Scripting Guide

Scripting for ServiceNow

Glide

Glide Stack

Note: *This article applies to Fuji. For more current information, see Glide Stack* [1] *at* http://docs.servicenow.com The ServiceNow Wiki is no longer being updated. Please refer to http://docs.servicenow.com for the latest product documentation.

Overview

Glide is an extensible Web 2.0 development platform written in Java that facilitates rapid development of forms-based workflow applications (work orders, trouble ticketing, and project management, for example).

User Interface Stack Technology Map

Java Packages		Technologies Used
	User Interface (Browser)	DHTML CSS JavaScript
com.glide.ui com.glide.jelly	GlideServlet	Apache Jelly
com.glide.script	Business Rules	Mozilla Rhino
com.glide.db	Persistence	JDBC

GlideServlet

The primary driver of Glide, and the only servlet in the system, is found in GlideServlet.java. The GlideServlet:

- Handles inbound action requests
- Renders pages
- Merges data with forms
- Presents to user
- Interfaces with script layer

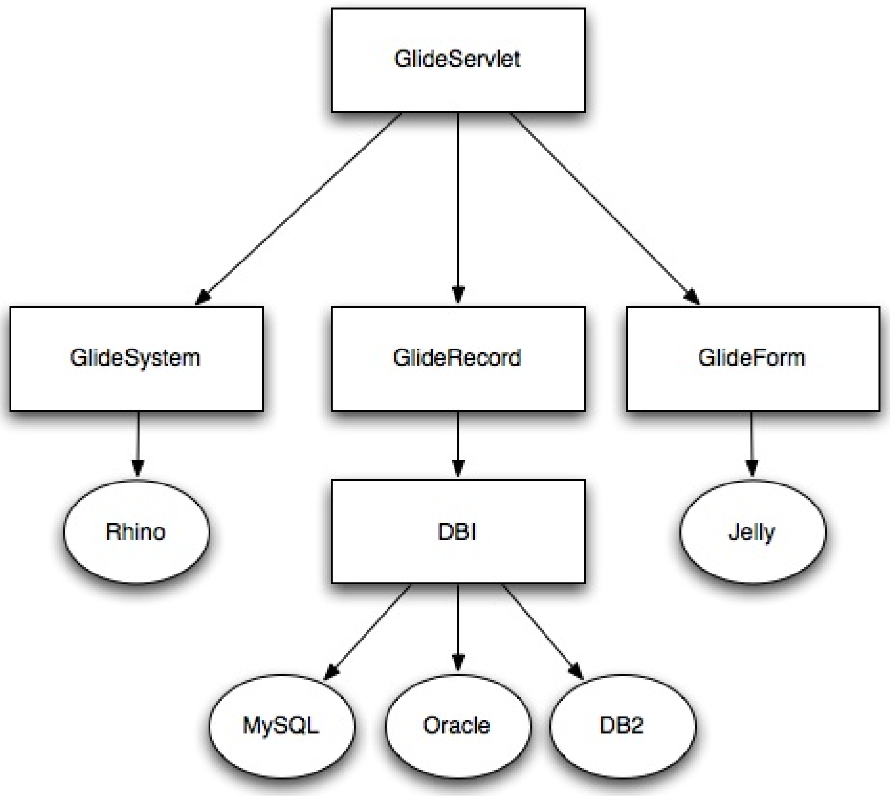
Business Rules

- ECMA / JavaScript implementation based on Mozilla Rhino [2]
- Interfaces with persistence layer using JDBC recordset interface
- Merges persistence layer meta-data with data for easy correlation

Persistence

- Persistence means any store
  - RDBMS
  - LDAP
  - File system
- Uniform access regardless of store type
- Provides QUID and meta-data capabilities
- Interfaces presented to callers
  - RecordSet
  - TableDescriptor
  - ElementDescriptor

Diagram



References

[1] https://docs.servicenow.com/bundle/jakarta-servicenow-platform/page/script/general-scripting/reference/r_GlideStack.html
[2] http://www.mozilla.org/rhino/

Glide Script Objects

❗ **Note:** *This article applies to Fuji and earlier releases. For more current information, see Glide Class Overview* [1] *at* http://docs.servicenow.com **The ServiceNow Wiki is no longer being updated. Visit** http://docs.servicenow.com **for the latest product documentation.**'

Overview

There are several important objects that are exposed when writing business rules. These objects are created when your script is called. You may access objects using a shorthand notation. These include:

- gs - GlideSystem most of the utility functions may be found in this object.
- system - Same as 'gs'
- current - Current GlideRecord (read/write) only available when writing business rules.
- previous - Previous GlideRecord values (read only): only available when writing business rules.

Some of the more important classes are documented along with the Glide Script Objects. They can be created and used within a script or business rule or returned by one or more of the Script Object methods.

Glide provides full scripting support when writing business rules. Business rules are written in Javascript and have access to the Glide scripting objects. The majority of Glide functionality is accessed using JavaScript.

References

[1] https://docs.servicenow.com/bundle/jakarta-servicenow-platform/page/script/general-scripting/reference/r_GlideClassOverview.html

Using GlideRecord to Query Tables

Note: *This article applies to Fuji and earlier releases. For more current information, see Using GlideRecord to Query Tables* [1] *at* http://docs.servicenow.com **The ServiceNow Wiki is no longer being updated. Visit** http://docs.servicenow.com **for the latest product documentation.**

Overview

In order to query a table, first create a ServiceNow object for the table. This object is called a GlideRecord. To create a GlideRecord, create the following in script:

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

This creates a variable called **target** which is a GlideRecord object for the **incident** table. The only parameter needed is the name of the table to be accessed.
To process *all* records from the incident table, add the following script:

```
target.query();   // Issue the query to the database to get all records
while (target.next()) {
  // add code here to process the incident record
}
```

This issues the **query()** to the database. Each call to **next()** would load the next record which you would process and do whatever you want to do.

But that is not the common case. Most of the time you actually want to retrieve a specific record or a specific set of records, and you have some criteria (query conditions) that define the records you want to obtain. For example, say you want to obtain all the incident records that have a priority value of 1. Here is the code that would accomplish that.

```
  var target = new GlideRecord('incident');
  target.addQuery('priority', 1);
  target.query();   // Issue the query to the database to get relevant
records
  while (target.next()) {
     // add code here to process the incident record
  }
```

Notice in the above code we added the line **target.addQuery('priority', 1);**. This is indicating that you only want the records where the **priority** field is equal to *1*. We assume that the majority of queries that you will want to do will be equality queries, queries where you want to find records where a field is equal to a value. Therefore we provide this format of the query and do not make you specify that you want an equals operation, we just assume it. However, lets say you wanted to find all incidents where the priority field is GREATER THAN 1. In that case you have to provide us with the operator that you want to apply to priority and this is done by providing the operator in the addQuery() request as is shown below.

```
   var target = new GlideRecord('incident');
   target.addQuery('priority', '>', 1);
   target.query();   // Issue the query to the database to get relevant
 records
```

```
   while (target.next()) {
     // add code here to process the incident record
   }
```

Available JavaScript Operators

So that leads to the question of what type of operators are possible. The table below contains the operators that can be supplied to the addQuery() request.

=	Field must be equal to value supplied.	addQuery('priority', '=', 1);
>	Field must be greater than value supplied	addQuery('priority', '>', 1);
<	Field must be less than value supplied.	addQuery('priority', '<', 3);
>=	Field must be equal or greater than value supplied	addQuery('priority', '>=', 1);
<=	Field must be equal or less than value supplied.	addQuery('priority', '<=', 3);
!=	Field must not equal the value supplied.	addQuery('priority', '!=', 1);
STARTSWITH	Field must start with the value supplied. The example shown on the right will get all records where the short_description field starts with the text **Error**.	addQuery('short_description', 'STARTSWITH', 'Error');
CONTAINS	Field must contain the value supplied somewhere in the text. The example shown on the right will get all records where the short_description field contains the text **Error** anywhere in the field.	addQuery('short_description', 'CONTAINS', 'Error');
IN	Takes a map of values that allows commas, and gathers a collection of records that meet some other requirement. Behaves as Select * from <table> where short_description IN ('Error'), which is identical to Select * from <table> where short_description='Error'. For example, to query all variable values that belong to a specific Activity, use the IN clause to query all Activities that are of that type, and store their sys_ids in a map, or comma-separated list. Then query the variable value table and supply this list of sys_ids.	addQuery('short_description', 'IN', 'Error,Success,Failure');
ENDSWITH	Field must terminate with the value supplied. The example shown on the right will get all records where the short_description field ends with text **Error**.	addQuery('short_description', 'ENDSWITH', 'Error');
DOES NOT CONTAIN	Selects records that do NOT match the pattern in the field. This operator does not retrieve empty fields. For empty values, use the operators "is empty" or "is not empty". The example shown on the right will get all records where the short_description field does not have the word **Error**.	addQuery('short_description', 'DOES NOT CONTAIN', 'Error');
NOT IN	Takes a map of values that allows commas, and gathers a collection of records that meet some other requirement. Behaves as: Select * from <table> where short_description NOT IN ('Error').	addQuery('short_description', 'NOT IN', 'Error,Success,Failure');
INSTANCEOF	Special operator that allows you to retrieve only records of a specified "class" for tables which are extended. For example when going after configuration items (cmdb_ci table) you many want to retrieve all configuration items that are have are classified as computers. The code to the right will do that.	addQuery('sys_class_name', 'INSTANCEOF', 'cmdb_ci_computer');

There are also some special methods that can be used when you want to search for data that is NULL or NOT NULL. To search for all incidents where the short_description field has *not* been supplied (is null), use the following query:

```
  var target = new GlideRecord('incident');
  target.addNullQuery('short_description');
  target.query();   // Issue the query to the database to get all
records
  while (target.next()) {
     // add code here to process the incident record
  }
```

To find all incidents in which a short_description has been supplied, use the following query:

```
  var target = new GlideRecord('incident');
  target.addNotNullQuery('short_description');
  target.query();   // Issue the query to the database to get all
records
  while (target.next()) {
     // add code here to process the incident record
  }
```

GlideRecord Query Examples

To view additional examples as well as additional query related methods, refer to GlideRecord.

query

```
var rec = new GlideRecord('incident');
rec.query();
while (rec.next()) {
  gs.print(rec.number + ' exists');
}
```

update

```
var rec = new GlideRecord('incident');
rec.addQuery('active',true);
rec.query();
while (rec.next()) {
  rec.active = false;
  gs.print('Active incident ' + rec.number = ' closed');
  rec.update();
}
```

insert

```
var rec = new GlideRecord('incident');
rec.initialize();
rec.short_description = 'Network problem';
rec.caller_id.setDisplayValue('Joe Employee');
rec.insert();
```

delete

```
var rec = new GlideRecord('incident');
rec.addQuery('active',false);
rec.query();
while (rec.next()) {
  gs.print('Inactive incident ' + rec.number + ' deleted');
  rec.deleteRecord();
}
```

OR query

For an example of how to do an OR query see scripting an OR condition.

Querying Service Catalog Tables

You cannot directly query the variables of the Service Catalog Request Item table [sc_req_item]. Instead, query the Variable Ownership table [sc_item_option_mtom] by adding two queries, one for the variable name and another for the value. The query returns the many-to-many relationship, which you can dot-walk to the requested item. The following example finds the request items that have the variable named 'item_name' with a value of 'item_value' and displays the request item numbers:

```
var gr = new GlideRecord('sc_item_option_mtom');
gr.addQuery('sc_item_option.item_option_new.name','item_name');
gr.addQuery('sc_item_option.value','item_value');
gr.query();

while (gr.next()) {
    gs.addInfoMessage(gr.request_item.number);
}
```

References

[1] https://docs.servicenow.com/bundle/jakarta-servicenow-platform/page/script/server-scripting/concept/c_UsingGlideRecordToQueryTables.html

Getting a User Object

Note: *This article applies to Fuji and earlier releases. For more current information, see Get a User Object* [1] *at* http://docs.servicenow.com **The ServiceNow Wiki is no longer being updated. Visit** http://docs.servicenow.com **for the latest product documentation.**'

Note: This functionality requires a knowledge of **Javascript**.

Overview

In a business rule or other server javascript, gs.getUser() returns a user object.

The user object is ServiceNow's internal representation of the currently logged in user. It has a lot of information about the user, and a variety of utility functions that make JavaScripting easier.

Getting a User Object

```
var myUserObject = gs.getUser()
```

To get user information for a particular user, first retrieve the current user, and then use that object's getUserByID method to fetch a different user using the user_name field or sys_id on the target record. For example:

```
var ourUser = gs.getUser();
gs.print(ourUser.getFirstName()); //should print out the first name
of the user you are currently logged in as
ourUser = ourUser.getUserByID('abel.tuter'); //fetched a different
user, using the user_name field or sys_id on the target user record.
gs.print(ourUser.getFirstName()); //this should be the first name of
 the user you fetched above
gs.print(ourUser.isMemberOf('Capacity Mgmt'));
```

Descriptive Functions

The following functions take no arguments:

```
myUserObject.getFirstName() -- returns first name of current user
myUserObject.getLastName() -- returns the last name of the current user
myUserObject.getFullName() -- returns the current user's full name
myUserObject.getDisplayName() -- returns the current user's display name, typically the same as full name
myUserObject.getEmail() -- returns the email address of the current user
myUserObject.getMobileNumber() — returns the mobile number of the current user
myUserObject.getDepartmentID() -- returns the sys_id of the current user's department
myUserObject.getCompanyID() -- returns the sys_id of the current user's company
myUserObject.getCompanyRecord() -- returns the current user's company GlideRecord
myUserObject.getLanguage() -- returns the current user's language
myUserObject.getLocation() -- returns the current user's location
myUserObject.getCountry() -- returns the current user’s country
myUserObject.getManagerName() -- returns the user_name of the current user’s manager
myUserObject.getManagerID() -- returns the sys_id of the current user's manager
myUserObject.getDomainID() -- returns the domain ID of the current user
myUserObject.getDomainDisplayValue() -- returns the domain display value for the current user
myUserObject.getTZ() -- returns the timezone of the current user
myUserObject.getUserRoles() -- returns the roles explicitly granted to the current user
```

The following function is valid for instances using the Simple Security Manager. Instances using the Contextual Security Manager must use gs.getSession().getRoles().

```
myUserObject.getRoles() -- returns all of the roles of the current user
```

Security and Groups

`myUserObject.getMyGroups()`	-- an iterator that returns a list of all groups to which the current user belongs (takes no argument). For an example of how to consume an iterator, see Examples of Extracting the Contents of an Iterator into an Array. `getMyGroups()` only returns groups that are active.
`myUserObject.isMemberOf()`	-- returns true or false if the current user is a member of the provided group (takes a group name or sys_id as its argument). `isMemberOf()` returns false if the group is inactive, even if the user is a member.
`myUserObject.hasRole()`	-- returns true or false if the current user has the provided role (takes a role name as its argument)

Method Detail

getUserByID	`public User getUserByID(java.lang.String id)`	Get a user by the User ID or sys_id in sys_user. **Parameters:** id - User ID or sys_id to find **Returns:** Reference to the User object associated with the specified User ID or sys_id. This is not a GlideRecord.
getCompanyID	`public User getCompanyID()`	Get the company associated with a User object. **Parameters:** None **Returns:** sys_id of the company record associated with the User object. **Example:** `current.company = gs.getUser().getCompanyID()`

Examples of Extracting the Contents of an Iterator into an Array

Example 1

```
var groupsArray = gs.getUser().getMyGroups().toArray();
```

Example 2

```
function returnCurrentUserGroup(){
    var myUserObject = gs.getUser();
    var myUserGroups = myUserObject.getMyGroups();
    var groupsArray = new Array();
    var it = myUserGroups.iterator();
    var i=0;
    while(it.hasNext()){
        var myGroup = it.next();
        groupsArray[i]=myGroup;
        i++;
    }
    return groupsArray;
}

var test = returnCurrentUserGroup();
gs.print(test[0]);
```

References

[1] https://docs.servicenow.com/bundle/jakarta-servicenow-platform/page/script/server-scripting/task/t_GetAUserObject.html

Scripting Alert, Info, and Error Messages

Note: *This article applies to Fuji and earlier releases. For more current information, see Scripting Alert, Info, and Error Messages* [1] *at* http://docs.servicenow.com **The ServiceNow Wiki is no longer being updated. Visit** http://docs.servicenow.com **for the latest product documentation.**

Overview

There are several ways to send alert messages to customers.

Business rule and other general use

- current.*field_name*.setError("*Hello World*"); Will put "Hello World" below the specified field
- gs.addInfoMessage("*Hello World*"); Will put "Hello World" on the top of the screen
- gs.print("*Hello World*"); Will write to the text log on the file system but not to the sys_log table in the database
- gs.log("*Hello World*"); Will write to the database and the log file. Too much of this can adversely affect performance

Client side (NOTE: only for use with client scripting)

- alert(*"Hello World"*); Will pop up a window with "Hello World" and an 'OK' button.
- confirm(*"Hello World"*); Will pop up a window with "Hello World?" and a 'Ok' and 'Cancel' buttons.
- g_form.showErrorBox(*"field_name"*, *"Hello World"*); Will put "Hello World" in an error message below the specified field.
- g_form.hideErrorBox(*"field_name"*); Will hide an error box that is visible under the specified field.

It's also possible to add other custom messages to your forms if necessary using client scripting.

Text size of error and alert messages

The text size of info and error messages at the top of the screen is customizable. Two properties control this. If you configured your forms, you may need to add this property.

- The **css.outputmsg.info.text.font-size** property sets the size for info messages. Default is 11pt
- The **css.outputmsg.error.text.font-size** property sets the size for error messages. Default is 11pt

References

[1] https://docs.servicenow.com/bundle/jakarta-servicenow-platform/page/script/general-scripting/reference/r_ScriptingAlertInfoAndErrorMsgs.html

Display Field Messages

Note: *This article applies to Fuji. For more current information, see Display Field Messages* [1] *at* http://docs.servicenow.com
The ServiceNow Wiki is no longer being updated. Please refer to http://docs.servicenow.com for the latest product documentation.

Overview

Rather than use *JavaScript alert()*, for a cleaner look, you can display an error on the form itself. The methods *showFieldMsg()* and *hideFieldMsg()* can be used to display a message just below the field itself.

showFieldMsg and hideFieldMsg are methods that can be used with the g_form object (see GlideForm (g_form)).

These methods are used to make changes to the form view of records (Incident, Problem, and Change forms). These methods may also be available in other client scripts, but must be tested to determine whether they will work as expected.

When a field message is displayed on a form on load, the form will scroll to ensure that the field message is visible, ensuring that users will not miss a field message because it was off the screen.

A global property (glide.ui.scroll_to_message_field) is now available that controls automatic message scrolling when the form field is offscreen (scrolls the form to the control or field).

Method

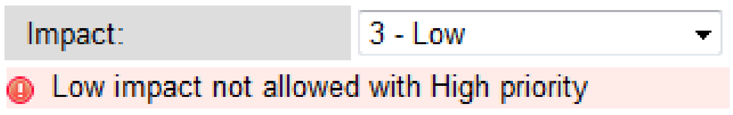
showFieldMsg(input, message, type, [scrollform]);
input - name of the field or control
message - message you would like to display
type - either 'info' or 'error', defaults to *info* if not supplied
scroll form - (optional) Set scrollForm to false to prevent scrolling to the field message offscreen.

hideFieldMsg(input);
input - name of the field or control **clearAll* - (optional) boolean parameter indicating whether to clear all messages. If true, all messages for the field will be cleared; if false or empty, only the last message will be removed.

Example

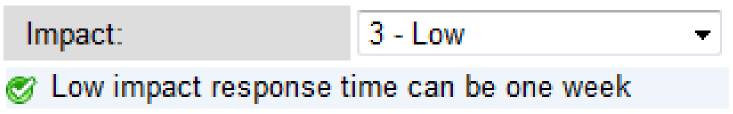
Error Message

```
g_form.showFieldMsg('impact','Low impact not allowed with High
priority','error');
```


Impact: 3 - Low
Low impact not allowed with High priority

Informational Message

```
g_form.showFieldMsg('impact','Low impact response time can be one
week','info');
//or this defaults to info type
//g_form.showFieldMsg('impact','Low impact response time can be one
week');
```


Impact: 3 - Low

Low impact response time can be one week

Removing a Message

```
//this will clear the last message printed to the field
g_form.hideFieldMsg('impact');
```

Legacy Support

The showErrorBox() and hideErrorBox() are still available but simply call the new methods with type of error. It is recommended that you use the new methods.

References

[1] https://docs.servicenow.com/bundle/jakarta-servicenow-platform/page/script/useful-scripts/reference/r_DisplayFieldMessages.html

Setting a GlideRecord Variable to Null

Note: *This article applies to Fuji and earlier releases. For more current information, see Setting a GlideRecord Variable to Null* [1] *at* http://docs.servicenow.com **The ServiceNow Wiki is no longer being updated. Visit** http://docs.servicenow.com **for the latest product documentation.**

Note: This functionality requires a knowledge of **Javascript**.

Functionality described here requires the **Admin** role.

Overview

GlideRecord variables (including current) are initially NULL in the database. Setting these back to an empty string, a space, or the javascript null value will not result in a return to this initial state. To set it back to the initial state, simply set the value to "NULL". Note that the update() function does not run on the current object but rather on the record. The object displays the initial value until it is called again from the record.

Example 1

```
var gr1 = new GlideRecord('incident');
gr1.query();
while(gr1.next()) {
  gr1.priority = "NULL";
  gr1.update();
}
```

Example 2 (Business Rule)

```
current.u_affected_value = 'NULL';
current.update();
```

References

[1] https://docs.servicenow.com/bundle/jakarta-servicenow-platform/page/script/server-scripting/reference/r_SettingAGlideRecordVariableToNull.html

Referencing a Glide List from a Script

Note: *This article applies to Fuji and earlier releases. For more current information, see Reference a Glide List from a Business Rule* [1] *at* http://docs.servicenow.com **The ServiceNow Wiki is no longer being updated. Visit** http://docs.servicenow.com **for the latest product documentation.**

Overview

A field defined as a glide_list is essentially an array of values that are stored within a single field. Here are some examples of how to process a glide_list field when writing business rules. Generally a glide_list field contains a list of reference values to other tables.

Examples

For example, the **Watch list** field within tasks is a glide_list containing references to user records.

The code below shows how to reference the field.

```
  // list will contain a series of reference (sys_id) values separated
by a comma
  // array will be a javascript array of reference values
  var list = current.watch_list.toString();
  var array = list.split(",");
  for (var i=0; i < array.length; i++) {
     gs.print("Reference value is: " + array[i]);
  }
```

You can also get the display values associated with the reference values by using the getDisplayValue() method as shown below

```
  // list will contain a series of display values separated by a comma
  // array will be a javascript array of display values
  var list = current.watch_list.getDisplayValue();
  var array = list.split(",");
  for (var i=0; i < array.length; i++) {
     gs.print("Display value is: " + array[i]);
  }
```

Note: *For a script example of how to edit a glide list, see Add All Email Recipients to Watch List.*

Using indexOf("searchString")

Use `indexOf("searchString")` to return the location of the string passed into the method if the glide list field, such as a Watch list, has at least one value in it. If the field is empty, it returns **undefined**. To avoid returning an undefined value, do any of the following:

- Force the field to a string, such as: `watch_list.toString().indexOf("searchString")`
- Check for an empty Glide list field with a condition before using `indexOf()`, such as: `if (watch_list.nil() || watch_list.indexOf("searchString") == -1)`

References

[1] https://docs.servicenow.com/bundle/jakarta-servicenow-platform/page/script/business-rules/reference/r_ReferencingAGlideListFromAScript.html

Jelly

Extensions to Jelly Syntax

Note: *This article applies to Fuji and earlier releases. For more current information, see Extensions to Jelly Syntax* [1] *at* http://docs.servicenow.com **The ServiceNow Wiki is no longer being updated. Visit** http://docs.servicenow.com **for the latest product documentation.**'

Overview

Apache's Jelly syntax is used to render forms, lists, UI pages, and many other things rendered in ServiceNow. With Jelly, logic can be embedded within static content and computed values may be inserted into the static content.

Note: *This functionality requires a knowledge of Apache Jelly (a Java and XML based scripting and processing engine for turning XML into executable code).*

Namespaces

In ServiceNow, Jelly often includes multiple namespaces when invoking tags. The “j” namespaces are standard Jelly whereas the “g” namespaces are unique to ServiceNow. For example, the <g:evaluate> tag is supplied by ServiceNow to allow you to compute a value using JavaScript. The standard Jelly tag <j:test> is used to evaluate a condition.

Phases

Usually, there are two phases indicated by namespaces <j> versus <j2> and <g> versus <g2>. The namespaces without the “2” happen in the first phase of processing and these are cached except when used in a UI page. Those with the “2” are never cached. Care must be taken when selecting whether to use phase 1 or phase 2 for efficiency and correct results.

In addition to the namespaces, the syntax used to insert values into static content differs depending on which phase is to supply the value. A dollar with braces surrounding a value inserts the value in phase 1. For example, `${jvar_ref}` inserts the value **jvar_ref** during phase 1 of the jelly process. A dollar with brackets surrounding a value inserts the value in phase 2. For example, `$[jvar_ref]` inserts the value **jvar_ref** during phase 2. A value surrounded by quotes is treated as a string. For example, `'[jvar_ref]'` inserts the value **jvar_ref** as a string during phase 2.

```
<script>
if (confirm("$[gs.getMessage('home.delete.confirm') ]"))
   ...
</script>
```

```
<input type="hidden" id="${jvar_name}" name="${jvar_name}" value="${jvar_value}" class="${jvar_class}" />
```

If Tests

Testing whether something is true or not can be done as follows:

```
<j:if test="${jvar_something}">...do something...</j:if>
<j:if test="${!jvar_something}">...do something...</j:if>
```

The reason this works, is that, in Jelly, a term like jvar_something is "truthful" in an if tag if:

1. it is Boolean and true
2. it is a String and = "true", "yes", "on", or "1"

Testing whether something exists can be done as follows:

```
<j:if test="${empty(jvar_something)}">...do something...</j:if>
```

The reason this works is that the jexl empty function returns true if its argument is:

1. null
2. an emptry string
3. a zero length collection
4. a map with no keys
5. an empty array

Set_If

Sets a variable to one of two different values depending on whether a test is true or false.

```
<g2:set_if var="jvar_style" test="$[gs.getPreference('table.compact') != 'false']"
   true="margin-top:0px; margin-bottom:0px;"
   false="margin-top:2px; margin-bottom:2px;" />
```

<g:insert> Versus <g:inline> Versus <g:call>

The <g:insert> tag inserts a Jelly file into your Jelly in a new context. This means you cannot access the variables previously established in your Jelly. The <g:inline> tag inserts a Jelly file into your Jelly in the same context. This means that the inserted Jelly can access the variables you previously established and it can change the values of those variables.

```
<g:insert template="get_target_form_function.xml" />
```

```
<g:inline template="element_default.xml" />
```

<g:call>

For better encapsulation, the <g:call> tag may be used. Your function will only have access to the values passed to it. The Jelly context will look the same after a call as before the call. This means you cannot set a global variable here and read it later. This also means you can't mistakenly set a global variable called "jvar_temp" and overwrite a variable that somebody else was relying on.

Passing values, if needed, is done explicitly by including the name of the parameter on the <g:call> line followed by the equal sign followed by the value in quotes:

```

<g:call function="collapsing_image.xml" id="${jvar_section_id}" image="$[jvar_cimg]"
   first_section_id="${jvar_first_section_id}"
```

```
image_alt="${jvar_cimg_alt}"/>
```

If values are passed, and you want to have defaults or required parameters, your Jelly referenced in the function must then include a line to declare whether the parameters are required or have a default value:

```
<g:function id="REQUIRED" image="REQUIRED" image_prefix="" image_alt="REQUIRED"/>
```

The example above indicates that 3 of the parameter are required and one parameter is option with a blank default value. Note that if you are not passing values or if you do want to have default or required values, you do not need to include the <g:function> line at all. In general, however, you will want to include a <g:function> line.

The value can then be referenced in your template by prepending the "jvar_" prefix to the parameter's name:

```
<img id="img.${jvar_id}" src="images/${jvar_image}" alt="${jvar_image_alt}"
   onclick="toggleSectionDisplay('${jvar_id}',
'${jvar_image_prefix}','${jvar_first_section_id}');"/>
```

For <g:call>, parameters may also be pass implicitly as a list of named variables in an "arguments" parameter:

```
  <g:call function="item_link_default.xml" arguments="sysparm_view,ref_parent,jvar_target_text"/>
```

As an alternative to passing variables into the function via separate tag arguments, it is possible to pass a list of variables in a single 'arguments' argument. All variables identified by name (comma separated) in the *argument* parameter are re-introduced within the function under the exact same name (e.g. inside the function template, we'd have variables sysparm_view, ref_parent, and jvar_target_text available to us).

The function template may return a value to the calling template using the **return=** attribute. Within the function the **jvar_answer** variable sets the return value.

```
  <g:call function="item_body_cell_calc_style.xml" arguments="jvar_type" return="jvar_style"/>
```

<g:evaluate>

The <g:evaluate> tag is used to evaluate an expression written in Rhino JavaScript and sometimes to set a variable to the value of the expression. The last statement in the expression is the value the variable will contain.

```
<g2:evaluate var="jvar_page" jelly="true">
     var page = "";
     var pageTitle = "";
     var pageGR = new GlideRecord("cmn_schedule_page");
     pageGR.addQuery("type", jelly.jvar_type");
     pageGR.query();
     if (pageGR.next()) {
        page = pageGR.getValue("sys_id");
        pageTitle = pageGR.getDisplayValue();
     }
     page;
</g2:evaluate>
```

```
<g2:evaluate var="not_important" expression="sc_req_item.popCurrent()"/>
```

object="true"

If you would like to have the evaluate return an object (for example an array), use the argument object="true".

Note: *This API call changed in the Calgary release:*

- *SncRelationships* replaces *Packages.com.glideapp.ecmdb.Relationships*

The new script object calls apply to the Calgary release and beyond. For releases prior to Calgary, substitute the packages call as described above. Packages calls are not valid beginning with the Calgary release. For more information, see Scripting API Changes.

```
<g2:evaluate object="true" var="jvar_items" expression="SncRelationships.getCMDBViews()" />
```

jelly="true"

If you would like to access Jelly variables inside an evaluate, include jelly="true" in the evaulate and add "jelly." before the Jelly variable's name. For example, to access the GlideJellyContext:

```
<g2:evaluate var="jvar_row_no" jelly="true">
   var gf = jelly.context.getGlideForm();
   var row = gf.getRowNumber();
   row;
</g2:evaluate>
```

Another example of accessing a jvar using the jelly="true" parameter. The value of **jvar_h** was set previously and we can access it inside the **evaluate**:

```
$[NLBR:jvar_h.getHTMLValue('newvalue')]
<g2:evaluate var="jvar_fix_escaping" jelly="true">
    var auditValue = jelly.jvar_h.getHTMLValue('newvalue');
    gs.log("************ " + auditValue);
</g2:evaluate>
```

copyToPhase2="true"

If you have a need to take the results of an evaluation that occurs in phase 1 and propagate it to phase 2, use copyToPhase2="true". There is some protection for escaping in this use. For example:

```
<g:evaluate var="jvar_has_special_inc" copyToPhase2="true">
   var specialInc = gs.tableExists("special_incident");
   specialInc;
</g:evaluate>
$[jvar_has_special_inc]
```

If you do not need to evaluate something, you can do this more directly. Beware of escaping issues here (double quotes in jvar_rows would cause a problem in the example):

```
<j2:set var="jvar_rows" value="${jvar_rows}"/>
```

<g:breakpoint/>

This tag can be used to display the current Jelly variables and their values in the log. Be sure to remove this tag before going to production.

<g:include_script/>

In pre-Fuji releases, this tag can be used to include UI scripts for client side access to reusable javascript. For example, the jquery library is included as a UI script. To view this library, navigate to **System UI > UI Scripts > jquery.min**. To include this library in a UI page or macro, add **<g:include_script src="jquery.min.jsdbx"/>** to the jelly script. This example uses the name of the UI script with the extension **.jsdbx** appended.

Starting with the Fuji release, you cannot use this include.

<g:ui_form/>

This tag defines a form on the UI page. For example, if your form contained the **application_sys_id** field:

```
<g:ui_form>
    <p>Click OK to run the processing script.</p>
   <g:dialog_buttons_ok_cancel ok="return true" />
   <input type="hidden" name="application_sys_id" value="499836460a0a0b1700003e7ad950b5da"/>
</g:ui_form>
```

The g:ui_form may benefit greatly from a processing script.

<g:ui_input_field />

This tag adds a reference to a UI macro that creates an input field on a page that allows users to input information. The ui_input_field passes a label, name, value, and size into the UI macro. Here is an example from a UI page:

```
<g:ui_input_field label="sys_id" name="sysid" value="9d385017c611228701d22104cc95c371" size="50"/>
```

<g:ui_checkbox/>

This tag puts a user-editable check mark on a page. The name and value are passed into the UI macro. Here is an example from a table on a UI page:

```
<table>
    <tr>
      <td nowrap="true">
          <label>Time Card Active:</label>
      </td>
      <td>
          <g:ui_checkbox name="timecard_active" value="${sysparm_timecard_active}"/>
      </td>
   </tr>
</table>
```

<g:dialog_buttons_ok_cancel/>

This tag puts buttons on the UI page that run a specified processing script if the tag returns true. If your UI page contains a form (uses the <g:form> tag), you can submit the form and have the Processing Script run. The Processing Script can naturally access fields on the form. For example, if your form contained the application_sys_id field:

```
<g:ui_form>
   <p>Click OK to run the processing script.</p>
   <g:dialog_buttons_ok_cancel ok="return true" />
   <input type="hidden" name="application_sys_id" value="499836460a0a0b1700003e7ad950b5da"/>
</g:ui_form>
```

<g:ui_reference/>

This tag adds a reference to a page that can be referenced by a Processing Script. The following example creates a reference defined by name, id, and table parameters in the tag:

```
<g:ui_reference name="QUERY:active=true^roles=itil" id="assigned_to" table="sys_user" />
```

Then in the Processing Script, reference the name field like this:

```
newTask.assigned_to =
request.getParameter("QUERY:active=true^roles=itil");
```

Beginning with the Calgary release, this tag has been enhanced to allow you to specify a reference qualifier, so that the "name" attribute can be unique. The following example creates a reference defined by name, id, and table parameters in the tag. Note: the "columns" attribute only applies to the auto-completer.

```
<g:ui_reference name="parent_id" id="parent_id" table="pm_project" query="active=true" completer="AJAXTableCompleter"
columns="project_manager;short_description"/>
```

Ampersand

Ampersands in Jelly can cause you grief because Jelly is XML. Use **${AMP}** to insert an ampersand in Jelly. If you are writing JavaScript that appears in the HTML part of say a UI page or UI macro that is actually going to run on the browser you are better off putting this code in the "client script" field and that way you can avoid escaping issues. However, if you really must put it in the "HTML" field, you will need to do something like this:

```
ta = ta[1].split('$[AMP]');
```

And

For the same reasons given for the ampersand, use **${AND}** to insert a JavaScript `and` in Jelly. For example:

```
if (d ${AND} e)
   var color = d.value;
```

Alternately, in a Jelly **test** you would use &amp&amp. For example:

```
<j:if test="${jvar_form_name == 'sys_form_template' &amp;&amp; !RP.isDialog()}">
```

Less Than

Similar to ampersands, less than ("<") signs can also cause problems due to Jelly being XML. This can be resolved by reversing your test such that it is not necessary or by using **${AMP}lt;** in place of the less than sign. For example in an evaluate you could do:

```
<g2:evaluate var="jvar_text">
     var days = "";
     var selectedDays = '$[${ref}]';
     for (var i = 1; i ${AMP}lt;= 7; i++) {
        if (selectedDays.indexOf(i.toString()) >= 0) {
           days += gs.getMessage("dow" + i);
           days += " ";
        }
     }
     days;
 </g2:evaluate>
```

Many times you can avoid the "less than" operator all together by just using "not equals" which doesn't have escaping issues. For example:

```
for (var i=0; i != ta.length; i++) {
}
```

Whitespace

Normally, white space is removed by Jelly. To keep it, you must specify that it not be trimmed. For example, the following keeps the space after the colon.

```
<j2:whitespace trim="false">${gs.getMessage('Did you mean')}: </j2:whitespace>
```

Spaces

To encode a non-breaking space (&nbsp;), you can use **$[SP]**. For example:

```
<span id="gsft_domain" style="display: inline">
    ${gs.getMessage('Domain')}:$[SP]
    <span id="domainDD" class="drop_down_element" style="text-decoration: none; color: white">
        ${gs.getMessage("Loading...")}
```

```
   </span>
</span>
```

RP

An object, RP, has various methods available to you. Here are a few of its methods:

- RP.getReferringURL() - returns to URL that got you to this page.
- RP.isDialog() - returns true if the page is in dialog mode
- RP.isHomePage - returns true if the page is in home page mode
- RP.isPrint() - returns true if the page is in print mode (user clicked the print button on the top banner)
- RP.isMobile() - returns true if the page is in mobile mode (browsing using a Blackberry, etc.)
- RP.getParameterValue(parameterName) - returns the value of the specified parameter passed on the URL
- RP.getParameters() - returns the parameters passed on the URL
- RP.getViewID() - returns the view in use

An example of using RP.isPrint():

```

<j:if test="${!RP.isPrint()}">
   <a onclick="mailTo('${ref}');">
      <img src="images/email.gifx" style="margin-left: 2px;" width="16" height="16"/>
   </a>
</j:if>
```

An example of using RP.getParameterValue(x) assuming a parameter was passed for "roles":

```
roles = "${RP.getParameterValue('roles')}";
```

Tracing Jelly

ServiceNow has a feature which allows the evaluation of Jelly to be traced. The trace is sent to the log. This should only be turned on during debugging as this produces a lot of logging. To turn on the trace, set the property glide.ui.template.trace to true. For example, the following script can be executed to do this:

Note: *This API call changed in the Calgary release:*

- *GlideProperties* replaces *Packages.com.glide.util.GlideProperties*

The new script object calls apply to the Calgary release and beyond. For releases prior to Calgary, substitute the packages call as described above. Packages calls are not valid beginning with the Calgary release. For more information, see Scripting API Changes.

```
GlideProperties.set('glide.ui.template.trace', true);
```

If you want to see your log entries on your web browser at the bottom of each page, navigate to *System Diagnostics > Debug Log*.

Standard Jelly

This page from Apache has a summary of the standard Jelly tags: http://commons.apache.org/jelly/tags.html

This PDF includes a nice summary of standard Jelly tags: http://www.softwaresummit.com/2003/speakers/GillardJelly.pdf

Escaping Jelly

Note: *This is an advanced procedure.*

For information about how to handle character escaping in Jelly files, see **How to Escape in Jelly**.

References

[1] https://docs.servicenow.com/bundle/jakarta-servicenow-platform/page/script/general-scripting/concept/c_ExtensionsToJellySyntax.html

How to Escape in Jelly

Note: *This article applies to Fuji and earlier releases. For more current information, see Jelly Tags* [1] *at* http://docs.servicenow.com **The ServiceNow Wiki is no longer being updated. Visit** http://docs.servicenow.com **for the latest product documentation.**

Note: This functionality requires a knowledge of **JavaScript, HTML, and Apache Jelly (a Java and XML based scripting and processing engine for turning XML into executable code)**.

Overview

This page describes the methods used to handle character escaping in Jelly files. These are advanced procedures. XML escaping behavior can be modified only by users with the elevated security_admin role.

Escaping Types

There are two different types of escaping that is required when generating output from Jelly:

- JavaScript
- HTML

The escaping for each of these consists of:

Type	From	To
JavaScript	' (single quote)	\'
	" (double quote)	\"
	CR (carriage return)	(blank)
	NL (newline)	\n ('\' followed by 'n')
HTML	& (ampersand)	&amp;
	< (less than)	&lt;
	> (greater than)	&gt;

You can also escape HTML using the getHTMLValue() [2] function which will enforce all line breaks and escape the characters mentioned above. It can be used as follows:

```
${test.getHTMLValue()}
```

Jelly Escaping

To add escaping to a jelly replacement, add a prefix to the ${expression} or $[expression] indicating the escaping to be performed:

```
${JS:expression}
${HTML:expression}
```

The prefix tells the system to take the result of the expression and escape it before outputting.

The escaping may be combined by specifying a comma-separated list of prefixes:

```
${JS,HTML:expression}
```

Examples

These examples use the following string as **jvar_test**:

```
This 'is' a test string & contains <stuff>
```

Best Practices

Escape on Output

Always use escaping when a string is written to the output stream (ie, ${} or $[]).

Old way:

```
<g2:evaluate var="jvar_current_user" expression="gs.escaper(gs.getUserDisplayName())"/>
```

New way:

```
<g2:evaluate var="jvar_current_user" expression="gs.getUserDisplayName()"/>
$[JS:jvar_current_user]
```

Escape Labels

Use HTML escaping for labels, but not for table and element names. These names are always alphanumeric to ensure that any character may be used in labels, messages, etc.. Script injection attacks are prevented by putting <script> tags in field text to force the script to execute when loading a form.

```
${HTML:jvar_title}
${HTML:ref.sys_meta.label}
${ref}    <== no escaping needed here
```

Other HTML Escaping

Use HTML rather than **GlideStringUtil.escapeHTML**.

Note: *This API call changed in the Calgary release:*

- *GlideStringUtil* replaces *Packages.com.glide.util.StringUtil*

The new script object calls apply to the Calgary release and beyond. For releases prior to Calgary, substitute the packages call as described above. Packages calls are not valid beginning with the Calgary release. For more information, see Scripting API Changes.

Old way:

```
<g:evaluate var="jvar_t" expression="GlideStringUtil.escapeHTML(${ref}.sys_meta.label)" />
```

New way:

```
<g:evaluate var="jvar_t" expression="${ref}.sys_meta.label" />
    ...
    ${HTML:jvar_t}
```

JavaScript

Perform escaping when adding a label or message (among other strings that might contain single quotes and newlines) in JavaScript to ensure valid syntax: Old way:

```
 <j:set var="jvar_edit_msg" value="${gs.getMessage('Edit Application')}"/>
      <script language="javascript">
         var id = "${jvar_application.ID}";
         var acm${jvar_application.ID} = new
GwtContextMenu('context_menu_app_' +id);
         acm${jvar_application.ID}.clear();
         acm${jvar_application.ID}.addURL("${jvar_edit_msg}",
"sys_app_application.do?sys_id=" + id, "gsft_main");
```

New way:

```
<j:set var="jvar_edit_msg" value="${gs.getMessage('Edit Application')}"/>
      <script language="javascript">
         var id = "${jvar_application.ID}";
         var acm${jvar_application.ID} = new
GwtContextMenu('context_menu_app_' +id);
         acm${jvar_application.ID}.clear();
         acm${jvar_application.ID}.addURL("${JS:jvar_edit_msg}",
            "sys_app_application.do?sys_id=" + id, "gsft_main");
```

References

[1] https://docs.servicenow.com/bundle/jakarta-servicenow-platform/page/script/general-scripting/reference/r_JellyTags.html

[2] http://wiki.service-now.com/index.php?title=GlideElement#getHTMLValue.28Int.29

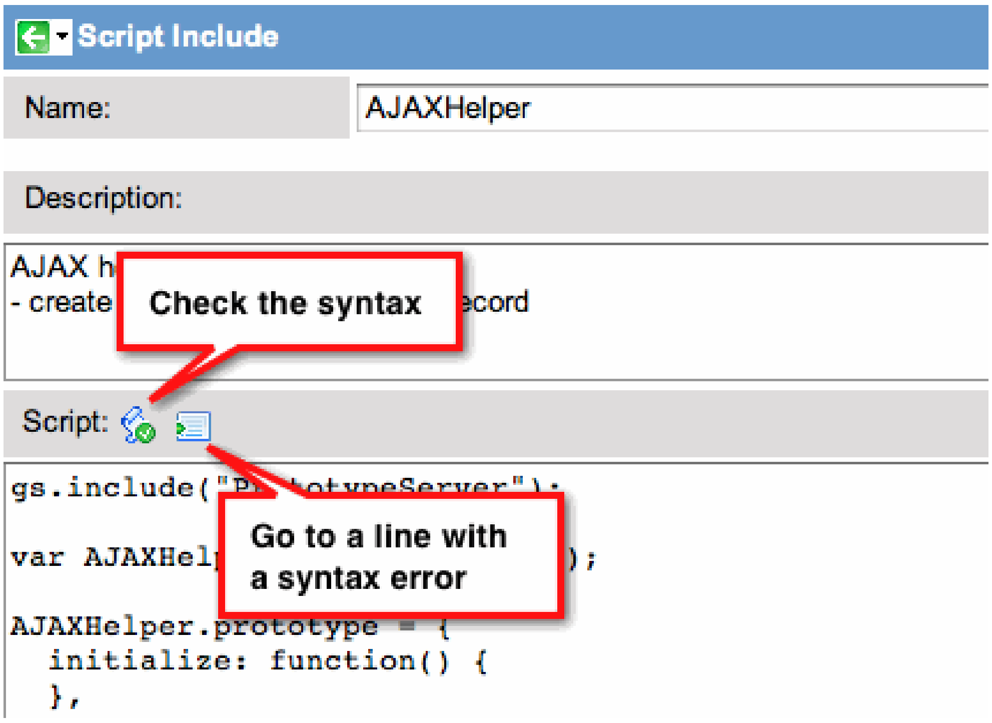
Debugging

Script Syntax Error Checking

Note: *This article applies to Fuji and earlier releases. For more current information, see Syntax Editor* [1] *at* http://docs.servicenow.com **The ServiceNow Wiki is no longer being updated. Visit** http://docs.servicenow.com **for the latest product documentation.**

Overview

All ServiceNow script fields provide controls for checking the syntax for errors and for locating the error easily when one occurs. The script editor places the cursor at the site of a syntax error and lets you search for errors in scripts by line number.

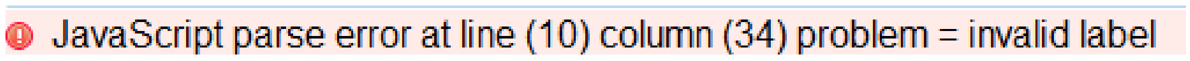


```
gs.include("PrototypeServer");

var AJAXHel                    );

AJAXHelper.prototype = {
  initialize: function() {
  },
```

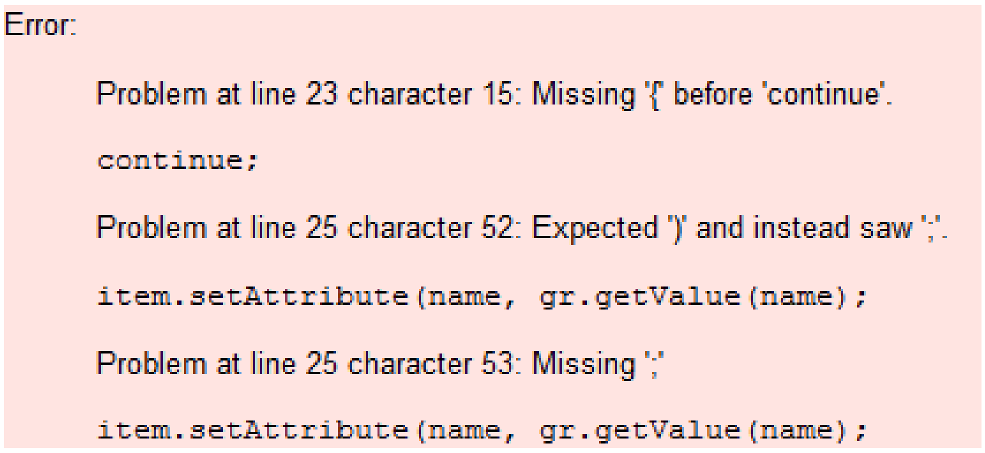
Error Checking

The ServiceNow script editor notifies you of syntax errors in your scripts when you:

- Save a new record or update an existing record. A banner appears at the bottom of the editor showing the location of the first error (line number and column number), and the cursor appears at the site of the error. Warnings presented at **Save** or **Update** show only one error at a time.


JavaScript parse error at line (10) column (34) problem = invalid label

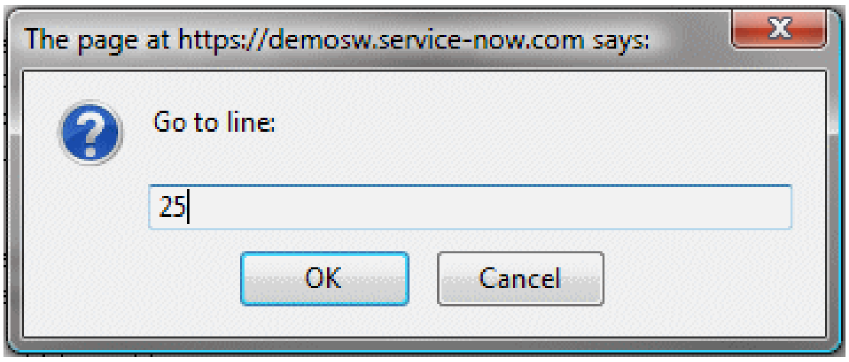
- Click on the syntax checking icon before saving or updating a record. A banner appears at the bottom of the editor showing the location of all errors in the script, and the cursor appears at the site of the first error.


```
Error:

Problem at line 23 character 15: Missing '{' before 'continue'.

continue;

Problem at line 25 character 52: Expected ')' and instead saw ';'.

item.setAttribute(name, gr.getValue(name);

Problem at line 25 character 53: Missing ';'

item.setAttribute(name, gr.getValue(name);
```

Searching for Errors by Line

To locate the exact position of the error in a large script, click the **Go to line** icon. This feature is particularly useful when you are encounter a syntax error in a log file rather than in the ServiceNow record itself. In this case, you can navigate to the record and search for errors by line number. In the dialog box that appears, enter the line number of an error, and then click **OK**. Your view moves to the site of the error, and the cursor marks the correct line and column.

Note: *For this feature to function, you must disable the Syntax Editor.*


The page at https://demosw.service-now.com says:

Go to line:

25

References

[1] https://docs.servicenow.com/bundle/jakarta-servicenow-platform/page/script/general-scripting/concept/c_SyntaxEditor.html

Syntax Editor

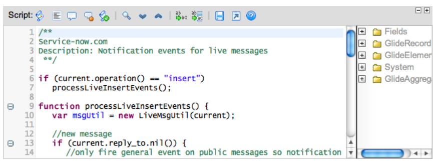
Note: *This article applies to Fuji and earlier releases. For more current information, see Syntax Editor* [1] *at* http://docs.servicenow.com **The ServiceNow Wiki is no longer being updated. Visit** http://docs.servicenow.com **for the latest product documentation.**

Overview

The syntax editor enables the following features for all script fields:

- JavaScript syntax coloring, indentation, line numbers, and automatic creation of closing braces and quotes
- Code editing functions
- Code syntax checking
- Script macros for common code shortcuts

This article helps administrators activate and configure the Syntax Editor plugin. To learn how to use this feature, see Using the Syntax Editor.



```
/**
Service-now.com
Description: Notification events for live messages
 **/

if (current.operation() == "insert")
   processLiveInsertEvents();

function processLiveInsertEvents() {
   var msgUtil = new LiveMsgUtil(current);

   //new message
   if (current.reply_to.nil()) {
      //only fire general event on public messages so notification
```

JavaScript syntax editor

Managing Script Macros

Script macros provide shortcuts for typing commonly used code. Several script macros are available by default. Administrators can define new or modify existing script macros.

1. Navigate to **System Definition > Syntax Editor Macros**.
2. Click **New** or select the macro to edit.
3. Define the macro details with the following fields:
   - **Name:** shortcut that users type to use the macro.
   - **Comments:** description of the macro. Appears when the user types **help**.
   - **Text:** full macro text that replaces the name in the editor.
4. [Optional] Define the the cursor position after the macro is executed by entering **$0** in the macro text. For example, in the `vargr` macro, the cursor is placed between the quotes in the first line. The $0 is not written out as part of the macro text.

Disabling the Syntax Editor

When the syntax editor is enabled, users can choose whether to use it. This choice is stored by the glide.ui.javascript_editor user preference.

Administrators can disable the syntax editor for all users, regardless of user preference.

1. Enter **sys_properties.list** in the navigation filter.
2. Locate the `glide.ui.javascript_editor` property.
3. Set the **Value** to **false**.

Configuring Syntax Checking

Administrators can configure the syntax editor to show real-time error and warning indicators beside a line of code that contains a syntax error.

1. Enter **sys_properties.list** in the navigation filter.
2. Locate the `glide.ui.syntax_editor.show_warnings_errors` property.
3. Set the **Value** to one of the following options:
   - **HIDE_ALL:** hides errors and warnings (default).
   - **ERRORS_ONLY:** shows errors but hides warnings.
   - **SHOW_ALL:** shows errors and warnings.

Installed Components

Tables

The following table is added:

- **Editor Macro [syntax_editor_macro]:** manages script macros, which are shortcuts for typing commonly used code. Navigate to **System Definition > Syntax Editor Macros**. Several script macros are added by default.

Properties

The following properties are added:

- `glide.ui.javascript_editor`: enables or disables the syntax editor for all users.
- `glide.ui.syntax_editor.show_warnings_errors`: configures syntax checking (warnings and errors) for the syntax editor.

Using the Syntax Editor

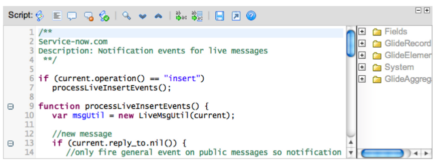
Note: *This article applies to Fuji and earlier releases. For more current information, see Syntax Editor* [1] *at* http://docs.servicenow.com **The ServiceNow Wiki is no longer being updated. Visit** http://docs.servicenow.com **for the latest product documentation.**

Overview

The syntax editor enables the following features for all script fields:

- JavaScript syntax coloring, indentation, line numbers, and automatic creation of closing braces and quotes
- Code editing functions
- Code syntax checking
- Script macros for common code shortcuts

This feature requires the Syntax Editor plugin.



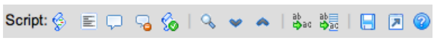
```
/**
Service-now.com
Description: Notification events for live messages
 **/

if (current.operation() == "insert")
   processLiveInsertEvents();

function processLiveInsertEvents() {
   var msgUtil = new LiveMsgUtil(current);

   //new message
   if (current.reply_to.nil()) {
      //only fire general event on public messages so notification
```

JavaScript syntax editor

Enabling the Syntax Editor

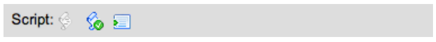
You can choose whether to use the syntax editor. This choice is stored as a user preference, and the editor opens in the last mode selected the next time you access a script field. To disable or enable the syntax editor, click the **Toggle syntax editor** ( ) icon.

The syntax editor is enabled when the **Toggle syntax editor** icon and the code appear in color. The code editing tools are available in this mode only.



Syntax editor is enabled

The syntax editor is disabled when the **Toggle syntax editor** icon appears in gray and the code appears in black. The **Go to line** icon is available in this mode only.



Syntax editor is disabled

Navigating to a Line Number

To navigate to a specific line in the code when the syntax editor is disabled:

1. Click the **Go to line** ( ) icon. A text field appears.

   This icon is not available when the editor is enabled.
2. Enter a number in the field and then press **Enter**.

Editing JavaScript with the Syntax Editor

The following table lists the JavaScript editing functions that are available on the syntax editor toolbar.

Icon	Keyboard Shortcut	Name	Description
	N/A	Toggle Syntax Editor	Disables the syntax editor. Click the button again to enable the syntax editor. See Enabling the Syntax Editor.
	Access Key [1] + R	Format Code	Applies the proper indentation to the script.
	Access Key + C	Comment Selected Code	Comments out the selected code.
	Access Key + U	Uncomment Selected Code	Removes comment codes from the selected code.
	N/A	Check Syntax	Checks the code for syntax errors. See Checking Code Syntax.
	Access Key + \	Start Searching	Highlights all occurrences of a search term in the script field and locates the first occurrence. Click the icon, then enter the search term and press **Enter**. You can use regular expressions [2] enclosed in slashes to define the search term. For example, the term **/a{3}/** locates **aaa**.
	Access Key + [	Find Next	Locates the next occurrence of the current search term in the script field. Use **Start Searching** to change the current search term.
	Access Key +]	Find Previous	Locates the previous occurrence of the current search term in the script field. Use **Start Searching** to change the current search term.
	Access Key + W	Replace	Replaces the next occurrence of a text string in the script field. 1. Click the icon, then enter the string to replace and press **Enter**. You can use regular expressions enclosed in slashes to define the string to replace. For example, the term **/a{3}/** locates **aaa**. 2. Enter the the replacement string and press **Enter**.
	Access Key + ;	Replace All	Replaces all occurrences of a text string in the script field. 1. Click the icon, then enter the string to replace and press **Enter**. You can use regular expressions enclosed in slashes to define the string to replace. For example, the term **/a{3}/** locates **aaa**. 2. Enter the the replacement string and press **Enter**.
	N/A	Save	Saves changes without leaving the current view. Use this button in full screen mode to save without returning to standard form view.
	Access Key + L	Toggle Full Screen Mode	Expands the script field to use the full form view for easier editing. Click the button again to return to standard form view. This feature is not available for Internet Explorer.
	Access Key + P	Help	Displays the keyboard shortcuts help screen.

Additional Editing Tips

- To fold a code block, click the minus sign beside the first line of the block. The minus sign only appears beside blocks that can be folded. To unfold the code block, click the plus sign.
- To insert a fixed space anywhere in your code, press **Tab**.
- To indent a single line of code, click in the leading whitespace of the line and then press **Tab**.
- To indent one or more lines of code, select the code and then press **Tab**. To decrease the indentation, press **Shift + Tab**.
- To remove one tab from the start of a line of code, click in the line and press **Shift + Tab**.

Checking Code Syntax

To check for syntax errors in a script field, click the **Check syntax** ( ) icon. Error details appear below the script field. To learn more, see Script Syntax Error Checking.

The syntax editor also provides real-time error ( ) and warning ( ) indicators beside a line of code that contains a syntax error. To view the error or warning details, point to the indicator beside the line number.

Note:

- *Administrators must configure real-time syntax checking to show errors only or both error and warnings.*
- *The system checks for syntax errors when you save a record. If an error is found, details appear beneath the field and the record is not saved. This check occurs on all script fields and cannot be disabled.*

Using Script Macros

Script macros provide shortcuts for typing commonly used code. To use a macro, type the macro name in a script field and press **Tab**. The macro name is replaced with the full macro text.

The following macros are available by default. Administrators can define additional script macros.

To view a list of the macros that are available in your system, type **help** in a script field and press **Tab**.

Name	Description	Code text

`vargror`	GlideRecord query with OR condition	`var gr = new GlideRecord('$0');`
		`var qc = gr.addQuery('field', 'value1');`
		`qc.addOrCondition('field', 'value2');`
		`gr.query();`
		`while (gr.next()) {`
		
		`}`

`vargr`	GlideRecord query example	`var gr = new GlideRecord("$0");`
		`gr.addQuery("name", "value");`
		`gr.query();`
		`if (gr.next()) {`
		
		`}`

`method`	Standard JavaScript class method	`/*________________________________________`
		`    * Description:`
		`    * Parameters:`
		`    * Returns:`
		`    ________________________________________*/`
		`    $0: function() {`
		
		`    },`

`info`	Add an info message to the current session	`gs.addInfoMessage("$0");`

`for`	Standard loop for arrays	`for (var i=0; i< myArray.length; i++) {`
		`//myArray[i];`
		`}`

`doc`	Code documentation (comment) header	`/**`
		`* Description: $0`
		`* Parameters:`
		`* Returns:`
		`*/`

JavaScript Resources

Scripts in ServiceNow use ECMA 262 standard JavaScript. Helpful resources include:

- Mozilla: http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference
- ECMA Standard in PDF format: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf
- History and overview: http://javascript.crockford.com/survey.html
- JavaScript number reference: http://www.hunlock.com/blogs/The_Complete_Javascript_Number_Reference

References

[1] http://en.wikipedia.org/wiki/Access_key
[2] http://www.regular-expressions.info/reference.html

JavaScript Debug Window

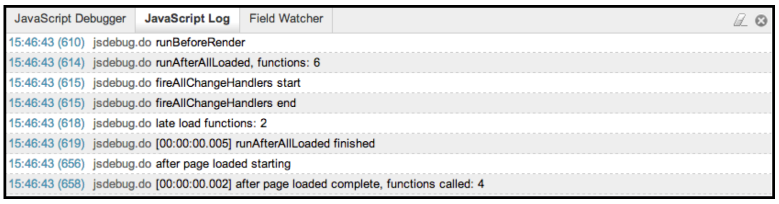
Note: *This article applies to Fuji and earlier releases. For more current information, see JavaScript Debug Window* [1] *at* http://docs.servicenow.com **The ServiceNow Wiki is no longer being updated. Visit** http://docs.servicenow.com **for the latest product documentation.'**

Overview

The JavaScript debug window appears in a bottom pane of the user interface when an administrator turns on debugging. Use the debug window to access these tools:

- **JavaScript Debugger:** a tool that allows application developers and administrators to efficiently debug scripts that drive the applications they develop and support (available starting with the Dublin release).
- **JavaScript Log:** JavaScript that runs on the browser, such as client scripts, can include a call to `jslog()` to send information to the JavaScript log.
- **Field Watcher:** a tool that tracks and displays all actions that the system performs on a selected form field (available starting with the Dublin release).

Note: *In versions prior to the Dublin release, the JavaScript log displays in another browser window.*



```
15:46:43 (610)  jsdebug.do runBeforeRender
15:46:43 (614)  jsdebug.do runAfterAllLoaded, functions: 6
15:46:43 (615)  jsdebug.do fireAllChangeHandlers start
15:46:43 (615)  jsdebug.do fireAllChangeHandlers end
15:46:43 (618)  jsdebug.do late load functions: 2
15:46:43 (619)  jsdebug.do [00:00:00.005] runAfterAllLoaded finished
15:46:43 (656)  jsdebug.do after page loaded starting
15:46:43 (658)  jsdebug.do [00:00:00.002] after page loaded complete, functions called: 4
```

An example of the JavaScript log

Using the JavaScript Debug Window

1. Click the debug icon ( ).
   - **UI14:** click the gear icon ( ) on the banner frame and then click the debug icon.
   - **UI11 and Classic:** click the debug icon on the banner frame.

   The JavaScript debug window opens at the bottom of the screen. The tab that is currently active in the window is the last tab that was active when the window was closed.
2. Click a tab to use one of the debug window features.
   - JavaScript Debugger
   - JavaScript Log
   - Field Watcher

Enhancements

Eureka

- The JavaScript debug icon and window remain available to the system administrator while impersonating another user with the itil or ESS roles.

References

[1] https://docs.servicenow.com/bundle/jakarta-servicenow-platform/page/script/debugging/concept/c_JavaScriptDebugWindow.html

Glide Reference

GlideRecord

Note: *This article applies to Fuji. For more current information, see GlideRecord* [1] *at* http://docs.servicenow.com **The ServiceNow Wiki is no longer being updated. Visit** http://docs.servicenow.com **for the latest product documentation.**

Overview

GlideRecord is a special Java class (GlideRecord.java) that can be used in JavaScript exactly as if it was a native JavaScript class.

GlideRecord

- is used for database operations instead of writing SQL queries.
- is an object that contains zero or more records from one table. Another way to say this is that a GlideRecord is an ordered list.

Note: *The global Glide APIs are not available in scoped scripts. There are scoped Glide APIs for use in scoped scripts. The scoped Glide APIs provide a subset of the global Glide API methods. For information on scoped applications see Application Scope, scripting in scoped applications, and the list of scoped APIs.*

Use this API to:

- Query
- Get
- Set
- Update
- Insert
- Delete

A GlideRecord contains both records (rows) and fields (columns). The field names are the same as the underlying database column names.

For more information on available JavaScript operators and additional GlideRecord examples, see Using GlideRecord to Query Tables.

Query

Query Method Summary

Return Value	Details
QueryCondition	**addActiveQuery**() Adds a filter to return active records.
void	**addDomainQuery**(`Object o`) Changes the domain used for the query from the user's domain to the domain of the provided GlideRecord.
void	**addEncodedQuery**(`String query`) Adds an encoded query. Adds an encoded query to the other queries that may have been set.
QueryCondition	**addInactiveQuery**() Adds a filter to return inactive records.
QueryCondition	**addJoinQuery**(`joinTable, [primaryField], [joinTableField]`) Adds a filter to return records based on a relationship in a related table.
QueryCondition	**addNotNullQuery**(`String fieldName`) Adds a filter to return records where the specified field is not null.
Query Condition	**addNullQuery**(`String fieldName`) Adds a filter to return records where the specified field is null.
QueryCondition	**addOrCondition**(`String fieldName, Object operator, Object value`) Adds an alternative filter to an existing query to return records based on 1, 2 or 3 arguments. • 1 argument adds an encoded query string. • 2 arguments return records where the field is equal to the value (or is in a list of values). • 3 arguments return records where the field meets the specified condition (field, operator and value).
QueryCondition	**addQuery**(`String fieldName, Object operator, Object value`) Adds a filter to return records based on 1, 2 or 3 arguments. • 1 argument adds an encoded query string. • 2 arguments return records where the field is equal to the value (or is in a list of values). • 3 arguments return records where the field meets the specified condition (field, operator and value).
boolean	**canCreate**() Determines if the Access Control Rules (which includes the user's role) permit inserting new records in this table.
boolean	**canDelete**() Determines if the Access Control Rules (which includes the user's role) permit deletion of records in this table.
boolean	**canRead**() Determines if the Access Control Rules (which includes the user's role) permit reading this table.
boolean	**canWrite**() Determines if the Access Control Rules (which includes the user's role) permit updates to records in this table.
boolean	**changes**() Determines whether any of the fields in the record have changed.
boolean	**find**(`columnName, value`) Returns true if any record has a matching value in the specified column. If found, it also moves to the first record that matches, essentially executing next() until the record is returned.
boolean	**hasAttachments**() Determines if the current record has any attachments.
boolean	**hasNext**() Determines if there are any more records in the GlideRecord.

boolean	**instanceOf**(`String className`) Checks a table for a type/class of record.
boolean	**isNewRecord**() Determines whether the current record has been inserted into the database.
boolean	**isValid**() Determines whether the table exists or not.
boolean	**isValidField**(`String columnName`) Determines if the given field is defined in the current table.
boolean	**isValidRecord**() Determine if there is another record in the GlideRecord.
boolean	**next**() Moves to the next record in the GlideRecord.
boolean	**_next**() Provides the same functionality as `next`, intended to be used in cases where the GlideRecord has a column named **next.**
String	**operation**() Retrieves the current operation being performed, such as insert, update, delete, etc.
void	**orderBy**(`String name`) Specifies an orderBy column (this may be called more than once to order by multiple columns).
void	**orderByDesc**(`String name`) Specifies a decending orderBy column.
void	**query**(`Object field, Object value`) Runs the query against the table based on the specified filters by addQuery and addEncodedQuery. This will query the GlideRecord table as well as any references of the table. One argument adds a query string. Usually this is performed without arguments, but a name/value pair can be specified.
void	**queryNoDomain**(`Object field, Object value`) Used only in domain separated instances. Performs the same function as `query();`, but turns off domain processing.
void	**_query**(`Object field, Object value`) Identical to `query();`, intended to be used on tables where there is a column named **'query**, which would interfere with using `query();`.
void	**restoreLocation**() Sets the current record to be the record that was saved with `saveLocation()`. If saveLocation has not been called yet, the current record is set to be the first record of the GlideRecord.
void	**saveLocation**() Save the current row number so that we can get back to this location using `restoreLocation()`.

Get

Get Method Summary

Return Value	Details
boolean	**get**(`Object name, Object value`) Defines a GlideRecord based on the specified expression of 'name = value'. If value is not specified, then the expression used is 'sys_id = name'. This method is expected to be used to query for single records, so a 'next' operation on the GlideRecord is performed by this method before returning.
String	**getAttribute**(`String attribute`) Retrieves the attributes on the field in question from the dictionary.
String	**getClassDisplayValue**() Retrieves the label for the table.
String	**getDisplayValue**() Retrieves the display value for the current record.
ElementDescriptor	**getED**() Retrieves the element's descriptor.
GlideElement	**getElement**(`String columnName`) Retrieves the GlideElement for a specified field.
String	**getEncodedQuery**() Retrieves the encoded query as a string.
String	**getEscapedDisplayValue**() Retrieves the field value of the current record and escapes it for use in Jelly scripts.
ArrayList	**getFields**() Retrieves an array of fields in the current record.
String	**getLabel**() Retrieves the appropriate label for a field.
String	**getLink**(`boolean noStack`) Retrieves the link for the current record.
int	**getLocation**() Retrieves the current row number.
string	**getPlural**() Retrieves the plural label of the GlideRecord table.
String	**getRecordClassName**() Retrieves the class name for the current record.
HashMap	**getRelatedLists**() Retrieves a list of names and display values of tables that **refer** to the current record.
HashMap	**getRelatedTables**() Retrieves a list of names and display values of tables that are **referred to** by the current record.
int	**getRowCount**() Retrieves the number of rows in the GlideRecord.
int	**getRowNumber**() Retrieves the row number set by saveLocation() or setLocation().
String	**getTableName**() Retrieves the table name associated with this GlideRecord.

String	**getValue**(`String fieldName`) Retrieves the string value of an underlying element in a field.

Set

Set Method Summary	
Return Value	**Details**
void	**autoSysFields**(`Boolean e`) Enables or disables the update to the fields sys_updated_by, sys_updated_on, sys_mod_count, sys_created_by, and sys_created_on. This is often used for manually updating field values on a record while leaving historical information unchanged. **Note:** This is not available for scoped apps, starting with the Fuji release. See the Scoped GlideRecord API Reference for a list of what APIs are available for scoped apps.
void	**setAbortAction**(`boolean b`) Sets a flag to indicate if the next database action (insert, update, delete) is to be aborted.
void	**setDisplayValue**(`String name, Object value`) Sets the field name to the specified display value. • For a reference field this is the display value for the table. • For a date/time field this is the time in the caller's current timezone.
void	**setForceUpdate**(`boolean e`) Updates the record even if fields have not been changed.
void	**setLimit**(`int limit`) Sets the limit for how many records in the GlideRecord.
void	**setLocation**(`int Number`) Sets the current row location.
void	**setNewGuid**() Generates a new GUID and sets it as the unique id for the current record.
void	**setNewGuidValue**(`String guid`) Generates a new GUID and sets it as the unique id for the current record, when inserting a new record.
void	**setQueryReferences**(`boolean queryReferences`) Enables or disables using the reference field's display name when querying a reference field.
void	**setUseEngines**(`boolean e`) Disable or enable the running of any engines (approval rules / assignment rules).
void	**setValue**(`String fieldName, Object value`) Sets the specified field to the specified value. Normally a script would do a gr.category = value. However, if the element name is a variable then gr.setValue(elementName, value) can be used.
return	**setWorkflow**(`boolean e`) Enables or disables the running of business rules that might normally be triggered by subsequent actions.

Update

Update Method Summary	
Return Value	**Details**
void	**applyTemplate**(`String template`) Apply a template record (from sys_templates) to the current record. If the specified template is not found, no action is taken.
string	**update**(`Object reason`) Updates the GlideRecord with any changes that have been made. If the record does not already exist, it is inserted.
string	**updateWithReferences**(`Object reason`) Updates a record and also inserts or updates any related records with the information provided.

Insert

Insert Method Summary	
Return Value	**Details**
void	**initialize**() Creates an empty record suitable for population before an insert.
string	**insert**() Insert a new record using the field values that have been set for the current record.
string	**insertWithReferences**() Inserts a new record and also inserts or updates any related records with the information provided.
void	**newRecord**() Creates a GlideRecord, set the default values for the fields and assign a unique id to the record. See Also: initialize().

Delete

Delete Method Summary	
Return Value	**Details**
void	**deleteMultiple**() Delete multiple records according to the current "where" clause. Does not delete attachments.
boolean	**deleteRecord**() Delete a single record.

Query Methods

addActiveQuery

public QueryCondition addActiveQuery()

Adds a filter to return active records.

Returns:

`QueryCondition` - filter to return active records.

Example:

```
var inc = new GlideRecord('incident');
inc.addActiveQuery();
inc.query();
```

addEncodedQuery

public void addEncodedQuery(`String query`)

Adds an encoded query. Adds an encoded query to the other queries that may have been set.

Parameters:

`query` - An encoded query string to add to the record.

Example:

```
var queryString = "priority=1^ORpriority=2";
 var gr = new GlideRecord('incident');

 gr.addEncodedQuery(queryString);
 gr.query();
 while (gr.next()) {
   gs.addInfoMessage(gr.number);
 }
```

Use the breadcrumbs and filters to generate encoded query strings.

addDomainQuery

public void addDomainQuery(`Object o`)

Changes the domain used for the query from the user's domain to the domain of the provided GlideRecord.

Parameters:

`o` - A GlideRecord from which to draw the domain.

Example:

```
//This example requires the Domain plugin be active, the Group table is
 the specified Domain table,
//and the ITIL user is in the Database Atlanta domain
//From any domain (using queryNoDomain()) look up the incidents that an
 ITIL user can only see
//who is in the Database Atlanta domain, should expect all incidents
with the global or the
//Database Atlanta domain specified.
var domain = new GlideRecord('sys_user');
domain.addQuery('user_name', 'itil');
domain.queryNoDomain();
if (domain.next()) {
    var domainQuery = new GlideRecord('incident');
    domainQuery.addQuery('active', true);
    domainQuery.addDomainQuery(domain);
    domainQuery.query();
    gs.print('Number of Incidents for ITIL user: ' +
domainQuery.getRowCount());
    while (domainQuery.next())
        gs.print(domainQuery.number);
}
```

addInactiveQuery

public QueryCondition addInactiveQuery()

Adds a filter to return inactive records.

Returns:

`QueryCondition` - records where the active flag is false.

Example:

```
var inc = new GlideRecord('incident');
inc.addInactiveQuery();
inc.query();
```

addJoinQuery

public QueryCondition addJoinQuery(`joinTable, [primaryField], [joinTableField]`)

Adds a filter to return records based on a relationship in a related table.

For example, find all the users that are in the database group (users via *sys_user_grmember* table). Another example would be find all problems that have an assigned incident (problems via the *incident.problem_id* relationship).

Note: this is not a true database join; rather, `addJoinQuery` adds a subquery. So, while the result set is limited based on the join, the only fields that you have access to are those on the base table (those which are in the table with which the GlideRecord was initialized).

Parameters:

`joinTable` - table name.

`primaryField` (optional) - if other than sys_id, the primary field.

`joinTableField` (optional) - if other than sys_id, the field that joins the tables.

Returns:

`QueryCondition` - records where the relationships match.

Example:

Find problems that have an incident attached. This example returns problems that have associated incidents. However, it won't pull values from the incidents that are returned as a part of the query.

```
var prob = new GlideRecord('problem');
prob.addJoinQuery('incident');
prob.query();
```

Example:

Find active=false problems with associated incidents.

```
/ Look for Problem records
var gr = new GlideRecord('problem');

// That have associated Incident records
var grSQ = gr.addJoinQuery('incident');

// Where the Problem records are "active=false"
gr.addQuery('active', 'false');

// And the Incident records are "active=true"
grSQ.addCondition('active', 'true');

// Query
gr.query();

// Iterate and print results
while (gr.next()) {
    gs.print(gr.getValue('number'));
}
```

Example:

Find problems that have incidents associated where the incident **caller_id** field value matches that of the problem **opened_by** field.

```
var gr = new GlideRecord('problem');
gr.addJoinQuery('incident', 'opened_by', 'caller_id');
gr.query();
```

addNotNullQuery

public QueryCondition addNotNullQuery(`String fieldName`)

Adds a filter to return records where the specified field is not null.

Parameters:

`fieldName` - string name for a field.

Returns:

`QueryCondition` - QueryCondition of records where the parameter field is not null.

Example:

```
var target = new GlideRecord('incident');
  target.addNotNullQuery('short_description');
  target.query();    // Issue the query to the database to get all
records
  while (target.next()) {
     // add code here to process the incident record
  }
```

addNullQuery

public QueryCondition addNullQuery(`String fieldName`)

Adds a filter to return records where the specified field is null.

Parameters:

`fieldName` - string name for a field.

Returns:

`QueryCondition` - QueryCondition of records where the parameter field is null.

Example:

```
var target = new GlideRecord('incident');
  target.addNullQuery('short_description');
  target.query();    // Issue the query to the database to get all
records
  while (target.next()) {
     // add code here to process the incident record
  }
```

addOrCondition

public QueryCondition addOrCondition(`String name, String oper, Object value`)

Appends an OR condition to an existing condition based on the parameters entered. When using multiple query parameters, the addOrCondition method adds an OR condition to the last query.

Adds a disjunctive filter condition to return records based on 1, 2 or 3 arguments.

- 1 argument adds an encoded query string.
- 2 arguments return records where the field is equal to the value (or is in a list of values).
- 3 arguments return records where the field meets the specified condition (field, operator and value).

Parameters:

`name` - string name of a field.

`oper` - an operator for the query.

`value` - value to query on.

Returns:

`QueryCondition` - a reference to the QueryCondition that was added to the GlideRecord.

Example:

```
var gr = new GlideRecord('incident');
var qc = gr.addQuery('category', 'hardware');
qc.addOrCondition('category', 'software');
gr.addQuery('priority', '1');
```

addQuery

public QueryCondition addQuery(`String name, Object operator, Object value`)

Adds a filter to return records based on 1, 2 or 3 arguments.

- 1 argument adds an encoded query string.
- 2 arguments return records where the field is equal to the value (or is in a list of values).
- 3 arguments return records where the field meets the specified condition (field, operator and value).

Can accept an `IN` operator (see example 2).

Parameters:

`name` - string name of a field.

`operator` - an operator for the query.

`value` - value to query on.

Returns:

`QueryCondition` - a reference to the QueryCondition that was added to the GlideRecord.

Example 1:

```
var rec = new GlideRecord('incident');
rec.addQuery('active',true);
rec.addQuery('sys_created_on', ">", "2010-01-19 04:05:00");
rec.query();
while (rec.next()) {
 rec.active = false;
 gs.print('Active incident ' + rec.number + ' closed');
 rec.update();
}
```

Example 2 - Using the IN Operator:

```
var que = new GlideRecord('incident');
que.addQuery('number','IN','INC00001,INC00002');
que.query();
while(que.next()) {
 //do something....
}
```

Controlling invalid queries

By default queries with invalid field names run but ignore the invalid condition. For more strict query control you can enable the glide.invalid_query.returns_no_rows property which will result in an empty result set for invalid queries.

When this property is enabled you can temporarily override this logic on the session by turning off strict query control.

```
gs.getSession().setStrictQuery(false);
```

Example:

```
/*
 * Default behavior if no property or property = false
 * Remove invalid query condition
 */
var gr = new GlideRecord("incident");
gr.addQuery("field_that_doesnt_exist", "my value");
gr.addQuery("active", true);
gr.query();
//returns records matching active=true

/*
 * Property = true
 * Invalidate entire query if invalid condition exists
 */
var gr = new GlideRecord("incident");
gr.addQuery("field_that_doesnt_exist", "my value");
gr.addQuery("active", true);
gr.query();
// returns no records
//also generated log message "field name 'field_that_doesnt_exist' not
found in table 'incident'"


/*
 * Property = true, override strict query for session
 * Same behavior as setting property = false except only applies to
this session
 */
//disable strict query
gs.getSession().setStrictQuery(false);

var gr = new GlideRecord("incident");
gr.addQuery("field_that_doesnt_exist", "my value");
gr.addQuery("active", true);
gr.query();
//returns records matching active=true

//restore strict query
gs.getSession().setStrictQuery(true);
```

canCreate

public boolean canCreate()

Determines if the Access Control Rules (which includes the user's role) permit inserting new records in this table.

Returns:

`boolean` - true if the user's role permits creation of new records in this file.

canDelete

public boolean canDelete()

Determines if the Access Control Rules (which includes the user's role) permit deletion of records in this table.

Returns:

`boolean` - true if can delete or false if cannot delete.

Example:

Note: *This API call changed in the Calgary release:*

- *GlideSecurityManager* replaces *Packages.com.glide.sys.security.GlideSecurityManager*

The new script object calls apply to the Calgary release and beyond. For releases prior to Calgary, substitute the packages call as described above. Packages calls are not valid beginning with the Calgary release. For more information, see Scripting API Changes.

```
var att = new GlideRecord('sys_attachment');
att.get('$[sys_attachment.sys_id]');
var sm = GlideSecurityManager.get();
var checkMe = 'record/sys_attachment/delete';
var canDelete = sm.hasRightsTo(checkMe, att);
gs.log('canDelete: ' + canDelete);
```

canRead

public boolean canRead()

Determines if the Access Control Rules (which includes the user's role) permit reading this table.

Returns:

`boolean` - true if can read or false if cannot read.

canWrite

public boolean canWrite()

Determines if the Access Control Rules (which includes the user's role) permit updates to records in this table.

Returns:

`boolean` - true if can write or false if cannot write to the table.

changes

public boolean changes()

Determines whether any of the fields in the record have changed.

Returns:

`boolean` - true if any of the fields in the record have changed.

find

public boolean find(`columnName, value`)

Returns true if any record has a matching value in the specified column. If found, it also moves to the first record that matches, essentially executing next() until the record is returned.

Parameters:

`columnName` - specifies the column name in a record.

`value` - specifies the value to check for in the specified column.

Returns:

`boolean` - true if any record has a matching value in the specified column.

hasAttachments

public boolean hasAttachments()

Determines if the current record has any attachments.

Returns:

`boolean` - true if the current record has attachments.

Example:

```
  //Check for attachments and add link if there are any
  var attachment_link = '';
  var rec = new GlideRecord('sc_req_item');
  rec.addQuery('sys_id', current.request_item);
  rec.query();
  if(rec.next()){
    if(rec.hasAttachments()){
      attachment_link = gs.getProperty('glide.servlet.uri') +
rec.getLink();
    }
  }
```

hasNext

public boolean hasNext()

Determines if there are any more records in the GlideRecord.

Parameters:

`boolean` - true if there are more records in the query set.

Example:

```
if (gr.hasNext()) {
  dothis(); // found it, do it
} else {
  dothat(); // didn't find it
}
;
```

instanceOf

public boolean instanceOf(`String className`)

Checks a table for a type/class of record.

Parameters:

`className` - A string name of a type or class of record.

Returns:

`boolean` - true if table is instance of specified class.

Example:

isNewRecord

public boolean isNewRecord()

Determines whether the current record has been inserted into the database. This method returns true only if the `newRecord()` method has been called. This method is useful for scripted ACL, and in the condition of UI actions, but should not be used in background scripts.

Note: This method returns true for any new record during a business rule, or if the `newRecord()` method is used to initialize a record with default values and a unique ID (sys_id). In all other cases, it returns false.

Returns:

`boolean` - true if the current record is new (not inserted into the database).

Example:

Example of scripted ACL:

```
answer = gs.hasRole("filter_admin") || current.isNewRecord()
```

Example UI action condition:

```
*** Script: true
*** Script: false
```

isValid

public boolean isValid()

Determines whether the table exists or not.

Returns:

`boolean` - true if table is valid or if record was successfully fetched, or false if table is invalid or record was not successfully fetched.

Example:

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

var testTable = new GlideRecord('wrong_table_name');
gs.print(testTable.isValid());
```

Output:

```
*** Script: true
*** Script: false
```

isValidField

public boolean isValidField(`String columnName`)

Determines if the given field is defined in the current table.

Parameters:

`columnName` - column name of a field as a string.

Returns:

`boolean` - true if field is valid.

isValidRecord

public boolean isValidRecord()

Determine if there is another record in the GlideRecord.

Returns:

`boolean` - true if the current record is valid or false if past the end of the recordset.

next

public boolean next()

Moves to the next record in the GlideRecord.

Returns:

`boolean` - false if there are no more records in the query set .

Example:

```
var rec = new GlideRecord('incident');
rec.query();
while (rec.next()) {
  gs.print(rec.number + ' exists');
}
```

_next

public boolean _next()

Provides the same functionality as `next`, intended to be used in cases where the GlideRecord has a column named **next.**

Moves to the next record in the GlideRecord.

Returns:

`boolean` - false if there are no more records in the query set .

Example:

```
var rec = new GlideRecord('sys_template');
rec.query();
while (rec._next()) {
  gs.print(rec.number + ' exists');
}
```

operation

public String operation()

Retrieves the current operation being performed, such as insert, update, delete, etc.

Returns:

`String` - the current operation.

orderBy

public void orderBy(`String name`)

Specifies an orderBy column (this may be called more than once to order by multiple columns).

Parameters:

`name` - a column name used to order the recordset.

Example:

```
function UpdateProjectWBS(project) {
  var count = 0;
  var child = new GlideRecord('pm_project_task');
  child.addQuery('parent', project.sys_id);
  child.orderBy('order');
  child.orderBy('number');
  child.query();
  var len = child.getRowCount().toString().length;
  var seq = 0;
  while (child.next()) {
    count += UpdateProjectTaskWBS(child, 1, ++seq, len, '');
  }
  gs.addInfoMessage(count + ' Project Tasks updated');
}
```

orderByDesc

public void orderByDesc(`String name`)

Specifies a decending orderBy column.

Parameters:

`name` - a column name used to order the GlideRecord.

query

public void query(`Object field, Object value`)

Runs the query against the table based on the filters specified by addQuery and addEncodedQuery. This will query the GlideRecord table as well as any references of the table. One argument adds a query string. Usually this is performed without arguments, but a name/value pair can be specified.

Parameters:

`name` - a column name to query on.

`value` - a value to query for.

Example:

```
var rec = new GlideRecord('incident');
rec.query();
while (rec.next()) {
 gs.print(rec.number + ' exists');
}
```

queryNoDomain

public void query(`Object field, Object value`)

Used in domain separated instances. Similar to #query, runs the query against the table based on the filters specified by addQuery and addEncodedQuery, but ignores domains. This will query the GlideRecord table as well as any references of the table. One argument adds a query string. Usually this is performed without arguments, but a name/value pair can be specified.

Parameters:

`name` - a column name to query on.

`value` - a value to query for.

Example:

```
var rec = new GlideRecord('incident');
rec.queryNoDomain();
while (rec.next()) {
 gs.print(rec.number + ' exists');
}
```

_query

public void _query(`Object field, Object value`)

Identical to `query();`, intended to be used on tables where there is a column named **'query**, which would interfere with using query();.

Runs the query against the table based on the filters specified by addQuery and addEncodedQuery. This will query the GlideRecord table as well as any references of the table. One argument adds a query string. Usually this is performed without arguments, but a name/value pair can be specified.

Parameters:

`name` - a column name to query on.

`value` - a value to query for.

Example:

```
var rec = new GlideRecord('sys_app_module');
rec._query();
while (rec.next()) {
 gs.print(rec.number + ' exists');
}
```

---

restoreLocation

public void restoreLocation(`boolean b`)

Set the current record to be the record that was saved with saveLocation(). If saveLocation has not been called yet, the current record is set to be the first record of the GlideRecord.

---

saveLocation

public void saveLocation()

Save the current row number so that we can get back to this location using restoreLocation().

Get Methods

get

public boolean get(`Object name, Object value`)

Defines a GlideRecord based on the specified expression of 'name = value'. If value is not specified, then the expression used is 'sys_id = name'. This method is expected to be used to query for single records, so a 'next' operation on the recordset is performed by this method before returning.

Parameters:

`name` - column name.

`value` - (optional) value to match.

Returns:

`name` - true if a matching record(s) was found or false if no matches found.

Example:

```
function kbWriteComment(id, comments) {
 var fb = new GlideRecord('kb_feedback');
 fb.initialize();
 fb.article = id;
 fb.comments = comments;
 fb.insert();
}

function kbGetText(id) {
 var gr = new GlideRecord('kb_knowledge');
 if (gr.get(id)) {
 gr.u_times_used = gr.u_times_used + 1;
 gr.u_last_used = gs.nowDateTime();
 gr.update();
 // return "Knowledge article " + gr.number + ":\n" +
 // "[code]" + gr.text + "[/code]";
 return gr.number;
 }
```

getAttribute

Gets the attributes on the field in question from the dictionary.

public String getAttribute(`String`)

Retrieves the attributes on the field in question from the dictionary.

Parameters:

`String` - the column name as a string to get the attributes from.

Returns:

`String` - the attribute values as a string.

Example:

```
doit();
function doit() {
  var gr = new GlideRecord('sys_user');
  gr.query("user_name","admin");
  if (gr.next()) {
    gs.print("we got one");
    gs.print(gr.location.getAttribute("tree_picker"));
  }
}
```

getClassDisplayValue

public String getClassDisplayValue()

Retrieves the label for the table.

Returns:

`String` - label that describes the table.

getDisplayValue

public String getDisplayValue()

Retrieves the display value for the current record.

Returns:

`String` - a string display value for the current record.

Example:

```
// list will contain a series of display values separated by a comma
  // array will be a javascript array of display values
  var list = current.watch_list.getDisplayValue();
  var array = list.split(",");
  for (var i=0; i < array.length; i++) {
     gs.print("Display value is: " + array[i]);
  }
```

getED

public ElementDescriptor getED()

Retrieves the element's descriptor.

Returns:

`ElementDescriptor` - the current element's descriptor.

Example:

```
var totalCritical  = 0;
var filledCritical = 0;
var fields         = current.getFields();

gs.print(fields);

for (var num = 0; num < fields.size(); num++) {

      gs.print("RUNNING ARRAY VALUE " + num);

      var ed = fields.get(num).getED();

      if(ed.hasAttribute("tiaa_critical")) {
            gs.print("CRITICAL FIELD FOUND");
            totalCritical ++;

            if (!fields.get(num).isNil()) {
                  filledCritical ++;
            }
      }

}

var answer = 0;
gs.print("TOTAL - " + totalCritical);
gs.print("FILLED - " + filledCritical);

if (filledCritical > 0 && totalCritical > 0) {

      var pcnt = (filledCritical/totalCritical)*100;
      answer = pcnt.toFixed(2);;

}

answer;
```

getElement

public GlideElement getElement(`String columnName`)

Retrieves the GlideElement for a specified field.

Parameters:

`columnName` - a column name to get the element from.

Returns:

`GlideElement` - a GlideElement.

getEncodedQuery

public String getEncodedQuery(`boolean b`)

Retrieves the encoded query as a string.

Parameters:

`String` - Gets the encoded query as a string.

getEscapedDisplayValue

public String getEscapedDisplayValue()

Retrieves the field value for the display field of the current record and escape it for use in Jelly scripts.

Returns:

`String` - escaped value of display column.

getFields

public ArrayList getFields()

Retrieves a Java ArrayList of fields in the current record.

Parameters:

`ArrayList` - a Java ArrayList of fields in the current record.

Example:

```
// This can be run in "Scripts - Background" for demonstration purposes

// Get a single incident record
var grINC = new GlideRecord('incident');
grINC.query();
grINC.next();
gs.print('Using ' + grINC.getValue('number'));
gs.print('');

// getFields() returns a Java ArrayList
var fields = grINC.getFields();

// Enumerate GlideElements in the GlideRecord object that have values
gs.print('Enumerating over all fields with values:');
for (var i = 0; i < fields.size(); i++) {
  var glideElement = fields.get(i);
  if (glideElement.hasValue()) {
    gs.print('  ' + glideElement.getName() + '\t' + glideElement);
  }
}
gs.print('');

// Get a specific GlideElement: number
gs.print('Getting the number field:');
for (var i = 0; i < fields.size(); i++) {
  var glideElement = fields.get(i);
  if (glideElement.hasValue() && glideElement.getName() == 'number') {
    gs.print('  ' + glideElement.getName() + '\t' + glideElement);
  }
}
```

getLabel

public String getLabel()

Retrieves the appropriate label for a field.

Returns:

`String` - the label of the field as a string.

Example:

```
template.print("Summary of Requested items:\n");
var gr = new GlideRecord("sc_req_item");
gr.addQuery("request", current.sysapproval);
gr.query();
while(gr.next()) {
    var nicePrice = gr.price.toString();
    if (nicePrice != '') {
        nicePrice = parseFloat(nicePrice);
        nicePrice = nicePrice.toFixed(2);
    }
    template.print(gr.number + ":  " + gr.quantity + " X " +
gr.cat_item.getDisplayValue() + " at $" + nicePrice + " each \n");
    template.print("    Options:\n");
    for (key in gr.variables) {
      var v = gr.variables[key];
      if(v.getGlideObject().getQuestion().getLabel() != '') {
         template.space(4);
         template.print('      ' +
v.getGlideObject().getQuestion().getLabel() + " = " +
v.getDisplayValue() + "\n");
      }
    }
}
```

getLink

public String getLink(`boolean noStack`)

Retrieves the link for the current record.

Parameters:

`noStack` - if true, the link generated will not append **&sysparm_stack=[tablename]_list.do?sysparm_query=active=true** to the end of the URL; if false, the link will. Leaving the argument empty will default to false.

Returns:

`String` - a URL.

Example:

```
  //Check for attachments and add link if there are any
  var attachment_link = '';
  var rec = new GlideRecord('sc_req_item');
  rec.addQuery('sys_id', current.request_item);
  rec.query();
  if(rec.next()){
    if(rec.hasAttachments()){
      attachment_link = gs.getProperty('glide.servlet.uri') +
rec.getLink();
    }
  }
```

getLocation

public int getLocation(`boolean b`)

Retrieves the current row number.

Returns:

`int` - Returns the row number of the current record.

getPlural

public string getPlural()

Retrieves the plural label of the GlideRecord table.

Returns:

`string` - the plural label of the GlideRecord's table.

For example, if the table name is "Change Request," this method would return "Change Requests."

getRecordClassName

public String getRecordClassName()

Retrieves the class name for the current record.

Parameters:

`String` - class or table name.

Example:

```
function TaskAssignmentFilter() {
  var classname = current.getRecordClassName();
  var filter = "type=null";
  if (classname == "incident" && current.category == "database") {
    filter = GetGroupFilter("database");
  }
  else {
    // append exclusion for 'catalog' to the filter
    var cat = new GlideRecord("sys_user_group_type");
    cat.addQuery("name", "catalog");
    cat.query();
    if (cat.next()) {
      filter += "^ORtype!=" + cat.sys_id;
    }
  }
  gs.log("TaskAssignmentFilter: " + filter);
  return filter;
}
```

getRelatedLists

public HashMap getRelatedLists(`boolean b`)

Retrieves a list of names and display values of tables that **refer to** the current record.

Returns:

`HashMap` - hashmap with names and display values of related tables.

---

getRelatedTables

public HashMap getRelatedTables(`boolean b`)

Retrieves a list of names and display values of tables that are**referred to by** the current record.

Parameters:

`HashMap` - hashmap with names and display values of related tables.

---

getRowCount

public int getRowCount()

Retrieves the number of rows in the GlideRecord.

Returns:

`int` - an integer count of rows.

Example:

```
var gr = new GlideRecord('incident')
gr.query();
gs.info("Records in incident table: " + gr.getRowCount());
```

---

getRowNumber

public int getRowNumber()

Retrieves the row number set by saveLocation() or setLocation(). To get the current row, use getLocation() instead.

Returns:

`int` - the saved row number.

---

getTableName

public String getTableName()

Retrieves the table name associated with this GlideRecord.

Returns:

`String` - table name.

Example:

```
gs.log('Table: ' + current.getTableName());
gs.log('Parent: ' + current.parent.sys_id);
var item = new GlideRecord('sc_req_item');
item.addQuery('sys_id', current.parent.sys_id);
item.query();
if(item.next()){
  for(var variable in item.variable_pool) { gs.log(variable);
  var answer = eval ("item.variable_pool." + variable +
".getDisplayValue()");
gs.log(answer);}
}
```

getValue

public String getValue(`String fieldName`)

Retrieves the string value of an underlying element in a field.

Parameters:

`fieldName` - the name of the field from which to get the value.

Returns:

`String` - the string value of the underlying element. Returns null if the field is empty, or the field does not exist.

Set Methods

autoSysFields

public void autoSysFields(`boolean e`)

Note: *This is not available for scoped apps, starting with the Fuji release. See the Scoped GlideRecord API Reference for a list of what APIs are available for scoped apps.*

Enables or disables the update to the fields sys_updated_by, sys_updated_on, sys_mod_count, sys_created_by, and sys_created_on. This is often used for manually updating field values on a record while leaving historical information unchanged.

Note: Use caution if you use this method. When you use this method the **sys_mod_count** field will not be incremented, and other **sys_** fields will not be updated. This can break functionality including, but not limited to, the Activity Formatter, History Sets, and Metrics.

Parameters:

`e` - Boolean variable that if false disables updates to sys_updated_by, sys_updated_on, sys_mod_count, sys_created_by, and sys_created_on.

Example:

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

// Change all Open(1) incidents to Active(2)

inc.addQuery('state', 1);
inc.query();

while (inc.next()) {
  inc.autoSysFields(false);  // Do not update sys_updated_by,
sys_updated_on, sys_mod_count, sys_created_by, and sys_created_on
  inc.setWorkflow(false);    // Do not run any other business rules
  inc.setValue('state', 2);
  inc.update();
}
```

---

setAbortAction

public void setAbortAction(`boolean b`)

Sets a flag to indicate if the next database action (insert, update, delete) is to be aborted.

Parameters:

`b` - true to abort next action or false if action is to be allowed.

Example:

```
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);
  }
}
```

---

setDisplayValue

public void setDisplayValue(`String name, Object value`)

Sets the field name to the specified display value.

- For a reference field this is the display value for the table.
- For a date/time this is the time in the caller's current timezone.

Parameters:

`name` - String value of the field name.

`value` - Display value to set for the field.

Example:

```
var gr = new GlideRecord('incident');
gr.get('46f09e75a9fe198100f4ffd8d366d17b');
gr.setDisplayValue('opened_at','2011-02-13 4:30:00');
gr.update();
```

setForceUpdate

public void setForceUpdate(`boolean force`)

Updates the record even if fields have not been changed.

Parameters:

`force` - true to update even if fields have not changed, otherwise false.

setLimit

public void setLimit(`int`)

Sets the limit for how many records are in the GlideRecord.

Parameters:

`int` - Integer limit for records to fetch.

Example:

```
var gr = new GlideRecord('incident');
gr.orderByDesc('sys_created_on');
gr.setLimit(10);
gr.query();
```

setLocation

public void setLocation(`int rowNumber`)

Sets the current row location.

Parameters:

`rowNumber` - integer that is the number of the row to set as the current row.

setNewGuid

public void setNewGuid()

Generates a new GUID and sets it as the unique id for the current record. This function applies only to new records. The GUID for an existing record cannot be changed.

Example:

```
var tsk_id = task.setNewGuid();

task.description = "Request: " + current.request.number;
task.description = task.description + "\n" + "Requested by: " +
current.request.u_requested_by.name;
task.description = task.description + "\n" + "Requested for: " +
current.request.u_requested_for.name;
task.description = task.description + "\n" + "Item: " +
current.cat_item.name;

var gr = new GlideRecord ('task_rel_task');
//link the incident to the request (may need to review if it needs to
be the item)
gr.parent = current.request;
gr.child = tsk_id;
gr.insert();
```

setNewGuidValue

public void setNewGuidValue(`String guid`)

Generates a new GUID and sets it as the unique id for the current record, when inserting a new record.

Parameters:

`guid` - a string value for the new GUID.

setQueryReferences

public void setQueryReferences(`boolean queryReferences`)

Enables or disables using the reference field's display name when querying a reference field.

Parameters:

`queryReferences` - Boolean variable. If set to true, will generate a string of display names; if false, will generate a string of sys_ids.

setUseEngines

public void setUseEngines(`boolean e`)

Disable or enable the running of any engines (approval rules / assignment rules).

Parameters:

`e` - if true enables engines, and if false disables engines.

setValue

public void setValue(`String name, Object value`)

Sets the specified field to the specified value. Normally a script would do a direct assignment, for example, gr.category = value. However, if in a script the element name is a variable, then `gr.setValue(elementName, value)` can be used. When setting a value, ensure the data type of the field matches the data type of the value you enter. This method cannot be used on journal fields.

If the value parameter is null, the record is not updated, and an error is not thrown.

Parameters:

`name` - a field name.

`value` - an object value.

---

setWorkflow

public void setWorkflow(`boolean e`)

Enables or disables the running of business rules that might normally be triggered by subsequent actions. If the `e` parameter is set to false, an insert/update will not be audited. Auditing only happens when the parameter is set to true for a GlideRecord operation.

Parameters:

`e` - Boolean variable that if true (default) enables business rules, and if false to disables them.

Example:

```
doit('name1','name2');

function doit(username1,username2) {

  var usr1 = new GlideRecord('sys_user');
  var usr2 = new GlideRecord('sys_user');
  var num = 0;

  if (usr1.get('user_name',username1) &&
usr2.get('user_name',username2)) {
    var ref;
    var dict = new GlideRecord('sys_dictionary');
    dict.addQuery('reference','sys_user');
    dict.addQuery('internal_type','reference');
    dict.query();
    while (dict.next()) {
      num = 0;
      ref = new GlideRecord(dict.name.toString());
      ref.addQuery(dict.element,usr1.sys_id);
      ref.query();
      while (ref.next()) {
        ref.setValue(dict.element.toString(),usr2.sys_id);
        ref.setWorkflow(false);
        ref.update();
        num++;
      }
      if (num > 0) {
        gs.print(dict.element + ' changed from ' + usr1.user_name +
          ' to ' + usr2.user_name + ' in ' + num + ' ' + dict.name + '
records');
      }
    }
  }
}
```

Note:setWorkflow() is ignored when subsequently using either deleteProblem() or deleteMultiple() to cascade delete.

Update Methods

applyTemplate

public void applyTemplate(`String template`)

Apply a template record (from **sys_template**) to the current record. If the specified template is not found, no action is taken.

Parameters:

`template` - Takes a name of a template on the **sys_template** table.

Example:

```
var rec1 = new GlideRecord("incident");
rec1.initialize();
rec1.applyTemplate("my_incident_template");

//...possibly more code here...
rec1.insert();
```

update

public string update(`Object reason`)

Updates the GlideRecord with any changes that have been made. If the record does not already exist, it is inserted.

Parameters:

`reason` - Takes a string designating the reason for the update if necessary. The reason will be displayed in the audit record.

Returns:

`String` - unique id of new or updated record.

Example:

```
  var gr = new GlideRecord('task_ci');
gr.addQuery();
gr.query();
var count = gr.getRowCount();
if (count > 0) {
   var allocation = parseInt(10000 / count) / 100;
   while (gr.next()) {
      gr.u_allocation = allocation;
      gr.update();
   }
}
```

updateWithReferences

public string updateWithReferences(`Object reason`)

Updates a record and also inserts or updates any related records with the information provided.

Returns:

`string` - unique ID for the record being updated.

Parameters:

`reason` - takes a string designating the reason for the update if necessary. The reason will be displayed in the audit record.

Example

- if processing a incident where the Caller ID is set to reference sys_user record 'David Loo,' then the following code would update David Loo's user record.
- if processing a incident where there is no Caller ID specified, then the following code would create a new sys_user record with the provided information (first_name, last_name) and set the Caller ID value to the newly created sys_user record.

```
var inc = new GlideRecord('incident');
inc.get(inc_sys_id);  // Looking up an existing incident record where
'inc_sys_id' represents the sys_id of a incident record
inc.caller_id.first_name = 'John';
inc.caller_id.last_name = 'Doe';
inc.updateWithReferences();
}
```

Insert Methods

initialize

Creates an empty record suitable for population before an insert.

public void initialize()

Example:

```
var gr = new GlideRecord('to_do');
gr.initialize();
gr.name = 'first to do item';
gr.description = 'learn about GlideRecord';
gr.insert();
```

insert

public string insert()

Inserts a new record using the field values that have been set for the current record.

Returns:

`string` - unique id of the inserted record or null if record is not inserted.

Example:

```
var gr = new GlideRecord('to_do');
gr.initialize();
gr.name = 'first to do item';
gr.description = 'learn about GlideRecord';
gr.insert();
```

insertWithReferences

public string insertWithReferences()

Inserts a new record and also inserts or updates any related records with the information provided.

Returns:

`string` - unique ID for the record being inserted.

Example

If a reference value is not specified (as below), then a new user record will be created with the provided first_name, last_name, and the caller_id value is set to this newly created sys_user record.

The result is a new sys_user record with the provided first_name, last_name and a new incident record with the provided short_description and caller_id.

```
var inc = new GlideRecord(‘incident’);
inc.initialize();
inc.short_description = 'New incident 1';
inc.caller_id.first_name = 'John';
inc.caller_id.last_name = 'Doe';
inc.insertWithReferences();
}
```

Example

If a caller_id value is specified, then that caller_id is updated with the provided first_name, last_name.

The result is a newly created incident record with values set for short_description and caller_id.

```
var inc = new GlideRecord(‘incident’);
inc.initialize();
inc.short_description = 'New incident 1';
inc.caller_id.setDisplayValue('David Loo');
inc.caller_id.first_name = 'John';
inc.caller_id.last_name = 'Doe';
inc.insertWithReferences();
}
```

newRecord

public void newRecord()

Creates a GlideRecord, set the default values for the fields and assign a unique id to the record. See Also: initialize().

Delete Methods

deleteMultiple

public void deleteMultiple()

Deletes multiple records according to the current "where" clause. Does not delete attachments.

Example:

```
function nukeCart() {
    var cart = getCart();
    var id = cart.sys_id;
    var kids = new GlideRecord('sc_cart_item');
    kids.addQuery('cart', cart.sys_id);
    kids.deleteMultiple();
```

Note: Dot-walking is not supported for this method. When deploying the deleteMultiple() function on referenced tables, all the records in the table will be deleted. Also, when using deleteRecord() to cascade delete, prior calls to setWorkflow() on the same GlideRecord object are ignored.

deleteRecord

public boolean deleteRecord()

Deletes a single record.

Parameters:

`boolean` - true if record was deleted or false if none found to delete.

Example:

```
var gr = new GlideRecord('incident')
gr.addQuery('sys_id','99ebb4156fa831005be8883e6b3ee4b9'); //to delete
one record
gr.query();
gr.next();
gr.deleteRecord();
}
```

Note: When using deleteMultiple() to cascade delete, prior calls to setWorkflow() on the same GlideRecord object are ignored.

References

[1] https://docs.servicenow.com/bundle/jakarta-servicenow-platform/page/script/glide-server-apis/concept/c_GlideRecord.html

GlideSystem

Note: *This article applies to Fuji and earlier releases. For more current information, see GlideSystem* [1] *at* http://docs.servicenow.com **The ServiceNow Wiki is no longer being updated. Visit** http://docs.servicenow.com **for the latest product documentation.**

Overview

The GlideSystem (referred to by the variable name 'gs' in Business Rules) provides a number of convenient methods to get information about the system, the current logged in user, etc. For example, the method addInfoMessage() permits communication with the user.

```
gs.addInfoMessage('Email address added for notification');
```

Many of the GlideSystem methods facilitate the easy inclusion of dates in query ranges and are most often used in filters and reporting.

Note: *The global Glide APIs are not available in scoped scripts. There are scoped Glide APIs for use in scoped scripts. The scoped Glide APIs provide a subset of the global Glide API methods. For information on scoped applications see Application Scope, scripting in scoped applications, and the list of scoped APIs.*

The Method Detail is in three sections:

- General functions
- Date and Time functions
- User Session functions

Method Summary

General

Method Summary	Description
eventQueue(String, Object, String, String, String)	Queues an event for the event manager.
getCurrentScopeName()	Gets the name of the current scope.
getDisplayColumn(String)	Gets the display column for the table.
getDisplayValueFor(String, String, String)	Gets the display value for a given field.
getEscapedProperty(String, Object)	Gets the property and escapes it for XML parsing.
getMessage(String, Object)	Retrieves translated messages to display in the UI. If the specified string exists in the database for the current language, then the translated message is returned. If the specified string does not exist for the current language, then the English version of the string is returned. If the string does not exist at all in the database, then the ID itself is returned.

getMessageS(String, Object)	Retrieves translated messages to display in the UI and escapes all ticks ('). If the specified string exists in the database for the current language, then the translated message is returned. If the specified string does not exist for the current language, then the English version of the string is returned. If the string does not exist at all in the database, then the ID itself is returned.
getNodeValue(object, Integer)	Gets the node value for specified index.
getNodeName(Object, Integer)	Returns the node name for specified index.
getProperty(String, Object)	Gets the value of a Glide property.
getScriptError(String)	Returns the script error found in the specified script, if there is one. The script is not executed by this function, only checked for syntax errors.
getStyle(String, String, String)	Returns the style defined for the table, field and value.
getXMLText (String, String)	Gets the xml text for the first node in the xml string that matches the path query.
getXMLNodeList(String)	Constructs an Array of all the nodes and values in an XML document.
log(String message, String source)	Logs a message to the system log and saves it to the syslog table.
logError(String message, String source)	Logs an error to the system log and saves it to the syslog table.
logWarning(String message, String source)	Logs a warning to the system log and saves it to the syslog table.
nil(Object)	Queries an object and returns **true** if the object is null or contains an empty string.
print(String)	Writes a message to the system log. This method does not write the message to the syslog table unless debug has been activated.
tableExists(String)	Determines if a database table exists.
workflowFlush(Object)	Deletes all existing workflow operations for the specified GlideRecord.

Date and Time Functions

Method Summary	Description
beginningOfLastMonth()	Gets the date and time for the beginning of last month in GMT.
beginningOfLastWeek()	Gets the date and time for the beginning of last week in GMT.
beginningOfNextWeek()	Gets the date and time for the beginning of next week in GMT.
beginningOfNextMonth()	Gets the date and time for the beginning of next month in GMT.
beginningOfNextYear()	Gets the date and time for the beginning of next year in GMT.
beginningOfThisMonth()	Gets the date and time for the beginning of this month in GMT.
beginningOfThisQuarter()	Gets the date and time for the beginning of this quarter in GMT.
beginningOfThisWeek()	Gets the date and time for the beginning of this week in GMT.
beginningOfThisYear()	Gets the date and time for the beginning of this week in GMT.
beginningOfToday()	Gets the date and time for the beginning of today in GMT.
beginningOfYesterday()	Gets the date and time for the beginning of yesterday in GMT.
calDateDiff(String, String, boolean)	Calculate the difference between two dates using the default calendar. **Note:** Calendars are now legacy. If Schedules are being used, see Calculate Duration Given a Schedule.
dateDiff(String, String, boolean)	Calculates the difference between two dates. The parameters must be in the user/system date time format.
dateGenerate(String, String)	Generates a date and time for the specified date in GMT.
daysAgo(int)	Gets a date and time for a certain number of days ago. The result is expressed in GMT.
daysAgoEnd(int)	Gets a date and time for end of the day a certain number of days ago.The result is expressed in GMT.

daysAgoStart(int)	Gets a date and time for beginning of the day a certain number of days ago. The result is expressed in GMT.
endOfLastMonth()	Gets the date and time for the end of last month in GMT.
endOfLastWeek()	Gets the date and time for the end of last week in GMT, in the format **yyyy-mm-dd hh:mm:ss.**
endOfLastYear()	Gets the date and time for the end of last year in GMT.
endOfNextMonth()	Gets the date and time for the end of next month in GMT.
endOfNextWeek()	Gets the date and time for the end of next week in GMT.
endOfNextYear()	Gets the date and time for the end of next year in GMT.
endOfThisMonth()	Gets the date and time for the end of this month in GMT.
endOfThisQuarter()	Gets the date and time for the end of this quarter in GMT.
endOfThisWeek()	Gets the date and time for the beginning of this week in GMT.
endOfThisYear()	Gets the date and time for the end of this year in GMT.
endOfToday()	Gets the date and time for the end of today in GMT.
endOfYesterday()	Gets the date and time for the end of yesterday in GMT.
hoursAgo(int)	Gets a date and time for a certain number of hours ago.The result is expressed in GMT.
hoursAgoEnd(int)	Gets a date and time for the end of the hour a certain number of hours ago.The result is expressed in GMT.
hoursAgoStart(int)	Gets a date and time for the start of the hour a certain number of hours ago.The result is expressed in GMT.
lastWeek()	Date and time one week ago in GMT.
minutesAgo(int)	Gets a date and time for a certain number of minutes ago.The result is expressed in GMT.
minutesAgoEnd(int)	Gets a date and time for the end of the minute a certain number of minutes ago.The result is expressed in GMT.
minutesAgoStart(int)	Gets a date and time for a certain number of minutes ago. The result is expressed in GMT.
monthsAgo(int)	Gets a date and time for a certain number of months ago.The result is expressed in GMT.
monthsAgoEnd(int)	Gets a date and time for the last day of the month a certain number of months ago.The result is expressed in GMT.
monthsAgoStart(int)	Gets a date and time for the first day of the month a certain number of months ago.The result is expressed in GMT.
now()	Gets the current date using GMT date time.
nowNoTZ()	Gets the current GMT date time.
nowDateTime()	Gets the current date and time in the user's time zone.
quartersAgo(int)	Gets a date and time for a certain number of quarters ago. The result is expressed in GMT.
quartersAgoEnd(int)	Gets a date and time for the last day of the quarter a certain number of quarters ago. The result is expressed in GMT.
quartersAgoStart(int)	Gets a date and time for the first day of the quarter a certain number of quarters ago. The result is expressed in GMT.
yearsAgo(int)	Gets a date and time for a certain number of years ago.The result is expressed in GMT.
yesterday()	Gets yesterday's time. The result is expressed in GMT.
isFirstDayOfMonth(Object)	Checks whether the date is the first day of the month.
isFirstDayOfWeek(Object)	Checks whether the date is the first day of the week. This uses the ISO standard of Monday being the first day of the week.
isFirstDayOfYear(Object)	Checks whether the date is the first day of the year
isLastDayOfMonth(Object)	Checks whether the date is the last day of the month.
isLastDayOfWeek(Object)	Checks whether the date is the last day of the week.

isLastDayOfYear(Object)	Checks whether the date is the last day of the year.

User Session Functions

Method Summary	**Description**
addErrorMessage(Object)	Adds an error message for the current session. Session error messages are shown at the top of the form. Use getErrorMessages() to retrieve the list of error messages that are being shown.
addInfoMessage(Object)	Adds an info message for the current session. Session info messages are shown at the top of the form below any error messages. Use getInfoMessages() to retrieve the list of info messages being shown.
addMessage(String, Object)	Adds a message for the current session. Can be called using getMessages(String).
flushMessages()	Clears session messages saved using addErrorMessage(Object) or addInfoMessage(Object). Session messages are shown at the top of the form. In client side scripts use g_form.clearMessages() to clear all session messages.
getErrorMessages()	Gets the list of error messages for the session that were added by addErrorMessage(Object).
getImpersonatingUserDisplayName()	Returns the display name of the impersonating user.
getImpersonatingUserName()	Returns the name of the impersonating user or null if not impersonating.
getInfoMessages()	Gets the list of info messages for the session that were added via addInfoMessage(Object.
getMessages(String)	Gets the list of messages of the specified type for the session that were added via addMessage(String, Object).
getPreference(String, Object)	Gets a user preference.
getSession()	Returns a GlideSession object.
getSessionID()	Accesses the GlideSession Session ID.
getTrivialMessages()	Gets the list of error messages for the session that were added with the trivial flag.
getUser()	Returns a reference to the User object for the current user. More information is available here.
getUserDisplayName()	Returns the name field of the current user (e.g. John Smith, as opposed to smith).
getUserID()	Returns the sys_id of the current user.
getUserName()	Returns the username of the current user (for example, jsmith).
getUserNameByUserID(String)	Gets the username based on a user ID.
hasRole(String)	Determines if the current user has the specified role.
hasRoleInGroup(Object, Object)	Determines if the current user has the specified role within a specified group.
isInteractive()	Checks if the current session is interactive.
isLoggedIn()	Determines if the current user is currently logged in.
setRedirect(Object)	Sets the redirect URI for this transaction. This determines the next page the user will see.
setReturn(Object)	Sets the return URI for this transaction. This determines what page the user will be directed to when they return from the next form.
userID()	Returns the sys_id of the user associated with this session. A shortcut for the more proper getUserID().

General

eventQueue(String, Object, String, String, String)

Queues an event for the event manager.

Input Fields

Parameters:

- Name of the event being queued.
- A GlideRecord object, such as "current".
- An optional parameter, saved with the instance if specified.
- A second optional parameter, saved with the instance if specified.
- An event queue to add the event to.

Example

```
if (current.operation() != 'insert' && current.comments.changes()) {
    gs.eventQueue('incident.commented', current, gs.getUserID(),
gs.getUserName());
}
```

getCurrentScopeName()

Gets the name of the current scope.

Input Fields

Parameters:

- None

Example

```
gs.getCurrentScopeName();
```

getDisplayColumn(String)

Gets the display column for the table.

Input Fields

Parameters: name of table from which to get the display column name.

Output Fields

Returns: name of the display column for the table.

Example

```
// Return the sys_id value for a given table and its display value
function GetIDValue(table, displayValue) {
    var rec = new GlideRecord(table);
    var dn = gs.getDisplayColumn(table);
    if (rec.get(dn, displayValue))
        return rec.sys_id;
    else
        return null;
}
```

getDisplayValueFor(String, String, String)

Gets the display value for a specified field on a record.

Input Fields

Parameters:

- String - name of table for the record
- String - the sysid for the record to get the display value from
- String - the field to get a display value from

Output Fields

Returns: name of the display value for the field's value.

getEscapedProperty(String, Object)

Gets the property and escapes it for XML parsing.

Input Fields

Parameters:

- String key for the property whose value should be returned
- Alternate object to return if the property is not found.

Output Fields

Returns: the property as a string, or the alternate object specified above.

getMessage(String, Object)

Retrieves translated messages to display in the UI. If the specified string exists in the database for the current language, then the translated message is returned. If the specified string does not exist for the current language, then the English version of the string is returned. If the string does not exist at all in the database, then the ID itself is returned.

Note: if the UI message has a tick ('), there may be issues with the message in the script; to escape the ticks ('), use getMessageS(String, Object).

Input Fields

Parameters:

- ID of message
- (Optional) A list of strings or other values defined by java.text.MessageFormat [2], which allows you to produce language-neutral messages for display to users.

Output Fields

Returns: the UI message as a string.

Examples

```
var my_message = '${gs.getMessage("This is a message.")}';
alert(my_message);
```

Using the optional parameter as in:

```
gs.getMessage('"{0}" is not a Client Callable Script Include','BAR');
```

Returns: "BAR" is not a Client Callable Script Include

getMessageS(String, Object)

Retrieves translated messages to display in the UI and escapes all ticks ('). If the specified string exists in the database for the current language, then the translated message is returned. If the specified string does not exist for the current language, then the English version of the string is returned. If the string does not exist at all in the database, then the ID itself is returned. Useful if you are inserting into a JavaScript expression from Jelly.

For instance, the following code snippet will fail if the returned snippet has a tick ('):

```
var my_message = '${gs.getMessage("I love France")}';
alert(my_message);
```

The solution is to use a snippet like the following:

```
var my_message = '${gs.getMessageS("I love France")}';
alert(my_message);
```

Input Fields

Parameters:

- ID of message
- Optional message arguments.

Output Fields

Returns: the message as a string, with the ticks escaped.

Example

```
var my_message = '${gs.getMessageS("I love France")}';
alert(my_message);
```

getProperty(String, Object)

Gets the value of a Glide property. If the property is not found, return an alternate value. Use getEscapedProperty(String, Object) to escape the property.

Input Fields

Parameters:

- String key for the property whose value should be returned.
- Alternate object to return if the property is not found.

Output Fields

Returns: (String) the value of the Glide property, or the alternate object defined above.

Example

```
  //Check for attachments and add link if there are any
  var attachment_link = '';
  var rec = new GlideRecord('sc_req_item');
  rec.addQuery('sys_id', current.request_item);
  rec.query();
  if(rec.next()){
    if(rec.hasAttachments()){
      attachment_link = gs.getProperty('glide.servlet.uri') +
rec.getLink();
    }
  }
```

getScriptError(String)

Returns the script error found in the specified script, if there is one. The script is not executed by this function, only checked for syntax errors.

Input Fields

Parameters: string script to check for errors.

Output Fields

Returns: the script error message or, if none, then **null.**

getStyle(String, String, String)

Returns the style defined for the table, field and value.

Input Fields

Parameters:

- Table name
- Field name
- Field value

Output Fields

Returns the style as a string.

log(String message, String source)

Logs a message to the system log and saves it to the syslog table.

Input Fields

Parameters:

- **String message** - message to log, for the log's **Message** field.
- **String source** - (optional) the source of the message, for the log's **Source** field.

Example

```
var count = new GlideAggregate('incident');
count.addQuery('active', 'true');
count.addAggregate('COUNT', 'category');
count.query();
while (count.next()) {
   var category = count.category;
   var categoryCount = count.getAggregate('COUNT', 'category');
   gs.log("The are currently " + categoryCount + " incidents with a
category of " + category, "Incident Counter");
}
```

logError(String message, String source)

Logs an error to the system log and saves it to the syslog table.

Input Fields

Parameters:

- **String message** - message to log, for the log's **Message** field.
- **String source** - (optional) the source of the message, for the log's **Source** field.

logWarning(String message, String source)

Logs a warning to the system log and saves it to the syslog table.

Input Fields

Parameters:

- **String message** - message to log, for the log's **Message** field.
- **String source** - (optional) the source of the message, for the log's **Source** field.

nil(Object)

Queries an object and returns **true** if the object is null or contains an empty string.

Input Fields

Parameters: name of an object

Output Fields

Returns: true if null or empty string; otherwise, returns false

Example

```
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);
  }
}
```

print(String)

Writes a message to the system log. This method does not write the message to the syslog table unless debug has been activated.

Input Fields

Parameters: message to log

Example

```
var rec = new GlideRecord('incident');
rec.addQuery('active',false);
rec.query();
while (rec.next()) {
 gs.print('Inactive incident ' + rec.number + ' deleted');
 rec.deleteRecord();
}
```

tableExists(String)

Determines if a database table exists.

Input Fields

Parameters: name of table to check for existence

Output Fields

Returns: true if table exists or false is not found

workflowFlush(Object)

Deletes all existing workflow operations for the specified GlideRecord.

Input Fields

Parameters: the GlideRecord to flush the workflow for

Output Fields

Returns: void

Date/Time

beginningOfLastMonth()

Gets the date and time for the beginning of last month in GMT.

Output Fields

Returns: the GMT beginning of last month, in the format **yyyy-mm-dd hh:mm:ss.**

beginningOfLastWeek()

Gets the date and time for the beginning of last week in GMT.

Output Fields

Returns: the GMT beginning of last week, in the format **yyyy-mm-dd hh:mm:ss.**

beginningOfNextWeek()

Gets the date and time for the beginning of next week in GMT.

Output Fields

Returns: the GMT beginning of next week, in the format **yyyy-mm-dd hh:mm:ss.**

beginningOfNextMonth()

Gets the date and time for the beginning of next month in GMT.

Output Fields

Returns: the GMT beginning of next month, in the format **yyyy-mm-dd hh:mm:ss.**

beginningOfNextYear()

Gets the date and time for the beginning of next year in GMT.

Output Fields

Returns: the GMT beginning of next year, in the format **yyyy-mm-dd hh:mm:ss.**

beginningOfThisMonth()

Gets the date and time for the beginning of this month in GMT.

Output Fields

Returns: the GMT beginning of this month, in the format **yyyy-mm-dd hh:mm:ss.**

beginningOfThisQuarter()

Gets the date and time for the beginning of this quarter in GMT.

Output Fields

Returns: the GMT beginning of this quarter, in the format **yyyy-mm-dd hh:mm:ss.**

beginningOfThisWeek()

Gets the date and time for the beginning of this week in GMT.

Output Fields

Returns: the GMT beginning of this week, in the format **yyyy-mm-dd hh:mm:ss.**

beginningOfThisYear()

Gets the date and time for the beginning of this week in GMT.

Output Fields

Returns: the GMT beginning of this week, in the format **yyyy-mm-dd hh:mm:ss.**

beginningOfToday()

Gets the date and time for the beginning of today in GMT.

Output Fields

Returns: the GMT beginning of today, in the format **yyyy-mm-dd hh:mm:ss.**

beginningOfYesterday()

Gets the date and time for the beginning of yesterday in GMT.

Output Fields

Returns: the GMT beginning of yesterday, in the format **yyyy-mm-dd hh:mm:ss.**

calDateDiff(String, String, boolean)

Calculate the difference between two dates using the default calendar. **Note:** Calendars are now legacy. If Schedules are being used, see Calculate Duration Given a Schedule.

Input Fields

Parameters:

- **startDate** - a starting date to compare, in the current user's date format
- **endDate** - an ending date to compare, in the current user's date format
- **boolean numericValue** - if true, the return will be formatted in number of seconds; if false, the return will be formatted ddd hh:mm:ss.

Output Fields

Returns: if the numericValue boolean parameter is true, returns the difference between the two dates as an integer number of seconds; if false, returns the difference between the two dates in the format ddd hh:mm:ss.

dateDiff(String, String, boolean)

Calculates the difference between two dates. This method expects the earlier date as the first parameter and the later date as the second parameter; otherwise, the method returns the difference as a negative value. **Note:** Use getDisplayValue() to convert the strings to the expected format.

Input Fields

Parameters:

- **startDate** - a starting date to compare, in the current user's date format.
- **endDate** - an ending date to compare, in the current user's date format.
- **boolean bnumericValue** - true to return difference in number of seconds as a string, false to return difference in the format ddd hh:mm:ss.

Output Fields

Returns: if boolean bnumericValue is true, the difference in number of seconds; if false, the difference in the format ddd hh:mm:ss.

Example

For more examples, see Setting the Duration Field Value.

```
// Given two date/times as DateTime objects
// Set the values this way to ensure a consistent input time
var date1 = new GlideDateTime();
var date2 = new GlideDateTime();
date1.setDisplayValueInternal('2014-01-01 12:00:00');
date2.setDisplayValueInternal('2014-01-01 13:00:00');

// Determine the difference as number of seconds (returns a string)
// Use getDisplayValue() to convert the string to the format expected
by dateDiff()
var diffSeconds = gs.dateDiff(date1.getDisplayValue(),
date2.getDisplayValue(), true);

// JavaScript will coerce diffSeconds from a string to a number
// since diffSeconds is being compared to a number
var msg = (diffSeconds <= 0) ? ' is on or after ' : ' is before ';
gs.print(date1.getDisplayValue() + msg + date2.getDisplayValue())
```

dateGenerate(String, String)

Generates a date and time for the specified date in GMT.

Input Fields

Parameters:

- **Date** - format: yyyy-mm-dd
- **Range** - **start**, **end**, or a time in the 24 hour format **hh:mm:ss**.

Output Fields

Returns: a date and time in the format yyyy-mm-dd hh:mm:ss. If range is **start**, the returned value is yyyy-mm-dd 00:00:00; If range is **end** the return value is yyyy-mm-dd 23:59:59.

daysAgo(int)

Gets a date and time for a certain number of days ago.

Input Fields

Parameters: An integer number of days ago.

Output Fields

Returns: The (GMT) beginning of the days that was the specified number of days ago, in the format **yyyy-mm-dd hh:mm:ss.**

Example

```
function contractNoticeDue() {

    var gr = new GlideRecord("contract");

    gr.addQuery("u_contract_status", "Active");

    gr.query();

    while (gr.next()) {

        if ((gr.u_termination_date <= gs.daysAgo(-90)) && (gr.u_contract_duration == "Long")) {

            gr.u_contract_status = "In review";

        }

        else if ((gr.u_termination_date <= gs.daysAgo(-50)) && (gr.u_contract_duration == "Medium")) {

            gr.u_contract_status = "In review";

        }

        else if ((gr.u_termination_date <= gs.daysAgo(-10)) && (gr.u_contract_duration == "Short")) {

            gr.u_contract_status = "In review";

        }

    }

    gr.update();

}
```

daysAgoEnd(int)

Gets a date and time for end of the day a certain number of days ago.

Input Fields

Parameters: An integer number of days ago.

Output Fields

Returns: The (GMT) end of the day that was the specified number of days ago, in the format **yyyy-mm-dd hh:mm:ss.**

daysAgoStart(int)

Gets a date and time for beginning of the day a certain number of days ago.

Input Fields

Parameters: An integer number of days ago.

Output Fields

Returns: The (GMT) start of the day that was the specified number of days ago, in the format **yyyy-mm-dd hh:mm:ss.**

Example

```
var gr = new GlideRecord('sysapproval_approver');
gr.addQuery('state', 'requested');
gr.addQuery('sys_updated_on', '<', gs.daysAgoStart(5));
gr.query();
```

endOfLastMonth()

Gets the date and time for the end of last month in GMT.

Output Fields

Returns: the GMT end of last month, in the format **yyyy-mm-dd hh:mm:ss.**

endOfLastWeek()

Gets the date and time for the end of last week in GMT, in the format **yyyy-mm-dd hh:mm:ss.**

Output Fields

Returns: the GMT end of last week, in the format **yyyy-mm-dd hh:mm:ss.**

endOfLastYear()

Gets the date and time for the end of last year in GMT.

Output Fields

Returns: the GMT end of last year, in the format **yyyy-mm-dd hh:mm:ss.**

endOfNextMonth()

Gets the date and time for the end of next month in GMT.

Output Fields

Returns: the GMT end of next month, in the format **yyyy-mm-dd hh:mm:ss.**

endOfNextWeek()

Gets the date and time for the end of next week in GMT.

Output Fields

Returns: the GMT end of next week, in the format **yyyy-mm-dd hh:mm:ss.**

endOfNextYear()

Gets the date and time for the end of next year in GMT.

Output Fields

Returns: the GMT end of next year, in the format **yyyy-mm-dd hh:mm:ss.**

endOfThisMonth()

Gets the date and time for the end of this month in GMT.

Output Fields

Returns: the GMT end of this month, in the format (yyyy-mm-dd huh:mm:ss)

endOfThisQuarter()

Gets the date and time for the end of this quarter in GMT.

Output Fields

Returns: the GMT end of this quarter, in the format **yyyy-mm-dd hh:mm:ss.**

endOfThisWeek()

Gets the date and time for the beginning of this week in GMT.

Output Fields

Returns: the GMT beginning of this week, in the format **yyyy-mm-dd hh:mm:ss.**

endOfThisYear()

Gets the date and time for the end of this year in GMT.

Output Fields

Returns: the GMT end of this year, in the format **yyyy-mm-dd hh:mm:ss.**

endOfToday()

Gets the date and time for the end of today in GMT.

Output Fields

Returns: the GMT end of today, in the format **yyyy-mm-dd hh:mm:ss.**

endOfYesterday()

Gets the date and time for the end of yesterday in GMT.

Output Fields

Returns: the GMT end of yesterday, in the format (yyyy-mm-dd huh:mm:ss).

hoursAgo(int)

Gets a date and time for a certain number of hours ago.

Input Fields

Parameters: An integer number of hours ago.

Output Fields

Returns: The (GMT) time that was the specified number of hours ago,in the format **yyyy-mm-dd hh:mm:ss.**

Example

```
if (current.operation() == 'insert') {
 // If no due date was specified, calculate a default
 if (current.due_date == '') {

  if (current.urgency == '1') {
   // Set due date to 4 hours ahead of current time
   current.due_date = gs.hoursAgo(-4);
  }

  if (current.urgency == '2') {
   // Set due date to 2 days ahead of current time
   current.due_date = gs.daysAgo(-2);
  }

  if (current.urgency == '3') {
   // Set due date to 7 days ahead of current time
   current.due_date = gs.daysAgo(-7);
  }
 }
}
```

hoursAgoEnd(int)

Gets a date and time for the end of the hour a certain number of hours ago.

Input Fields

Parameters: An integer number of hours ago.

Output Fields

Returns: The (GMT) end of the hour that was the specified number of hours ago, in the format **yyyy-mm-dd hh:mm:ss.**

hoursAgoStart(int)

Gets a date and time for the start of the hour a certain number of hours ago.

Input Fields

Parameters: An integer number of hours ago.

Output Fields

Returns: The (GMT) start of the hour that was the specified number of hours ago, in the format **yyyy-mm-dd hh:mm:ss.**

lastWeek()

Date and time one week ago in GMT.

Output Fields

Returns: the date and time one week ago, in the format **yyyy-mm-dd hh:mm:ss.**

minutesAgo(int)

Gets a date and time for a certain number of minutes ago.

Input Fields

Parameters: An integer number of minutes ago.

Output Fields

Returns: The (GMT) time that was the specified number of minutes ago, in the format **yyyy-mm-dd hh:mm:ss.**

Example

```
//
// Check to see if the user has failed to login too many times
// when the limit is reached, lock the user out of the system
//
  //Check failed logins in the last 15 minutes
  var gr = new GlideRecord('sysevent');
  gr.addQuery('name', 'login.failed');
  gr.addQuery('parm1', event.parm1.toString());
  gr.addQuery('sys_created_on','>=', gs.minutesAgo(15));
  gr.query();
  var rowCount = gr.getRowCount();
  if(rowCount >= 5){
      var gr = new GlideRecord("sys_user");
      gr.addQuery("user_name", event.parm1.toString());
      gr.query();
      if (gr.next()) {
           gr.locked_out = true;
           gr.update();
           gs.log("User " + event.parm1 + " locked out due to too many
invalid login attempts");
      }
  }
```

minutesAgoEnd(int)

Gets a date and time for the end of the minute a certain number of minutes ago.

Input Fields

Parameters: An integer number of minutes ago.

Output Fields

Returns: The (GMT) end of the minute that was the specified number of minutes ago, in the format **yyyy-mm-dd hh:mm:ss.**

minutesAgoStart(int)

Gets a date and time for the start of the minute a certain number of minutes ago.

Input Fields

Parameters: An integer number of minutes ago.

Output Fields

Returns: The (GMT) start of the minute that was the specified number of minutes ago, in the format **yyyy-mm-dd hh:mm:ss.**

monthsAgo(int)

Gets a date and time for a certain number of months ago.

Input Fields

Parameters: An integer number of months ago.

Output Fields

Returns: A value (GMT) on the current (today's) date of the specified month, in the format **yyyy-mm-dd hh:mm:ss.**

monthsAgoEnd(int)

Gets a date and time for the last day of the month a certain number of months ago.

Input Fields

Parameters: An integer number of months ago.

Output Fields

Returns: The (GMT) end of the month that was the specified number of months ago, in the format **yyyy-mm-dd hh:mm:ss.**

monthsAgoStart(int)

Gets a date and time for the start of the minute a certain number of minutes ago.

Input Fields

Parameters: An integer number of minutes ago.

Output Fields

Returns: The (GMT) start of the minute that was the specified number of minutes ago, in the format **yyyy-mm-dd hh:mm:ss.**

now()

Gets the current date using GMT.

Output Fields

Returns: The current date in the user defined format, according to GMT.

Example

```
// When the user password changes then set the u_password_last_reset
field
// to now so we know when to force another update

var gr = new GlideRecord("sys_user");
if (gr.get(event.parm1.toString())) {
    // Do something based on the Password Changing
    gs.log("The user password changed so do something else...");
    gr.u_password_last_reset = gs.now();
    gr.update();
}
```

nowNoTZ()

Gets the current date and time in UTC format.

Output Fields

Returns: the current UTC date time.

Example

```
// When the user password changes then set the u_password_last_reset
field
// to now so we know when to force another update

var gr = new GlideRecord("sys_user");
if (gr.get(event.parm1.toString())) {
    // Do something based on the Password Changing
    gs.log("The user password changed so do something else...");
    gr.u_password_last_reset = gs.nowNoTZ();
    gr.update();
}
```

nowDateTime()

Gets the current date and time in the user-defined format.

Output Fields

Returns: The current date and time in the user-defined format.

Example

The following script sets the field **u_target_date** to the current date and time:

```
current.u_target_date = gs.nowDateTime();
```

After the script is run, the **u_target_date** field will hold the date and time of the moment the script is run, in the system format.

Example 2

When setting a variable in a workflow script to the current date and time, use the setDisplayValue method. The following script sets the workflow variable **end_date** to the current date and time:

```
current.variables.end_date.setDisplayValue(gs.nowDateTime());
```

After the script is run, the **end_date** field will hold the date and time of the moment the script is run, in the system format.

quartersAgo(int)

Gets a date and time for a certain number of quarters ago.

Input Fields

Parameters: An integer number of quarters ago.

Output Fields

Returns: The (GMT) beginning of the quarter that was the specified number of quarters ago, in the format **yyyy-mm-dd hh:mm:ss.**

quartersAgoEnd(int)

Gets a date and time for the last day of the quarter a certain number of quarters ago.

Input Fields

Parameters: An integer number of quarters ago.

Output Fields

Returns: The (GMT) end of the quarter that was the specified number of quarters ago, in the format **yyyy-mm-dd hh:mm:ss.**

quartersAgoStart(int)

Gets a date and time for the first day of the quarter a certain number of quarters ago.

Input Fields

Parameters: An integer number of quarters ago.

Output Fields

Returns: The (GMT) end of the month that was the specified number of quarters ago, in the format **yyyy-mm-dd hh:mm:ss.**

yearsAgo(int)

Gets a date and time for a certain number of years ago.

Input Fields

Parameters: An integer number of years ago.

Output Fields

Returns: The (GMT) beginning of the years that was the specified number of years ago, in the format **yyyy-mm-dd hh:mm:ss.**

yesterday()

Gets yesterday's time.

Output Fields

Returns: The (GMT) for 24 hours ago, in the format **yyyy-mm-dd hh:mm:ss.**

isFirstDayOfMonth(Object)

Checks whether the date is the first day of the month.

Input Fields

Parameters: date object.

Output Fields

Returns: true if date is the first day of the month, false if it is not.

isFirstDayOfWeek(Object)

Checks whether the date is the first day of the week. This uses the ISO standard of Monday being the first day of the week.

Input Fields

Parameters: date object.

Output Fields

Returns: true if date is the first day of the week, false if it is not.

isFirstDayOfYear(Object)

Checks whether the date is the first day of the year.

Input Fields

Parameters: date object.

Output Fields

Returns: true if date is the first day of the year, false if it is not.

isLastDayOfMonth(Object)

Checks whether the date is the last day of the month.

Input Fields

Parameters: date object.

Output Fields

Returns: true if date is the last day of the month, false if it is not. This uses the ISO standard of Sunday being the last day of the week.

isLastDayOfWeek(Object)

Checks whether the date is the last day of the week.

Input Fields

Parameters: date object.

Output Fields

Returns: true if date is the last day of the week, false if it is not.

isLastDayOfYear(Object)

Checks whether the date is the last day of the year.

Input Fields

Parameters: date object.

Output Fields

Returns: true if date is the last day of the year, false if it is not.

User Session

addErrorMessage(Object)

Adds an error message for the current session. Use getErrorMessages() to retrieve a list of error messages currently being shown.

Input Fields

Parameters: an error object.

Example

```
 gs.include("PrototypeServer");
  var ValidatePasswordStronger = Class.create();
  ValidatePasswordStronger.prototype = {
       process : function() {
          var user_password = request.getParameter("user_password");
          var min_len = 8;
          var rules = "Password must be at least " + min_len +
             " characters long and contain a digit, an uppercase
letter, and a lowercase letter.";
          if (user_password.length() < min_len) {
             gs.addErrorMessage("TOO SHORT: " + rules);
             return false;
          }
          var digit_pattern = new RegExp("[0-9]", "g");
          if (!digit_pattern.test(user_password)) {
             gs.addErrorMessage("DIGIT MISSING: " + rules);
```

```
                return false;
            }
            var upper_pattern = new RegExp("[A-Z]", "g");
            if (!upper_pattern.test(user_password)) {
                gs.addErrorMessage("UPPERCASE MISSING: " + rules);
                return false;
            }
            var lower_pattern = new RegExp("[a-z]", "g");
            if (!lower_pattern.test(user_password)) {
                gs.addErrorMessage("LOWERCASE MISSING: " + rules);
                return false;
            }
            return true; // password is OK
        }
    }
```

addInfoMessage(Object)

Adds an info message for the current session. Use getInfoMessages() to retrieve the list of info messages being shown. **Note**: This method is not supported for asynchronous business rules and cannot be used within transform scripts.

Input Fields

Parameters: an info message object.

Example

```
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);
  }
}
```

addMessage(String, Object)

Adds a message for the current session. Can be called using getMessages(String).

Input Fields

Parameters:

- String type of message
- Message to store

Example

```
gs.include("FormInfoHeader");
var fi = new FormInfoHeader();
var s = 'An incident ' + current.number + ' has been opened for your
request.<br/>';
s += 'The IT department will contact you when the password is reset or
for further information.<br/>';
//s += 'You can track status from the <a href="home.do" class="breadcrumb" >Home Page</a> <br/>';
fi.addMessage(s);
producer.redirect = 'home.do?sysparm_view=ess';
```

flushMessages()

Clears session messages saved using addErrorMessage(Object) or addInfoMessage(Object). Session messages are shown at the top of the form. In client side scripts use g_form.clearMessages() to remove session messages.

Output Fields

Returns: void

Example

```
gs.flushMessages();
```

getErrorMessages()

Gets the list of error messages for the session that were added by addErrorMessage(Object)

Output Fields

Returns: list of error messages.

getImpersonatingUserDisplayName()

Returns the display name of the impersonating user.

Output Fields

Returns: display name of impersonating user.

getImpersonatingUserName()

Returns the name of the impersonating user or null if not impersonating

Output Fields

Returns: name of the impersonating user.

getInfoMessages()

Gets the list of info messages for the session that were added via addInfoMessage(Object).

Output Fields

Returns: the list of info messages.

getMessages(String)

Gets the list of messages of the specified type for the session that were added via addMessage(String, Object).

Input Fields

Parameters: string type of message.

Output Fields

Returns: list of messages of string type.

getNodeValue(object, Integer)

Gets the node value for specified index.

Input Fields

Parameters

- Object to examine.
- Integer for the index to get a node value from.

Output Fields

Returns: the node's value.

getNodeName(Object, Integer)

Returns the node name for specified index.

Input Fields

Parameters

- Object to examine.
- Integer for the index to get a node value from.

Output Fields

Returns: the node's name.

getPreference(String, Object)

Gets a user preference.

Input Fields

Parameters:

- String key for the preference.
- Object default value

Output Fields

Returns: a string value for the preference. If null, uses the default value specified above.

getSession()

Returns a GlideSession object. See the GlideSession APIs wiki page for methods to use with the GlideSession object.

Output Fields

Returns: a GlideSession object for the current session.

Example

```
var session = gs.getSession();
```

getSessionID()

Accesses the GlideSession Session ID.

Output Fields

Returns: the Session ID.

getTrivialMessages()

Gets the list of error messages for the session that were added with the trivial flag.

Output Fields

Returns: the list of messages.

getUser()

Returns a reference to the User object for the current user. More information is available here.

Output Fields

Returns: a reference to a User object.

Example

```
var myUserObject = gs.getUser()
```

getUserDisplayName()

Returns the name field of the current user (e.g. John Smith, as opposed to smith).

Output Fields

Returns: the name field of the current user (e.g. John Smith, as opposed to jsmith).

Example

```
    <g2:evaluate var="jvar_current_user" expression="gs.getUserDisplayName()"/>
     $[JS:jvar_current_user]
```

getUserID()

Returns the sys_id of the current user.

Output Fields

Returns: the sys_id of the current user.

Example

```
if (current.operation() != 'insert' && current.comments.changes()) {
    gs.eventQueue("incident.commented", current, gs.getUserID(),
gs.getUserName());
}
```

getUserName()

Returns the username of the current user (e.g., jsmith).

Output Fields

Returns: the username of the current user (e.g., jsmith).

Example

```
//Add a comment when closing
    current.comments = "Closed by " + gs.getUserName() + " at " +
gs.nowDateTime();
    gs.addInfoMessage("Close comment added");
}
```

getUserNameByUserID(String)

Gets the username based on a user ID.

Input Fields

Parameters: a sys_id as a string.

Output Fields

Returns: the username.

getXMLText (String, String)

Gets the xml text for the first node in the xml string that matches the xpath query.

Input Fields

Parameters:

- XML to search within
- xpath query to match

Output Fields

Returns: the XML node value as text.

getXMLNodeList(String)

Constructs an Array of all the nodes and values in an XML document.

Input Fields

Parameters: XML document to parse.

Output Fields

Returns: an array list of names and values.

hasRole(String)

Determines if the current user has the specified role. This method returns true for users with the administrator role.

Input Fields

Parameters: the role to check.

Output Fields

Returns: true if the user has the role or false if the user does not have the role.

Example

```
if (!gs.hasRole("admin") && !gs.hasRole("groups_admin") &&
gs.getSession().isInteractive()) {
  var qc = current.addQuery("u_hidden", "!=", "true"); //cannot see
hidden groups...
```

```
  qc.addOrCondition("sys_id", "javascript:getMyGroups()"); //...unless
in the hidden group
}
```

hasRoleInGroup(Object, Object)

Determines if the current user has the specified role within a specified group.

Input Fields

Parameters:

- The name of the role to check for.
- A GlideRecord or the sys_id of the group to check within.

Output Fields

Returns: Returns true if all of the following conditions are met:

1. The logged-in user HAS the role in question
2. The "Granted by" field on the user role record is set to the specified group
3. The "inherited" field on the user role record is false

Example

```
var group = new GlideRecord('sys_user_group');
group.addQuery('name', 'GROUP_NAME');
group.setLimit(1);
group.query();
if (group.next()) {
   if (gs.hasRoleInGroup('role_name', group)) {
      gs.print('User has role in group');
   } else {
      gs.print('User does NOT have role in group');
   }
}
```

isInteractive()

Checks if the current session is interactive. An example of an interactive session is when a user logs in using the log-in screen. An example of a non-interactive session is using a SOAP request to retrieve data.

Output Fields

Returns: true if the session is interactive.

Example

```
if (!gs.hasRole('admin') && gs.isInteractive()) {
    var qc1 = current.addQuery('u_group', '');
    var gra = new GlideRecord('sys_user_grmember');
    gra.addQuery('user', gs.getUserID());
    gra.query();
    while (gra.next()) {
      qc1.addOrCondition('u_group', gra.group);
   }
}
```

isLoggedIn()

Determines if the current user is currently logged in.

Output Fields

Returns: true if the current user is logged in, false if the current user is not.

setRedirect(Object)

Sets the redirect URI for this transaction. This determines the next page the user will see.

Input Fields

Parameters: URI object to set as redirect.

Example

The following example redirects the user to a particular Catalog Item, and passes along the current email as a parameter:

```
gs.setRedirect("com.glideapp.servicecatalog_cat_item_view.do?sysparm_id=d41ce5bac611227a0167f4bf8109bf70&sysparm_user="


+ current.sys_id + "&sysparm_email=" + current.email)
```

setReturn(Object)

Sets the return URI for this transaction. This determines what page the user will be directed to when they return from the next form.

Input Fields

Parameters: URI object to set as return.

Example

The following example ensures that the user will be returned to the current page when they are done with the next one.

```
   gs.setReturn (current.getLink(true));
```

userID()

Returns the sys_id of the user associated with this session. A shortcut for the more proper getUserID().

Output Fields

Returns: sys_id of current user.

References

[1] https://docs.servicenow.com/bundle/jakarta-servicenow-platform/page/script/glide-server-apis/concept/c_GlideSystem.html
[2] http://docs.oracle.com/javase/6/docs/api/?java/text/MessageFormat.html

GlideElement

Note: *This article applies to Fuji. For more current information, see Server API Reference* [1] *at* http://docs.servicenow.com The ServiceNow Wiki is no longer being updated. Please refer to http://docs.servicenow.com for the latest product documentation.

Overview

GlideElement provides a number of convenient script methods for dealing with fields and their values. GlideElement methods are available for the fields of the current GlideRecord.

Note: *The global Glide APIs are not available in scoped scripts. There are scoped Glide APIs for use in scoped scripts. The scoped Glide APIs provide a subset of the global Glide API methods. For information on scoped applications see Application Scope, scripting in scoped applications, and the list of scoped APIs.*

Method Summary

Method Summary	Description
canCreate()	Determines if the user's role permits creation of new records in this field.
canRead()	Determines if the GlideRecord table can be read from.
canWrite()	Determines if the GlideRecord table can be written to.
changes()	Determines if the current field has been modified.
changesFrom(Object)	Determines the previous value of the current field matched a certain object.
changesTo(Object)	Determines if the new value of a field after a change matches a certain object.
debug(Object)	Debugs the object and adds debug messages using setError(String).
getAttribute(String)	Gets the value of the attribute on the field in question from the dictionary as a string. If the attribute is a boolean attribute, use getBooleanAttribute(String) to get the value as a boolean rather than as a string.
getBaseTableName()	Gets the base table of the field.
getBooleanAttribute(String)	Gets the value of the attribute on the field in question from the dictionary as a boolean. To get the value as a string, use getAttribute(string).
getChoices(String)	Generates a choice list for a field. Returns the choice values from the base table only, not from the extended table.
getChoiceValue()	Gets the choice label for the current choice value.

getDebugCount()	Gets the number of debug messages logged by debug()
getDependent()	Gets the field that this field is dependent on.
getDependentTable()	Gets the table that the current table depends on.
getDisplayValue(Int)	Gets the formatted display value of the field.
getDisplayValueExt(Int, String)	Gets the formatted display value of a field, or a specified substitute value if the display value is null or empty.
getED()	Gets an element descriptor.
getElementValue(String)	Gets the value for a given element.
getError()	Gets error debug messages.
getEscapedValue()	Gets the escaped value for the current element.
getFieldStyle()	Gets the CSS style for the field.
getGlideObject()	Gets a glide object.
getGlideRecord()	Gets the GlideRecord object that contains the element. To get a GlideRecord for a reference element, use getRefRecord().
getHTMLValue(Int)	Gets the HTML value of a field.
getHTMLValueExt(Int, String)	Gets the HTML value of a field, or a specified substitute value if the HTML value is null or empty.
getJournalEntry(int)	Gets either the most recent journal entry or all journal entries.
getLabel()	Gets the object's label.
getName()	Gets the name of the field.
getRefRecord()	Gets a GlideRecord object for a given reference element.
getStyle()	Get a CSS style for the value.
getTableName()	Gets the name of the table the field is on.
getTextAreaDisplayValue()	Gets the value and escapes the HTML.
getXHTMLValue()	Gets the XHTML value of a field as a string.
getXMLValue()	Gets the XML value of a field as a string.
hasAttribute(String)	Determines whether a field has a particular attribute.
hasRightsTo(String)	Determines if the user has the right to perform a particular operation.
hasValue()	Determines if the field has a value.
nil()	Determines whether the field is null.
setDisplayValue(Object)	Sets the display value of the field.
setError(String)	Adds an error message. Can be retrieved using getError().
setInitialValue(String)	Sets the initial value of a field.
setJournalEntry(Object, String)	Sets a journal entry.
setValue(Object)	Sets the value of a field.
toString()	Converts the value to a string.

Method Detail

canCreate()

Determines if the user's role permits creation of new records in this field.

Output Fields

Returns: true if the field can be created, false if the field cannot.

canRead()

Determines if the GlideRecord table can be read from.

Output Fields

Returns: true if the field can be read, false if the field cannot.

canWrite()

Determines if the GlideRecord table can be written to.

Output Fields

Returns: true if the field can be written to, false if the field cannot.

changes()

Determines if the current field has been modified, if the field is a reference, integer, or string field. Note that changes to Journal fields are not detected by this method.

Output Fields

Returns: true if the fields have been changed, false if the field has not.

changesFrom(Object)

Determines the previous value of the current field matched a certain object.

Input Fields

Parameters: an object value to check against the previous value of the current field.

Output Fields

Returns: true if the field's previous value matches, false if it does not.

Example

```
if (theState.changesTo(resolvedState)) {
  operation = 4; //Resolved
}
else if (theState.changesTo(closedState)) {
  operation = 11; //Resolution Accepted
}
else if (theState.changesFrom(resolvedState) ||
theState.changesFrom(closedState)) {
  operation = 10; //Re-open
}
else {
  operation = 6; //Update
}
```

changesTo(Object)

Determines if the new value of a field after a change matches a certain object.

Input Fields

Parameters: an object value to check against the new value of the current field.

Output Fields

Returns: true if the field's new value matches, false if it does not.

Example

```
if (theState.changesTo(resolvedState)) {
  operation = 4; //Resolved
}
else if (theState.changesTo(closedState)) {
  operation = 11; //Resolution Accepted
}
else if (theState.changesFrom(resolvedState) ||
theState.changesFrom(closedState)) {
  operation = 10; //Re-open
}
else {
  operation = 6; //Update
}
```

debug(Object)

Debugs the object and adds debug messages using setError(String).

Input Fields

Parameters: an object to debug.

getAttribute(String)

Gets the value of the attribute on the field in question from the dictionary as a string. If the attribute is a boolean attribute, use getBooleanAttribute(String) to get the value as a boolean rather than as a string.

Input Fields

Parameters: a string attribute name to get a value from.

Output Fields

Returns: the attribute's value as a string.

Example

```
doit();
function doit() {
  var gr = new GlideRecord('sys_user');
  gr.query("user_name","admin");
  if (gr.next()) {
    gs.print("we got one");
    gs.print(gr.location.getAttribute("tree_picker"));
  }

}
```

getBaseTableName()

Gets the base table of the field.

Note: *This may be different from the table that the field is defined on. See Tables and Classes.*

Output Fields

Returns: name of the base table.

getBooleanAttribute(String)

Gets the value of the attribute on the field in question from the dictionary as a boolean. To get the value as a string, use getAttribute(string).

Input Fields

Parameters: the string name of the attribute to get a value from.

Output Fields

Returns: the boolean value of the string.

getChoices(String)

Generates a choice list for a field. Returns the choice values from the base table only, not from the extended table.

Input Fields

Parameters: a dependent value (optional).

Output Fields

Returns: an array list of choices.

Example

```
var field = gr.getElement('os');
var choices = field.getChoices();
```

getChoiceValue()

Gets the choice label for the current choice value.

Output Fields

Returns: a string choice label.

getDebugCount()

Gets the number of debug messages logged by debug()

Output Fields

Returns: an integer number of debug messages.

getDependent()

Checks whether or not the field is dependent on another field.

Output Fields

Returns: the name of the field the current field depends on.

getDependentTable()

Gets the table that the current table depends on.

Output Fields

Returns: the string name of the table.

getDisplayValue(Int)

Gets the formatted display value of the field.

Input Fields

Parameters: an integer number of maximum characters desired (optional).

Output Fields

Returns: the display value of the field.

Example

```
var fields = current.getFields();
for (var i = 0; i < fields.size(); i++) {
  var field = fields.get(i);
  var name = field.getName();
  var value = field.getDisplayValue();
  gs.print(i + ". " + name + "=" + value);
}
```

getDisplayValueExt(Int, String)

Gets the formatted display value of a field, or a specified substitute value if the display value is null or empty.

Input Fields

Parameters:

- **maxCharacters** - (int) the number of maximum characters desired (optional).
- **nullSub** - (String) the value to return if the display value is null or empty.

Output Fields

Returns: (String) the formatted display value of a field, or a specified substitute value

getED()

Gets an element descriptor.

Output Fields

Returns: the element descriptor.

Example

```
var fields = current.getFields();
for (i=0; i<fields.size(); i++) {
  var field = fields.get(i);
  var descriptor = field.getED();
  gs.print("type=" + descriptor.getType() +
    " internalType=" + descriptor.getInternalType());
}
```

getElementValue(String)

Gets the value for a given element.

Input Fields

Parameters: an element to get a value from.

Output Fields

Returns: the value of the element.

getError()

Gets error debug messages.

Output Fields

Returns: a string of debug messages.

getEscapedValue()

Gets the escaped value for the current element.

Output Fields

Returns: the escaped value of the current element.

getFieldStyle()

Gets the CSS style for the field.

Output Fields

Returns: the CSS style for the field as a string.

Example

```
var fields = current.getFields();
for (var i = 0; i < fields.size(); i++) {
  var field = fields.get(i);
  var name = field.getName();
  var value = field.getDisplayValue();
  gs.print(i + ". " + name + "=" + value);
}
```

getGlideObject()

Gets a glide object.

Output Fields

Returns: a glide object.

Example

Note: *This API call changed in the Calgary release:*

- *GlideCalendar* replaces *Packages.com.glide.schedule.GlideCalendar*

The new script object calls apply to the Calgary release and beyond. For releases prior to Calgary, substitute the packages call as described above. Packages calls are not valid beginning with the Calgary release. For more information, see Scripting API Changes.

```
function calcDateDelta(start, end, calendar) {
  var cal = GlideCalendar.getCalendar(calendar);
  if (!cal.isValid())
      return null;
  var realStart = start.getGlideObject();
  var realEnd = end.getGlideObject();
  var duration = cal.subtract(realStart, realEnd);
  return duration;
}
```

getGlideRecord()

Gets a glide record.

Output Fields

Returns: a glide record.

Example

```
function task_ci_allocate() {
   var cnt = g_list.getRowCount();
   if (cnt == 0)
      return;

   var pct = 100.0 / cnt;
   var pct = (parseInt((pct + .005) * 100)) / 100;
   var gr = g_list.getGlideRecord();
```

```
    gr.query();
    while (gr.next()) {
       gr.u_allocation = pct;
       gr.update();
    }
}
```

getHTMLValue(Int)

Gets the HTML value of a field.

Parameters:

- **maxChars** - (int) the number of maximum characters desired (optional).

Output Fields

Returns: an integer value of maximum characters to limit the output.

Example

```
var inccause = new GlideRecord("incident");
inccause.short_description = current.short_description;
inccause.comments = current.comments.getHTMLValue();
inccause.insert();
```

getHTMLValueExt(Int, String)

Gets the HTML value of a field, or a specified substitute value if the HTML value is null or empty.

Input Fields

Parameters:

- **maxCharacters** - (int) the number of maximum characters desired (optional).
- **nullSub** - (String) the value to return in the HTML value is null or empty.

Output Fields

Returns: the HTML value of a field, or a specified substitute value.

getJournalEntry(int)

Gets either the most recent journal entry or all journal entries.

Parameters: -1: get all journal entries, 1: get the most recent journal entry.

Output Fields

Returns: (Sting) For the most recent entry, returns a sting that contains the field label, timestamp, and user display name of the journal entry. For all journal entries, returns the same information for all journal entries ever entered as a single string with each entry delimited by "\n\n".

Example

```
var notes = current.work_notes.getJournalEntry(-1); //gets all journal
entries as a string where each entry is delimited by '\n\n'
var na = notes.split("\n\n");                       //stores each entry
 into an array of strings

for (var i = 0; i < na.length; i++)
  gs.print(na[i]);
```

getLabel()

Gets the object's label.

Output Fields

Returns: the label as a string.

Example

```
var gr = new GlideRecord("sc_req_item");
gr.addQuery("request", current.sysapproval);
gr.query();
while(gr.next()) {
    var nicePrice = gr.price.toString();
    if (nicePrice != ) {
        nicePrice = parseFloat(nicePrice);
        nicePrice = nicePrice.toFixed(2);
    }
    template.print(gr.number + ":  " + gr.quantity + " X " +
gr.cat_item.getDisplayValue() + " at $" + nicePrice + " each \n");
    template.print("    Options:\n");
    for (key in gr.variables) {
      var v = gr.variables[key];
      if(v.getGlideObject().getQuestion().getLabel() != ) {
         template.space(4);
         template.print('     ' +
v.getGlideObject().getQuestion().getLabel() + " = " +
v.getDisplayValue() + "\n");
      }
    }
}
```

getName()

Gets the name of the field.

Output Fields

Returns: string name of the field.

Example

Note: *This API call changed in the Calgary release:*

- *GlideMutex* replaces *Packages.com.glide.sys.lock.Mutex*

The new script object calls apply to the Calgary release and beyond. For releases prior to Calgary, substitute the packages call as described above. Packages calls are not valid beginning with the Calgary release. For more information, see Scripting API Changes.

```
var lock_name = "my_lock";
gs.print("getting lock");
var lock = new GlideMutex.get(lock_name);
    //.get(lock_name) gets an exclusive lock; replace with
.getNonExclusive(lock_name) for a non-exclusive lock)
while (lock.toString()+'' == "undefined") {''
   gs.print("waiting for lock: " + lock_name);
   gs.sleep(1000); // sleep for 1000 milliseconds
   lock = new GlideMutex.get(lock_name);
}
gs.print("got lock for " + lock.getName());
//***********************************
// do your activity requiring a lock here
//***********************************
lock.release();
gs.print("released lock");
```

getRefRecord()

Gets a GlideRecord object for a given reference element.

Output Fields

Returns: a GlideRecord object.

Example

```
var grINC = new GlideRecord('incident');
grINC.notNullQuery('caller_id');
grINC.query();
if (grINC.next()) {

  // Get a GlideRecord object for the referenced sys_user record
  var grUSER = grINC.caller_id.getRefRecord();
  gs.print( grUSER.getValue('name') );

}
```

getStyle()

Get a CSS style for the value.

Output Fields

Returns: the CSS style for the value.

Example

```
// Get string of style field from Field Style record, if applicable
var cssStyle = gr.state.getStyle();
```

getTableName()

Gets the name of the table the field is on.

Note: *This may be different from the table class that the record is in. See Tables and Classes.*

Output Fields

Returns: string name of the table.

Example

```
 if (current.getTableName() == "sysapproval_approver") {
  if (current.approver == email.from_sys_id)  {
     current.comments = "reply from: " + email.from + "\n\n" +
email.body_text;

   // if it's been cancelled, it's cancelled.
  var doit = true;
  if (current.state=='cancelled')
      doit = false;

  if (email.body.state != undefined)
     current.state= email.body.state;

   if (doit)
      current.update();
} else {
   gs.log("Approval for task
("+current.sysapproval.getDisplayValue()+") rejected because user
sending
           email( "+email.from+") does not match the approver
("+current.approver.getDisplayValue()+")");
}
}
```

getTextAreaDisplayValue()

Gets the value and escapes the HTML.

Output Fields

Returns: the string value with the HTML escaped.

getXHTMLValue()

Gets the XHTML value of a field as a string.

Output Fields

Returns: the XHTML value as a string.

getXMLValue()

Gets the XML value of a field as a string.

Output Fields

Returns: the XML value as a string.

hasAttribute(String)

Determines whether a field has a particular attribute.

Input Fields

Parameters: the string attribute to check for.

Output Fields

Returns: true if the field has the attribute, false if not.

Example

```
var totalCritical = 0;

var filledCritical = 0; var fields = current.getFields();
gs.print(fields); for (var num = 0; num &lt; fields.size(); num++) {

    gs.print("RUNNING ARRAY VALUE " + num);
   var ed = fields.get(num).getED();
   if(ed.hasAttribute("tiaa_critical")) {
       gs.print("CRITICAL FIELD FOUND");
       totalCritical ++;
       if (!fields.get(num).isNil()) {
           filledCritical ++;
       }
   }

} var answer = 0; gs.print("TOTAL - " + totalCritical);
gs.print("FILLED - " + filledCritical); if (filledCritical &gt; 0
&amp;&amp; totalCritical &gt; 0){

    var pcnt = (filledCritical/totalCritical)*100;
   answer = pcnt.toFixed(2);;

} answer;
```

hasRightsTo(String)

Determines if the user has the right to perform a particular operation.

Input Fields

Parameters: the string name of the operation to check for.

Output Fields

Returns: true of the user has the right, false if the user does not.

hasValue()

Determines if the field has a value.

Output Fields

Returns: true if the field has a value, false if not.

nil()

Determines whether the field is null.

Output Fields

Returns: true if the field is null or an empty string, false if not.

Example

```
 if (current.start_date.changes() || current.end_date.changes() ||
current.assigned_to.changes()) {
  if (!current.start_date.nil() && !current.end_date.nil() &&
!current.assigned_to.nil()) {
 gs.eventQueue("change.calendar.notify", current, current.assigned_to,
previous.assigned_to);

}
```

setDisplayValue(Object)

Sets the display value of the field.

Input Fields

Parameters: the object to serve as the display value.

Example

```
current.assignment_group.setDisplayValue('Network');
```

setError(String)

Adds an error message. Can be retrieved using getError().

Input Fields

Parameters: a string error message.

Example

```
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);
  }
}
```

setInitialValue(String)

Sets the initial value of a field.

Input Fields

Parameters: a string value of a field.

setJournalEntry(Object, String)

Sets a journal entry. The username parameter does not set the journal entry's created by field.

Input Fields

Parameters:

- The value to set to the journal entry.
- The username to attribute the journal entry to. Does not set the journal entry's created by field.

setValue(Object)

Sets the value of a field.

Input Fields

Parameters: object value to set the field to.

Example

```
// Using GlideElement.setValue (equivalent to GlideRecord.setValue)
gr.short_description.setValue('This is the short description.');

// Using GlideRecord.setValue
gr.setValue('short_description', 'This is the short description.');
```

toString()

Converts the value to a string.

Output Fields

Returns: the value as a string.

Example

```
doit();

function doit() {

  var gr = new GlideRecord('sys_user');
  gr.query();
  while (gr.next()) {
    if ((gr.first_name.toString().length !=
gr.first_name.toString().trim().length) ||
(gr.last_name.toString().length
         != gr.last_name.toString().trim().length)) {
      gr.first_name = gr.first_name.toString().trim();
      gr.last_name = gr.last_name.toString().trim();
      gr.autoSysFields(false);
      gr.update();
    }
  }

}
```

References

[1] https://docs.servicenow.com/bundle/jakarta-servicenow-platform/page/script/server-api/topic/p_ServerAPIReference.html

GlideAggregate

Note: *This article applies to Fuji. For more current information, see GlideAggregate* [1] *at* http://docs.servicenow.com The ServiceNow Wiki is no longer being updated. Please refer to http://docs.servicenow.com for the latest product documentation.

Note: This functionality requires a knowledge of **JavaScript**.

Overview

The GlideAggregate class is an extension of GlideRecord and allows database aggregation (COUNT, SUM, MIN, MAX, AVG) queries to be done. This can be helpful in creating customized reports or in calculations for calculated fields. GlideAggregation is an extension of GlideRecord and its use is probably best shown through a series of examples.

Note: *The global Glide APIs are not available in scoped scripts. There are scoped Glide APIs for use in scoped scripts. The scoped Glide APIs provide a subset of the global Glide API methods. For information on scoped applications see Application Scope, scripting in scoped applications, and the list of scoped APIs.*

Examples

Here is an example that simply gets a count of the number of records in a table:

```
var count = new GlideAggregate('incident');
count.addAggregate('COUNT');
count.query();
var incidents = 0;
if (count.next())
   incidents = count.getAggregate('COUNT');
```

There is no query associated with the above example but if you wanted to get a count of the incidents that were open then you simply add a query just as is done with GlideRecord. Here is an example to get a count of the number of active incidents.

```
var count = new GlideAggregate('incident');
count.addQuery('active', 'true');
count.addAggregate('COUNT');
count.query();
var incidents = 0;
if (count.next())
   incidents = count.getAggregate('COUNT');
```

To get a count of all of the open incidents by category the code is:

```
var count = new GlideAggregate('incident');
count.addQuery('active', 'true');
count.addAggregate('COUNT', 'category');
count.query();
while (count.next()) {
   var category = count.category;
   var categoryCount = count.getAggregate('COUNT', 'category');
   gs.log("The are currently " + categoryCount + " incidents with a
category of " + category);
}
```

The output is:

```
       *** Script: The are currently 1.0 incidents with a category of Data
       *** Script: The are currently 11.0 incidents with a category of Enhancement
       *** Script: The are currently 1.0 incidents with a category of Implementation
       *** Script: The are currently 197.0 incidents with a category of inquiry
       *** Script: The are currently 13.0 incidents with a category of Issue
       *** Script: The are currently 1.0 incidents with a category of
       *** Script: The are currently 47.0 incidents with a category of request
```

Below is an example that shows that you can ask for multiple aggregations. We are asking to see how many times records have been modified and we want the MIN, MAX, and AVG values.

```
var count = new GlideAggregate('incident');
count.addAggregate('MIN', 'sys_mod_count');
count.addAggregate('MAX', 'sys_mod_count');
count.addAggregate('AVG', 'sys_mod_count');
count.groupBy('category');
count.query();
while (count.next()) {
   var min = count.getAggregate('MIN', 'sys_mod_count');
   var max = count.getAggregate('MAX', 'sys_mod_count');
   var avg = count.getAggregate('AVG', 'sys_mod_count');
   var category = count.category.getDisplayValue();
   gs.log(category + " Update counts: MIN = " + min + " MAX = " + max +
 " AVG = " + avg);
}
```

The output is:

```
       *** Script: Data Import Update counts: MIN = 4.0 MAX = 21.0 AVG = 9.3333
       *** Script: Enhancement Update counts: MIN = 1.0 MAX = 44.0 AVG = 9.6711
       *** Script: Implementation Update counts: MIN = 4.0 MAX = 8.0 AVG = 6.0
       *** Script: inquiry Update counts: MIN = 0.0 MAX = 60.0 AVG = 5.9715
       *** Script: Inquiry / Help Update counts: MIN = 1.0 MAX = 3.0 AVG = 2.0
       *** Script: Issue Update counts: MIN = 0.0 MAX = 63.0 AVG = 14.9459
       *** Script: Monitor Update counts: MIN = 0.0 MAX = 63.0 AVG = 3.6561
       *** Script: request Update counts: MIN = 0.0 MAX = 53.0 AVG = 5.0987
```

Here is a somewhat more complex example that shows how to compare activity from one month to the next.

```
var agg = new GlideAggregate('incident');
agg.addAggregate('count','category');
agg.orderByAggregate('count', 'category');
agg.orderBy('category');
agg.addQuery('opened_at', '>=', 'javascript:gs.monthsAgoStart(2)');
agg.addQuery('opened_at', '<=', 'javascript:gs.monthsAgoEnd(2)');
agg.query();
while (agg.next()) {
  var category = agg.category;
  var count = agg.getAggregate('count','category');
  var query = agg.getQuery();
  var agg2 = new GlideAggregate('incident');
  agg2.addAggregate('count','category');
  agg2.orderByAggregate('count', 'category');
  agg2.orderBy('category');
  agg2.addQuery('opened_at', '>=', 'javascript:gs.monthsAgoStart(3)');
  agg2.addQuery('opened_at', '<=', 'javascript:gs.monthsAgoEnd(3)');
  agg2.addEncodedQuery(query);
  agg2.query();
  var last = "";
  while (agg2.next()) {
     last = agg2.getAggregate('count','category');
  }
  gs.log(category + ": Last month:" + count + " Previous Month:" +
last);

}
```

The output is:

```
*** Script: Monitor: Last month:6866.0 Previous Month:4468.0
 *** Script: inquiry: Last month:142.0 Previous Month:177.0
 *** Script: request: Last month:105.0 Previous Month:26.0
 *** Script: Issue: Last month:8.0 Previous Month:7.0
 *** Script: Enhancement: Last month:5.0 Previous Month:5.0
 *** Script: Implementation: Last month:1.0 Previous Month:0
```

Here is an example to get distinct count of a field on a group query.

```
var agg = new GlideAggregate('incident');
agg.addAggregate('count');
agg.addAggregate('count(distinct','category');
agg.addQuery('opened_at', '>=', 'javascript:gs.monthsAgoStart(2)');
agg.addQuery('opened_at', '<=', 'javascript:gs.monthsAgoEnd(2)');
//
agg.groupBy('priority');
agg.query();
while (agg.next()) {
// Expected count of incidents and count of categories within each
```

```
priority value (group)
gs.info('Incidents in priority ' + agg.priority + ' = ' +
agg.getAggregate('count') +
            ' (' + agg.getAggregate('count(distinct','category') + '
categories)');
}
```

The output is:

```
*** Script: Incidents in priority 1 = 13 (3 categories)
*** Script: Incidents in priority 2 = 10 (5 categories)
*** Script: Incidents in priority 3 = 5 (3 categories)
*** Script: Incidents in priority 4 = 22 (6 categories)
```

Method Summary

Method Summary

Return Value	Details
void	**addEncodedQuery**(`String query`) Adds a query to the Aggregate. Adds an encoded query to the other queries that may have been set for this aggregate.
void	**addHaving**(`String name, String operator, String value`) Adds a "having" element to the aggregate e.g. select category, count(*) from incident group by category HAVING count(*) > 5
void	**addAggregate**(`String agg, String name`) Adds an aggregate.
void	**addTrend**(`String fieldName, String timeInterval`) Adds a trend for a field.
String	**getAggregate**(`String agg, String name`) Gets the value of an aggregate from the current record.
String	**getQuery**() Gets the query necessary to return the current aggregate.
int	**getTotal**(`String agg, String name`) Gets the total number of records by summing an aggregate.
String	**getValue**(`String name`) Gets the value of a field.
void	**groupBy**(`String name`) Provides the name of a field to use in grouping the aggregates. May be called numerous times to set multiple group fields.
void	**orderBy**(`String name`) Provides the name of a field that should be used to order the aggregates. The field will also be added to the group-by list.
void	**orderByAggregate**(`String agg, String name`) Provides the name of an aggregate that should be used to order the result set.
void	**query**() Issues the query and get some results.

void **setGroup**(`Boolean b`)

Sets whether grouping is true or false.

Method Detail

addEncodedQuery

public void addEncodedQuery(`String query`)

Adds a query to the Aggregate. Adds an encoded query to the other queries that may have been set for this aggregate.

Parameters:

`query` - An encoded query string to add to the aggregate.

Example:

```
var agg = new GlideAggregate('incident');
agg.addAggregate('count','category');
agg.orderByAggregate('count', 'category');
agg.orderBy('category');
agg.addQuery('opened_at', '>=', 'javascript:gs.monthsAgoStart(2)');
agg.addQuery('opened_at', '<=', 'javascript:gs.monthsAgoEnd(2)');
agg.query();
while (agg.next()) {
  var category = agg.category;
  var count = agg.getAggregate('count','category');
  var query = agg.getQuery();
  var agg2 = new GlideAggregate('incident');
  agg2.addAggregate('count','category');
  agg2.orderByAggregate('count', 'category');
  agg2.orderBy('category');
  agg2.addQuery('opened_at', '>=', 'javascript:gs.monthsAgoStart(3)');
  agg2.addQuery('opened_at', '<=', 'javascript:gs.monthsAgoEnd(3)');
  agg2.addEncodedQuery(query);
  agg2.query();
  var last = "";
  while (agg2.next()) {
     last = agg2.getAggregate('count','category');
  }
  gs.log(category + ": Last month:" + count + " Previous Month:" +
last);

}
```

addHaving

public void addHaving(String name, String operator, String value)

Adds a "having" element to the aggregate e.g. select category, count(*) from incident group by category HAVING count(*) > 5

Parameters:

`String name` - the aggregate to filter on (e.g. COUNT).

`String operator` - for the operator symbol (e.g. <, >, =, !=)

`String value` to query on (e.g. '5' or '69')

addAggregate

public void addAggregate(String agg, String name)

Adds an aggregate.

Parameters:

`String agg` - name of aggregate to add.

`String name` - name of column to aggregate.

Example:

```
function doMyBusinessRule(assigned_to, number) {
  var agg = new GlideAggregate('incident');
  agg.addQuery('assigned_to', assigned_to);
  agg.addQuery('category', number);
  agg.addAggregate("COUNT");
  agg.query();
  var answer = 'false';
  if (agg.next()) {
    answer = agg.getAggregate("COUNT");
    if (answer > 0)
      answer = 'true';
    else
      answer = 'false';
  }
  return answer;
}
```

addTrend

public void addTrend(String fieldName, String timeInterval)

Adds a trend for a field.

Parameters:

`fieldName` - The string name of the field for which trending should occur.

`timeInterval` - The time interval for the trend. The following choices are available:

- Year
- Quarter
- Date
- Week
- DayOfWeek
- Hour
- Value

---

getAggregate

public String getAggregate(String agg, String name)

Gets the value of an aggregate from the current record.

Parameters:

`agg` - String type of the aggregate (e.g. SUM or COUNT)

`name` - String name of the field to get aggregate from.

Returns:

`String` - the value of the aggregate

Example:

```
function doMyBusinessRule(assigned_to, number) {
  var agg = new GlideAggregate('incident');
  agg.addQuery('assigned_to', assigned_to);
  agg.addQuery('category', number);
  agg.addAggregate("COUNT");
  agg.query();
  var answer = 'false';
  if (agg.next()) {
    answer = agg.getAggregate("COUNT");
    if (answer > 0)
      answer = 'true';
    else
      answer = 'false';
  }
  return answer;
}
```

---

getQuery

public String getQuery()

Gets the query necessary to return the current aggregate.

Returns:

`String` - the string query.

Example:

```
var agg = new GlideAggregate('incident');
agg.addAggregate('count','category');
agg.orderByAggregate('count', 'category');
agg.orderBy('category');
agg.addQuery('opened_at', '>=', 'javascript:gs.monthsAgoStart(2)');
agg.addQuery('opened_at', '<=', 'javascript:gs.monthsAgoEnd(2)');
agg.query();
while (agg.next()) {
  var category = agg.category;
  var count = agg.getAggregate('count','category');
  var query = agg.getQuery();
  var agg2 = new GlideAggregate('incident');
  agg2.addAggregate('count','category');
  agg2.orderByAggregate('count', 'category');
  agg2.orderBy('category');
  agg2.addQuery('opened_at', '>=', 'javascript:gs.monthsAgoStart(3)');
  agg2.addQuery('opened_at', '<=', 'javascript:gs.monthsAgoEnd(3)');
  agg2.addEncodedQuery(query);
  agg2.query();
  var last = "";
  while (agg2.next()) {
     last = agg2.getAggregate('count','category');
  }
  gs.log(category + ": Last month:" + count + " Previous Month:" +
last);

}
```

---

getTotal

public int getTotal(String agg, String name)

Gets the total number of records by summing an aggregate.

Parameters:

`agg` - String type of aggregate

`agg` - String name of field to aggregate from

Returns:

`int` - the total.

---

getValue

public String getValue(String name)

Gets the value of a field.

Parameters:

`String name` - the string name of the field.

Returns:

`String value` - the string value of the field.

groupBy

public void groupBy(String name)

Provides the name of a field to use in grouping the aggregates. May be called numerous times to set multiple group fields.

Parameters:

`name` - name of the field to group-by.

Example:

```
Referencing the example in wiki @ Aggregation Support:

  var count = new GlideAggregate('incident');
  count.addAggregate('MIN', 'sys_mod_count');
  count.addAggregate('MAX', 'sys_mod_count');
  count.addAggregate('AVG', 'sys_mod_count');
  count.groupBy('category');
  count.query();
  while (count.next()) {
     var min = count.getAggregate('MIN', 'sys_mod_count');
     var max = count.getAggregate('MAX', 'sys_mod_count');
     var avg = count.getAggregate('AVG', 'sys_mod_count');
     var category = count.category.getDisplayValue();
     gs.log(category + " Update counts: MIN = " + min + " MAX = " + max
 + " AVG = " + avg);
  }
```

orderBy

public void orderBy(String name)

Provides the name of a field that should be used to order the aggregates. The field will also be added to the group-by list.

Parameters:

`name` - name of the field used to order the aggregates.

Example:

```
var agg = new GlideAggregate('incident');
agg.addAggregate('count','category');
agg.orderByAggregate('count', 'category');
agg.orderBy('category');
agg.addQuery('opened_at', '>=', 'javascript:gs.monthsAgoStart(2)');
agg.addQuery('opened_at', '<=', 'javascript:gs.monthsAgoEnd(2)');
agg.query();
while (agg.next()) {
  var category = agg.category;
  var count = agg.getAggregate('count','category');
  var query = agg.getQuery();
  var agg2 = new GlideAggregate('incident');
  agg2.addAggregate('count','category');
  agg2.orderByAggregate('count', 'category');
  agg2.orderBy('category');
  agg2.addQuery('opened_at', '>=', 'javascript:gs.monthsAgoStart(3)');
  agg2.addQuery('opened_at', '<=', 'javascript:gs.monthsAgoEnd(3)');
  agg2.addEncodedQuery(query);
  agg2.query();
  var last = "";
  while (agg2.next()) {
     last = agg2.getAggregate('count','category');
  }
  gs.log(category + ": Last month:" + count + " Previous Month:" +
last);

}
```

orderByAggregate

public void orderByAggregate(String agg, String name)

Provides the name of an aggregate that should be used to order the result set.

Parameters:

`agg` - String type of aggregate (e.g. SUM, COUNT, MIN, MAX)

`name` - String name of field to aggregate

Example:

```
var agg = new GlideAggregate('incident');
agg.addAggregate('count','category');
agg.orderByAggregate('count', 'category');
agg.orderBy('category');
agg.addQuery('opened_at', '>=', 'javascript:gs.monthsAgoStart(2)');
agg.addQuery('opened_at', '<=', 'javascript:gs.monthsAgoEnd(2)');
agg.query();
while (agg.next()) {
  var category = agg.category;
  var count = agg.getAggregate('count','category');
  var query = agg.getQuery();
  var agg2 = new GlideAggregate('incident');
  agg2.addAggregate('count','category');
  agg2.orderByAggregate('count', 'category');
  agg2.orderBy('category');
  agg2.addQuery('opened_at', '>=', 'javascript:gs.monthsAgoStart(3)');
  agg2.addQuery('opened_at', '<=', 'javascript:gs.monthsAgoEnd(3)');
  agg2.addEncodedQuery(query);
  agg2.query();
  var last = "";
  while (agg2.next()) {
     last = agg2.getAggregate('count','category');
  }
  gs.log(category + ": Last month:" + count + " Previous Month:" +
last);

}
```

query

public void query()

Issues the query and gets some results.

Example:

```
var agg = new GlideAggregate('incident');
agg.addAggregate('count','category');
agg.orderByAggregate('count', 'category');
agg.orderBy('category');
agg.addQuery('opened_at', '>=', 'javascript:gs.monthsAgoStart(2)');
agg.addQuery('opened_at', '<=', 'javascript:gs.monthsAgoEnd(2)');
agg.query();
while (agg.next()) {
  var category = agg.category;
  var count = agg.getAggregate('count','category');
  var query = agg.getQuery();
  var agg2 = new GlideAggregate('incident');
  agg2.addAggregate('count','category');
  agg2.orderByAggregate('count', 'category');
  agg2.orderBy('category');
  agg2.addQuery('opened_at', '>=', 'javascript:gs.monthsAgoStart(3)');
  agg2.addQuery('opened_at', '<=', 'javascript:gs.monthsAgoEnd(3)');
  agg2.addEncodedQuery(query);
  agg2.query();
  var last = "";
  while (agg2.next()) {
     last = agg2.getAggregate('count','category');
  }
  gs.log(category + ": Last month:" + count + " Previous Month:" + 
last);
```

setGroup

public void setGroup(boolean b)

Sets whether grouping is true or false.

Parameters:

`b` - true if grouping is true, false if it is false.

References

[1] https://docs.servicenow.com/bundle/jakarta-servicenow-platform/page/script/glide-server-apis/concept/c_GlideAggregate.html

Script Objects

Script Includes

Note: *This article applies to Fuji and earlier releases. For more current information, see Script Includes* [1] *at* http://docs.servicenow.com **The ServiceNow Wiki is no longer being updated. Visit** http://docs.servicenow.com **for the latest product documentation.**'

Overview

Create script includes to store JavaScript functions and classes for use by server scripts. Each script include defines either an object class or a function. Script includes run only when called by a server script.

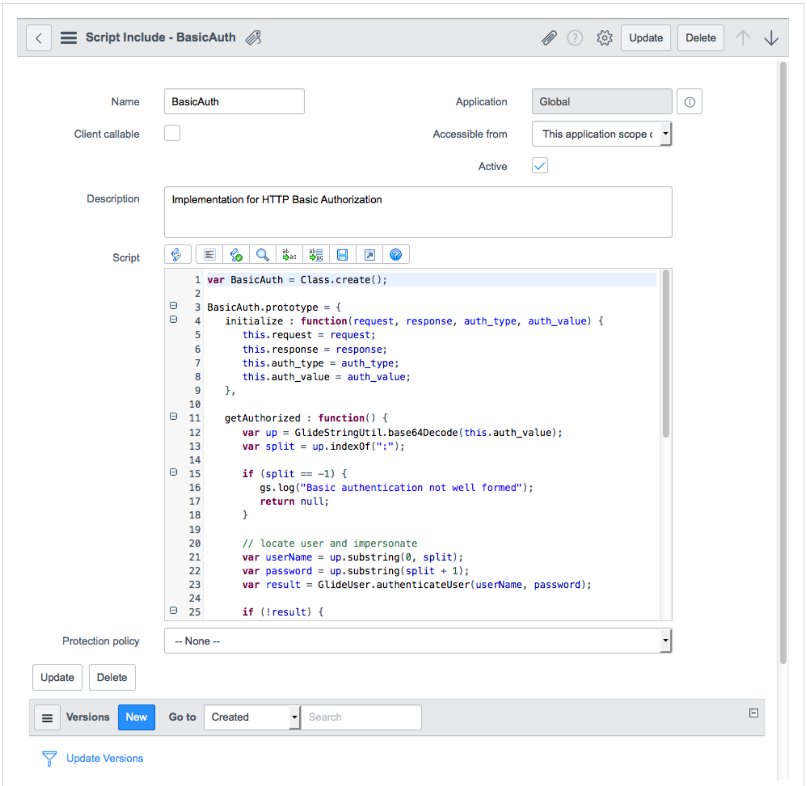
Consider using script includes instead of global business rules because script includes are only loaded on request.

Script Include Form

To access script includes, navigate to **System Definitions > Script Includes**. Script includes have a name, description and script. They also specify whether they are active or not, and whether they can be called from a Client Script.

Field	Description
Name	The name of the script include. If you are defining a class, this must match the name of the class, prototype, and type. If you are using a classless (on-demand) script include, the name must match the function name.
Client callable	Makes the script include available to client scripts, list/report filters, reference qualifiers, or if specified as part of the URL.
Application	The application where this script include resides. This field is available starting with the Fuji release.
Accessible from	Sets what applications can access this script include: • **All application scopes:** Can be accessed from any application scope. • **This application scope only:** Can be accessed only from the current application scope. See Scripting in Scoped Applications for information on scoped scripts. This field is available starting with the Fuji release.
Active	Enables the script include when selected. Uncheck the active field to disable the script include.
Description	Provides descriptive content regarding the script include.
Script	Defines the server side script to run when called from other scripts. The script must define a single JavaScript class or a global function. The class or function name must match the Name field.
Package	The package that contains this script include. This field is available starting with the Fuji release.
Created by	The user who created this script include. This field is available starting with the Fuji release.
Updated by	The user who most recently updated this script include. This field is available starting with the Fuji release.

Protection policy	Sets the level of protection for this script include: • **None:** Allows anyone to read and edit this downloaded or installed script include. • **Read-only:** Allows anyone to read values from this downloaded or installed script include. No one can change script values on the instance on which they download or install it. • **Protected:** Provides intellectual property protection for application developers. Customers who download the script include cannot see the contents of the script field. The script is encrypted in memory to prevent unauthorized users from seeing it in plain text. This field is available starting with the Fuji release.
	Related lists on the form view:
Versions	Shows all versions of the script include. Use this list to compare versions or to revert to a previous version. See Versions.



Name: BasicAuth
Client callable: ☐
Application: Global
Accessible from: This application scope (
Active: ✓
Description: Implementation for HTTP Basic Authorization
Script:

```
var BasicAuth = Class.create();

BasicAuth.prototype = {
   initialize : function(request, response, auth_type, auth_value) {
      this.request = request;
      this.response = response;
      this.auth_type = auth_type;
      this.auth_value = auth_value;
   },

   getAuthorized : function() {
      var up = GlideStringUtil.base64Decode(this.auth_value);
      var split = up.indexOf(":");

      if (split == -1) {
         gs.log("Basic authentication not well formed");
         return null;
      }

      // locate user and impersonate
      var userName = up.substring(0, split);
      var password = up.substring(split + 1);
      var result = GlideUser.authenticateUser(userName, password);

      if (!result) {
```

Protection policy: -- None --

The Script Include form

How Do I Use Them?

Script includes are found under System Definition or System UI. You can call existing script includes from a script.

To create an entirely new script include, you can follow the format of any of the existing script includes. In the example, the name of your Script Include is 'NewInclude' and there is a single function called 'myFunction.' It is important that the name of the script include match the name of the class, prototype, and type. When you create a new script include and give it a name, the system provides you a code snippet with the class and prototype set up properly.

```
var NewInclude = Class.create();

NewInclude.prototype = {
   initialize : function() {
   },

   myFunction : function() {
      //Put function code here
   },

   type : 'NewInclude'
};
```

You could then use the 'myFunction' line like this:

```
var foo = new NewInclude();
foo.myFunction();
```

Note: *Try not to modify a ServiceNow supplied script include. If you want a script include that does something similar to an existing one, copy it and make changes to the copy or consider extending the object. This is a common practice when using GlideAjax.*

Privacy Settings

Most client-callable script includes are marked private by default (starting with the Dublin release). This privacy setting means that guests who access public pages cannot access client-callable script includes. Only the following script includes remain public by default because public pages need to access them:

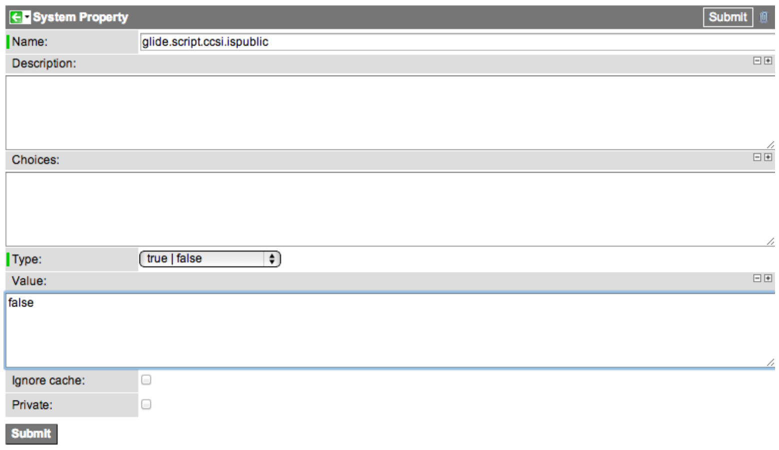
- GlideSystemAjax
- SysMessageAjax
- KnowledgeMessagingAjax
- KnowledgeAjax
- PasswordResetAjax

Client-callable script includes that you created or modified before upgrade to Dublin also remain public.

Changing Privacy on All Client-Callable Script Includes

To provide further control over all client-callable script includes, administrators can add the property `glide.script.ccsi.ispublic`. This property changes the visibility of client-callable script includes by making them all public or private. Configure the property as follows:

- **Name**: glide.script.ccsi.ispublic
- **Type**: true|false
- **Value**: false



Name:	glide.script.ccsi.ispublic
Description:	
Choices:	
Type:	true | false
Value:	false
Ignore cache:	☐
Private:	☐

The glide.script.ccsi.ispublic property.

Changing Privacy on a Single Client Callable Script Include

To change the privacy for a single client-callable script include, add the following method to the script include:

```
  isPublic: function() {
   return [true / false];
  },
```

Use either **true** or **false** for the script include.

Enhancements

Fuji

- Application scoping rules now determine script include visibility.

Dublin

- All client-callable script includes are private by default unless public pages need to access them. If an administrator creates a new client-callable script include in Dublin, it is private by default.

References

[1] https://docs.servicenow.com/bundle/jakarta-servicenow-platform/page/script/server-scripting/concept/c_ScriptIncludes.html

Script Actions

Note: *This article applies to Fuji and earlier releases. For more current information, see Script Actions* [1] *at* http://docs.servicenow.com **The ServiceNow Wiki is no longer being updated. Visit** http://docs.servicenow.com **for the latest product documentation.**'

Overview

You can use Script Actions to create server-side scripts that perform a variety of tasks, such as modifying a configuration item (CI), or managing failed login attempts. Script actions are triggered by events only.

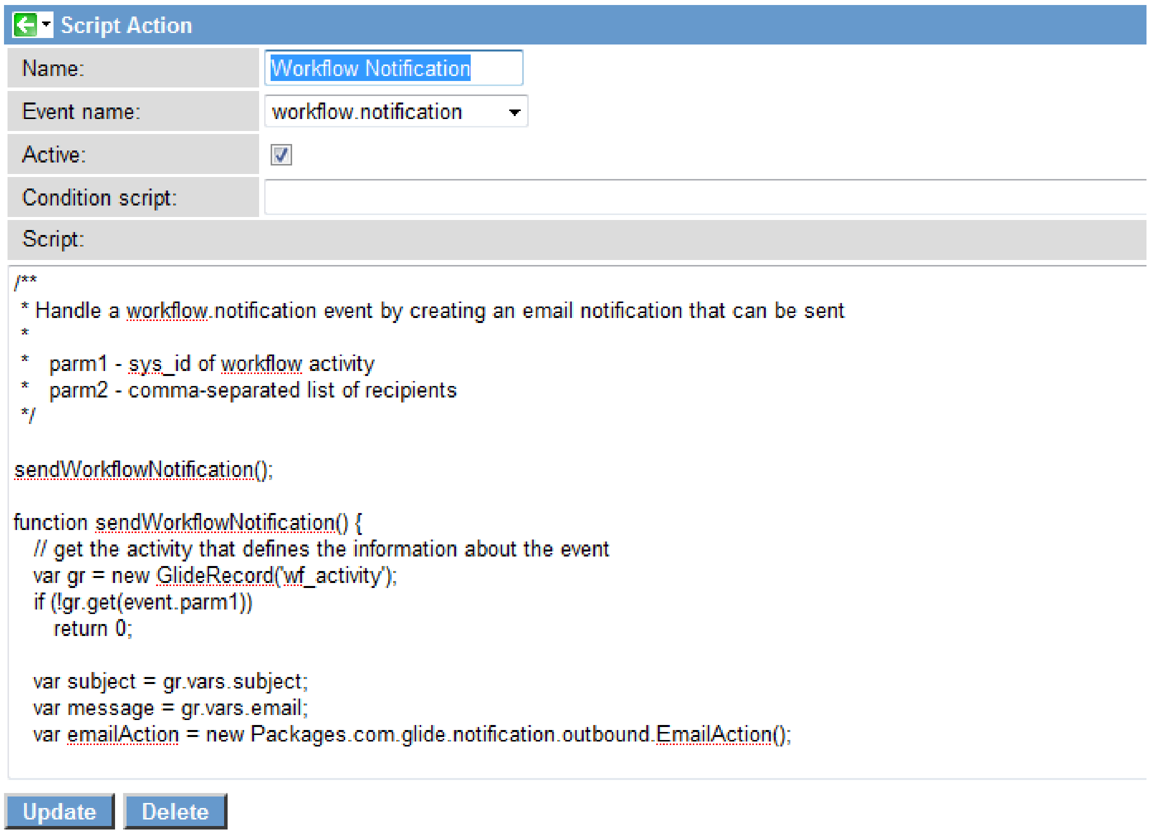
Configuration

To create a new script action, navigate to **System Policy > Events > Script Actions** and click **New**.

Field	Input Value
Name	Type a unique name for your script action.
Event name	Select the event to use for this script. If you do not find an event for your script action that suits your purpose, you can create a new one in Business Rules.
Application	The application that contains this script. This field is available starting with the Fuji release.
Order	The order in which the script will be executed.
Active	Select the check box (*true*) to enable this script action.
Condition script	Create a statement for a condition under which this script should execute. By adding the condition statement to this field, you tell ServiceNow to evaluate the condition separately and parse the script only if the condition is true. If you decide to include the condition statement in the script, leave this field blank.

Script	Create a script that runs when the condition you define evaluates to *true*. Two additional objects are available in this script: • **event**: a GlideRecord - the sysevent that caused this to be invoked. If you wanted so get this first parameter on the event, you would use event.parm1 or event.parm2 for the second parameter. For the date/time of the event, use event.sys_created_on. To get the user ID that created the event (if there was a user associated), use event.user_id. • **current**: a GlideRecord - the event scheduled on behalf of (incident for example).

This is a sample of a script action that creates an email notification for Workflow activity:



Name:	Workflow Notification
Event name:	workflow.notification
Active:	✓
Condition script:	
Script:	

```
/**
 * Handle a workflow.notification event by creating an email notification that can be sent
 *
 *   parm1 - sys_id of workflow activity
 *   parm2 - comma-separated list of recipients
 */

sendWorkflowNotification();

function sendWorkflowNotification() {
   // get the activity that defines the information about the event
   var gr = new GlideRecord('wf_activity');
   if (!gr.get(event.parm1))
      return 0;

   var subject = gr.vars.subject;
   var message = gr.vars.email;
   var emailAction = new Packages.com.glide.notification.outbound.EmailAction();
```

Attachment Logging

Whenever a user downloads an attachment, the action writes an **attachment.read** event record to the event log. If desired, you can process these events with a Script Action or an Email Notification. This can be useful if you want to do something when an attachment is read. For example, you can record when and by whom certain attachments are downloaded. For this functionality, the **current** variable must point to a *sys_attachment* record, and the *event* record must use the following parameters:

- **parm1**: File name
- **parm2**: Table name

References

[1] https://docs.servicenow.com/bundle/jakarta-servicenow-platform/page/administer/platform-events/reference/r_ScriptActions.html

Article Sources and Contributors

Glide Stack *Source*: http://wiki.servicenow.com/index.php?oldid=250292 *Contributors*: CapaJC, Cturner, Emily.partridge, Eric.jacobson, G.yedwab, Guy.yedwab, John.ramos, Neola, Wallymarx

Glide Script Objects *Source*: http://wiki.servicenow.com/index.php?oldid=250603 *Contributors*: CapaJC, Carleen.greenlee, Eric.jacobson, G.yedwab, Guy.yedwab, John.ramos, Neola, Steven.wood, Vhearne, Wallymarx

Using GlideRecord to Query Tables *Source*: http://wiki.servicenow.com/index.php?oldid=250992 *Contributors*: Amy.bowman, CapaJC, Danijel.stanojevic, Don.Goodliffe, G.yedwab, George.rawlins, Gflewis, Guy.yedwab, John.ramos, Joseph.messerschmidt, Mark.stanger, Neola, Phillip.salzman, Steven.wood, Vhearne, Wallymarx

Getting a User Object *Source*: http://wiki.servicenow.com/index.php?oldid=250595 *Contributors*: CapaJC, G.yedwab, George.rawlins, Guy.yedwab, Jim.uebbing, John.ramos, John.roberts, Joseph.messerschmidt, Mark.odonnell, Neola, Pat.Casey, Phillip.salzman, Vaughn.romero, Vhearne

Scripting Alert, Info, and Error Messages *Source*: http://wiki.servicenow.com/index.php?oldid=250521 *Contributors*: CapaJC, Eric.jacobson, Fuji.publishing.user, G.yedwab, Guy.yedwab, Jared.laethem, Jennifer.thorburn, Jerrod.bennett, John.ramos, Joseph.messerschmidt, Mark.stanger, Neola, Steven.wood, Vhearne

Display Field Messages *Source*: http://wiki.servicenow.com/index.php?oldid=250273 *Contributors*: G.yedwab, George.rawlins, Guy.yedwab, John.ramos, John.roberts, Joseph.messerschmidt, Neola, Steven.wood

Setting a GlideRecord Variable to Null *Source*: http://wiki.servicenow.com/index.php?oldid=250884 *Contributors*: G.yedwab, Gflewis, Guy.yedwab, John.ramos, Joseph.messerschmidt, Neola, Steven.wood

Referencing a Glide List from a Script *Source*: http://wiki.servicenow.com/index.php?oldid=251087 *Contributors*: Brad.hicks, Don.Goodliffe, Emily.partridge, G.yedwab, Guy.yedwab, John.ramos, Neola, Phillip.salzman, Steven.wood, Vhearne

Extensions to Jelly Syntax *Source*: http://wiki.servicenow.com/index.php?oldid=250575 *Contributors*: CapaJC, Emily.partridge, Eric.jacobson, George.rawlins, Guy.yedwab, John.maher, John.ramos, Joseph.messerschmidt, Neola, Phillip.salzman, Steven.wood, Vhearne, Wallymarx

How to Escape in Jelly *Source*: http://wiki.servicenow.com/index.php?oldid=250622 *Contributors*: CapaJC, G.yedwab, George.rawlins, Jay.berlin, Jessi.graves, John.ramos, Joseph.messerschmidt, Mark.stanger, Neola, Publishing.user, Steven.wood, Suzanne.smith

Script Syntax Error Checking *Source*: http://wiki.servicenow.com/index.php?oldid=250849 *Contributors*: Eric.jacobson, John.ramos, Joseph.messerschmidt, Neola, Rachel.sienko, Steven.wood

Syntax Editor *Source*: http://wiki.servicenow.com/index.php?oldid=250927 *Contributors*: Dawn.bunting, John.ramos, Rachel.sienko

Using the Syntax Editor *Source*: http://wiki.servicenow.com/index.php?oldid=251029 *Contributors*: John.ramos, Rachel.sienko

JavaScript Debug Window *Source*: http://wiki.servicenow.com/index.php?oldid=250660 *Contributors*: CapaJC, David Loo, Eric.jacobson, Guy.yedwab, Jim.uebbing, John.ramos, John.roberts, Joseph.messerschmidt, Liz.malone, Mark.stanger, Neola, Pat.Casey, Steven.wood, Vaughn.romero, Vhearne

GlideRecord *Source*: http://wiki.servicenow.com/index.php?oldid=250207 *Contributors*: Anat.kerry, Andrew.Kincaid, Brent.bahry, Carleen.greenlee, Chris.henson, Chuck.tomasi, David.Bailey, Eric.jacobson, Fuji.publishing.user, G.yedwab, George.rawlins, Guy.yedwab, Jeremy.norris, Jim.uebbing, John.ramos, John.roberts, Joseph.messerschmidt, Kevin.pickard, Mark.stanger, Neola, Phillip.salzman, Rachel.sienko, Russ.sarbora, Steven.wood, Tom.dilatush, Vaughn.romero, Wallymarx

GlideSystem *Source*: http://wiki.servicenow.com/index.php?oldid=250606 *Contributors*: Amy.bowman, Anat.kerry, CapaJC, Chuck.tomasi, David.Bailey, Fred.luddy, Fuji.publishing.user, G.yedwab, George.rawlins, Guy.yedwab, Jim.uebbing, John.ramos, Joseph.messerschmidt, Mark.stanger, Neola, Phillip.salzman, Rachel.sienko, Steven.wood

GlideElement *Source*: http://wiki.servicenow.com/index.php?oldid=250298 *Contributors*: Amy.bowman, Andrew.Kincaid, CapaJC, Emily.partridge, Eric.jacobson, Fred.luddy, Fuji.publishing.user, G.yedwab, George.rawlins, Gflewis, Guy.yedwab, Jim.uebbing, John.ramos, Joseph.messerschmidt, Neola, Rachel.sienko, Steven.wood

GlideAggregate *Source*: http://wiki.servicenow.com/index.php?oldid=250293 *Contributors*: Anat.kerry, Don.Goodliffe, G.yedwab, George.rawlins, Guy.yedwab, Jim.uebbing, John.ramos, Joseph.messerschmidt, Mark.stanger, Matt.kilbride, Neola, Rachel.sienko, Steven.wood, Vaughn.romero, Vhearne

Script Includes *Source*: http://wiki.servicenow.com/index.php?oldid=250848 *Contributors*: Ajandersen, CapaJC, Chuck.tomasi, David Loo, Fuji.publishing.user, G.yedwab, George.rawlins, Guy.yedwab, Jim.uebbing, John.ramos, Joseph.messerschmidt, Mark.stanger, Neola, Phillip.salzman, Rachel.sienko, Rlandrum, Steven.wood, Vaughn.romero, Vhearne

Script Actions *Source*: http://wiki.servicenow.com/index.php?oldid=250846 *Contributors*: Eric.jacobson, Fuji.publishing.user, George.rawlins, Guy.yedwab, John.ramos, Joseph.messerschmidt, Neola, Steven.wood

Image Sources, Licenses and Contributors