ZABBIX

ZQL – THE QUERY LANGUAGE

Petite et dabitur vobis quaerite et invenietis ...

Vladimir Ulogov

PREFACE

What are the ZQ and ZQL ?

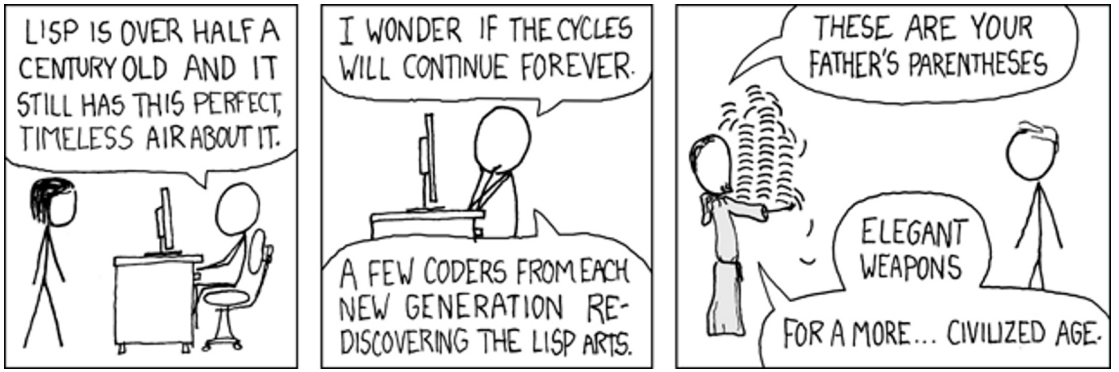
When the ZQ stands for “Zabbix Query”, “ZQL” is the “Zabbix Query Language”, which is the interactove and batch query language and Unix command oine tool, for translating ZQL to a subset of the Zabbix API calls.

Is that a server, database library, or what ?

It it nether. ZQ is the Python module, which is compiled into a machine code for your platform using Cython. ZQL is a interpreted language, implemented inside this module with few external dependencies. The distribution also provides you a command-line tool ZQL, which act as standalone interpreter and a query executor.

PREFACE

So, what kind of language is ZQL ?



ZQL is a built around stack operations, LISP-based extendable interpreted language implemented inside Python interpreter.

PREFACE

Is that subset of SQL ?

ZQL have no relation with Structured Query Language and never will be, thanks to it's implementation.

PREFACE

Show me some example of what's ZQL is looks like ….

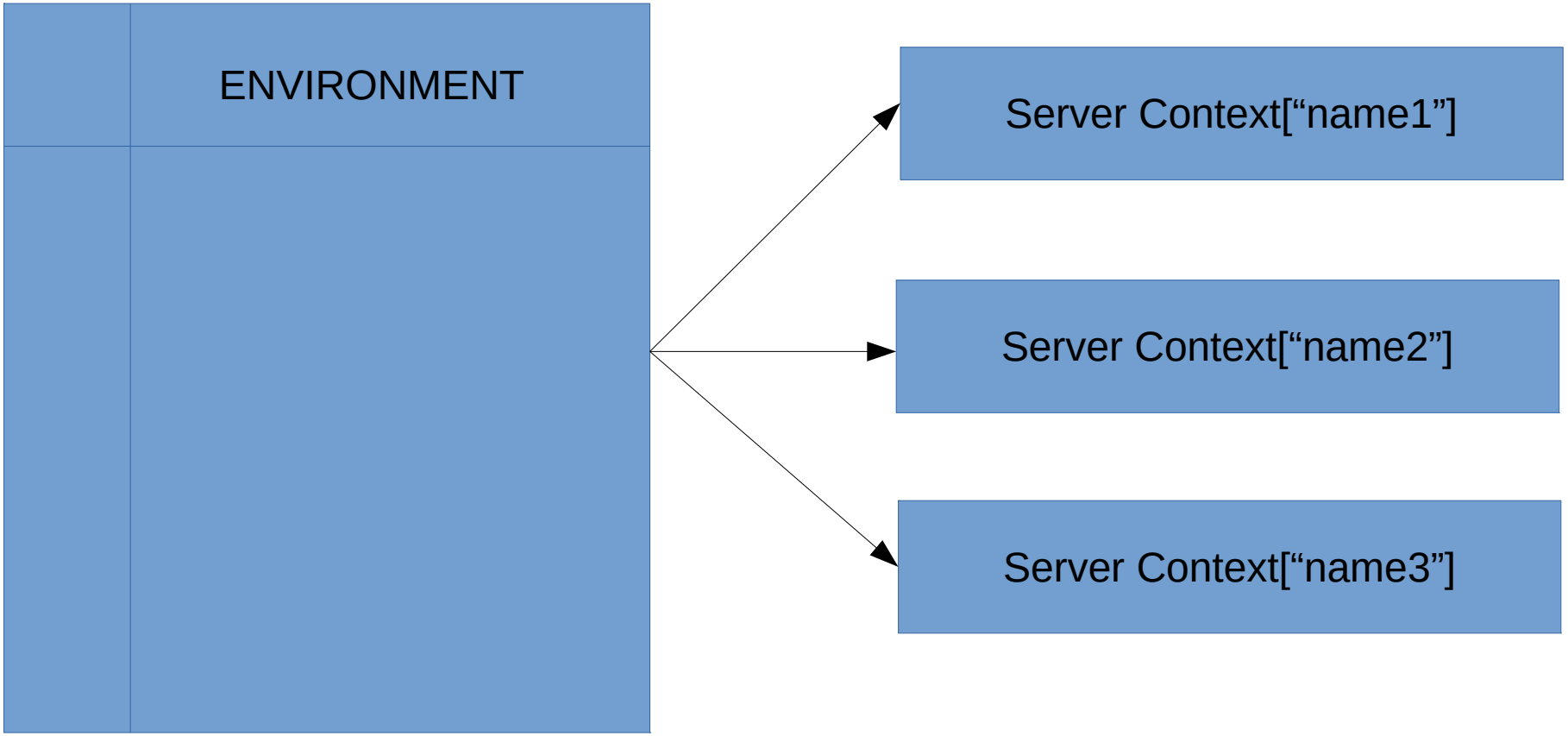
As you wish, here is an example of the correct ZQL, which connects to the server, grabs the list of the hosts, filter the hosts which do have a proxy configuration, merge proxy information and display output as JSON

(ZBX NEW "zabbix") (Hosts) (Filter TRUE [proxy_hostid Ne 0]) (Proxies) (Merge proxy_hostid proxyid) (Out) (Pretty_Json)

- Are you serious !? How can anybody understand this mumbo-jumbo !
- Well, in fact, this is very simple, but powerful language. Quite easy to understand, once you've get the grasp on few concepts…..

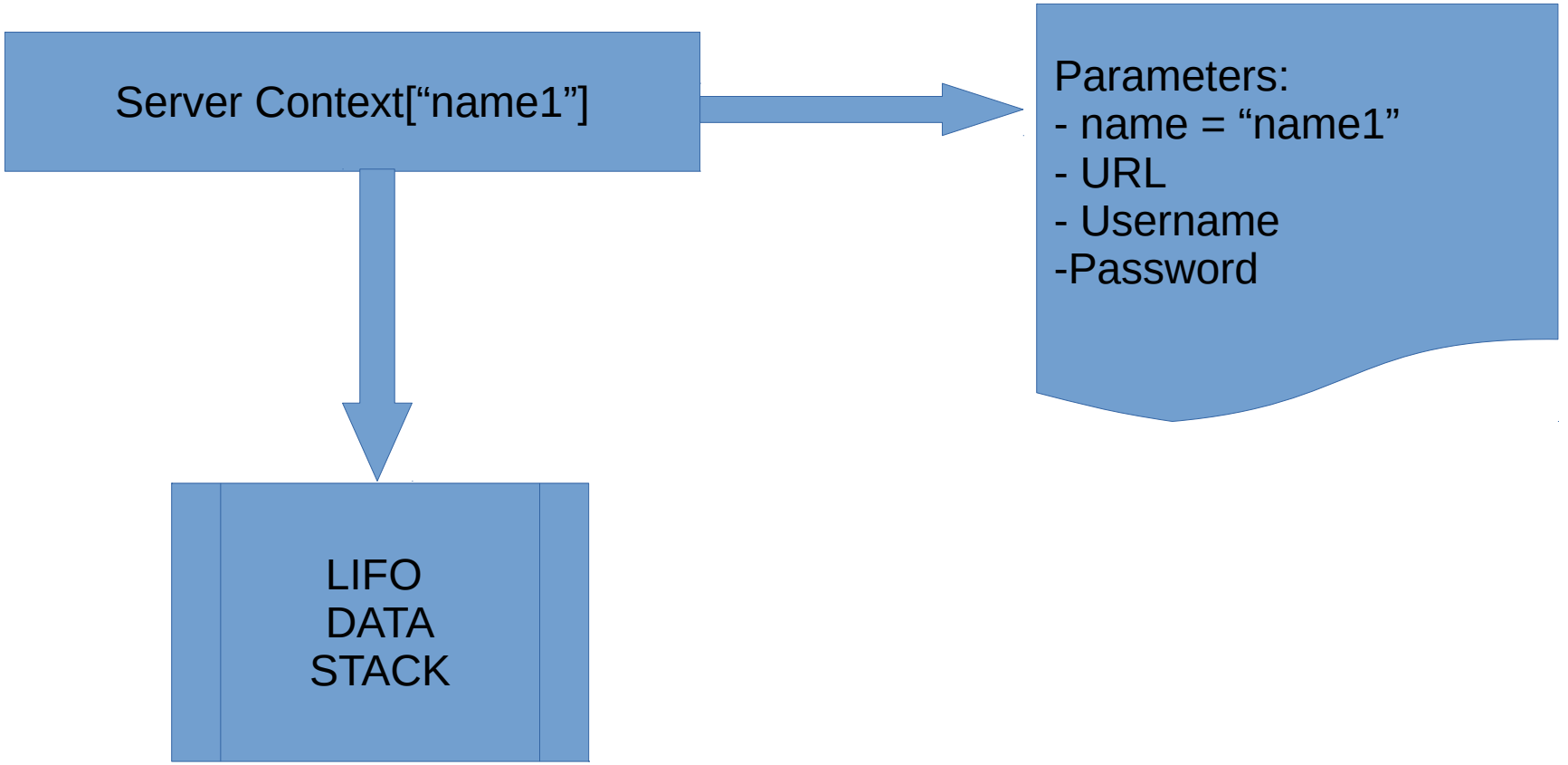
Servers and Context

First, you have to understand the Notation of the Environment and Context.



Servers and Context

First, you have to understand the Notation of the Environment and Context.

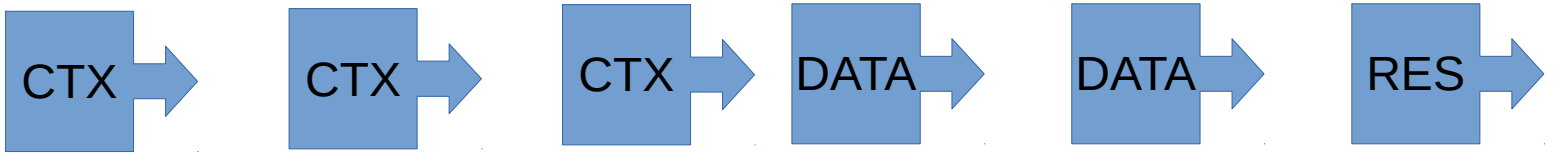


Query structure

ZQL query consists of the “words”.

- Each word is a single S-Expression: (x y z)
- The “x” is the first element of an expression and the “name” of the “word”.
- The “words” in the query are organized in a sequential pipeline.
- Return value of the previous S-Expression passed to the next “word” down to the pipeline in indirectly defined first parameter.
- Return value of the last “word” will be the return value of the ZQL query.
- There are three types of the “words”:
  - Context manipulating “words”;
  - Stack manipulating “words”;
  - Data manipulating “words”.

(Context) (Stack …) (Stack …) (Data) (Data …) (Data …)



Query structure

Wait ! But this is not a LISP S-Expressions !

(LISP)

And you are right. The S-Expression is a notation of nested tree structure (list) data, for the Lisp programming language, used for source code and source data.

That's is why I call it a "Query Language", or "Zabbix Query Language" to link the purpose of this language with the specific application. The ZQL query is consisting of the "words", each of this words is expression and it is passing data from word to word through the pipeline. This shall be very familiar for everyone, who is using Unix. ZQL is essentially is:

(ZQL)

```
cat /data/file | grep "pattern" | wc -l
```

Query structure

… so here is the last note about relationship, between nested s-expressions in LISP and ZQL pipelines:

(ZQL)

You have a query like this one in ZQL pipeline format:

(word1) (word2) (word3)

(LISP)

The same query in s-expression will look like this:

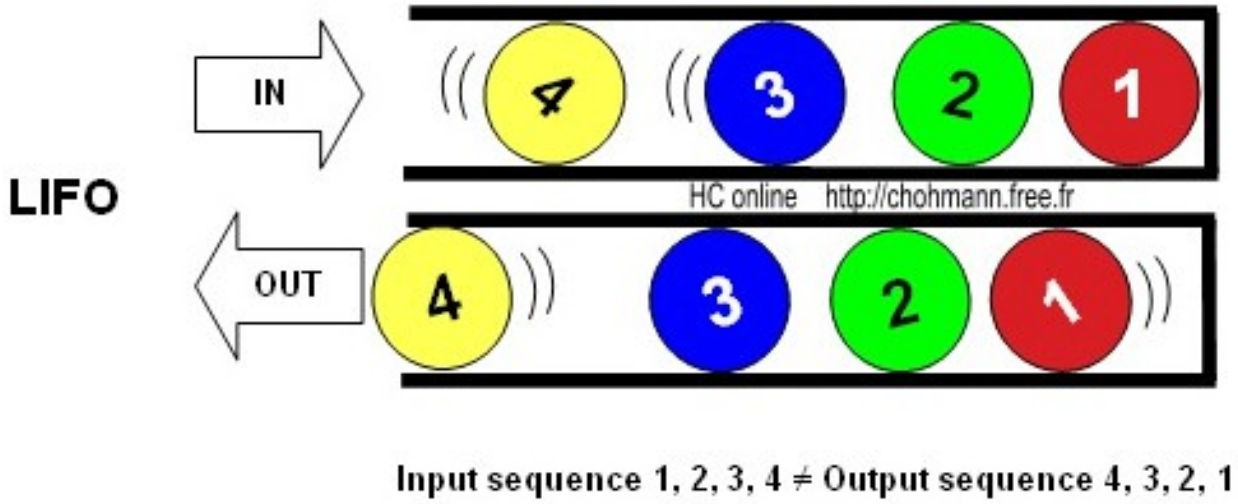
(word3 (word2 (word1)))

Further, I will be focusing on ZQL pipeline query format, but in ZQ module documentation and ZQL command-line tool documentation, I will show you, how you can send a Queries to the Zabbix Lisp-Style… Stay tuned.

Stack

In the beginning, I was talked about a stack. Yes, ZQL Query processor is a nearly full implementation of the stack machine. All “context” and “stack” words, responsible for data acquisition and manipulations operates with that LIFO stack.

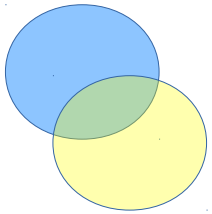
The “context” and “stack” words are passing the reference to a “context”. The context is a data structure inside the ZQ Environment, which containing the parameters for connection to the Zabbix server and



a set of the methods for the stack manipulation. You can push the data to the stack, pull the data from the stack, peek the data (i.e. get the reference on the top element of the stack without pulling it from the stack), and the others.

Context words

In the beginning, thy shall be a “context word”.



“Context word” serves as important “capital letter” of your ZQL query. It creates and passes the first context. There are special case, where you can skip the first context word, but in order to do so, you do need to prepare your context in a special way. I will explain, how to do this later.

There is currently only one “context word” is useful for the user, is (ZBX …)

As you already figure out, the ZQL words are context sensitive. So, the (ZBX) is (ZBX), not (Zbx) or (zbx).

YES

NO

Context words

Since we've started to talk about the "words", it is a right time to talk about positional parameters, symbols and strings.

symbol – is a pre-defined constant. Symbols are alpha-numeric, case-sensitive, can not begin with number or special character and must be pre-defined before use. Otherwise, your query will throw an error

Example:

NONE, TRUE, FALSE, Ne, Eq ...

Example:

"Hello world!"
"Yes, I can contain the \"quotas\""

string – is a list of characters, enclosed in quotas ("). If you are using quota as a part of the string, the quota must be escaped with forward "\".

Context words

Positional parameters is a values (each of them could be an s-expression), which are passed to the ZQL “word”. Positional parameters are separated with one or more spaces.

(word TRUE “String” 42 3.14 (time.time))

If word requires some positional parameters and you do not provide them in your query, defaults may be used. If the word do require parameter and you didn’t provide it and there is no default specified, the ZQL will throw an exception

Context words

The only “context word” useful to the ZQL user at this point is (ZBX). Again, please do remember, the “words” are case sensitive

(ZBX NEW “<server>” “<environment>”)

This “word” takes three positional parameters:

- Reference to the previous context. At this moment, only accepts symbol NEW
- Name of the Zabbix server context in the environment
- Name of the environment in the “multi-environment” application. For the ZQL command-line tool there is only one environment, and it’s name “default”

(ZBX) “word” returns the reference to the Zabbix server context.

Stack manupulating words

There are two types of the “words” which is manipulating the stack. The “words” which performs the data acquisitions and place acquired data on the stack. And the “words”, which have nothing to do with data acquisition, they just manipulating already data in the stack.

ZQL is trying to be safe with the data. If in doubt, it will tend to do “be safe then sorry” and perform as little destructive operations with the data as possible. Unless you ask for it.

Stack manipulating words accepts context as the first parameter. But you do not have to pass it. ZQL runtime will do it for you. And stack manipulating “words” must return the current context. Please note this, if you are planning to implement your own “words”.

Getting data from Zabbix

(Hosts)

Requesting the list of the Hosts from the Zabbix server, and places result in the stack as hash table, where the key is ‘HOST’ and the value is information about hosts as it returned by ZabbixAPI: list of the dictionaries.

Example query:

(ZBX) (Hosts)

This query will return the server context and hosts information will be stored as last value of the context stack

Getting data from Zabbix

(Interfaces)

Requesting the list of the Hosts Interfaces from the Zabbix server, and places result in the stack as hash table, where the key is ‘INTERFACES’ and the value is information about interfaces as it returned by ZabbixAPI: list of the dictionaries.

Example query:

(ZBX) (Interfaces)

This query will return the server context and interfaces information will be stored as last value of the context stack

Getting data from Zabbix

(Hostgroups)

Requesting the list of the Host Groups from the Zabbix server, and places result in the stack as hash table, where the key is ‘HOSTGROUPS’ and the value is information about host groups as it returned by ZabbixAPI: list of the dictionaries.

Example query:

(ZBX) (Hostgroups)

This query will return the server context and host groups information will be stored as last value of the context stack

Getting data from Zabbix

(Templates)

Requesting the list of the Templates from the Zabbix server, and places the result in the stack as hash table, where the key is ‘TEMPLATES’ and the value is information about templates as it returned by ZabbixAPI: list of the dictionaries.

Example query:

(ZBX) (Templates)

This query will return the server context and templates information will be stored as last value of the context stack

Getting data from Zabbix

(Items)

Requesting the list of the Items from the Zabbix server, and places the result in the stack as hash table, where the key is ‘ITEMS’ and the value is information about the items as it returned by ZabbixAPI: list of the dictionaries.

Example query:

(ZBX) (Items)

This query will return the server context and items information will be stored as last value of the context stack

Getting data from Zabbix

(Triggers)

Requesting the list of the Triggers from the Zabbix server, and places the result in the stack as hash table, where the key is ‘TRIGGERS’ and the value is information about the triggers as it returned by ZabbixAPI: list of the dictionaries.

Example query:

(ZBX) (Triggers)

This query will return the server context and triggers information will be stored as last value of the context stack

(Proxies)

Requesting the list of the Proxies from the Zabbix server, and places the result in the stack as hash table, where the key is ‘PROXIES’ and the value is information about the proxies as it returned by ZabbixAPI: list of the dictionaries.

Example query:

(ZBX) (Proxies)

This query will return the server context and proxies information will be stored as last value of the context stack

From Context to a Data

(Out)

(Out) is a “border crossing” word in ZQL. All it’s doing, is pulling data from the stack and return it as the value. In ZQL pipeline all words after (Out) will be manipulating with the reference to the data, returned by that “border word”, not with the context.

Example query:

(ZBX) (Hosts) (Out)

The “word” (Hosts) will send an Zabbix API call to the Zabbix server, which context is returned by (ZBX), word, then push the result of this query to the stack. Word (Out) will pull data from the stack and as it is no more further processing, the result of this query will be returned to the caller.

(Pretty_Json)

Sometimes, we do need to do the pretty formatting of the JSON data returned by ZAPI (and ZQL). For that, we do have the “word” (Pretty_Json). This part of the pipeline will take data on the input and if incoming data is dictionary, it will format it in human-pleasing JSON.

Example query:

(ZBX) (Hosts) (Out) (Pretty_Json)

The “word” (Hosts) will send an Zabbix API call to the Zabbix server, which context is returned by (ZBX), word, then push the result of this query to the stack. Word (Out) will pull data from the stack and pass it to (Pretty_Json) “word”. The “word” (Pretty_Json) will convert data into JSON and will return it as result of that query.

Data, data and more data

Before introducing some more advanced functions, I will discuss the “keyword argument”. We already spoke of the “positional arguments” to the ZQL words. Let me remind you:

(word “parameter” “parameter”)

Those are space separated elements of the s-expressions. As an extension to the s-expression, ZQL supports positional parameters described above and a keyword parameters, whoch passed to the “word” as a Dictionary or a Hash table structure. So, how do you pass a keyword parameters ?

Example:

(word :key value :key2 value2 ...)

As you see, keyword parameters are passed as a Dictionary, consisting of space-separated key-value pairs. Key is pre-pended with a colon “:”. All standard requirements for the keys or values must be met. Keys are the strings and must be defines as ones. Values must be one of the supported types, such as : Numbers, Strings, List, Dictionary. Anything, supported by the Python.

(Filter)

So, here is the “word” which intended to filter the values inside ZQL. It is close to the “select” keyword of the SQL. (Filter) accepts the following parameters:

```
(Filter COMPARATIVE_FUNCTION
                    *[optional positional filtering parameters]
                     optional keyword filtering patameters
)
```

Data, data and more data

Before discussing of what is COMPARATIVE FUNCTION, let me introduce to you the format of positional filtering parameters. It consists of the one or more 3-element lists. The format of the filtering element is:

[<keyword in the dataset> COMPARATIVE_FUNCTION <value>]

The subelements of the filtering element are space-separated and enclosed in "[" "]"

Generally, your dataset is returned as the List of the Dictionaries, containing key-value pairs. <keyword> parameter, or the first element of the filtering element will define the key which will refer the value from the dataset which we will test with the help of the comparator. The <value>, or the third element in the filtering element, will define "comparative or base value" which will be used during the filtering. And finally, <COMPARATIVE_FUNCTION>, or the second value of the filtering element, is a name of the "lambda expression" which will perform operation on the value from the dataset and the <value> from the filter. If lambda-expression returns "True", then data element is passed the filter.

Data, data and more data

COMPARATIVE FUNCTIONS

Name	Description
Eq	One parameter is equal to another
Ne	One parameter is not equal to another
Lg	Data from dataset is smaller than the <value>
Lge	Data from dataset is smaller or equal than the <value>
Gt	Data from dataset is larger than the <value>
Gte	Data from dataset is larger or equal thean the value

Data, data and more data

Examples:

[“hostid” Eq 10084] or [hostid Eq 10084]

Filter data after the (Hosts), filter-out everything but the record where key “hostid” is 10084. You can use second form, because “hostid” also defined as a symbol

[“proxy_id” Ne 0] or [proxy_id Ne 0]

Filter data after the (Hosts), filter-out all records for which value of the key “proxy_id” is not 0. This filter-expression will remove hosts which is not handled by the proxy.

Data, data and more data

Keyword-based filtering expressions.

:key <value>

Since, the keywords are nothing but the key-value pairs, here where the first parameter of the (Filter) “word”, the COMPARATIVE_FUNCTION comes to play. So, what is the COMPARATIVE_FUNCTION passed as first parameter ? It is a function applied to all keyword-based expressions. So the expressions:

(Filter TRUE [“hostid Eq 10084])

and

(Filter Eq :hostid 10084)

Are essentially the same. In fact, you can mix positional and keyword-based filtering expressions in the same “word”. When processed, positional filters comes first in the order in which they are positioned and keyword-based expressions comes second. But bear in mind. The first parameter, the COMPARATIVE_FUNCTION is applicable **ONLY** to the keyword-based filters. If you are not using keyword-based filter expressions, you still **can not omit it**. Pass the TRUE symbol.

Data, data and more data

Examples:

(Hosts) (Filter TRUE [“status” Ne 0]) (Out)

This query will return you the list of the hosts which status is not 0

(Hosts) (Filter Eq :proxy_id 0) (Out)

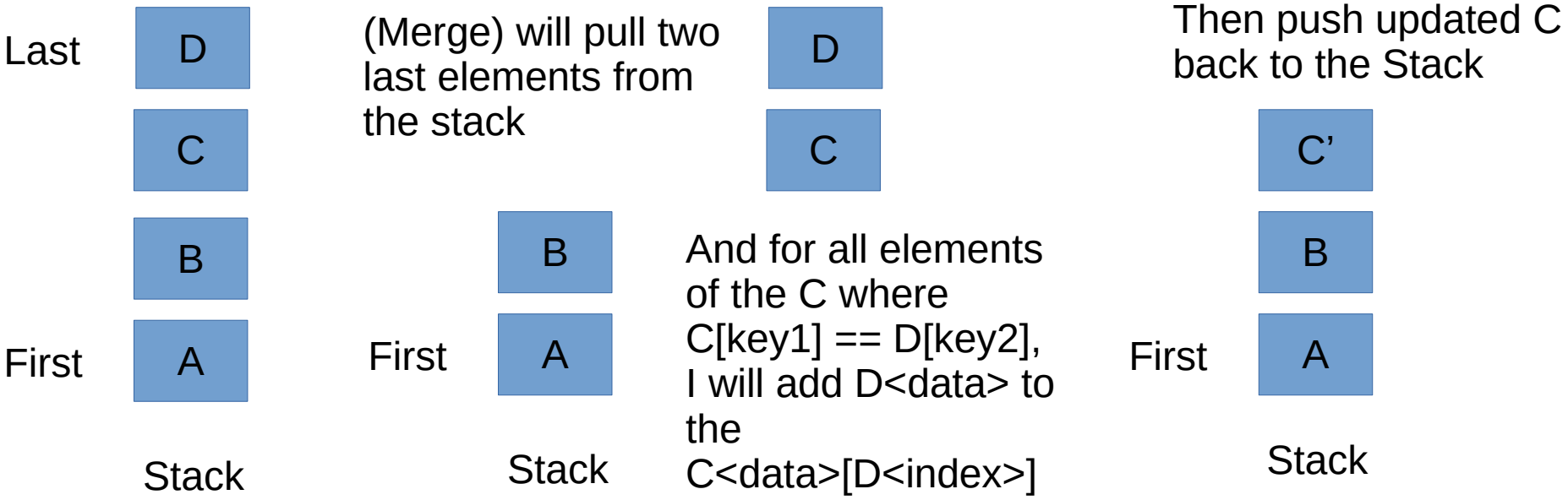
This query will return you the list of the hosts which is not handled by the proxy

(Hosts) (Filter Eq [proxy_hostid Eq 0] :host "Zabbix server") (Out) (Pretty_Json)

This query return you the information about host “Zabbix server” if it is not handled by a proxy.

Stack, stack and more Stack....

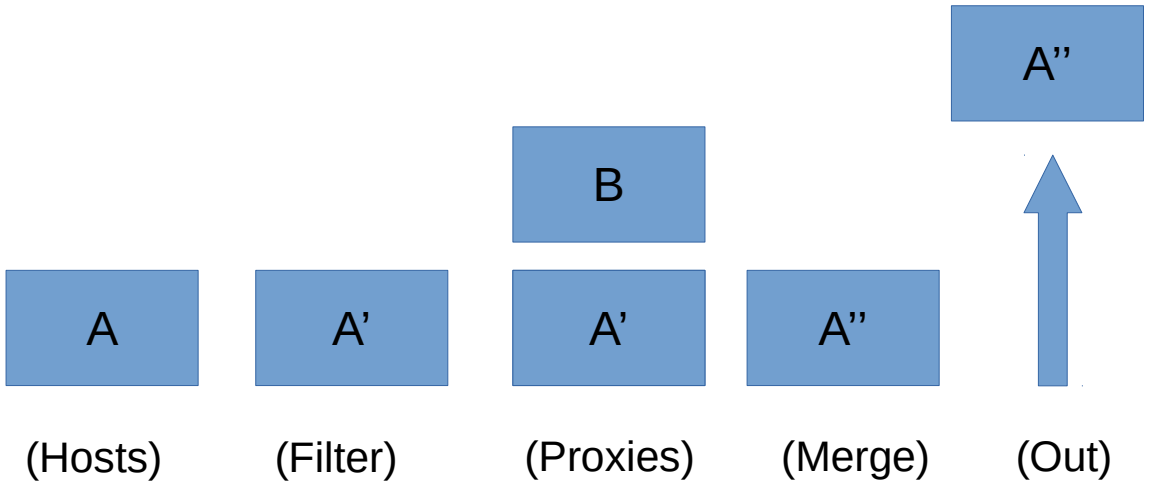
ZQL is a pipeline and stack-oriented language. This means that the “words” of the language ether operates with the data or with the stack. I’ve introduces some words operating with the data. Now, it is a time to discuss the stack operations. And we will start with most complicated “word” - (Merge <key1> <key2>)



Stack, stack and more Stack....

Example:

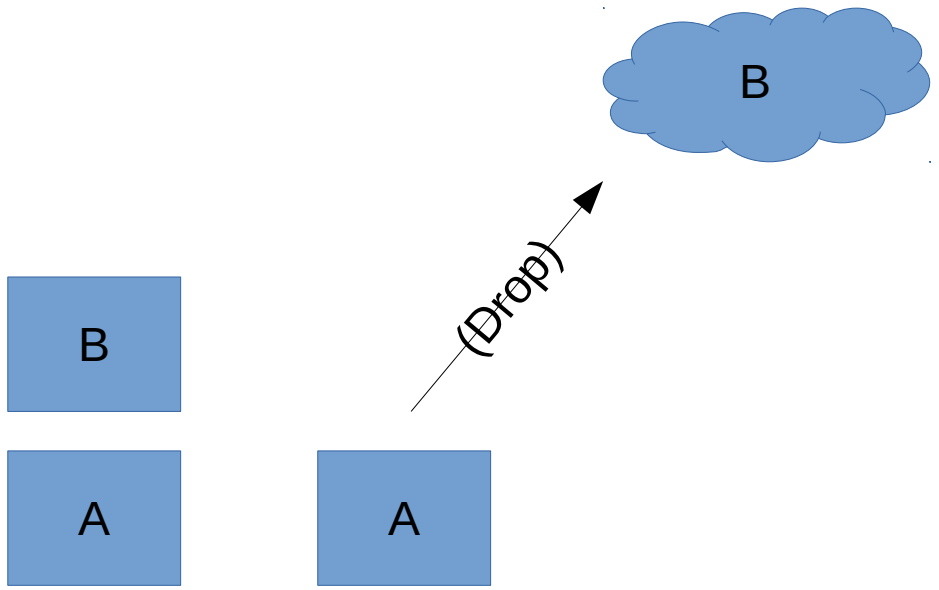
(Hosts) (Filter TRUE [proxy_hostid Ne 0])
(Proxies) (Merge proxy_hostid proxyid)
(Out)



Stack, stack and more Stack....

(Drop)

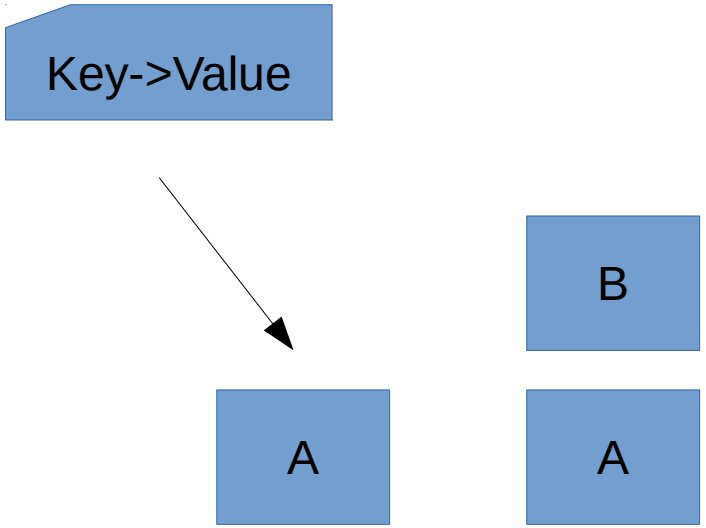
Discards the element from the top of the stack.



Stack, stack and more Stack....

(Push <key> <value>)

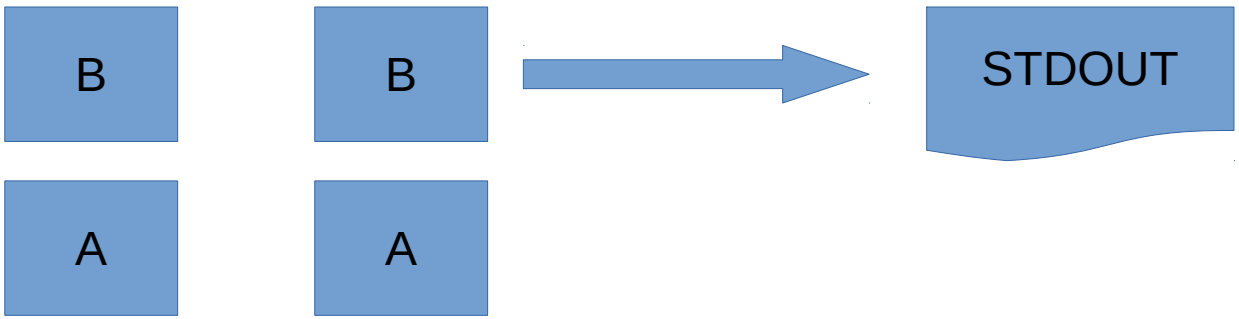
Push the key-value pair to the top of the stack.



Stack, stack and more Stack....

(Peek)

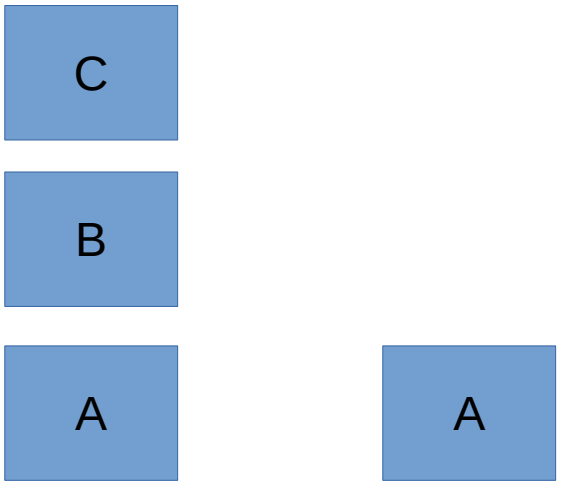
Print the value of the last element of the stack
on STDOUT without removing it from the stack



Stack, stack and more Stack....

(Until <key>)

Remove elements from the stack, until you'll come to the element with specified <key>. If <key> not found, stack will leave empty.



(Until "A")

Final note

So, I am welcoming you to the world where you can query and manipulate the Zabbix data and configuration ether programmatically, from the Python module, or from your shell scripts. The world, in which you shall only have the “domain specific” knowledge, means knowledge of Zabbix. ZQL will take care of the rest.

And now, it is a time for Questions.