SaveQuery

Description

The **SaveQuery** API allows the Account Owner to create and save *Queries* that can be executed in the future.

> In apstrata, Documents that are saved in a Store can be retrieved by running a *Dynamic Query* or a *Saved Query*.
>
> A *Dynamic Query* is one that is built every time on the fly by specifying the condition, fields to return, sort order, and aggregates.
>
> A *Saved Query* is one whose definition is saved for later use. Rather than every time specifying the condition, fields to return, sort order, and aggregates, a developer can add a *Saved Query* whose definition specifies all the previously mentioned query constructs. Each *Saved Query* has a name which is used to run the *Query* and get back the results.
>
> Moreover, a developer can implement a pagination model for the matching results by specifying the number of *Documents* per page as well as the number of the page to return. This is very useful for cases where a huge number of Documents matches the *Query* condition. In these cases, the results will be returned in pages or chunks rather than a whole block which impacts the performance of the application at hand.
>
> Furthermore, when running a *Query*, a developer can look for documents matching a specific full text search criteria. In this case, documents satisfying both the *Query* condition and the full text search criteria will be returned in the results.
>
> In terms of security, *Documents* or fields within a *Document* are returned in the results only if their read ACL contains the User or one of the Groups of the User running the *Query*.
>
> A developer may run as many *Queries* as needed per store. However, based on the apstrata account type, there is a limit on the number of predicates used in the *Query* condition, on the number of *Documents* returned in the results, and on the total size of the result set.

It is often preferable to predefine available queries for an application, rather than rely on allowing dynamic Query by any account users (this can lead to higher costs if users, especially anonymous users, are allowed to make queries with any query condition). The customer can set some ACL on each saved query to control which account users (or anonymous) will be allowed to execute it.

Query Schema

Along with the SaveQuery request, the developer should send an xml schema containing all the details of the query to be saved.

Below is an example of the xml schema:

```
<query>
        <executeACL>all</executeACL>
        <store>testStore</store>
        <condition>
                <![CDATA[ firstName<string>={name}? AND lastName<string>={lastname} AND location<geospatial>
WITHIN (0, 0, {dist}) ]]>
        </condition>
        <ftsQuery>test</ftsQuery>
        <returnedFields>
                <field>age</field>
                <field>apsdb.documentKey</field>
                <field>location</field>
                <field>distance(location,'1','1')</field>
        </returnedFields>
        <sort>
                <![CDATA[ lastName<string:ASC> ]]>
        </sort>
        <count>true</count>
        <aggregate>
                <expression>avg($age)</expression>
                <page>true</page>
                <global>false</global>
                <groupBy>
                        <field>
                                <name>age</name>
                                <type>numeric</type>
                        </field>
                        <field>
                                <name>lastName</name>
                                <type>string</type>
                        </field>
                </groupBy>
        </aggregate>
        <resultsPerPage>10</resultsPerPage>
        <forceCurrentSnapShot>true</forceCurrentSnapShot>
</query>
```

The following table explains each tag of the above xml schema:

Name	Description	Default Value
executeACL	The list of users, groups or identifiers that can execute the saved query.	all
store	The name of the store on which the query will be executed.	
condition	The query condition to execute. Refer the the Query Condition section below for more details.	
returnedFields	The list of fields that should be returned when the query gets executed, note that in addition to the optional returnedFields requested, the query will return the document_key.	
sort	The list of fields on which to sort when the query gets executed.	
resultsPerPage	Determines the number of results per page to return when the query gets executed.	10
ftsQuery	Specifies a string value that is matched against the resulting fields that are indexed for full text search when the query gets executed.	
count	Specifies if the records count should be returned or not when the query gets executed (true or false)	False
forceCurrentSnapshot	In order to avoid data inconsistency, even for a short duration of time, the user can send this parameter with value "true" to query data from the master database server. <br /> Will query from slave database server if the value is set to “false”.	False

aggregate	The aggregate function to execute when running the query. It contains four elements: - The aggregate expression - The page and global flags that can be set to true or false to specify on which scope the aggregate function will be executed - The groupBy statement that is used in conjunction with the aggregate functions to group the result-set. For each field in the groupBy statement, you should specify its name and type as shown in the example above	

Query Condition

The query "condition" syntax is the same for saved queries and dynamic queries except for the concept of condition's variables that will be explained here.

When writing the condition of a saved query, the fields' values can be replaced by variables and set when running the query by passing the values as parameters to the Query API.
Let's take a look at the following example.

```
<query>
    <executeACL>anonymous</executeACL>
    <condition>
            <![CDATA[  firstName<string>={name} and lastName<string> = "Doe" ]]>
    </condition>
    <returnedFields>
            <field>*</field>
    </returnedFields>
</query>
```

The value of the field **firstName** has been replaced by the variable **name**. When calling the Query API to run this saved query, the parameter **name** will have to be sent in the request and the query will use its value to execute the condition. Passing the value "Jane" to the parameter **name** will return all the documents containing the fields **firstName** with the value "Jane" and **lastName** with the value "Doe".

The parameter name can set to be optional by adding a question mark after the closing bracket. If the optional parameter is not sent in the Query request then the whole predicate will be ignored.

For example, if the condition is **`firstName<string>={name}? and lastName<string> = "Doe"`** and the optional parameter **name** is not passed in the Query request, then all documents containing the field **lastName** with the value "Doe" will be returned.

Special mention should be made when dealing with geospatial fields and the "within" operator that consists of two parameters:

a geospatial reference point (represented as a pair of latitude/longitude decimal degrees), and a distance value (in kilometers).
Each parameters can be replaced by a variable like illustrated by the following examples.

```
<condition>
        <![CDATA[  location<geospatial> within (0,0,{distance}) ]]>
</condition>
```

```
<condition>
        <![CDATA[  location<geospatial> within ({point},{distance}) ]]>
</condition>
```

```
<condition>
        <![CDATA[  location<geospatial> within ({point},0.200) ]]>
</condition>
```

Note that the first parameter (the geospatial reference point) can only be replaced by one variable representing the latitude/longitude pair, the latitude and longitude cannot be replaced by two different variables.

Query ACLs

Query ACLs can be set using predefined ACL identifiers:

1. creator, meaning that the creator of the query can execute it. (<executeACL>creator</executeACL>).

2. anonymous, meaning that anyone can execute the query (<executeACL>anonymous</executeACL>)

3. authenticated-users, only authenticated users can execute the query.

4. nobody, no one can execute the query. Note that the owner will have access even if ACL is set to "nobody".

Schema Validation

The schema sent by the user should conform to the xsd. All ACLs should be of the form ((group:){0,1}[0-9A-Za-z-_]{1,128}(;){0,1})*

Specific Request Parameters

(Refer to Common Request Parameters)

Name	Description	Required	Default	Possible Values
apsdb.queryName	Represents the full query name that can consist of up to 5 folders and the query file name separated by the character '/'. *e.g.*, application/provisioning/user/list In the example above the **list** query is saved under the three folders **application** > **provisioning** > **user**	Yes		The same set of rules applies to a query name and a directory name: • Must begin with an alphabetic character. • Must end with an alphanumeric character or underscore (_). • Can contain alphanumeric characters, underscores (_) and periods (.). • Cannot have two or more consecutive periods. Note: the full query name including any folders must be minimum 3 characters and maximum 64 characters long.
apsdb.query	The xml schema containing the details of the query to be saved.	Yes		Xml schema
apsdb.update	Should be set to true when updating an existing query.	No	False	True False

Specific Response Elements

(Refer to Common Response Elements)

Specific Logical Errors

(Refer to Common Logical Error Codes)

Error	Message	Status Code
PARAMETER_REQUIRED		400
QUERY_SCHEMA_REQUIRED	The query parameter containing the query details is missing in the request.	400
SAVED_QUERY_NOT_FOUND	The query [queryName] that you are trying to update was not found.	404
DUPLICATE_QUERY_NAME	Cannot create query [saveQueryName] because it already exists.	400
INVALID_QUERY_NAME	Invalid query name [queryName].	400
INVALID_QUERY_WILDCARD_USE	Wrong syntax, cannot use wildcards at the beginning of the pattern.	400
INVALID_AGGREGATE_SYNTAX	Aggregate [aggValue] is invalid.	400
INVALID_AGGREGATE_SYNTAX_MISSING_FIELD	The field is missing from the aggregate expression [aggValue].	400
BAD_AGGREGATE_FIELD_TYPE	Cannot have an aggregate for field [fieldName]. Metadata fields and non-numeric fields cannot be used in the aggregate functions.	400
INVALID_QUERY_FIELDS_SYNTAX	You can send either a comma separated list of field names or the symbol * in the parameter "queryFields" but not both.	400
INVALID_QUERY_SYNTAX	The query request must contain either requested fields, a count, or an aggregate expression.

 Or

 The query request must contain either a query or a full text search.	400
MAX_PREDICATES_EXCEEDED	The maximum number of predicates allowed is: [predicatesLimit].	400

INVALID_QUERY_CONDITION		400
INVALID_PARAMETER_VALUE	The parameter [parameterName] must have a value of 'true' or 'false'.	400
INVALID_SCHEMA	[schema] does not match [pattern].	400

Examples

Sample Request

```
Request URL: http://sandbox.apstrata.com/apsdb/rest/[authenticationkey]/SaveQuery?apsws.time=[timestamp]&apsws.
authSig=[signature]
```

POST parameters:

```
apsdb.queryName = [query_name]
apsdb.query = [query_xml_definition]
apsdb.update = [true|false]
```

Sample XML Response

Success XML:

```
< response xmlns="http://www.apstrata.com/services/schemas/apstrata_database_response.xsd" >
   < metadata >
      < requestId >xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx< /requestId >
      < status >success< /status >
   < /metadata >
< /response >
```

Failure XML:

```
< response xmlns="http://www.apstrata.com/services/schemas/apstrata_database_response.xsd" >
   < metadata >
      < requestId >xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx< /requestId >
      < status >failure< /status >
      < errorCode >[errorCode]< /errorCode >
      < errorDetail >[failMsg]< /errorDetail >
   < /metadata >
< /response >
```

Sample JSON Response

```
{"response": {
   "metadata": {
      "requestId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
      "status": "success"
   }
}}
```