Delegate Queryables
—
A relational querying language based on JSON.

Through the development of the Delegate Protocol, it was readily apparent that there is a need for a sandboxed abstraction of SQL queries. The act of continually reinventing the wheel in order to provide an interface for querying certain fields is menial and quite time consuming—we just had to abstract it all away and make our lives easier. The *Delegate Queryables* interface started off as a simple component to the much larger Delegate Protocol, but it has become large enough to warrant its own document.

A Delegate Queryable shall be written using JSON. Each field within the JSON object shall correspond to the field that shall be queried in conjunction with other fields provided. The value of each field will specify how it should be queried. For example, we may wish to query all users whose informal user name contains the word ‘hello’ *AND* also have a creation date greater than some value:

```
{
        “name”: “hello”,
        “created”: “>1664562460”
}
```

Every field within a query will be on an AND basis within that query, meaning we are looking for instances where all of the conditions are present, not on an OR basis.

Fields within a Delegate Queryable can theoretically have any of the following types, which limits the querying options one may use on it:

- *string*
- *integer*
- *array*
- *bool*

The following operations can exist—usually being prefixed before the value of the field—and will be tied to the following types:

Operation:	Purpose:	Supported Types:	Example:
(none)	Finds an exact match.	*integer*, *string*, *array*, and *bool*—all types.	“name”: “John” “tags”: “programmer” “created”:

			“282828282” “friendly”: true
!	Negates the condition of the non-array query. MUST BE the first operation, or else it will be erroneous.	*integer*, *string*, and *bool*	“name”: “!john” *Find all names not john*
>	Finds greater than. For integers, it finds instances greater than the one provided. For strings, it finds strings with a length longer than the one provided.	*integer*, *string*	“created”: “>4939292292” “name”: “>60”
>=	Same as above, but it’s now greater than or equal to.	Same as above, but it’s now greater than or equal to.	Same as above, but it’s now greater than or equal to.
<	Same as above above, but it’s now less than.	Same as above above, but it’s now less than.	Same as above above, but it’s now less than.
<=	Again, same as above above above, but it’s now less than or equal to.	Again, same as above above above, but it’s now less than or equal to.	Again, same as above above above, but it’s now less than or equal to.
{	Find all instances where the string field contains something.	*string*	“username”: “{apple” *Find all usernames containing the word apple:* - *appleslayer* - *appleeater* - …
[“OR”, …]	Find all fields which have at least one of these items in its array.	*array*	“tags”: [“OR”, “programming”, “linux”, “chat”] *Find all channels with at least ‘programming’, ‘linux’, OR ‘chat’ as a tag.*
[“AND”, …]	Find all fields which must have	*array*	“tags”: [“AND”,

	every element in this list in their array.		"programming", "linux", "chat"] *Find all channels with 'programming' AND 'linux' AND 'chat' in its tags array.*
[!, *b*, …]	Negate the array query, where *b* is any of the specifiers above for the array searching. Note, if there is no *b* specifier, then you are negating an *array* equality query. The negation is distributive, following the rules of logic: NOT(A or B) = NOT A or NOT B NOT(A and B) = NOT A and NOT B	*array*	"tags": ["!", "AND", "programming", "linux", "chat"] *Find all channels that* ***don't*** *have 'programming' AND 'linux' AND 'chat' in its tags array.* "tags": ["!", "OR", "programming", "linux", "chat"] *Find all channels that is at least* ***without*** *'programming', 'linux', OR 'chat' as a tag.*