Realtime Scenario with Glide Record API

Vaishnavi Lathk
Tera Guru

on 05-22-2022
11:23 PM

What is Glide API?

Service now Developer often using **Glide API** in now platform to change default behavior of the application and customize existing functionality.

Glide Classes are providing more flexibility to interacting with snow application through scripting. Using by Glide API's we can perform some database operation instead of writing any **SQL Queries**

Each API contain lot of methods and each method perform different operations in Service now Applications

Types of Glide API's

Glide API's

Client Side	Server Side
Glide Form	Glide Record
Glide User	Glide System
Glide Ajax	Glide Date and Time
Glide Dialog Window	Glide Aggregation
Glide List	Glide Date
Glide Menu	Glide Element

What is Glide Record and usage?

This is most common and important API and frequently using in now platform

Glide Record is special Java class its native java script class and mainly running from **Server Side** .Glide Record is used to perform CRUD operation instead of writing **SQL Queries.**

This API handle both **rows** and **columns** in underlining database

In service now platform directly we can't interact with database to perform any CRUD with **SQL Queries** operations if we want to do this activities then we can go for **Glide Record**

Note: Always we need to test queries on a non-production instance before deploying them into production environment. An incorrectly constructed encoded query, such as including an invalid field name, produces an invalid query. When the invalid query is run, an **insert (), update (), deleteRecord (),** method on bad query results can result in data loss.

1. Most Common API
2. Running from server side

3. Used to generate SQL Queries
4. Perform CRUD operations

Glide Record Architecture

Instead of passing **SQL Queries,** we can use **Java Script**

Glide Record API Mapping

Glide Record Methods

Glide Record Methods	
Query()	Insert()
addQuery()	deleteRecord()
addActiveQuery()	update ()
addEncodedQuery()	initialize ()
addInactiveQuery()	deleteMultiple ()
next()	updateMultiple()
get()	addNullQuery()
setLimit()	addNotNullQuery()
orderBy()	autoSysFields()
orderByDesc()	canCreate()
getRowNumber()	canRead()
hasNext()	canWrite()
getRowCount()	canDelete()
chooseWindow()	changes()
addjoinQuery()	Find()
getClassDisplayValue()	getAttribute()
getDisplayValue()	getElement()
getLabel()	getFields()
getLink()	getValue()
getLocaton	has Attachments()
getRecordClassName()	insertWithReferences()
getRelatedLists()	isNewRecord()
getRowNumber()	isValid()
getTableName()	isValidField()
restoreLocation()	isValidRecord()
saveLocation()	newRecord()
setAbortAction()	setDisplayValue()
setLocation	setForceUpdate()
setNewGuid()	setValue()
addFunction()	setWorkflow()

_next	_query
Practically work with these all methods	

Glide Record Exercises

1. How to get **result(output)** in Servicneow

```
gs.print ('Welcome to Servicnow Academy');
gs.info ('Welcome to Servicnow Academy');
```

Result Welcome to Servicenow Academy

1. Write a simple program **add two numbers**

```
var a = 10;
var b = 20;
var c = a+b;
    gs.print (a+b);
```

Result 30

1. Working with **query()** method

```
var inc = new GlideRecord ('incident')
//GlideRecord is main Oject and Incident is Table
inc.query (); //Query is execute in the table
while (inc.next ()) {//Loop will runs on the table
gs.print (inc.number); //Printing all incidets
}
```

Result Print all records numbers in **Incident Table**

- Working with **query()** and **addQuery()** and **next()** and **While** methods

Exercise -1: Display **priority -1** tickets from incident table with **addQuery** methods

```
var inc = new GlideRecord ('incident');
inc.addQuery ('priority=1');// Add the query
inc.query ();
while(inc.next()){
gs.print(inc.number);
}
```

Result Printing all **prority-1 tickets**

1. Working with **Multiple Queries**

Exercise-2: Passing **Multiple Queries** using by same methods

```
var inc = new GlideRecord('incident');
inc.addQuery ('active', true);         //Query 1
inc.addQuery ('priority=1');          //Query 2
inc.addQuery ('category','software');   //Query 3
inc.query ();
while(inc.next()){
gs.print (inc.number);
}
```

Result Print all records where your **Condition meet**

1. Working with **addEncodedQuery ()** method

Exercise-3: we can use **addEncodedQuery** method Instead of passing multiple queries into our script

Step-1: Navigate to Incident **list view** and apply condition
Step-2: Condition: active = true and priority =1 and category = software
Step-3: Click on **Run**
Step-4: Copy applied query through **Copy query**
Step-5: Use this entire query into your script
Step-6: Script

```
var inc = new GlideRecord ('incident');
inc.addEncodedQuery('active=true^category=software^priority=1');
inc.query();
while(inc.next()){
gs.print(inc.number);
}
```

Exercise-4: Encoded Query set to a variable that variable to call into code

```
var ecq = 'active=true^category=software^priority=1';
//encodedquery set to a variable
var inc = new GlideRecord('incident');
inc.addEncodedQuery (ecq);
inc.query();
while (inc.next()){
gs.print (inc.number);
}
```

Result Print all records where this meet **'active=true^category=software^priority=1';**

Working with **addQuery** ('String','Operator','Value')

Practice with these all **SQL Operators** for better experience

- =
- !=
- >
- >=
- <
- <=

Strings (must be in upper case):

- =
- !=
- IN
- NOT IN
- STARTSWITH
- ENDSWITH
- CONTAINS
- DOES NOT CONTAIN
- INSTANCEOF

Exercise-5: Get **Active** and **Priority is less than** or **equal to 2**

```
var inc = new GlideRecord('incident');
inc.addActiveQuery();
inc.addQuery('priority','<=',2);
inc.query();
```

```
while(inc.next()){
  gs.print(inc.number);
}
```

Result Print **Critical-1** and **High-2** tickets

Exercise-7: Working with **SQL operators <=** and **CONTAINS**

```
var inc = new GlideRecord('incident');
inc.addActiveQuery();
inc.addQuery('priority','<=',2);
inc.addQuery('short_description','CONTAINS','SAP');
inc.query();
while(inc.next()){
gs.print(inc.number + ' ' + inc.short_description);
}
```

ResultPrint all records where our condition meet like **(<=2 and CONTAINS)**

Exercise-8: Working with **IN** operator and print category of **Software** and **Hardware**

```
var cat = ['software', 'hardware'];
var inc = new GlideRecord('incident');
inc.addQuery('category', 'IN', cat);
inc.query();
while(inc.next()) {
gs.print(inc.getValue('number') + ' ' + inc.getValue('short_description')) ;
 }
```

Result Print where category is **Software ad Hardware**

Exercise-9: Working with **STARTSWITH** Operator

```
var inc = new GlideRecord('incident');
inc.addQuery('category', 'STARTSWITH', 'net');
inc.query();
while(inc.next()) {
gs.print(inc.number);
 }
```

8. Working with **addActiveQuery ()** method

Exercise-10: Instead of use **active=true** this method directly we can use **addActiveQuery**

```
var inc = new GlideRecord('incident');
inc.addActiveQuery ();// instead if passing active = true
inc.addQuery ('priority',1);
```

```
inc.query ();
while (inc.next ()){
  gs.info (inc.number);
}
```

Result Print all records where condition is equal to **active** is **true** and **priority-1**

9. Working with **addInactiveQuery ()** method

Exercise-10: Instead of use **active=false** this method directly we can use **addInactiveQuery**

```
var inc = new GlideRecord ('incident');
inc.addInactiveQuery (); //Opposite of active query
inc.addQuery ('priority=1');
inc.query ();
while (inc.next ()) {
  gs.print (inc.number);
}
```

Result Print only inactive Records like Incident state is **Closed**

10. Working with **getEncodedQuery () method**

Exercise-11: getEncodedQuery from our code

```
var inc = new GlideRecord ('incident');
inc.addEncodedQuery ('active=true^category=software^priority=1');
inc.query();
while (inc.next ()) {
gs.print (inc.getEncodedQuery ());
}
```

Example: 2

Result Print our encodedQuery

11. Working with **orderBy ()** method

Exercise-12: Display all records in order wise **(Assending)** it depends on field values

```
var inc = new GlideRecord('incident');
inc.addQuery('priority=1');
inc.addQuery('category=software');
inc.orderBy('short_description');
inc.query();
while(inc.next()){
gs.print(inc.number + ' ' + inc.short_description);
}
```

Result Print all incidents order wise depends on **Short Description**

12. Working with **orderByDesc ()** method

Exercise-12: Display all records in order wise **(Descending)** it depends on field values

```
var inc = new GlideRecord('incident');
inc.addQuery('priority=1');
inc.addQuery('category=software');
inc.orderByDesc('short_description');
inc.query();
while(inc.next()){
gs.print(inc.number + ' ' + inc.short_description);
}
```

ResultPrint all records in descending order **(short_description)**

13. Working with **setLimit ()** method

Exercise-13: Display limited records from specified table

```
var inc = new GlideRecord('incident');
inc.addQuery('priority=1');
inc.orderByDesc('short_description');
inc.setLimit(10);
inc.query();
while(inc.next()){
gs.print(inc.number + ' ' + inc.short_description);
}
```

Result Print only latest **10 records** created from given table

14. Working with **get ()** Method

Exercise-14: Get record **sys_id** depends on **INC number** or Get incident record number depends on **sys_id**

```
var inc = new GlideRecord('incident');
inc.get('number','INC0009005');
gs.print(inc.sys_id);
```

Result Print sys_id related to **incident number**

Example-2

Result Print Incident number related to **sys_id**

15. **15**. Working with **chooseWindow ()** method

Exercise-15: Display records between two numbers

```
var inc = new GlideRecord('incident');
```

```
inc.addQuery('priority=1');
inc.addActiveQuery();
inc.chooseWindow(3,7)//include first value exclued secod value
inc.query()
while(inc.next()){
  gs.print(inc.number);;
}
```

Result Print records from number 3 to 7 (4 records) **first number included** and **second number excluded**

16. Working with **getRowCount ()** method

Exercise-16: Display all records from particular table (Incident)

```
 var inc = new GlideRecord('incident');
 inc.query()
 gs.print(inc.getRowCount());
```

Result Print number of records in particular table

Example-2: Display all active users in our **sys_user** tables

```
var inc = new GlideRecord('sys_user');
inc.addQuery('active=true');
inc.query();
gs.print ('Active users are:' + inc.getRowCount ());
```

Result Print number of record in table

17. **Working getTableName () method**

Exercise-17: This method is used to get glide record table name

```
var inc = new GlideRecord ('change_request');
gs.print (inc.getTableName ());
```

Result Display current table name from glide record

18. **Working getValue () method**

Exercise-18: Get value of particular field in the table

```
var inc = new GlideRecord('incident');
inc.addQuery('active=true');
inc.query();
while(inc.next()){
  gs.print(inc.getValue('short_description'));
}
```

Result Print the value of field from particular table

19. **Working getDisplayValue () method**

20. Print display value instead of actual value

```
var inc = new GlideRecord('incident');
inc.addQuery ('priority=1')
inc.query ();
while (inc.next ()){
gs.print (inc.priority.getDisplayValue ());
}
```

Result Print display value of respective field

20. **Working hasNext () method**

Exercise-19: This method will returns true if iterator have more elements.

```
var inc = new GlideRecord ('incident');
inc.query ();
gs.print (inc.hasNext ());
```

Result Print Boolean value **(True)**

21. Working with getUniqueValue () method

Exercise-20: Gets the uniue key of the record, which is usually the sys_id unless otherwise specified.

```
var inc = new GlideRecord('incident');
inc.query();
inc.next();
var uniqvalue = inc.getUniqueValue();
gs.print(uniqvalue);
```

22. **22**. Working with **setValue ()** method

Exercise-22: This method is used to sets the value of the specific field with the specified value.

```
var attriName = 'category';
var inc = new GlideRecord ('incident');
inc.initialize ();
inc.setValue(attriName,'network');
inc.setValue('short_description','Critical VPN Issue');
inc.insert();
gs.print ('Category is ' + inc.category + ' and ' + 'issue is: ' + inc.short_description);
```

Result Create a new record and **Set** a value into **category** field

23. **23**. Working with **getElement ()** method

Exercise-23: This is used to get the specified column of the current record.

```
var elementName = 'short_description'
var inc = new GlideRecord('incident');
```

```
inc.initialize();
inc.setValue (elementName,'I am facing VPN Problem');
inc.insert ();
gs.print(inc.getElement('short_description'));
```

Result Print current record column value

24. **24**. Working with **getRecordClassName ()** method

Exercise-24: Retrieves the class name for the current record.

```
var inc = new GlideRecord('change_request');
var grcn = inc.getRecordClassName ();
gs.info (grcn);
```

Result Print record class name (Table Name)

25. Working with **initialize ()** and **insert ()** method

Exercise-25: These methods are used to **Inserts a new record** using the field values that have been set for the current record

```
var inc = new GlideRecord ('incident');
inc.initialize (); //Compose incident form
inc.category = 'network'; // set field values
inc.short_description = 'Firewall Issue';
inc.priorty = 1;
inc.insert (); // create new record
gs.print (inc.number);// print new record incident number
```

Result Create new record and print new record number

26. Working with **isNewRecord ()** and **newRecord ()** method

Exercise-26: Checks if the current record is a new record that has not yet been inserted into the database.

```
var inc = new GlideRecord ('incident');
inc.newRecord ();
gs.info (inc.isNewRecord());
```

Result Return boolion value true or false **(value is True)**

27. Working with **isValid ()** method

Exercise-27: Define the current table exist or not. If table exist display **true** not exist display **false**

```
var inc = new GlideRecord ('incident');
gs.print (inc.isValid ());
```

Result: True

Example: 2

```
var inc = new GlideRecord ('srinivas');
gs.print (inc.isValid ());
```

Result: False

28. Working with **isValidField ()** method

Exercise-27: Determines if the specified field is defined in the current table. If Field exist in current record display true not exist display false

```
var inc = new GlideRecord ('incident');
gs.print (inc.isValidField (‘category’);
```

Result: Boolion value is **True**

29. Working with **getLink()** and **getProperty()** method

Exercise-29: Retrieves a **link** to the current record.

```
var inc = new GlideRecord('incident');
inc.addActiveQuery();
inc.addQuery('category','software');
inc.addQuery('priority=1');
inc.query();
while(inc.next()){
gs.print(gs.getProperty('glide.servlet.uri') + inc.getLink(false));
}
```

Result: Return the link of record

Example: 2 Return first record **Link** from query

29. Working with **isValidRecord ()** method

Exercise-29: Determines if a record was actually returned by the **query/get** record operation.

```
var inc = new GlideRecord ('incident');
inc.get ('number','INC0010012 ');
gs.print (inc.number + ' exists:' + inc.isValidRecord ());
```

Result Display boolion value either true or false, **True**

30. Working with **newRecord ()** method

Exercise-30: Creates a new GlideRecord record, sets the default values for the fields, and assigns a unique ID to the record.

```
var inc = new GlideRecord ('incident');
inc.newRecord ();
inc.short_description = 'Creting new record';
inc.category = 'software';
inc.insert ();
gs.print (inc.number);
```

Result Create new record and print

30. Working with **addNullQuery ()** method

Exercise-30: display all records where the value of the specified field is null.

```
var inc = new GlideRecord('incident');
inc.addNullQuery ('short_description')
inc.query ();
while (inc.next ()) {
gs.print (inc.number)
}
```

Result Print all records where the specific field value is **Null**

31. Working with **addNotNullQuery ()** method

Exercise-30: Opposite of addNullQuery methods display all records where the value of the specified field is **not null.**

```
var inc = new GlideRecord('incident');
inc.addNotNullQuery ('short_description')
inc.query ();
while (inc.next ()) {
gs.print (inc.number)
}
```

Result Print all records where the specific field value is **not null**

32. Working with **update ()** method single record

Exercise-32: Update specific record from table

```
var inc = new GlideRecord ('incident');
inc.get ('number','INC0000057');
inc.setValue ('state', 2);
inc.update ();
```

Result update record as expected

33. Working with **updateMultiple ()** method multiple record

Exercise-33: Updates multiple records in a stated query with a specified set of changes from respected table**.**

```
var inc = new GlideRecord('incident');
inc.addQuery ('category', 'hardware');
inc.setValue('category',  'software');
inc.updateMultiple ();
```

Result Update multiple records as expected
Exercise: 2
34. Working with **deleteRecord ()** method single record
Exercise-33: This method is used to delete single record from table

```
var inc = new GlideRecord('incident');
inc.get ('number','INC0010013');// need to delete this record
inc.deleteRecord ();
```

Result Delete single record as expected

35. Working with **deleteMultiple ()** method multiple record

Exercise-35: Deletes multiple records that satisfy the query condition.

```
var inc = new GlideRecord('incident');
inc.addQuery('priority', 4);
```

inc.query ();
inc.deleteMultiple ();
Result Delete multiple records as expected

36. Working with **canCreate ()** method
Exercise-36: Determines if the Access Control Rules, which include the user's roles, permit create new records in this table.

var inc = new GlideRecord ('incident');
gs.print (inc.canCreate ());

Result **True** (user have permission to create incident record)

37. Working with **canRead ()** method
Exercise-36: Determines if the Access Control Rules, which include the user's roles, permit reading records in this table.

var inc = new GlideRecord('incident');
gs.print (inc.canRead ());

Result **True** (user have permission to read incident record)

38. Working with **canWrite ()** method
Exercise-37: Determines if the Access Control Rules, which include the user's roles, permit editing records in this table.
var inc = new GlideRecord ('incident');
gs.print (inc.canWrite ());

Result **True** (user have permission to write incident record)

39. Working with **canDelete ()** method

Exercise-38: Determines if the Access Control Rules, which include the user's roles, permit deleting records in this table.
var inc = new GlideRecord ('incident');
gs.print (inc.canDelete ());

Result **True** (user have permission to delete incident record)

40. Working with **autoSysFields ()** and **setWorkflow ()** methods

Enables or disables the update to the fields, this is often used for manually updating field values on a record while leaving historical information unchanged.

sys_updated_by
sys_updated
sys_updated_on

```
sys_mod_count
sys_created_by
sys_created_on
```

Note: autoSysFields method is not working on scoped application.

Note: setWorkflow (false) not run any other business rule

Exercise-40: Update multiple records without update any system fields

```
var inc = new GlideRecord ('incident');
inc.addQuery ('state', 1);
inc.query ();
while (inc.next ()) {
inc.autoSysFields (false);
inc.setWorkflow (false);
inc.setValue ('state', 2);
inc.update ();
}
```

Result Updating records without update system fields

41. Working with **addJoinQuery ()** methods

Exercise-40: Find problems that have an incident attached. This example returns problems that have associated incidents.

```
var prob = new GlideRecord('problem');
prob.addJoinQuery('incident', 'opened_by', 'caller_id');
prob.query();
while(prob.next()){
 gs.print(prob.number);
}
```

Result Display all problem records associated incident

42. Working with **getGlideObject ()** and **getNumericValue () and setAbortAction** methods

Exercise-41: This is method is used to cancel current action when condition is false

```
if ((!current.u_date1.nil()) && (!current.u_date2.nil())) {
var start = current.u_date1.getGlideObject().getNumericValue();
var end = current.u_date2.getGlideObject().getNumericValue();
if (start > end) {
gs.addInfoMessage('start must be before end');
current.u_date1.setError('start must be before end');
```

```
current.setAbortAction(true);
}
}
```

Mark the article as helpful if you found this useful

Regards,
Vaishnavi

San Diego | Script Debugger | Scripting and Coding

11 Helpfuls

Comments

taylor13
Giga Guru

05-24-2022 04:36 AM

Great Stuff! Thank you so much for sharing

0 Helpfuls

Vaishnavi Lathk
Tera Guru

05-24-2022 04:59 AM

Happy to help you 🙂

2 Helpfuls