ServiceNow Application Developer

Server-side Scripting > GlideRecord

The *GlideRecord* class is the way to interact with the ServiceNow database from a script. See the ***GlideRecord* API (https://developer.servicenow.com/app.do#!/api_doc?v=madrid&id=c_GlideRecordScopedAPI)** reference for a complete list of methods.

GlideRecord interactions start with a database query. The generalized strategy is:

1. Create a *GlideRecord* object for the table of interest.
2. Build the query condition(s).
3. Execute the query.
4. Apply script logic to the records returned in the *GlideRecord* object.

Here is what the generalized strategy looks like in pseudo-code:

```
// 1. Create an object to store rows from a table
var myObj = new GlideRecord('table_name');

// 2. Build query
myObj.addQuery('field_name','operator','value');
myObj.addQuery('field_name','operator','value');

// 3. Execute query
myObj.query();

// 4. Process returned records
while(myObj.next()){
    //Logic you want to execute.
    //Use myObj.field_name to reference record fields
}
```

> **NOTE**: The *GlideRecord* API discussed here is a server-side API. There is a client-side *GlideRecord* API for *global* applications. The client-side *GlideRecord* API cannot be used in scoped applications.

Build the Query Condition(s)

Use the *addQuery()* method to add query conditions. The *addQuery* operators are:

- ***Numbers***: *=*, *!=*, *>*, *>=*, *<*, *<=*
- ***Strings***: *=*, *!=*, *STARTSWITH*, *ENDSWITH*, *CONTAINS*, *DOES NOT CONTAIN*, *IN*, *NOT IN*, *INSTANCEOF*

The *addQuery()* method is typically passed three arguments: field name, operator, and value. In some scripts you will see only two arguments: field name and value. When the *addQuery()* method is used without an operator, the operation is assumed to be *=*.

When there are multiple queries, each additional clause is treated as an AND.

Queries with no query conditions return all records from a table.

If a malformed query executes in runtime, all records from the table are returned. For more strict query control you can enable the *glide.invalid_query.returns_no_rows* property which returns no records for invalid queries.

Iterating through Returned Records

There are several strategies for iterating through returned records.

The *next()* method and a *while* loop iterates through all returned records to process script logic:

```
// iterate through all records in the GlideRecord and set the Priority field
value to 4 (low priority).
// update the record in the database
while(myObj.next()){
  myObj.priority = 4;
  myObj.update();
}
```

The *next()* method and an *if* processes only the first record returned.

```
// Set the Priority field value to 4 (low priority) for the first record in the
GlideRecord
// update the record in the database
if(myObj.next()){
  myObj.priority = 4;
  myObj.update();
}
```

Use the *updateMultiple()* method to update all records in a *GlideRecord*. Scripts using the *updateMultiple()* method you MUST set field values using the *setValue()* method.

```
// When using updateMultiple() use the setValue() method.  If you do
myObj.priority = 4, ALL
// records in the table will be updated and not just the GlideRecord records.
myObj.setValue('priority',4);
myObj.updateMultiple();
```

Counting Records in a GlideRecord

The *GlideRecord* API has a method for counting the number of records returned by a query: *getRowCount()*. Do not use the *getRowCount()* method on a production instance as there could be a negative performance impact on the database. To determine the number of rows returned by a query on a production instance, use *GlideAggregate*.

```
// If you need to know the row count for a query on a production instance do
this
var count = new GlideAggregate('x_snc_needit_needit');
count.addAggregate('COUNT');
count.query();
var recs = 0;
    if (count.next()){
      recs = count.getAggregate('COUNT');
  }
gs.info("Returned number of rows = " +recs);

// Do not do this on a production instance.
var myObj = new GlideRecord('x_snc_needit_needit');
myObj.query();
gs.info("Returned record count = " + myObj.getRowCount());
```

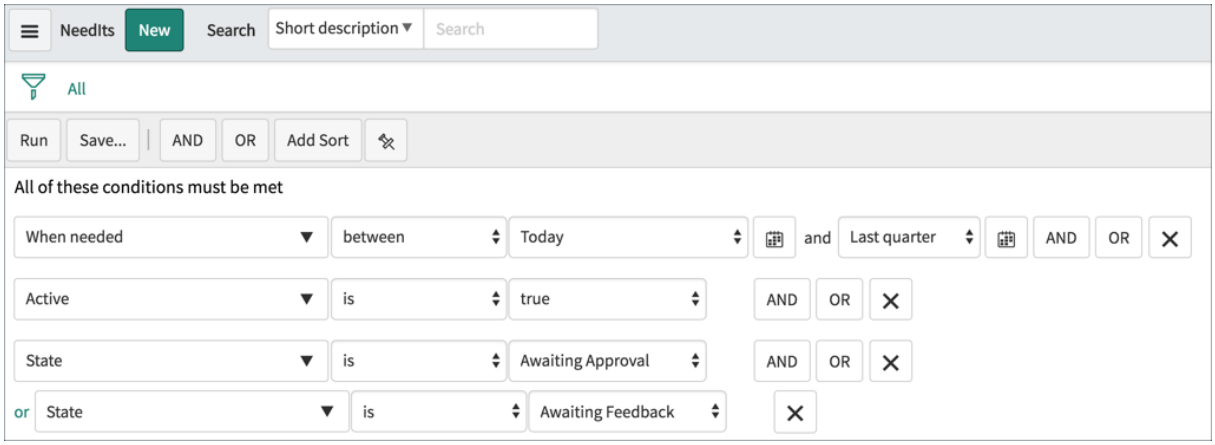
Encoded Queries

As already discussed, if there are multiple conditions in query, the conditions are ANDed. To use ORs or create technically complex queries, use encoded queries. The code for using an encoded query looks like this:

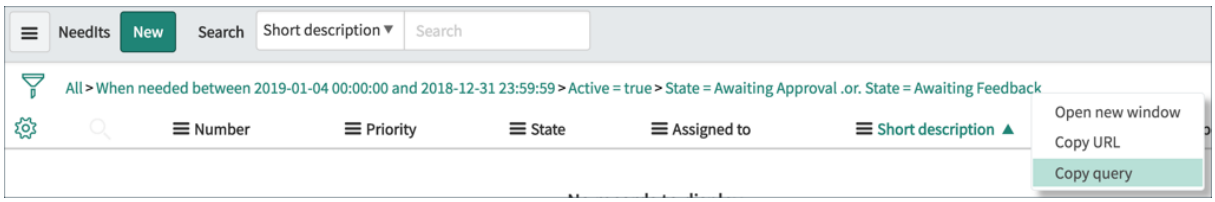
```
var myObj = new GlideRecord("x_snc_needit_needit");
myObj.addEncodedQuery('<your_encoded_query>');
myObj.query();
while(myObj.next()){
  // Logic you want to execute for the GlideRecord records
}
```

The trick to making this work is to know the encoded query syntax. The syntax is not documented so the best thing to do is let ServiceNow build the encoded query for you. In the main ServiceNow browser window, use the Application Navigator to open the list for the table of interest. If there is no module to open the list, type *<table_name>.list* in the *filter* field in the Application Navigator.

Use the Filter to build the query condition.



Click the **Run** button to execute the query. Right-click the breadcrumbs and select the **Copy query** menu item. Where you click in the breadcrumbs matters. The copied query includes the condition you right-clicked on and all conditions to the left. To copy the entire query, right-click the condition farthest to the right.



Return to the script and paste the encoded query into the *addEncodedQuery()* method. Be sure to enclose the encoded query in "" or ''.

```
var myObj = new GlideRecord("x_snc_needit_needit");
myObj.addEncodedQuery("u_when_neededBETWEENjavascript:gs.daysAgoStart(0)@javasc
ript:gs.quartersAgoEnd(1)^active=true^state=14^ORstate=16");
    myObj.query();
    while(myObj.next()){
        // Logic you want to execute for the GlideRecord records
    }
```