Use the Bullhorn SOAP API Query Operation

One of the most powerful operations in the Bullhorn SOAP-based web services APIs is the query operation. This operation allows you to construct SQL-like queries for a particular type of entity.

The Bullhorn query operation is built on top of Hibernate, an object-relational mapping tool that exposes relational data as a series of objects. The Hibernate Query Language (HQL) is based on standard SQL but adapts its concepts to an object-oriented language (Java). The query operation exposes a subset of the operations supported by HQL.

Developers already familiar with Hibernate (or other O/R environments) will find the syntax used by the Bullhorn APIs familiar. For those developers accustomed to writing straight SQL, the following examples should give you a good starting point.

For detailed information about the query operation, consult the reference documentation.

Building a basic query

To execute a query, you must construct a query DTO and then pass it in when you call the query operation. The two most important fields in the query DTO are the entityName property, where you specify the name of the entity you are querying for, and the where property, which contains the where clause for your query. Within the where property, you can specify a single parameter or create a more complex query using AND, OR, or NOT. Following are a few examples of simple queries.

```
// Specify that you are querying for candidate DTOs.
myQuery.entityName = "Candidate";
// Find candidates whose last name begins with Smith (e.g., Smithson, etc.)
myQuery.where = "lastName LIKE 'Smith%'";

// Find candidates that were added in 2009 or later
myQuery.where = "dateAdded > '1/1/09'";

// Find candidates with the last name Smith or Jones
myQuery.where = "lastName = 'Smith' OR lastName = 'Jones' ";
```

A few important notes about dates. While you specify date values using a variety of string formats, before the query is executed they are converted into datetime values. This is important in specifying date ranges. For example, the expression shown above would return any candidates that were added after 12:00:01 AM on January 1, 2009, not just candidates add on January 2nd or later (all times EDT).

Using relationships to build more complex queries

To perform more complex queries, you can use the properties of entities that are linked to the DTO, either through a 1:1 relationship or through a 1:many relationship (called an association).

To use a 1:1 relationship in a query, you use object syntax. For example, the Candidate DTO's owner property refers to an instance of the CorporateUser DTO. You can include properties of the CorporateUser DTO in your query using "dot" syntax.

```
// Find candidates whose owner has the last name "Beeblebrox"
myQuery.entityName = "Candidate";
myQuery.where = "owner.lastName = 'Beeblebrox'";
// Find candidates whose owner has the ID of 12345
myQuery.entityName = "Candidate";
myQuery.where = "owner.id = 12345";
```

You need to use a different syntax to include an association in you query. Associations are used for 1:many relationships. For instance, a Candidate can have more than one category. You can find the categories for any particular Candidate using the getAssociationIds operation. To use associations in your query, you must use the special elements() function, as follows.

```
\\ Find candidates of category 254991
myQuery.entityName = "Candidate";
```

```
myQuery.where = "254991 = some elements(categories)";
\\ Find candidates of category 254991 owned by the user with ID 12345
myQuery.entityName = "Candidate";
myQuery.where = "owner.id = 12345 AND 254991 = some elements(categories)";
```

Customizing your query

In addition to the query parameters included in the where clause, you can also control your query using the other properties of the query DTO. You can limit the number of results that will be returned using the maxResults properties, as follows:

```
myQuery.maxResults = 300;
```

You can also control the order in which the ids of your results are returned using the orderBys property. In C#, this argument takes an array of String objects, as follows:

```
// Create the string array to hold the properties you want to use to order your results
String[] myStringArray = new String[2];

// Add the names of the fields by which you want to order results. Similar to the
relationships
// example above, you must use object syntax to refer to the id of the entity you are
querying
// for or any related entities. The following will order the results first by the IDs of
the
// recruiter that owns the candidate, then by the ID of the candidate.

myStringArray[0] = "owner.id";
myStringArray[1] = "id";
myQuery.orderBys = myStringArray;
```

Summary

This short overview should give you the tools you need to get started with the query operation. For more information, see the API documentation, or the other examples published on this site.