EWSearchTableWithQuery

Executes a preconfigured named Saved Search against the specified table and/or an ad-hoc query and returns and array of record data for the records that match the criteria.

Syntax

```
EWWSBaseUserObject[] os = ew.EWSearchTableWithQuery(String sessionId, String
              tableName, String[] fieldNames, String searchName, String query);
```

Usage

Use EWSearchTableWithQuery call to search for records in the specified table based on a Saved Search pre-configured in the GUI and/or to filter the records with an ad-hoc query.

Rules and Guidelines

When querying records, consider the following rules and guidelines:

- The username that was used to obtain the specified session token must have sufficient access rights to read individual records within the specified table. Please verify specific permissions via **Setup > Access > Manage Groups > *(Edit Group*) > Table > (*Edit Table*) > Permissions.**
- Agiloft allows specifying fine-grained access permissions on the field level. The username that was used to obtain the specified session token must have sufficient access rights to be able to read field content. Please verify specific permissions via **Setup > Access > Manage Groups > (*Edit Group*) > Table > *(Edit Table*) > Field Permissions**.
- This call does not return records that have been deleted.
- This method never returns null. In the case when no records are found an empty array is returned.
- **Special note on memory management:** As WS integration implies pass-by-value semantics, the memory allocated for the resulting array will be released once the data is sent to the client and the server-side JVM's garbage collector considers it eligible for discarding.
- If the client-side environment is the one with a garbage collector, the memory used by client-side array will be cleared once the client-side garbage collector considers it eligible.

- When the client environment uses explicit memory management, the client is responsible for freeing up the used memory explicitly.
- When using EWSearchTableWithQuery method, only the fields explicitly listed in the call are read. Within the values returned for the fields requested explicitly, a null value, where nillable="true", means the actual null value was retrieved. However, the rest of the fields – those not listed in the fieldNames array – will also appear on the wire as nillable="true" elements due to limitations of the underlying Web Services stack.
- To read all fields, use "*" string constant as the only element in the fieldNames array.
- The main difference from using the EWSelectAndRead method, which uses an SQL where clause, is that ad-hoc queries operate on a higher level, can use logical field names, are capable of recognizing choice values and high-level relationships between table fields, and can use advanced and time-based criteria.
- If the query doesn't parse according to the grammar, an attempt is made to parse the parameter value as a sequence of identifiers using a different grammar. If both fail, the parameter value is treated as a Full-Text Search query.
- The ad-hoc query grammar is described at the end of this section.

Steps for Searching Records with a Saved Search and/or Ad-hoc Query

1. Optionally create a Saved Search in the GUI.
2. Perform the call using the name of the search and additionally filter the results with an ad-hoc query or use the ad-hoc query without the search.
3. Handle the results, specifically the situations where there are no elements, one element, or more than one element in the returned array.

Example Task

In MyKB knowledgebase, as user A, find all cases assigned to the user used to login with low priority. Return summaries as a String array.

Completion of the task is performed by the following steps:

1. Login to MyKB with "A" and "password" and English as the local language.
2. Search for cases using My Assigned search, additionally filtering by low priority.
3. Logout

Sample Code - Java

You can generate sample Web Services code for any table by selecting **Setup > Table > [Select Table to Edit] > API > Download Sample**.

```
 public String[] search() throws Exception {
  EWServiceAPI binding = new EWServiceAPIServiceLocator().getDemo();
  try {
    String sessionId = binding.EWLogin("MyKB", "A", "password", "en");
    EWWSBaseUserObject[] records = binding.EWSearchTableWithQuery(sessionId, "case",
        new String[] {"summary"}, "My Assigned", "Priority=Low");
    String[] result = new String[records.length];
    for (int i=0; i<records.length; i++) {
          result[i] = records[i].getSummary();
    }
    return result;
  } finally {
    binding.EWLogout(sessionId);
  }
}
```

Arguments

Name	Type	Description
sessionId	String	Session token
tableName	String	The name of the table where the query has to be performed.
fieldNames	String array	The list of fields to read
searchName	String	The optional name of the Saved Search to run
query	String	The ad-hoc query

Response

An array of the records as descendants of EWWSBaseUserObject - a complex structure described in WSDL.

Faults

EWSessionException - client not logged in or session has expired; client should re-login.

EWPermissionException - user used to create the session lacks sufficient privileges to run the query.

EWWrongDataException - client has supplied wrong data.

EWOperationException - the operation has been blocked by an Agiloft function, for example a table-level lock.

EWIntegrityException - specified table cannot be found or its primary key cannot be identified.

EWUnexpectedException - an unexpected exception has happened; the admin user should report this for investigation.

Informal Grammar Description for Ad-hoc Queries

Field names are usually column labels as seen in the UI. However, DB and User column names are accepted too. Both field names and values may be surrounded by single quotes ('). If they contain spaces or some weird characters then quoting is mandatory. For example:

Example	Result
Priority=Low	OK
'Priority'='Low'	OK
Bug Priority=Low	Invalid
'Bug Priority'=Low	OK
'Bug Priority'=Very Low	Invalid
'Bug Priority'='Very Low'	OK

Simple criteria

Simple criteria has the form of

```
<field name><operator><value>
```

where operator is one of:

Operator	Definition
=	equals
!=	not equals
~=	contains
!~=	doesn't contain
>=	greater or equals
<=	lesser or equals
>	greater
<	lesser
<<	included by
!<<	not included by

The included/not included by operators (<<, !<<) expect a comma-separated list of values, without spaces, at the right-hand side of the equation and checks if field value is included (or not-included) in this list. In other words, Priority << High,Low is a short-hand for Priority=High || Priority=Low, where || is the OR operator, as described below.

Logical criteria

Allows to combine other criterias using AND and OR operators. '&&' is AND, '||' is OR.

Operator precedence

Expression is evaluated from left to right, braces may be used for grouping. For example 'A && B || C' means '(A && B) || C'.

Time-based criteria

Allows to set relative date constraints. The form is <field name><operator><mode><value>, where operator is one of =,!=,<,>,<=,>=, mode is either '-', which means 'old', '+', which means 'in the future' or '#', which means 'absolute'. 'value' is an integer followed by a single character:

m	minute
h	hour
w	week
M	month
y	year

Examples:

Date<-1y	'Date' is less than one year old
Date>=+10m	'Date' is greater or equal than 10 minutes in the future
Duration=#2h	'Duration' is exactly two hours

Currently, more complex expressions like 'two years, one month and three hours' are not supported.

Advanced criteria

Advanced criteria has the form of <field name>:<from>-><to> and means 'field <field name> has changed from *from* to *to*' Either from or to – but not both at the same time – may hold '?' meaning 'any value'.

This criteria searches through record history, and thus the history column must exist and must track changes to the specified field. All simple and time-based criteria have implicit 'now' flag set, which means that they will match the current record state, not the state when advanced criteria has been satisfied. In the other words, if we have a record with the following modification history, with the bottom state being the most recent:

State=Open, Priority=Low State=Closed, Priority=Low State=Closed, Priority=High
Then 'State:Open->Closed && Priority=Low' will not find it, but 'State:Open->Closed && Priority=High' will.

More examples

- Status=Open && ('Assigned To' = john || 'Assigned To' = jane)
- (Priority>High || Summary ~= Urgent) && State:Closed->Reopened
- OS=Windows,Linux && 'Modification Date' < -1y