

anatomy of a read query

A query is a JSON object: a comma-separated list of name:value pairs, enclosed in curly braces. These name:value pairs can be **properties**, **wildcards**, **comparisons**, or **directives**, and can be combined and nested to create complex queries.

```
{
  "query": {
    "name": "The Police",
    "type": "/music/artist",
    "track": [
      {
        "name": null,
        "name-=": "Message",
        "limit": 1
      }
    ]
  }
}
```

the MQL query object (must be called "query")
a property with a known value
constrain matches to properties of this type
a nested query with a "[{}]" placeholder returns a list of tracks
a wildcard
a property being requested with the "null" placeholder
a comparison
a directive

Returns all properties (the **wildcard**) related to no more than 1 (the **limit directive**) track (the **"track" property**, with values enclosed by []) by the musical artist (the **"type" property**) "The Police" (the **"name" property**) that start with the word "Message" (the **"name" comparison**).

Properties are name:value pairs that begin with a quoted **property name** followed by a colon and a quoted **property value** (for values you know, used to constrain your query) or a **placeholder** (for values you want Freebase to complete).

Identifiers are ways of referring to objects in Freebase:		
Name	A human-readable reference to an object. Names are not unique.	"The Police"
guid	Globally Unique Identifier. Each object has one guid. Can't change, but where it points can.	9202a8c04000641f800000000006df1b
id	A fully qualified hierarchical name of /type/id/. Can be deleted or re-assigned.	/en/the_police or /guid/9202a8c04000641f800000000006df1b
Namespace & Key	Keys define the fully qualified name by the value of the key and its namespace.	namespace = /en key value = "the_police"

Read Directives are reserved words used to refine your query:		
<pre>{ "type": "/music/artist", "name": null, "name-=": "The", "album": [{ "name": null, "name-=": "greatest hits", "optional": true }], "sort": "name", "limit": 10 }</pre>	<i>Returns the first 10 (the limit directive) musical artists whose names start with "The", sorted alphabetically by name (the sort directive). If they have an album with the phrase "greatest hits" in it, return that too, but don't fail to return if they don't (the optional directive).</i>	
"limit": 10	return a maximum of 10 results	
"sort": "name"	sort results by the property "name"	
"sort": "-name"	reverse sort results by the property "name"	
"sort": ["name", "date"]	sort on the property "name" then "date"	
"sort": "index"	request a sorting index and sort the results by it	
"optional": true	when added to a nested query, allows the parent query to return results when the nested query doesn't	
"optional": "forbidden"	in a nested query, prevents the parent query from matching if the nested query matches	
"return": "count"	returns the number of matching results	
"count": null	includes the total count of matching elements within each result	
"return": "estimate-count"	returns the estimated number of matching results, much faster but less accurate than "return": "count"	
"estimate-count": null	includes an estimated count of matching elements within each result, much faster but less accurate than "count": null	

Placeholders specify what values you want your query to return:		
returns that contains		
null	a single value	- the value of "name" or "id" for object types - the value of "value" for value types
[]	a list of values	like null, for properties with multiple values (avoids uniqueness errors!)
{ }	a single object	- "name", "id" and "type" objects for object types - "value" and "type" objects for value types - /type/text includes "lang" object - /type/key includes "namespace" object
[{ }]	a list of objects	like { }, for properties with multiple objects

Wildcards are placeholders that return all of the properties of the parent object:

"*": <placeholder> returns the expanded placeholder query for each property of the parent object's expected type (unless the parent property is already mentioned in the query alongside "**")

Numeric Comparisons are made by adding mathematical operators to a repeated property name:		
>	greater than	"age": null, "age>=": 18, "age<": 21
>=	greater than or equal to	
<	less than	"name": null, "name<": "bob", "name>": "bill", "type": "/music/track"
<=	less than or equal to	

Textual Comparisons are made by adding the pattern matching operator (==) to a repeated property name:		
matches strings containing	examples	
love	the word "love"	"love" or "I love you", not "glove" or "lover"
love*	words beginning with "love"	"love" or "lover", but not "glove"
*love	words ending with "love"	"love" or "glove", but not "lover"
love	words that contain "love"	"love", "glove" and "lover"
love you	the words "love" and "you"	"I love you" or "you I love", not "love your"
\love you\	the phrase "love you"	"I love you", not "you I love" or "love your"
^	matches string beginning	^love matches strings beginning with "love"
\$	matches string ending	love\$ matches strings ending with "love"
*	alone matches any single word	\I * you\ matches "I love you" or "I hate you"
-	matches one or zero characters	free-base matches "freebase", "free base"
\	is the escape character	\- \\$ matches "-\$"

advanced read syntax

Property Prefixes are optional arbitrary identifiers, followed by colons, that can prefix any property name.

These can be used to add multiple AND-like constraints to the same property:

```
{ "type": "/music/artist",
  "name": null,
  "a:album": "Greatest Hits",
  "b:album": "Super Hits" }
```

Returns artists who have an album named "Greatest Hits" and an album named "Super Hits."

They can also be used to constrain a property and query it at the same time:

```
{ "type": "/music/artist",
  "name": null,
  "album": [ ],
  "inc:album": "Super Hits" }
```

Returns artists who have an album named "Super Hits" and returns a list of their albums.

Reciprocal Properties All objects in Freebase are linked, and the reciprocal property of a given property can be found by using "!".

```
{ "type": "/location/country",
  "name": "Monaco",
  "!/people/person/nationality": [ ] }
```

Returns persons who are citizens of Monaco by tracing the link from people/person/nationality back to location/country

"One of" and "but not" Operators can be used to match any one of the values in an array, or to exclude a value from read results.

|= is used to match any of the values in an array:

```
{ "type": "/music/artist",
  "name": null,
  "album|=": ["Greatest Hits", "Super Hits"],
  "album": [ ] }
```

Returns one or more albums named either "Greatest Hits" or "Super Hits."

!= is used to indicate that the results should not contain the given value:

```
{ "type": "/music/artist",
  "name": "The Police",
  "album": [
    {
      "name": null,
      "name!=": "Greatest Hits"
    }
  ] }
```

Returns all albums by The Police except "Greatest Hits"

The "but not" operator functions like the "optional": "forbidden" directive, but is best used to exclude a unique property expressed as as single JSON literal.

Numbers can be used in the same way as strings:

```
{ "type": "/chemistry/chemical_element",
  "name": null,
  "atomic_number|=": [1,2,3],
  "atomic_number": null,
  "sort": "atomic_number" }
```

Returns the first three chemical elements.

```
{ "type": "/chemistry/chemical_element",
  "name": null,
  "atomic_number!=": 1,
  "a:atomic_number!=": 2,
  "b:atomic_number!=": 3 }
```

Returns all but the first three chemical elements.

the mqlread service

responds to HTTP GET requests here: <http://api.freebase.com/api/service/mqlread>

Input is passed to mqlread inside a JSON-serialized URI-encoded *query envelope* object containing a MQL query object:

```
{
  "query": {
    "id": "/en/the_police",
    "name": null
  }
}
```

the query envelope object
the MQL query object (must be called "query")
a property with a known value
a property being requested with the "null" placeholder

The JSON object is sent to mqlread as the value of a URL parameter named `query`:
[http://api.freebase.com/api/service/mqlread?query={\"query\":{\"id\":\"/en/the_police\",\"name\":null}}](http://api.freebase.com/api/service/mqlread?query={\)

Multiple queries can be passed to mqlread by placing MQL query objects in multiple, uniquely named *query envelope* objects enclosed by an outer envelope object:

```
{
  "q1": {
    "query": {
      "id": "/en/the_police",
      "name": null
    }
  },
  "q2": {
    "query": {
      "id": "/en/led_zeppelin",
      "name": null
    }
  }
}
```

the outer envelope object
the first query envelope object ("q1" is arbitrary)
the first MQL query object (must be called "query")
the second query envelope object ("q2" is arbitrary)
the second MQL query object (must again be called "query")

Output responses are symmetrical to their input queries, adding a status code, transaction id, and the property values requested by your query to a JSON-serialized envelope object contained in the body of a `text/plain` HTTP response. Here's the output of the first query above:

```
{
  "status": "200 OK",
  "code": "/api/status/ok",
  "result": {
    "id": "/en/the_police",
    "name": "The Police"
  },
  "transaction_id": "cache:cache01.p01.sjc1:8101:2008-09-19T21:33:22Z:0010"
}
```

the outer envelope object
http status code
query status code
a MQL result object
the property value you knew
the property value you requested, now filled in
the transaction id

the mqlwrite service

responds to HTTP POST requests here: <http://api.freebase.com/api/service/mqlwrite>

Input and Output work much like mqlread. A single write query is wrapped in an envelope object and submitted in the body of an HTTP POST request as the value of the `query` parameter.

A `Cookie` header is required for authentication credentials, and the `X-Metaweb-Request` header is required as a measure against XSS attacks. Like mqlread, mqlwrite output is a JSON-serialized envelope object contained in the body of an HTTP response.

Write experiments should be conducted on the Freebase Sandbox at <http://sandbox.freebase.com>

JavaScript Callback (JSONP) allows you to dynamically inject mqlread output into the calling JavaScript as a function. Add a URL parameter named `callback` to the mqlread request. The value given to `callback` will be used as the name of the JavaScript function that the output will be wrapped in.

Query Envelope Parameters can be used to retrieve a large set of results in batches, and to make queries against past contents of the Freebase database. When making multiple queries, these are placed inside each uniquely named query envelope.

<code>"cursor":true</code>	Results will include a cursor property in the inner response envelope. If the value of <code>CURSOR</code> is false, then all results have been returned. If not, then the value will be a string of opaque data. Insert this value in the inner envelope of your next query to retrieve the next batch of results.
<code>"as_of_time":<timestamp></code>	Use with an ISO8601 timestamp to query the database contents as they were on a specific date and/or time.
<code>"uniqueness_failure":"soft"</code>	Prevents an error when more than one result is returned and the query is not enclosed in square brackets indicating that multiple results are expected.
<code>"escape":false</code>	Disables escaping of <code><</code> , <code>></code> , and <code>&</code> characters in mqlread responses. Should not be used to display output in Web browsers since it opens vulnerability to script injection attacks.
<code>"lang":"lang/language id"</code>	For objects of <code>/type/text</code> , will provide the results in the language specified. Language id parameter should be an id in the <code>/lang</code> namespace.

Write Envelope Parameters are added to envelope objects in the same way *query envelope parameters* are:

<code>"use_permission_of":<id></code>	sets the permissions of the created object to be the same as that of the object identified by the id.
---	---

write syntax

Add write directives to create new objects and connect existing ones.

Write Directives

```
{
  "create": "unless_exists",
  "name": "New Object",
  "id": null
}
```

Creates a new object unless an object named "New Object" already exists.
"id":null returns the id of the new or existing object.

The create directive creates new objects:

<code>"create": "unless_exists"</code>	look for a matching object and create it if it doesn't exist
<code>"create": "unless_connected"</code>	look for a matching object connected to the parent query, and create and connect it if it doesn't exist
<code>"create": "unconditional"</code>	create the specified object without looking for a match (dangerous; use carefully!)

Possible responses to create directives:

<code>"create": "created"</code>	a new object has been created
<code>"create": "existed"</code>	no object was created, because a matching object exists
<code>"create": "connected"</code>	object was not created, but a matching existing object was connected

The connect directive connects existing objects:

<code>"connect": "insert"</code>	attach a value or object to a non-unique property, or attach the first value or object to a unique property
<code>"connect": "update"</code>	attach a value or object to a unique property replacing any value or object that was previously connected
<code>"connect": "replace"</code>	updates unique properties and performs an insert for non-unique properties
<code>"connect": "delete"</code>	detach a value or object from any property

Possible responses to connect directives:

<code>"connect": "inserted"</code>	the insert directive was successful.
<code>"connect": "updated"</code>	the update directive was successful.
<code>"connect": "deleted"</code>	the delete directive was successful.
<code>"connect": "present"</code>	an insert or update directive was unnecessary because the connection was already present.
<code>"connect": "absent"</code>	a delete directive was unnecessary because an object being connected did not exist.

questions?

The Freebase Developer's Email List is a great place to ask questions and get input from other developers. The Freebase Blog also features postings from Metaweb's internal developers, updates on new releases, monthly data dumps, and data mob projects.

Freebase Developer's Email List
<http://lists.freebase.com/mailman/listinfo/developers>

Freebase Blog
<http://blog.freebase.com>

Freebase IRC Channel
<irc://irc.freenode.net#freebase>