

asRelationType

```
asRelationType(): RelationType
```

Casts the concept to `RelationType`.

Returns

```
RelationType
```

Code examples

```
concept.asRelationType()
```

asRoleType

```
asRoleType(): RoleType
```

Casts the concept to `RoleType`.

Returns

```
RoleType
```

Code examples

```
concept.asRoleType()
```

asThing

```
asThing(): Thing
```

Casts the concept to `Thing`.

Returns

```
Thing
```

Code examples

```
concept.asThing()
```

asThingType

```
asThingType(): ThingType
```

Casts the concept to `ThingType`.

Returns

```
ThingType
```

Code examples

```
concept.asThingType()
```

asType

```
asType(): Type
```

Casts the concept to `Type`.

Returns

```
Type
```

Code examples

```
concept.asType()
```

asValue

```
asValue(): Value
```

Casts the concept to `Value`.

Returns

```
Value
```

Code examples

```
concept.asValue()
```

create

```
create(transaction): Promise<Entity>
```

Input parameters

Name	Description	Type
<code>transaction</code>		<code>TypeDBTransaction</code>

Returns

```
Promise<Entity>
```

delete

```
delete(transaction): Promise<void>
```

Deletes this type from the database.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

```
Promise<void>
```

Code examples

```
type.delete(transaction)
```

equals

```
equals(concept): boolean
```

Checks if this concept is equal to the argument `concept`.

Input parameters

Name	Description	Type
<code>concept</code>	The concept to compare to.	<code>Concept</code>

Returns

`boolean`

getInstances

```
getInstances(transaction): Stream<Entity>
```

Retrieves all direct and indirect `Thing` objects that are instances of this `ThingType`.
Equivalent to `getInstances(transaction, Transitivity.TRANSITIVE)`

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<Entity>`

Code examples

```
thingType.getInstances(transaction)
```

getInstances

```
getInstances(transaction, transitivity): Stream<Entity>
```

Retrieves all direct and indirect (or direct only) `Thing` objects that are instances of this `ThingType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>transitivity</code>	<code>Transitivity.TRANSITIVE</code> for direct and indirect instances, <code>Transitivity.EXPLICIT</code> for direct instances only	<code>Transitivity</code>

Returns

`Stream<Entity>`

Code examples

```
thingType.getInstances(transaction, Transitivity.EXPLICIT)
```

getOwns

```
getOwns(transaction): Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<AttributeType>`

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,  
Transitivity.EXPLICIT, [Annotation.KEY])
```

getOwns

```
getOwns(transaction, valueType): Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>valueType</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>ValueType</code>

Returns

```
Stream<AttributeType>
```

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT, [Annotation.KEY])
```

getOwns

```
getOwns(transaction, annotations): Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Name	Description	Type
<code>annotations</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>Annotation[]</code>

Returns

`Stream<AttributeType>`

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT,[Annotation.KEY])
```

getOwns

```
getOwns(transaction, valueType, annotations): Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>valueType</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>ValueType</code>
<code>annotations</code>	Only retrieve attribute types owned with annotations.	<code>Annotation[]</code>

Returns

`Stream<AttributeType>`

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT,[Annotation.KEY])
```

getOwns

```
getOwns(transaction, transitivity): Stream<AttributeType>
```

Retrieves **AttributeType** that the instances of this **ThingType** are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
transaction	The current transaction	TypeDBTransaction
transitivity	If specified, only attribute types of this ValueType will be retrieved.	Transitivity

Returns

Stream<AttributeType>

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT,[Annotation.KEY])
```

getOwns

```
getOwns(transaction, valueType, transitivity): Stream<AttributeType>
```

Retrieves **AttributeType** that the instances of this **ThingType** are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
------	-------------	------

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>valueType</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>ValueType</code>
<code>transitivity</code>	Only retrieve attribute types owned with annotations.	<code>Transitivity</code>

Returns

`Stream<AttributeType>`

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT, [Annotation.KEY])
```

getOwns

```
getOwns(transaction, annotations, transitivity): Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>annotations</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>Annotation[]</code>
<code>transitivity</code>	Only retrieve attribute types owned with annotations.	<code>Transitivity</code>

Returns

`Stream<AttributeType>`

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,  
Transitivity.EXPLICIT,[Annotation.KEY])
```

getOwns

```
getOwns(transaction, valueType, annotations, transitivity):  
Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>valueType</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>ValueType</code>
<code>annotations</code>	Only retrieve attribute types owned with annotations.	<code>Annotation[]</code>
<code>transitivity</code>	<code>Transitivity.TRANSITIVE</code> for direct and inherited ownership, <code>Transitivity.EXPLICIT</code> for direct ownership only	<code>Transitivity</code>

Returns

`Stream<AttributeType>`

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT,[Annotation.KEY])
```

getOwnsOverridden

```
getOwnsOverridden(transaction, attributeType): Promise<AttributeType>
```

Retrieves an `AttributeType`, ownership of which is overridden for this `ThingType` by a given `attribute_type`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeType</code>	The <code>AttributeType</code> that overrides requested <code>AttributeType</code>	<code>AttributeType</code>

Returns

```
Promise<AttributeType>
```

Code examples

```
thingType.getOwnsOverridden(transaction, attributeType)
```

getPlays

```
getPlays(transaction): Stream<RoleType>
```

Retrieves all direct and inherited (or direct only) roles that are allowed to be played by the instances of this `ThingType`.

Input parameters

Name	Description	Type
------	-------------	------

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<RoleType>`

Code examples

```
thingType.getPlays(transaction) thingType.getPlays(transaction,
Transitivity.EXPLICIT)
```

getPlays

```
getPlays(transaction, transitivity): Stream<RoleType>
```

Retrieves all direct and inherited (or direct only) roles that are allowed to be played by the instances of this `ThingType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>transitivity</code>	<code>Transitivity.TRANSITIVE</code> for direct and indirect playing, <code>Transitivity.EXPLICIT</code> for direct playing only	<code>Transitivity</code>

Returns

`Stream<RoleType>`

Code examples

```
thingType.getPlays(transaction) thingType.getPlays(transaction,
Transitivity.EXPLICIT)
```

getPlaysOverridden

```
getPlaysOverridden(transaction, role): Promise<RoleType>
```

Retrieves a `RoleType` that is overridden by the given `role_type` for this `ThingType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>role</code>	The <code>RoleType</code> that overrides an inherited role	<code>RoleType</code>

Returns

```
Promise<RoleType>
```

Code examples

```
thingType.getPlaysOverridden(transaction, role)
```

getSubtypes

```
getSubtypes(transaction): Stream<EntityType>
```

Retrieves all direct and indirect subtypes of the `ThingType`. Equivalent to `getSubtypes(transaction, Transitivity.TRANSITIVE)`

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

```
Stream<EntityType>
```

Code examples

```
thingType.getSubtypes(transaction)
```

getSubtypes

```
getSubtypes(transaction, transitivity): Stream<EntityType>
```

Retrieves all direct and indirect (or direct only) subtypes of the `ThingType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>transitivity</code>	<code>Transitivity.TRANSITIVE</code> for direct and indirect subtypes, <code>Transitivity.EXPLICIT</code> for direct subtypes only	<code>Transitivity</code>

Returns

```
Stream<EntityType>
```

Code examples

```
thingType.getSubtypes(transaction, Transitivity.EXPLICIT)
```

getSupertype

```
getSupertype(transaction): Promise<EntityType>
```

Retrieves the most immediate supertype of the `ThingType`.

Input parameters

Name	Description	Type
------	-------------	------

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<EntityType>`

Code examples

```
thingType.getSupertype(transaction)
```

getSupertypes

```
getSupertypes(transaction): Stream<EntityType>
```

Retrieves all supertypes of the `ThingType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<EntityType>`

Code examples

```
thingType.getSupertypes(transaction)
```

getSyntax

```
getSyntax(transaction): Promise<string>
```

Produces a pattern for creating this `ThingType` in a `define` query.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<string>`

Code examples

```
thingType.getSyntax(transaction)
```

isAttribute

```
isAttribute(): boolean
```

Checks if the concept is an `Attribute`.

Returns

`boolean`

Code examples

```
concept.isAttribute()
```

isAttributeType

```
isAttributeType(): boolean
```

Checks if the concept is an `AttributeType`.

Returns

`boolean`

Code examples

```
concept.isAttributeType()
```

isDeleted

```
isDeleted(transaction): Promise<boolean>
```

Check if the concept has been deleted

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

```
Promise<boolean>
```

isEntity

```
isEntity(): boolean
```

Checks if the concept is an `Entity`.

Returns

```
boolean
```

Code examples

```
concept.isEntity()
```

isEntityType

```
isEntityType(): boolean
```

Checks if the concept is an `EntityType`.

Returns

```
boolean
```

Code examples

```
concept.isEntityType()
```

isRelation

```
isRelation(): boolean
```

Checks if the concept is a **Relation**.

Returns

boolean

Code examples

```
concept.isRelation()
```

isRelationType

```
isRelationType(): boolean
```

Checks if the concept is a **RelationType**.

Returns

boolean

Code examples

```
concept.isRelationType()
```

isRoleType

```
isRoleType(): boolean
```

Checks if the concept is a **RoleType**.

Returns

boolean

Code examples

```
concept.isRoleType()
```

isThing

```
isThing(): boolean
```

Checks if the concept is a **Thing**.

Returns

boolean

Code examples

```
concept.isThing()
```

isThingType

```
isThingType(): boolean
```

Checks if the concept is a **ThingType**.

Returns

boolean

Code examples

```
concept.isThingType()
```

isType

```
isType(): boolean
```

Checks if the concept is a **Type**.

Returns

boolean

Code examples


```
concept.isType()
```

isValue

```
isValue(): boolean
```

Checks if the concept is a `Value`.

Returns

```
boolean
```

Code examples

```
concept.isValue()
```

setAbstract

```
setAbstract(transaction): Promise<void>
```

Set a `ThingType` to be abstract, meaning it cannot have instances.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

```
Promise<void>
```

Code examples

```
thingType.setAbstract(transaction)
```

setLabel

```
setLabel(transaction, label): Promise<void>
```

Renames the label of the type. The new label must remain unique.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>label</code>	The new <code>Label</code> to be given to the type.	<code>string</code>

Returns

```
Promise<void>
```

Code examples

```
type.setLabel(transaction, label)
```

setOwns

```
setOwns(transaction, attributeType): Promise<void>
```

Allows the instances of this `ThingType` to own the given `AttributeType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeType</code>	The <code>AttributeType</code> to be owned by the instances of this type.	<code>AttributeType</code>

Returns

```
Promise<void>
```

Code examples

```
thingType.setOwns(transaction, attributeType)
thingType.setOwns(transaction, attributeType, overriddenType,
[Annotation.KEY])
```

setOwns

```
setOwns(transaction, attributeType, annotations): Promise<void>
```

Allows the instances of this `ThingType` to own the given `AttributeType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeType</code>	The <code>AttributeType</code> to be owned by the instances of this type.	<code>AttributeType</code>
<code>annotations</code>	The <code>AttributeType</code> that this attribute ownership overrides, if applicable.	<code>Annotation[]</code>

Returns

```
Promise<void>
```

Code examples

```
thingType.setOwns(transaction, attributeType)
thingType.setOwns(transaction, attributeType, overriddenType,
[Annotation.KEY])
```

setOwns

```
setOwns(transaction, attributeType, overriddenType): Promise<void>
```

Allows the instances of this `ThingType` to own the given `AttributeType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeType</code>	The <code>AttributeType</code> to be owned by the instances of this type.	<code>AttributeType</code>
<code>overriddenType</code>	The <code>AttributeType</code> that this attribute ownership overrides, if applicable.	<code>AttributeType</code>

Returns

`Promise<void>`

Code examples

```
thingType.setOwns(transaction, attributeType)
thingType.setOwns(transaction, attributeType, overriddenType,
[Annotation.KEY])
```

setOwns

```
setOwns(transaction, attributeType, overriddenType, annotations):
Promise<void>
```

Allows the instances of this `ThingType` to own the given `AttributeType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeType</code>	The <code>AttributeType</code> to be owned by the instances of this type.	<code>AttributeType</code>

Name	Description	Type
<code>overriddenType</code>	The <code>AttributeType</code> that this attribute ownership overrides, if applicable.	<code>AttributeType</code>
<code>annotations</code>	Adds annotations to the ownership.	<code>Annotation[]</code>

Returns

`Promise<void>`

Code examples

```
thingType.setOwns(transaction, attributeType)
thingType.setOwns(transaction, attributeType, overriddenType,
[Annotation.KEY])
```

setPlays

```
setPlays(transaction, role): Promise<void>
```

Allows the instances of this `ThingType` to play the given role.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>role</code>	The role to be played by the instances of this type	<code>RoleType</code>

Returns

`Promise<void>`

Code examples

```
thingType.setPlays(transaction, role) thingType.setPlays(transaction, role,
overriddenType)
```

setPlays

```
setPlays(transaction, role, overriddenType): Promise<void>
```

Allows the instances of this `ThingType` to play the given role.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>role</code>	The role to be played by the instances of this type	<code>RoleType</code>
<code>overriddenType</code>	The role type that this role overrides, if applicable	<code>RoleType</code>

Returns

```
Promise<void>
```

Code examples

```
thingType.setPlays(transaction, role) thingType.setPlays(transaction, role,
overriddenType)
```

setSupertype

```
setSupertype(transaction, superEntityType): Promise<void>
```

Input parameters

Name	Description	Type
------	-------------	------

Name	Description	Type
<code>transaction</code>		<code>TypeDBTransaction</code>
<code>superEntityType</code>		<code>EntityType</code>

Returns

`Promise<void>`

toJSONRecord

```
toJSONRecord(): Record<string, string | number | boolean>
```

Retrieves the concept as JSON.

Returns

`Record<string, string | number | boolean>`

Code examples

```
concept.toJSONRecord()
```

unsetAbstract

```
unsetAbstract(transaction): Promise<void>
```

Set a `ThingType` to be non-abstract, meaning it can have instances.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<void>`

Code examples

```
thingType.unsetAbstract(transaction)
```

unsetOwns

```
unsetOwns(transaction, attributeType): Promise<void>
```

Disallows the instances of this `ThingType` from owning the given `AttributeType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeType</code>	The <code>AttributeType</code> to not be owned by the type.	<code>AttributeType</code>

Returns

`Promise<void>`

Code examples

```
thingType.unsetOwns(transaction, attributeType)
```

unsetPlays

```
unsetPlays(transaction, role): Promise<void>
```

Disallows the instances of this `ThingType` from playing the given role.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Name	Description	Type
<code>role</code>	The role to not be played by the instances of this type.	<code>RoleType</code>

Returns

`Promise<void>`

Code examples

```
thingType.unsetPlays(transaction, role)
```

RelationType

Supertypes:

- `ThingType`

Fields

Name	Type	Description
<code>abstract</code>	<code>boolean</code>	Whether the type is prevented from having data instances (i.e., abstract).
<code>label</code>	<code>Label</code>	The unique label of the type.
<code>root</code>	<code>boolean</code>	Whether the type is a root type.

asAttribute

```
asAttribute(): Attribute
```

Casts the concept to `Attribute`.

Returns

`Attribute`

Code examples

```
concept.asAttribute()
```

asAttributeType

```
asAttributeType(): AttributeType
```

Casts the concept to `AttributeType`.

Returns

`AttributeType`

Code examples

```
concept.asAttributeType()
```

asEntity

```
asEntity(): Entity
```

Casts the concept to `Entity`.

Returns

`Entity`

Code examples

```
concept.asEntity()
```

asEntityType

```
asEntityType(): EntityType
```

Casts the concept to `EntityType`.

Returns

`EntityType`

Code examples

```
concept.asEntityType()
```

asRelation

```
asRelation(): Relation
```

Casts the concept to `Relation`.

Returns

`Relation`

Code examples

```
concept.asRelation()
```

asRelationType

```
asRelationType(): RelationType
```

Casts the concept to `RelationType`.

Returns

`RelationType`

Code examples

```
concept.asRelationType()
```

asRoleType

```
asRoleType(): RoleType
```

Casts the concept to `RoleType`.

Returns

`RoleType`

Code examples

```
concept.asRoleType()
```

asThing

```
asThing(): Thing
```

Casts the concept to `Thing`.

Returns

`Thing`

Code examples

```
concept.asThing()
```

asThingType

```
asThingType(): ThingType
```

Casts the concept to `ThingType`.

Returns

`ThingType`

Code examples

```
concept.asThingType()
```

asType

```
asType(): Type
```

Casts the concept to `Type`.

Returns

`Type`

Code examples

```
concept.asType()
```

asValue

```
asValue(): Value
```

Casts the concept to `Value`.

Returns

`Value`

Code examples

```
concept.asValue()
```

create

```
create(transaction): Promise<Relation>
```

Creates and returns an instance of this `RelationType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<Relation>`

Code examples

```
relationType.create(transaction)
```

delete

```
delete(transaction): Promise<void>
```

Deletes this type from the database.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

```
Promise<void>
```

Code examples

```
type.delete(transaction)
```

equals

```
equals(concept): boolean
```

Checks if this concept is equal to the argument `concept`.

Input parameters

Name	Description	Type
<code>concept</code>	The concept to compare to.	<code>Concept</code>

Returns

```
boolean
```

getInstances

```
getInstances(transaction): Stream<Relation>
```

Retrieves all direct and indirect `Thing` objects that are instances of this `ThingType`.

Equivalent to `getInstances(transaction, Transitivity.TRANSITIVE)`

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<Relation>`

Code examples

```
thingType.getInstances(transaction)
```

getInstances

```
getInstances(transaction, transitivity): Stream<Relation>
```

Retrieves all direct and indirect (or direct only) `Thing` objects that are instances of this `ThingType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>transitivity</code>	<code>Transitivity.TRANSITIVE</code> for direct and indirect instances, <code>Transitivity.EXPLICIT</code> for direct instances only	<code>Transitivity</code>

Returns

`Stream<Relation>`

Code examples

```
thingType.getInstances(transaction, Transitivity.EXPLICIT)
```

getOwns

```
getOwns(transaction): Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

```
Stream<AttributeType>
```

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,  
Transitivity.EXPLICIT, [Annotation.KEY])
```

getOwns

```
getOwns(transaction, valueType): Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Name	Description	Type
<code>valueType</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>ValueType</code>

Returns

`Stream<AttributeType>`

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT, [Annotation.KEY])
```

getOwns

```
getOwns(transaction, annotations): Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>annotations</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>Annotation[]</code>

Returns

`Stream<AttributeType>`

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT, [Annotation.KEY])
```

getOwns

```
getOwns(transaction, valueType, annotations): Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>valueType</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>ValueType</code>
<code>annotations</code>	Only retrieve attribute types owned with annotations.	<code>Annotation[]</code>

Returns

```
Stream<AttributeType>
```

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT, [Annotation.KEY])
```

getOwns

```
getOwns(transaction, transitivity): Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Name	Description	Type
<code>transitivity</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>Transitivity</code>

Returns

`Stream<AttributeType>`

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT,[Annotation.KEY])
```

getOwns

```
getOwns(transaction, valueType, transitivity): Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>valueType</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>ValueType</code>
<code>transitivity</code>	Only retrieve attribute types owned with annotations.	<code>Transitivity</code>

Returns

`Stream<AttributeType>`

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT,[Annotation.KEY])
```

getOwns

```
getOwns(transaction, annotations, transitivity): Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>annotations</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>Annotation[]</code>
<code>transitivity</code>	Only retrieve attribute types owned with annotations.	<code>Transitivity</code>

Returns

```
Stream<AttributeType>
```

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT,[Annotation.KEY])
```

getOwns

```
getOwns(transaction, valueType, annotations, transitivity):
Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>valueType</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>ValueType</code>
<code>annotations</code>	Only retrieve attribute types owned with annotations.	<code>Annotation[]</code>
<code>transitivity</code>	<code>Transitivity.TRANSITIVE</code> for direct and inherited ownership, <code>Transitivity.EXPLICIT</code> for direct ownership only	<code>Transitivity</code>

Returns

`Stream<AttributeType>`

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT, [Annotation.KEY])
```

getOwnsOverridden

```
getOwnsOverridden(transaction, attributeType): Promise<AttributeType>
```

Retrieves an `AttributeType`, ownership of which is overridden for this `ThingType` by a given `attribute_type`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Name	Description	Type
<code>attributeType</code>	The <code>AttributeType</code> that overrides requested <code>AttributeType</code>	<code>AttributeType</code>

Returns

`Promise<AttributeType>`

Code examples

```
thingType.getOwnsOverridden(transaction, attributeType)
```

getPlays

```
getPlays(transaction): Stream<RoleType>
```

Retrieves all direct and inherited (or direct only) roles that are allowed to be played by the instances of this `ThingType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<RoleType>`

Code examples

```
thingType.getPlays(transaction) thingType.getPlays(transaction,
Transitivity.EXPLICIT)
```

getPlays

```
getPlays(transaction, transitivity): Stream<RoleType>
```

Retrieves all direct and inherited (or direct only) roles that are allowed to be played by the instances of this `ThingType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>transitivity</code>	<code>Transitivity.TRANSITIVE</code> for direct and indirect playing, <code>Transitivity.EXPLICIT</code> for direct playing only	<code>Transitivity</code>

Returns

`Stream<RoleType>`

Code examples

```
thingType.getPlays(transaction) thingType.getPlays(transaction,
Transitivity.EXPLICIT)
```

getPlaysOverridden

```
getPlaysOverridden(transaction, role): Promise<RoleType>
```

Retrieves a `RoleType` that is overridden by the given `role_type` for this `ThingType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>role</code>	The <code>RoleType</code> that overrides an inherited role	<code>RoleType</code>

Returns

Promise<RoleType>

Code examples

```
thingType.getPlaysOverridden(transaction, role)
```

getRelates

```
getRelates(transaction): Stream<RoleType>
```

RelationType#getRelates:(1)

Input parameters

Name	Description	Type
<code>transaction</code>		<code>TypeDBTransaction</code>

Returns

`Stream<RoleType>`

getRelates

```
getRelates(transaction, transitivity): Stream<RoleType>
```

Retrieves roles that this `RelationType` relates to directly or via inheritance. If `role_label` is given, returns a corresponding `RoleType` or `None`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Name	Description	Type
<code>transitivity</code>	<code>Transitivity.TRANSITIVE</code> for direct and inherited relates, <code>Transitivity.EXPLICIT</code> for direct relates only	<code>Transitivity</code>

Returns

`Stream<RoleType>`

Code examples

```
relationType.getRelates(transaction, roleLabel, transitivity)
```

getRelatesForRoleLabel

```
getRelatesForRoleLabel(transaction, roleLabel): Promise<RoleType>
```

Input parameters

Name	Description	Type
<code>transaction</code>		<code>TypeDBTransaction</code>
<code>roleLabel</code>		<code>string</code>

Returns

`Promise<RoleType>`

getRelatesOverridden

```
getRelatesOverridden(transaction, roleLabel): Promise<RoleType>
```

Retrieves a `RoleType` that is overridden by the role with the `role_label`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>roleLabel</code>	Label of the role that overrides an inherited role	<code>string</code>

Returns

`Promise<RoleType>`

Code examples

```
relationType.getRelatesOverridden(transaction, roleLabel)
```

getSubtypes

```
getSubtypes(transaction): Stream<RelationType>
```

Retrieves all direct and indirect subtypes of the `ThingType`. Equivalent to `getSubtypes(transaction, Transitivity.TRANSITIVE)`

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<RelationType>`

Code examples

```
thingType.getSubtypes(transaction)
```

getSubtypes

```
getSubtypes(transaction, transitivity): Stream<RelationType>
```

Retrieves all direct and indirect (or direct only) subtypes of the `ThingType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>transitivity</code>	<code>Transitivity.TRANSITIVE</code> for direct and indirect subtypes, <code>Transitivity.EXPLICIT</code> for direct subtypes only	<code>Transitivity</code>

Returns

```
Stream<RelationType>
```

Code examples

```
thingType.getSubtypes(transaction, Transitivity.EXPLICIT)
```

getSupertype

```
getSupertype(transaction): Promise<RelationType>
```

Retrieves the most immediate supertype of the `ThingType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

```
Promise<RelationType>
```

Code examples

```
thingType.getSupertype(transaction)
```

getSupertypes

```
getSupertypes(transaction): Stream<RelationType>
```

Retrieves all supertypes of the `ThingType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

```
Stream<RelationType>
```

Code examples

```
thingType.getSupertypes(transaction)
```

getSyntax

```
getSyntax(transaction): Promise<string>
```

Produces a pattern for creating this `ThingType` in a `define` query.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

```
Promise<string>
```

Code examples

```
thingType.getSyntax(transaction)
```

isAttribute

```
isAttribute(): boolean
```

Checks if the concept is an `Attribute`.

Returns

```
boolean
```

Code examples

```
concept.isAttribute()
```

isAttributeType

```
isAttributeType(): boolean
```

Checks if the concept is an `AttributeType`.

Returns

```
boolean
```

Code examples

```
concept.isAttributeType()
```

isDeleted

```
isDeleted(transaction): Promise<boolean>
```

Check if the concept has been deleted

Input parameters

Name	Description	Type
------	-------------	------

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<boolean>`

isEntity

```
isEntity(): boolean
```

Checks if the concept is an `Entity`.

Returns

`boolean`

Code examples

```
concept.isEntity()
```

isEntityType

```
isEntityType(): boolean
```

Checks if the concept is an `EntityType`.

Returns

`boolean`

Code examples

```
concept.isEntityType()
```

isRelation

```
isRelation(): boolean
```

Checks if the concept is a `Relation`.

Returns

`boolean`

Code examples

```
concept.isRelation()
```

isRelationType

```
isRelationType(): boolean
```

Checks if the concept is a `RelationType`.

Returns

`boolean`

Code examples

```
concept.isRelationType()
```

isRoleType

```
isRoleType(): boolean
```

Checks if the concept is a `RoleType`.

Returns

`boolean`

Code examples

```
concept.isRoleType()
```

isThing

```
isThing(): boolean
```

Checks if the concept is a `Thing`.

Returns

`boolean`

Code examples

```
concept.isThing()
```

isThingType

```
isThingType(): boolean
```

Checks if the concept is a `ThingType`.

Returns

`boolean`

Code examples

```
concept.isThingType()
```

isType

```
isType(): boolean
```

Checks if the concept is a `Type`.

Returns

`boolean`

Code examples

```
concept.isType()
```

isValue

```
isValue(): boolean
```


Checks if the concept is a `Value`.

Returns

`boolean`

Code examples

```
concept.isValue()
```

setAbstract

```
setAbstract(transaction): Promise<void>
```

Set a `ThingType` to be abstract, meaning it cannot have instances.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<void>`

Code examples

```
thingType.setAbstract(transaction)
```

setLabel

```
setLabel(transaction, label): Promise<void>
```

Renames the label of the type. The new label must remain unique.

Input parameters

Name	Description	Type
------	-------------	------

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>label</code>	The new <code>Label</code> to be given to the type.	<code>string</code>

Returns

`Promise<void>`

Code examples

```
type.setLabel(transaction, label)
```

setOwns

```
setOwns(transaction, attributeType): Promise<void>
```

Allows the instances of this `ThingType` to own the given `AttributeType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeType</code>	The <code>AttributeType</code> to be owned by the instances of this type.	<code>AttributeType</code>

Returns

`Promise<void>`

Code examples

```
thingType.setOwns(transaction, attributeType)
thingType.setOwns(transaction, attributeType, overriddenType,
[Annotation.KEY])
```

setOwns

```
setOwns(transaction, attributeType, annotations): Promise<void>
```

Allows the instances of this `ThingType` to own the given `AttributeType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeType</code>	The <code>AttributeType</code> to be owned by the instances of this type.	<code>AttributeType</code>
<code>annotations</code>	The <code>AttributeType</code> that this attribute ownership overrides, if applicable.	<code>Annotation[]</code>

Returns

```
Promise<void>
```

Code examples

```
thingType.setOwns(transaction, attributeType)
thingType.setOwns(transaction, attributeType, overriddenType,
[Annotation.KEY])
```

setOwns

```
setOwns(transaction, attributeType, overriddenType): Promise<void>
```

Allows the instances of this `ThingType` to own the given `AttributeType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Name	Description	Type
<code>attributeType</code>	The <code>AttributeType</code> to be owned by the instances of this type.	<code>AttributeType</code>
<code>overriddenType</code>	The <code>AttributeType</code> that this attribute ownership overrides, if applicable.	<code>AttributeType</code>

Returns

`Promise<void>`

Code examples

```
thingType.setOwns(transaction, attributeType)
thingType.setOwns(transaction, attributeType, overriddenType,
[Annotation.KEY])
```

setOwns

```
setOwns(transaction, attributeType, overriddenType, annotations):
Promise<void>
```

Allows the instances of this `ThingType` to own the given `AttributeType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeType</code>	The <code>AttributeType</code> to be owned by the instances of this type.	<code>AttributeType</code>
<code>overriddenType</code>	The <code>AttributeType</code> that this attribute ownership overrides, if applicable.	<code>AttributeType</code>

Name	Description	Type
<code>annotations</code>	Adds annotations to the ownership.	<code>Annotation[]</code>

Returns

`Promise<void>`

Code examples

```
thingType.setOwns(transaction, attributeType)
thingType.setOwns(transaction, attributeType, overriddenType,
[Annotation.KEY])
```

setPlays

```
setPlays(transaction, role): Promise<void>
```

Allows the instances of this `ThingType` to play the given role.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>role</code>	The role to be played by the instances of this type	<code>RoleType</code>

Returns

`Promise<void>`

Code examples

```
thingType.setPlays(transaction, role) thingType.setPlays(transaction, role,
overriddenType)
```

setPlays

```
setPlays(transaction, role, overriddenType): Promise<void>
```

Allows the instances of this `ThingType` to play the given role.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>role</code>	The role to be played by the instances of this type	<code>RoleType</code>
<code>overriddenType</code>	The role type that this role overrides, if applicable	<code>RoleType</code>

Returns

```
Promise<void>
```

Code examples

```
thingType.setPlays(transaction, role) thingType.setPlays(transaction, role, overriddenType)
```

setRelates

```
setRelates(transaction, roleLabel, overriddenLabel?): Promise<void>
```

Sets the new role that this `RelationType` relates to. If we are setting an overriding type this way, we have to also pass the overridden type as a second argument.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Name	Description	Type
<code>roleLabel</code>	The new role for the <code>RelationType</code> to relate to	<code>string</code>
<code>overriddenLabel</code>	The label being overridden, if applicable	<code>string</code>

Returns

`Promise<void>`

Code examples

```
relationType.setRelates(transaction, roleLabel)
relationType.setRelates(transaction, roleLabel, overriddenLabel)
```

setSupertype

```
setSupertype(transaction, type): Promise<void>
```

Input parameters

Name	Description	Type
<code>transaction</code>		<code>TypeDBTransaction</code>
<code>type</code>		<code>RelationType</code>

Returns

`Promise<void>`

toJSONRecord

```
toJSONRecord(): Record<string, string | number | boolean>
```

Retrieves the concept as JSON.

Returns

```
Record<string, string | number | boolean>
```

Code examples

```
concept.toJSONRecord()
```

unsetAbstract

```
unsetAbstract(transaction): Promise<void>
```

Set a `ThingType` to be non-abstract, meaning it can have instances.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

```
Promise<void>
```

Code examples

```
thingType.unsetAbstract(transaction)
```

unsetOwns

```
unsetOwns(transaction, attributeType): Promise<void>
```

Disallows the instances of this `ThingType` from owning the given `AttributeType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Name	Description	Type
<code>attributeType</code>	The <code>AttributeType</code> to not be owned by the type.	<code>AttributeType</code>

Returns

`Promise<void>`

Code examples

```
thingType.unsetOwns(transaction, attributeType)
```

unsetPlays

```
unsetPlays(transaction, role): Promise<void>
```

Disallows the instances of this `ThingType` from playing the given role.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>role</code>	The role to not be played by the instances of this type.	<code>RoleType</code>

Returns

`Promise<void>`

Code examples

```
thingType.unsetPlays(transaction, role)
```

unsetRelates

```
unsetRelates(transaction, roleLabel): Promise<void>
```

Disallows this `RelationType` from relating to the given role.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>roleLabel</code>	The role to not relate to the relation type.	<code>string</code>

Returns

```
Promise<void>
```

Code examples

```
relationType.unsetRelates(transaction, roleLabel)
```

RoleType

Supertypes:

- `Type`

Roles are special internal types used by relations. We can not create an instance of a role in a database. But we can set an instance of another type (role player) to play a role in a particular instance of a relation type. Roles allow a schema to enforce logical constraints on types of role players.

Fields

Name	Type	Description
<code>abstract</code>	<code>boolean</code>	Whether the type is prevented from having data instances (i.e., abstract).
<code>label</code>	<code>Label</code>	The unique label of the type.

Name	Type	Description
<code>root</code>	<code>boolean</code>	Whether the type is a root type.

asAttribute

```
asAttribute(): Attribute
```

Casts the concept to `Attribute`.

Returns

`Attribute`

Code examples

```
concept.asAttribute()
```

asAttributeType

```
asAttributeType(): AttributeType
```

Casts the concept to `AttributeType`.

Returns

`AttributeType`

Code examples

```
concept.asAttributeType()
```

asEntity

```
asEntity(): Entity
```

Casts the concept to `Entity`.

Returns

Entity

Code examples

```
concept.asEntity()
```

asEntityType

```
asEntityType(): EntityType
```

Casts the concept to `EntityType`.

Returns

EntityType

Code examples

```
concept.asEntityType()
```

asRelation

```
asRelation(): Relation
```

Casts the concept to `Relation`.

Returns

Relation

Code examples

```
concept.asRelation()
```

asRelationType

```
asRelationType(): RelationType
```

Casts the concept to `RelationType`.

Returns

RelationType

Code examples

```
concept.asRelationType()
```

asRoleType

```
asRoleType(): RoleType
```

Casts the concept to **RoleType**.

Returns

RoleType

Code examples

```
concept.asRoleType()
```

asThing

```
asThing(): Thing
```

Casts the concept to **Thing**.

Returns

Thing

Code examples

```
concept.asThing()
```

asThingType

```
asThingType(): ThingType
```

Casts the concept to **ThingType**.

Returns

ThingType

Code examples

```
concept.asThingType()
```

asType

```
asType(): Type
```

Casts the concept to **Type**.

Returns

Type

Code examples

```
concept.asType()
```

asValue

```
asValue(): Value
```

Casts the concept to **Value**.

Returns

Value

Code examples

```
concept.asValue()
```

delete

```
delete(transaction): Promise<void>
```

Deletes this type from the database.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<void>`

Code examples

```
type.delete(transaction)
```

equals

```
equals(concept): boolean
```

Checks if this concept is equal to the argument `concept`.

Input parameters

Name	Description	Type
<code>concept</code>	The concept to compare to.	<code>Concept</code>

Returns

`boolean`

getPlayerInstances

```
getPlayerInstances(transaction): Stream<Thing>
```

Retrieves the `Thing` instances that play this role.

Input parameters

Name	Description	Type
------	-------------	------

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<Thing>`

Code examples

```
roleType.getPlayerInstances(transaction, transitivity)
```

getPlayerInstances

```
getPlayerInstances(transaction, transitivity): Stream<Thing>
```

Retrieves the `Thing` instances that play this role.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>transitivity</code>	<code>Transitivity.TRANSITIVE</code> for direct and indirect playing, <code>Transitivity.EXPLICIT</code> for direct playing only	<code>Transitivity</code>

Returns

`Stream<Thing>`

Code examples

```
roleType.getPlayerInstances(transaction, transitivity)
```

getPlayerTypes


```
getPlayerTypes(transaction): Stream<ThingType>
```

Retrieves the `ThingType`s whose instances play this role.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

```
Stream<ThingType>
```

Code examples

```
roleType.getPlayerTypes(transaction, transitivity)
```

getPlayerTypes

```
getPlayerTypes(transaction, transitivity): Stream<ThingType>
```

Retrieves the `ThingType`s whose instances play this role.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>transitivity</code>	<code>Transitivity.TRANSITIVE</code> for direct and indirect playing, <code>Transitivity.EXPLICIT</code> for direct playing only	<code>Transitivity</code>

Returns

```
Stream<ThingType>
```

Code examples

```
roleType.getPlayerTypes(transaction, transitivity)
```

getRelationInstances

```
getRelationInstances(transaction): Stream<Relation>
```

Retrieves the `Relation` instances that this role is related to.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

```
Stream<Relation>
```

Code examples

```
roleType.getRelationInstances(transaction, transitivity)
```

getRelationInstances

```
getRelationInstances(transaction, transitivity): Stream<Relation>
```

Retrieves the `Relation` instances that this role is related to.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Name	Description	Type
<code>transitivity</code>	<code>Transitivity.TRANSITIVE</code> for direct and indirect relation, <code>Transitivity.EXPLICIT</code> for direct relation only	<code>Transitivity</code>

Returns

`Stream<Relation>`

Code examples

```
roleType.getRelationInstances(transaction, transitivity)
```

getRelationType

```
getRelationType(transaction): Promise<RelationType>
```

Retrieves the `RelationType` that this role is directly related to.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<RelationType>`

Code examples

```
roleType.getRelationType(transaction)
```

getRelationTypes

```
getRelationTypes(transaction): Stream<RelationType>
```

Retrieves `RelationType`s that this role is related to (directly or indirectly).

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<RelationType>`

Code examples

```
roleType.getRelationTypes(transaction)
```

getSubtypes

```
getSubtypes(transaction): Stream<RoleType>
```

Retrieves all direct and indirect (or direct only) subtypes of the type.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<RoleType>`

Code examples

```
type.getSubtypes(transaction) type.getSubtypes(transaction,  
Transitivity.EXPLICIT)
```

getSupertype

```
getSupertype(transaction): Promise<RoleType>
```

Retrieves the most immediate supertype of the type.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<RoleType>`

Code examples

```
type.getSupertype(transaction)
```

getSupertypes

```
getSupertypes(transaction): Stream<RoleType>
```

Retrieves all supertypes of the type.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<RoleType>`

Code examples

```
type.getSupertypes(transaction)
```

isAttribute

```
isAttribute(): boolean
```

Checks if the concept is an `Attribute`.

Returns

`boolean`

Code examples

```
concept.isAttribute()
```

isAttributeType

```
isAttributeType(): boolean
```

Checks if the concept is an `AttributeType`.

Returns

`boolean`

Code examples

```
concept.isAttributeType()
```

isDeleted

```
isDeleted(transaction): Promise<boolean>
```

Check if the concept has been deleted

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<boolean>`

isEntity

```
isEntity(): boolean
```

Checks if the concept is an `Entity`.

Returns

```
boolean
```

Code examples

```
concept.isEntity()
```

isEntityType

```
isEntityType(): boolean
```

Checks if the concept is an `EntityType`.

Returns

```
boolean
```

Code examples

```
concept.isEntityType()
```

isRelation

```
isRelation(): boolean
```

Checks if the concept is a `Relation`.

Returns

```
boolean
```

Code examples

```
concept.isRelation()
```

isRelationType

```
isRelationType(): boolean
```

Checks if the concept is a `RelationType`.

Returns

```
boolean
```

Code examples

```
concept.isRelationType()
```

isRoleType

```
isRoleType(): boolean
```

Checks if the concept is a `RoleType`.

Returns

```
boolean
```

Code examples

```
concept.isRoleType()
```

isThing

```
isThing(): boolean
```

Checks if the concept is a `Thing`.

Returns

```
boolean
```

Code examples

```
concept.isThing()
```

isThingType


```
isThingType(): boolean
```

Checks if the concept is a `ThingType`.

Returns

```
boolean
```

Code examples

```
concept.isThingType()
```

isType

```
isType(): boolean
```

Checks if the concept is a `Type`.

Returns

```
boolean
```

Code examples

```
concept.isType()
```

isValue

```
isValue(): boolean
```

Checks if the concept is a `Value`.

Returns

```
boolean
```

Code examples

```
concept.isValue()
```

setLabel

```
setLabel(transaction, label): Promise<void>
```

Renames the label of the type. The new label must remain unique.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>label</code>	The new <code>Label</code> to be given to the type.	<code>string</code>

Returns

```
Promise<void>
```

Code examples

```
type.setLabel(transaction, label)
```

toJSONRecord

```
toJSONRecord(): Record<string, string | number | boolean>
```

Retrieves the concept as JSON.

Returns

```
Record<string, string | number | boolean>
```

Code examples

```
concept.toJSONRecord()
```

AttributeType

Supertypes:

- `ThingType`

Attribute types represent properties that other types can own. Attribute types have a value

type. This value type is fixed and unique for every given instance of the attribute type. Other types can own an attribute type. That means that instances of these other types can own an instance of this attribute type. This usually means that an object in our domain has a property with the matching value. Multiple types can own the same attribute type, and different instances of the same type or different types can share ownership of the same attribute instance.

Fields

Name	Type	Description
<code>abstract</code>	<code>boolean</code>	Whether the type is prevented from having data instances (i.e., abstract).
<code>label</code>	<code>Label</code>	The unique label of the type.
<code>root</code>	<code>boolean</code>	Whether the type is a root type.
<code>valueType</code>	<code>ValueType</code>	The ValueType of this AttributeType.

asAttribute

```
asAttribute(): Attribute
```

Casts the concept to `Attribute`.

Returns

`Attribute`

Code examples

```
concept.asAttribute()
```

asAttributeType

```
asAttributeType(): AttributeType
```

Casts the concept to `AttributeType`.

Returns

`AttributeType`

Code examples

```
concept.asAttributeType()
```

asEntity

```
asEntity(): Entity
```

Casts the concept to `Entity`.

Returns

`Entity`

Code examples

```
concept.asEntity()
```

asEntityType

```
asEntityType(): EntityType
```

Casts the concept to `EntityType`.

Returns

`EntityType`

Code examples

```
concept.asEntityType()
```

asRelation

```
asRelation(): Relation
```

Casts the concept to `Relation`.

Returns

Relation

Code examples

```
concept.asRelation()
```

asRelationType

```
asRelationType(): RelationType
```

Casts the concept to **RelationType**.

Returns

RelationType

Code examples

```
concept.asRelationType()
```

asRoleType

```
asRoleType(): RoleType
```

Casts the concept to **RoleType**.

Returns

RoleType

Code examples

```
concept.asRoleType()
```

asThing

```
asThing(): Thing
```

Casts the concept to **Thing**.

Returns

Thing

Code examples

```
concept.asThing()
```

asThingType

```
asThingType(): ThingType
```

Casts the concept to **ThingType**.

Returns

ThingType

Code examples

```
concept.asThingType()
```

asType

```
asType(): Type
```

Casts the concept to **Type**.

Returns

Type

Code examples

```
concept.asType()
```

asValue

```
asValue(): Value
```

Casts the concept to **Value**.

Returns

`Value`

Code examples

```
concept.asValue()
```

delete

```
delete(transaction): Promise<void>
```

Deletes this type from the database.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<void>`

Code examples

```
type.delete(transaction)
```

equals

```
equals(concept): boolean
```

Checks if this concept is equal to the argument `concept`.

Input parameters

Name	Description	Type
<code>concept</code>	The concept to compare to.	<code>Concept</code>

Returns

`boolean`

get

```
get(transaction, value): Promise<Attribute>
```

Retrieves an `Attribute` of this `AttributeType` with the given value if such `Attribute` exists. Otherwise, returns `None`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>value</code>	<code>Attribute</code> 's value	<code>Value</code>

Returns

`Promise<Attribute>`

Code examples

```
attribute = attributeType.get(transaction, value)
```

getBoolean

```
getBoolean(transaction, value): Promise<Attribute>
```

Retrieves an `Attribute` of this `AttributeType` with the given value if such `Attribute` exists. Otherwise, returns `None`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Name	Description	Type
<code>value</code>	<code>Attribute</code> 's value	<code>boolean</code>

Returns

`Promise<Attribute>`

Code examples

```
attribute = attributeType.get(transaction, value)
```

getDateTime

```
getDateTime(transaction, value): Promise<Attribute>
```

Retrieves an `Attribute` of this `AttributeType` with the given value if such `Attribute` exists. Otherwise, returns `None`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>value</code>	<code>Attribute</code> 's value	<code>Date</code>

Returns

`Promise<Attribute>`

Code examples

```
attribute = attributeType.get(transaction, value)
```

getDouble

```
getDouble(transaction, value): Promise<Attribute>
```

Retrieves an `Attribute` of this `AttributeType` with the given value if such `Attribute` exists. Otherwise, returns `None`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>value</code>	<code>Attribute</code> 's value	<code>number</code>

Returns

`Promise<Attribute>`

Code examples

```
attribute = attributeType.get(transaction, value)
```

getInstances

```
getInstances(transaction, transitivity): Stream<Attribute>
```

Retrieves all direct and indirect (or direct only) `Attributes` that are instances of this `AttributeType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>transitivity</code>	<code>Transitivity.TRANSITIVE</code> for direct and indirect subtypes, <code>Transitivity.EXPLICIT</code> for direct subtypes only	<code>Transitivity</code>

Returns

`Stream<Attribute>`

Code examples

```
attributeType.getInstances(transaction)
attributeType.getInstances(transaction, Transitivity.EXPLICIT)
```

getInstances

```
getInstances(transaction): Stream<Attribute>
```

Retrieves all direct and indirect (or direct only) **Thing** objects that are instances of this **ThingType**.

Input parameters

Name	Description	Type
transaction	The current transaction	TypeDBTransaction

Returns

Stream<Attribute>

Code examples

```
thingType.getInstances(transaction, Transitivity.EXPLICIT)
```

getLong

```
getLong(transaction, value): Promise<Attribute>
```

Retrieves an **Attribute** of this **AttributeType** with the given value if such **Attribute** exists. Otherwise, returns **None**.

Input parameters

Name	Description	Type
transaction	The current transaction	TypeDBTransaction

Name	Description	Type
<code>value</code>	<code>Attribute</code> 's value	<code>number</code>

Returns

`Promise<Attribute>`

Code examples

```
attribute = attributeType.get(transaction, value)
```

getOwners

```
getOwners(transaction, annotations, transitivity): Stream<ThingType>
```

Retrieve all `Things` that own an attribute of this `AttributeType`. Optionally, filtered by `Annotation`s.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>annotations</code>	Only retrieve <code>ThingTypes</code> that have an attribute of this <code>AttributeType</code> with all given <code>Annotation</code> s	<code>Annotation[]</code>
<code>transitivity</code>	<code>Transitivity.TRANSITIVE</code> for direct and inherited ownership, <code>Transitivity.EXPLICIT</code> for direct ownership only	<code>Transitivity</code>

Returns

`Stream<ThingType>`

Code examples

```
attributeType.getOwners(transaction) attributeType.getOwners(transaction,
[Annotation.UNIQUE]) attributeType.getOwners(transaction,
Transitivity.TRANSITIVE) attributeType.getOwners(transaction,
[Annotation.UNIQUE], Transitivity.TRANSITIVE)
```

getOwners

```
getOwners(transaction): Stream<ThingType>
```

Input parameters

Name	Description	Type
<code>transaction</code>		<code>TypeDBTransaction</code>

Returns

```
Stream<ThingType>
```

getOwners

```
getOwners(transaction, annotations): Stream<ThingType>
```

Input parameters

Name	Description	Type
<code>transaction</code>		<code>TypeDBTransaction</code>
<code>annotations</code>		<code>Annotation[]</code>

Returns

```
Stream<ThingType>
```

getOwners

```
getOwners(transaction, transitivity): Stream<ThingType>
```

Input parameters

Name	Description	Type
<code>transaction</code>		<code>TypeDBTransaction</code>
<code>transitivity</code>		<code>Transitivity</code>

Returns

```
Stream<ThingType>
```

getOwns

```
getOwns(transaction): Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

```
Stream<AttributeType>
```

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,  
Transitivity.EXPLICIT,[Annotation.KEY])
```

getOwns

```
getOwns(transaction, valueType): Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>valueType</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>ValueType</code>

Returns

```
Stream<AttributeType>
```

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,  
Transitivity.EXPLICIT, [Annotation.KEY])
```

getOwns

```
getOwns(transaction, annotations): Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>annotations</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>Annotation[]</code>

Returns

`Stream<AttributeType>`

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT,[Annotation.KEY])
```

getOwns

```
getOwns(transaction, valueType, annotations): Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>valueType</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>ValueType</code>
<code>annotations</code>	Only retrieve attribute types owned with annotations.	<code>Annotation[]</code>

Returns

`Stream<AttributeType>`

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT,[Annotation.KEY])
```

getOwns

```
getOwns(transaction, transitivity): Stream<AttributeType>
```


Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>transitivity</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>Transitivity</code>

Returns

`Stream<AttributeType>`

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT, [Annotation.KEY])
```

getOwns

```
getOwns(transaction, valueType, transitivity): Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>valueType</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>ValueType</code>
<code>transitivity</code>	Only retrieve attribute types owned with annotations.	<code>Transitivity</code>

Returns

`Stream<AttributeType>`

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT,[Annotation.KEY])
```

getOwns

```
getOwns(transaction, annotations, transitivity): Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>annotations</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>Annotation[]</code>
<code>transitivity</code>	Only retrieve attribute types owned with annotations.	<code>Transitivity</code>

Returns

`Stream<AttributeType>`

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,
Transitivity.EXPLICIT,[Annotation.KEY])
```

getOwns

```
getOwns(transaction, valueType, annotations, transitivity):  
Stream<AttributeType>
```

Retrieves `AttributeType` that the instances of this `ThingType` are allowed to own directly or via inheritance.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>valueType</code>	If specified, only attribute types of this <code>ValueType</code> will be retrieved.	<code>ValueType</code>
<code>annotations</code>	Only retrieve attribute types owned with annotations.	<code>Annotation[]</code>
<code>transitivity</code>	<code>Transitivity.TRANSITIVE</code> for direct and inherited ownership, <code>Transitivity.EXPLICIT</code> for direct ownership only	<code>Transitivity</code>

Returns

```
Stream<AttributeType>
```

Code examples

```
thingType.getOwns(transaction) thingType.getOwns(transaction, valueType,  
Transitivity.EXPLICIT,[Annotation.KEY])
```

getOwnsOverridden

```
getOwnsOverridden(transaction, attributeType): Promise<AttributeType>
```

Retrieves an `AttributeType`, ownership of which is overridden for this `ThingType` by a given `attribute_type`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeType</code>	The <code>AttributeType</code> that overrides requested <code>AttributeType</code>	<code>AttributeType</code>

Returns

`Promise<AttributeType>`

Code examples

```
thingType.getOwnsOverridden(transaction, attributeType)
```

getPlays

```
getPlays(transaction): Stream<RoleType>
```

Retrieves all direct and inherited (or direct only) roles that are allowed to be played by the instances of this `ThingType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<RoleType>`

Code examples

```
thingType.getPlays(transaction) thingType.getPlays(transaction,
Transitivity.EXPLICIT)
```

getPlays

```
getPlays(transaction, transitivity): Stream<RoleType>
```

Retrieves all direct and inherited (or direct only) roles that are allowed to be played by the instances of this `ThingType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>transitivity</code>	<code>Transitivity.TRANSITIVE</code> for direct and indirect playing, <code>Transitivity.EXPLICIT</code> for direct playing only	<code>Transitivity</code>

Returns

```
Stream<RoleType>
```

Code examples

```
thingType.getPlays(transaction) thingType.getPlays(transaction,  
Transitivity.EXPLICIT)
```

getPlaysOverridden

```
getPlaysOverridden(transaction, role): Promise<RoleType>
```

Retrieves a `RoleType` that is overridden by the given `role_type` for this `ThingType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Name	Description	Type
<code>role</code>	The <code>RoleType</code> that overrides an inherited role	<code>RoleType</code>

Returns

`Promise<RoleType>`

Code examples

```
thingType.getPlaysOverridden(transaction, role)
```

getRegex

```
getRegex(transaction): Promise<string>
```

Retrieves the regular expression that is defined for this `AttributeType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<string>`

Code examples

```
attributeType.getRegex(transaction)
```

getString

```
getString(transaction, value): Promise<Attribute>
```

Retrieves an `Attribute` of this `AttributeType` with the given value if such `Attribute` exists. Otherwise, returns `None`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>value</code>	<code>Attribute</code> 's value	<code>string</code>

Returns

`Promise<Attribute>`

Code examples

```
attribute = attributeType.get(transaction, value)
```

getSubtypes

```
getSubtypes(transaction): Stream<AttributeType>
```

Retrieves all direct and indirect subtypes of the `ThingType`. Equivalent to `getSubtypes(transaction, Transitivity.TRANSITIVE)`

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<AttributeType>`

Code examples

```
thingType.getSubtypes(transaction)
```

getSubtypes

```
getSubtypes(transaction, valueType): Stream<AttributeType>
```

Retrieves all direct and indirect (or direct only) subtypes of the `ThingType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>valueType</code>	<code>Transitivity.TRANSITIVE</code> for direct and indirect subtypes, <code>Transitivity.EXPLICIT</code> for direct subtypes only	<code>ValueType</code>

Returns

```
Stream<AttributeType>
```

Code examples

```
thingType.getSubtypes(transaction, Transitivity.EXPLICIT)
```

getSubtypes

```
getSubtypes(transaction, transitivity): Stream<AttributeType>
```

Input parameters

Name	Description	Type
<code>transaction</code>		<code>TypeDBTransaction</code>
<code>transitivity</code>		<code>Transitivity</code>

Returns

```
Stream<AttributeType>
```


getSubtypes

```
getSubtypes(transaction, valueType, transitivity): Stream<AttributeType>
```

Input parameters

Name	Description	Type
<code>transaction</code>		<code>TypeDBTransaction</code>
<code>valueType</code>		<code>ValueType</code>
<code>transitivity</code>		<code>Transitivity</code>

Returns

```
Stream<AttributeType>
```

getSupertype

```
getSupertype(transaction): Promise<AttributeType>
```

Retrieves the most immediate supertype of the `ThingType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

```
Promise<AttributeType>
```

Code examples

```
thingType.getSupertype(transaction)
```

getSupertypes

```
getSupertypes(transaction): Stream<AttributeType>
```

Retrieves all supertypes of the `ThingType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

```
Stream<AttributeType>
```

Code examples

```
thingType.getSupertypes(transaction)
```

getSyntax

```
getSyntax(transaction): Promise<string>
```

Produces a pattern for creating this `ThingType` in a `define` query.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

```
Promise<string>
```

Code examples

```
thingType.getSyntax(transaction)
```

isAttribute

```
isAttribute(): boolean
```

Checks if the concept is an `Attribute`.

Returns

```
boolean
```

Code examples

```
concept.isAttribute()
```

isAttributeType

```
isAttributeType(): boolean
```

Checks if the concept is an `AttributeType`.

Returns

```
boolean
```

Code examples

```
concept.isAttributeType()
```

isDeleted

```
isDeleted(transaction): Promise<boolean>
```

Check if the concept has been deleted

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

```
Promise<boolean>
```

isEntity

```
isEntity(): boolean
```

Checks if the concept is an `Entity`.

Returns

```
boolean
```

Code examples

```
concept.isEntity()
```

isEntityType

```
isEntityType(): boolean
```

Checks if the concept is an `EntityType`.

Returns

```
boolean
```

Code examples

```
concept.isEntityType()
```

isRelation

```
isRelation(): boolean
```

Checks if the concept is a `Relation`.

Returns

```
boolean
```

Code examples

```
concept.isRelation()
```

isRelationType

```
isRelationType(): boolean
```

Checks if the concept is a `RelationType`.

Returns

```
boolean
```

Code examples

```
concept.isRelationType()
```

isRoleType

```
isRoleType(): boolean
```

Checks if the concept is a `RoleType`.

Returns

```
boolean
```

Code examples

```
concept.isRoleType()
```

isThing

```
isThing(): boolean
```

Checks if the concept is a `Thing`.

Returns

```
boolean
```

Code examples

```
concept.isThing()
```

isThingType

```
isThingType(): boolean
```

Checks if the concept is a `ThingType`.

Returns

```
boolean
```

Code examples

```
concept.isThingType()
```

isType

```
isType(): boolean
```

Checks if the concept is a `Type`.

Returns

```
boolean
```

Code examples

```
concept.isType()
```

isValue

```
isValue(): boolean
```

Checks if the concept is a `Value`.

Returns

```
boolean
```

Code examples

```
concept.isValue()
```

put

```
put(transaction, value): Promise<Attribute>
```

Adds and returns an `Attribute` of this `AttributeType` with the given value.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>value</code>	New <code>Attribute</code> 's value	<code>Value</code>

Returns

`Promise<Attribute>`

Code examples

```
attribute = attributeType.put(transaction, value)
```

putBoolean

```
putBoolean(transaction, value): Promise<Attribute>
```

Adds and returns an `Attribute` of this `AttributeType` with the given value.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>value</code>	New <code>Attribute</code> 's value	<code>boolean</code>

Returns

`Promise<Attribute>`

Code examples

```
attribute = attributeType.put(transaction, value)
```

putDateTime

```
putDateTime(transaction, value): Promise<Attribute>
```

Adds and returns an `Attribute` of this `AttributeType` with the given value.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>value</code>	New <code>Attribute</code> 's value	<code>Date</code>

Returns

```
Promise<Attribute>
```

Code examples

```
attribute = attributeType.put(transaction, value)
```

putDouble

```
putDouble(transaction, value): Promise<Attribute>
```

Adds and returns an `Attribute` of this `AttributeType` with the given value.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Name	Description	Type
<code>value</code>	New <code>Attribute</code> 's value	<code>number</code>

Returns

`Promise<Attribute>`

Code examples

```
attribute = attributeType.put(transaction, value)
```

putLong

```
putLong(transaction, value): Promise<Attribute>
```

Adds and returns an `Attribute` of this `AttributeType` with the given value.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>value</code>	New <code>Attribute</code> 's value	<code>number</code>

Returns

`Promise<Attribute>`

Code examples

```
attribute = attributeType.put(transaction, value)
```

putString

```
putString(transaction, value): Promise<Attribute>
```

Adds and returns an `Attribute` of this `AttributeType` with the given value.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>value</code>	New <code>Attribute</code> 's value	<code>string</code>

Returns

`Promise<Attribute>`

Code examples

```
attribute = attributeType.put(transaction, value)
```

setAbstract

```
setAbstract(transaction): Promise<void>
```

Set a `ThingType` to be abstract, meaning it cannot have instances.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<void>`

Code examples

```
thingType.setAbstract(transaction)
```

setLabel

```
setLabel(transaction, label): Promise<void>
```

Renames the label of the type. The new label must remain unique.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>label</code>	The new <code>Label</code> to be given to the type.	<code>string</code>

Returns

`Promise<void>`

Code examples

```
type.setLabel(transaction, label)
```

setOwns

```
setOwns(transaction, attributeType): Promise<void>
```

Allows the instances of this `ThingType` to own the given `AttributeType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeType</code>	The <code>AttributeType</code> to be owned by the instances of this type.	<code>AttributeType</code>

Returns

`Promise<void>`

Code examples

```
thingType.setOwns(transaction, attributeType)
thingType.setOwns(transaction, attributeType, overriddenType,
[Annotation.KEY])
```

setOwns

```
setOwns(transaction, attributeType, annotations): Promise<void>
```

Allows the instances of this `ThingType` to own the given `AttributeType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeType</code>	The <code>AttributeType</code> to be owned by the instances of this type.	<code>AttributeType</code>
<code>annotations</code>	The <code>AttributeType</code> that this attribute ownership overrides, if applicable.	<code>Annotation[]</code>

Returns

`Promise<void>`

Code examples

```
thingType.setOwns(transaction, attributeType)
thingType.setOwns(transaction, attributeType, overriddenType,
[Annotation.KEY])
```

setOwns

```
setOwns(transaction, attributeType, overriddenType): Promise<void>
```

Allows the instances of this `ThingType` to own the given `AttributeType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeType</code>	The <code>AttributeType</code> to be owned by the instances of this type.	<code>AttributeType</code>
<code>overriddenType</code>	The <code>AttributeType</code> that this attribute ownership overrides, if applicable.	<code>AttributeType</code>

Returns

`Promise<void>`

Code examples

```
thingType.setOwns(transaction, attributeType)
thingType.setOwns(transaction, attributeType, overriddenType,
[Annotation.KEY])
```

setOwns

```
setOwns(transaction, attributeType, overriddenType, annotations):
Promise<void>
```

Allows the instances of this `ThingType` to own the given `AttributeType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeType</code>	The <code>AttributeType</code> to be owned by the instances of this type.	<code>AttributeType</code>

Name	Description	Type
<code>overriddenType</code>	The <code>AttributeType</code> that this attribute ownership overrides, if applicable.	<code>AttributeType</code>
<code>annotations</code>	Adds annotations to the ownership.	<code>Annotation[]</code>

Returns

`Promise<void>`

Code examples

```
thingType.setOwns(transaction, attributeType)
thingType.setOwns(transaction, attributeType, overriddenType,
[Annotation.KEY])
```

setPlays

```
setPlays(transaction, role): Promise<void>
```

Allows the instances of this `ThingType` to play the given role.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>role</code>	The role to be played by the instances of this type	<code>RoleType</code>

Returns

`Promise<void>`

Code examples

```
thingType.setPlays(transaction, role) thingType.setPlays(transaction, role, overriddenType)
```

setPlays

```
setPlays(transaction, role, overriddenType): Promise<void>
```

Allows the instances of this `ThingType` to play the given role.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>role</code>	The role to be played by the instances of this type	<code>RoleType</code>
<code>overriddenType</code>	The role type that this role overrides, if applicable	<code>RoleType</code>

Returns

```
Promise<void>
```

Code examples

```
thingType.setPlays(transaction, role) thingType.setPlays(transaction, role, overriddenType)
```

setRegex

```
setRegex(transaction, regex): Promise<void>
```

Sets a regular expression as a constraint for this `AttributeType`. `Values` of all `Attribute`s of this type (inserted earlier or later) should match this regex. Can only be applied for `AttributeType`s with a `string` value type.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>regex</code>	Regular expression	<code>string</code>

Returns

`Promise<void>`

Code examples

```
attributeType.setRegex(transaction, regex)
```

setSupertype

```
setSupertype(transaction, type): Promise<void>
```

Input parameters

Name	Description	Type
<code>transaction</code>		<code>TypeDBTransaction</code>
<code>type</code>		<code>AttributeType</code>

Returns

`Promise<void>`

toJSONRecord

```
toJSONRecord(): Record<string, string | number | boolean>
```

Retrieves the concept as JSON.

Returns

`Record<string, string | number | boolean>`

Code examples

```
concept.toJSONRecord()
```

unsetAbstract

```
unsetAbstract(transaction): Promise<void>
```

Set a `ThingType` to be non-abstract, meaning it can have instances.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<void>`

Code examples

```
thingType.unsetAbstract(transaction)
```

unsetOwns

```
unsetOwns(transaction, attributeType): Promise<void>
```

Disallows the instances of this `ThingType` from owning the given `AttributeType`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeType</code>	The <code>AttributeType</code> to not be owned by the type.	<code>AttributeType</code>

Returns

`Promise<void>`

Code examples

```
thingType.unsetOwns(transaction, attributeType)
```

unsetPlays

```
unsetPlays(transaction, role): Promise<void>
```

Disallows the instances of this `ThingType` from playing the given role.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>role</code>	The role to not be played by the instances of this type.	<code>RoleType</code>

Returns

`Promise<void>`

Code examples

```
thingType.unsetPlays(transaction, role)
```

unsetRegex

```
unsetRegex(transaction): Promise<void>
```

Removes the regular expression that is defined for this `AttributeType`.

Input parameters

Name	Description	Type
------	-------------	------

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<void>`

Code examples

```
attributeType.unsetRegex(transaction)
```

Annotation

Annotations for ownership declarations.

Fields

Name	Type	Description
<code>KEY</code>	<code>Annotation</code>	Annotation to specify the attribute owned is a KEY
<code>UNIQUE</code>	<code>Annotation</code>	Annotation to specify the owned is UNIQUE

parse

```
parse(string): Annotation
```

Returns the relevant `Annotation` given the name as a string

Input parameters

Name	Description	Type
<code>string</code>	name of the attribute as a string. e.g.: "key", "unique"	<code>string</code>

Returns

`Annotation`

toString

```
toString(): string
```

Printable string

Returns

string

Transitivity

Namespace variables

Name
EXPLICIT
TRANSITIVE

new Transitivity

```
new Transitivity(transitivity): Transitivity
```

Input parameters

Name	Description	Type
transitivity		TypeTransitivity

Returns

Transitivity

ValueType

TypeQL value types for attributes & value concepts.

Namespace variables

Name
BOOLEAN
DATETIME
DOUBLE
LONG
OBJECT
STRING

name

```
name(): string
```

Returns

string

new ValueType

```
new ValueType(type, name): ValueType
```

Input parameters

Name	Description	Type
type		ValueType
name		string

Returns

ValueType

toString

```
toString(): string
```

Returns

`string`

Data

Thing

Supertypes:

- `Concept`

Fields

Name	Type	Description
<code>iid</code>	<code>string</code>	Retrieves the unique id of the Thing.
<code>inferred</code>	<code>boolean</code>	Checks if this Thing is inferred by a [Reasoning Rule].
<code>type</code>	<code>ThingType</code>	Retrieves the type which this Thing belongs to.

asAttribute

```
asAttribute(): Attribute
```

Casts the concept to `Attribute`.

Returns

`Attribute`

Code examples

```
concept.asAttribute()
```

asAttributeType

```
asAttributeType(): AttributeType
```

Casts the concept to `AttributeType`.

Returns

`AttributeType`

Code examples

```
concept.asAttributeType()
```

asEntity

```
asEntity(): Entity
```

Casts the concept to `Entity`.

Returns

`Entity`

Code examples

```
concept.asEntity()
```

asEntityType

```
asEntityType(): EntityType
```

Casts the concept to `EntityType`.

Returns

`EntityType`

Code examples

```
concept.asEntityType()
```

asRelation

```
asRelation(): Relation
```

Casts the concept to `Relation`.

Returns

`Relation`

Code examples

```
concept.asRelation()
```

asRelationType

```
asRelationType(): RelationType
```

Casts the concept to `RelationType`.

Returns

`RelationType`

Code examples

```
concept.asRelationType()
```

asRoleType

```
asRoleType(): RoleType
```

Casts the concept to `RoleType`.

Returns

`RoleType`

Code examples


```
concept.asRoleType()
```

asThing

```
asThing(): Thing
```

Casts the concept to `Thing`.

Returns

`Thing`

Code examples

```
concept.asThing()
```

asThingType

```
asThingType(): ThingType
```

Casts the concept to `ThingType`.

Returns

`ThingType`

Code examples

```
concept.asThingType()
```

asType

```
asType(): Type
```

Casts the concept to `Type`.

Returns

`Type`

Code examples

```
concept.asType()
```

asValue

```
asValue(): Value
```

Casts the concept to `Value`.

Returns

`Value`

Code examples

```
concept.asValue()
```

delete

```
delete(transaction): Promise<void>
```

Deletes this `Thing`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<void>`

Code examples

```
thing.delete(transaction)
```

equals

```
equals(concept): boolean
```

Checks if this concept is equal to the argument `concept`.

Input parameters

Name	Description	Type
<code>concept</code>	The concept to compare to.	<code>Concept</code>

Returns

`boolean`

getHas

```
getHas(transaction): Stream<Attribute>
```

Retrieves the `Attribute`s that this `Thing` owns. Optionally, filtered by an `AttributeType` or a list of `AttributeType`s. Optionally, filtered by `Annotation`s.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<Attribute>`

Code examples

```
thing.getHas(transaction) thing.getHas(transaction, attributeType,  
[Annotation.KEY])
```

getHas

```
getHas(transaction, annotations): Stream<Attribute>
```

Retrieves the `Attribute` s that this `Thing` owns. Optionally, filtered by an `AttributeType` or a list of `AttributeType` s. Optionally, filtered by `Annotation` s.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>annotations</code>	The <code>AttributeType</code> s to filter the at-tributes by	<code>Annotation[]</code>

Returns

`Stream<Attribute>`

Code examples

```
thing.getHas(transaction) thing.getHas(transaction, attributeType,
[Annotation.KEY])
```

getHas

```
getHas(transaction, attributeType): Stream<Attribute>
```

Retrieves the `Attribute` s that this `Thing` owns. Optionally, filtered by an `AttributeType` or a list of `AttributeType` s. Optionally, filtered by `Annotation` s.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeType</code>	The <code>AttributeType</code> s to filter the at-tributes by	<code>AttributeType</code>

Returns

`Stream<Attribute>`

Code examples

```
thing.getHas(transaction) thing.getHas(transaction, attributeType,  
[Annotation.KEY])
```

getHas

```
getHas(transaction, attributeTypes): Stream<Attribute>
```

Retrieves the `Attribute`s that this `Thing` owns. Optionally, filtered by an `AttributeType` or a list of `AttributeType`s. Optionally, filtered by `Annotation`s.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeTypes</code>	The <code>AttributeType</code> s to filter the attributes by	<code>AttributeType[]</code>

Returns

```
Stream<Attribute>
```

Code examples

```
thing.getHas(transaction) thing.getHas(transaction, attributeType,  
[Annotation.KEY])
```

getHas

```
getHas(transaction, attributeTypes, annotations): Stream<Attribute>
```

Retrieves the `Attribute`s that this `Thing` owns. Optionally, filtered by an `AttributeType` or a list of `AttributeType`s. Optionally, filtered by `Annotation`s.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeTypes</code>	The <code>AttributeType</code> s to filter the attributes by	<code>AttributeType[]</code>
<code>annotations</code>	Only retrieve attributes with all given <code>Annotation</code> s	<code>Annotation[]</code>

Returns

`Stream<Attribute>`

Code examples

```
thing.getHas(transaction) thing.getHas(transaction, attributeType,
[Annotation.KEY])
```

getPlaying

```
getPlaying(transaction): Stream<RoleType>
```

Retrieves the roles that this `Thing` is currently playing.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<RoleType>`

Code examples

```
thing.getPlaying(transaction)
```

getRelations

```
getRelations(transaction): Stream<Relation>
```

Retrieves all the **Relations** which this **Thing** plays a role in, optionally filtered by one or more given roles.

Input parameters

Name	Description	Type
transaction	The current transaction	TypeDBTransaction

Returns

```
Stream<Relation>
```

Code examples

```
thing.getRelations(transaction, roleTypes)
```

getRelations

```
getRelations(transaction, roleTypes): Stream<Relation>
```

Retrieves all the **Relations** which this **Thing** plays a role in, optionally filtered by one or more given roles.

Input parameters

Name	Description	Type
transaction	The current transaction	TypeDBTransaction
roleTypes	The list of roles to filter the relations by.	RoleType[]

Returns

```
Stream<Relation>
```

Code examples

```
thing.getRelations(transaction, roleTypes)
```

isAttribute

```
isAttribute(): boolean
```

Checks if the concept is an `Attribute`.

Returns

```
boolean
```

Code examples

```
concept.isAttribute()
```

isAttributeType

```
isAttributeType(): boolean
```

Checks if the concept is an `AttributeType`.

Returns

```
boolean
```

Code examples

```
concept.isAttributeType()
```

isDeleted

```
isDeleted(transaction): Promise<boolean>
```

Checks if this `Thing` is deleted.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<boolean>`

Code examples

```
thing.isDeleted(transaction)
```

isEntity

```
isEntity(): boolean
```

Checks if the concept is an `Entity`.

Returns

`boolean`

Code examples

```
concept.isEntity()
```

isEntityType

```
isEntityType(): boolean
```

Checks if the concept is an `EntityType`.

Returns

`boolean`

Code examples

```
concept.isEntityType()
```

isRelation

```
isRelation(): boolean
```

Checks if the concept is a `Relation`.

Returns

```
boolean
```

Code examples

```
concept.isRelation()
```

isRelationType

```
isRelationType(): boolean
```

Checks if the concept is a `RelationType`.

Returns

```
boolean
```

Code examples

```
concept.isRelationType()
```

isRoleType

```
isRoleType(): boolean
```

Checks if the concept is a `RoleType`.

Returns

```
boolean
```

Code examples

```
concept.isRoleType()
```

isThing

```
isThing(): boolean
```

Checks if the concept is a `Thing`.

Returns

```
boolean
```

Code examples

```
concept.isThing()
```

isThingType

```
isThingType(): boolean
```

Checks if the concept is a `ThingType`.

Returns

```
boolean
```

Code examples

```
concept.isThingType()
```

isType

```
isType(): boolean
```

Checks if the concept is a `Type`.

Returns

```
boolean
```

Code examples

```
concept.isType()
```

isValue

```
isValue(): boolean
```

Checks if the concept is a `Value`.

Returns

`boolean`

Code examples

```
concept.isValue()
```

setHas

```
setHas(transaction, attribute): Promise<void>
```

Assigns an `Attribute` to be owned by this `Thing`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attribute</code>	The <code>Attribute</code> to be owned by this <code>Thing</code> .	<code>Attribute</code>

Returns

`Promise<void>`

Code examples

```
thing.setHas(transaction, attribute)
```

toJSONRecord

```
toJSONRecord(): Record<string, string | number | boolean>
```

Retrieves the concept as JSON.

Returns

```
Record<string, string | number | boolean>
```

Code examples

```
concept.toJSONRecord()
```

unsetHas

```
unsetHas(transaction, attribute): Promise<void>
```

Unassigns an `Attribute` from this `Thing`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attribute</code>	The <code>Attribute</code> to be disowned from this <code>Thing</code> .	<code>Attribute</code>

Returns

```
Promise<void>
```

Code examples

```
thing.unsetHas(transaction, attribute)
```

Entity

Supertypes:

- `Thing`

Instance of data of an entity type, representing a standalone object that exists in the data model independently. Entity does not have a value. It is usually addressed by its ownership over attribute instances and/or roles played in relation instances.

Fields

Name	Type	Description
<code>iid</code>	<code>string</code>	Retrieves the unique id of the Thing.
<code>inferred</code>	<code>boolean</code>	Checks if this Thing is inferred by a [Reasoning Rule].
<code>type</code>	<code>EntityType</code>	The type which this Entity belongs to.

asAttribute

```
asAttribute(): Attribute
```

Casts the concept to `Attribute`.

Returns

`Attribute`

Code examples

```
concept.asAttribute()
```

asAttributeType

```
asAttributeType(): AttributeType
```

Casts the concept to `AttributeType`.

Returns

`AttributeType`

Code examples

```
concept.asAttributeType()
```

asEntity

```
asEntity(): Entity
```

Casts the concept to `Entity`.

Returns

`Entity`

Code examples

```
concept.asEntity()
```

asEntityType

```
asEntityType(): EntityType
```

Casts the concept to `EntityType`.

Returns

`EntityType`

Code examples

```
concept.asEntityType()
```

asRelation

```
asRelation(): Relation
```

Casts the concept to `Relation`.

Returns

`Relation`

Code examples

```
concept.asRelation()
```

asRelationType

```
asRelationType(): RelationType
```

Casts the concept to `RelationType`.

Returns

`RelationType`

Code examples

```
concept.asRelationType()
```

asRoleType

```
asRoleType(): RoleType
```

Casts the concept to `RoleType`.

Returns

`RoleType`

Code examples

```
concept.asRoleType()
```

asThing

```
asThing(): Thing
```

Casts the concept to `Thing`.

Returns

`Thing`

Code examples


```
concept.asThing()
```

asThingType

```
asThingType(): ThingType
```

Casts the concept to `ThingType`.

Returns

`ThingType`

Code examples

```
concept.asThingType()
```

asType

```
asType(): Type
```

Casts the concept to `Type`.

Returns

`Type`

Code examples

```
concept.asType()
```

asValue

```
asValue(): Value
```

Casts the concept to `Value`.

Returns

`Value`

Code examples

```
concept.asValue()
```

delete

```
delete(transaction): Promise<void>
```

Deletes this `Thing`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<void>`

Code examples

```
thing.delete(transaction)
```

equals

```
equals(concept): boolean
```

Checks if this concept is equal to the argument `concept`.

Input parameters

Name	Description	Type
<code>concept</code>	The concept to compare to.	<code>Concept</code>

Returns

`boolean`

getHas

```
getHas(transaction): Stream<Attribute>
```

Retrieves the `Attribute`s that this `Thing` owns. Optionally, filtered by an `AttributeType` or a list of `AttributeType`s. Optionally, filtered by `Annotation`s.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

```
Stream<Attribute>
```

Code examples

```
thing.getHas(transaction) thing.getHas(transaction, attributeType,  
[Annotation.KEY])
```

getHas

```
getHas(transaction, annotations): Stream<Attribute>
```

Retrieves the `Attribute`s that this `Thing` owns. Optionally, filtered by an `AttributeType` or a list of `AttributeType`s. Optionally, filtered by `Annotation`s.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>annotations</code>	The <code>AttributeType</code> s to filter the attributes by	<code>Annotation[]</code>

Returns

```
Stream<Attribute>
```

Code examples

```
thing.getHas(transaction) thing.getHas(transaction, attributeType,  
[Annotation.KEY])
```

getHas

```
getHas(transaction, attributeType): Stream<Attribute>
```

Retrieves the `Attribute`s that this `Thing` owns. Optionally, filtered by an `AttributeType` or a list of `AttributeType`s. Optionally, filtered by `Annotation`s.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeType</code>	The <code>AttributeType</code> s to filter the attributes by	<code>AttributeType</code>

Returns

```
Stream<Attribute>
```

Code examples

```
thing.getHas(transaction) thing.getHas(transaction, attributeType,  
[Annotation.KEY])
```

getHas

```
getHas(transaction, attributeTypes): Stream<Attribute>
```

Retrieves the `Attribute`s that this `Thing` owns. Optionally, filtered by an `AttributeType` or a list of `AttributeType`s. Optionally, filtered by `Annotation`s.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeTypes</code>	The <code>AttributeType</code> s to filter the attributes by	<code>AttributeType[]</code>

Returns

`Stream<Attribute>`

Code examples

```
thing.getHas(transaction) thing.getHas(transaction, attributeType,
[Annotation.KEY])
```

getHas

```
getHas(transaction, attributeTypes, annotations): Stream<Attribute>
```

Retrieves the `Attribute`s that this `Thing` owns. Optionally, filtered by an `AttributeType` or a list of `AttributeType`s. Optionally, filtered by `Annotation`s.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeTypes</code>	The <code>AttributeType</code> s to filter the attributes by	<code>AttributeType[]</code>
<code>annotations</code>	Only retrieve attributes with all given <code>Annotation</code> s	<code>Annotation[]</code>

Returns

`Stream<Attribute>`

Code examples

```
thing.getHas(transaction) thing.getHas(transaction, attributeType,
[Annotation.KEY])
```

getPlaying

```
getPlaying(transaction): Stream<RoleType>
```

Retrieves the roles that this **Thing** is currently playing.

Input parameters

Name	Description	Type
transaction	The current transaction	TypeDBTransaction

Returns

Stream<RoleType>

Code examples

```
thing.getPlaying(transaction)
```

getRelations

```
getRelations(transaction): Stream<Relation>
```

Retrieves all the **Relations** which this **Thing** plays a role in, optionally filtered by one or more given roles.

Input parameters

Name	Description	Type
transaction	The current transaction	TypeDBTransaction

Returns

`Stream<Relation>`

Code examples

```
thing.getRelations(transaction, roleTypes)
```

getRelations

```
getRelations(transaction, roleTypes): Stream<Relation>
```

Retrieves all the `Relations` which this `Thing` plays a role in, optionally filtered by one or more given roles.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>roleTypes</code>	The list of roles to filter the relations by.	<code>RoleType[]</code>

Returns

`Stream<Relation>`

Code examples

```
thing.getRelations(transaction, roleTypes)
```

isAttribute

```
isAttribute(): boolean
```

Checks if the concept is an `Attribute`.

Returns

`boolean`

Code examples

```
concept.isAttribute()
```

isAttributeType

```
isAttributeType(): boolean
```

Checks if the concept is an `AttributeType`.

Returns

`boolean`

Code examples

```
concept.isAttributeType()
```

isDeleted

```
isDeleted(transaction): Promise<boolean>
```

Checks if this `Thing` is deleted.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<boolean>`

Code examples

```
thing.isDeleted(transaction)
```

isEntity


```
isEntity(): boolean
```

Checks if the concept is an `Entity`.

Returns

```
boolean
```

Code examples

```
concept.isEntity()
```

isEntityType

```
isEntityType(): boolean
```

Checks if the concept is an `EntityType`.

Returns

```
boolean
```

Code examples

```
concept.isEntityType()
```

isRelation

```
isRelation(): boolean
```

Checks if the concept is a `Relation`.

Returns

```
boolean
```

Code examples

```
concept.isRelation()
```

isRelationType

```
isRelationType(): boolean
```

Checks if the concept is a `RelationType`.

Returns

```
boolean
```

Code examples

```
concept.isRelationType()
```

isRoleType

```
isRoleType(): boolean
```

Checks if the concept is a `RoleType`.

Returns

```
boolean
```

Code examples

```
concept.isRoleType()
```

isThing

```
isThing(): boolean
```

Checks if the concept is a `Thing`.

Returns

```
boolean
```

Code examples

```
concept.isThing()
```

isThingType

```
isThingType(): boolean
```

Checks if the concept is a `ThingType`.

Returns

```
boolean
```

Code examples

```
concept.isThingType()
```

isType

```
isType(): boolean
```

Checks if the concept is a `Type`.

Returns

```
boolean
```

Code examples

```
concept.isType()
```

isValue

```
isValue(): boolean
```

Checks if the concept is a `Value`.

Returns

```
boolean
```

Code examples

```
concept.isValue()
```

setHas

```
setHas(transaction, attribute): Promise<void>
```

Assigns an `Attribute` to be owned by this `Thing`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attribute</code>	The <code>Attribute</code> to be owned by this <code>Thing</code> .	<code>Attribute</code>

Returns

```
Promise<void>
```

Code examples

```
thing.setHas(transaction, attribute)
```

toJSONRecord

```
toJSONRecord(): Record<string, string | number | boolean>
```

Retrieves the concept as JSON.

Returns

```
Record<string, string | number | boolean>
```

Code examples

```
concept.toJSONRecord()
```

unsetHas

```
unsetHas(transaction, attribute): Promise<void>
```

Unassigns an `Attribute` from this `Thing`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attribute</code>	The <code>Attribute</code> to be disowned from this <code>Thing</code> .	<code>Attribute</code>

Returns

`Promise<void>`

Code examples

```
thing.unsetHas(transaction, attribute)
```

Relation

Supertypes:

- `Thing`

Relation is an instance of a relation type and can be uniquely addressed by a combination of its type, owned attributes and role players.

Fields

Name	Type	Description
<code>iid</code>	<code>string</code>	Retrieves the unique id of the Thing.
<code>inferred</code>	<code>boolean</code>	Checks if this Thing is inferred by a [Reasoning Rule].
<code>type</code>	<code>RelationType</code>	The type which this Relation belongs to.

addRolePlayer

```
addRolePlayer(transaction, roleType, player): Promise<void>
```

Adds a new role player to play the given role in this `Relation`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>roleType</code>	The role to be played by the <code>player</code>	<code>RoleType</code>
<code>player</code>	The thing to play the role	<code>Thing</code>

Returns

```
Promise<void>
```

Code examples

```
relation.addRolePlayer(transaction, roleType, player)
```

asAttribute

```
asAttribute(): Attribute
```

Casts the concept to `Attribute`.

Returns

```
Attribute
```

Code examples

```
concept.asAttribute()
```

asAttributeType

```
asAttributeType(): AttributeType
```

Casts the concept to `AttributeType`.

Returns

`AttributeType`

Code examples

```
concept.asAttributeType()
```

asEntity

```
asEntity(): Entity
```

Casts the concept to `Entity`.

Returns

`Entity`

Code examples

```
concept.asEntity()
```

asEntityType

```
asEntityType(): EntityType
```

Casts the concept to `EntityType`.

Returns

`EntityType`

Code examples

```
concept.asEntityType()
```

asRelation

```
asRelation(): Relation
```

Casts the concept to `Relation`.

Returns

`Relation`

Code examples

```
concept.asRelation()
```

asRelationType

```
asRelationType(): RelationType
```

Casts the concept to `RelationType`.

Returns

`RelationType`

Code examples

```
concept.asRelationType()
```

asRoleType

```
asRoleType(): RoleType
```

Casts the concept to `RoleType`.

Returns

`RoleType`

Code examples

```
concept.asRoleType()
```

asThing

```
asThing(): Thing
```


Casts the concept to `Thing`.

Returns

`Thing`

Code examples

```
concept.asThing()
```

asThingType

```
asThingType(): ThingType
```

Casts the concept to `ThingType`.

Returns

`ThingType`

Code examples

```
concept.asThingType()
```

asType

```
asType(): Type
```

Casts the concept to `Type`.

Returns

`Type`

Code examples

```
concept.asType()
```

asValue

```
asValue(): Value
```

Casts the concept to `Value`.

Returns

`Value`

Code examples

```
concept.asValue()
```

delete

```
delete(transaction): Promise<void>
```

Deletes this `Thing`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<void>`

Code examples

```
thing.delete(transaction)
```

equals

```
equals(concept): boolean
```

Checks if this concept is equal to the argument `concept`.

Input parameters

Name	Description	Type
------	-------------	------

Name	Description	Type
<code>concept</code>	The concept to compare to.	<code>Concept</code>

Returns

`boolean`

getHas

```
getHas(transaction): Stream<Attribute>
```

Retrieves the `Attribute`s that this `Thing` owns. Optionally, filtered by an `AttributeType` or a list of `AttributeType`s. Optionally, filtered by `Annotation`s.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<Attribute>`

Code examples

```
thing.getHas(transaction) thing.getHas(transaction, attributeType,
[Annotation.KEY])
```

getHas

```
getHas(transaction, annotations): Stream<Attribute>
```

Retrieves the `Attribute`s that this `Thing` owns. Optionally, filtered by an `AttributeType` or a list of `AttributeType`s. Optionally, filtered by `Annotation`s.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>annotations</code>	The <code>AttributeType</code> s to filter the attributes by	<code>Annotation[]</code>

Returns

`Stream<Attribute>`

Code examples

```
thing.getHas(transaction) thing.getHas(transaction, attributeType,  
[Annotation.KEY])
```

getHas

```
getHas(transaction, attributeType): Stream<Attribute>
```

Retrieves the `Attribute`s that this `Thing` owns. Optionally, filtered by an `AttributeType` or a list of `AttributeType`s. Optionally, filtered by `Annotation`s.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeType</code>	The <code>AttributeType</code> s to filter the attributes by	<code>AttributeType</code>

Returns

`Stream<Attribute>`

Code examples

```
thing.getHas(transaction) thing.getHas(transaction, attributeType,
[Annotation.KEY])
```

getHas

```
getHas(transaction, attributeTypes): Stream<Attribute>
```

Retrieves the `Attribute`s that this `Thing` owns. Optionally, filtered by an `AttributeType` or a list of `AttributeType`s. Optionally, filtered by `Annotation`s.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeTypes</code>	The <code>AttributeType</code> s to filter the attributes by	<code>AttributeType[]</code>

Returns

```
Stream<Attribute>
```

Code examples

```
thing.getHas(transaction) thing.getHas(transaction, attributeType,
[Annotation.KEY])
```

getHas

```
getHas(transaction, attributeTypes, annotations): Stream<Attribute>
```

Retrieves the `Attribute`s that this `Thing` owns. Optionally, filtered by an `AttributeType` or a list of `AttributeType`s. Optionally, filtered by `Annotation`s.

Input parameters

Name	Description	Type
------	-------------	------

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeTypes</code>	The <code>AttributeType</code> s to filter the attributes by	<code>AttributeType[]</code>
<code>annotations</code>	Only retrieve attributes with all given <code>Annotation</code> s	<code>Annotation[]</code>

Returns

`Stream<Attribute>`

Code examples

```
thing.getHas(transaction) thing.getHas(transaction, attributeType,
[Annotation.KEY])
```

getPlayersByRoleType

```
getPlayersByRoleType(transaction): Stream<Thing>
```

Retrieves all role players of this `Relation`, optionally filtered by given role types.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<Thing>`

Code examples

```
relation.getPlayersByRoleType(transaction)
relation.getPlayersByRoleType(transaction, [roleType1, roleType2])
```

getPlayersByRoleType

```
getPlayersByRoleType(transaction, roleTypes): Stream<Thing>
```

Retrieves all role players of this `Relation`, optionally filtered by given role types.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>roleTypes</code>	0 or more role types	<code>RoleType[]</code>

Returns

`Stream<Thing>`

Code examples

```
relation.getPlayersByRoleType(transaction)
relation.getPlayersByRoleType(transaction, [roleType1, roleType2])
```

getPlaying

```
getPlaying(transaction): Stream<RoleType>
```

Retrieves the roles that this `Thing` is currently playing.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<RoleType>`

Code examples

```
thing.getPlaying(transaction)
```

getRelating

```
getRelating(transaction): Stream<RoleType>
```

Retrieves all role types currently played in this **Relation**.

Input parameters

Name	Description	Type
transaction	The current transaction	TypeDBTransaction

Returns

```
Stream<RoleType>
```

Code examples

```
relation.getRelating(transaction)
```

getRelations

```
getRelations(transaction): Stream<Relation>
```

Retrieves all the **Relations** which this **Thing** plays a role in, optionally filtered by one or more given roles.

Input parameters

Name	Description	Type
transaction	The current transaction	TypeDBTransaction

Returns

```
Stream<Relation>
```


Code examples

```
thing.getRelations(transaction, roleTypes)
```

getRelations

```
getRelations(transaction, roleTypes): Stream<Relation>
```

Retrieves all the **Relations** which this **Thing** plays a role in, optionally filtered by one or more given roles.

Input parameters

Name	Description	Type
transaction	The current transaction	TypeDBTransaction
roleTypes	The list of roles to filter the relations by.	RoleType[]

Returns

```
Stream<Relation>
```

Code examples

```
thing.getRelations(transaction, roleTypes)
```

getRolePlayers

```
getRolePlayers(transaction): Promise<Map<RoleType, Thing[]>>
```

Retrieves a mapping of all instances involved in the **Relation** and the role each play.

Input parameters

Name	Description	Type
transaction	The current transaction	TypeDBTransaction

Returns

```
Promise<Map<RoleType, Thing[]>>
```

Code examples

```
relation.getRolePlayers(transaction)
```

isAttribute

```
isAttribute(): boolean
```

Checks if the concept is an `Attribute`.

Returns

```
boolean
```

Code examples

```
concept.isAttribute()
```

isAttributeType

```
isAttributeType(): boolean
```

Checks if the concept is an `AttributeType`.

Returns

```
boolean
```

Code examples

```
concept.isAttributeType()
```

isDeleted

```
isDeleted(transaction): Promise<boolean>
```

Checks if this `Thing` is deleted.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<boolean>`

Code examples

```
thing.isDeleted(transaction)
```

isEntity

```
isEntity(): boolean
```

Checks if the concept is an `Entity`.

Returns

`boolean`

Code examples

```
concept.isEntity()
```

isEntityType

```
isEntityType(): boolean
```

Checks if the concept is an `EntityType`.

Returns

`boolean`

Code examples

```
concept.isEntityType()
```

isRelation

```
isRelation(): boolean
```

Checks if the concept is a `Relation`.

Returns

```
boolean
```

Code examples

```
concept.isRelation()
```

isRelationType

```
isRelationType(): boolean
```

Checks if the concept is a `RelationType`.

Returns

```
boolean
```

Code examples

```
concept.isRelationType()
```

isRoleType

```
isRoleType(): boolean
```

Checks if the concept is a `RoleType`.

Returns

```
boolean
```

Code examples

```
concept.isRoleType()
```

isThing

```
isThing(): boolean
```

Checks if the concept is a `Thing`.

Returns

```
boolean
```

Code examples

```
concept.isThing()
```

isThingType

```
isThingType(): boolean
```

Checks if the concept is a `ThingType`.

Returns

```
boolean
```

Code examples

```
concept.isThingType()
```

isType

```
isType(): boolean
```

Checks if the concept is a `Type`.

Returns

```
boolean
```

Code examples

```
concept.isType()
```

isValue

```
isValue(): boolean
```

Checks if the concept is a `Value`.

Returns

`boolean`

Code examples

```
concept.isValue()
```

removeRolePlayer

```
removeRolePlayer(transaction, roleType, player): Promise<void>
```

Removes the association of the given instance that plays the given role in this `Relation`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>roleType</code>	The role to no longer be played by the thing in this <code>Relation</code>	<code>RoleType</code>
<code>player</code>	The instance to no longer play the role in this <code>Relation</code>	<code>Thing</code>

Returns

`Promise<void>`

Code examples

```
relation.removeRolePlayer(transaction, roleType, player)
```

setHas

```
setHas(transaction, attribute): Promise<void>
```

Assigns an `Attribute` to be owned by this `Thing`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attribute</code>	The <code>Attribute</code> to be owned by this <code>Thing</code> .	<code>Attribute</code>

Returns

```
Promise<void>
```

Code examples

```
thing.setHas(transaction, attribute)
```

toJSONRecord

```
toJSONRecord(): Record<string, string | number | boolean>
```

Retrieves the concept as JSON.

Returns

```
Record<string, string | number | boolean>
```

Code examples

```
concept.toJSONRecord()
```

unsetHas

```
unsetHas(transaction, attribute): Promise<void>
```

Unassigns an `Attribute` from this `Thing`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attribute</code>	The <code>Attribute</code> to be disowned from this <code>Thing</code> .	<code>Attribute</code>

Returns

```
Promise<void>
```

Code examples

```
thing.unsetHas(transaction, attribute)
```

Attribute

Supertypes:

- `Thing`

Attribute is an instance of the attribute type and has a value. This value is fixed and unique for every given instance of the attribute type. Attributes can be uniquely addressed by their type and value.

Fields

Name	Type	Description
<code>iid</code>	<code>string</code>	Retrieves the unique id of the Thing.
<code>inferred</code>	<code>boolean</code>	Checks if this Thing is inferred by a [Reasoning Rule].

Name	Type	Description
<code>type</code>	<code>AttributeType</code>	The type which this Attribute belongs to.
<code>value</code>	<code>string</code>	The value which the Attribute instance holds.
<code>valueType</code>	<code>ValueType</code>	The type of the value which the Attribute instance holds.

asAttribute

```
asAttribute(): Attribute
```

Casts the concept to `Attribute`.

Returns

`Attribute`

Code examples

```
concept.asAttribute()
```

asAttributeType

```
asAttributeType(): AttributeType
```

Casts the concept to `AttributeType`.

Returns

`AttributeType`

Code examples

```
concept.asAttributeType()
```

asEntity

```
asEntity(): Entity
```

Casts the concept to `Entity`.

Returns

`Entity`

Code examples

```
concept.asEntity()
```

asEntityType

```
asEntityType(): EntityType
```

Casts the concept to `EntityType`.

Returns

`EntityType`

Code examples

```
concept.asEntityType()
```

asRelation

```
asRelation(): Relation
```

Casts the concept to `Relation`.

Returns

`Relation`

Code examples

```
concept.asRelation()
```

asRelationType

```
asRelationType(): RelationType
```

Casts the concept to `RelationType`.

Returns

```
RelationType
```

Code examples

```
concept.asRelationType()
```

asRoleType

```
asRoleType(): RoleType
```

Casts the concept to `RoleType`.

Returns

```
RoleType
```

Code examples

```
concept.asRoleType()
```

asThing

```
asThing(): Thing
```

Casts the concept to `Thing`.

Returns

```
Thing
```

Code examples

```
concept.asThing()
```

asThingType

```
asThingType(): ThingType
```

Casts the concept to `ThingType`.

Returns

```
ThingType
```

Code examples

```
concept.asThingType()
```

asType

```
asType(): Type
```

Casts the concept to `Type`.

Returns

```
Type
```

Code examples

```
concept.asType()
```

asValue

```
asValue(): Value
```

Casts the concept to `Value`.

Returns

```
Value
```

Code examples

```
concept.asValue()
```

delete

```
delete(transaction): Promise<void>
```

Deletes this `Thing`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Promise<void>`

Code examples

```
thing.delete(transaction)
```

equals

```
equals(concept): boolean
```

Checks if this concept is equal to the argument `concept`.

Input parameters

Name	Description	Type
<code>concept</code>	The concept to compare to.	<code>Concept</code>

Returns

`boolean`

getHas

```
getHas(transaction): Stream<Attribute>
```

Retrieves the `Attribute`s that this `Thing` owns. Optionally, filtered by an `AttributeType`

or a list of `AttributeType` s. Optionally, filtered by `Annotation` s.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<Attribute>`

Code examples

```
thing.getHas(transaction) thing.getHas(transaction, attributeType,
[Annotation.KEY])
```

getHas

```
getHas(transaction, annotations): Stream<Attribute>
```

Retrieves the `Attribute` s that this `Thing` owns. Optionally, filtered by an `AttributeType` or a list of `AttributeType` s. Optionally, filtered by `Annotation` s.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>annotations</code>	The <code>AttributeType</code> s to filter the at-tributes by	<code>Annotation[]</code>

Returns

`Stream<Attribute>`

Code examples

```
thing.getHas(transaction) thing.getHas(transaction, attributeType,
[Annotation.KEY])
```

getHas

```
getHas(transaction, attributeType): Stream<Attribute>
```

Retrieves the `Attribute`s that this `Thing` owns. Optionally, filtered by an `AttributeType` or a list of `AttributeType`s. Optionally, filtered by `Annotation`s.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeType</code>	The <code>AttributeType</code> s to filter the attributes by	<code>AttributeType</code>

Returns

```
Stream<Attribute>
```

Code examples

```
thing.getHas(transaction) thing.getHas(transaction, attributeType, [Annotation.KEY])
```

getHas

```
getHas(transaction, attributeTypes): Stream<Attribute>
```

Retrieves the `Attribute`s that this `Thing` owns. Optionally, filtered by an `AttributeType` or a list of `AttributeType`s. Optionally, filtered by `Annotation`s.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Name	Description	Type
<code>attributeTypes</code>	The <code>AttributeType</code> s to filter the attributes by	<code>AttributeType[]</code>

Returns

`Stream<Attribute>`

Code examples

```
thing.getHas(transaction) thing.getHas(transaction, attributeType,
[Annotation.KEY])
```

getHas

```
getHas(transaction, attributeTypes, annotations): Stream<Attribute>
```

Retrieves the `Attribute`s that this `Thing` owns. Optionally, filtered by an `AttributeType` or a list of `AttributeType`s. Optionally, filtered by `Annotation`s.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attributeTypes</code>	The <code>AttributeType</code> s to filter the attributes by	<code>AttributeType[]</code>
<code>annotations</code>	Only retrieve attributes with all given <code>Annotation</code> s	<code>Annotation[]</code>

Returns

`Stream<Attribute>`

Code examples


```
thing.getHas(transaction) thing.getHas(transaction, attributeType,
[Annotation.KEY])
```

getOwners

```
getOwners(transaction, ownerType?): Stream<Thing>
```

Retrieves the instances that own this `Attribute`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>ownerType</code>	If specified, filter results for only owners of the given type	<code>ThingType</code>

Returns

```
Stream<Thing>
```

Code examples

```
attribute.getOwners(transaction) attribute.getOwners(transaction,
ownerType)
```

getPlaying

```
getPlaying(transaction): Stream<RoleType>
```

Retrieves the roles that this `Thing` is currently playing.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<RoleType>`

Code examples

```
thing.getPlaying(transaction)
```

getRelations

```
getRelations(transaction): Stream<Relation>
```

Retrieves all the `Relations` which this `Thing` plays a role in, optionally filtered by one or more given roles.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

`Stream<Relation>`

Code examples

```
thing.getRelations(transaction, roleTypes)
```

getRelations

```
getRelations(transaction, roleTypes): Stream<Relation>
```

Retrieves all the `Relations` which this `Thing` plays a role in, optionally filtered by one or more given roles.

Input parameters

Name	Description	Type
------	-------------	------

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>roleTypes</code>	The list of roles to filter the relations by.	<code>RoleType[]</code>

Returns

`Stream<Relation>`

Code examples

```
thing.getRelations(transaction, roleTypes)
```

isAttribute

```
isAttribute(): boolean
```

Checks if the concept is an `Attribute`.

Returns

`boolean`

Code examples

```
concept.isAttribute()
```

isAttributeType

```
isAttributeType(): boolean
```

Checks if the concept is an `AttributeType`.

Returns

`boolean`

Code examples

```
concept.isAttributeType()
```

isDeleted

```
isDeleted(transaction): Promise<boolean>
```

Checks if this `Thing` is deleted.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>

Returns

```
Promise<boolean>
```

Code examples

```
thing.isDeleted(transaction)
```

isEntity

```
isEntity(): boolean
```

Checks if the concept is an `Entity`.

Returns

```
boolean
```

Code examples

```
concept.isEntity()
```

isEntityType

```
isEntityType(): boolean
```

Checks if the concept is an `EntityType`.

Returns

`boolean`

Code examples

```
concept.isEntityType()
```

isRelation

```
isRelation(): boolean
```

Checks if the concept is a `Relation`.

Returns

`boolean`

Code examples

```
concept.isRelation()
```

isRelationType

```
isRelationType(): boolean
```

Checks if the concept is a `RelationType`.

Returns

`boolean`

Code examples

```
concept.isRelationType()
```

isRoleType

```
isRoleType(): boolean
```

Checks if the concept is a `RoleType`.

Returns

`boolean`

Code examples

```
concept.isRoleType()
```

isThing

```
isThing(): boolean
```

Checks if the concept is a `Thing`.

Returns

`boolean`

Code examples

```
concept.isThing()
```

isThingType

```
isThingType(): boolean
```

Checks if the concept is a `ThingType`.

Returns

`boolean`

Code examples

```
concept.isThingType()
```

isType

```
isType(): boolean
```

Checks if the concept is a `Type`.

Returns

`boolean`

Code examples

```
concept.isType()
```

isValue

```
isValue(): boolean
```

Checks if the concept is a `Value`.

Returns

`boolean`

Code examples

```
concept.isValue()
```

setHas

```
setHas(transaction, attribute): Promise<void>
```

Assigns an `Attribute` to be owned by this `Thing`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attribute</code>	The <code>Attribute</code> to be owned by this <code>Thing</code> .	<code>Attribute</code>

Returns

`Promise<void>`

Code examples

```
thing.setHas(transaction, attribute)
```

toJSONRecord

```
toJSONRecord(): Record<string, string | number | boolean>
```

Retrieves the concept as JSON.

Returns

```
Record<string, string | number | boolean>
```

Code examples

```
concept.toJSONRecord()
```

unsetHas

```
unsetHas(transaction, attribute): Promise<void>
```

Unassigns an `Attribute` from this `Thing`.

Input parameters

Name	Description	Type
<code>transaction</code>	The current transaction	<code>TypeDBTransaction</code>
<code>attribute</code>	The <code>Attribute</code> to be disowned from this <code>Thing</code> .	<code>Attribute</code>

Returns

```
Promise<void>
```

Code examples

```
thing.unsetHas(transaction, attribute)
```


Value

Supertypes:

- `Concept`

Fields

Name	Type	Description
<code>value</code>	<code>string</code>	Retrieves the value which this value concept holds.
<code>valueType</code>	<code>ValueType</code>	The ValueType of this value concept

asAttribute

```
asAttribute(): Attribute
```

Casts the concept to `Attribute`.

Returns

`Attribute`

Code examples

```
concept.asAttribute()
```

asAttributeType

```
asAttributeType(): AttributeType
```

Casts the concept to `AttributeType`.

Returns

`AttributeType`

Code examples

```
concept.asAttributeType()
```

asBoolean

```
asBoolean(): boolean
```

Returns a `boolean` value of this value concept. If the value has another type, raises an exception.

Returns

`boolean`

Code examples

```
value.asBoolean()
```

asDateTime

```
asDateTime(): Date
```

Returns a `datetime` value of this value concept. If the value has another type, raises an exception.

Returns

`Date`

Code examples

```
value.asDatetime()
```

asDouble

```
asDouble(): number
```

Returns a `number` value of this value concept. If the value has another type, raises an exception.

Returns

`number`

Code examples

```
value.asDouble()
```

asEntity

```
asEntity(): Entity
```

Casts the concept to `Entity`.

Returns

`Entity`

Code examples

```
concept.asEntity()
```

asEntityType

```
asEntityType(): EntityType
```

Casts the concept to `EntityType`.

Returns

`EntityType`

Code examples

```
concept.asEntityType()
```

asLong

```
asLong(): number
```

Returns a `number` value of this value concept. If the value has another type, raises an exception.

Returns

`number`

Code examples

```
value.asLong()
```

asRelation

```
asRelation(): Relation
```

Casts the concept to `Relation`.

Returns

`Relation`

Code examples

```
concept.asRelation()
```

asRelationType

```
asRelationType(): RelationType
```

Casts the concept to `RelationType`.

Returns

`RelationType`

Code examples

```
concept.asRelationType()
```

asRoleType

```
asRoleType(): RoleType
```

Casts the concept to `RoleType`.

Returns

`RoleType`

Code examples

```
concept.asRoleType()
```

asString

```
asString(): string
```

Returns a `string` value of this value concept. If the value has another type, raises an exception.

Returns

`string`

Code examples

```
value.asString()
```

asThing

```
asThing(): Thing
```

Casts the concept to `Thing`.

Returns

`Thing`

Code examples

```
concept.asThing()
```

asThingType

```
asThingType(): ThingType
```

Casts the concept to `ThingType`.

Returns

ThingType

Code examples

```
concept.asThingType()
```

asType

```
asType(): Type
```

Casts the concept to `Type`.

Returns

`Type`

Code examples

```
concept.asType()
```

asValue

```
asValue(): Value
```

Casts the concept to `Value`.

Returns

`Value`

Code examples

```
concept.asValue()
```

equals

```
equals(concept): boolean
```

Checks if this concept is equal to the argument `concept`.

Input parameters

Name	Description	Type
<code>concept</code>	The concept to compare to.	<code>Concept</code>

Returns

`boolean`

isAttribute

```
isAttribute(): boolean
```

Checks if the concept is an `Attribute`.

Returns

`boolean`

Code examples

```
concept.isAttribute()
```

isAttributeType

```
isAttributeType(): boolean
```

Checks if the concept is an `AttributeType`.

Returns

`boolean`

Code examples

```
concept.isAttributeType()
```

isBoolean

```
isBoolean(): boolean
```

Returns **True** if the value which this value concept holds is of type **boolean**. Otherwise, returns **False**.

Returns

boolean

Code examples

```
value.isBoolean()
```

isDateTime

```
isDateTime(): boolean
```

Returns **True** if the value which this value concept holds is of type **datetime**. Otherwise, returns **False**.

Returns

boolean

Code examples

```
value.isDatetime()
```

isDouble

```
isDouble(): boolean
```

Returns **True** if the value which this value concept holds is of type **double**. Otherwise, returns **False**.

Returns

boolean

Code examples

```
value.isDouble()
```

isEntity


```
isEntity(): boolean
```

Checks if the concept is an `Entity`.

Returns

```
boolean
```

Code examples

```
concept.isEntity()
```

isEntityType

```
isEntityType(): boolean
```

Checks if the concept is an `EntityType`.

Returns

```
boolean
```

Code examples

```
concept.isEntityType()
```

isLong

```
isLong(): boolean
```

Returns `True` if the value which this value concept holds is of type `long`. Otherwise, returns `False`.

Returns

```
boolean
```

Code examples

```
value.isLong()
```

isRelation

```
isRelation(): boolean
```

Checks if the concept is a `Relation`.

Returns

```
boolean
```

Code examples

```
concept.isRelation()
```

isRelationType

```
isRelationType(): boolean
```

Checks if the concept is a `RelationType`.

Returns

```
boolean
```

Code examples

```
concept.isRelationType()
```

isRoleType

```
isRoleType(): boolean
```

Checks if the concept is a `RoleType`.

Returns

```
boolean
```

Code examples

```
concept.isRoleType()
```

isString

```
isString(): boolean
```

Returns **True** if the value which this value concept holds is of type **string**. Otherwise, returns **False**.

Returns

boolean

Code examples

```
value.isString()
```

isThing

```
isThing(): boolean
```

Checks if the concept is a **Thing**.

Returns

boolean

Code examples

```
concept.isThing()
```

isThingType

```
isThingType(): boolean
```

Checks if the concept is a **ThingType**.

Returns

boolean

Code examples

```
concept.isThingType()
```

isType

```
isType(): boolean
```

Checks if the concept is a `Type`.

Returns

```
boolean
```

Code examples

```
concept.isType()
```

isValue

```
isValue(): boolean
```

Checks if the concept is a `Value`.

Subscribe to Newsletter

Returns

```
boolean
```



Code examples

Cor

```
concept.isValue()
```

 [Discuss on Forum](#)
[toJSONRecord](#)



```
toJSONRecord(): Record<string, string | number | boolean>
```

Retrieves the concept as JSON.

Technology

Documentation

Resources

Company

Returns

```
Record<string, string | number | boolean>
```

[Philosophy](#)

[Overview](#)

[Webinars](#)

[LinkedIn](#)

Code examples

Logic

© 2023 Vaticle Ltd

Vaticle™, TypeDB™ and TypeQL™ are trademarks of Vaticle Ltd

<> with ❤️ by **Vaticle**

LogicManager

Provides methods for manipulating rules in the database.

getRule

```
getRule(label): Promise<Rule>
```

Retrieves the Rule that has the given label.

Input parameters

Name	Description	Type
<code>label</code>	The label of the Rule to create or retrieve	<code>string</code>

Returns

`Promise<Rule>`

Code examples

```
transaction.logic.getRule(label)
```

getRules

```
getRules(): Stream<Rule>
```

Retrieves all rules.

Returns

`Stream<Rule>`

Code examples

```
transaction.logic.getRules()
```

putRule

```
putRule(label, when, then): Promise<Rule>
```

Creates a new Rule if none exists with the given label, or replaces the existing one.

Input parameters

Name	Description	Type
<code>label</code>	The label of the Rule to create or replace	<code>string</code>
<code>when</code>	The when body of the rule to create	<code>string</code>
<code>then</code>	The then body of the rule to create	<code>string</code>

Returns

```
Promise<Rule>
```

Code examples

```
transaction.logic.putRule(label, when, then)
```

Rule

Rules are a part of schema and define embedded logic. The reasoning engine uses rules as a set of logic to infer new data. A rule consists of a condition and a conclusion, and is uniquely identified by a label.

Fields

Name	Type	Description
<code>label</code>	<code>string</code>	The unique label of the rule.

Name	Type	Description
<code>then</code>	<code>string</code>	The single statement that constitutes the 'then' of the rule.
<code>when</code>	<code>string</code>	The statements that constitute the 'when' of the rule.

delete

```
delete(transaction): Promise<void>
```

Deletes this rule.

Input parameters

Name	Description	Type
<code>transaction</code>	The current <code>Transaction</code>	<code>TypeDBTransaction</code>

Returns

```
Promise<void>
```

Code examples

```
rule.delete(transaction)
```

isDeleted

```
isDeleted(transaction): Promise<boolean>
```

Check if this rule has been deleted.

Input parameters

Name	Description	Type
------	-------------	------

Name	Description	Type
<code>transaction</code>	The current <code>Transaction</code>	<code>TypeDBTransaction</code>

Returns

`Promise<boolean>`

Code examples

```
rule.isDeleted(transaction)
```

setLabel

```
setLabel(transaction, label): Promise<void>
```

Renames the label of the rule. The new label must remain unique.

Input parameters

Name	Description	Type
<code>transaction</code>	The current <code>Transaction</code>	<code>TypeDBTransaction</code>
<code>label</code>		<code>string</code>

Returns

`Promise<void>`

Code examples

```
rule.setLabel(transaction, newLabel)
```

Errors

TypeDBDriverError

Supertypes:

- Error

Errors encountered when interacting with TypeDB

Fields

Name	Type	Description
message	string	
name	string	
prepareStackTrace	any	
stack	string	
stackTraceLimit	number	

messageTemplate

```
get messageTemplate(): ErrorMessage
```

Returns the message template for this error.

Returns

ErrorMessage

captureStackTrace

```
captureStackTrace(targetObject, constructorOpt?): void
```

Create .stack property on a target object

Input parameters

Name	Description	Type
------	-------------	------

Name	Description	Type
<code>targetObject</code>		<code>object</code>
<code>constructorOpt</code>		<code>Function</code>

Returns

`void`

new TypeDBDriverError

```
new TypeDBDriverError(error): TypeDBDriverError
```

Input parameters

Name	Description	Type
<code>error</code>		<code>string ErrorMessage Error ServiceError</code>

Returns

`TypeDBDriverError`

ErrorMessage

Class defining the error-code and message template for `TypeDBDriverError`s

code

```
code(): string
```

Retrieves the error-code for this ErrorMessage

Returns

`string`

message

```
message(...args): string
```

Generates the error message by substituting `args` into the `messageTemplate`

Input parameters

Name	Description	Type
<code>args</code>	The format arguments to the message-template.	<code>Stringable[]</code>

Returns

`string`

new ErrorMessage

```
new ErrorMessage(codePrefix, codeNumber, messagePrefix, messageBody):  
ErrorMessage
```

Input parameters

Name	Description	Type
<code>codePrefix</code>		<code>string</code>
<code>codeNumber</code>		<code>number</code>
<code>messagePrefix</code>		<code>string</code>
<code>messageBody</code>		<code>((args) => string)</code>

Returns

`ErrorMessage`

toString

```
toString(): string
```

