# Javascript Object Paths

# By Trevor Scharf

This framework was designed to dig through large amounts of data fast. I loved the jquery selectors that were used for traversing the DOM and decided to create one for javascript arrays/objects since I couldn’t find anything similar on the web outside of DOM selection. It follows many of the concepts of the jquery selectors but has quite a few differences in some cases.

So. Lets say I have an array of people. I want to find all people who have a first name that is ‘Trevor’. I will do something like the below. Digging deeper down the path is used by the separator ‘=>’.

var trevors = $filter(people, "[firstName=Trevor]");

Now lets say I want an array of people whos firstName contain ‘k’ or ‘e’.

var newCol = $filter(people, " [firstName=~(k|e)] ");

I want to find all the pets whos names contain k or e that are pets on the people.

var newCol = $select(people, "pets=>[name=~(k|e)] ");

The difference between filter and select is that filter returns the top level collection filtered down from the query while select digs through and will actually return let’s say the pets that were pulled in the query. If I used $filter on the previous one it would return all people who have pets that have a name containing with k or e.

Now let’s say I want to dig through the people, find the pets and call a method on all pets.

This can easily be done by the following:

$invoke(people, "pets=>[name=~(k|e)]=>sayName");

By doing this all pets of the people whose names contain a k or an e will have their method sayName() called. And to pass parameters into these methods just do it as so:

$invoke(people, "pets=>getBabies=>[name=~(k|e)]=>sayName('HELLO')");

Now this would be cool if I could execute these methods as saved delegates. This can be done as so.

var delegates = $select(people, "pets=>getBabies=>[name=~(k|e)]=>sayName('HELLO')");

Now to invoke them you can do the following:

$invoke(delegates); or

$invoke(delegates.skip(1).take(2));

If through the path no ‘()’ is provided in the path you can invoke all the delegates through another parameter with their own custom parameters as so.

$invoke(delegates,’Hello World’);

The functions can easily be passed around and filtered as objects.

Now you may be wondering about the formatting of the selectors. All types are supported for filtering by properties like below:

[x^=y] : prop[x] starts with y

[x$=y] : prop[x] ends with y

[x\*=y] : prop[x] contains y

[x=~y] : prop[x] matches y

[x>y] : prop[x] > y

[x>=y] : prop[x] >= y

[x<y] : prop[x] < y

[x=y] : prop[x] equals y

[x!=y] : prop[x] not equals y

This works properly with dates also. It converts y if formatted as a date to be converted to a date object and compares them appropriately

Stored Path Procedures can be store and executed as shown below:

addProcedure("pets\_children", "pets=>getBabies=>[name\*='B']", "select");

addProcedure("sayName", "sayName('HELLO')", "invoke");

procedure(people, "pets\_children","sayName")

The defined structure is show below:

addProcedure(proc\_name,selector\_path,selector\_type(‘select’,’filter’,’invoke’))

and

proceduere(collection,\*procNames)