

Tokyo API Reference

Last updated: July 15, 2023

Some examples and graphics depicted herein are provided for illustration only. No real association or connection to ServiceNow products or services is intended or should be inferred.

This PDF was created from content on docs.servicenow.com. The web site is updated frequently. For the most current ServiceNow product documentation, go to docs.servicenow.com.

Company Headquarters

2225 Lawson Lane
Santa Clara, CA 95054
United States
(408)501-8550

Scripting

Use scripts to extend your instance beyond standard configurations. With scripts, you may automate processes, add functionality, integrate your instance with an outside application and more.

APIs (Application Programming Interfaces) provide classes and methods that you can use in scripts to define functionality. ServiceNow provides APIs as JavaScript classes, web services, and other points of connection for integrations. Note that you cannot access commonly used JavaScript objects (such as DOM or Window). Jelly scripts are also used in some modules. Jelly is used to turn XML into HTML and may include both client-side and server-side scripts.

Scripts may be server-side (run on the server or database), client-side (run in the user's browser), or run on the MID server.

Note: When you are writing scripts, you cannot use [reserved words](#).

Understand JavaScript before you begin customizing your instance, and with Jelly if you intend to deploy Jelly scripts.

Server-side scripts

Perform database operations. For example, use a server-side script to update a record. Create a script in a scoped application or in the global scope. Each execution context includes a set of available APIs.

Scoped environment

Use scoped APIs when scripting in a scoped application. Scoped Glide APIs do not include all the methods included in the global Glide APIs, and you cannot call a global Glide API in a scoped application.

Global environment

The global scope is a special application scope that identifies applications developed prior to application scoping, or applications intended to be accessible to all other global applications. Use global APIs when scripting in the global scope.

To learn more about server-side scripting, see [Server-side scripting](#). To learn more about application scope, see [Application scope](#).

Client-side scripts

Make changes to the appearance of forms, display different fields based on values that are entered, or change other custom display options.

- onLoad client scripts run when the form or page is loaded
- onChange client scripts run when something specific gets changed AND also when the form or page loads
- onSubmit client scripts run when the form is submitted

Client Scripts can also be called by other scripts or modules, including UI policies. To learn more about client-side scripting, see [Client-side scripting](#).

- [Available script types](#)

Scripts can be used in many places. The most important detail is whether the script runs on the client or the server.

- [Glide class overview](#)

The ServiceNow Glide classes expose JavaScript APIs that enable you to conveniently work with tables using scripts.

- [Execution order of scripts and engines](#)

Scripts, assignment rules, business rules, workflows, escalations, and engines all take effect in relation to a database operation, such as insert or update. In many cases, the order of these events is important.

- [Script evaluation of fields by data type](#)

Script fields evaluate data based on the field type of the input.

- [Scripting alert, info, and error messages](#)

You can send messages to customers as alerts, informational messages, or error messages.

- [Using regular expressions in server-side scripts](#)

JavaScript regular expressions automatically use an enhanced regex engine, which provides improved performance and supports all behaviors of standard regular expressions as defined by Mozilla JavaScript. The enhanced regex engine supports using Java syntax in regular expressions.

- [JavaScript syntax editor](#)

The syntax editor provides support for editing JavaScript scripts.

- [HTML syntax editor](#)

The HTML syntax editor provides support for editing HTML and Jelly scripts and defines what's rendered when the page is displayed. The HTML syntax editor can contain either static XHTML or dynamically generated content defined as Jelly, and can call script includes and UI Macros.

- [Code editor](#)

The code editor provides support to use programming language services in a text editor and is used in scripts.

- [Server-side scripting](#)

Server scripts run on the server or database. They can change the appearance or behavior of ServiceNow or run as business rules when records and tables are accessed or modified.

- [Client-side scripting](#)

Run JavaScript on the client (web browser) when client-based events occur, such as when a form loads, after form submission, or when a field changes value.

- [Useful scripts](#)

Scripts that provide useful functionality not included in the core system.

- [Creating custom UI Pages and UI macros](#)

Use UI pages to create custom pages for an application and UI macros for custom controls or interfaces.

- [Debugging scripts](#)

Debug scripts using session logs and Now Platform debugging tools such as a walk-through script debugger and error messages that display in the UI.

- [Packages Call Removal tool](#)

The Packages Call Removal Tool provides modules to identify fields that might contain scripts, find scripts that contain Packages calls to ServiceNow Java classes, and to examine proposed script changes that eliminate those Packages calls.

Available script types

Scripts can be used in many places. The most important detail is whether the script runs on the client or the server.

Script types and where they run

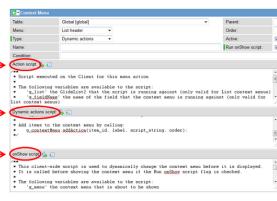
Script	Description	Runs on
Access Control	<p>Determines whether access will be granted for a specified operation to a specific entity.</p> <ul style="list-style-type: none">• type of entity being secured• operation being secured• unique identifier describing the object <p>Can be defined by roles, conditional expressions or scripts.</p>	server - script and any condition run on the server

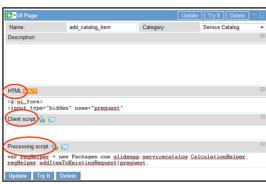
Script	Description	Runs on
Ajax Scripts	<p>Enables the client to get data from the server to dynamically incorporate into a page without reloading the whole page.</p> <ul style="list-style-type: none"> Ajax Client Scripts request that information be returned, or that action be taken, or sometimes both Ajax Server Scripts fulfill Ajax Client Script requests 	<ul style="list-style-type: none"> client - Ajax Client Scripts run on the client server - Ajax Server Scripts run on the server
Business Rules	<p>Customizes system behavior</p> <ul style="list-style-type: none"> runs when a database action occurs (query, insert, update or delete) the script may run <ul style="list-style-type: none"> before or after the database action is performed (runs as part of the database operation) asynchronously (at some point after the database operation) 	server - script and any condition run on the server

Script	Description	Runs on
	<ul style="list-style-type: none"> on display (when displaying the data in a form) 	
Service Catalog UI policies	<p>Defines the display of a variable set or a catalog item (from the service catalog).</p>	<ul style="list-style-type: none"> client - scripts in the "execute if true" field or "execute if false" field run on the client server - all conditions run on the server
Client Scripts	<p>Used for making changes to the appearance of forms, displaying different fields based on values that are entered or other custom display options.</p> <ul style="list-style-type: none"> onLoad means the Client Script runs when the form or page is loaded onChange means the Client Script runs when something specific gets changed AND also when the form or page loads onSubmit means the Client Script runs when the form is submitted 	client

Script	Description	Runs on
	Client Scripts can also be called by other scripts or modules, including UI policies.	
Script actions	<p>Contains scripts which run when an event occurs, for example</p> <ul style="list-style-type: none">• approval is cancelled• change is approved• problem is assigned <p>Can have a condition which must be true for the script to run. Commonly used to call a Script Include.</p>	server - script and any condition run on the server
Script Includes	<p>Contains scripts which can be functions or classes. These scripts run only when called by other scripts (often Business Rules).</p> <p>Any server script which is complicated or reusable should be a Script Include (especially complicated Business Rules).</p>	server

Script	Description	Runs on
Transform maps	<p>Used for importing data.</p> <ul style="list-style-type: none"> defines mapping relationships between tables can use Business Rules, other scripts and/or other options to import that data <p>Do not always include scripts.</p>	server
UI actions	<p>Creates the ability to choose a specific action such as clicking a button or a link.</p> <p>UI Actions put these items on forms and lists:</p> <ul style="list-style-type: none"> buttons links context menu items list choices 	<ul style="list-style-type: none"> client - when the "Client" box is checked, the script in the script field runs on the client server - when the "Client" box is unchecked, the script in the script field runs on the server client - when the "Client" box is checked, the onClick script is available, which can contain any JavaScript but normally calls a function which is specified in the script field

Script	Description	Runs on
		<ul style="list-style-type: none"> server - all conditions run on the server
UI Context Menus	<p>Defines which "right-click menu" will pop-up in which area, and the menu choices that will be available</p>  <p>Note: If you use a left-handed mouse configuration, right-click means "click the other button."</p>	<ul style="list-style-type: none"> client - onShow scripts run on the client client - action scripts run on the client server - dynamic action scripts run on the server server - all conditions run on the server
UI Macros	<p>Contains modular, reusable components that can contain Jelly and are called by UI pages. They also contain different types of scripts and may be called multiple times on the same page.</p> <p>Note: Jelly turns XML into HTML.</p>	<ul style="list-style-type: none"> server - the UI Macro itself executes on the Server server - may contain content that runs on the server (Jelly expressions or JavaScript inside Jelly constructs) client - may generate output that runs on the client (embedded)

Script	Description	Runs on
		JavaScript within <script> tags)
UI Pages	<p>Used to create and display pages, forms, dialogs, lists and other UI components. Can be displayed on a standalone basis, or called as a usable component, as part of a larger page.</p>  <p>Can contain</p> <ul style="list-style-type: none"> Client Scripts, processing scripts (which are server scripts), HTML, Jelly, UI Macros, and also can call other scripts. <p>Note: Jelly turns XML into HTML.</p>	<ul style="list-style-type: none"> server - Jelly XML runs on the server to produce HTML client - HTML may contain embedded JavaScript that runs on the client client - client scripts run on the client server - processing scripts run on the server

Script	Description	Runs on
UI Policies	<p>Defines the behavior and visibility of fields on a form.</p> <ul style="list-style-type: none"> • mandatory • visible • read only <p>Use UI Policies rather than client scripts whenever possible.</p> <ul style="list-style-type: none"> • UI Policies are always attached to one table • UI Policies often have a condition which must be true in order for them to run 	<ul style="list-style-type: none"> • client - scripts in the "execute if true" field or "execute if false" field run on the client • server - all conditions run on the server
UI Properties	<p>Designates what the instance will look like.</p>	<ul style="list-style-type: none"> • server - properties set on the server • client - the results get rendered on the client <p>no scripts</p>
UI Scripts	<p>Contains client scripts stored for re-use. Only used when called from other scripts.</p>	client

Script	Description	Runs on
	Not recommended for use.	
Validation Scripts	<p>Validates that values are in a specified format.</p> <p>For example, a validation script can verify that the only value allowed in a specific field is an integer.</p>	client
Workflow editor	<p>Used to create or change a workflow. Scripts can be run at any point in a workflow, or different scripts can be run at different points.</p> <p>Scripts also can be found inside every workflow activity and can be modified (although do so with extreme caution).</p>	server - script and any conditions run on the server

Glide class overview

The ServiceNow Glide classes expose JavaScript APIs that enable you to conveniently work with tables using scripts.

Using the Glide APIs, you can perform database operations without writing SQL queries, display UI pages, and define UI actions. The following

tables provide brief descriptions of the Glide classes and links to detailed information.

Server-side Glide classes

Class	Description
GlideRecord	Use this class for database operations instead of writing SQL queries. GlideRecord is a special Java class that can be used in JavaScript exactly as if it were a native JavaScript class. A GlideRecord is an object that contains records from a single table. See GlideRecord .
GlideElement	Use this class to operate on the fields of the current GlideRecord. See GlideElement .
GlideSystem	Use this class to get information about the system. See GlideSystem .
GlideAggregate	Use this class to perform database aggregation queries, such COUNT, SUM, MIN, MAX, and AVG, for creating customized reports or calculations in calculated fields. See GlideAggregate .
GlideDateTime	Use this class to perform date-time operations, such as date-time calculations, formatting a date-time, or converting between date-time formats. See GlideDateTime .

Client-side Glide Classes

Class	Description
GlideAjax	Use this class to execute server-side code from the client. See GlideAjax .
GlideDialogWindow	Use this class to display a dialog window. See GlideDialogWindow .
GlideForm	Use this class to customize forms. See GlideForm .
GlideList2	Use this class to customize (v2) lists, including normal lists and related lists. See GlideList2 .
GlideMenu	Use this class to customize UI Context Menu items. See GlideMenu .
GlideUser	Use this class to get session information about the current user and current user roles. See GlideUser .

- [Glide stack](#)

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).

Glide stack

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

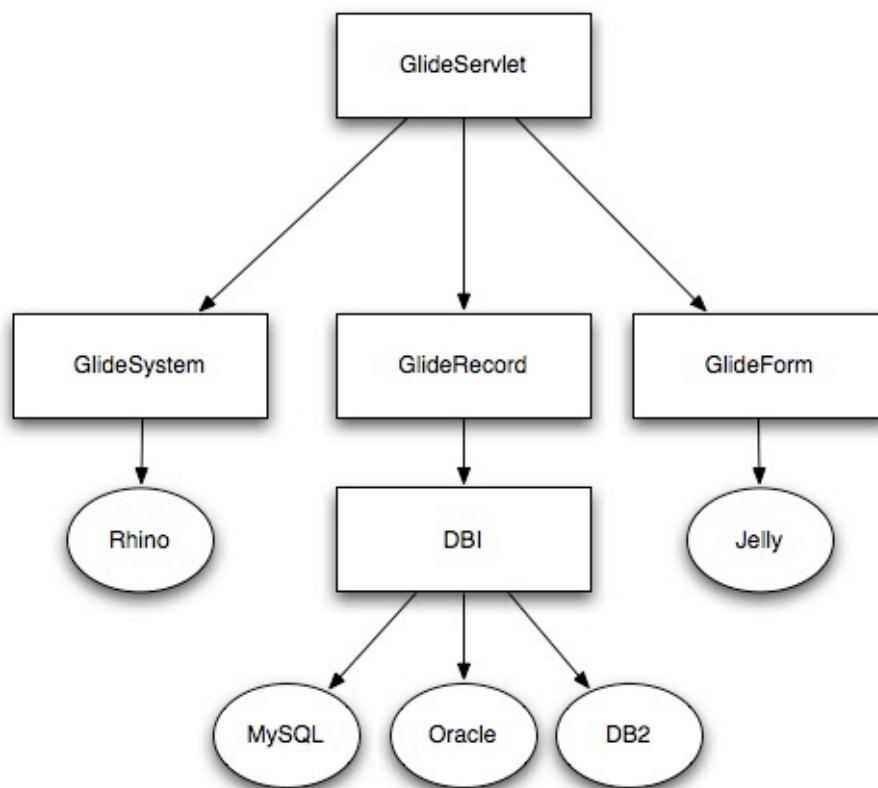
Java packages		Technologies used
	User Interface (Browser)	<ul style="list-style-type: none">AngularJSHTMLCSSJavaScript
com.glide.ui com.glide.jelly	GlideServlet	Apache Jelly
com.glide.script	Business Rules	Mozilla Rhino
com.glide.db	Persistence	JDBC

User interface stack technology map descriptions

Name	Description	Attributes
GlideServlet	The primary driver of Glide, and the only servlet in the system, is found in GlideServlet.java. The GlideServlet:	<ul style="list-style-type: none">Handles inbound action requestsRenders pagesMerges data with formsPresents to userInterfaces with script layer

Name	Description	Attributes
Business Rules		<ul style="list-style-type: none">• ECMA / JavaScript implementation based on Mozilla Rhino• Interfaces with persistence layer using JDBC recordset interface• Merges persistence layer meta-data with data for easy correlation
Persistence		<ul style="list-style-type: none">• Persistence means any store<ul style="list-style-type: none">• RDBMS• LDAP• File system• Uniform access regardless of store type• Provides QUID and meta-data capabilities• Interfaces presented to callers<ul style="list-style-type: none">• RecordSet• TableDescriptor

Name	Description	Attributes
Glide servlet		<ul style="list-style-type: none">• ElementDescriptor



Execution order of scripts and engines

Scripts, assignment rules, business rules, workflows, escalations, and engines all take effect in relation to a database operation, such as insert or update. In many cases, the order of these events is important.

Note: Client-based code that executes in the browser, using Ajax or running as JavaScript, will always execute before the form submission to the server.

The order of execution is as follows:

1. Before business rules: Scripts configured to execute before the database operation with an order less than 1000.
2. Before engines. The following are not executed in any specific order:
 - Approval engine (for task and sys_approval_approver tables)
 - Assignment rules engine (for task tables)
 - Data policy engine
 - Escalation engine
 - Field normalization engine
 - Role engine - keeps role changes in sync with sys_user_has_role table (for sys_user, sys_user_group, sys_user_grmember, and sys_user_role tables)
 - Execution plan engine (for task tables)
 - Update version engine - creates version entry when sys_update_xml entry is written (for sys_update_xml table)
 - Data lookup engine inserts or updates
 - Workflow engine (for default workflows)
3. Before business rules: Scripts configured to execute before the database operation with an order greater than or equal to 1000.
4. The data base operation (insert, update, delete).
5. After business rules: Scripts configured to execute after the database operation with an order less than 1000.
6. After engines. The following are not executed in any specific order:
 - Label engine
 - Listener engine
 - Table notifications engine

- Role engine - keeps role changes in sync with sys_user_has_role table (for sys_user, sys_user_group, sys_user_grmember and sys_user_role tables)
 - Text indexing engine
 - Update sync engine
 - Workflow engine (for deferred workflows)
 - Trigger engine (for all Flow Designer flows)
7. Email notifications. The following are executed based on the weight of the notification record:
- Notifications sent on an insert, update, or delete
 - Event-based notifications
8. After business rules (Only active records). Scripts configured to execute after the database operation with an order greater than or equal to 1000.

Script evaluation of fields by data type

Script fields evaluate data based on the field type of the input.

Evaluation of fields by data types

Type	Evaluates to in script	Example
String	The string	"dog" > "dog"
Decimal	A number with up to two decimal points	12.34 > 12.34
Integer	A number with zero decimal points	12 > 12
True / False	true or false	 > true

Type	Evaluates to in script	Example
		<input type="checkbox"/> > false
Date	A date formatted as yyyy-mm-dd	2008-11-04
Date-time	A day and time formatted as yyyy-mm-dd hh:mm:ss	2008-11-04 06:46:20
Duration	<p>A date that is equal to January 1st 1970 00:00:00 + the amount of time of the duration being stored</p> <p>Note: This date corresponds to the system time zone. If a different user time zone has been specified, the date and time value may appear different for that user.</p>	<input type="text"/> Days 00 <input type="text"/> Hours 00 : 00 : 00 > "1970-01-01 00:00:00" <input type="text"/> Days 1 <input type="text"/> Hours 02 : 03 : 04 > "1970-01-02 02:03:04"
Choice	<p>Returns the contents of the value field for the sys_choice record associated with that choice.</p> <p>See: Choice List for more information on returning the value associated with a particular item in a choice list.</p>	<input type="text"/> Incident state: <input type="text"/> New > "2" (Note that this value is a string)

Type	Evaluates to in script	Example
Journal	Returns a string of all entries made to that journal field. See Journal Fields for scripting of journal type fields	The web server is down > The web server is down
Reference	Returns the sys_id of the record that is referenced	> "287ee6fea9fe198100ada7950d0b1b73"
Image	Returns the path to the image	> images/icons/image_name.gif
URL	Returns a string	> "http://www.service-now.com"
Glide Lists	Returns a string of comma-separated Sys IDs	> 5137153cc611227c000 bbd1bd8cd2007,46d1 4f04a9fe19810142e40c 6b071512

Scripting alert, info, and error messages

You can send messages to customers as alerts, informational messages, or error messages.

Business rule and other general use scripts

Script	Result
<pre>current.field_name.setError("Hello World");</pre>	Adds Hello World below the specified field in a red error message.

Script	Result
<code>gs.addInfoMessage("Hello World");</code>	Adds Hello World at the top of the browser window in a blue info message.
<code>gs.addErrorMessage("Hello World");</code>	Adds Hello World at the top of the browser window in a red error message.
<code>gs.print("Hello World");</code>	Writes Hello World to the text log on the file system but not to the sys_log table in the database.
<code>gs.log("Hello World");</code>	Writes Hello World to the database and the log file. Note: <code>gs.log</code> can adversely affect performance if used too frequently.

Client side scripts

Script	Result
<code>alert("Hello World");</code>	Displays a window with Hello World and an OK button.
<code>confirm("Hello World");</code>	Displays a window with Hello World? and OK and Cancel buttons.
<code>g_form.showFieldMsg("field_name", "Hello World", "error");</code>	Adds Hello World below the specified field in a red error message.
<code>g_form.hideFieldMsg("field_name");</code>	Hides the most recent message for the specified field.

Important: The methods in this table are only for use in client scripts.

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

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 these properties.

Note: The client script alert option is not available for use with Automated Test Framework (ATF).

Error and alert text size properties

Property	Description
<code>css.outputmsg.info.text.fontSize</code>	Sets the size for info messages. Default is 11pt.
<code>css.outputmsg.error.text.fontSize</code>	Sets the size for error messages. Default is 11pt.

Using regular expressions in server-side scripts

JavaScript regular expressions automatically use an enhanced regex engine, which provides improved performance and supports all behaviors of standard regular expressions as defined by Mozilla JavaScript. The enhanced regex engine supports using Java syntax in regular expressions.

The SNC.Regex API is not available for scoped applications. For scoped applications, remove the SNC.Regex API and use standard JavaScript regular expressions.

For more information on JavaScript regular expressions, see the Mozilla JavaScript documentation on [regular expressions](#) and [RegExp](#).

- [Using Java syntax in JavaScript regular expressions](#)

The enhanced regex engine includes an additional flag to allow Java syntax to be used in JavaScript regular expressions.

- [Convert SNC Regex expressions to enhanced regex expressions](#)

When you upgrade to Eureka Patch 5 or later releases, you should convert scripts that use the SNC.Regex API to use regular JavaScript expressions.

Using Java syntax in JavaScript regular expressions

The enhanced regex engine includes an additional flag to allow Java syntax to be used in JavaScript regular expressions.

Regular expressions with the additional flag work in all places that expect a regular expression, such as `String.prototype.split` and `String.prototype.replace`. To use Java syntax in a regular expression, use the Java inline flag `j`, for example `/(?ims)ex(am)ple/j`

Extended regular expression flags

Flag	Description
<code>j</code>	Defines a regular expression that executes using the Java regular expression engine. It can be used to access Java-only features of regular expressions (such as look behind, negative look behind) or to use Java regular expressions without translating them into JavaScript regular expressions. For example: <code>var regex = /ex(am)ple/j;</code>

Convert SNC Regex expressions to enhanced regex expressions

When you upgrade to Eureka Patch 5 or later releases, you should convert scripts that use the `SNC.Regex` API to use regular JavaScript expressions.

Procedure

- From the original expression, such as: `SNC.Regex("/expr/is")`, create a new regular expression object using the pattern with the slashes stripped.

```
new RegExp('expr');
```

2. Move the SNC.Regex flags to the start of the expression using Java's inline flag special construct.

```
new RegExp('(?is)expr');
```

3. Add the j flag to the `RegExp` to tell the engine to treat the expression as a Java expression.

Note: If you know that the script being converted does not use Java syntax, it is not necessary to use the j flag.

```
new RegExp('(?is)expr', 'j');
```

4. Add the g flag to handle multiple matches or a global replace.

```
new RegExp('(?is)expr', 'jg');
```

Example

Using SNC.Regex

```
var r = new SNC.Regex('/world/');
var str = 'helloworld';
var replaced = r.replaceAll(str, 'there');
// replaced == 'hellothere'
```

Using a JavaScript regular expression

```
var r = new RegExp('world', 'jg');
var str = 'helloworld';
var replaced = str.replace(r, 'there');
// replaced == 'hellothere'
```

JavaScript syntax editor

The syntax editor provides support for editing JavaScript scripts.

The syntax editor has these features.

- JavaScript syntax coloring, indentation, line numbers, and automatic creation of closing braces and quotes
- JavaScript support

- Linting using the ESLint utility

Note: Modify or view default linting configurations by accessing the glide.ui.syntax_editor.linter.eslint_config property in the System Property [sys_properties] table. See [Available system properties](#) for more information.

- Context menu for script includes, API, and tables
- Script macros for common code shortcuts

This feature requires the Syntax Editor (com.glide.syntax_editor) plugin.

JavaScript editor

```
1 /**
2  * Service-now.com
3  * Description: Notification events for live messages
4  */
5
6 var live_feed_version = new LiveFeedUtil().getLiveFeedVersion();
7 if (live_feed_version == '1.0') {
8     if (current.operation() == "insert")
9         processLiveInsertEvents();
10 }
11
12 function processLiveInsertEvents() {
13     var msgUtil = new LiveMsgUtil(current);
14
15     //new message
16     if (current.reply_to.nil()) {
17         //only fire general event on public messages so notification subscribers
18         wont get private messages
19         if (!current.private_message)
20             ...
21     }
22 }
```

- Syntax editor plugin

Enable the syntax editor plugin to use the syntax editor.

- Context menu

Enable the context menu for script includes, Glide APIs, and tables in the JavaScript editor.

- Script syntax error checking

All 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.

- Searching for errors by line

To locate the exact position of the error in a large script, click the **Go to line** icon.

Syntax editor plugin

Enable the syntax editor plugin to use the syntax editor.

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

JavaScript syntax editor

```
1 /**
2  * Service-now.com
3  * Description: Notification events for live messages
4  */
5
6 var live_feed_version = new LiveFeedUtil().getLiveFeedVersion();
7 if (live_feed_version == '1.0') {
8     if (current.operation() == "insert")
9         processLiveInsertEvents();
10 }
11
12 function processLiveInsertEvents() {
13     var msgUtil = new LiveMsgUtil(current);
14
15     //new message
16     if (current.reply_to.nil()) {
17         //only fire general event on public messages so notification subscribers
18         //won't get private messages
19         if (!current.private_message)
20             ...
21     }
22 }
```

The syntax editor can be disabled or enabled by modifying the `glide.ui.javascript_editor` property in the `sys_properties.list`. In addition, administrators can configure the syntax editor to show error and warning indicators next to a line of code that contains an error by modifying the `glide.ui.syntax_editor.show_warnings_errors` property. For information on the `sys_properties.list`, refer to [Available system properties](#).

Note: Administrators can disable or enable the syntax editor for all users, regardless of user preference.

- [Syntax editor JavaScript support](#)

The syntax editor provides editing functions to support editing JavaScript scripts.

- [Syntax editor keyboard shortcuts and actions](#)

The syntax editor offers keyboard shortcuts and actions to assist in writing code.

- [Syntax editor macros](#)

Script macros provide shortcuts for typing commonly used code. To insert macro text into a script field, enter the macro keyword followed by the Tab.

- [Script macro maintenance](#)

Administrators can define new script macros or modify existing script macros.

Syntax editor JavaScript support

The syntax editor provides editing functions to support editing JavaScript scripts.

JavaScript editing functions

Icon	Keyboard Shortcut	Name	Description
	N/A	Toggle Syntax Editor	Disables the syntax editor. Click the button again to enable the syntax editor.

Icon	Keyboard Shortcut	Name	Description
	Access Key + 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. By default, the system automatically checks for syntax errors as you type in a script field. If an error or warning is found, the syntax editor displays a bullet beside the script line containing the error or warning. This check occurs on all script fields.
	Access Key + \	Start Searching	Highlights all occurrences of a search term in the script field and locates the first occurrence.

Icon	Keyboard Shortcut	Name	Description
			Click the icon, then enter the search term and press Enter . You can use regular expressions 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.

Icon	Keyboard Shortcut	Name	Description
			<p>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 / aaa/ locates aaa.</p> <p>2. Enter the replacement string and press Enter.</p>
	Access Key + ;	Replace All	<p>Replaces all occurrences of a text string in the script field.</p> <p>1. Click the icon, then enter the string to replace and press Enter. You can use regular expressions enclosed in slashes to define the</p>

Icon	Keyboard Shortcut	Name	Description
			<p>string to replace. For example, the term / a{3}/ locates aaa.</p> <p>2. Enter the replacement string and press Enter.</p>
	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.

JavaScript 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 white space 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.

JavaScript resources

Scripts use ECMA 262 standard JavaScript. Helpful resources include:

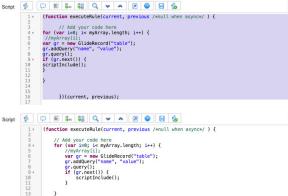
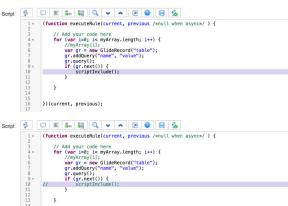
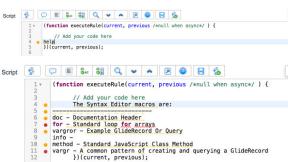
- Mozilla: http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference
- ECMA Standard in PDF format: <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>
- History and overview: <http://javascript.crockford.com/survey.html>
- JavaScript number reference: http://www.hunlock.com/blogs/The_Complete_Javascript_Number_Reference

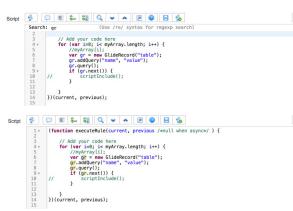
Syntax editor keyboard shortcuts and actions

The syntax editor offers keyboard shortcuts and actions to assist in writing code.

Syntax editor keyboard shortcuts and actions for writing code

Keyboard shortcut or action	Description	Example
Write code		

Keyboard shortcut or action	Description	Example
Enter an open parenthesis character after a valid class, function, or method name.	<p>Displays the expected parameters for the class or method.</p> <p>Enter the expected parameters as needed.</p>	
Toggle full screen mode Control+M	Switches between displaying the form with the full screen and displaying it normally.	
Format code <ul data-bbox="339 1142 605 1290" style="list-style-type: none"> Windows: Control+Shift+B Mac: Command+Shift+B 	Formats the selected lines to improve readability.	
Toggle comment <ul data-bbox="339 1438 605 1522" style="list-style-type: none"> Windows: Control+/- Mac: Command+/- 	Adds or removes the comment characters // from the selected lines.	
Insert macro text <ol data-bbox="355 1681 616 1803" style="list-style-type: none"> In the Script field, type the macro keyword text. For example <code>help</code>. 	Inserts macro text at the current position.	

Keyboard shortcut or action	Description	Example
2. Press Tab.		
Search		
<p>Start search</p> <ul style="list-style-type: none"> • Windows: Control+F • Mac: Command+F 	<p>Highlights all occurrences of a search term in the script field and locates the first occurrence.</p> <p>You can create regular expressions by enclosing the search terms between slash characters . For example, the search term /a{ 3 }/ locates the string aaa .</p>	 <pre data-bbox="971 1045 1264 1227"> Search: /a/ Script: 1 // (function executeSearch(current, previous, result) { 2 // Add your code here 3 for (var i=0; i<myArray.length; i++) { 4 var gr = myArray[i].getElementsByTagName("table"); 5 if (gr.length>0) { 6 gr[0].style.backgroundColor="yellow"; 7 gr[0].style.color="black"; 8 gr[0].style.fontWeight="bold"; 9 gr[0].style.fontSize="16px"; 10 if (gr[0].innerHTML.indexOf(result) > -1) { 11 gr[0].style.backgroundColor="red"; 12 gr[0].style.color="white"; 13 gr[0].style.fontWeight="normal"; 14 gr[0].style.fontSize="14px"; 15 } 16 } 17 } 18 (function executeSearch(current, previous, result) { 19 // Add your code here 20 for (var i=0; i<myArray.length; i++) { 21 var gr = myArray[i].getElementsByTagName("table"); 22 if (gr.length>0) { 23 gr[0].style.backgroundColor="yellow"; 24 gr[0].style.color="black"; 25 gr[0].style.fontWeight="bold"; 26 gr[0].style.fontSize="16px"; 27 if (gr[0].innerHTML.indexOf(result) > -1) { 28 gr[0].style.backgroundColor="red"; 29 gr[0].style.color="white"; 30 gr[0].style.fontWeight="normal"; 31 gr[0].style.fontSize="14px"; 32 } 33 } 34 } 35 })(current, previous); </pre>
<p>Find next</p> <ul style="list-style-type: none"> • Windows: Control+G • Mac: Command+G 	<p>Locates the next occurrence of the current search term in the script field. Use Start Searching to change the current search term.</p>	 <pre data-bbox="971 1404 1264 1467"> Search: /a/ Script: 1 // (function executeSearch(current, previous, result) { 2 // Add your code here 3 for (var i=0; i<myArray.length; i++) { 4 var gr = myArray[i].getElementsByTagName("table"); 5 if (gr.length>0) { 6 gr[0].style.backgroundColor="yellow"; 7 gr[0].style.color="black"; 8 gr[0].style.fontWeight="bold"; 9 gr[0].style.fontSize="16px"; 10 if (gr[0].innerHTML.indexOf(result) > -1) { 11 gr[0].style.backgroundColor="red"; 12 gr[0].style.color="white"; 13 gr[0].style.fontWeight="normal"; 14 gr[0].style.fontSize="14px"; 15 } 16 } 17 } 18 (function executeSearch(current, previous, result) { 19 // Add your code here 20 for (var i=0; i<myArray.length; i++) { 21 var gr = myArray[i].getElementsByTagName("table"); 22 if (gr.length>0) { 23 gr[0].style.backgroundColor="yellow"; 24 gr[0].style.color="black"; 25 gr[0].style.fontWeight="bold"; 26 gr[0].style.fontSize="16px"; 27 if (gr[0].innerHTML.indexOf(result) > -1) { 28 gr[0].style.backgroundColor="red"; 29 gr[0].style.color="white"; 30 gr[0].style.fontWeight="normal"; 31 gr[0].style.fontSize="14px"; 32 } 33 } 34 } 35 })(current, previous); </pre>
<p>Find previous</p> <ul style="list-style-type: none"> • Windows: Control+Shift+G • Mac: Command+Shift+G 	<p>Locates the previous occurrence of the current search term in the script field. Use Start Searching to change the current search term.</p>	 <pre data-bbox="971 1657 1264 1742"> Search: /a/ Script: 1 // (function executeSearch(current, previous, result) { 2 // Add your code here 3 for (var i=0; i<myArray.length; i++) { 4 var gr = myArray[i].getElementsByTagName("table"); 5 if (gr.length>0) { 6 gr[0].style.backgroundColor="yellow"; 7 gr[0].style.color="black"; 8 gr[0].style.fontWeight="bold"; 9 gr[0].style.fontSize="16px"; 10 if (gr[0].innerHTML.indexOf(result) > -1) { 11 gr[0].style.backgroundColor="red"; 12 gr[0].style.color="white"; 13 gr[0].style.fontWeight="normal"; 14 gr[0].style.fontSize="14px"; 15 } 16 } 17 } 18 (function executeSearch(current, previous, result) { 19 // Add your code here 20 for (var i=0; i<myArray.length; i++) { 21 var gr = myArray[i].getElementsByTagName("table"); 22 if (gr.length>0) { 23 gr[0].style.backgroundColor="yellow"; 24 gr[0].style.color="black"; 25 gr[0].style.fontWeight="bold"; 26 gr[0].style.fontSize="16px"; 27 if (gr[0].innerHTML.indexOf(result) > -1) { 28 gr[0].style.backgroundColor="red"; 29 gr[0].style.color="white"; 30 gr[0].style.fontWeight="normal"; 31 gr[0].style.fontSize="14px"; 32 } 33 } 34 } 35 })(current, previous); </pre>

Keyboard shortcut or action	Description	Example
<p>Replace</p> <ul style="list-style-type: none"> Windows: Control+E Mac: Command+E 	<p>Replaces the next occurrence of a text string in the script field.</p>	
<p>Replace all</p> <ul style="list-style-type: none"> Windows: Control+Shift+E Mac: Command+Shift+E 	<p>Replaces all occurrences of a text string in the script field.</p>	
<p>Help</p> <p>Help</p> <ul style="list-style-type: none"> Windows: Control+H Mac: Command+H 	<p>Displays the list of syntax editor keyboard shortcuts.</p>	
<p>Show description</p> <ul style="list-style-type: none"> Windows: Control+J Mac: Command+J 	<p>Displays API documentation for the scripting element at the cursor's current location.</p>	
<p>Show macros</p> <ol style="list-style-type: none"> In the Script field, type help. 	<p>Displays the list of available syntax editor macros as text within the script field.</p>	

Keyboard shortcut or action	Description	Example
2. Press Tab.		

Syntax editor macros

Script macros provide shortcuts for typing commonly used code. To insert macro text into a script field, enter the macro keyword followed by the Tab.

vargr

- Inserts a standard GlideRecord query for a single value.
- Output:

```
var now_GR = new GlideRecord("");
gr.addQuery("name", "value");
gr.query();
if (gr.next()) {

}
```

vargror

- Inserts a GlideRecord query for two values with an OR condition.
- Output:

```
var now_GR = new GlideRecord('');

var qc = gr.addQuery('field', 'value1');

qc.addOrCondition('field', 'value2');
gr.query();

while (gr.next()) {

}
```

for

- Inserts a standard recursive loop with an array.
- Output:

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

info

- Inserts a GlideSystem information message.
- Output:

```
gs.addInfoMessage("");
```

method

- Inserts a blank JavaScript function template.
- Output:

```
/* _____  
 * Description:  
 * Parameters:  
 * Returns:  
 _____ */  
: function() {  
},
```

doc

- Inserts a comment block for describing a function or parameters.
- Output:

```
/**  
 * Description:
```

```
* Parameters:  
* Returns:  
*/
```

Script macro maintenance

Administrators can define new script macros or modify existing script macros.

Before you begin

Role required: admin

About this task

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.

Procedure

1. Navigate to **All > System Definition > Syntax Editor Macros**.
2. Click **New** or select the macro to edit.
3. Define the macro details with the fields listed in the table below.

Editor macro fields

Field	Description
Name	Macro keyword text users type to insert macro text.
Comments	Description of the macro. This text appears when the user types help.
Text	Full macro text that replaces the name in the editor.

Context menu

Enable the context menu for script includes, Glide APIs, and tables in the JavaScript editor.

With the context menu options, your users can navigate to:

- Script include definitions
- Glide API documentation
- System and custom table definitions and data

In the syntax editor, bold font is used for tokens that have a context menu. Right-click the token to view context menu options. If you use a Mac, you can use the Command-click shortcut.

Context menu options

Token type	Context menu option	Description
Script include	Open Definition	Definition of the script include in a new window.
	Find References	List of files referencing the script include.

Token type	Context menu option	Description
		<p>Note:</p> <ul style="list-style-type: none"> To view the preview of the file, click the preview script icon . <p>To open the file in a new window, click Open File.</p> <ul style="list-style-type: none"> To view all files that reference the script include, click Show All Files.
Glide API	Show Documentation	Documentation page of the Glide API.
Table	Show Definition	Definition of the system or custom table in a new window.
	Show Data	Records in the table that are based on the role of the current user.

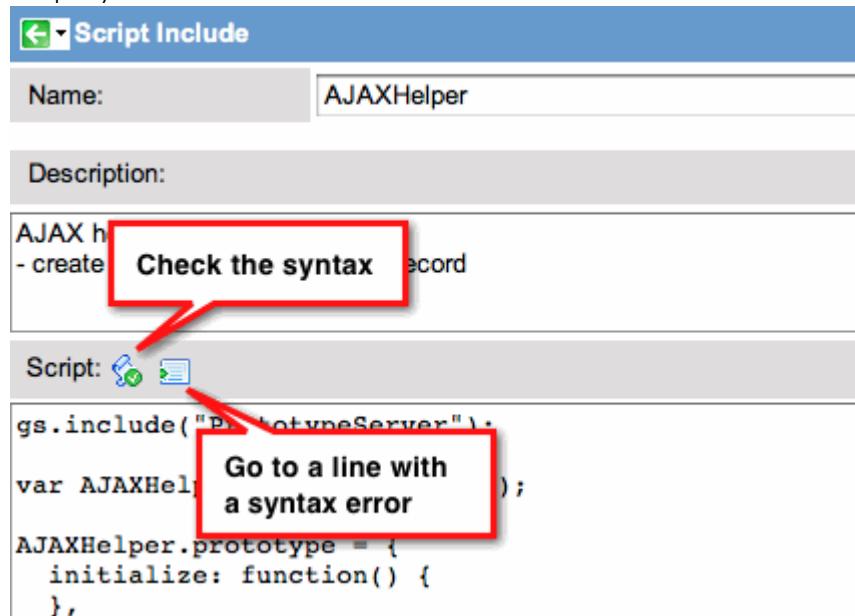
Enable or disable the context menu in the script editor using the `glide.ui.syntax_editor.context_menu` property in the System Property [sys_properties] table. See [Available system properties](#) for more information.

Note: Context menu options can be accessed only if the browser supports SharedWorker. For example, Google Chrome and Mozilla Firefox.

Script syntax error checking

All 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.

Script syntax check



The script editor notifies you of syntax errors in your scripts in the following situations.

- 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.

Script syntax error (short)

① **JavaScript parse error at line (10) column (34) problem = invalid label**

- Click 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.

Script syntax 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));
```

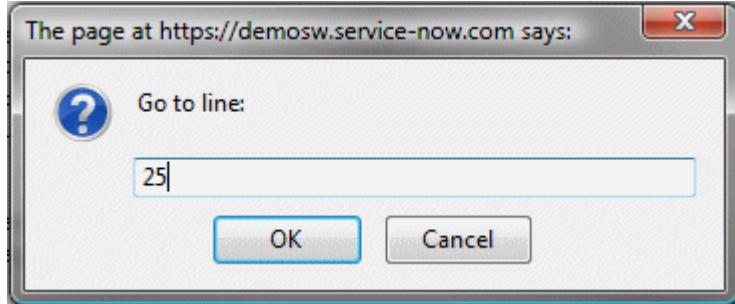
Searching for errors by line

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 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.

Go to script error



- [Navigate to a line number](#)

When the syntax editor is disabled, users can navigate to a specific line in the code using the Go to line icon ().

Navigate to a line number

When the syntax editor is disabled, users can navigate to a specific line in the code using the Go to line icon ().

Before you begin

Disable the Syntax Editor.

Role required: admin

Procedure

1. Click the Go to line icon ().

Note: This icon is not available when the editor is enabled.

2. Enter a number in the field and then press **Enter**.

HTML syntax editor

The HTML syntax editor provides support for editing HTML and Jelly scripts and defines what's rendered when the page is displayed. The HTML syntax editor can contain either static XHTML or dynamically generated content defined as Jelly, and can call script includes and UI Macros.

The syntax editor has these features.

- HTML and Jelly script support
- HTML and Jelly syntax coloring, indentation, line numbers, and automatic creation of closing braces and quotes
- Auto-suggestions for HTML and Jelly tags

Note: The keyboard shortcut is Ctrl+Space.

- Script macros for common code shortcuts

HTML syntax editor



```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
3  	<gui_form>
4  		<input type="hidden" name="selection_result" id="selection_result" value="" />
5  		<input type="hidden" name="my_sys_id" id="my_sys_id" value="${P.getWindowProperties().get('sys_id'))}" />
6  		<div style="text-align:center;">
7  			<div ${gs.getMessage('Are you sure you want to close this interaction?')}> </div><br/>
8
9  			<div align="right">
10  				<:dialog_buttons_ok_cancel ok_text="${gs.getMessage('Close')}" ok_title="${gs.getMessage('Close current interaction')}" ok="return actionOK();" cancel_text="${gs.getMessage('No')}" cancel_title="${gs.getMessage('Go back to record')}" cancel="return cancel();"/>
11
12
13 	</div>
14 	</gui_form>
15 </j:jelly>
```

HTML and Jelly editing functions

Icon	Keyboard shortcut	Name	Description
	N/A	Toggle syntax editor	Disables the syntax editor. Click the Toggle syntax editor icon () again to enable the syntax editor.
	Cmd+/	Toggle comment	Comments the selected code.
	Cmd+E	Replace	Replaces the next occurrence of a text string in the script field.

Icon	Keyboard shortcut	Name	Description
			<p>1. Click the Replace 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.</p> <p>2. Enter the replacement string and press Enter.</p>
	Cmd	Replace All	<p>Replaces all occurrences of a text string in the script field.</p> <p>1. Click the Replace all icon () , then enter the string to replace and press Enter. You can use regular</p>

Icon	Keyboard shortcut	Name	Description
			<p>expressions enclosed in slashes to define the string to replace. For example, the term / a{3} / locates aaa.</p> <p>2. Enter the replacement string and press Enter.</p>
	Cmd+F	Start Searching	<p>Highlights all occurrences of a search term in the script field and locates the first occurrence. Click the Start searching icon () , then enter the search term and press Enter.</p>
	Cmd+G	Find Next	<p>Locates the next occurrence of the current search term in the script field. Click the Start searching icon () to change the current search term.</p>

Icon	Keyboard shortcut	Name	Description
	Cmd+Shift+G	Find Previous	Locates the previous occurrence of the current search term in the script field. Click the Start searching icon () to change the current search term.
	Ctrl+M	Toggle Full Screen	Expands the script field to use the full form view for easier editing. Click the Toggle full screen icon () again to return to standard form view. This feature is not available for Internet Explorer.
	Cmd+H	Help	Displays the keyboard shortcuts help screen.
	N/A	Save	Saves changes without leaving the current view. Click the Save icon () in full screen mode to save without

Icon	Keyboard shortcut	Name	Description
			returning to standard form view.

Editing tips

- To insert a fixed space anywhere in your code, press Tab.
- To indent a single line of code, click in the leading white space 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.

Code editor

The code editor provides support to use programming language services in a text editor and is used in scripts.

The code editor has these features for the supported language services and [Inline scripts](#).

- Syntax coloring, indentation, line numbers, and automatic creation of closing braces and quotes
- Auto-suggestions and auto-completions

Code editor

The screenshot shows the ServiceNow code editor interface. At the top, there are buttons for 'Log Level' (set to 'info') and a search bar. Below that is a toolbar with icons for file operations. The main area is a code editor window titled 'Log Message'. It contains the following code:

```
1  /*
2   **Access Flow/Action data using the fd_data object. Script must return a value.
3   **example: var shortDesc = fd_data.trigger.current.short_description;
4   **return shortDesc;
5   */
6  return(math.sqrt(64));
```

The code is syntax-highlighted, with comments in grey and keywords in blue. A tooltip 'fb()' is visible near the bottom right of the editor window.

Editing tips

- To insert a fixed space anywhere in your code, press Tab.
- To indent a single line of code, click in the leading white space 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.
- To declare variables, use the `var` keyword so that they remain within the proper JavaScript scope.

Server-side scripting

Server scripts run on the server or database. They can change the appearance or behavior of ServiceNow or run as business rules when records and tables are accessed or modified.

Server-side Glide APIs (Application Programming Interfaces) provide classes and methods that you can use in scripts to perform server-side tasks.

Immediately invoked function expressions

The system uses immediately invoked function expressions when a script runs in a single context, such as in a [Create a transform map](#). Functions that run from multiple contexts use [Script includes](#) instead.

By enclosing a script in an immediately invoked function expression, you can:

- Ensure that the script does not impact other areas of the product, such as by overwriting global variables.
- Pass useful variables or objects as parameters.
- Identify function names in stack traces.
- Eliminate having to make separate function calls.

An immediately invoked function expression follows this format:

```
(function functionName(parameter) {  
    //The script you want to run  
})('value');//Note the parenthesis indicating this function should run.
```

You can declare functions within the immediately invoked function expression. These inner functions are accessible only from within the immediately invoked function expression.

```
(function functionName(parameter) {  
    function helperFunction(parameter) {//return some value}  
    var value = helperFunction(parameter); //Valid function call.  
    //perform any other script actions  
})('value');  
  
var value2 = helperFunction(parameter); //Invalid. This function is not accessible from outside the self-executing function.
```

- [Glide Server APIs](#)

ServiceNow provides APIs for the Glide Server.

- [Business rules](#)

A business rule is a server-side script that runs when a record is displayed, inserted, updated, or deleted, or when a table is queried.

- [Script includes](#)

Script includes are used to store JavaScript that runs on the server.

- [Processors](#)

Processors provide a customizable URL endpoint that can execute arbitrary server-side JavaScript code and produce output such as TEXT or JSON. Creating custom processors is deprecated.

- [Scripts - Background module](#)

Administrators can use the Scripts - Background module to run arbitrary JavaScript code from the server.

- [Installation settings](#)

Installation settings are global business rules with calculated names. Installation settings are calculated just before a record is displayed and facilitate dynamic determination of access and roles. Installation Settings permit the programmatic determination of a setting.

- [Using DurationCalculator to calculate a due date](#)

Using the DurationCalculator script include, you can calculate a due date, using either a simple duration or a relative duration base on schedules.

- [Querying tables in script](#)

Using methods in the GlideRecord API, you can return all records from a table, return records from a table that satisfy specific conditions, or return records that include a string from a single table or from multiple tables in a text index group.

- [Running order guides automatically](#)

Service catalog order guides allow customers to make a single service catalog request that can generate several ordered items. Administrators can configure order guides to run automatically, from a workflow or a script to generate a set of ordered items without manually submitting a service catalog request. Administrators can also review and reprocess the order guide failures.

- [Scriptable assignment of execution plans](#)

Each catalog item has an associated execution plan, used whenever an item of that type is ordered; if no plan is specified, the default plan is used. This default is effective for most organizations, but your execution plan may need to vary based on additional criteria.

- [Scriptable service catalog variables](#)

You can use scripting to reference any request item variable from a table in scoped and non-scoped environment.

- [Setting a GlideRecord variable to null](#)

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.

- [Schedule Pages](#)

A schedule page is a record that contains a collection of scripts that allow for custom generation of a calendar or timeline display.

- [XMLDocument script object](#)

A JavaScript object wrapper for parsing and extracting XML data from an XML document (String).

- [JavaScript engine on the platform](#)

The JavaScript engine that evaluates server-side scripts supports the ECMAScript 2021 (ES12) standard.

- [JavaScript API Context-sensitive help](#)

The syntax editor can display context-sensitive API information.

Glide Server APIs

ServiceNow provides APIs for the Glide Server.

GlideAggregate

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.

Note: This functionality requires a knowledge of JavaScript.

For additional information, refer to [GlideAggregate API](#).

GlideAggregate examples

GlideAggregate is an extension of GlideRecord and its use is probably best shown through a series of examples.

Note: This functionality requires a knowledge of JavaScript.

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 preceding example. If you want to get a count of the incidents that were open, simply add a query 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 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("There are currently "+ categoryCount +" incidents  
with a category of "+ category);}
```

The output is:

```
*** Script: There are currently 1.0 incidents with a category of Data  
*** Script: There are currently 11.0 incidents with a category of Enhancement  
*** Script: There are currently 1.0 incidents with a category of Implementation  
*** Script: There are currently 197.0 incidents with a category of inquiry  
*** Script: There are currently 13.0 incidents with a category of Issue  
*** Script: There are currently 1.0 incidents with a category of  
category of  
*** Script: There are currently 47.0 incidents with a category of request
```

The following is an example that uses multiple aggregations to see how many times records have been modified using 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 =
```

```
x = 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
```

The following is a 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.monthsAgoStar
t(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.monthsAgoS
tart(3)');
    agg2.addQuery('opened_at','<=','javascript:gs.monthsAgoE
nd(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:446
8.0
*** 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
```

The following is an example to obtain 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)
```

You can implement the SUM aggregate with or without the use of the groupBy() method. If you do not use the groupBy() method, the result of the SUM is the cumulative value for each different value of the field for which you request the SUM. For example, if you SUM the total_cost field in the Fixed Asset table, and the Fixed Asset table contains 12 total records:

- Three records with a total_cost of \$12
- Four records with a total_cost of \$10
- Five records with a total_cost of \$5

When you SUM the record set, the getAggregate() method returns three different sums: \$36, \$40, and \$25.

The following code illustrates implementing the SUM aggregate without using the groupBy() method:

```
var totalCostSum = new GlideAggregate('fixed_asset');
totalCostSum.addAggregate('SUM', 'total_cost');
totalCostSum.query();

while (totalCostSum.next()) {
    var allTotalCost = 0;
    allTotalCost = totalCostSum.getAggregate('SUM', 'total_cost');
    aTotalCost = totalCostSum.getValue('total_cost');
    gs.print('Unique field value: ' + aTotalCost + ', SUM =
' + allTotalCost + ', ' + allTotalCost/aTotalCost + ' records');
}
```

The output for this example is:

```
*** Script: Unique field value: 12, SUM = 36, 3 records
*** Script: Unique field value: 10, SUM = 40, 4 records
*** Script: Unique field value: 5, SUM = 25, 5 records
```

Using the same data points as the prior example, if you use the `groupBy()` method, the `SUM` aggregate returns the sum of all values for the specified field.

The following example illustrates implementing the `SUM` aggregate using the `groupBy()` method:

```
var totalCostSum = new GlideAggregate('fixed_asset');
totalCostSum.addAggregate('SUM', 'total_cost');
totalCostSum.groupBy('total_cost');
totalCostSum.query();
if(totalCostSum.next()) { // in case there is no result
    var allTotalCost = 0;
    allTotalCost = totalCostSum.getAggregate('SUM', 'total_cost');
    gs.print('SUM of total_cost: = ' + allTotalCost);
}
```

The output for this example is:

```
*** Script: SUM of total_cost: 101
```

GlideRecord

`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.

A `GlideRecord` contains both records (rows) and fields (columns). The field names are the same as the underlying database column names. For additional information, refer to [GlideRecord - Scoped](#).

Note: Use of `gs.sql()` scripting syntax was discontinued in Geneva. Use standard `GlideRecord` syntax in its place.

Using GlideRecordSecure

GlideRecordSecure is a class inherited from GlideRecord that performs the same functions as GlideRecord, and also enforces ACLs.

Non-writable fields

Be aware that, when using GlideRecordSecure, non-writable fields are set to NULL when trying to write to the database. By default, canCreate() on the column is replaced with canWrite() on the column. If that returns false, the column value is set to NULL.

Checking for NULL values

If an element cannot be read because an ACL restricts access, a NULL value is created in memory for that record. With GlideRecord, you must explicitly check for any ACLs that might restrict read access to the record. To do so, an if statement such as the following is required to check if the record can be read:

```
if ( !grs.canRead() ) continue;
```

With GlideRecordSecure, you do not need to explicitly check for read access using canRead(). Instead, you can use next() by itself to move to the next record. The following example provides a comparison between GlideRecord and GlideRecordSecure.

```
var count = 0;
var now_GR = new GlideRecord('mytable');
now_GR.query();
while (now_GR.next()) {
    if (!now_GR.canRead()) continue;
    if (!now_GR.canWrite()) continue;
    if (!now_GR.val.canRead() || !now_GR.val.canWrite())
        now_GR.val = null;
    else
        now_GR.val = "val-" + now_GR.id;
    if (now_GR.update())
        count++;
}
```

```
var count = 0;
var grs = new GlideRecordSecure('mytable');
grs.query();
```

```
while (grs.next()) {  
    grs.val = "val-" + grs.id;  
    if (grs.update())  
        count++;  
}
```

Examples

These are two simple examples using GlideRecordSecure.

```
var att = new GlideRecordSecure('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);  
canDelete;
```

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

GlideSystem

The GlideSystem API provides methods for retrieving information.

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.

For additional information, see [GlideSystem](#).

GlideDateTime

The GlideDateTime class provides methods for performing operations on GlideDateTime objects, such as instantiating GlideDateTime objects or working with glide_date_time fields.

In addition to the instantiation methods described below, a GlideDateTime object can be instantiated from a glide_date_time field using the getGlideObject() method (for example, var gdt = gr.my_datetime_field.getGlideObject();).

Some methods use the Java Virtual Machine time zone when retrieving or modifying a date and time value. Using these methods may result in unexpected behavior. Use equivalent local time and UTC methods whenever possible.

For additional information, refer to [GlideDateTime](#).

GlideDate and GlideDateTime examples

The GlideDate and GlideDateTime APIs are used to manipulate date and time values.

Note: This functionality requires a knowledge of JavaScript.

For additional information, refer to [GlideDate API](#) and [GlideDateTime API](#).

You can create a GlideDateTime object from a GlideDate object by passing in the GlideDate object as a parameter to the GlideDateTime constructor. By default, the GlideDateTime object is expressed in the internal format, yyyy-MM-dd HH:mm:ss and the system time zone UTC.

```
var gDate = new GlideDate();
gDate.setValue('2015-01-01');
gs.info(gDate);

var gDT = new GlideDateTime(gDate);
gs.info(gDT);
```

Output:

```
2015-01-01  
2015-01-01 00:00:00
```

Set a duration field value in script

Examples of JavaScript that can be used to set the value of a duration field.

Note: Negative duration values are not supported.

Using the GlideDateTime.subtract() method

The subtract(GlideDateTime start, GlideDateTime end) method in [GlideDateTime](#) enables you to set the duration value using a given start date/time and end date/time. An example on how to set the duration for the time a task was opened is:

```
var duration = GlideDateTime.subtract(start, end);
```

If you want to work with the value returned as a number to use in date or duration arithmetic, convert the return to milliseconds:

```
var time = GlideDateTime.subtract(start,end).getNumericValue();
```

If you want to set a duration to the amount of time between some event and the current date/time:

```
<duration_field> = GlideDateTime.subtract(new GlideDateTime(<start_time>.getValue()),gs.nowDateTime());
```

The time values presented to GlideDateTime.subtract are expected to be in the user's time zone and in the user's format.

Setting a default value of a duration field

Setting the default value for a duration field is similar to the method used in the previous topic.

Setting the duration field value in a client script

This script sets a duration_field value in a client script. Replace duration_field with the field name from your instance.

```
g_form.setValue('<duration_field>', '11 01:02:03');
```

Calculating and setting a duration using a client script

Here is an example of how to return a value and populate it using a client script.

Create an `onChange` client script that includes the following code. You can modify this script if you need the calculation to happen in an `onLoad` script or some other way.

```
function onChange(control, oldValue, newValue, isLoading){  
    var strt = g_form.getValue('<start_field>');  
    var end = g_form.getValue('<end_field>');  
    var ajax = new GlideAjax('AjaxDurCalc');  
        ajax.addParam('sysparm_name','durCalc');  
        ajax.addParam('sysparm strt',strt);  
        ajax.addParam('sysparm_end',end);  
        ajax.getXMLWait();  
        var answer = ajax.getAnswer();  
        g_form.setValue('<duration_field>', answer);}
```

Create a system script include file called AjaxDurCalc that handles the request. It may be reused for other functions as well.

```
var AjaxDurCalc = Class.create();  
AjaxDurCalc.prototype = Object.extendsObject(AbstractAjaxProcessor,{  
    durCalc:function(){return GlideDuration.subtract(this.getParameter('sysparm_start'),this.getParameter('sysparm_end'))}});
```

Changing the duration field value

If you manipulate a duration value with addition/subtraction of some amount of time, use the functions that allow you to get and set the numeric value of the duration. A unit of measure for a duration numeric value is milliseconds. The following is an example that adds 11 seconds to the duration field in the current record.

```
var timems = current.duration.dateNumericValue();  
timems = timems + 11*1000;  
current.duration.setDateNumericValue(timems);
```

Formatting the Resolve Time

To format the **Resolve Time** or the **Business Resolve Time** fields as durations, which displays them as a duration instead of a large integer, add the following attribute to those fields:

```
format=glide_duration
```

Modify the dictionary entry for the field and add the attribute. If there is an existing attribute, separate multiple attributes with commas.

Setting the maximum unit of measurement

The max_unit dictionary attribute defines the maximum unit of time used in a duration. For example, if `max_unit=minutes`, a duration of 3 hours 5 minutes 15 seconds appears as 185 minutes 15 seconds. To set the maximum unit of duration measurement, add the following dictionary attribute to the duration field:

```
max_unit=<unit>
```

Business rules

A business rule is a server-side script that runs when a record is displayed, inserted, updated, or deleted, or when a table is queried.

Use business rules to accomplish tasks like automatically changing values in form fields when certain conditions are met, or to create events for email notifications and script actions.

Note: Business rules can make use of scripts to take actions on records in the database. However, there are several other scripting options available on the platform, such as client scripts and UI actions.

How business rules work

To configure business rules, you first need to determine when the business rule should run and what action it should take.

When business rules run

Business rules run based on two sets of criteria:

- The time that the business rule is configured to run relative to a record being modified or accessed.
- The database operation that the system takes on the record.

The following options are provided to determine the time the business rule should run:

Time the business rule should run

Option	When the rule runs
Before	After the user submits the form but before any action is taken on the record in the database.
After	After the user submits the form and after any action is taken on the record in the database.
Async	When the scheduler runs the scheduled job created from the business rule. The system creates a scheduled job from the business rule after the user submits the form and after any action is taken on the record in the database. Note: Newly created business rules will run during upgrades.
Display	Before the form is presented to the user, just after the data is read from the database.

Note:

- Asynchronous business rules do not have access to the previous version of a record. Therefore, the changes(), changesTo(), and changesFrom() GlideElement methods do not work with async rule script. However, the condition builder and condition field (advanced view) both support the changes(), changesTo(), and changesFrom() methods.
- Business rules do not honor ACLs until you want them to be honored. For more information, see [Relationship between Business Rules and Access Control Rules \(ACLS\)](#)

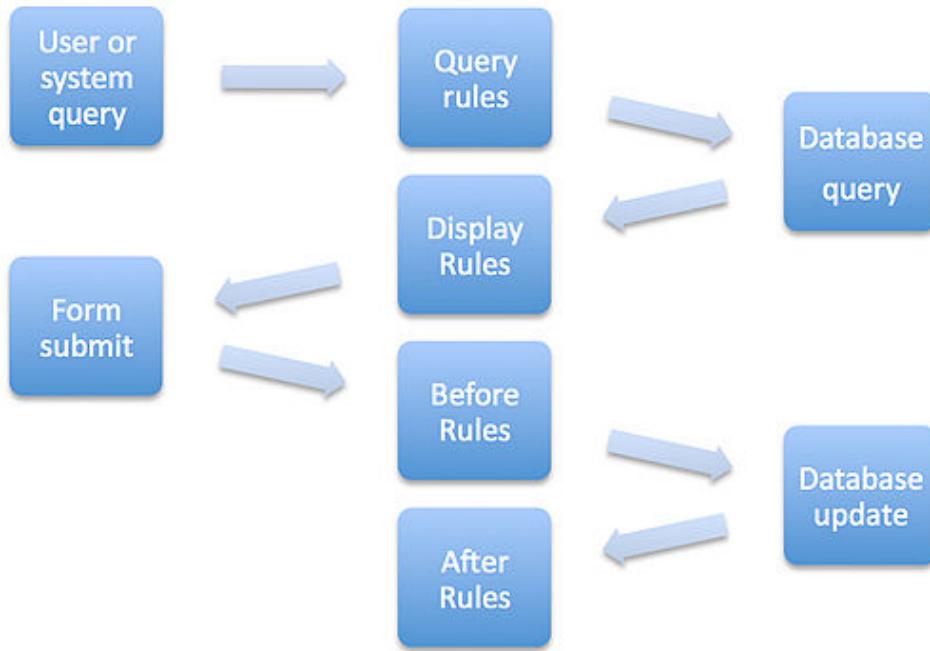
The following options are provided to determine the database operation that the system takes on the record:

Database operation that the system takes on the record

Option	When the rule runs
Insert	When the user creates a new record and the system inserts it into the database.
Update	When the user modifies an existing record.
Query	Before a query for a record or list of records is sent to the database. Typically you should use query for before business rules.
Delete	When the user deletes a record.

This image shows when different types of business rules run:

Business rule processing flow



Note: Business rules apply consistently to records regardless of whether they are accessed through forms, lists, or web services. This is one major difference between business rules and client scripts, which apply only when the form is edited.

Business rule actions

Business rules can perform a variety of actions. Common types of actions are:

- Changing field values on a form that the user is updating. Field values can be set to specific values available for that field, values copied from other fields, and relative values determined by the user's role.
- Displaying information messages to the user.
- Changing values of child tasks based on changes to parent tasks.
- Preventing users from accessing or modifying certain fields on a form.

- Aborting the current database transaction. For example, if certain conditions are met, prevent the user from saving the record in the database.

Administrators can set field values, create information messages, and abort transactions without writing a script.

Prevent recursive business rules

Avoid using `current.update()` in a business rule script. The `update()` method triggers business rules to run on the same table for insert and update operations, leading to a business rule calling itself over and over. Changes made in before business rules are automatically saved when all before business rules are complete, and after business rules are best used for updating related, not current, objects. When a recursive business rule is detected, the system stops it and logs the error in the system log. However, `current.update()` causes system performance issues and is never necessary.

You can prevent recursive business rules by using the `setWorkflow()` method with the `false` parameter. The combination of the `update()` and `setWorkflow()` methods is only recommended in special circumstances where the normal before and after guidelines mentioned above do not meet your requirements.

Related tasks

- [Create a business rule](#)

Business rules in scoped applications

Every business rule is assigned to either a private application scope or to the global scope.

The types of business rules you can create and how you can access those rules varies depending on the scope of the business rule and the scope of the table it runs on.

Note: The term global can refer to two different aspects of a business rule: the table it runs on and the scope it runs in. Business rules can either run on specific tables or be global. In addition, they can be in the global scope or in a private application scope.

Business rules on specific tables

Most business rules run on a specific table, which is defined in the **Table** field. You can create business rules on tables in the same scope and on tables that allow configuration records from another application scope.

For tables that are in a different scope than the business rule record, the types of rules are limited.

- You can create a rule where **When** is **async** with any of the following options:
 - **Insert**, **Update**, and **Delete** database operations. You cannot select **Query**.
 - **Set field values** actions and scripts (the **Script** field).
- You can create a rule where When is before with any of the following options:
 - **Insert**, **Update**, and **Delete** database operations. You cannot select **Query**.
 - **Set field values** actions only. You cannot write scripts and you cannot abort the database transaction.
- You cannot create any other types of business rules on tables in a different scope.

Business rules on specific tables cannot be accessed by other business rules or scripts.

Global business rules

Warning: Consider using script includes instead of global business rules. Script includes load only on request while global business rules load on every page in the system.

Global business rules are business rules where the **Table** field is set to **Global**. Global business rules may be accessible on multiple tables and from other scripts, depending on their scope protection. For a global business rule, define the scope protection by setting the **Accessible from** field:

- **This application scope only:** prevents applications in a different scope than the business rule from calling this business rule.

- **All application scopes:** allows any application to call this business rule.

Note: Global business rules do not support domain separation.

Scripts in scoped business rules

When you write a script in a business rule, you can access:

- Any script includes and global business rules in the same scope as the business rule.
- Script includes and global business rules that allow applications in a different scope to call them. To call functions from another scope, you must specify the scope of the function.
- For business rules in a unique scope, you can access the scoped system APIs only.

Create a business rule

You can create any type of business rule to run when a record is displayed, inserted, updated, or deleted, or when a table is queried.

About this task

Note: These instructions and examples provide general guidance for how to implement this functionality. For help with unique use cases, refer to the [Developer Community Forum](#), where you can ask questions, interact with other developers, and search for existing solutions.

Procedure

1. Navigate to **All > System Definition > Business Rules**.
2. Click **New**.
3. Fill in the fields, as appropriate.

Note: You might need to configure the form to see all fields.

Business Rule fields

Field	Description
Name	Enter a name for the business rule.
Table	Select the table that the business rule runs on. Note: The list shows only tables and database views that meet the scope protections for business rules. Business rules defined for a database view can run only on Query . A business rule for a database view cannot run on insert, update, or delete.
Application	Application that contains this business rule.
Accessible from	Scope protection for a global business rule. Note: This field is visible only when the Table field is set to Global . It does not apply to rules that run on specific tables.
Active	Select this check box to enable the business rule.
Advanced	Select this check box to see the advanced version of the form.
When to run	

Field	Description
When	<p>[Advanced] Select when this business rule should execute: display, before, async, or after the database operation is complete.</p> <p>Note: Consider setting the Priority for async business rules as the system uses this value when creating the associated scheduled job.</p> <p>Note: Newly created async business rules run automatically on upgrade.</p> <p>Note: Existing async business rules can be migrated to use the new async behavior.</p>
Order	[Advanced] Enter a number indicating the sequence in which this business rule should run. If there are multiple rules on a particular activity, the rules run in the order specified here, from lowest to highest.
Insert	Select this check box to execute the business rule when a record is inserted into the database.
Update	Select this check box to execute the business rule when a record is update.

Field	Description
Delete	[Advanced] Select this check box to execute the business rule when a record is deleted from the database.
Query	[Advanced] Select this check box to execute the business rule when a table is queried.
Filter Conditions	<p>Use the condition builder to determine when the business rule should run based on the field values in the selected Table. You can also use the Condition field to build a condition with a script.</p> <p>Note: Filters based on string compares are case-sensitive.</p>
Role Conditions	Select the roles that users who are modifying records in the table must have for this business rule to run.
Actions	
Set field values	<p>Set values for fields in the selected Table using the choice lists:</p> <ul style="list-style-type: none"> • The field • The assignment operator: <ul style="list-style-type: none"> • To: An exact value • Same as: The value of another field • To (dynamic): A value relative to the user

Field	Description
	configuring the business rule or a user with a specific role <ul style="list-style-type: none">The value
Add message	Select this check box and enter a message that appears when this business rule is run
Abort action	Select this check box to abort the current database transaction. For example, on a before insert business rule, if the conditions are met, do not insert the record into the database. If you select this option, you cannot perform additional actions on the record, such as setting field values and running scripts. You can still display a message to users by selecting the Add message check box and composing the message.
Advanced	
Condition	Create a JavaScript conditional statement to specify when the business rule should run. By adding the condition statement to this field, you tell the system to evaluate the condition separately and run the business rule only if the condition is true. If you decide to include the condition statement in the Script field or if you use the condition builder, leave this field blank. To

Field	Description
	have the instance reevaluate the condition statement a second time before running an async business rule, add the system property <code>glide.businessrule.async_condition_check</code> and set the value to true.
Script	<p>[Advanced] Create a script that runs when the defined condition is true. The system automatically populates this field with a function name that matches the When value.</p> <ul style="list-style-type: none">• onAfter• onAsync• onBefore• onDisplay <p>Note: The function name must match the When value.</p> <p>For more information and examples, see Example business rule scripts.</p>
Related list: Versions	
Versions	Shows all versions of the business rule. Use this list to compare versions or to revert to a previous version.

4. Click **Submit**.

Related reference

- [How business rules work](#)

Global variables in business rules

Predefined global variables are available for use in business rules.

Use the following predefined global variables to reference the system in a business rule script.

Global variable	Description
current	The current state of the record being referenced. Check for null before using this variable.
previous	The state of the referenced record prior to any updates made during the execution context, where the execution context begins with the first update or delete operation and ends after the script and any referenced business rules are executed. If multiple updates are made to the record within one execution context, previous will continue to hold the state of the record before the first update or delete operation. Available on update and delete operations only. Not available on asynch operations. Check for null before using this variable.
g_scratchpad	The scratchpad object is available on display rules, and is used to pass information to the client to be accessed from client scripts.

Global variable	Description
gs	References to GlideSystem functions.

The variables current, previous, and g_scratchpad are global across all business rules that run for a transaction.

Prevent null pointer exceptions

In some cases, there may not be a current or previous state for the record when a business rule runs, which means that the variables will be null. To check for null before using a variable, add the following code to your business rule:

```
if (current == null) // to prevent null pointer exceptions  
.  
return;
```

Define variables

User-defined variables are globally scoped by default. If a new variable is declared in an order 100 business rule, the business rule that runs next at order 200 also has access to the variable. This may introduce unexpected behavior.

To prevent such unexpected behavior, always wrap your code in a function. This protects your variables from conflicting with system variables or global variables in other business rules that are not wrapped in a function. Additionally, variables such as current must be available when a function is invoked in order to be used.

The following script is vulnerable to conflicts with other code. If the variable now_GR is used in other rules, the value of the variable may unexpectedly change.

```
var now_GR = new GlideRecord('incident');  
now_GR.query();  
while(now_GR.next()) {  
  
    //do something  
  
}
```

When this script is wrapped in a function, the variable is available only within the function and does not conflict with other functions using a variable named now_GR.

```
myFunction();  
  
function myFunction() {  
    var now_GR = new GlideRecord('incident');  
    now_GR.query();  
    while(now_GR.next()) {  
        //do something  
    } }
```

Use business rules and client scripts to control field values

Implement both business rules and client scripts for a field to enable users to set record values properly using both forms and lists, and to see immediate changes to the values in forms as edits are made.

The problem with using only a client script or a business rule to control updates to a field is that fields can be changed on either a form or a list. Client scripts and UI policies run on forms only (client-side) and do not apply to list editing. Allowing list editing with client scripts running on fields in a form can result in incorrect data being saved to the record. For systems in which client scripts or UI policies apply to forms, either disable list editing or create appropriate business rules or access control to control the setting of values in the list editor. A side effect of this is that security measures implemented in client scripts are easy to circumvent. The user only needs to edit the field in a list.

Business rules on a form are not dynamic, the user must update the record for the change to be seen. This makes using client scripts the preferred method for controlling field values on forms.

When using both a business rule and client script to control field values, the update behavior is the same across the system. This means that updated values are not different depending on whether a list or form is used to make the change. This means that the same functionality must be implemented twice, once in a client script and once in a business rule or access control.

Example: Use a business rule to create email addresses during user record import

An organization has a client script that sets the email address for a user to first.last@company.com. Administrators do this so they can see the email address immediately when they enter the user's information. The administrator then performs a bulk import of users from a spreadsheet containing the users' first and last names. The expectation is that each user's email address will be set automatically, as they are when they edit the form. Since the client script runs only on the form (the interface to the record), it has no effect on data imported into the record from outside that interface, and no email addresses are created. To solve this problem, the administrator implements a business rule that runs when the import occurs and creates the email addresses.

Example: Prevent list edit for a field that is not editable in the form

An organization wants to hide the **Priority** field on an incident form if the assignment group is **Development**. They create a UI policy on the incident form to do this, but their users can still see and edit the **Priority** field using the list editor. To rectify this, apply an access control to prevent read access to the **Priority** field when the assignment group is **Development**.

Using NULL as a field value

The string NULL has a particular role in scripts and is a reserved word.

The reserved word is NULL in all capital letters. A field with the value **Null** or **null**, for example, is acceptable. Only use NULL to clear out a particular field.

Any NULL field values obtained from an import set data source are inserted into the staging table as empty field values. You should not use the term NULL as a field value in import set transform maps or anywhere in the **First name** or **Last name** fields. Also, do not use NULL in reference fields as the system interprets the value as a string containing the word NULL, not as a reserved word.

Display business-rules

Display rules are processed when a user requests a record form.

The data is read from the database, display rules are executed, and the form is presented to the user. The current object is available and represents the record retrieved from the database. Any field changes are temporary since they are not yet submitted to the database. To the client, the form values appear to be the values from the database; there is no indication that the values were modified from a display rule. This is a similar concept to calculated fields.

The primary objective of display rules is to use a shared scratchpad object, g_scratchpad, which is also sent to the client as part of the form. This can be useful when you need to build client scripts that require server data that is not typically part of the record being displayed. In most cases, this would require a client script making a call back to the server. If the data can be determined prior to the form being displayed, it is more efficient to provide the data to the client on the initial load. The form scratchpad object is an empty object by default, and used only to store name:value pairs of data.

To populate the form scratchpad with data from a display rule:

```
// From display business rule
g_scratchpad.someName = "someValue";
g_scratchpad.anotherName = "anotherValue";

// If you want the client to have access to record fields
// not being displayed on the form
g_scratchpad.created_by = current.sys_created_by;

// These are simple examples, in most cases you'll probably
// perform some other
// queries to test or get data
```

To access the form scratchpad data from a client script:

```
// From client script
if(g_scratchpad.someName == "someValue") {
    //do something special
}
```

Task Active State Management business rule

This business rule determines whether the active field value needs to change based on changes to the **State** field.

The Task Active State Management business rule is executed when the **State** is changed for a task record. Its execution order is 50, and it runs before most other task business rules.

If the current task table has the close_states attribute defined on its table, or if it is inherited from a higher-level table, then the rule determines whether the active field needs to change. This is done by comparing the previous and current state values.

- If the state changes from an active state to an inactive state, the **Active** field is set to false.
- If the state changes from an inactive state to an active state, the **Active** field is set to true, effectively re-activating or re-opening the task.

It is recommended that you leverage the (current.active.changesTo([true/false])) action in your business rule, as opposed to creating rules on each task table that mark tasks as inactive or active.

Example business rule scripts

Find an example business rule script that helps you with a requirement of your organization.

Note: These instructions and examples provide general guidance for how to implement this functionality. For help with unique use cases, refer to the [Developer Community Forum](#), where you can ask questions, interact with other developers, and search for existing solutions.

Compare date fields in a business rule

It is possible to compare two date fields or two date and time fields in a business rule, and abort a record insert or update if they are not correct.

For example, you may want a start date to be before an end date. The following is an example script:

```
if ((!current.u_date1.nil()) && (!current.u_date2.nil()))
{
    var start = current.u_date1.glideObject().getNumericValue();
    var end = current.u_date2.glideObject().getNumericValue();
```

```
var end = current.u_date2.glideObject().getNumericValue();
if (start > end) {
    gs.addInfoMessage('start must be before end');
    current.u_date1.setError('start must be before end') ;
    current.setAbortAction(true);
}
```

This example has been tested in global scripts, and may need changes to work in scoped scripts. In addition to possibly needing API changes, security is more strict in scoped scripts.

As a good practice, make the business rule a before rule for insert and update actions. In the example script:

- u_date1 and u_date2 are the names of the two date fields. Replace these names with your own field names.
- The first line checks that both fields actually have a value.
- The next two lines create variables that have the dates' numerical values.
- The next two lines create different alert messages for the end user: one at the top of the form and one by the u_date1 field in the form.
- The last line aborts the insert or update if the date fields are not correct.

Here is a more complex example of the above comparison. If you have more than one pair of start and end dates, you can use arrays as shown. Additionally, this script requires the input dates to be within a certain range, in this case, no fewer than 30 days in the past and no more than 365 days in the future.

```
// Enter all start and end date fields you wish to check,
as well as the previous values
// Make sure that you keep the placement in the sequence the same for all pairs
var startDate = new Array(current.start_date,current.work_start);
var prevStartDate = new Array(previous.start_date,previous.work_start);
var endDate = new Array(current.end_date,current.work_end)
;
var prevEndDate = new Array(previous.end_date,previous.work_end)
```

```
k_end);

// The text string below is added to the front of ' start
must be before end'
var userAlert = new Array('Planned','Work');

// Set the number of Previous Days you want to check
var pd = 30;
// Set the number of Future Days you want to check
var fd = 365;

// You shouldn't have to modify anything below this line

var nowdt = new GlideDateTime();
nowdt.setDisplayValue(gs.nowDateTime());
var nowMs = nowdt.getNumericValue();
var pdms = nowMs;

// Subtract the product of previous days to get value in m
illiseconds
pdms -= pd * 24 * 60 * 60 * 1000;
var fdms = nowMs;

// Add the product of future days to get value in miliseco
nds
fdms += fd * 24 * 60 * 60 * 1000;
var badDate = false;

// Iterate through all start and end date / time fields
for (x = 0; x < startDate.length; x++) {
    if ((!startDate[x].nil()) && (!endDate[x].nil())) {
        var start = startDate[x].getGlideObject().getNumericVa
lue();
        var end = endDate[x].getGlideObject().getNumericValue(
);
        if (start > end) {
            gs.addInfoMessage(userAlert[x] + ' start must be bef
ore end');
            startDate[x].setError(userAlert[x] + ' start must b
e before end');
            badDate = true;
        } else if ((prevStartDate[x]) != (startDate[x])) {
            if (start < pdms) {
                gs.addInfoMessage(userAlert[x] + ' start must be
```

```
fewer than ' + pd + ' days ago');
    startDate[x].setError(userAlert[x] + ' start must be fewer than ' + pd + ' days ago');
    badDate = true; } }
else if ((prevEndDate[x]) != (endDate[x])) {
    if (end > fdms) {
        gs.addInfoMessage(userAlert[x] + ' end must be fewer than ' + fd + ' days ahead');
        endDate[x].setError(userAlert[x] + ' end must be fewer than ' + fd + ' days ahead');
        badDate = true ;
    } } }
if (badDate == true ) {
    current. setAbortAction ( true ) ; }
```

Parse XML payloads

Fields in XML format can be parsed with the system's getXMLText function.

Fields that get inserted into the database in XML format, such as the payload of an `ecc_event` row, can be parsed with the system's `getXMLText` function. The `getXMLText` function takes a string and an XPATH expression. For example:

```
var name = gs.getXMLText("<name>joe</name>", "//name");
```

returns the string 'joe'.

Assuming that the field "payload" contains XML, the function call might look like:

```
var name = gs.getXMLText(current.payload, "//name");
```

For information on XPATH, visit [w3schools](#).

Abort a database action in a before business-rule

In a before business rule script, you can cancel or abort the current database action using the `setAbortAction()` method.

For example, if the before business rule is executed during an insert action, and you have a condition in the script that calls `current.setAbortAction(true)`, the new record stored in `current` is not created in the database. The business rule continues to run after

calling `setAbortAction()` and all subsequent business rules will execute normally. Calling this method only prevents the database action from occurring.

You can use the `isActionAborted()` method to determine if the current database action (insert, update, delete) is going to be aborted. `isActionAborted()` is initialized for new threads and the `next()` method explicitly sets its value to false.

Note: `setAbortAction()` can only be executed from the same scope as the record whose action is being aborted. `current.setAbortAction` is not honored if executed in a business rule that is defined in a different scope.

Determine the operation that triggered the business rule

You can write a script for a business rule that is triggered on more than one database action.

If you want the business rule script to dynamically branch depending on the action that triggered the event, you can use the `operation()` function. For example:

```
if(current.operation() == "update") {  
    current.updates++; }  
if(current.operation() == "insert") {  
    current.updates = 0; }
```

Use an OR condition in a business rule

An **OR** condition can be added to any query part within a business rule.

An **OR** condition can be added to any query part within a business rule with the `addOrCondition()` method. The example below shows a query for finding all the incidents that have either a 1 or a 2 priority. The first `addQuery()` condition is defined as a variable and is used in the **OR** condition.

```
var inc = new GlideRecord('incident');  
var qc = inc.addQuery('priority','1');  
qc.addOrCondition('priority','2');  
inc.query();  
while(inc.next()) {
```

```
// processing for the incident goes here  
}
```

The following script is a more complex example, using two query condition variables doing the equivalent of (priority = 1 OR priority = 2) AND (impact = 2 OR impact = 3). The results of the **OR** condition are run with two variables, qc1 and qc2. This allows you to manipulate the query condition object later in the script, such as inside an **IF** condition or **WHILE** loop.

```
var inc = new GlideRecord('incident');  
var qc1 = inc.addQuery('priority','1');  
qc1.addOrCondition('priority','2');  
var qc2 = inc.addQuery('impact','2');  
qc2.addOrCondition('impact','3');  
inc.query();  
while(inc.next()) {  
    // processing for the incident goes here  
}
```

Reference a Glide list from a business rule

A field defined as a glide list is an array of values stored in 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]);  
}
```

Output:

```
*** Script: Reference value is: 62826bf03710200044e0bfc8bc  
be5df1  
*** Script: Reference value is: c2826bf03710200044e0bfc8bc  
be5d45  
*** Script: Reference value is: 5f74e421c0a8010e01ec0d74a7  
ee2cc6  
*** Script: Reference value is: 06826bf03710200044e0bfc8bc  
be5d57
```

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]);  
}
```

Output:

```
*** Script: Display value is: Abel Tuter  
*** Script: Display value is: Ashley Leonesio  
*** Script: Display value is: Charles Beckley  
*** Script: Display value is: Cherie Fuhri
```

Use `indexOf("searchString")` to find a string in a Glide list

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)`

Lock user accounts

You can lock user accounts if the user is not active.

The following business rule script locks user accounts if the user is not active in the LDAP directory or the user does not have self-service, itil, or admin access to the instance.

```
// Lock accounts if bcNetIDStatus != active in LDAP and user does not
// have self-service, itil or admin role
var rls = current.accumulated_roles.toString();
if(current.u_bcnetidstatus == 'active' && (rls.indexOf(',itil,') > 0 ||
    rls.indexOf(',admin,') > 0 ||
    rls.indexOf(',ess,') > 0 )) {
    current.locked_out = false;
} else {
    current.locked_out = true;

var now_GR = new GlideRecord("sys_user");
now_GR.query();
while(now_GR.next()) {
    now_GR.update();
    gs.info("updating " + gr.getDisplayValue());
}
```

Default before-query business rule

You can use a query business rule that executes before a database query is made.

Use this query business rule to prevent users from accessing certain records. Consider the following example from a default business rule that limits access to incident records.

- Name: incident query
- Table: Incident
- When: before, query
- Script:

```
if(!gs.hasRole("itil") && gs.isInteractive()) {  
    var u = gs.getUserID();  
    var qc = current.addQuery("caller_id",u).addOrCondition(  
        "opened_by",u).addOrCondition("watch_list","CONTAINS",u);  
    gs.print("query restricted to user: " + u); }
```

This example prevents users from accessing incident records unless they have the itil role, or are listed in the **Caller** or **Opened by** field. So, for example, when self-service users open a list of incidents, they can only see the incidents they submitted.

Note: You can also use access controls to restrict the records that users can see.

Script includes

Script includes are used to store JavaScript that runs on the server.

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.

Consider using script includes instead of global business rules because script includes are only loaded on request.

See [Privacy settings on client-callable script includes](#) and [Discovery script includes](#) for more information.

Script include form

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. View existing or create a new script include using the Script Include form.

To access script includes, navigate to **System Definitions > Script Includes**.

Script include form

Field	Description
Name	The name of the script include. If you are defining a class, this

Field	Description
	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.
API Name	Read-only and auto-populated API name.
Client callable	Makes the script include available to client scripts, list/report filters, reference qualifiers, or if specified as part of the URL. When selected, the Access Controls Related Link is available. See Privacy settings on client-callable script includes for more information.
Application	The application where this script include resides.
Accessible from	<p>Sets which applications can access this script include:</p> <p>All application scopes</p> <p>Can be accessed from any application scope.</p> <p>This application scope only</p> <p>Can be accessed only from the current application scope.</p>
Active	Enables the script include when selected. Deselect the active field to disable the script include.
Description	Provides descriptive content regarding the script include.

Field	Description
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.
Created by	The user who created this script include.
Updated by	The user who most recently updated this script include.
Protection policy	Sets the level of protection for the 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 the script include. Protected Provides intellectual property protection for application developers. Customers who download the script include

Field	Description
	cannot see the contents of the script field. The script is encrypted in memory to prevent unauthorized users from seeing it in plain text.
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 .
Access Controls	Becomes available when the Client callable check box is selected and is hidden from standard script includes. Use to protect a CCSI against unauthorized use when public access is not granted.

Use script includes

Script includes are found under System Definition or System UI. You can call existing script includes from a script or create a new script include.

To create an entirely new script include, you can follow the format of any of the existing script includes. In this 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 a code snippet with the class and prototype properly set up.

```
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](#).

Client-callable script includes

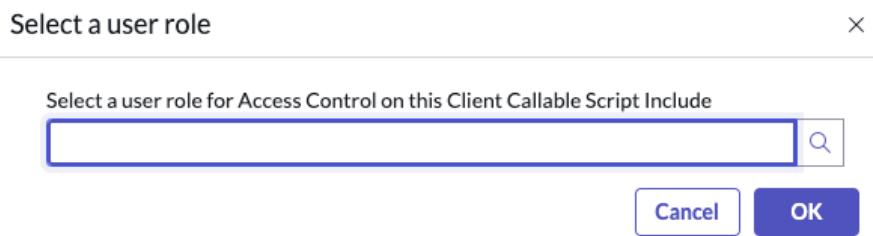
Client Callable Script Includes (CCSI) make the script include available to client scripts, list/report filters, reference qualifiers, or if specified as part of the URL.

Before you begin

Role required: admin

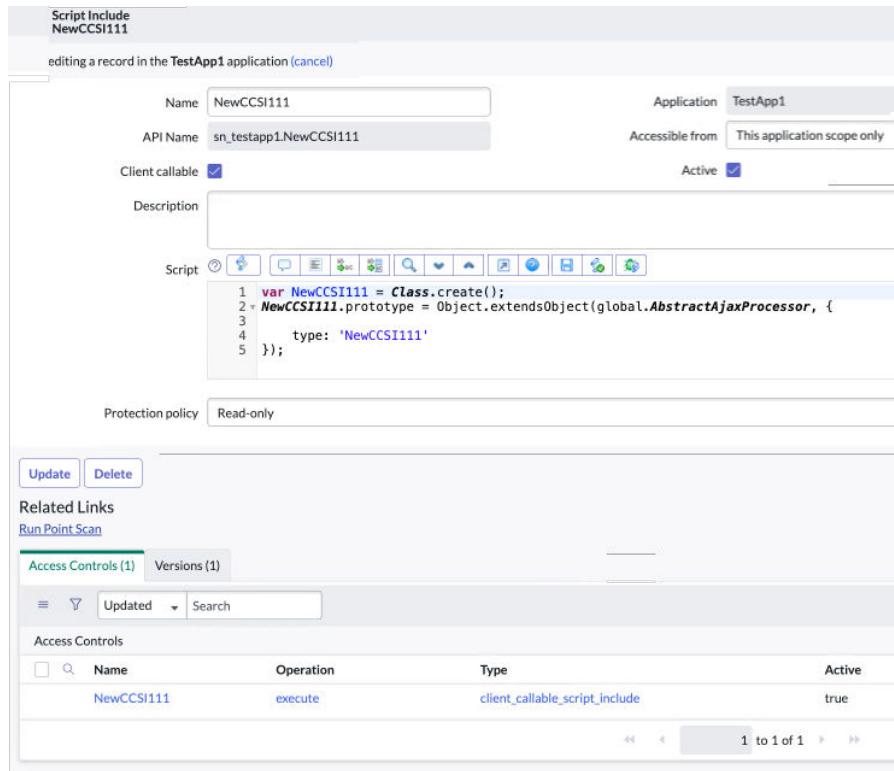
Procedure

1. Navigate to **System Definition > Script Includes**.
2. Select **New** or select an existing script include for viewing or editing.
See [Use script includes](#) for additional information on writing script includes.
3. Complete the form and select the **Client callable** check box.
A role selector pops up to select a user role and automatically create an Access Control entry. Select a user role and click **OK**.



Note: To disable the role selector window, set the `glide.script.ccsi.enable_acl_create_ux` to **false**.

A new CCSI record with a role-based Access Control is created. The Access Control Related Link becomes available with the selection of the **Client callable** check box.



Privacy settings on client-callable script includes

Privacy settings for client-callable script includes (CCSI) determine who can access a client-callable script include.

Private privacy-setting

The private privacy-setting means that guests who access public pages cannot access the client-callable script-include. A private script cannot be executed by a non-logged-in user.

Public privacy-setting

A public privacy-setting means that the client script can be executed by non-logged-in users that create an appropriate HTTP request. This can create a security problem if the client script provides confidential information.

The following script includes remain public by default because [Make UI pages public or private](#) need to access them:

- GlideSystemAjax
- SysMessageAjax
- KnowledgeMessagingAjax
- KnowledgeAjax
- PasswordResetAjax

Set privacy on all client-callable script includes

Change the privacy setting on all client-callable script includes.

To provide further control over all client-callable script includes, administrators can add the glide.script.ccsi.ispublic property. This property changes the visibility of client-callable script includes by making them all public or private. Configure the property as follows:

Configure property

Title	Property
Name	glide.script.ccsi.ispublic
Type	true false
Value	false

Note: To learn more about this property, see [Privacy on client-callable script includes](#) in Instance Security Hardening Settings.

Change privacy on a single client callable script include

Change the privacy setting for a single client-callable script include by adding the `isPublic()` function.

The `isPublic()` setting takes precedence over the `glide.script.ccsi.ispublic` property. For example, if the property is set to **false**, making all client-callable script-includes private, and a script sets `isPublic()` to **true**, the script is public.

To change the privacy for a single client-callable script include, add the following method to the script include:

```
isPublic:function(){return[true/false];},
```

Example

Make the `NewInclude` client script private.

```
var NewInclude =Class.create();

NewInclude.prototype={
    initialize:function() {},
    myFunction:function(){//Put function code here},
    isPublic:function(){return false;},
    type:'NewInclude'}
```

Security on client callable script includes

Protect your client callable script include (CCSI) against unauthorized use. For all CCSI records which are created a customer application, recommendations display that may help reduce security risk.

When creating a CCSI, the system displays the following security recommendations if they have not yet been configured:

- Add or define an Access Control, unless the CCSI has public access.
- Use `GlideRecordSecure` instead of `GlideRecord` API for better security, if the script queries the database.

Script Include
NotSecureCCSI

ⓘ We recommend Client Callable Script Include use GlideRecordSecure instead of GlideRecord API [More Info](#)

ⓘ We recommend you add a role based Access Control to the Client Callable Script Include [More Info](#)

Note: To disable the security recommendation messages, set the property `glide.script.ccsi.customerScoped.security_msgs_enabled` to **false** in the `sys_properties` table. The default value is set to **true**.

See [Instance Security Hardening Settings](#) for additional information on security compliance.

Discovery script includes

Discovery script includes define JavaScript classes that you can use to accomplish Discovery tasks.

Note: Users with `discovery_admin` role can write script includes. Follow best practices for server-side and client-side scripting to prevent security issues. See knowledge article [KB0550828](#) for more information.

Using GlideRecordUtil to Work with GlideRecords

`GlideRecordUtil` is a utility class that provides methods that are useful for working with `GlideRecords` during Discovery. Refer to [GlideRecordUtil](#) for descriptions of available methods.

Getting a GlideRecord Instance

To get a `GlideRecord` instance for a given configuration item, and of the correct class and table, use the `getCIGR(sys_id)` method. For example, the following code gets the `GlideRecord` of a CI with the `sys_id` of `2dfd7c8437201000deeabfc8bcbe5d56`:

```
var now_GR = new GlideRecordUtil().getCIGR("2dfd7c8437201000deeabfc8bcbe5d56");
```

To retrieve any hierarchical table without knowing its class type, use the `getGR(base_table, sys_id)` method. For instance, to get a `GlideRecord`

for a computer class CI, you might have to distinguish whether it is a computer class, Windows server, or Linux server class. Using this method guarantees a GlideRecord with the correct class. Different classes have different attributes. In this use case, a Windows server has attributes different from a Linux server. The following example shows how to get a GlideRecord in the correct class with its attributes.

```
var now_GR = new GlideRecordUtil().getGR("cmdb_ci_computer", "2dfd7c8437201000deeabfc8bcbe5d56");
```

Getting All the Fields In a GlideRecord

The getFields(now_GR) method returns a JavaScript object, such as a hashmap, of all the fields or attributes that exist in a given GlideRecord.

```
var now_GR = new GlideRecordUtil().getGR("cmdb_ci_computer", "2dfd7c8437201000deeabfc8bcbe5d56");
var fields = new GlideRecordUtil().getFields(now_GR);
gs.log(fields.join(" ")); // List all the fields that are in a computer CI
```

Populating GlideRecord Object Fields

The populateFromGR(hashmap, gr, ignore) method enables you to take a GlideRecord object and populate its fields and values into a JavaScript object. The third argument (ignore) is an optional JavaScript object that enables you to exclude certain fields. For example, you may not care about sys_created_by or sys_updated_by fields in a GlideRecord.

```
var objectToPopulate = {};
var now_GR = new GlideRecordUtil().getGR("cmdb_ci_computer", "2dfd7c8437201000deeabfc8bcbe5d56");
var ignore = {"sys_created_on": true, "sys_updated_by": true};
new GlideRecordUtil.03().populateFromGR(objectToPopulate, gr, ignore);
// Now the objectToPopulate contains field/value pairs from the computer GlideRecord
```

The mergeToGR(hashmap, gr, ignore) method enables you to populate a GlideRecord with a field/value-paired object. The ignore argument stops specified fields from being updated. The following code example updates a computer record's name and os fields, but does not update the sys_created_by field:

```
var now_GR = new GlideRecordUtil().getGR("cmdb_ci_computer", "2dfd7c8437201000deeabfc8bcbe5d56");
var obj = {"name": "xyz", "os": "windows 2000", "sys_created_by": "aleck.lin"};
var ignore = {"sys_created_by": true};
new GlideRecordUtil().mergeToGR(obj, gr, ignore);
gr.update();
```

Getting Table Hierarchies

The `getTables(table)` method returns a list of table hierarchies, as shown in the following example:

```
var tables = new GlideRecordUtil().getTables("cmdb_ci_linux_server");
gs.log(tables.join(","));
// The result would be "cmdb_ci, cmdb_ci_computer, cmdb_ci_server, cmdb_ci_linux_server".
```

Using DiscoveryException and AutomationException

When writing Discovery sensors and sensor-related scripts, you may want to use `DiscoveryException` or `AutomationException` to indicate that an exception has come from Discovery.

The `DiscoveryException` script include extends `AutomationException`, which extends the `GenericException` class. The following example uses `DiscoveryException` to throw an exception:

```
function foo() {
    if(//condition matches) throw new DiscoveryException("The message", "The cause"); }
```

The first argument takes the message of the exception and the second argument (optional) takes the cause of the exception. You may also want to catch the exception and log it as shown in the example below:

```
try {
    foo();
}
catch(e) {
    if(e instanceof DiscoveryException)
        gs.log("A DiscoveryException occurred. It is " + e.getMessage() + " caused by " + e.getCause()); }
```

The above example also applies for AutomationException. DiscoveryException is typically used to provide exception processing specifically for Discovery, while AutomationException is used for exception processing that applies to both Orchestration and Discovery.

Processors

Processors provide a customizable URL endpoint that can execute arbitrary server-side JavaScript code and produce output such as TEXT or JSON. Creating custom processors is deprecated.

Note: This feature is deprecated. While legacy, custom processors will continue to be supported, creating new custom processors has been deprecated. Instead, please use the [Scripted REST APIs](#). The following information is left in the documentation for existing processors only.

Warning: When creating a processor, ensure that you use parameter names that are specific to your processor. For example, if your processor exports a list of legal records, and a necessary parameter is the recipient's email address, don't use "email" as the parameter name. Create a more processor specific parameter name, such as legal_export_recipient_email. Failure to do so, and using instance parameter names, such as id, table, sys_id, service, catalog_id, or view (and others), can cause unexpected results.

When to create processors

Do not create custom processors. This feature is deprecated. Please use the REST APIs instead of creating custom processors. The remaining information is left for existing processors only.

Processor form

Field	Description
Name	Unique name of the processor.

Field	Description
Type	<p>Programming language of the processor script.</p> <p>Options include:</p> <ul style="list-style-type: none">• java: do not select this option• script
Application	Application containing this record.
Active	Flag to enable or disable the record.
CSRF protect	Option to protect the processor from running unless the instance uses a CSRF token.
Description	Description of the processor's function or purpose.
Parameters	<p>List of available input parameters.</p> <p>Specify parameter values in the URL as <parameter name>=<parameter value>.</p>

Field	Description
	<p>Note: Parameter names must be processor-specific. Do not choose common parameter names that another processor might use. If you use a common parameter name, such as <code>id</code>, <code>sys_id</code> or <code>table</code> in a processor, it can break other functionality, since the processor wins when that parameter exists in a URL. For example, a processor with an <code>id</code> parameter, regardless of the <code>Path</code> value in the same record, breaks the Service Portal, which depends on that parameter for page identification.</p>
Path	<p>URI path used to call this processor.</p> <p>Call a processor from the URL as:</p> <pre>https://<instance name>.service-now.com/ <Path>.do</pre>
Script	<p>Immediately Invoked Function Expression to run when the system calls this processor.</p> <p>The function automatically provides input parameters for the following API objects.</p> <ul style="list-style-type: none">• <code>g_request</code>• <code>g_response</code>

Field	Description
Protection policy	<ul style="list-style-type: none">g_processor <p>Policy to use to protect this record's script.</p> <p>Options include:</p> <ul style="list-style-type: none">NoneRead-onlyProtected

- Processor API components

Processors have access to dedicated API classes, objects, and methods.

- Secure and protect a processor

You can protect your processor against unauthorized use by using role restrictions, and protect it by requiring a CSRF token.

- Create a simple processor

Create a simple processor to execute a script from a URL query. This feature is deprecated.

- Create a multi-table processor

Create a multi-table processor that reports the number of rows in any table on your instance. This feature is deprecated.

- Create a custom processor

You can create a custom processor to execute a script from a URL query. This feature is deprecated.

Processor API components

Processors have access to dedicated API classes, objects, and methods.

Processor API components

Class, object, or method	Description
<code>g_response</code>	An object of type <code>HttpServletResponse</code> . See GlideServletResponse .
<code>setContentType('text/html; charset=UTF-8')</code>	A GlideServletResponse method to set the content type of the response being sent to the client.
<code>g_request</code>	An object of type <code>HttpServletRequest</code> . See HttpServletRequest .
<code>getParameter()</code>	A glide method to get the value of a URL parameter.
<code>canRead()</code>	A GlideRecord method to determine if the user can read data from a table. See GlideRecord .
<code>g_processor</code>	A simplified servlet for processors. See GlideScriptedProcessor .
<code>writeOutput()</code>	A GlideScriptedProcessor method to display information on the client.
<code>g_target</code>	An object containing the target table name of a processor URL. For example, a processor containing the URL <code>incident.do</code> applies to the <code>Incident</code> table.

Secure and protect a processor

You can protect your processor against unauthorized use by using role restrictions, and protect it by requiring a CSRF token.

About this task

Note: This feature is deprecated. While legacy, custom processors will continue to be supported, creating new custom processors has been deprecated. Instead, please use the [Scripted REST APIs](#). The following information is left in the documentation for existing processors only.

You can re-use a table's user role restrictions to protect it from access by your processor. This protection method assumes the processor will access table data.

Procedure

1. Create or select a user role that has access to the table the processor script calls.
2. Navigate to **System Definition > Processors**.
3. In **Script**, add the following code block.

```
var now_GR = new GlideRecord('your_table_name');
// canRead() compares the table's ACL to the user making this request, and returns true if the logged-in user has read access to this table
if(gr.canRead())
{
    // Perform table query here
    g_processor.writeOutput('Success!');
} else {
    g_processor.writeOutput('You do not have permission to read table your_table_name');
}
```

4. Update the code block to use other access restrictions as needed.

Available access functions include:

- canCreate()
- canRead()
- canWrite()
- canDelete()

5. Click **Update**.

Protect a processor with a CSRF token

You can protect a processor by requiring a CSRF token.

About this task

Script type processors can require a CSRF token check before the processor runs.

Procedure

1. Navigate to **All > System Definition > Processors**.
2. Open a processor record.
3. Select the **CSRF protect** option.
4. Click **Update**.

Create a simple processor

Create a simple processor to execute a script from a URL query. This feature is deprecated.

About this task

Note: This feature is deprecated. While legacy, custom processors will continue to be supported, creating new custom processors has been deprecated. Instead, please use the [Scripted REST APIs](#). The following information is left in the documentation for existing processors only.

The following steps assume that you have your own demonstration instance.

Procedure

1. Navigate to **All > System Definition > Processors**.
2. Click **New**.
3. Enter the following information.

Field	Value
Name	Hello
Type	Script
Path	Hello
Script	<pre>var name= g_request.getParameter("name"); g_processor.writeOutput("text/plain","Hello "+name);</pre>

4. Click **Submit**.
5. Enter a URL query to the instance with the following format: `https://instance.service-now.com/processor_name.do?parameter=value`. For example: `https://<instancename>.service-now.com/Hello.do?name=world`.

Create a multi-table processor

Create a multi-table processor that reports the number of rows in any table on your instance. This feature is deprecated.

About this task

Note: This feature is deprecated. While legacy, custom processors will continue to be supported, creating new custom processors has been deprecated. Instead, please use the [Scripted REST APIs](#). The following information is left in the documentation for existing processors only.

The multi-table processor protects itself from performance and security violations by confirming that the user is authorized to read the table. It does not report on certain tables that are too large to query safely.

Procedure

1. Navigate to **All > System Definition > Processors**.
The list of processors appears.
2. Click **New**.
3. Enter the following information.

Name	TableSize
Type	Choose Javascript
Description	Return number of records in a table
Parameters	SIZE
Path	<leave empty>

Script

```
g_response.setContentType('text/html;charset=UTF-8');
if(g_target === 'sys_email' || g_target === 'sys_log')
{
    g_processor.writeOutput(g_target + ' table is too large to quickly count');
} else {
```

```
var count = new GlideAggregate(g_target);
if( count.canRead() ) {
    count.addAggregate('COUNT');
    count.query();
    var records = 0;
    if (count.next()) {
        records = count.getAggregate('COUNT');
    }
    g_processor.writeOutput('table ' + g_target + ' '
has ' + records + ' records');
    } else {
        g_processor.writeOutput('You do not have access
to table ' + g_target);
    }
}
```

4. Click **Save**.

5. Test the new processor by entering the following URLs:

<https://<instancename>.service-now.com/incident.do?SIZE>
https://<instancename>.service-now.com/sys_user.do?SIZE
Your instance reports the number of records in the table. For example, table incident has 82 records.

Create a custom processor

You can create a custom processor to execute a script from a URL query. This feature is deprecated.

About this task

Note: This feature is deprecated. While legacy, custom processors will continue to be supported, creating new custom processors has been deprecated. Instead, please use the [Scripted REST APIs](#). The following information is left in the documentation for existing processors only.

When complete, you will be able to:

- Create a new processor
- Run a script from a URL query

The following example steps assume you have your own demonstration instance.

Procedure

1. Navigate to **All > System Definition > Processors**.

2. Click **New**.

3. For **Name**, enter Hello.

4. For **Type**, select script.

5. For **Path**, enter Hello.

6. For **Script**, enter the following code.

```
var name = g_request.getParameter("name");
g_processor.writeOutput("text/plain", "Hello " + name);
```

7. Click **Submit**.

8. Logout of the instance and open a new browser window.

9. Enter a URL query to the instance with the following format: `https://instance.service-now.com/processor_name.do?parameter=value`.
For example: `https://<instance name>.service-now.com>Hello.do?name=world`.

10. When prompted, enter credentials for valid user.

Scripts - Background module

Administrators can use the Scripts - Background module to run arbitrary JavaScript code from the server.

The Scripts - Background module consists of the following components.

- A text field to enter JavaScript
- A selector to specify the application scope
- A **Run script** button
- A list of available scripts

- A **Record for rollback?** check box. Selected by default, this creates a rollback context for the script execution. After the script is run, click the **Script execution and recovery available here** link to go to the **Script Execution History** form where you can rollback the script.

Administrators can run any valid JavaScript that uses the [Glide API](#). The system displays results, information, and error messages at the top of the screen.

Note: Running free-form JavaScript can cause system disruption or data loss. Do not run free-form scripts from a production instance.

By default, administrators can access this module without elevating privileges. If you want to require elevated privileges to access this module, set the system property `glide.script_processor.admin` to `security_admin`.

Installation settings

Installation settings are global business rules with calculated names. Installation settings are calculated just before a record is displayed and facilitate dynamic determination of access and roles. Installation Settings permit the programmatic determination of a setting.

Installation settings controlling access to fields and records are:

- `CanRead()`
- `CanWrite()`
- `CanCreate()`
- `CanDelete()`

Functions can return true if access is permitted, false if not. No return value uses the permission calculated using roles. The function has access to the current record through the variable `current code`.

The name of the function checking the permission on a record is formed by prefixing the setting name with the record name:

```
record_nameCanRead()
```

Similarly, the permission on a field in a record is formed by prefixing the function name with the record name, underscore, and field name:

```
record_name_field_nameCanRead()
```

Naming examples:

```
function incidentCanWrite() {} // can user write to this
record?
function incident_numberCanWrite() {} // can user write
to the number field?
```

This sample business rule restricts the writing of the name field in the sys_dictionary file when the entry exists:

```
// the element name cannot be written unless this is a n
ew record (not yet in database)
function sys_dictionary_nameCanWrite() {
    if (current.isNewRecord())
        return;

    return false;
}
```

Using DurationCalculator to calculate a due date

Using the DurationCalculator script include, you can calculate a due date, using either a simple duration or a relative duration base on schedules.

The following script demonstrates how to use the global API [DurationCalculator](#) to calculate a due date. The first part of the script illustrates how to set a start datetime using the `setStartTime()` method and then use the `calcDuration()` method to determine a due date that is "x" amount of continuous time (seconds) from the specified start datetime. The second half of the script illustrates how to use DurationCalculator to calculate a due date based on a schedule. Schedules enable you to apply a "filter" on future time, such as only including the days in a work week within the calculation. For example, if you apply a schedule "weekdays" (which only includes Monday through Friday) to your duration calculation, and the start datetime is Friday at 5:00 pm, when you add a duration of two days, your due date would be Tuesday at 5:00 pm. If you did not use a schedule, your due date would be Sunday at 5:00 pm. For additional information on schedules, see [Creating and using schedules](#).

This script can be cut and pasted into the Scripts Background page and run as is. It can also serve as an example for authoring business rules, UI actions, or used any other place that server-side script can be authored.

```
/*
 * Demonstrate the use of DurationCalculator to compute a
due date.
 *
 * You must have a start date and a duration. Then you ca
n compute a
 * due date using the constraints of a schedule.
 */

gs.include('DurationCalculator');
executeSample();

/**
 * Function to house the sample script.
 */
function executeSample(){

    // First we need a DurationCalculator object.
    var dc = new DurationCalculator();

    // ----- No schedule examples -----
    //

    // Simple computation of a due date without using a sc
heme. Seconds
    // are added to the start date continuously to get to
a due date.
    var gdt = new GlideDateTime("2012-05-01 00:00:00");
    dc.setStartTime(gdt);
    if(!dc.calcDuration(2*24*3600)){ // 2 days
        gs.log("*** Error calculating duration");
        return;
    }
    gs.log("calcDuration no schedule: " + dc.getEndDateTim
e()); // "2012-05-03 00:00:00" two days later

    // Start in the middle of the night (2:00 am) and comp
ute a due date 1 hour in the future
    // Without a schedule this yields 3:00 am.
    var gdt = new GlideDateTime("2012-05-03 02:00:00");
```

```
dc.setStartTime(gdt);
if(!dc.calcDuration(3600)){
    gs.log("*** Error calculating duration");
    return;
}
gs.log("Middle of night + 1 hour (no schedule): "+ dc.
getEndDateTime()); // No scheduled start date, just add 1
hour

// ----- Add a schedule to the date calculato
r -----
addSchedule(dc);

// Start in the middle of the night and compute a due
date 1 hour in the future.
// Since we start at 2:00 am the computation adds the
1 hour from the start
// of the day, 8:00am to get to 9:00am
var gdt = new GlideDateTime("2012-05-03 02:00:00");
dc.setStartTime(gdt);
if(!dc.calcDuration(3600)){
    gs.log("*** Error calculating duration");
    return;
}
gs.log("Middle of night + 1 hour (with 8-5 schedule): "
" + dc.getEndDateTime()); // 9:00 am

// Start in the afternoon and add hours beyond quitin
g time. Our schedule says the work day
// ends at 5:00pm, if the duration extends beyond that
, we roll over to the next work day.
// In this example we are adding 4 hours to 3:00pm whi
ch gives us 10:00 am the next day.
var gdt = new GlideDateTime("2012-05-03 15:00:00");
dc.setStartTime(gdt);

if(!dc.calcDuration(4*3600)){
    gs.log("*** Error calculating duration");
    return;
}
gs.log("Afternoon + 4 hour (with 8-5 schedule): " + dc
.getEndDateTime()); // 10:00 am.

// This is a demo of adding 2 hours repeatedly and exa
```

```
mine the result. This
    // is a good way to visualize the result of a due date
    // calculation.
    var gdt = new GlideDateTime("2012-05-03 15:00:00");
    dc.setStartTime(gdt);
    for(var i=2; i<24; i+=1){
        if(!dc.calcDuration(i*3600)){
            gs.log("*** Error calculating duration");
            return;
        }
        gs.log("add "+ i +" hours gives due date: " + dc.get
    EndDateTime());
    }

    // Setting the timezone causes the schedule to be interpreted in the specified timezone.
    // Run the same code as above with different timezone
    . Note that the 8 to 5 workday is
        // offset by the two hours as specified in our timezone.
    dc.setTimeZone("GMT-2");
    var gdt = new GlideDateTime("2012-05-03 15:00:00");
    dc.setStartTime(gdt);
    for(var i=2; i<24; i+=1){
        if(!dc.calcDuration(i*3600)){
            gs.log("*** Error calculating duration");
            return;
        }
        gs.log("add "+ i +" hours gives due date (GMT-2) :
" + dc.getEndTime());
    }

/**
 * Add a specific schedule to the DurationCalculator object.
 *
 * @param durationCalculator An instance of DurationCalculator
 */
function addSchedule(durationCalculator){
    // Load the "8-5 weekdays excluding holidays" schedule into our duration calculator.
    var scheduleName = "8-5 weekdays excluding holidays";
```

```
var grSched = new GlideRecord('cmn_schedule');
grSched.addQuery('name', scheduleName);
grSched.query();
if(!grSched.next()){
    gs.log('*** Could not find schedule "'+ scheduleName +'"');
    return;
}
durationCalculator.setSchedule(grSched.getUniqueValue(), "GMT");
```

Using DurationCalculator to compute a simple duration

A simple duration is the number of seconds between two date times.

If no schedule is used then this is a simple time date subtraction. If a schedule is used, then the schedule is consulted to remove non-work hours from the computation. Suppose schedule "8-5 weekdays excluding holidays" is used. In this case, the number of work hours from noon Monday to noon Tuesday is nine hours. To compute a simple duration, initialize the global API DurationCalculator and call the [calcScheduleDuration\(\)](#) method.

This script demonstrates how to use DurationCalculator to compute a simple duration.

```
/**
 * Sample script demonstrating use of DurationCalculator to
 * compute simple durations
 */

gs.include('DurationCalculator');
executeSample();

/**
 * Function to house the sample script.
 */
function executeSample() {

    // First we need a DurationCalculator object.
    var dc = new DurationCalculator();
```

```
// Compute a simple duration without any schedule. The arguments
// can also be of type GlideDateTime, such as fields from a GlideRecord.
var dur = dc.calcScheduleDuration("2012-05-01", "2012-05-02");
gs.log("calcScheduleDuration no schedule: " + dur); // 86400 seconds (24 hours)

// The above sample is useful in limited cases. We almost always want to
// use some schedule in a duration computation, let's load a schedule.
addSchedule(dc);

// Compute a duration using the schedule. The schedule
// specifies a nine hour work day. The output of this is 32400 seconds, or
// a nine hour span.
dur = dc.calcScheduleDuration("2012-05-23 12:00:00", "2012-05-24 12:00:00");
gs.log("calcScheduleDuration with schedule: " + dur);
// 32400 seconds (9 hours)

// Compute a duration that spans a weekend and holiday.
// Even though this
// spans three days, it only spans 9 work hours based on the schedule.
dur = dc.calcScheduleDuration("2012-05-25 12:00:00", "2012-05-29 12:00:00");
gs.log("calcScheduleDuration with schedule spanning holiday: " + dur); // 32400 seconds (9 hours)

// Use the current date time in a calculation. The output of this is
// dependent on when you run it.
var now = new Date();
dur = dc.calcScheduleDuration("2012-05-15", new GlideDateTime());
gs.log("calcScheduleDuration with schedule to now: " + dur); // Different on every run.
}

/**
```

```
* Add a specific schedule to the DurationCalculator object.  
*  
* @param durationCalculator An instance of DurationCalculator  
*/  
function addSchedule(durationCalculator){  
    // Load the "8-5 weekdays excluding holidays" schedule into our duration calculator.  
    var scheduleName = "8-5 weekdays excluding holidays";  
    var grSched = new GlideRecord('cmn_schedule');  
    grSched.addQuery('name', scheduleName);  
    grSched.query();  
    if(!grSched.next()){  
        gs.log('*** Could not find schedule "'+ scheduleName +'");  
        return;  
    }  
    durationCalculator.setSchedule(grSched.getUniqueValue());  
}
```

Using relative duration

Relative duration is very similar to simple duration except a piece of script is used to determine what parts of a day to remove from the difference calculation.

This script is stored in table cmn_relative_duration and can be examined by navigating to **System Scheduler > Schedules > Relative Durations**. There are some example relative duration scripts in the out-of-the-box instance.

A relative duration sys_id is passed to the method calcRelativeDuration() of the global API [DurationCalculator](#) class after initialization. When this method is called, the DurationCalculator object is passed to the relative duration script (stored in table cmn_relative_duration) as the variable calculator. So, the relative duration script you write and store in cmn_relative_duration has access to the executing DurationCalculator through the variable calculator.

The following script demonstrates how to use DurationCalculator to calculate a relative duration.

```
/**  
 * Sample use of relative duration calculation.  
 */  
  
gs.include('DurationCalculator');  
executeSample();  
  
/**  
 * Function to house the sample script.  
 */  
function executeSample(){  
  
    // First we need a DurationCalculator object. We will  
also use  
    // the out-of-box relative duration "2 bus days by 4pm  
"  
    var dc = new DurationCalculator();  
    var relDur = "3bf802c20a0a0b52008e2859cd8abcf2"; // 2  
bus days by 4pm if before 10am  
    addSchedule(dc);  
  
    // Since our start date is before 10:00am our result i  
s two days from// now at 4:00pm.  
    var gdt = new GlideDateTime("2012-05-01 09:00:00");  
    dc.setStartTime(gdt);  
    if(!dc.calcRelativeDuration(relDur)){  
        gs.log("**** calcRelativeDuration failed");  
        return;  
    }  
    gs.log("Two days later 4:00pm: "+ dc.getEndTime())  
;  
  
    // Since our start date is after 10:00am our result i  
s three days from  
    // now at 4:00pm.  
    var gdt = new GlideDateTime("2012-05-01 11:00:00");  
    dc.setStartTime(gdt);  
    if(!dc.calcRelativeDuration(relDur)){  
        gs.log("**** calcRelativeDuration failed");  
        return;  
    }  
    gs.log("Three days later 4:00pm: "+ dc.getEndTime())
```

```
) );}

/**
 * Add a specific schedule to the DurationCalculator object.
 *
 * @param durationCalculator An instance of DurationCalculator
 */
function addSchedule(durationCalculator) {
    // Load the "8-5 weekdays excluding holidays" schedule into our duration calculator.
    var scheduleName = "8-5 weekdays excluding holidays";
    var grSched = new GlideRecord('cmn_schedule');
    grSched.addQuery('name', scheduleName);
    grSched.query();
    if(!grSched.next()){
        gs.log('*** Could not find schedule "' + scheduleName + '!"');
        return;
    }
    durationCalculator.setSchedule(grSched.getUniqueValue(), "GMT");}
```

Querying tables in script

Using methods in the GlideRecord API, you can return all records from a table, return records from a table that satisfy specific conditions, or return records that include a string from a single table or from multiple tables in a text index group.

Query tables using the GlideRecord API. For API reference, see [GlideRecord - Scoped](#).

Return all records from a table

To query a table, first create a GlideRecord object. To create a GlideRecord, create the following in script:

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

Creating a GlideRecord creates a target variable which is a GlideRecord object for the incident table.

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 script issues the `query()` to the database. Each call to `next()` would load the next record for processing.

Return records from a table that satisfy query conditions

Most of the time, you 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 that 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 that the example script includes `target.addQuery('priority', 1)`. This line indicates that you only want the records where the priority field is equal to 1. In general, most queries that you want to perform are equality queries; queries where you want to find records with a field equal to a value. For this reason, you do not need to provide an equality operator. However, lets say you wanted to find all incidents where the priority field is greater than 1. In this case, you would provide the operator that you want to apply to the query.

```
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  
}
```

Return records from a table that include a string

Use the '123TEXTQUERY321' reserved name to search for string matches in all fields on a table. For example, this script returns records from the Incident table with field values that include the 'email' string.

```
var now_GR = new GlideRecord('incident');
gr.addQuery('123TEXTQUERY321', 'email');
gr.query();
```

'123TEXTQUERY321' is a reserved option for the name parameter in the `addQuery()` method. You can use this option in an encoded query string. For example, instead of `gr.addQuery('123TEXTQUERY321', 'email');`, you can use `gr.addEncodedQuery('123TEXTQUERY321=email')`.

String search is case-insensitive. The system returns the same results whether you search for `email`, `Email`, or `EMAIL`.

Note: Before you can query using a string search, you must configure text indexing (and optionally search attributes) for the table you want to search. For more information, see [Configure a single table for indexing and searching](#).

Return records from multiple tables in a text index group that include a string

Use the '123TEXTINDEXGROUP321' reserved name to search for a string in a table from a text index group. This option returns results with relevancy scores computed using the text index group's settings.

Note: You can only query one table in the text index group at a time. Use this option when you want to issue multiple single-table queries and merge their results, ordering them by relevancy score. The advantage of this option is that search result relevancy scores for the individual table queries are normalized consistently for all tables in the text index group.

For example, this script returns records from the Knowledge table that include the keyword 'email', with relevancy scores computed for the 'portal' index group.

```
var now_GR = new GlideRecord('kb_knowledge');
gr.addQuery('123TEXTQUERY321', 'email');
gr.addQuery('123TEXTINDEXGROUP321', 'portal');
gr.query();
```

You can create a similar query for each additional table in the 'portal' index group that you want to search, and then merge the individual queries' results, displaying the results with the highest relevancy scores first. Because all of these search queries use the same text index group search settings, the relevancy scores for their results are all normalized consistently. If you searched the same set of tables without using the `gr.addQuery('123TEXTINDEXGROUP321', 'portal')` method, the individual queries' relevancy scores would be normalized differently and would not be a useful basis for sorting the merged result set.

'123TEXTINDEXGROUP321' is a reserved option for the name parameter in the `addQuery()` method. You can use this option in an encoded query string. For example, instead of `gr.addQuery('123TEXTINDEXGROUP321', 'portal');`, you can use `gr.addEncodedQuery('123TEXTINDEXGROUP321=portal')`.

Multi-table string search is case-insensitive. The system returns the same results whether you search for `email`, `Email`, or `EMAIL`.

Note: Before you can query tables in an index group, you must configure text indexing and search attributes for those tables and include them in the index group. For more information, see [Configure multiple tables for indexing and searching](#). All tables in the index group must use the V4 indexing format.

Available JavaScript operators

Describes the operators that can be used within an `addQuery()` request.

Available JavaScript Operators

Field	Definition	addQuery
=	Field must be equal to value supplied.	<code>addQuery('priority', '=', 1);</code>
>	Field must be greater than value supplied.	<code>addQuery('priority', '>', 1);</code>

Field	Definition	addQuery
<	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 returns 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 returns 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', 'Success', 'Failure'), 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, 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 returns all records where the short_description field ends with text Error.	addQuery('short_description', 'ENDSWITH', 'Error');

Field	Definition	addQuery
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 returns all records where the short_description field does not have the word "Error."	addQuery('short_description', '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 retrieves only records of a specified "class" for extended tables. The code example on the right, shows how to retrieve all configuration items that are classified as computers.	addQuery('sys_class_name', 'INSTANCEOF', 'cmdb_ci_computer');

For additional information on the operators that are available for filters and queries, see [Operators available for filters and queries](#).

There are also some special methods that you can use 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
}
```

For more information on the GlideRecord API and its available methods, see [GlideRecord](#).

GlideRecord query examples

These examples demonstrate how to perform various GlideRecord queries.

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(); }
```

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 item_name with a value of item_value and displays the request item numbers:

```
var now_GR = new GlideRecord('sc_item_option_mtom');  
gr.addQuery('sc_item_option.item_option_new.name','item_na  
me');  
gr.addQuery('sc_item_option.value','item_value');  
gr.query();  
  
while(gr.next()) {  
    gs.addInfoMessage(gr.request_item.number); }
```

For additional information see [GlideRecord](#).

Running order guides automatically

Service catalog order guides allow customers to make a single service catalog request that can generate several ordered items. Administrators can configure order guides to run automatically, from a workflow or a script to generate a set of ordered items without manually submitting a service catalog request. Administrators can also review and reprocess the order guide failures.

As a use case, an onboarding workflow for a new employee can run an order guide to automatically order items for that employee.

Note: You can only save catalog items, not the order guide (that is, initial landing page options).

Running order guides from scripts

Running order guides with a server-side script is more complex than using workflows, but it allows more flexibility and can be used in non-workflow situations.

For example, you can use order guide scripts with UI actions or server-side business rules.

Note: When order guides run automatically, order guide UI policies are not enforced. Also, options in the Choose Options screen cannot be selected, so make sure that order guide rules define sensible defaults for these options to avoid processing failures.

Use the SNC.ScriptableOrderGuide Java class to run order guides with a script.

Use the SNC.ScriptableOrderGuide(String orderGuidId) constructor to create a new ScriptableOrderGuide object.

Method summary

Method	Return Value	Description
process(String json)	boolean	Runs the order guide using the JSON encoded string parameter as the input for the order guide. Returns true or false depending on whether processing was successful or not.

Method	Return Value	Description
		<p>Note: Both opened_by and requested_for parameters must be passed to the order guide, and both must have valid user record sys_id values.</p> <p>If processing is successful and a request is created by the order guide, you can retrieve the request GlideRecord using getRequest().</p> <p>If the processing fails, you can retrieve the failure GlideRecord using getFailure(), then submit the script for reprocessing using reprocess.</p>
reprocess(GlideRecord failure)	boolean	Runs the order guide again using the JSON encoded string parameter stored in the failure GlideRecord.
getMessage()	String	Retrieves the message populated after processing or reprocessing.

Method	Return Value	Description
getRequest()	GlideRecord	Retrieves the request GlideRecord.
getFailure()	GlideRecord	Retrieves the failure GlideRecord from the Scriptable Order Guide Failures [sc_script_order_guide_failure] table.

Script example

This script processes an order guide called IT Onboarding SOG.

```
// Creating the object to later be JSON encoded
var json = {"opened_by":"62826bf03710200044e0bfc8bcbe5df1",
,"requested_for":"06826bf03710200044e0bfc8bcbe5d8a","department":"221f3db5c6112284009f4becd3039cc9"};

var now_GR = new GlideRecord("sc_cat_item_guide");
if (gr.get("name","IT Onboarding SOG")) {
    var sog = new SNC.ScriptableOrderGuide(gr.getValue("sys_id"));
    var result = sog.process(new JSON().encode(json));
    if(!result)
        gs.log("Processing the scriptable order guide failed with message: " + sog.getMessage());
    else {
        var request = sog.getRequest();
        gs.log("Request created - " + request.sys_id); } }
```

Running order guides from workflows

Running an order guide from a workflow is suitable if you include order guides as part of a broader workflow-based process.

For example, an activity within an onboarding workflow for a new employee can automatically run an order guide to order items for that employee.

Note: When order guides run automatically, order guide UI policies are not enforced. Also, options in the Choose Options screen cannot be selected, so make sure that order guide rules define sensible defaults for these options to avoid processing failures.

To run order guides from a workflow, use the **Scriptable Order Guide** workflow activity.

Running order guides from workflows

Field	Description
Order Guide	The name of the order guide that this activity processes. For example, Example Employee Onboarding IT .
Script	A script passing information to the order guide. This information is sent as a JSON encoded string parameter assigned to the answer variable. The script must meet these requirements: <ul style="list-style-type: none">• The names of the variables in the script must match the names used within the order guide. For example, if the order guide uses a department variable in a rule condition, the script must also pass a department parameter.• Both opened_by and requested_for parameters must be passed to the order guide, and both must have valid user record sys_id values.

Results

- **Success:** the activity successfully processed the order guide. This does not mean a request was created. If a request was created, the request

sys_id is added to the workflow scratchpad under the sc_request variable.

- **Failure:** while processing the order guide a failure occurred, creating a failure record. If the processing fails, you can view and edit the failure record.

Workflow example

The **Example Employee Onboarding IT Workflow** workflow uses this example to generate IT catalog items for a new employee as part of an onboarding process.

The activity uses this script to:

1. Take a JSON string generated previously from the HR change record.
2. Append the mandatory opened_by and requested_for parameters to that string.
3. Submit the new string for processing by the order guide.

```
var parameters = new JSON().decode(current.payload);

// Need to amend the json object to include additional values.
parameters.opened_by = current.opened_by + "";
parameters.requested_for = current.opened_for + "";

answer = new JSON().encode(parameters);
```

View order guide failures

Order guide processing can fail, for example if the order guide being run does not exist. When a failure occurs during the order guide processing, the **Scriptable Order Guide Failures** submodule allows you to review and reprocess the failures. A record is created for each failure and once you fix the errors that caused the initial failure, you can reprocess the failed order guides.

About this task

If a failure occurs, a failure record is created in the Scriptable Order Guide Failures [sc_script_order_guide_failure] table.

To view details of a failure, navigate to **Service Catalog > Catalog Administration > Scriptable Order Guide Failures**, then open a failure record.

Reprocessing Failures

If you have fixed the error that caused the initial failure, you can reprocess failed order guides.

Procedure

1. Navigate to **Service Catalog > Catalog Administration > Scriptable Order Guide Failures**.
2. Open the failure record.
3. Click the **Reprocess** related link.
To reprocess one or more failures:
 - a. Navigate to **Service Catalog > Catalog Administration > Scriptable Order Guide Failures**.
 - b. Select the check box beside one or more records to reprocess.
 - c. Select **Reprocess** from the **Actions** choice list.

Scriptable assignment of execution plans

Each catalog item has an associated execution plan, used whenever an item of that type is ordered; if no plan is specified, the default plan is used. This default is effective for most organizations, but your execution plan may need to vary based on additional criteria.

For example, in the base system service catalog, a request for a new PC always uses the PC Delivery Plan. However, this plan may need to vary for unusual circumstances - such as when a requester is working from home, at a remote location.

To provide this flexibility, you can use a script to override the default execution plan on a specific catalog item.

Limitations during script execution

Execution plan scripts have limitations that need to be considered during their implementation.

While the execution plan script runs:

- You cannot interact with any catalog tasks as catalog tasks are only created after the execution plan is selected.
- Some fields such as **total delivery time** and **due date** are not yet calculated, although the request itself is available within the script via `current.request()`.
- Approvals have not yet been generated.

Writing the scripts

Follow these guidelines when writing execution plan scripts.

Execution plan scripts can access the same global variables and other functions as in any other server side execution plan.

- `current` is the currently-requested catalog item, `sc_req_item`.
- `current.delivery_plan()` is the assigned execution plan for this catalog item.

The evaluated value from the script is used as the `sys_id` of the execution plan.

Simple example:

```
current.delivery_plan.setDisplayValue('PC Delivery Plan')
```

If an invalid value is returned, such as undefined or not found, then the existing assigned value is used.

More complex example:

```
getexecutionplan();
function getexecutionplan() {
var location = current.request.requested_for.location.getDisplayValue();
```

```
// if we're in Atlanta
if (location == 'Atlanta') {
    // use the remote pc delivery plan instead of the normal one
    var remote_plan = new GlideRecord('sc_cat_item_delivery_plan');
    remote_plan.addQuery('name', 'Remote PC Delivery Plan')
    ;
    remote_plan.query();
    remote_plan.next();
    current.delivery_plan = remote_plan.sys_id;
    return remote_plan.sys_id;
}
return current_delivery_plan;
}
```

In this example, any time a request is for a user in Atlanta, ServiceNow uses the Remote PC Delivery Plan. Otherwise, the execution plan is not overridden and ServiceNow uses the catalog item's normal execution plan, the PC Delivery Plan.

Add a script to a catalog item

You can add a script to a catalog item so that the script runs each time a user requests that item.

Procedure

1. Navigate to **All > Service Catalog > Maintain Items**.
2. Select the relevant catalog item to which you wish to add the script.
3. Configure the catalog item form to add the execution plan script field, often named **Delivery Plan Script**.
4. Fill in the script details.

Item

Name:	Executive Desktop	Price:	1,875
Category:	Computers and Handhelds	Active:	<input checked="" type="checkbox"/>
Delivery plan:	PC Delivery Plan	Icon:	[add] Click to add...
Template:			
Short description:	Dell Precision 690		
Description:	<p>Arial 1 (8 pt) Heading 1 B I U </p> <p>Ideal owner: Corporate customers seeking next-generation design, advanced video card options, dual monitor support and a wide selection of form factors</p> <p>- Speed</p> <p>Path:</p> <p>Picture: [update] [delete]</p> 		
Delivery plan script:	<pre>var location = current.request.requested_for.location.getDisplayValue(); // if we're in Atlanta if (location == 'Atlanta') { // use the remote pc delivery plan instead of the normal one var remote_plan = new GlideRecord('sc_cat_item_delivery_plan'); remote_plan.addQuery('name', 'Remote PC Delivery Plan'); remote_plan.query(); remote_plan.next(); current.delivery_plan = remote_plan.sys_id; }</pre>		
<p>Update Copy Delete</p>			

5. Update the item form with your changes.

Result

The script runs each time that item is requested, selecting the execution plan to run with that item.

Use a script to approve an execution plan

You can use an approval rule script to approve an execution plan.

Procedure

1. Retrieve an approval execution plan task.
2. View the **Approval Script** field.
3. Fill in an approval script using the same syntax and rules you would use on an approval rule.

Example

For example, in the script below, the requester's manager is the approver.

The screenshot shows the 'Execution Plan Approval Task' screen for a task named 'Manager Approval'. The task has an order of 100. It is associated with a delivery plan 'Ergonomic Furniture Deliver' and a delivery time of 2 days, 0 hours, 0 minutes. The SLA is set to 'choose field'. The condition is defined as 'choose field > oper value'. The short description is 'Doctor's recommendation required.' and the instructions are 'Verify that the employee provides a valid doctor's recommendation.' In the 'Approval script' section, there is a code editor containing the following JavaScript:

```
1 * (function manager_as_approver () {  
2     var request = current.request_item.request;  
3     var rc = request.requested_for.manager;  
4     return rc;  
5 })();|
```

Scriptable service catalog variables

You can use scripting to reference any request item variable from a table in scoped and non-scoped environment.

An example of a variable reference follows.

```
current.variables.<variable_name>
```

Where current refers to the current record, and <variable_name> is the name of your variable.

Note: In order to reference a variable from JavaScript, it must have a name.

When a variable is part of a variable set, you can reference it as current.variables.<variable_name> or current.variables.<variable_set_name>.<variable_name>.

Variable set is also a first-class citizen in Service Catalog. Like variables, a variable set has read, write, and create roles. If roles are provided for a variable set, the roles are applicable for the variables within the set. Roles of an individual variable are overridden by the roles of the variable set.

Print a variable

```
var original = current.variables.original_number;  
gs.print(original);
```

Set a variable

```
current.variables.name = "Auto-Generated:" + current.variables.asset_tag;
```

Create an inventory item with fields set from variables

```
doCreation();  
  
function doCreation () {  
var create = current.variables.create_item;  
if (create == 'true') { // we want to create an asset  
    var computer = new GlideRecord('cmdb_ci_computer');  
    computer.initialize();  
    computer.asset_tag = current.variables.asset_tag;  
    computer.serial_number = current.variables.serial_number  
;  
    computer.name = current.variables.name;  
    computer.manufacturer = current.variables.company;  
    computer.insert(); } }
```

Get GlideElementVariable of variables and variable sets associated with a GlideRecord

```
now_GR.variables
```

Get the name value pair of variables associated with a GlideRecord

```
now_GR.variables.getVariableValue();
```

Get a list of GlideElementVariable for variables within a task record

```
now_GR.variables.getElements();
```

Get a list of GlideElementVariable for variables (including multi-row variable set) within a task record

```
now_GR.variables.getElements(true);
```

Note: The getElements() method returns all the variables present on the given task record except for the formatter variables like label, break, container end, container split, and so on.

APIs for GlideElementVariable

- now_GR.variables.<var_name>.isMultiRow(): Get whether the GlideElementVariable is a multi-row variable set or a variable.
- now_GR.variables.<var_name>.getQuestion(): Get the Question object for a variable. Applicable only for a variable (isMultiRow() is false) and not for a multi-row variable set.
- now_GR.variables.<var_name>.getLabel(): Get the label of the GlideElementVariable. For a variable, the label of the variable is returned. For multi-row variable set, the title of the variable set is returned.
- now_GR.variables.<var_name>.canRead(): Get whether the user can view a variable or multi-row variable set.
- now_GR.variables.<var_name>.canWrite(): Get whether the user can edit a variable or multi-row variable set.

- now_GR.variables.<var_name>.getDecryptedValue(): Get the decrypted value for a masked variable. Applicable only for a masked variable.
- now_GR.variables.<var_name>.getRows(): Get the list of row objects for a multi-row variable set. Applicable only for a multi-row variable set (isMultiRow() is true).
- now_GR.variables.<var_name>.getRowCount(): Get the number of rows for multi-row variable set. Applicable only for a multi-row variable set (isMultiRow() is true).

Example to access variables of GlideRecord for the Task table

```
var now_GR = new GlideRecord('sc_req_item');
if (now_GR.get('635a1f5387320300e0ef0cf888cb0b73')) {
    var variables = now_GR.variables.getElements();
    for (var i=0;i<variables.length;i++) {
        var question = variables[i].getQuestion();
        gs.log(question.getLabel() + ":" + question.getValue())
    }
}
```

Example to access a multi-row variable set of GlideRecord for the Task table

```
var now_GR = new GlideRecord('sc_req_item');
now_GR.get('02c38dc87013300e0ef0cf888cb0bb2');

var vars = now_GR.variables.getElements(true);

for (var i=0; i<vars.length; i++) {
    var now_V = vars[i];
    if (now_V.isMultiRow()) {
        var rows = now_V.getRows();
        for (var j=0; j<now_V.getRowCount(); j++) {
            var row = rows[j];
            var cells = row.getCells();
            for (var k=0; k<cells.length; k++) {
                var cell = cells[k];
                gs.info(cell.getLabel() + ":" + cell.getCellDisplayValue())
            }
        }
    }
}
```

```
}
```

Multi-row variable set

Multi-row variable set

Operation	Usage
Table operations	
Return JSON array value as String	<code>now_GR.variables.table_var</code>
Set value of a multi-row variable set	<code>now_GR.variables.table_var = <val></code> Note: An array of ordered (key, value) pairs is also applicable as input.
Get value of column, var1, of a multi-row variable set	<code>now_GR.variables.table_var.var1</code>
Set value of a variable set, var1	<code>now_GR.variables.table_var.var1 = <val></code> Note: An array of ordered (key, value) pairs is also applicable as input.
Row operations	
Get the current row count	<code>now_GR.variables.table_var.getRowCount()</code>

Operation	Usage
Returns the row specified by the variable "i" - <code>getRow(<int> i)</code>	<pre>var row = now_GR.variables.table_var.getRow(<int> i);</pre>
Get the cell value for a question column mapped to <code><var_name></code>	<pre>row.<var_name></pre>
Set the cell value for a question column mapped to <code><var_name></code>	<pre>row.setCellValue('<var_name>', value)</pre>
Set the cell value for a question column mapped to <code><var_name></code>	<pre>row.<var_name> = value</pre>
Add an empty row at the end of the table and return a scriptable object	<pre>var row = now_GR.variables.table_var.addRow()</pre>
Delete a row	<pre>row.deleteRow()</pre>

Notes and limitations

- //Single column of table_Var
 - now_GR.variables.table_var.var1
 - now_GR.variables.table_var.var1 = <val>
1. You can only set a variable in a before business rule. Variables set in an after rule are not written to the database.
 2. There is nothing in place to prevent namespace collision with variables. Creating two variables named computer_speed would result in only one of them showing up; the second one would overwrite the first one.
 3. Date/time variables use the same time zone formatting and storage rules as all other dates in the system. They are stored internally in GMT, but translated into the user's local time zone and format for display.

Setting a GlideRecord variable to null

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.

Note: This functionality requires a knowledge of JavaScript.

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.

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

Example 1

```
var grIncident = new GlideRecord('incident');
grIncident.addNotNullQuery("assigned_to");
grIncident.query();
if (grIncident.next()) {
    gs.log("The incident record that is going to be updated
is " + grIncident.number);
    gs.log("Previous Value of 'Assigned To' field: " + grIncident.assigned_to);
    grIncident.assigned_to = "NULL";
    grIncident.update();
    gs.log("Current Value of 'Assigned To' field: " + grIncident.assigned_to);
}
```

Example 2 (Business Rule)

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

Schedule Pages

A schedule page is a record that contains a collection of scripts that allow for custom generation of a calendar or timeline display.

Creation of timeline schedule pages requires understanding of the page/event flow and the ability to write client and server side JavaScript.

Schedule pages form

To access schedule pages, navigate to **System Scheduler > Schedules > Schedule Pages**.

The form provides the following fields, depending upon the View Type selected:

Schedule pages form

Field	Field Type	Description
Name	String	General name that is used to identify the current schedule page.
Schedule type	String	<p>The schedule type is a string that is used to uniquely identify the schedule page via the "sysparm_page_schedule_type" URI parameter. For example, a schedule page could be accessed as follows:</p> <pre>/show_schedule_page.do?sysparm_type=gantt_chart&sysparm_timeline_task_id=d530bf907f0000015ce594fd929cf6a4</pre> <p>Alternatively, the schedule page can also be accessed by setting the</p>

Field	Field Type	Description
		<p>"sysparm_page_sys_id"</p> <p>URL parameter to that of the unique 32 character hexadecimal system identifier of the schedule page.</p>
View Type	Choice	<p>Each view type displays different field combinations. There are two options available:</p> <ul style="list-style-type: none"> • Calendars • Timelines
Description	String	<p>General description that provides additional information about the current schedule page. This field is not necessary.</p>
Init function name	String	<p>Note: This functionality is only used by Calendar type schedule pages.</p> <p>The init function name specifies the name of the JavaScript function to call inside the Client script function for calendar type schedule pages.</p>

Field	Field Type	Description
HTML	String	<p>Note: This functionality is only used by Calendar type schedule pages.</p> <p>The HTML field is a scriptable section that is parsed by Jelly and injected into the display page prior to the rest of the calendar. It can be used to pass in variables from the server and define extra fields are necessary.</p>
Client script	String	<p>The client script is a scriptable section that allows for configuring options of the schedule page display. The API is different depending on the schedule page view type and is discussed below.</p>
Server AJAX processor	String	<p>Note: This functionality is only used by Calendar type schedule pages.</p> <p>The Server AJAX processor is specific to calendar type schedule pages that is used to return a set</p>

Field	Field Type	Description
		of schedule items and spans to be displayed.

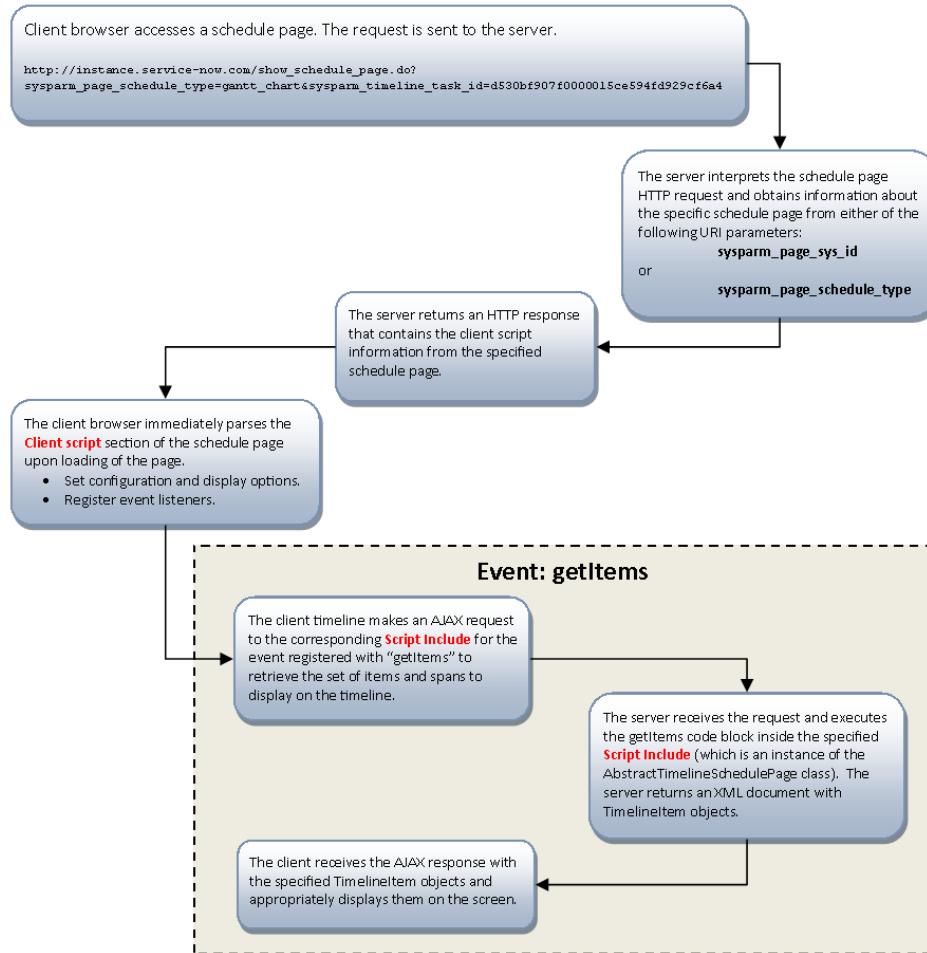
Timeline schedule pages

A Timeline Schedule Page is a specific record that contains configuration information for displaying time based points and spans in a "timeline" like fashion.

The timeline schedule page references a script include that extends from AbstractTimelineSchedulePage to perform dynamic modification to the timeline based on different events and conditions. Both the schedule page and the script include for timeline generation allow for extreme customization and their corresponding application programming interface (API) is documented below.

The following diagram shows the series of events that occur when a timeline schedule page is accessed. Once the timeline has been loaded, all subsequent events, such as events resulting from timeline interaction (e.g. moving a timeline span), follow the same logic flow shown in the gray event box.

Timeline Flow



Applications that use schedule pages to generate time lines

- Project Management
- Maintenance Schedules
- Group On-Call Rotation
- Field Service Management

Timeline schedule page example

The following example demonstrates how to create a timeline schedule page with corresponding script include utilizing a majority of the API described above.

For this example we are going to create an Incident Summary Timeline for a project support manager to visualize all of the new incidents. All new incidents should be displayed as single points where the priority of the incident is distinguished by a different point icon. Additionally, all closed incidents should be displayed on the timeline in a separate group that shows the duration of the incident before it was closed. Since the Project Manager wants to be able to easily close new items that are resolved without using any form lists, we will handle the vertical move event allowing the new incidents to be dragged into the closed incident group or items within.

Schedule Page

Create a new schedule page with the following properties:

- **Name:** Hardware Incidents
- **Schedule type:** incident_timeline
- **View Type:** Timeline
- **Client Script:**

```
// Set our page configuration
glideTimeline.setReadOnly(false);
glideTimeline.showLeftPane(true);
glideTimeline.showLeftPaneAsTree(true);
glideTimeline.showTimelineText(true);
glideTimeline.showDependencyLines(false);
glideTimeline.groupByParent(true);
glideTimeline.setDefaultPointIconClass('milestone');

// We will define what items to display and provide a custom event handler for moving new items to the closed state
glideTimeline.registerEvent('getItems', 'IncidentTimelineScriptInclude');
glideTimeline.registerEvent('elementMoveY', 'IncidentTimelineScriptInclude');
```

Script Include

Now that the schedule page has been created we need to generate a matching script include for the events that were registered. Create a new script include with the following properties:

- **Name:** IncidentTimelineScriptInclude
- **Active:** Checked
- **Client callable:** Checked
- **Script:**

```
// Class Imports

var IncidentTimelineScriptInclude = Class.create();
IncidentTimelineScriptInclude.prototype = Object.extendObject(AbstractTimelineSchedulePage, {

    ////////////////////////////// // GET_ITEMS ///////////////////////
    /////////////////////////////
    getItems:function() {
        // Specify the page title
        this.setPageTitle('My Custom Incident Summary Timeline');
    }

    var groupNew = new GlideTimelineItem('new');
    groupNew.setLeftLabelText('New Incidents');
    groupNew.setImage('../images/icons/all.gifx');
    groupNew.setTextBold(true);
    this.add(groupNew);

    var groupClosed = new GlideTimelineItem('closed');
    groupClosed.setLeftLabelText('Closed Incidents');
    groupClosed.setImage('../images/icons/all.gifx');
    groupClosed.setTextBold(true);
    groupClosed.setIsDroppable(true);

    // This allows us to drag an open incident onto the closed group row.
    this.add(groupClosed);

    // Get all the incidents and let's add only the new/closed
});
```

```
osed ones appropriately
    var now_GR = new GlideRecord('incident');
    gr.query();
    while(gr.next()) {
        // Only loop through new/closed incidents
        if(gr.incident_state != '1' && gr.incident_state !
= '7') continue;

        // Ok, we have a new/closed incident. Create the item and the span first.
        var item = new GlideTimelineItem(gr.getTableName()
, gr.sys_id);
        var span = item.createTimelineSpan(gr.getTableName(
), gr.sys_id);

        // Specific properties for a new incident
        if(gr.incident_state == '1') { // New
            item.setParent(groupNew.getSysId());
            item.setImage('../images/icons/open.gifx');
            span.setTimeSpan(gr.getElement('opened_at').getGlideObject().getNumericValue(),
                            gr.getElement('opened_at').getGlideObject().getNumericValue());

            // We'll show different colors based upon the priorities only for new incidents
            switch(gr.getElement('priority').toString()) {
                case '1': span.setPointIconClass('red_circle')
; break;
                case '2': span.setPointIconClass('red_square')
; break;
                case '3': span.setPointIconClass('blue_circle')
; break;
                case '4': span.setPointIconClass('blue_square')
; break;
                case '5': span.setPointIconClass('sepia_circle'
); break;
                default: // Otherwise, the default point icon class will be used (Milestone)
            }
        }
        // Specific properties for a closed incident
        else if(gr.incident_state == '7') {
            item.setParent(groupClosed.getSysId());
```

```
        item.setImage('../images/icons/closed.gifx');
        span.setTimeSpan(gr.getElement('opened_at').getGlideObject().getNumericValue(),
                          gr.getElement('closed_at').getGlideObject().getNumericValue()); }

        // Common item properties
        item.setLeftLabelText(gr.short_description);

        // Common span properties
        span.setSpanText(gr.short_description);
        span.setToolTip('<strong>' + GlideStringUtil.escapeHTML(gr.short_description) + '</strong><br>' + gr.number);
;
        span.setAllowXMove(false);
        span.setAllowYMove(gr.canWrite() ? true:false);
        span.setAllowYMovePredecessor(false);
        span.setAllowXDragLeft(false);
        span.setAllowXDragRight(false);

        // Now we add the actual item
        this.add(item);
    } } ,

////////////////////////////// ELEMENT_MOVE_Y ///////////////////
////////////////////////////// //////////////////////////////

/**
 * This is one of the AbstractTimelineSchedulePage event handler methods that corresponds to a vertical move. The arguments for this function are defined in the API section of the event handler methods.
 */
elementMoveY: function(spanId, itemId, newItemId) {

    // Get information about the current incident
    var now_GR = new GlideRecord('incident');
    gr.addQuery('sys_id', spanId);
    gr.query();
    if(!gr.next())
        return this.setStatusError('Error', 'Unable to look up the current incident.');
}
```

```
// Only allow the new incidents to have their state adjusted.  
if(gr.incident_state != '1')  
    return this.setStatusError('Error', 'Only new incidents can have their state adjusted.');  
  
// Get information about the dropped GlideTimelineItem  
. If it was dropped in an item on the "New Incidents"  
// group let's do nothing. If it was dropped in the "Closed Incidents" then let's adjust the state automatically  
. .  
var grDropped = new GlideRecord('incident');  
grDropped.addQuery('sys_id', newItemId );  
grDropped.query();  
if(!grDropped.next() || grDropped.incident_state == '7'  
) {  
    // This means the dropped item was either the 'Closed Incidents' group (which has no record or sys_id) or an  
    // existing incident that is closed. The 'New Incidents' also has no sys_id; however, the default behavior for  
    // items without a sysId is to be non-droppable. This is why we explicitly denoted the 'Closed Incidents' to  
    // be marked as "droppable".  
  
    // Return a dialog prompt  
    this.setStatusPrompt('Confirm', 'Are you sure you want to close: ' +  
        '<div style="margin:10px 0 10px 14px;padding:4px;background-color:#E8E8E8;"><strong>' +  
            GlideStringUtil.escapeHTML(gr.short_description) +  
        '</strong><br/><div class="font_smaller">' +  
        now_GR.number + '</div></div>',  
        'this._elementMoveYHandler_DoClose', // This function is for when the OK button is clicked.  
        'this._elementMoveYHandler_DoNothing', // This function is for when the Cancel button is clicked.  
        'this._elementMoveYHandler_DoNothing'); // This function is for when the Close button is clicked.  
    } } ,  
  
_elementMoveYHandler_DoClose: function(spanId, itemId, n
```

```
newItemId) {
    // Notice that this function takes the same function arguments as the original function for which it
    // is a custom event handler for.

    // Update the database record from 'New' to 'Closed'.
    var now_GR = new GlideRecord('incident');
    gr.addQuery('sys_id', spanId);
    gr.query();
    gr.next();
    gr.setValue('incident_state', '7');
    gr.update();

    // This will re-render the timeline showing the updated item in the closed group.
    this.setDoReRenderTimeline(true);

    // Let's show a success message
    this.setStatusSuccess('Success', '<strong>' + gr.short_description + '</strong> was successfully closed.'),

    // Since the user clicked cancel or close we simply do nothing.
    _elementMoveYHandler_DoNothing: function(spanId, itemId
    , newItemId) { }

});
```

Screenshots / Results

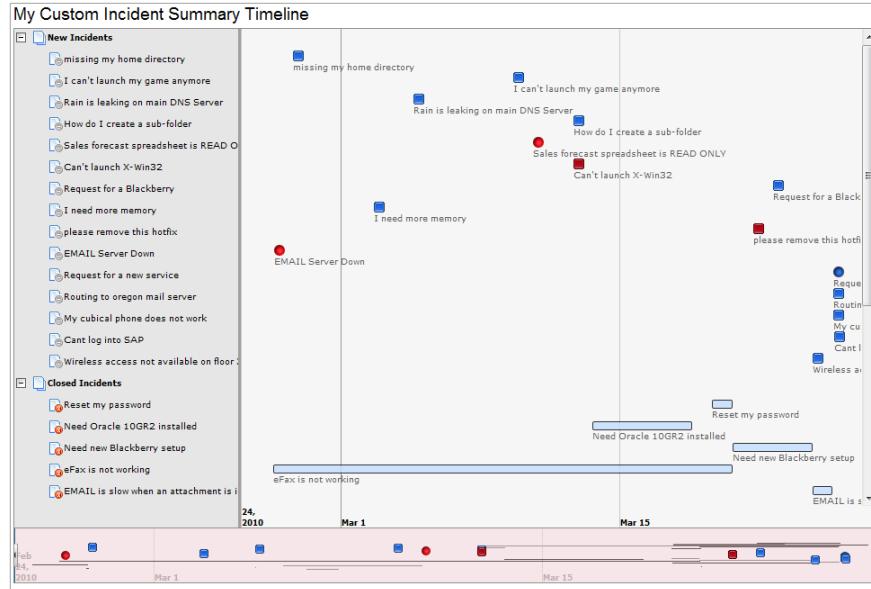
1. Navigate to:

[http://\[YOURINSTANCE\]:8080/show_schedule_page.do?
sysparm_page_schedule_type=incident_timeline](http://[YOURINSTANCE]:8080/show_schedule_page.do?sysparm_page_schedule_type=incident_timeline)

Notice the bold text is the value of the schedule page **Schedule type** field.

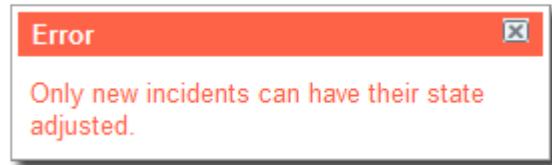
2. The page displays a timeline as specified by the schedule page and script include created. A link to this page can be created and placed as a module or UI action as necessary.

Timeline Example Incident Preview



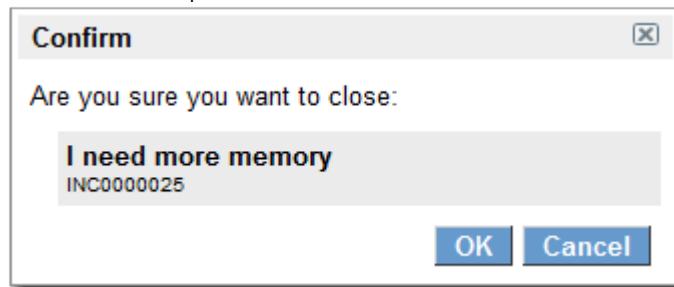
3. Attempting to move a closed incident anywhere displays the expected error message.

Timeline Example Error Moving



4. Moving the incident: 'I need more memory' displays the following confirmation box.

Timeline Example Confirm Close



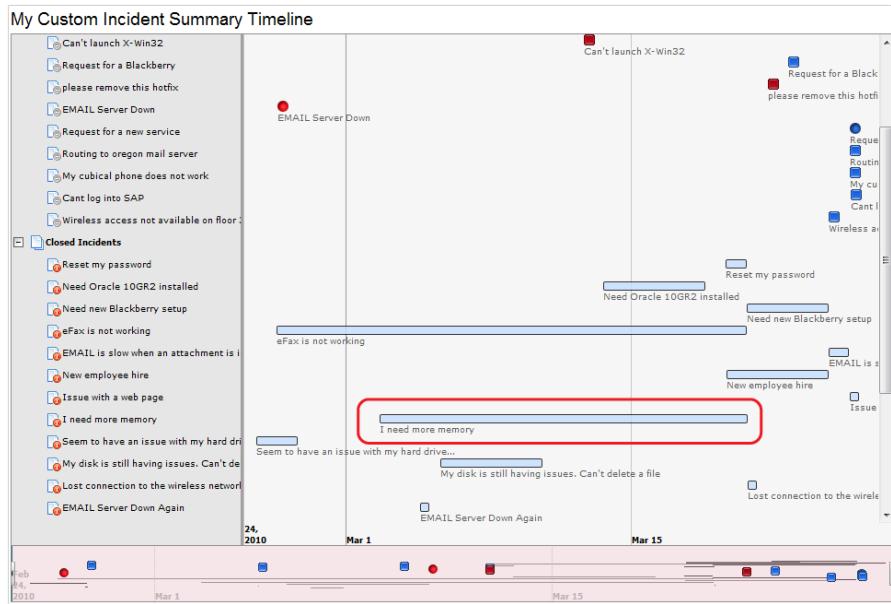
- Clicking the **Cancel** button closes the overlay. Clicking the **OK** button actually updates the incident_state of the record and then displays the following success box.

Timeline Example Close Success



- After clicking **OK**, it is clear the incident is now listed in the **Closed Incidents** group.

Timeline Example Incident Updated



XMLDocument script object

A JavaScript object wrapper for parsing and extracting XML data from an XML document (String).

Use this Javascript class to instantiate an object from an XML string, usually a return value from a Web Service invocation, or the XML payload of ECC Queue. Using the XMLDocument object in a Javascript business rule lets you query values from the XML elements and attributes directly.

Constructor

The constructor of a script object creates a new instance of the object to be used.

```
var xmlString = "<test>" +
    "  <one>" +
    "    <two att=\\"xxx\\">abcd1234</two>" +
    "    <three boo=\\"yah\\" att=\\"yyy\\">1234ab
cd</three>" +
    "    <two>another</two>" +
    "  </one>" +
    "  <number>1234</number>" +
"</test>";

var xmldoc = new XMLDocument(xmlString);
```

To perform XML parsing of an XML string that is name space qualified, specify "true" for the second argument for the XMLDocument constructor. The following is an example of parsing and XML string that contains name space qualification of its elements.

```
var xmlString = "<bk:book xmlns:bk='urn:loc.gov:books' xml
ns:isbn='urn:ISBN:0-395-36341-6'>" +
    "<bk:title>Cheaper by the Dozen</bk:title>" +
    "<isbn:number>1568491379</isbn:number>" +
"</bk:book>

var xmldoc = new XMLDocument(xmlString, true); // XML document is name space aware
```

Locating nodes and elements

Now that we have the XMLDocument object, we can call the following APIs to locate nodes and elements of the XML document, as well as extract text from it directly. The query syntax for locating nodes and attributes is based on XPath.

Note: XML parsing with this object is not namespace aware, this means that if you are locating a node name with namespace in it eg. "<names:myelement> ...", the XPath search string would be "// myelement"

The following are examples of locating a node by its XPath and getting the text value out of the resulting node.

```
var str = xmldoc.getNodeText("//two"); // returns the first occurrence of the node
// str == "abcd1234"

str = xmldoc.getNodeText("//three");
// str == "1234abcd"

str = xmldoc.getNodeText("/test/one/*");
// str == "abcd1234"

str = xmldoc.getNodeInt("//number");
// str == 1234
```

The following examples locates the node by XPath and uses the API on node and element to get attributes and value.

```
var node = xmldoc.getNode("/test/one/two");
// node.getNodeName() == "two"
// node.getNodeType() == "1" // 1 == ELEMENT_NODE
// node.getLastChild().getNodeType() == "3" // 3 == TEXT_NODE
// node.getLastChild().getNodeValue() == "abcd1234"
```

Or you can use the following convenience functions to get the node name and type

```
str = xmldoc.getNodeName("//three");
// str == "three"

str = xmldoc.getNodeType("//three");
// str == "1"
```

You can also get a list of nodes that you can iterate or access directly by position

```
var nodelist = xmldoc.getNodes("//one/*"); // two, three, and two
// nodelist.getLength() == "3"
// nodelist.item(0).getNodeName() == "two"
// nodelist.item(1).getNodeName() == "three"
```

The following is an example of parsing an XML string that is qualified with name spaces.

```
var xmlString = "<bk:book xmlns:bk='urn:loc.gov:books' xmlns:isbn='urn:ISBN:0-395-36341-6'>" +
```

```
e>" +
        "<bk:title>Cheaper by the Dozen</bk:title>
+
        "</bk:book>;

var xmldoc = new XMLDocument(xmlString, true);
var str = xmldoc.getNodeText("//bk:title"); // returns the first occurrence of the node
gs.log(str);

str = xmldoc.getNodeText("/bk:book/*");
gs.log(str);

str = xmldoc.getNodeInt("//isbn:number");
gs.log(str);
```

Getting attribute values

An attribute is just an extension of a node and so it has all the same APIs.

The following example shows how to query for a node to get its attribute by position

```
node = xmldoc.getNode("//two");
// node.getAttributes().item(0).getNodeValue() == "xxx"

str = xmldoc.getAttribute("//two", "att");
// str == "xxx"
```

XPath also has a query syntax for locating the attribute node directly, for example

```
str = xmldoc.getNodeText("//*[@att='yyy']");
// str == "1234abcd"

str = xmldoc.getNode("//@boo").getNodeValue();
// str == "yah"
```

Creating new elements

An XML element can be created at any level of the XML document once it has been created. Use the setCurrent function to position where the new element will be created as a child element, and use the createElement function to create the element.

```
var xmlString = "<test>" +
    "  <one>" +
    "    <two att=\"xxx\">abcd1234</two>" +
    "    <three boo=\"yah\" att=\"yyy\">1234ab
cd</three>" +
    "      <two>another</two>" +
    "    </one>" +
    "    <number>1234</number>" +
  "</test>";

var xmldoc = new XMLDocument(xmlString);
xmldoc.createElement("new", "test"); // creates the new el
ement at the document element level if setCurrent is neve
r called
xmldoc.createElement("new2"); // calling without a value w
ill create a new element by itself

var el = xmldoc.createElement("new3");
xmldoc.setCurrent(el); // this is now the parent of any ne
w elements created subsequently using createElement()
xmldoc.createElement("newChild", "test");
```

The resulting XML document looks like this

```
<test>
  <one>
    <two att="xxx">abcd1234</two>
    <three boo="yah" att="yyy">1234abcd</three>
    <two>another</two>
  </one>
  <number>1234</number>
  <new>test<new>
  <new2/>
  <new3>
    <newChild>test</newChild>
  </new3>
</test>
```

- [XMLHelper](#)

The XML helper script include makes it easy to parse XML in scripts.

XMLHelper

The XML helper script include makes it easy to parse XML in scripts.

Use the following methods to export XML to JSON, or JSON to XML.

- The `toObject()` method returns the XML elements as JSON properties. This method works properly whether the supplied parameter is an XML document or an XML string. This method has an optional parameter of the XML input for conversion as an alternative to specifying the XML input in the constructor.
- The `toXMLDoc()` method returns JSON provided as XML elements.

Note: You must escape ampersand characters (&) in your XML or the conversion silently fails.

Example

The following example shows how to convert an XML document into JSON and uses a recursive function to output each member. The recursive function helps indicate how the XML document structure is rendered as JSON.

```
var xmlString = "<company>" + "<employee>" + "<id>10</id>"  
    + "<firstname>Tom</firstname>" + "<lastname>Cruise</lastname>" + "<test>test1</test>"  
    + "<test>test3</test>" + "</employee>" + "<employee>" + "<id>20</id>" + "<firstname>Paul</firstname>"  
    + "<lastname>Enderson</lastname>" + "<test>test6</test>" + "<test>test5</test>" + "</employee>" + "<employee>"  
    + "<id>30</id>" + "<firstname>Paul</firstname>" + "<lastname>Bush</lastname>" + "<test>test2</test>"  
    + "<test>test4</test>" + "</employee>" + "</company>";  
var helper = new XMLHelper(xmlString);  
var obj = helper.toObject();  
logObj(obj, "*");  
  
function logObj(obj, sep) {  
    for (x in obj) {  
        if (typeof obj[x] != "function") {  
            gs.log(sep + x + ":: " + obj[x]);  
        }  
    }  
}
```

```
    logObj(obj[x], sep + "*");
}
}
```

Output:

```
*** Script: *employee:: [object Object],[object Object],[object Object]
*** Script: **2:: [object Object]
*** Script: ***id:: 30
*** Script: ***test:: test2,test4
*** Script: ****0:: test2
*** Script: ****1:: test4
*** Script: ***firstname:: Paul
*** Script: ***lastname:: Bush
*** Script: **0:: [object Object]
*** Script: ***id:: 10
*** Script: ***test:: test1,test3
*** Script: ****0:: test1
*** Script: ****1:: test3
*** Script: ***firstname:: Tom
*** Script: ***lastname:: Cruise
*** Script: *1:: [object Object]
*** Script: ***id:: 20
*** Script: ***test:: test6,test5
*** Script: ****0:: test6
*** Script: ****1:: test5
*** Script: ***firstname:: Paul
*** Script: ***lastname:: Enderson
```

JavaScript engine on the platform

The JavaScript engine that evaluates server-side scripts supports the ECMAScript 2021 (ES12) standard.

No plugins or properties are required to install the new JavaScript engine.

The benefits of the new engine are:

- Modern library code and scripting features, such as optional chaining, destructuring, const and let declarations, and arrow functions
- Follows standard ECMAScript 2021 behavior

What to know

The JavaScript engine provides an improved environment for developing scripts.

- Beginning with the Tokyo release, new and existing scoped applications can run in ECMAScript 2021 (ES12) mode.
- Compatibility mode supports the legacy modifications to the legacy JavaScript engine.
- Legacy code continues to work.

You configure the mode that the JavaScript engine uses in the design and runtime settings for applications. Available modes are ECMAScript 2021 (ES12), ES5 Standards, and Compatibility.

- [JavaScript modes](#)

JavaScript mode is a design and runtime setting for applications. To support existing server-side scripts and new scripts developed to the ECMAScript 2021 standard, the JavaScript engine has three modes: ECMAScript 2021 (ES12), ES5 Standards, and Compatibility.

- [JavaScript engine feature support](#)

Compare ECMAScript features between the ECMAScript 2021 (ES12) JavaScript mode released in Tokyo and the ES5 Standards mode. Both modes support a subset of ECMAScript features.

- [Porting code to ES5 standards mode scripts](#)

ES5 standards mode catches errors that compatibility mode allows.

JavaScript modes

JavaScript mode is a design and runtime setting for applications. To support existing server-side scripts and new scripts developed to the ECMAScript 2021 standard, the JavaScript engine has three modes: ECMAScript 2021 (ES12), ES5 Standards, and Compatibility.

You configure an application to use one of three JavaScript modes. The default mode for new applications is ES5 Standards. For more information, see [Update a custom application record](#).

ECMAScript 2021 (ES12) mode

ECMAScript 2021 (ES12) mode is an option for scoped applications and scripts to use the ECMAScript 2021 (ES12) standard. This mode does not preserve the legacy behaviors in the pre-Tokyo JavaScript engine or work with global scripts.

ECMAScript 2021 (ES12) mode supports a subset of ECMAScript 2021 syntax and features, including the following features:

- Default function parameters
- Rest parameters
- For-of loops
- Template literals
- Destructuring
 - Declarations
 - Assignment
 - Parameters
- Const declaration
- Let declaration
- Arrow functions
- Class declarations
- Map set
- Optional chaining operator (`?.`)

ES5 Standards mode

ES5 Standards mode is the default mode when you create new scoped applications. This mode does not preserve the legacy behaviors in the pre-Helsinki JavaScript engine.

ES5 standards mode supports ECMAScript5 syntax and features, including the following features:

- The "use strict" declaration
- Control over extensibility of objects
- Get and set properties on objects (accessors)
- Control over writability, configurability, and enumerability of object properties
- New Array and Date methods
- Native JSON support
- Support for modern third-party libraries such as lodash.js and moment.js

For more information about features supported by the ECMAScript 2021 (ES12) and ES5 Standards modes, see [JavaScript engine feature support](#).

Compatibility mode

Compatibility mode is used for all scripts developed prior to the addition of ES5 Standards mode. Compatibility mode has some differences from the previous JavaScript engine.

JSON support changes:

- `JSON.stringify()` and `JSON.parse()` are implemented using the ES5 Native JSON object.
- The new `JSON().encode()` and new `JSON().decode()` are still supported, but should only be used when the legacy behavior is required.

The use of third-party JavaScript libraries is not supported in Compatibility mode.

JavaScript engine feature support

Compare ECMAScript features between the ECMAScript 2021 (ES12) JavaScript mode released in Tokyo and the ES5 Standards mode. Both modes support a subset of ECMAScript features.

For more information about these features, see the ECMAScript language specifications (ECMA-262) on the [Ecma International website](#).

Support definitions

Supported

The feature has been tested and validated.

Not Supported

The feature has not been validated in the current release.

Disallowed

The feature does not align with the Now Platform programming model or poses a security or performance risk. Disallowed features result in an error.

ECMAScript 12 features

Promise.any

Feature	ECMAScript 2021 (ES12)	ES5 Standards
fulfillment	Disallowed	Disallowed
AggregateError	Disallowed	Disallowed

WeakReferences

Feature	ECMAScript 2021 (ES12)	ES5 Standards
WeakRef minimal support	Disallowed	Disallowed
FinalizationRegistry minimal support	Disallowed	Disallowed

Logical assignment

Feature	ECMAScript 2021 (ES12)	ES5 Standards
<code> =</code> basic support	Supported	Not Supported
<code> =</code> short-circuiting behavior	Supported	Not Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
<code> =</code> setter not unnecessarily invoked	Supported	Not Supported
<code>&&=</code> basic support	Supported	Not Supported
<code>&&=</code> short-circuiting behavior	Supported	Not Supported
<code>&&=</code> setter not unnecessarily invoked	Supported	Not Supported
<code>??=</code> basic support	Supported	Not Supported
<code>??=</code> short-circuiting behavior	Supported	Not Supported
<code>??=</code> setter not unnecessarily invoked	Supported	Not Supported

Numeric separators

Feature	ECMAScript 2021 (ES12)	ES5 Standards
numeric separators	Supported	Not Supported

String.prototype.replaceAll

Feature	ECMAScript 2021 (ES12)	ES5 Standards
<code>String.prototype.replaceAll</code>	Supported	Supported

ECMAScript 11 features

String.prototype.matchAll

Feature	ECMAScript 2021 (ES12)	ES5 Standards
basic functionality	Supported	Not Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
throws on non-global regex	Supported	Not Supported

BigInt

Feature	ECMAScript 2021 (ES12)	ES5 Standards
basic functionality	Not Supported	Not Supported
constructor	Not Supported	Not Supported
BigInt.asUintN	Not Supported	Not Supported
BigInt.asIntN	Not Supported	Not Supported
BigInt64Array	Not Supported	Not Supported
BigUint64Array	Not Supported	Not Supported
DataView.prototype.getBigInt64	Not Supported	Not Supported
DataView.prototype.getBigUint64	Not Supported	Not Supported

globalThis

Feature	ECMAScript 2021 (ES12)	ES5 Standards
"globalThis" global property is global object	Disallowed	Disallowed
"globalThis" global property has correct property descriptor	Disallowed	Disallowed

Optional chaining operator (?)

Feature	ECMAScript 2021 (ES12)	ES5 Standards
optional property access	Supported	Not Supported
optional bracket access	Supported	Not Supported
optional method call	Supported	Not Supported
optional function call	Supported	Not Supported
spread parameters after optional chaining	Supported	Not Supported

Promise.allSettled

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Promise.allSettled	Disallowed	Disallowed

Nullish coalescing operator (??)

Feature	ECMAScript 2021 (ES12)	ES5 Standards
nullish coalescing operator (??)	Supported	Not Supported

ECMAScript 10 features

Symbol.prototype.description

Feature	ECMAScript 2021 (ES12)	ES5 Standards
basic	Supported	Not Supported
empty description	Supported	Not Supported
undefined description	Supported	Not Supported

String trimming

Feature	ECMAScript 2021 (ES12)	ES5 Standards
String.prototype.trimLeft	Supported	Supported
String.prototype.trimRight	Supported	Supported
String.prototype.trimStart	Supported	Not Supported
String.prototype.trimEnd	Supported	Not Supported

Array.prototype.{flat, flatMap}

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Array.prototype.flat	Supported	Not Supported
Array.prototype.flatMap	Supported	Not Supported
flat and flatMap in Array.prototype[@@unscopables]	Supported	Not Supported

Object.fromEntries

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Object.fromEntries	Supported	Not Supported

Optional catch binding

Feature	ECMAScript 2021 (ES12)	ES5 Standards
basic	Disallowed	Disallowed
await	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
yield	Disallowed	Disallowed

Function.prototype.toString revision

Feature	ECMAScript 2021 (ES12)	ES5 Standards
functions created with the Function constructor	Disallowed	Disallowed
arrows	Disallowed	Disallowed
[native code]	Disallowed	Disallowed
class expression with implicit constructor	Disallowed	Disallowed
class expression with explicit constructor	Disallowed	Disallowed
unicode escape sequences in identifiers	Disallowed	Disallowed
methods and computed property names	Disallowed	Disallowed

JSON superset

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Line separator can appear in string literals	Disallowed	Disallowed
Paragraph separator can appear in string literals	Disallowed	Disallowed

Well-formed JSON.stringify

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Well-formed JSON.stringify	Disallowed	Disallowed

ECMAScript 9 features

Object rest/spread properties

Feature	ECMAScript 2021 (ES12)	ES5 Standards
object rest properties	Supported	Not Supported
object spread properties	Supported	Not Supported

Promise.prototype.finally

Feature	ECMAScript 2021 (ES12)	ES5 Standards
basic support	Disallowed	Disallowed
don't change resolution value	Disallowed	Disallowed
change rejection value	Disallowed	Disallowed

Asynchronous iterators

Feature	ECMAScript 2021 (ES12)	ES5 Standards
async generators	Disallowed	Disallowed
for-await-of loops	Disallowed	Disallowed

s (dotAll) flag for regular expressions

Feature	ECMAScript 2021 (ES12)	ES5 Standards
s (dotAll) flag for regular expressions	Not Supported	Not Supported

RegExp named capture groups

Feature	ECMAScript 2021 (ES12)	ES5 Standards
RegExp named capture groups	Not Supported	Not Supported

RegExp Lookbehind Assertions

Feature	ECMAScript 2021 (ES12)	ES5 Standards
RegExp Lookbehind Assertions	Not Supported	Not Supported

RegExp Unicode Property Escapes

Feature	ECMAScript 2021 (ES12)	ES5 Standards
RegExp Unicode Property Escapes	Not Supported	Not Supported

Template literal revision

Feature	ECMAScript 2021 (ES12)	ES5 Standards
template literal revision	Disallowed	Disallowed

ECMAScript 8 features

Object static methods

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Object.values	Supported	Not Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Object.entries	Supported	Not Supported
Object.getOwnPropertyDescriptors	Supported	Not Supported
Object.getOwnPropertyDescriptors doesn't provide undefined descriptors	Not Supported	Not Supported

String padding

Feature	ECMAScript 2021 (ES12)	ES5 Standards
String.prototype.padStart	Supported	Not Supported
String.prototype.padEnd	Supported	Not Supported

Trailing commas in function syntax

Feature	ECMAScript 2021 (ES12)	ES5 Standards
in parameter lists	Supported	Not Supported
in argument lists	Supported	Not Supported

Async functions

Feature	ECMAScript 2021 (ES12)	ES5 Standards
return	Disallowed	Disallowed
throw	Disallowed	Disallowed
no line break between async and function	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
no "prototype" property	Disallowed	Disallowed
await	Disallowed	Disallowed
await, rejection	Disallowed	Disallowed
must await a value	Disallowed	Disallowed
can await non-Promise values	Disallowed	Disallowed
cannot await in parameters	Disallowed	Disallowed
async methods, object literals	Disallowed	Disallowed
async methods, classes	Disallowed	Disallowed
async arrow functions in methods, classes	Disallowed	Disallowed
async arrow functions	Disallowed	Disallowed
correct prototype chain	Disallowed	Disallowed
async function prototype, Symbol.toStringTag	Disallowed	Disallowed
async function constructor	Disallowed	Disallowed

Shared memory and atomics

Feature	ECMAScript 2021 (ES12)	ES5 Standards
SharedArrayBuffer	Disallowed	Disallowed
SharedArrayBuffer[Symbol.species]	Disallowed	Disallowed
SharedArrayBuffer.prototype.byteLength	Disallowed	Disallowed
SharedArrayBuffer.prototype.slice	Disallowed	Disallowed
SharedArrayBuffer.prototype[Symbol.toStringTag]	Disallowed	Disallowed
Atomics.add	Disallowed	Disallowed
Atomics.and	Disallowed	Disallowed
Atomics.compareExchange	Disallowed	Disallowed
Atomics.exchange	Disallowed	Disallowed
Atomics.wait	Disallowed	Disallowed
Atomics.wake	Disallowed	Disallowed
Atomics.isLockFree	Disallowed	Disallowed
Atomics.load	Disallowed	Disallowed
Atomics.or	Disallowed	Disallowed
Atomics.store	Disallowed	Disallowed
Atomics.sub	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Atomics.xor	Disallowed	Disallowed

Object.prototype getter/setter methods

Feature	ECMAScript 2021 (ES12)	ES5 Standards
<code>_defineGetter_</code>	Disallowed	Disallowed
<code>_defineGetter_</code> , symbols	Disallowed	Disallowed
<code>_defineGetter_</code> , <code>ToObject(this)</code>	Disallowed	Disallowed
<code>_defineSetter_</code>	Disallowed	Disallowed
<code>_defineSetter_</code> , symbols	Disallowed	Disallowed
<code>_defineSetter_</code> , <code>ToObject(this)</code>	Disallowed	Disallowed
<code>_lookupGetter_</code>	Disallowed	Disallowed
<code>_lookupGetter_</code> , prototype chain	Disallowed	Disallowed
<code>_lookupGetter_</code> , symbols	Disallowed	Disallowed
<code>_lookupGetter_</code> , <code>ToObject(this)</code>	Disallowed	Disallowed
<code>_lookupGetter_</code> , data properties can shadow accessors	Disallowed	Disallowed
<code>_lookupSetter_</code>	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
<code>_lookupSetter_</code> , prototype chain	Disallowed	Disallowed
<code>_lookupSetter_</code> , symbols	Disallowed	Disallowed
<code>_lookupSetter_</code> , <code>ToObject(this)</code>	Disallowed	Disallowed
<code>_lookupSetter_</code> , data properties can shadow accessors	Disallowed	Disallowed

Proxy internal calls, getter/setter methods

Feature	ECMAScript 2021 (ES12)	ES5 Standards
<code>_defineGetter_</code>	Disallowed	Disallowed
<code>_defineSetter_</code>	Disallowed	Disallowed
<code>_lookupGetter_</code>	Disallowed	Disallowed
<code>_lookupSetter_</code>	Disallowed	Disallowed

ECMAScript 7 features

Exponentiation (**) operator

Feature	ECMAScript 2021 (ES12)	ES5 Standards
basic support	Supported	Not Supported
assignment	Supported	Not Supported
early syntax error for unary negation without parentheses	Disallowed	Disallowed

Array.prototype.includes

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Array.prototype.includes	Supported	Not Supported
Array.prototype.includes generic	Not Supported	Not Supported
%TypedArray%.prototype.includes	Disallowed	Disallowed

ECMAScript 6 features

Proper tail calls (tail call optimization)

Feature	ECMAScript 2021 (ES12)	ES5 Standards
direct recursion	Disallowed	Disallowed
mutual recursion	Disallowed	Disallowed

Default function parameters

Feature	ECMAScript 2021 (ES12)	ES5 Standards
basic functionality	Supported	Not Supported
explicit undefined defers to the default	Supported	Not Supported
defaults can refer to previous parameters	Supported	Not Supported
arguments object interaction	Supported	Not Supported
temporal dead zone	Disallowed	Disallowed
separate scope	Supported	Not Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
new Function() support	Disallowed	Disallowed

Rest parameters

Feature	ECMAScript 2021 (ES12)	ES5 Standards
basic functionality	Supported	Not Supported
function 'length' property	Supported	Not Supported
arguments object interaction	Not Supported	Not Supported
can't be used in setters	Disallowed	Disallowed
new Function() support	Disallowed	Disallowed

Spread syntax for iterable objects

Feature	ECMAScript 2021 (ES12)	ES5 Standards
with arrays, in function calls	Supported	Not Supported
with arrays, in array literals	Supported	Not Supported
with sparse arrays, in function calls	Not Supported	Not Supported
with sparse arrays, in array literals	Supported	Not Supported
with strings, in function calls	Not Supported	Not Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
with strings, in array literals	Supported	Not Supported
with astral plane strings, in function calls	Not Supported	Not Supported
with astral plane strings, in array literals	Supported	Not Supported
with generator instances, in calls	Disallowed	Disallowed
with generator instances, in arrays	Disallowed	Disallowed
with generic iterables, in calls	Supported	Not Supported
with generic iterables, in arrays	Supported	Not Supported
with instances of iterables, in calls	Supported	Not Supported
with instances of iterables, in arrays	Supported	Not Supported
spreading non-iterables is a runtime error	Not Supported	Not Supported

Object literal extensions

Feature	ECMAScript 2021 (ES12)	ES5 Standards
computed properties	Supported	Not Supported
shorthand properties	Supported	Not Supported
shorthand methods	Supported	Not Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
string-keyed shorthand methods	Supported	Not Supported
computed shorthand methods	Supported	Not Supported
computed accessors	Supported	Not Supported

For-of loops

Feature	ECMAScript 2021 (ES12)	ES5 Standards
with arrays	Supported	Not Supported
with sparse arrays	Supported	Not Supported
with strings	Supported	Not Supported
with astral plane strings	Supported	Not Supported
with generator instances	Disallowed	Disallowed
with generic iterables	Supported	Not Supported
with instances of generic iterables	Supported	Not Supported
iterator closing, break	Supported	Not Supported
iterator closing, throw	Supported	Not Supported

Octal and binary literals

Feature	ECMAScript 2021 (ES12)	ES5 Standards
octal literals	Supported	Not Supported
binary literals	Supported	Not Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
octal supported by Number()	Not Supported	Not Supported
binary supported by Number()	Not Supported	Not Supported

Template literals

Feature	ECMAScript 2021 (ES12)	ES5 Standards
basic functionality	Supported	Not Supported
toString conversion	Supported	Not Supported
tagged template literals	Supported	Not Supported
passed array is frozen	Supported	Not Supported
line break normalization	Disallowed	Disallowed
TemplateStrings call site caching	Supported	Not Supported
TemplateStrings permanent caching	Supported	Not Supported

RegExp "y" and "u" flags

Feature	ECMAScript 2021 (ES12)	ES5 Standards
"y" flag	Supported	Not Supported
"y" flag, lastIndex	Supported	Not Supported
"u" flag	Not Supported	Not Supported
"u" flag, non-BMP Unicode characters	Not Supported	Not Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
"u" flag, Unicode code point escapes	Not Supported	Not Supported
"u" flag, case folding	Not Supported	Not Supported

Destructuring, declarations

Feature	ECMAScript 2021 (ES12)	ES5 Standards
with arrays	Supported	Not Supported
with sparse arrays	Supported	Not Supported
with strings	Supported	Not Supported
with astral plane strings	Supported	Not Supported
with generator instances	Disallowed	Disallowed
with generic iterables	Supported	Not Supported
with instances of generic iterables	Supported	Not Supported
iterator closing	Supported	Not Supported
trailing commas in iterable patterns	Supported	Not Supported
with objects	Supported	Not Supported
object destructuring with primitives	Supported	Not Supported
trailing commas in object patterns	Supported	Not Supported
throws on null and undefined	Supported	Not Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
computed properties	Supported	Not Supported
multiples in a single var statement	Supported	Not Supported
nested	Supported	Not Supported
in for-in loop heads	Supported	Not Supported
in for-of loop heads	Supported	Not Supported
in catch heads	Supported	Not Supported
rest	Supported	Not Supported
defaults	Supported	Not Supported
defaults, let temporal dead zone	Disallowed	Disallowed

Destructuring, assignment

Feature	ECMAScript 2021 (ES12)	ES5 Standards
with arrays	Supported	Not Supported
with sparse arrays	Supported	Not Supported
with strings	Supported	Not Supported
with astral plane strings	Supported	Not Supported
with generator instances	Disallowed	Disallowed
with generic iterables	Supported	Not Supported
with instances of generic iterables	Supported	Not Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
iterator closing	Supported	Not Supported
iterable destructuring expression	Supported	Not Supported
chained iterable destructuring	Supported	Not Supported
trailing commas in iterable patterns	Supported	Not Supported
with objects	Supported	Not Supported
object destructuring with primitives	Supported	Not Supported
trailing commas in object patterns	Supported	Not Supported
object destructuring expression	Supported	Not Supported
parenthesized left-hand-side is a syntax error	Disallowed	Disallowed
chained object destructuring	Supported	Not Supported
throws on null and undefined	Supported	Not Supported
computed properties	Supported	Not Supported
nested	Supported	Not Supported
rest	Supported	Not Supported
nested rest	Supported	Not Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
empty patterns	Supported	Not Supported
defaults	Supported	Not Supported

Destructuring, parameters

Feature	ECMAScript 2021 (ES12)	ES5 Standards
with arrays	Supported	Not Supported
with sparse arrays	Supported	Not Supported
with strings	Supported	Not Supported
with astral plane strings	Supported	Not Supported
with generator instances	Disallowed	Disallowed
with generic iterables	Supported	Not Supported
with instances of generic iterables	Supported	Not Supported
iterator closing	Supported	Not Supported
trailing commas in iterable patterns	Supported	Not Supported
with objects	Supported	Not Supported
object destructuring with primitives	Supported	Not Supported
trailing commas in object patterns	Supported	Not Supported
throws on null and undefined	Supported	Not Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
computed properties	Supported	Not Supported
nested	Supported	Not Supported
'arguments' interaction	Supported	Not Supported
new Function() support	Disallowed	Disallowed
in parameters, function 'length' property	Supported	Not Supported
rest	Supported	Not Supported
empty patterns	Supported	Not Supported
defaults	Supported	Not Supported
defaults, separate scope	Supported	Not Supported
defaults, new Function() support	Disallowed	Disallowed
aliased defaults, arrow function	Supported	Not Supported
shorthand defaults, arrow function	Supported	Not Supported
duplicate identifier	Disallowed	Disallowed

Unicode code point escapes

Feature	ECMAScript 2021 (ES12)	ES5 Standards
in strings	Supported	Not Supported
in identifiers	Not Supported	Not Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
in property key definitions	Not Supported	Not Supported
in property key accesses	Not Supported	Not Supported

New.target

Feature	ECMAScript 2021 (ES12)	ES5 Standards
in constructors	Not Supported	Not Supported
assignment is an early error	Disallowed	Disallowed

Const

Feature	ECMAScript 2021 (ES12)	ES5 Standards
basic support	Supported	Supported
is block-scoped	Supported	Not Supported
scope shadow resolution	Supported	Not Supported
cannot be in statements	Disallowed	Disallowed
redefining a const is an error	Disallowed	Disallowed
for loop statement scope	Supported	Not Supported
for-in loop iteration scope	Supported	Not Supported
for-of loop iteration scope	Supported	Not Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
temporal dead zone	Not Supported	Not Supported
basic support (strict mode)	Supported	Supported
is block-scoped (strict mode)	Supported	Not Supported
scope shadow resolution (strict mode)	Supported	Not Supported
cannot be in statements (strict mode)	Disallowed	Disallowed
redefining a const (strict mode)	Disallowed	Disallowed
for loop statement scope (strict mode)	Supported	Not Supported
for-in loop iteration scope (strict mode)	Supported	Not Supported
for-of loop iteration scope (strict mode)	Supported	Not Supported
temporal dead zone (strict mode)	Not Supported	Not Supported

Let

Feature	ECMAScript 2021 (ES12)	ES5 Standards
basic support	Supported	Not Supported
is block-scoped	Supported	Not Supported
scope shadow resolution	Supported	Not Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
cannot be in statements	Disallowed	Disallowed
for loop statement scope	Supported	Not Supported
temporal dead zone	Not Supported	Not Supported
for/for-in loop iteration scope	Supported	Not Supported
for-in loop binding shadowing parameter	Disallowed	Disallowed
basic support (strict mode)	Supported	Not Supported
is block-scoped (strict mode)	Supported	Not Supported
scope shadow resolution (strict mode)	Supported	Not Supported
cannot be in statements (strict mode)	Disallowed	Disallowed
for loop statement scope (strict mode)	Supported	Not Supported
temporal dead zone (strict mode)	Not Supported	Not Supported
for/for-in loop iteration scope (strict mode)	Supported	Not Supported
for-in loop binding shadowing parameter (strict mode)	Disallowed	Disallowed

Block-level function declaration

Feature	ECMAScript 2021 (ES12)	ES5 Standards
block-level function declaration	Supported	Not Supported

Arrow functions

Feature	ECMAScript 2021 (ES12)	ES5 Standards
0 parameters	Supported	Not Supported
1 parameter, no brackets	Supported	Not Supported
multiple parameters	Supported	Not Supported
lexical "this" binding	Supported	Not Supported
"this" unchanged by call or apply	Supported	Not Supported
can't be bound, can be curried	Supported	Not Supported
lexical "arguments" binding	Supported	Not Supported
no line break between parameters and =>	Disallowed	Disallowed
correct precedence	Disallowed	Disallowed
no "prototype" property	Not Supported	Not Supported
lexical "super" binding in constructors	Supported	Not Supported
lexical "super" binding in methods	Supported	Not Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
lexical "new.target" binding	Not Supported	Not Supported

Class

Feature	ECMAScript 2021 (ES12)	ES5 Standards
class statement	Supported	Not Supported
is block-scoped	Supported	Not Supported
class expression	Supported	Not Supported
anonymous class	Supported	Not Supported
constructor	Supported	Not Supported
prototype methods	Supported	Not Supported
string-keyed methods	Supported	Not Supported
computed prototype methods	Supported	Not Supported
optional semicolons	Supported	Not Supported
static methods	Supported	Not Supported
computed static methods	Supported	Not Supported
accessor properties	Supported	Not Supported
computed accessor properties	Supported	Not Supported
static accessor properties	Supported	Not Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
computed static accessor properties	Supported	Not Supported
class name is lexically scoped	Supported	Not Supported
computed names, temporal dead zone	Supported	Not Supported
methods aren't enumerable	Supported	Not Supported
implicit strict mode	Not Supported	Not Supported
constructor requires new	Supported	Not Supported
extends	Supported	Not Supported
extends expressions	Supported	Not Supported
extends null	Supported	Not Supported
new.target	Supported	Not Supported

Super

Feature	ECMAScript 2021 (ES12)	ES5 Standards
statement in constructors	Supported	Not Supported
expression in constructors	Supported	Not Supported
in methods, property access	Supported	Not Supported
in methods, method calls	Supported	Not Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
method calls use correct "this" binding	Supported	Not Supported
constructor calls use correct "new.target" binding	Supported	Not Supported
is statically bound	Supported	Not Supported
super() invokes the correct constructor	Supported	Not Supported

Generators

Feature	ECMAScript 2021 (ES12)	ES5 Standards
basic functionality	Disallowed	Disallowed
generator function expressions	Disallowed	Disallowed
correct "this" binding	Disallowed	Disallowed
can't use "this" with new	Disallowed	Disallowed
sending	Disallowed	Disallowed
%GeneratorPrototype %	Disallowed	Disallowed
%GeneratorPrototype % prototype chain	Disallowed	Disallowed
%GeneratorPrototype %.constructor	Disallowed	Disallowed
%GeneratorPrototype %.throw	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
%GeneratorPrototype %.return	Disallowed	Disallowed
yield operator precedence	Disallowed	Disallowed
yield *, arrays	Disallowed	Disallowed
yield *, sparse arrays	Disallowed	Disallowed
yield *, strings	Disallowed	Disallowed
yield *, astral plane strings	Disallowed	Disallowed
yield *, generator instances	Disallowed	Disallowed
yield *, generic iterables	Disallowed	Disallowed
yield *, instances of iterables	Disallowed	Disallowed
yield * on non-iterables is a runtime error	Disallowed	Disallowed
yield *, iterator closing	Disallowed	Disallowed
yield *, iterator closing via throw()	Disallowed	Disallowed
shorthand generator methods	Disallowed	Disallowed
string-keyed shorthand generator methods	Disallowed	Disallowed
computed shorthand generators	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
shorthand generator methods, classes	Disallowed	Disallowed
computed shorthand generators, classes	Disallowed	Disallowed
shorthand generators can't be constructors	Disallowed	Disallowed

Typed arrays

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Int8Array	Disallowed	Disallowed
Uint8Array	Disallowed	Disallowed
Uint8ClampedArray	Disallowed	Disallowed
Int16Array	Disallowed	Disallowed
Uint16Array	Disallowed	Disallowed
Int32Array	Disallowed	Disallowed
Uint32Array	Disallowed	Disallowed
Float32Array	Disallowed	Disallowed
Float64Array	Disallowed	Disallowed
DataView (Int8)	Disallowed	Disallowed
DataView (Uint8)	Disallowed	Disallowed
DataView (Int16)	Disallowed	Disallowed
DataView (Uint16)	Disallowed	Disallowed
DataView (Int32)	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
DataView (Uint32)	Disallowed	Disallowed
DataView (Float32)	Disallowed	Disallowed
DataView (Float64)	Disallowed	Disallowed
ArrayBuffer[Symbol.species]	Disallowed	Disallowed
constructors require new	Disallowed	Disallowed
constructors accept generic iterables	Disallowed	Disallowed
correct prototype chains	Disallowed	Disallowed
%TypedArray%.from	Disallowed	Disallowed
%TypedArray%.of	Disallowed	Disallowed
%TypedArray%.prototype.subarray	Disallowed	Disallowed
%TypedArray%.prototype.join	Disallowed	Disallowed
%TypedArray%.prototype.indexOf	Disallowed	Disallowed
%TypedArray%.prototype.lastIndexOf	Disallowed	Disallowed
%TypedArray%.prototype.slice	Disallowed	Disallowed
%TypedArray%.prototype.every	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
%TypedArray%.prototype.filter	Disallowed	Disallowed
%TypedArray%.prototype.forEach	Disallowed	Disallowed
%TypedArray%.prototype.map	Disallowed	Disallowed
%TypedArray%.prototype.reduce	Disallowed	Disallowed
%TypedArray%.prototype.reduceRight	Disallowed	Disallowed
%TypedArray%.prototype.reverse	Disallowed	Disallowed
%TypedArray%.prototype.some	Disallowed	Disallowed
%TypedArray%.prototype.sort	Disallowed	Disallowed
%TypedArray%.prototype.copyWithin	Disallowed	Disallowed
%TypedArray%.prototype.find	Disallowed	Disallowed
%TypedArray%.prototype.findIndex	Disallowed	Disallowed
%TypedArray%.prototype.fill	Disallowed	Disallowed
%TypedArray%.prototype.keys	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
%TypedArray%.prototype.values	Disallowed	Disallowed
%TypedArray%.prototype.entries	Disallowed	Disallowed
%TypedArray%.prototype[Symbol.iterator]	Disallowed	Disallowed
%TypedArray% [Symbol.species]	Disallowed	Disallowed

Map

Feature	ECMAScript 2021 (ES12)	ES5 Standards
basic functionality	Supported	Not Supported
constructor arguments	Supported	Not Supported
constructor requires new	Supported	Not Supported
constructor accepts null	Supported	Not Supported
constructor invokes set	Supported	Not Supported
iterator closing	Supported	Not Supported
Map.prototype.set returns this	Supported	Not Supported
-0 key converts to +0	Supported	Not Supported
Map.prototype.size	Supported	Not Supported
Map.prototype.delete	Supported	Not Supported
Map.prototype.clear	Supported	Not Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Map.prototype.forEach	Supported	Not Supported
Map.prototype.keys	Supported	Not Supported
Map.prototype.values	Supported	Not Supported
Map.prototype.entries	Supported	Not Supported
Map.prototype[Symbol.iterator]	Supported	Not Supported
Map.prototype isn't an instance	Supported	Not Supported
Map iterator prototype chain	Supported	Not Supported
Map[Symbol.species]	Supported	Not Supported

Set

Feature	ECMAScript 2021 (ES12)	ES5 Standards
basic functionality	Supported	Not Supported
constructor arguments	Supported	Not Supported
constructor requires new	Supported	Not Supported
constructor accepts null	Supported	Not Supported
constructor invokes add	Supported	Not Supported
iterator closing	Supported	Not Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Set.prototype.add returns this	Supported	Not Supported
-0 key converts to +0	Supported	Not Supported
Set.prototype.size	Supported	Not Supported
Set.prototype.delete	Supported	Not Supported
Set.prototype.clear	Supported	Not Supported
Set.prototype.forEach	Supported	Not Supported
Set.prototype.keys	Supported	Not Supported
Set.prototype.values	Supported	Not Supported
Set.prototype.entries	Supported	Not Supported
Set.prototype[Symbol.iterator]	Supported	Not Supported
Set.prototype isn't an instance	Supported	Not Supported
Set iterator prototype chain	Supported	Not Supported
Set[Symbol.species]	Supported	Not Supported

WeakMap

Feature	ECMAScript 2021 (ES12)	ES5 Standards
basic functionality	Disallowed	Disallowed
constructor arguments	Disallowed	Disallowed
constructor requires new	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
constructor accepts null	Disallowed	Disallowed
constructor invokes set	Disallowed	Disallowed
frozen objects as keys	Disallowed	Disallowed
iterator closing	Disallowed	Disallowed
WeakMap.prototype.set returns this	Disallowed	Disallowed
WeakMap.prototype.delete	Disallowed	Disallowed
no WeakMap.prototype.clear method	Disallowed	Disallowed
.has, .get and .delete methods accept primitives	Disallowed	Disallowed
WeakMap.prototype isn't an instance	Disallowed	Disallowed

WeakSet

Feature	ECMAScript 2021 (ES12)	ES5 Standards
basic functionality	Disallowed	Disallowed
constructor arguments	Disallowed	Disallowed
constructor requires new	Disallowed	Disallowed
constructor accepts null	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
constructor invokes add	Disallowed	Disallowed
iterator closing	Disallowed	Disallowed
WeakSet.prototype.add returns this	Disallowed	Disallowed
WeakSet.prototype.delete	Disallowed	Disallowed
no WeakSet.prototype.clear method	Disallowed	Disallowed
.has and .delete methods accept primitives	Disallowed	Disallowed
WeakSet.prototype isn't an instance	Disallowed	Disallowed

Proxy

Feature	ECMAScript 2021 (ES12)	ES5 Standards
constructor requires new	Disallowed	Disallowed
no "prototype" property	Disallowed	Disallowed
"get" handler	Disallowed	Disallowed
"get" handler, instances of proxies	Disallowed	Disallowed
"get" handler invariants	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
"set" handler	Disallowed	Disallowed
"set" handler, instances of proxies	Disallowed	Disallowed
"set" handler invariants	Disallowed	Disallowed
"has" handler	Disallowed	Disallowed
"has" handler, instances of proxies	Disallowed	Disallowed
"has" handler invariants	Disallowed	Disallowed
"deleteProperty" handler	Disallowed	Disallowed
"deleteProperty" handler invariant	Disallowed	Disallowed
"getOwnPropertyDescriptor" handler	Disallowed	Disallowed
"getOwnPropertyDescriptor" handler invariants	Disallowed	Disallowed
"defineProperty" handler	Disallowed	Disallowed
"defineProperty" handler invariants	Disallowed	Disallowed
"getPrototypeOf" handler	Disallowed	Disallowed
"getPrototypeOf" handler invariant	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
"setPrototypeOf" handler	Disallowed	Disallowed
"setPrototypeOf" handler invariant	Disallowed	Disallowed
"isExtensible" handler	Disallowed	Disallowed
"isExtensible" handler invariant	Disallowed	Disallowed
"preventExtensions" handler	Disallowed	Disallowed
"preventExtensions" handler invariant	Disallowed	Disallowed
"ownKeys" handler	Disallowed	Disallowed
"ownKeys" handler invariant	Disallowed	Disallowed
"apply" handler	Disallowed	Disallowed
"apply" handler invariant	Disallowed	Disallowed
"construct" handler	Disallowed	Disallowed
"construct" handler invariants	Disallowed	Disallowed
Proxy.revocable	Disallowed	Disallowed
Array.isArray support	Disallowed	Disallowed
JSON.stringify support	Disallowed	Disallowed

Reflect

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Reflect.get	Disallowed	Disallowed
Reflect.set	Disallowed	Disallowed
Reflect.has	Disallowed	Disallowed
Reflect.deleteProperty	Disallowed	Disallowed
Reflect.getOwnPropertyDescriptor	Disallowed	Disallowed
Reflect.defineProperty	Disallowed	Disallowed
Reflect.getPrototypeOf	Disallowed	Disallowed
Reflect.setPrototypeOf	Disallowed	Disallowed
Reflect.isExtensible	Disallowed	Disallowed
Reflect.preventExtensions	Disallowed	Disallowed
Reflect.ownKeys, string keys	Disallowed	Disallowed
Reflect.ownKeys, symbol keys	Disallowed	Disallowed
Reflect.apply	Disallowed	Disallowed
Reflect.construct	Disallowed	Disallowed
Reflect.construct sets new.target meta-property	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Reflect.construct creates instances from third argument	Disallowed	Disallowed
Reflect.construct, Array subclassing	Disallowed	Disallowed
Reflect.construct, RegExp subclassing	Disallowed	Disallowed
Reflect.construct, Function subclassing	Disallowed	Disallowed
Reflect.construct, Promise subclassing	Disallowed	Disallowed

Promise

Feature	ECMAScript 2021 (ES12)	ES5 Standards
basic functionality	Disallowed	Disallowed
constructor requires new	Disallowed	Disallowed
Promise.prototype isn't an instance	Disallowed	Disallowed
Promise.all	Disallowed	Disallowed
Promise.all, generic iterables	Disallowed	Disallowed
Promise.race	Disallowed	Disallowed
Promise.race, generic iterables	Disallowed	Disallowed
Promise[Symbol.species]	Disallowed	Disallowed

Symbol

Feature	ECMAScript 2021 (ES12)	ES5 Standards
basic functionality	Supported	Not Supported
typeof support	Supported	Not Supported
symbol keys are hidden to pre-ES6 code	Supported	Not Supported
Object.defineProperty support	Supported	Not Supported
symbols inherit from Symbol.prototype	Supported	Not Supported
cannot coerce to string or number	Not Supported	Not Supported
can convert with String()	Not Supported	Not Supported
new Symbol() throws	Supported	Not Supported
Object(symbol)	Not Supported	Not Supported
JSON.stringify ignores symbol primitives	Supported	Not Supported
JSON.stringify ignores symbol objects	Not Supported	Not Supported
global symbol registry	Supported	Not Supported

Well-known symbols

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Symbol.hasInstance	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Symbol.isConcatSpreadable	Disallowed	Disallowed
Symbol.iterator, existence	Disallowed	Disallowed
Symbol.iterator, arguments object	Disallowed	Disallowed
Symbol.species, existence	Disallowed	Disallowed
Symbol.species, Array.prototype.concat	Disallowed	Disallowed
Symbol.species, Array.prototype.filter	Disallowed	Disallowed
Symbol.species, Array.prototype.map	Disallowed	Disallowed
Symbol.species, Array.prototype.slice	Disallowed	Disallowed
Symbol.species, Array.prototype.splice	Disallowed	Disallowed
Symbol.species, RegExp.prototype[Symbol.split]	Disallowed	Disallowed
Symbol.species, Promise.prototype.then	Disallowed	Disallowed
Symbol.replace	Disallowed	Disallowed
Symbol.search	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Symbol.split	Disallowed	Disallowed
Symbol.match	Disallowed	Disallowed
Symbol.match, RegExp constructor	Disallowed	Disallowed
Symbol.match, String.prototype.startsWith	Disallowed	Disallowed
Symbol.match, String.prototype.endsWith	Disallowed	Disallowed
Symbol.match, String.prototype.includes	Disallowed	Disallowed
Symbol.toPrimitive	Disallowed	Disallowed
Symbol.toStringTag	Disallowed	Disallowed
Symbol.toStringTag affects existing built-ins	Disallowed	Disallowed
Symbol.toStringTag, new built-ins	Disallowed	Disallowed
Symbol.toStringTag, misc. built-ins	Disallowed	Disallowed
Symbol.unscopables	Disallowed	Disallowed

Object static methods

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Object.assign	Supported	Not Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Object.is	Supported	Not Supported
Object.getOwnPropertySymbols	Supported	Not Supported
Object.setPrototypeOf	Not Supported	Not Supported

Function "name" property

Feature	ECMAScript 2021 (ES12)	ES5 Standards
function statements	Supported	Supported
function expressions	Supported	Supported
new Function	Not Supported	Not Supported
bound functions	Not Supported	Not Supported
variables (function)	Supported	Not Supported
object methods (function)	Supported	Not Supported
accessor properties	Not Supported	Not Supported
shorthand methods	Supported	Not Supported
shorthand methods (no lexical binding)	Supported	Not Supported
symbol-keyed methods	Not Supported	Not Supported
class statements	Supported	Not Supported
class expressions	Supported	Not Supported
variables (class)	Not Supported	Not Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
object methods (class)	Not Supported	Not Supported
class prototype methods	Supported	Not Supported
class static methods	Supported	Not Supported
isn't writable, is configurable	Not Supported	Not Supported

String static methods

Feature	ECMAScript 2021 (ES12)	ES5 Standards
String.raw	Supported	Not Supported
String.fromCodePoint	Supported	Not Supported

String.prototype methods

Feature	ECMAScript 2021 (ES12)	ES5 Standards
String.prototype.codePointAt	Supported	Supported
String.prototype.normalize	Supported	Supported
String.prototype.repeat	Supported	Supported
String.prototype.startsWith	Supported	Supported
String.prototype.startsWith throws on RegExp	Not Supported	Not Supported
String.prototype.endsWith	Supported	Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
String.prototype.endsWith throws on RegExp	Not Supported	Not Supported
String.prototype.includes	Supported	Supported
String.prototype[Symbol.iterator]	Supported	Not Supported
String iterator prototype chain	Supported	Not Supported

RegExp.prototype properties

Feature	ECMAScript 2021 (ES12)	ES5 Standards
RegExp.prototype.flags	Supported	Not Supported
RegExp.prototype[Symbol.match]	Not Supported	Not Supported
RegExp.prototype[Symbol.replace]	Supported	Not Supported
RegExp.prototype[Symbol.split]	Supported	Not Supported
RegExp.prototype[Symbol.search]	Not Supported	Not Supported
RegExp[Symbol.species]	Supported	Not Supported

Array static methods

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Array.from, array-like objects	Supported	Not Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Array.from, generator instances	Disallowed	Disallowed
Array.from, generic iterables	Supported	Not Supported
Array.from, instances of generic iterables	Supported	Not Supported
Array.from map function, array-like objects	Supported	Not Supported
Array.from map function, generator instances	Disallowed	Disallowed
Array.from map function, generic iterables	Supported	Not Supported
Array.from map function, instances of iterables	Supported	Not Supported
Array.from, iterator closing	Supported	Not Supported
Array.of	Supported	Not Supported
Array[Symbol.species]	Supported	Not Supported

Array.prototype methods

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Array.prototype.copyWithin	Supported	Not Supported
Array.prototype.find	Supported	Not Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Array.prototype.findIndex	Supported	Not Supported
Array.prototype.fill	Supported	Not Supported
Array.prototype.keys	Supported	Not Supported
Array.prototype.values	Supported	Not Supported
Array.prototype.entries	Supported	Not Supported
Array.prototype[Symbol.iterator]	Supported	Not Supported
Array iterator prototype chain	Supported	Not Supported
Array.prototype[Symbol.unscopables]	Supported	Not Supported

Number properties

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Number.isFinite	Supported	Not Supported
Number.isInteger	Supported	Not Supported
Number.isSafeInteger	Supported	Not Supported
Number.isNaN	Supported	Not Supported
Number.parseFloat	Disallowed	Disallowed
Number.parseInt	Disallowed	Disallowed
Number.EPSILON	Supported	Not Supported
Number.MIN_SAFE_INTEGER	Supported	Not Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Number.MAX_SAFE_INTEGER	Supported	Not Supported

Math methods

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Math.clz32	Supported	Not Supported
Math.imul	Supported	Not Supported
Math.sign	Supported	Not Supported
Math.log10	Supported	Not Supported
Math.log2	Supported	Not Supported
Math.log1p	Supported	Not Supported
Math.expm1	Supported	Not Supported
Math.cosh	Supported	Not Supported
Math.sinh	Supported	Not Supported
Math.tanh	Supported	Not Supported
Math.acosh	Supported	Not Supported
Math.asinh	Supported	Not Supported
Math.atanh	Supported	Not Supported
Math.trunc	Supported	Not Supported
Math.fround	Supported	Not Supported
Math.cbrt	Supported	Not Supported
Math.hypot	Supported	Not Supported

Date.prototype[Symbol.toPrimitive]

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Date.prototype[Symbol.toPrimitive]	Supported	Not Supported

Array is subclassable

Feature	ECMAScript 2021 (ES12)	ES5 Standards
length property (accessing)	Disallowed	Disallowed
length property (setting)	Disallowed	Disallowed
correct prototype chain	Disallowed	Disallowed
Array.isArray support	Disallowed	Disallowed
Array.prototype.concat	Disallowed	Disallowed
Array.prototype.filter	Disallowed	Disallowed
Array.prototype.map	Disallowed	Disallowed
Array.prototype.slice	Disallowed	Disallowed
Array.prototype.splice	Disallowed	Disallowed
Array.from	Disallowed	Disallowed
Array.of	Disallowed	Disallowed

RegExp is subclassable

Feature	ECMAScript 2021 (ES12)	ES5 Standards
basic functionality	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
correct prototype chain	Disallowed	Disallowed
RegExp.prototype.exec	Disallowed	Disallowed
RegExp.prototype.test	Disallowed	Disallowed

Function is subclassable

Feature	ECMAScript 2021 (ES12)	ES5 Standards
can be called	Disallowed	Disallowed
correct prototype chain	Disallowed	Disallowed
can be used with "new"	Disallowed	Disallowed
Function.prototype.cal	Disallowed	Disallowed
Function.prototype.ap	Disallowed	Disallowed
Function.prototype.bi	Disallowed	Disallowed

Promise is subclassable

Feature	ECMAScript 2021 (ES12)	ES5 Standards
basic functionality	Disallowed	Disallowed
correct prototype chain	Disallowed	Disallowed
Promise.all	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Promise.race	Disallowed	Disallowed

Miscellaneous subclassables

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Boolean is subclassable	Disallowed	Disallowed
Number is subclassable	Disallowed	Disallowed
String is subclassable	Disallowed	Disallowed
Error is subclassable	Disallowed	Disallowed
Map is subclassable	Disallowed	Disallowed
Set is subclassable	Disallowed	Disallowed

Prototype of bound functions

Feature	ECMAScript 2021 (ES12)	ES5 Standards
basic functions	Disallowed	Disallowed
generator functions	Disallowed	Disallowed
arrow functions	Disallowed	Disallowed
classes	Disallowed	Disallowed
subclasses	Disallowed	Disallowed

Proxy, internal 'get' calls

Feature	ECMAScript 2021 (ES12)	ES5 Standards
ToPrimitive	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
CreateListFromArrayLike	Disallowed	Disallowed
instanceof operator	Disallowed	Disallowed
HasBinding	Disallowed	Disallowed
CreateDynamicFunction	Disallowed	Disallowed
ClassDefinitionEvaluation	Disallowed	Disallowed
IteratorComplete, IteratorValue	Disallowed	Disallowed
ToPropertyDescriptor	Disallowed	Disallowed
Object.assign	Disallowed	Disallowed
Object.defineProperties	Disallowed	Disallowed
Function.prototype.bind	Disallowed	Disallowed
Error.prototype.toString	Disallowed	Disallowed
String.raw	Disallowed	Disallowed
RegExp constructor	Disallowed	Disallowed
RegExp.prototype.flags	Disallowed	Disallowed
RegExp.prototype.test	Disallowed	Disallowed
RegExp.prototype.toString	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
RegExp.prototype[Symbol.match]	Disallowed	Disallowed
RegExp.prototype[Symbol.replace]	Disallowed	Disallowed
RegExp.prototype[Symbol.search]	Disallowed	Disallowed
RegExp.prototype[Symbol.split]	Disallowed	Disallowed
Array.from	Disallowed	Disallowed
Array.prototype.concat	Disallowed	Disallowed
Array.prototype iteration methods	Disallowed	Disallowed
Array.prototype.pop	Disallowed	Disallowed
Array.prototype.reverse	Disallowed	Disallowed
Array.prototype.shift	Disallowed	Disallowed
Array.prototype.splice	Disallowed	Disallowed
Array.prototype.toString	Disallowed	Disallowed
JSON.stringify	Disallowed	Disallowed
Promise resolve functions	Disallowed	Disallowed
String.prototype.match	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
String.prototype.replace	Disallowed	Disallowed
String.prototype.search	Disallowed	Disallowed
String.prototype.split	Disallowed	Disallowed
Date.prototype.toJSON	Disallowed	Disallowed

Proxy, internal 'set' calls

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Object.assign	Disallowed	Disallowed
Array.from	Disallowed	Disallowed
Array.of	Disallowed	Disallowed
Array.prototype.copyWithin	Disallowed	Disallowed
Array.prototype.fill	Disallowed	Disallowed
Array.prototype.pop	Disallowed	Disallowed
Array.prototype.push	Disallowed	Disallowed
Array.prototype.reverse	Disallowed	Disallowed
Array.prototype.shift	Disallowed	Disallowed
Array.prototype.splice	Disallowed	Disallowed
Array.prototype.unshift	Disallowed	Disallowed

Proxy, internal 'defineProperty' calls

Feature	ECMAScript 2021 (ES12)	ES5 Standards
[[Set]]	Disallowed	Disallowed
SetIntegrityLevel	Disallowed	Disallowed

Proxy, internal 'deleteProperty' calls

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Array.prototype.copyWithin	Disallowed	Disallowed
Array.prototype.pop	Disallowed	Disallowed
Array.prototype.reverse	Disallowed	Disallowed
Array.prototype.shift	Disallowed	Disallowed
Array.prototype.splice	Disallowed	Disallowed
Array.prototype.unshift	Disallowed	Disallowed

Proxy, internal 'getOwnPropertyDescriptor' calls

Feature	ECMAScript 2021 (ES12)	ES5 Standards
[[Set]]	Disallowed	Disallowed
Object.assign	Disallowed	Disallowed
Object.prototype.hasOwnProperty	Disallowed	Disallowed
Function.prototype.bind	Disallowed	Disallowed

Proxy, internal 'ownKeys' calls

Feature	ECMAScript 2021 (ES12)	ES5 Standards
SetIntegrityLevel	Disallowed	Disallowed
TestIntegrityLevel	Disallowed	Disallowed
SerializeJSONObject	Disallowed	Disallowed

Object static methods accept primitives

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Object.getPrototypeOf	Disallowed	Disallowed
Object.getOwnPropertyDescriptor	Disallowed	Disallowed
Object.getOwnPropertyNames	Disallowed	Disallowed
Object.seal	Disallowed	Disallowed
Object.freeze	Disallowed	Disallowed
Object.preventExtensions	Disallowed	Disallowed
Object.isSealed	Disallowed	Disallowed
Object.isFrozen	Disallowed	Disallowed
Object.isExtensible	Disallowed	Disallowed
Object.keys	Disallowed	Disallowed

Own property order

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Object.keys	Disallowed	Disallowed
Object.getOwnPropertyNames	Disallowed	Disallowed
Object.assign	Disallowed	Disallowed
JSON.stringify	Disallowed	Disallowed
JSON.parse	Disallowed	Disallowed
Reflect.ownKeys, string key order	Disallowed	Disallowed
Reflect.ownKeys, symbol key order	Disallowed	Disallowed

Updated identifier syntax

Feature	ECMAScript 2021 (ES12)	ES5 Standards
var á, ¯;	Disallowed	Disallowed
var ð, €;	Disallowed	Disallowed
no escaped reserved words as identifiers	Disallowed	Disallowed

Non-strict function semantics

Feature	ECMAScript 2021 (ES12)	ES5 Standards
hoisted block-level function declaration	Disallowed	Disallowed
labeled function statements	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
function statements in if-statement clauses	Disallowed	Disallowed

__proto__ in object literals

Feature	ECMAScript 2021 (ES12)	ES5 Standards
basic support	Disallowed	Disallowed
multiple __proto__ is an error	Disallowed	Disallowed
not a computed property	Disallowed	Disallowed
not a shorthand property	Disallowed	Disallowed
not a shorthand method	Disallowed	Disallowed

Object.prototype.__proto__

Feature	ECMAScript 2021 (ES12)	ES5 Standards
get prototype	Disallowed	Disallowed
set prototype	Disallowed	Disallowed
absent from Object.create(null)	Disallowed	Disallowed
present in hasOwnProperty()	Disallowed	Disallowed
correct property descriptor	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
present in Object.getOwnPropertyNames()	Disallowed	Disallowed

String.prototype HTML methods

Feature	ECMAScript 2021 (ES12)	ES5 Standards
existence	Disallowed	Disallowed
tags' names are lowercase	Disallowed	Disallowed
quotes in arguments are escaped	Disallowed	Disallowed

RegExp.prototype.compile

Feature	ECMAScript 2021 (ES12)	ES5 Standards
basic functionality	Disallowed	Disallowed
returns this	Disallowed	Disallowed

RegExp syntax extensions

Feature	ECMAScript 2021 (ES12)	ES5 Standards
hyphens in character sets	Disallowed	Disallowed
invalid character escapes	Disallowed	Disallowed
invalid control-character escapes	Disallowed	Disallowed
invalid Unicode escapes	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
invalid hexadecimal escapes	Disallowed	Disallowed
incomplete patterns and quantifiers	Disallowed	Disallowed
octal escape sequences	Disallowed	Disallowed
invalid backreferences become octal escapes	Disallowed	Disallowed

ECMAScript 5 features

Object/array literal extensions

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Getter accessors	Supported	Supported
Setter accessors	Supported	Supported
Trailing commas in object literals	Supported	Supported
Trailing commas in array literals	Supported	Supported
Reserved words as property names	Supported	Supported

Object static methods

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Object.create	Supported	Supported
Object.defineProperty	Supported	Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Object.defineProperties	Supported	Supported
Object.getPrototypeOf	Supported	Supported
Object.keys	Supported	Supported
Object.seal	Supported	Supported
Object.freeze	Supported	Supported
Object.preventExtensions	Supported	Supported
Object.isSealed	Supported	Supported
Object.isFrozen	Supported	Supported
Object.isExtensible	Supported	Supported
Object.getOwnPropertyDescriptor	Supported	Supported
Object.getOwnPropertyNames	Supported	Supported

Array methods

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Array.isArray	Supported	Supported
Array.prototype.indexOf	Supported	Supported
Array.prototype.lastIndexOf	Supported	Supported
Array.prototype.every	Supported	Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Array.prototype.some	Supported	Supported
Array.prototype.forEach	Supported	Supported
Array.prototype.map	Supported	Supported
Array.prototype.filter	Supported	Supported
Array.prototype.reduce	Supported	Supported
Array.prototype.reduceRight	Supported	Supported
Array.prototype.sort: compareFn must be function or undefined	Not Supported	Not Supported
Array.prototype.sort: compareFn may be explicit undefined	Supported	Supported

String properties and methods

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Property access on strings	Supported	Supported
String.prototype.split	Supported	Not Supported
String.prototype.trim	Supported	Supported

Date methods

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Date.prototype.toISOString	Supported	Supported

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Date.now	Supported	Supported
Date.prototype.toJSON	Not Supported	Not Supported

Immutable globals

Feature	ECMAScript 2021 (ES12)	ES5 Standards
undefined	Not Supported	Supported
NaN	Not Supported	Supported
Infinity	Not Supported	Supported

Number methods

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Number.prototype.toExponential rounds properly	Supported	Supported
Number.prototype.toExponential throws on Â±Infinity fractionDigits	Supported	Supported
Number.prototype.toExponential does not throw on edge cases	Supported	Supported

Strict mode

Feature	ECMAScript 2021 (ES12)	ES5 Standards
reserved words	Disallowed	Disallowed
"this" is undefined in functions	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
"this" is not coerced to object in primitive methods	Disallowed	Disallowed
"this" is not coerced to object in primitive accessors	Disallowed	Disallowed
legacy octal is a SyntaxError	Disallowed	Disallowed
assignment to unresolvable identifiers is a ReferenceError	Disallowed	Disallowed
assignment to eval or arguments is a SyntaxError	Disallowed	Disallowed
assignment to non-writable properties is a TypeError	Disallowed	Disallowed
eval or arguments bindings is a SyntaxError	Disallowed	Disallowed
arguments.caller removed or is a TypeError	Disallowed	Disallowed
arguments.callee is a TypeError	Disallowed	Disallowed
(function(){}).caller and (function() {}).arguments is a TypeError	Disallowed	Disallowed

Feature	ECMAScript 2021 (ES12)	ES5 Standards
arguments is unmapped	Disallowed	Disallowed
eval() can't create bindings	Disallowed	Disallowed
deleting bindings is a SyntaxError	Disallowed	Disallowed
deleting non-configurable properties is a TypeError	Disallowed	Disallowed
"with" is a SyntaxError	Disallowed	Disallowed
repeated parameter names is a SyntaxError	Disallowed	Disallowed
function expressions with matching name and argument are valid	Disallowed	Disallowed

Function.prototype.bind

Feature	ECMAScript 2021 (ES12)	ES5 Standards
Function.prototype.bind	Supported	Supported

JSON

Feature	ECMAScript 2021 (ES12)	ES5 Standards
JSON	Supported	Supported

Porting code to ES5 standards mode scripts

ES5 standards mode catches errors that compatibility mode allows.

Things to watch for when porting code from existing scripts to new scoped scripts using ES5 standards mode.

ECMAScript5 evaluates the term `new Boolean(false)` to true. In compatibility mode, it evaluated to false.

ECMAScript5 throws an EcmaError when a non-existent property is referenced. In compatibility mode no error was thrown.

ECMAScript5 throws an EcmaError when a non-existent function is called. In compatibility mode, no error was thrown.

ECMAScript5 correctly handles new lines. In the past, a newline character after a comment was recognized, which is wrong. In this example, in compatibility mode, all three functions are called. In ECMAScript5, only the first function is called.

```
var expr = doFoo(); // do foo
               doBar(); // do bar
               finish(); // all done
eval(expr);
```

ECMAScript5 correctly handles postfix increment and decrement. In this example, in compatibility mode, the variable `x` gets the incremented value, which is wrong.

```
var x = gr.limit++;
```

JavaScript API Context-sensitive help

The syntax editor can display context-sensitive API information.

JavaScript API Context-sensitive help includes the ability to:

- List script elements that are valid at the cursor's location. The system displays suggestions in a pop-up window.
- Add a selected script element at the cursor's location. If the cursor is within or adjacent to a partial entry, the system completes the entry with the selected script element.
- View API documentation for a selected suggestion.
- View the expected parameters and format of the current script element.

If the cursor is adjacent to a text string, the system searches for script elements that start with this text string. For example, while the cursor is within or adjacent to the string `GlideR`, the system displays script elements such as:

- `GlideRecord`
- `GlideRecordSecure`

Context-sensitive suggestions are based on script type. For example, when working on a business rule, only suggestions from the server API and for objects such as current and previous display. When working on a client script, the system only displays suggestions from the client API.

Client-side scripting

Run JavaScript on the client (web browser) when client-based events occur, such as when a form loads, after form submission, or when a field changes value.

The client-side [Glide](#) API provides classes and methods that you can use in client scripts.

Before you begin

Understand the limitations of your scripting environment. For example, client scripts run on forms in the Service Portal or Mobile environments can only include certain APIs. For more information, see [Mobile client GlideForm \(g form\) scripting and migration](#).

- [Client-side scripting design and processing](#)

Well-designed client scripts can reduce the amount of time it takes users to complete a form.

- [Client scripts](#)

Client scripts allow the system to run JavaScript on the client (web browser) when client-based events occur, such as when a form loads, after form submission, or when a field changes value.

- [UI scripts](#)

UI scripts provide a way to package client-side JavaScript into a reusable form, similar to how script includes store server-side JavaScript. Administrators can create UI scripts and run them from client scripts and other client-side script objects and from HTML code.

- [Catalog client scripts](#)

Client-side scripts can add dynamic effects and validation to forms. Scripts can apply to service catalog items or variable sets, allowing administrators to use the same functionality that is available on other forms.

- [Mobile client GlideForm \(g_form\) scripting and migration](#)

Client scripting for mobile is identical to scripting for the web, with some exceptions. All new scripts must conform to certain guidelines. The following items are affected on the mobile platform: client scripts, UI policies, navigator modules, and UI actions.

- [AJAX](#)

AJAX (asynchronous JavaScript and XML) is a group of interrelated, client-side development techniques used to create asynchronous Web applications.

Client-side scripting design and processing

Well-designed client scripts can reduce the amount of time it takes users to complete a form.

Proper client-side processing depends on the form loading first. Making record updates prior to form load can produce unexpected results that bypass client-side processing.

If you create client scripts to control field values on a form, you must use another method to control these field values in a list. You can:

- Disable list editing for the table.
- Create appropriate business rules or access controls for list editing.
- Create data policies.
- Create a separate `onCellEdit` client script.

Restrict list editing

If you create UI policies or client scripts for fields on a form, you must use another method to ensure that data in those fields is similarly controlled in a list.

With the exception of onCellEdit client scripts, UI policies and client scripts apply to forms only. Use the following methods to restrict list editing when using client scripts:

- Disable list editing for the table.
- Create appropriate business rules or access controls for list editing.
- Create data policies.
- Create a separate onCellEdit client script.

Minimize server lookups

Use client data as much as possible to eliminate the need for time-consuming server lookups.

Client scripting uses either data available on the client or data retrieved from the server. The top ways to get information from the server are g_scratchpad and asynchronous GlideAjax lookup.

The primary difference between these methods is that g_scratchpad is sent once when a form is loaded (information is pushed from the server to the client), whereas GlideAjax is dynamically triggered when the client requests information from the server.

Note: GlideRecord and g_form.getReference() are also available for retrieving server information. However, these methods are no longer recommended due to their performance impact. Both methods retrieve all fields in the requested GlideRecord when most cases only require one field.

Example: Retrieve server data using g_scratchpad

The g_scratchpad object passes information from the server to the client, such as when the client requires information not available on the form.

For example, if you have a client script that needs to access the field `u_retrieve`, and the field is not on the form, the data is not available to the client script. A typical solution to this situation is to place the field on the form and then always hide it with a client script or UI policy. While this solution may be faster to configure, it is slower to execute.

If you know what information the client needs from the server before the form is loaded, a display business rule can create `g_scratchpad` properties to hold this information. The `g_scratchpad` is sent to the client when the form is requested, making it available to all client-side scripting methods. This is a very efficient means of sending information from the server to the client. However, you can only load data this way when the form is loaded. The business rule cannot be triggered dynamically. In those cases, use an asynchronous GlideAjax call.

For example, assume you open an incident and need to pass this information to the client:

- The value of the system property `css.base.color`
- Whether or not the current record has attachments
- The name of the caller's manager

A display business rule sends this information to the client using the following script:

```
g_scratchpad.css = gs.getProperty('css.base.color');
g_scratchpad.hasAttachments = current.hasAttachments();
g_scratchpad.managerName = current.caller_id.manager.getDisplayValue();
```

To access scratchpad data using a client script:

```
// Check if the form has attachments
if (g_scratchpad.hasAttachments)
    // do something interesting here
else
    alert('You need to attach a form signed by ' + g_scratchpad.managerName);
```

Example: Retrieve server data using asynchronous GlideAjax

Asynchronous GlideAjax allows you to dynamically request information from the server.

This script compares the support group of the CI and the assignment group of the incident by name:

```
//Alert if the assignment groups name matches the support group
function onChange(control, oldValue, newValue, isLoading)
{
    if (isLoading)
        return;

    var ga = new GlideAjax('ciCheck');

    ga.addParam('sysparm_name', 'getCiSupportGroup');
    ga.addParam('sysparm_ci', g_form.getValue('cmdb_ci'));
    ga.addParam('sysparm_ag', g_form.getValue('assignment_group'));
    ga.getXML(doAlert); // Always try to use asynchronous (getXML) calls rather than synchronous (getXMLWait)
}

// Callback function to process the response returned from the server
function doAlert(response) {

    var answer = response.responseXML.documentElement.getAttribute("answer");

    alert(answer);
}
```

This script relies on the accompanying script include:

```
var ciCheck = Class.create();

ciCheck.prototype = Object.extendsObject(AbstractAjaxProcessor, {
    getCiSupportGroup: function() {

        var retVal = '';// Return value
        var ciID   = this.getParameter('sysparm_ci');
        var agID   = this.getParameter('sysparm_ag');
        var ciRec  = new GlideRecord('cmdb_ci');
```

```
// If we can read the record, check if the sys_ids  
match  
    if (ciRec.get(ciID)) {  
        if (ciRec.getValue('support_group') == agID)  
            retVal = 'CI support group and assignment  
group match';  
        else  
            retVal = 'CI support group and assignment  
group do not match';  
  
        // Can't read the CI, then they don't match  
    } else {  
        retVal = 'CI support group and assignment grou  
p do not match';  
    }  
  
    return retVal;  
}  
});
```

Use the `setValue()` `displayValue` parameter for reference fields

When using `setValue()` on a reference field, include the `displayValue` parameter to avoid additional server calls.

When using `setValue()` on a reference field, be sure to include the reference field display value as the 3rd parameter. If you set the value without the `displayValue`, the instance does a synchronous call to retrieve the display value for the record you specified. This extra round trip to the server can impact performance.

Example

This example demonstrates the incorrect way to call `setValue`:

```
var id = '5137153cc611227c000bbd1bd8cd2005';  
  
g_form.setValue('assigned_to', id); // Client needs to go  
back to the server to  
                                // fetch the name tha  
t goes with this ID
```

Example

Instead, include the display value as an optional parameter in setValue():

```
var id = '5137153cc611227c000bbd1bd8cd2005';
var name = 'Fred Luddy';

g_form.setValue('assigned_to', id, name); // No server call required
```

Use UI policy instead of a client script

When possible, consider using a UI policy instead of a client script.

UI policies provide these benefits over client scripts:

- UI policies have an **Order** field to allow full control over the order in which client-side operations take place.
- UI policies do not require scripting to make a field mandatory, read-only, or visible.

Note: UI policies apply after client scripts.

Validate input using client scripts

An excellent use for client scripts is validating input from the user.

This validation improves the user experience because the user finds out if there are data issues before submitting the information.

Example

An example of validation is to verify that the **Impact** field value is valid with the **Priority** field value. In this example, **Low** impact is not allowed with **High** priority.

```
if (g_form.getValue('impact') == '3' && g_form.getValue('priority') == '1')
    g_form.showFieldMsg('impact', getMessage('Low impact not allowed with High priority'), 'error');
```

Set client script order

Control the order of execution for your client scripts using the Order field. To avoid having two or more client scripts run concurrently and then conflict, you can add an order for the scripts to run in.

Before you begin

Role required: admin

About this task

Adding an order to the client script creates a processing sequence, ordered from lowest to highest number. If two scripts conflict, the client script with the lower number executes first.

Procedure

1. Navigate to **All > System Definition > Client Script** and open an existing client script or click **New**.
2. [Configuring the form layout](#) to include the **Order** field.
3. Add a number to the order field based on what order you want it to run in relation to other client scripts. Choose a lower number for the script you want to execute first.

Avoid DOM manipulation

Avoid Document Object Model (DOM) manipulation if possible. It can cause a maintainability issue when browsers are updated.

Instead, use the GlideForm API or consider a different approach for the solution. In general, when using DOM manipulation methods, you have to reference an element in the DOM by ID or using a CSS selector. When referencing out-of-box DOM elements, there is a risk that the element ID or placement within the DOM could change, causing the code to stop working and/or generate errors. Use forethought, caution, and have a full understanding of the risk you are incurring. Review these objects and reduce the use of DOM manipulation methods as much as possible.

Avoid global client scripts

A global client script is any client script where the selected Table is Global. Global client scripts have no table restrictions, therefore they will load on every page in the system introducing browser load delay in the process.

There is no benefit to loading this kind of scripts on every page.

As an alternative, and for a more modular and scalable approach, consider moving client scripts to a base table (such as Task[task] or Configuration Item[cmdb_ci]) that can be derived for all the child/extending tables. This eliminates the system loading the scripts on every form in the UI - such as home pages or Service Catalog where they are rarely (if ever) needed.

Enclose code in functions

Enclose the code in a client script inside a function.

Client scripts without a function cause issues with variable scope. When code is not enclosed in a function, variables and other objects are available and shared to all other client-side scripts. If you are using the same variable names, it is possible that they could collide. This can lead to unexpected consequences that are difficult to troubleshoot.

Consider this example:

```
var state = "6";

function onSubmit() {

    if(g_form.getValue('incident_state') == state) {
        alert("This incident is Resolved");
    }
}
```

Because the state variable is not enclosed in a function, all client-side scripts, have access to it. Other scripts may also use the common variable name state. The duplicate names can conflict and lead to unexpected results. These issues are difficult to isolate and resolve. To avoid this issue, ensure that all code is wrapped in a function:

```
function onSubmit() {
```

```
var state = "6";

if(g_form.getValue('incident_state') == state) {
    alert("This incident is Resolved");
}
```

This solution is much safer because the scope of the variable state is limited to the onSubmit() function. Therefore, the state variable does not conflict with state variables in other client-side scripts.

Run only necessary scripts

To avoid running time-consuming scripts unnecessarily, make sure that client scripts perform only necessary tasks.

The following examples demonstrate improvements to the initial code sample. Each example demonstrates a particular enhancement to the script to improve performance and avoid unnecessary calls.

Example

Remember that client scripts have no **Condition** field. This means that onLoad() and onChange() scripts run in their entirety every time the appropriate form is loaded. This example is an inefficient onChange() client script set to run when the **Configuration item** field changes.

```
//Set Assignment Group to CI's support group if assignment group is empty
function onChange(control, oldValue, newValue, isLoading)
{
    var ciSupportGroup = g_form.getReference('cmdb_ci').support_group;
    if (ciSupportGroup != '' && g_form.getValue('assignment_group') != '')
        g_form.setValue('assignment_group', ciRec.support_group.sys_id);
}
```

Example

This example improves upon the first by replacing the `getReference()` or `GlideRecord` lookup with an asynchronous `GlideAjax` call.

```
//Set Assignment Group to support group if assignment group is empty
function onChange(control, oldValue, newValue, isLoading) {
    var ga = new GlideAjax('ciCheck');

    ga.addParam('sysparm_name', 'getSupportGroup');
    ga.addParam('sysparm_ci', g_form.getValue('cmdb_ci'));
    ga.getXML(setAssignmentGroup);
}

function setAssignmentGroup(response) {
    var answer = response.responseXML.documentElement.getAttribute("answer");

    g_form.setValue('assignment_group', answer);
}
```

Example

The `isLoading` flag is the simplest way to prevent unnecessary code from taking up browser time in `onChange` scripts. The `isLoading` flag should be used at the beginning of any script that is not required to run when the form is loading. There is no need to run this script on a form load because the logic would have already run when the field was last changed. Adding the `isLoading` check to the script prevents it from doing a `cmdb_ci` lookup on every form load.

The `isTemplate` flag indicates that a template is loading.

```
//Set Assignment Group to CI's support group if assignment group is empty
function onChange(control, oldValue, newValue, isLoading, isTemplate) {
    if (isLoading)
```

```
        return;

    var ga = new GlideAjax('ciCheck');

    ga.addParam('sysparm_name', 'getSupportGroup');
    ga.addParam('sysparm_ci', g_form.getValue('cmdb_ci'));
    ga.getXML(setAssignmentGroup);
}

function setAssignmentGroup(response) {

    var answer = response.responseXML.documentElement.getAttribute("answer");

    g_form.setValue('assignment_group', answer);
}
```

If the onChange script should run during loading, use the following convention:

```
function onChange(control, oldValue, newValue, isLoading,
isTemplate) {

    if (isLoading) {}; // run during loading

    // rest of script here

}
```

Example

The newValue check tells this script to continue only if there is a valid value in the relevant field. This prevents the script from running when the field value is removed or blanked out. This also ensures that there will always be a valid value available when the rest of the script runs.

```
//Set Assignment Group to CI's support group if assignment group is empty
function onChange(control, oldValue, newValue, isLoading,
isTemplate) {

    if (isLoading)
        return;

    if (newValue) {
```

```
var ga = new GlideAjax('ciCheck');

ga.addParam('sysparm_name', 'getSupportGroup');
ga.addParam('sysparm_ci', g_form.getValue('cmdb_ci'));
}
}

function setAssignmentGroup(response) {
    var answer = response.responseXML.documentElement.getAttribute("answer");
    g_form.setValue('assignment_group', answer);
}
```

Example

To have the script react to a value that changes after the form loads, use the newValue != oldValue check.

Note: This example does not catch users changing a value and then changing it back to its original value.

```
//Set Assignment Group to CI's support group if assignment group is empty
function onChange(control, oldValue, newValue, isLoading,
isTemplate) {

    if (isLoading)
        return;

    if (newValue) {
        if (newValue != oldValue) {
            var ga = new GlideAjax('ciCheck');

            ga.addParam('sysparm_name', 'getSupportGroup');
;
            ga.addParam('sysparm_ci', g_form.getValue('cmdb_ci'));
            ga.getXML(setAssignmentGroup);
        }
    }
}
```

```
}
```

```
function setAssignmentGroup(response) {
```

```
    var answer = response.responseXML.documentElement.getAttribute("answer");
```

```
    g_form.setValue('assignment_group', answer);
```

```
}
```

Example

In this example, the GlideAjax call is buried one level deeper by rearranging the script to check as many things available to the client as possible before running the server calls. The script checks the assignment before executing the GlideAjax call. This prevents the server lookup when the **assignment_group** field is already set.

```
//Set Assignment Group to CI's support group if assignment group is empty
function onChange(control, oldValue, newValue, isLoading,
isTemplate) {

    if (isLoading)
        return;

    if (newValue) {
        if (newValue != oldValue) {
            if (g_form.getValue('assignment_group') == '') {
                var ga = new GlideAjax('ciCheck');

                ga.addParam('sysparm_name', 'getSupportGroup');
                ga.addParam('sysparm_ci', g_form.getValue(
'cmdb_ci'));
                ga.getXML(setAssignmentGroup);
            }
        }
    }
}

function setAssignmentGroup(response) {
```

```
    var answer = response.responseXML.documentElement.getAttribute("answer");

    g_form.setValue('assignment_group', answer);
}
```

Client scripts

Client scripts allow the system to run JavaScript on the client (web browser) when client-based events occur, such as when a form loads, after form submission, or when a field changes value.

Use client scripts to configure forms, form fields, and field values while the user is using the form. Client scripts can:

- make fields hidden or visible
- make fields read only or writable
- make fields optional or mandatory based on the user's role
- set the value in one field based on the value in other fields
- modify the options in a choice list based on a user's role
- display messages based on a value in a field

Where client scripts run

With the exception of onCellEdit() client scripts, client scripts only apply to forms and search pages. If you create a client script to control field values on a form, you must use one of these other methods to control field values when on a list.

- Create an access control to restrict who can edit field values.
- Create a business rule to validate content.
- Create a data policy to validate content.
- Create an onCellEdit() client script to validate content.
- Disable list editing for the table.

Note: Client scripts are not supported on ServiceNow mobile applications.

Client script form

Field	Description
Name	Name of the client script.
Table	Table to which the client script applies.
UI Type	Target user interface to which the client script applies.
Type	<p>onLoad() — runs when the system first renders the form and before users can enter data. Typically, onLoad() client scripts perform client-side-manipulation of the current form or set default record values.</p> <p>onSubmit() — runs when a form is submitted. Typically, onSubmit() scripts validate things on the form and ensure that the submission makes sense. An onSubmit() client script can cancel form submission by returning a value of false.</p> <p>onChange() — runs when a particular field value changes on the form. The onChange() client script must specify these parameters.</p> <ul style="list-style-type: none">control: the DHTML widget whose value changed. Note: control is not accessible in mobile and service portal.oldValue: the value the widget had when the record was loaded.newValue: the value the widget has after the change.

Field	Description
	<ul style="list-style-type: none">• isLoading: identifies whether the change occurs as part of a form load.• isTemplate: identifies whether the change occurs as part of a template load. <p>onCellEdit() — runs when the list editor changes a cell value. The onCellEdit() client script must specify these parameters.</p> <ul style="list-style-type: none">• sysIDs: an array of the sys_ids for all items being edited.• table: the table of the items being edited.• oldValues: the old values of the cells being edited.• newValue: the new value for the cells being edited.• callback: a callback that continues the execution of any other related cell edit scripts. If true is passed as a parameter, the other scripts are executed or the change is committed if there are no more scripts. If false is passed as a parameter, any further scripts are not executed and the change is not committed.
Field Name	Name of the field to which the script applies. Available only if the script responds to a field value change (onChange or onCellEdit script types).
Application	Application where this client script resides.
Active	Enables the client script when selected. Unselect this field to disable the client script.
Inherited	Indicates whether the client script applies to extended tables.
Global	If true, the client script runs on all views of the table.

Field	Description
View	Only visible when Global is unselected. Views on which the client script will run.
Description	Content describing the functionality and purpose of the client script.
Messages	Text string (one per line) available to the client script as localized messages using <code>getmessage('[message]')</code> . For additional information, see Translate a client script message .
Script	Contains the client script.
Isolate script	New client scripts are run in strict mode, with direct DOM access disabled. Access to jQuery, prototype, and the window object are also disabled. To disable this on a per-script basis, configure this form and select the Isolate script check box. To disable this feature for all new globally-scoped client-side scripts set the system property <code>glide.script.block.client.globals</code> to false.

UI scripts

UI scripts provide a way to package client-side JavaScript into a reusable form, similar to how script includes store server-side JavaScript. Administrators can create UI scripts and run them from client scripts and other client-side script objects and from HTML code.

UI scripts are not supported for mobile.

Global UI scripts

You can create a UI script and designate it as global, which makes the script available on any form in the system. You cannot create a global UI script in a scoped application.

You can mark a UI script as Global to make it available on any form in the system. For example, you can create a UI script that has a function `helloWorld()`, and has the **Global** field checked:

```
function helloWorld() {  
    alert('Hi');  
}
```

After you create this global UI script, you can call the `helloWorld()` function from any client script or UI policy you write.

Create a UI script

Create a UI script to define reusable client-side JavaScript code.

To create UI scripts, navigate to **System UI > UI Scripts** and create or edit a record (see table for field descriptions).

UI scripts

Field	Description
Script Name	Name of the UI script. Ensure the name is unique on your system.
API Name	The API name of the UI script, including the scope and script name (for example, <code>x_custom_app.HelloWorld</code>).
Application	Application that contains the UI script.
Active	Indicator of whether the UI script is active. Only active UI scripts can run.
Global	Indicator of whether the script loads on every page in the system.

Field	Description
	Note: Use caution when creating global UI scripts because they can impact performance. You cannot create a global UI script in a scoped application.
Description	Summary of the purpose of the script.
Script	Client-side script to run when called from other scripts.

Run UI scripts

Follow these guidelines when running UI scripts.

Run a UI script from a form

To run a UI script on a form, [Create a formatter and add it to a form](#). In the associated [UI macro](#), include a `g:requires` tag and specify the `name=` parameter as the name of the UI script followed by the `.jsdbx` extension. Add the formatter on the form view.

This code ensures that the definitions and results of the UI script are immediately available in the browser.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide
" xmlns:j2="null" xmlns:g2="null">
    <g2:evaluate var="jvar_stamp">
        var now_GR = new GlideRecord('sys_ui_script');
        gr.orderByDesc('sys_updated_on');
        gr.query();
        gr.next();
        gr.getValue('sys_updated_on');
    </g2:evaluate>
    <g:requires name=".jsdbx" params="cach
e=$[jvar_stamp]" />
</j:jelly>
```

Call a UI script in HTML

To run a UI script from HTML code, use the `<script>` tag and specify the `src=` argument as the API name of the UI script followed by the `.jsdbx` extension. For example, include the UI script named CoolClock with this code:

```
<script language="javascript" src="CoolClock.jsdbx" />
```

Call a UI script from client-side code

Access UI scripts from within client-side code using the `g_ui_scripts` global object. For more information, see [GlideUIScripts - Client](#).

Note: This class does not support UI scripts with the **Global** field set to true.

Catalog client scripts

Client-side scripts can add dynamic effects and validation to forms. Scripts can apply to service catalog items or variable sets, allowing administrators to use the same functionality that is available on other forms.

You can use client side scripts to:

- Get or set variable values.
- Hide or display variables.
- Make variables mandatory or not.
- Validate form submission.
- Add something to the cart.
- Order something immediately.

Catalog client script considerations

When you create catalog client scripts, be aware of the following considerations.

- Catalog client scripts run when a user orders an item from the service catalog. Catalog client scripts can also run when variables or variable sets for a catalog item are displayed when a user requests that item.
- For a variable to be accessible using a catalog client script, it must have a variable name. Variables without names do not appear in the list of available variables.
- When using standard client scripts on a Requested Item or Catalog Task form, make a note of fields with the same name as variables. If a table field and a variable of the same name are both present on a form, the table field is matched when it is accessed using a script. If this happens, specifically address the variable by naming it `variables.variable` name. For example: `g_form.setValue('variables.replacement', 'false');`
- If you are using record producers to pass variables from the service catalog to other types of records, these variables are made visible in those records with a variable editor, such as the Change Variable Editor UI formatter on Change request forms. You can manipulate these variables using standard client script methods, such as `setDisplay`, `setMandatory`, `setValue`, and `getValue`.
- Catalog client scripts can be used for catalog items included in a wizard.
- You can use the `g_form.refreshSlushbucket(fieldName)` API to update a list collector variable.

Catalog client script differences

Catalog client scripts are very similar to standard client scripts, with a few important differences.

- Instead of selecting a table such as Incident for the script, select a catalog item or variable set. As your system may have a large number of catalog items, you should select a catalog item or variable set using a reference field instead of the choice list that the standard Client Script form uses.
- When using an `onChange()` catalog client script, it is linked to a particular variable instead of a field. The system automatically populates the variable name selection list with any named variables from the catalog item or variable set selected.

Create a catalog client script

Follow this procedure to create a catalog client script.

1. Navigate to **All > Service Catalog > Catalog Administration > Catalog Client Scripts**. A list of current custom catalog client scripts appears.
2. Click **New**.
3. Fill in the fields, as appropriate (see table).

Field	Description
Name	Enter a unique name for the catalog client script.
Applies to	Select the item type this client script applies to: <ul style="list-style-type: none">• A Catalog Item: enables the Catalog item field.• A Variable Set: enables the Variable set field.
Active	Select the check box to enable the client script. Clear the check box to disable the script.
UI Type	Whether to apply this to desktop, mobile, or both.
Script	Enter the client script that should run on the service catalog item.
Type	Select when the script should run, such as onLoad or onSubmit .
Catalog item or Variable set	Select a catalog item or variable set from the list. The field name and options available depend on the selection in the Applies to field.

Field	Description
Applies on a Catalog Item view	Select the check box to apply the catalog client script to catalog items displayed within the order screen on the service catalog. Available in the requester view.
Applies on Requested Items	Select the check box to apply the catalog client script on a Requested Item form, after the item is requested. Available in the fulfiller view. See VEditor .
Applies on Catalog Tasks	Select the check box to apply the catalog client script when a Catalog Task form for the item is being displayed. Available in the fulfiller view. See VEditor .
Applies on the Target Record	Select the check box to support the catalog UI policy on a record created for task-extended tables via record producers. See Default variable editor .

4. Click **Submit**.

Catalog client script examples

Examples of client scripts to perform common actions.

Example: Get the value of a variable

Use the following syntax to obtain the value of a catalog variable. Note that the variable must have a name. Replace `variable_name` with the name of the variable.

```
g_form.getValue('variable_name');
```

Example: Restrict the number of characters a user can type in a variable

This is an example of a script that runs when the variable is displayed, rather than when the item is ordered.

```
function onLoad() {  
    var sd = g_form.getControl('short_description');  
    sd.maxLength=80;  
}
```

Mobile client GlideForm (g_form) scripting and migration

Client scripting for mobile is identical to scripting for the web, with some exceptions. All new scripts must conform to certain guidelines. The following items are affected on the mobile platform: client scripts, UI policies, navigator modules, and UI actions.

Client scripts

For new or existing scripts to be valid for mobile, they must conform to the following requirements:

- Use the new mobile methods in place of `g_form.getControl()`.
- Do not use deprecated methods.
- Do not reference unsupported browser objects.
- Do not make synchronous JavaScript, GlideAjax, and GlideRecord calls.
- Do not call methods that are not available for mobile.
- Enable scripts to run on the mobile UI.

Requirements

Use the new mobile methods

Several new methods are available for modifying form fields instead of directly manipulating the HTML. These methods

replace previous usages of `g_form.getControl()`, which is deprecated for the mobile platform. In your existing scripts, ensure that the new methods are used in place of methods that are not valid on the mobile platform. For information on these new methods, refer to [Mobile GlideForm\(\) API](#).

The following methods have been deprecated for the mobile platform because direct access to HTML elements is not allowed:

- `g_form.getControl()`
- `g_form.getFormElement()`
- `g_form.getElement()`

Do not use deprecated methods

To ensure that existing scripts are compatible, remove all calls to deprecated methods from your code. For new scripts, do not use deprecated methods if you want the script to be valid for mobile.

For `g_form.getControl()`, some of the functionality previously included with this method has been extracted to individual methods. Instead of `g_form.getControl()`, use the new methods described on the developer site.

Do not reference unsupported browser objects

The following browser objects are not supported in mobile scripts:

- Window
- jQuery or Prototype (\$, \$j, or \$\$)
- Document

Make sure that new scripts do not use these objects, and remove any usage of these objects from your existing scripts. Use GlideForm (g_form) instead, which provides methods such as setLabel(), addDecoration(), and hasField() for accomplishing the same tasks.

The mobile platform does not allow synchronous JavaScript calls. The g_form.getReference() method must now have the callback parameter defined. For example:

```
g_form.getReference(fieldName, callback)
```

Do not make synchronous
JavaScript calls

Be sure that all g_form.getReference() calls include the callback parameter. For example, the following script does not include a callback and is incompatible with the mobile platform:

```
var userName = g_form.getReference('assigned_to').user_name;
g_form.setValue('u_assigned_user_name', userName);
```

The following script has been updated to include the callback

and is compatible with the mobile platform:

```
g_form.getReference('assigned_to', function(now_GR) {  
    g_form.setValue('u_assigned_user_name', gr.user_name);  
});
```

Do not make synchronous Ajax calls

The mobile platform does not allow synchronous GlideAjax calls. Any use of `getXMLWait()` in a GlideAjax call will not work on the mobile platform. Be sure that all GlideAjax calls are asynchronous. For more on synchronous versus asynchronous GlideAjax calls and `getXMLWait()`, see [AJAX](#). For information on the available GlideAjax methods, refer to the [GlideAjax API](#).

Do not make synchronous GlideRecord calls

The mobile platform does not allow synchronous [GlideRecord](#) calls. Make sure that any existing GlideRecord calls include a callback. For example, the following script does not include a callback and is incompatible with the mobile platform:

```
var now_GR = new GlideRecord('incident');  
gr.addQuery('number', g_form.getValue('related_incident'));  
gr.query();  
gr.next();  
g_form.setValue('u_related
```

```
_incident_description', gr.  
short_description);
```

The following script has been updated to include the callback, and is compatible with the mobile platform:

```
var now_GR = new GlideRecord('incident');  
gr.addQuery('number', g_form.getValue('related_incident'));  
gr.query(function(now_GR)  
{  
    gr.next();  
    g_form.setValue('u_related_incident_description'  
, gr.short_description);  
});
```

Due to the limitations and reduced functionality that is imposed by the mobile platform, the following methods are not deprecated but are not available on the mobile platform. If these run on the mobile platform, no action occurs:

- showRelatedList ()
- hideRelatedList ()
- showRelatedLists ()
- hideRelatedLists()
- flash()
- getSections()
- enableAttachments()
- disableAttachments()

Do not use methods unavailable on the mobile platform

-
- `setReadonly()` (Note that `setReadOnly()` is available)
 - `getParameter()`
-

Enable scripts for mobile

Scripts must be enabled for the mobile platform. See [Enable client scripts for the mobile browser for the ServiceNow Classic mobile app.](#)

Note: Focusing an element on a mobile form is not supported.

UI policies

Use the **Run scripts in UI type** field to determine whether scripts run on the mobile platform, the desktop, or both. Update existing policies so that they apply to either the mobile platform or both. For new scripts, also ensure that the mobile option or both is selected. For more on UI policies for mobile, see [Enable UI policies for the mobile browser](#).

Navigator modules

For existing code, modules must be transferred to either the `sys_ui_application` or `sys_ui_module` tables to be available on the mobile platform. When developing new code, be sure that all modules are created in the `sys_ui_application` or `sys_ui_module` tables. For more information, see [Enable an application menu for the ServiceNow Classic mobile app](#).

UI actions

UI actions must be transferred to the `sys_ui_ng_action` table to appear on the mobile platform. UI action scripts that do not use deprecated methods do not require changes to the script itself. For new UI actions, be sure that they are created in the `sys_ui_ng_action` table. For more information, see [Mobile UI actions for the ServiceNow Classic mobile app](#).

AJAX

AJAX (asynchronous JavaScript and XML) is a group of interrelated, client-side development techniques used to create asynchronous Web applications.

AJAX enables web applications to send and retrieve information to and from a server in the background, without impacting the user experience with the displayed web page.

GlideAjax

The GlideAjax class allows the execution of server-side code from the client. GlideAjax calls pass parameters to the script includes, and, using naming conventions, allows the use of these parameters.

Note: This functionality requires a knowledge of JavaScript.

Using GlideAjax:

- Initialize GlideAjax with the name of the script include that you want to use.
- When creating the script include, you must set the name field to be exactly the same as the class name.
- When creating the script include, you must select the **Client callable** check box.
- Specify the parameter sysparm_name. GlideAjax uses sysparm_name to find which function to use.
- Any extra parameters may be passed in, all of which must begin with sysparm_. Avoid using predefined parameter names:
 - sysparm_name
 - sysparm_function
 - sysparm_value
 - sysparm_type
- Code is then executed with the getXML() or getXMLWait() functions.

For additional information, refer to the [GlideAjax API](#).

Examples of asynchronous GlideAjax

There are two parts to the asynchronous GlideAjax script: client-side and server-side code.

Hello World: Returning a value from the server

Example: Client side

This code runs on the client (the web browser). Create a client script as normal. This sends the parameters to server, which then does the processing. So that the client does not wait for the result, a callback function is used to return the result, passed to the getXML() function. (In this case it is called `HelloWorldParse`.)

The `getXMLWait()` function does not need a separate callback function, but this will block the client. If the client-server communication takes a long time (for example on slow networks), the application will seem unresponsive and slow. An example of `getXMLWait()` is in the following section.

```
var ga = new GlideAjax('HelloWorld');
ga.addParam('sysparm_name', 'helloWorld');
ga.addParam('sysparm_user_name', "Bob");
ga.getXML(HelloWorldParse);

function HelloWorldParse(response) {
    var answer = response.responseXML.documentElement.getAttribute("answer");
    alert(answer);
}
```

Example: Server side

The server-side code for the above function. Do not create a business rule, but instead navigate to **System Definition > Script Include** and create a new script. Paste in the code below.

Note: You must set the name of the script include to `HelloWorld`.

- The `sys_script_include` code must extend the `AbstractAjaxProcessor` class and be client-callable.

- Function names starting with "_" are considered private and are not callable from the client.
- Avoid overriding methods of AbstractAjaxProcessor, including `initialize`. While it is possible to invoke methods of your superclass object which you have overridden, it is complicated and best avoided altogether.

```
var HelloWorld = Class.create();
HelloWorld.prototype = Object.extendObject(AbstractAjaxProcessor, {
    helloWorld:function() { return "Hello " + this.getParameter('sysparm_user_name') + "!"; },
    _privateFunction: function() { // this function is not
        client callable
    }
});
```

This results in an alert box that says 'Hello Bob!' when you visit the form.

Returning multiple values

Since the response is an XML document we are not limited to returning a single answer value. Here is a more complex example returning multiple XML nodes and attributes.

Example: AJAX processor script include

```
/*
 * MyFavoritesAjax script include Description - sample AJAX processor returning multiple value pairs
 */
var MyFavoritesAjax = Class.create();
MyFavoritesAjax.prototype = Object.extendObject(AbstractAjaxProcessor, {

    /*
     * method available to client scripts call using:
     * var gajax = new GlideAjax("MyFavoritesAjax");
     * gajax.addParam("sysparm_name","getFavorites");
     */
    getFavorites: function() { // build new response XML element for result
        var result = this.newItem("result");
```

```
        result.setAttribute("message", "returning all favorites");

        //add some favorite nodes with name and value attributes
        this._addFavorite("color", "blue");
        this._addFavorite("beer", "lager");
        this._addFavorite("pet", "dog");
    },
    // all items are returned to the client through the inherited methods of AbstractAjaxProcessor
    _addFavorite: function(name, value) {
        var favs = this newItem("favorite");
        favs.setAttribute("name", name);
        favs.setAttribute("value", value); },
    type: "MyFavoritesAjax"
};

}
```

Example: Client script

```
// new GlideAjax object referencing name of AJAX script include
var ga = new GlideAjax("MyFavoritesAjax");
// add name parameter to define which function we want to call
// method name in script include will be getFavorites
ga.addParam("sysparm_name", "getFavorites");

// submit request to server, call ajaxResponse function with server response

ga.getXML.ajaxResponse();

function ajaxResponse(serverResponse) {
    // get result element and attributes
    var result = serverResponse.responseXML.getElementsByTagName("result");
    var message = result[0].getAttribute("message");

    //check for message attribute and alert user
    if(message) alert(message);
```

```
//build output to display on client for testing
var output = "";

// get favorite elements
var favorites = serverResponse.responseXML.getElementsBy
TagName("favorite");
for(var i = 0; i < favorites.length; i++) {
    var name = favorites[i].getAttribute("name");
    var value = favorites[i].getAttribute("value");
    output += name + " = " + value + "\n ";
}

alert(output); }
```

Example: XML response

```
<xml sysparm_max= "15" sysparm_name="getFavorites" sysparm
_processor="MyFavoritesAjax">
<result message = "returning all favorites"></result>
<favorite name = "color" value = "blue"></favorite>
<favorite name = "beer" value = "lager"></favorite>
<favorite name = "pet" value = "dog"></favorite>
</xml>
```

Examples of synchronous GlideAjax

Use synchronous when your script cannot continue without the GlideAjax response. This stops the session until the response is received.

If your use case demands that no further processing can occur until the GlideAjax response has been received, you can use `getXMLWait()`. However, because this will slow down your code and lock the user session until the response is received, it is generally recommended that you use `getXML()` with a callback function.

Note: Do not use `AJAXEvaluateSynchronously`.

Note: The `getXMLWait()` method is not available in scoped applications.

This code results in a client-side alert that displays The Server Says Hello Bob!.

The client code.

```
var ga = new GlideAjax('HelloWorld') ;
ga.addParam('sysparm_name','helloWorld');
ga.addParam('sysparm_user_name','Bob');
ga.getXMLWait();
alert(ga.getAnswer());
```

The server-side script include code.

```
var HelloWorld = Class.create();
HelloWorld.prototype = Object.extendsObject(AbstractAjaxProcessor, {
    helloWorld: function()
    {
        return "The Server Says Hello " + this.getParameter('sysparm_user_name') + "!";
    }
});
```

AJAXClientHelper

Provides helper functions for Ajax clients to generate Choice Lists and retrieve displayed values from a choice list.

Where to use

Use this script include wherever your need to generate choice lists or retrieve a value from an Ajax client.

Method summary

Method summary	Description
getValues()	Gets the values from the choice list.
generateChoice()	Generates the choices for a choice list.
generateChoiceTable()	Generates the choice table.
getDisplay()	Gets the display value from the choice list.

Method detail

API method	Description	Input parameters	Output returns
getValues()	Gets the values from the choice list.	none	The choice list values.
generateChoice()	Generates the choices for a choice list.	none	The choice list.
generateChoiceTable()	Generates the choice table.	none	The choice table.
getDisplay()	Gets the display value from the choice list.	none	The display value.

AJAXClientTiming

Saves client timing values for a transaction.

Method summary

Method	Description
process()	Method called by the Prototype JavaScript Framework during object processing.

Method detail

Method Detail	Description	Input Fields	Output Fields
process()	Called by the Prototype JavaScript Framework during object processing. Do	none	Returns: void.

Method Detail	Description	Input Fields	Output Fields
	not call this method directly.		

Useful scripts

Scripts that provide useful functionality not included in the core system.

- [Get a user object](#)

In a business rule or other server script, the `gs.getUser()` method returns a user object. The user object is an internal representation of the currently logged in user and provides information about the user and various utility functions.

- [Accessing the workflow scratchpad from business rules](#)

A catalog item has been requested, the attached workflow contains a run script activity that populates a value in the scratchpad. From a business rule running on the requested item, we want to retrieve or set scratchpad values.

- [Add a field to the service catalog checkout](#)

This is an example of adding a **Company** field to the checkout below the **Requested for** field using non-cart layout macros, that is, `glide.sc.use_cart_layouts` is `false`.

- [Assign a catalog item to a group based on a delivery plan task](#)

This assignment rule assigns a service catalog item to the database group if it uses a delivery plan that has a catalog task assigned to the desktop group.

- [Change form color on state change](#)

Changes color of a form field of the form on state change. The script can easily be changed to adjust any property of any object on the page accessible via the HTML DOM.

- [Create a UI routing action](#)

This solution enables you to create a record with the service desk without knowing whether it is an incident or request item; the service desk can then route the record to the appropriate table.

- [Custom approval UI macro](#)

This section describes how to create a custom approval UI macro.

- [Display field messages](#)

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.

- [Log output](#)

GSLog is a script include that simplifies script logging and debugging by implementing levels of log output, selectable by per-caller identified sys_properties values.

- [Modify a GlideDateTime field value](#)

This example demonstrates how to modify a GlideDateTime field value using a script.

- [Sample ASP.NET with C Sharp redirect with cookies](#)

This sample ASP .NET code creates a simple authentication portal and passes an unencrypted HTTP header as a cookie.

- [Useful approval assignment scripts](#)

This is a searchable version of the useful approval and assignment scripts.

- [Useful field scripts](#)

Common use cases for field customization scripts.

- [Useful scheduling scripts](#)

A business rule script specifies the actions that the business rule takes. Scripts commonly include predefined global variables to reference

items in your system, such as the current record. Global variables are available to all business rules.

- [Using client and server code in a UI action](#)

You can use a script to validate input upon a UI Action click on the client side before updating the record on the server side. The user will not have to click the button twice to validate the required fields and update the record.

- [Sample ASP Script for unencrypted single sign-on](#)

This sample ASP .NET code creates a simple authentication portal and passes an unencrypted HTTP header as a URL parameter.

- [Validate date and time](#)

To validate the input of all date/time fields, you can use the following in a validation script (**System Definition > Validation Scripts**).

- [Calculating durations](#)

Often you may need to provide users with a way to specify when a task or process is due. Using the DurationCalculator script include, you can calculate the due date using either a simple duration or relative duration.

- [Simple duration vs relative duration](#)

How much work is required to complete a task can be expressed as a "relative duration".

- [How to implement a relative duration](#)

You can implement a relative duration by creating the cmn_relative_duration table and the DurationCalculator script include.

Get a user object

In a business rule or other server script, the `gs.getUser()` method returns a user object. The user object is an internal representation of the currently logged in user and provides information about the user and various utility functions.

About this task

For a list and description of the available scoped methods for the user object, see [GlideUser](#).

Procedure

1. Retrieve the current user.

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

2. Use the 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()); //print the first name
e of the user you are currently logged in as
newUser = ourUser.getUserByID(<user_sys_id>); //fetch
a different user, using the sys_id of the target user r
ecord.
gs.print(newUser.getFirstName()); //first name of the u
ser you fetched above
gs.print(newUser.isMemberOf('Capacity Mgmt'));
```

Accessing the workflow scratchpad from business rules

A catalog item has been requested, the attached workflow contains a run script activity that populates a value in the scratchpad. From a business rule running on the requested item, we want to retrieve or set scratchpad values.

Prerequisites

Role required: admin

Name: Access Workflow Scratchpad from Business Rules

Type: Business Rule

Table: sc_req_item (Requested Item)

Description: A catalog item has been requested, the attached workflow contains a run script activity that populates a value in the scratchpad.

From a business rule running on the requested item, we want to retrieve or set scratchpad values.

Parameters: n/a

Script:

```
//the run script activity sets a value in the scratchpad
workflow.scratchpad.important_msg = "scratch me";

//get the workflow script include helper
var workflow = new Workflow();

//get the requested items workflow context
//this will get all contexts so you'll need to get the proper one if you have multiple workflows for a record
var context = workflow.getContexts(current);
//make sure we have a valid context
if (context.next()) {
    //get a value from the scratchpad
    var msg = context.scratchpad.important_msg;
    //msg now equals "scratch me", that was set in the run script activity

    //add or modify a scratchpad value
    context.scratchpad.status = "completed";
    //we need to save the context record to save the scratchpad
    context.update();
}
```

Add a field to the service catalog checkout

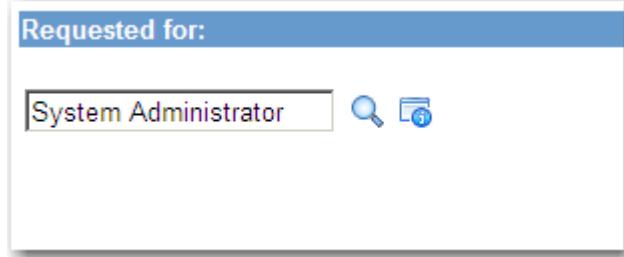
This is an example of adding a **Company** field to the checkout below the **Requested for** field using non-cart layout macros, that is, `glide.sc.use_cart_layouts` is false.

Before you begin

Role required: Admin

About this task

Requested for field



This field will then pass the value of that field to the **Company** field of the Service Catalog Request.

This example makes the following assumptions.

- This example is for an instance using two-step checkout. If two-step checkout is not enabled, enable it before beginning.
- This example populates the **Company** field on the Service Catalog Request form. If the field does not appear on the form, configure the form before beginning.

Procedure

1. Go to **System UI > UI Macros** and select **servicecatalog_cart_template**.
2. Find where there are hidden variables declared and add the following line:

```
<input type="HIDDEN" name="cart_id" id="cart_id" value=" ${sc_cart.sys_id} " />
```

3. Find the following code, which generates the **Requested For** code:

```
<tr class="header">
<td width = "30%">
    ${gs.getMessage('Requested for')}:</td>
<td width="70%">
    <label for="requestor_location">${gs.getMessage('De
liver to')}</label>
```

```
</td>
</tr>
<tr><td>$[SP]</td>
</tr>
<tr><td valign="top">
    <j2:if test="$[jvar_can_delta_rf == false]">
        $[sc_cart.requested_for.getDisplayValue()]
    </j2:if>
    <j2:if test="$[jvar_can_delta_rf != false]">
        <g2:catalog_requested_for />
    </j2:if>
</td>
<td>
    <textarea id="requestor_location" style="width:100%
" rows="4"
        name="requestor_location" wrap="soft" onChange="c
atDeliveryAddress('$[sc_cart.sys_id]', 'requestor_locat
ion');">
        $[sc_cart.delivery_address]
    </textarea>
</td>
</tr>
<tr>
    <td>$[SP]</td>
</tr>
```

4. Add the following code afterwards:

```
<tr class="header">
    <td colspan="2">Company</td>
</tr>
<tr>
    <td>$[SP]</td>
</tr>
<tr>
    <td colspan="2">
        <g2:ui_reference name="core_company" table="core_co
mpany" onchange="setCartValue()"/>
    </td>
</tr>
<tr>
    <td>$[SP]</td>
</tr>
```

Note: The 'ui_reference' macro defines a reference field. There are several macros for different field types. You can see examples of these field types under **System UI -> UI Macros**. These macros start with 'ui_'. For this example, the reference field created is named **core_company**.

5. Now navigate to **System UI > UI Pages** and select the **servicecatalog_checkout_one** UI Page. Add this script to the **Client script** field:

```
function setCartValue() {
    var newField = gel('core_company');
    var myCart = gel('cart_id');
    var cart_item = new GlideRecord('sc_cart_item');
    cart_item.addQuery('cart', myCart.value);
    cart_item.query();
    if(cart_item.next()) {
        cart_item.hints = "<hints><entry key='sysparm_proce
ssing_hint' value='setfield:request.company=" + newFiel
d.value + "'/></hints>";
        cart_item.update();
    }
}
```

For this example, the reference field was called **core_company**, and the field being populated on the request is **company**. If different fields are used:

- Find this line: `var company = gel('core_company');` and replace **core_company** with the name of the field in the checkout.
- In the line that starts with '`cart_item.hints`' replace '`'request.company'`' with the name of the field to be populated on the request ticket where '`request`' is the request being generated and '`company`' is the name of the field.

Result

When an item is ordered, the company field appears on the Catalog form:

The screenshot shows the ServiceNow Shopping Cart interface. At the top, there's a header bar with a back arrow icon and the title "Shopping Cart". Below the header is a table with the following data:

Item	Delivery Time	Price (ea.)	Quantity	Total
iPhone 3g - The iPhone 3g...more than just a phone		\$400.00	1	\$400.00

Below the table, it says "Total: \$400.00".

There are two sections for addresses:

- Requested for:** System Administrator
- Deliver to:** 750 3rd Ave
New York, NY, 10017

A "Company" search field is present below the addresses.

In the "Special instructions" section, there's a link "Add attachment..." with a file icon.

At the bottom, there are two buttons: "Back to Catalog" and "Submit Order".

Assign a catalog item to a group based on a delivery plan task

This assignment rule assigns a service catalog item to the database group if it uses a delivery plan that has a catalog task assigned to the desktop group.

Prerequisites

Role required: admin

Warning: The customization described here was developed for use in specific instances, and is not supported by Now Support. This method is provided as-is and should be tested thoroughly before implementation. Post all questions and comments regarding this customization to our community [forum](#).

Name: Assign Catalog Item to Group Based on Delivery Plan Task

Type: Assignment Rule

Table:

Description: This assignment rule assigns a service catalog item to the database group if it uses a delivery plan that has a catalog task assigned to the desktop group.

Parameters:

Script:

```
//Return catalog items that have no group but do have a delivery plan assigned
var ri = new GlideRecord ("sc_cat_item");
ri.addQuery("group", "=", null);
ri.addQuery("delivery_plan", "!=" , null);
ri.query();
while(ri.next()) {
    gs.log("Found an item");
    //Return tasks that point to the same delivery plan as the above item
    var dptask = new GlideRecord("sc_cat_item_delivery_task");
    dptask.addQuery("delivery_plan", "=", ri.delivery_plan);
    dptask.query();
    while(dptask.next()) {
        gs.log("Found a task");
        var gp = dptask.group.getDisplayValue();
        gs.log(gp);
        //If the task is assigned to desktop, assign the item's group to desktop
        if (dptask.group.getDisplayValue() == "Desktop") {
            ri.group.setDisplayValue("Desktop");
            gs.log("updating " + ri.getDisplayValue());
            ri.update();
            break; } } }
```

Change form color on state change

Changes color of a form field of the form on state change. The script can easily be changed to adjust any property of any object on the page accessible via the HTML DOM.

Warning: The customization described here was developed for use in specific instances, and is not supported by Now Support. This method is provided as-is and should be tested thoroughly before implementation. Post all questions and comments regarding this customization to our community [forum](#).

Name: Change Form Color on State Change

Type: Client Script

Table:

Description: Changes color of a form field of the form on state change. The script can easily be changed to adjust any property of any object on the page accessible via the HTML DOM.

Parameters:

Script:

```
function onChange(control, oldValue, newValue, isLoading)
{
    var elementID = get("incident.priority");
    switch(newValue) {
        case "1": elementID.style.backgroundColor = "red"; break;
        case "2": elementID.style.backgroundColor = "tomato"; break;
        case "3": elementID.style.backgroundColor = "orange"; break;
        case "4": elementID.style.backgroundColor = "yellow"; break;
        case "5": elementID.style.backgroundColor = "green"; break;
        default: elementID.style.backgroundColor = "white";
    }
}
```

Create a UI routing action

This solution enables you to create a record with the service desk without knowing whether it is an incident or request item; the service desk can then route the record to the appropriate table.

About this task

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

To create a UI routing action:

Procedure

1. Create a new table that extends the `task` table (for example, New Call).
2. Create a module to create a new New Call record.
3. Create any fields that you want on the New Call table.

The only fields you need are those fields necessary to determine whether the new call should route to an Incident or a Request Item. Ensure that the form contains any fields that you want to pass to the Incident or Request Item. In this example, the following are created on the form:

- **Requested for** (reference)
- **Location** (reference)
- **Call type** (choice with two values--Incident and Request)
- **Request Item** (reference to the `sc_cat_item` Item table)

4. Add some UI policies to set a couple of fields to mandatory and hide the **Request item** field based on the **Call type** selection.
5. Remove unnecessary buttons and functionality from the form.
6. Create a new UI Action button. This button redirects the user to either an incident or a request. It also creates the incident record and copies values to the incident and the Request Item form.

```
var reqItem = current.u_item;
var requestedFor = current.u_requested_for;
var location = current.location;

if(current.u_incident_request == 'Incident'){
    //Create a new incident record and redirect to the new incident
    var rec = new GlideRecord('incident');
    rec.initialize();
    rec.caller_id = requestedFor;
    rec.location = location;
    rec.insert();
    action.setRedirectURL(rec);
}

if(current.u_incident_request == 'Request'){
    //Build the url and route the user to the request item
    var url = '';
    if(current.u_item.sys_class_name == 'sc_cat_item_guide'){
        url = 'com.glideapp.servicecatalog_cat_item_guide_view.do?sysparm_initial=true&sysparm_guide=' +
              reqItem + '&sysparm_user=' + requestedFor + '&sysparm_location=' + location;
    }
    else{
        url = 'com.glideapp.servicecatalog_cat_item_view.do?sysparm_id=' + reqItem + '&sysparm_user=' +
              requestedFor + '&sysparm_location=' + location;
    }
    action.setRedirectURL(url);
}
```

7. The **Route** button in the preceding example passes the **Requested for** and **Location** values in the URL to the Request Item form. Ensure that you have variables called `requested_for` and `location` on your item, record producer, or order guide that map these values using the following client script. There is a limit as to how much information you can pass, as the URL has a restricted length. Avoid sending information from long text fields using this method.

```
function onLoad() {
    var url = document.location.toString();
```

```
var userKey = 'sysparm_user=';
var locKey = 'sysparm_location=';
var userPosition = url.indexOf(userKey);
var locPosition = url.indexOf(locKey)
if (userPosition != -1) {
    var user = url.substr(userPosition+userKey.length,
32);
    g_form.setValue('requested_for',user);
}
if (locPosition != -1) {
    var loc = url.substr(locPosition+locKey.length, 32)
;
    g_form.setValue('location',loc);
}
}
```

Custom approval UI macro

This section describes how to create a custom approval UI macro.

Warning: The customization described here was developed for use in specific instances, and is not supported by Now Support. This method is provided as-is and should be tested thoroughly before implementation. Post all questions and comments regarding this customization to our community [forum](#).

Script Name: Custom Approval UI Macro

Type: UI Macro

Description: Here is an option to get more detail out of the My Approvals view of an Execution Plan. This can be done by creating a new UI macro. Navigate to **System UI** and click **UI Macros**. First, you will need to rename the existing `approval_summarizer_sc_task` to something like `approval_summarizer_sc_task_old` and deactivate it. Then you will need to create a new one using the same name (`approval_summarizer_sc_task`). The name should basically tell you what the macro does and what it applies to. In this case, we're replacing an existing one so we decided to re-use the existing name.

Approval macros

Macros	Name	Description	Active
approval_summarizer	approval_summarizer	true	true
approval_summarizer_change_request	approval_summarizer_change_request	true	true
approval_summarizer_default	approval_summarizer_default	true	true
approval_summarizer_ac_request	approval_summarizer_ac_request	true	true
approval_summarizer_sc_task	approval_summarizer_sc_task	true	true
approval_summarizer_sc_task_old	approval_summarizer_sc_task_old	false	true
approval_variable_summary	approval_variable_summary	true	true
cart_variable_summary	cart_variable_summary	true	true
catalog_cart_default	catalog_cart_default	true	true
catalog_model_info	catalog_model_info	true	true
catalog_user_info	catalog_user_info	true	true
context_form_header	Context menu for a form header	true	true
context_list_header	Context menu for a list header	true	true
decorate_list_actions	List actions are the column above the ch...	true	true
decorate_welcome_header_stripe	Provide content on the right side of the...	true	true
decorate_welcome_header_stripe_left	Provide content on the left side of the ...	true	true
diagrammer	Includes for the Visual Diagrammer	true	true
diagram_scale	Provides scale drop down select for diag...	true	true
dialog_button	Create a button that performs an arbitra...	true	true
dialog_buttons_ok_cancel	OK and Cancel buttons for a dialog Defa...	true	true

Then you should copy the xml script at the bottom of this article into the xml code window in the new UI macro. This is great way to give some detail to an approver when you are doing line item approvals via approval tasks within the Service Catalog Execution Plans.

The old way

This is the view you see in My Approvals when using an approval task via the old method.

My Approvals (old way)

Approver: Fred Lucy Approval for: TASK10041

State: Requested

Comments:

Summary of Task being approved:

Short Description:	Get the Department head approval for this test item
Assigned To:	
Priority:	4
Requested By:	Jamie Carvana
Requested For:	Christen Mitchell
Description:	

Buttons: Update, Approve, Reject, Delete

Notice there is not much detail telling the approver what they are actually approving. You can see the short description of the task but not much around what the item is.

The new way

This is the view you will see if you use the xml script below in place of the OOB (out-of-box) UI macro.

My Approvals

The screenshot shows a web-based application interface titled 'My Approvals'. At the top, there are search and filter fields for 'Approver' (set to 'Fred Luddy') and 'Approval for' (set to 'TASK10041'). Below these are buttons for 'Update', 'Approve', 'Reject', and 'Delete'. A large text area labeled 'Comments:' is empty. Underneath, a summary table provides details about the item being approved:

Summary of Item being approved:			
Opened by:	Jamie Carvina	Requested for:	Christen Mitchell
Total:	\$50.00	Price:	\$25.00
Number:	Description:	Quantity:	Total:
RITM10019	Test Item Short Description	2	\$50.00

At the bottom of the interface are three buttons: 'Update', 'Approve' (highlighted in yellow), 'Reject', and 'Delete'.

Using this method you can see details much like the request approval. You have a link into the item ordered, a short description (which contains the ability to expand the variables from the item), price, quantity and the total price. This helps the approver in that it shows more detail. They can now see what they are actually approving.

Script:

```
<?xml version="1.0" encoding="utf-8" ?>
    <j:jelly trim="true" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
        <tr>
            <td class="label_left" width="100%">
                <label style="margin-left: 10px">
                    ${gs.getMessage('Summary of Item being approved')}:
                    <input style="visibility: hidden" NAME="make_spacing_ok"></input>
                </label>
            </td>
        </tr>
        <g:evaluate var="jvar_ni" expression="
            var sc_task = ${ref}.sysapproval;
            var sc_req_labels = new GlideRecord('sc_req_item')
;
            sc_req_labels.initialize();
            var sc_req_item = new GlideRecord('sc_req_item');
            sc_req_item.addQuery('request_item', sc_task.reque
;
```

```
st_item.sys_id);
    sc_req_item.query();
" />
<tr>
    <td width="100%">
        <table width="100%">
            <tr>
                <td class="label_left" width="150px">
                    <label style="margin-left: 10px">
                        ${sc_task.request_item.request
.opened_by.sys_meta.label}:
                    </label>
                </td>
                <td>
                    ${sc_task.request_item.request.ope
ned_by.getDisplayValue()}
                </td>
            </tr>
            <tr>
                <td class="label_left" width="150px">
                    <label style="margin-left: 10px">
                        ${sc_task.request_item.request
.requested_for.sys_meta.label}:
                    </label>
                </td>
                <td>
                    ${sc_task.request_item.request.req
ested_for.getDisplayValue()}
                </td>
            </tr>
            <tr>
                <td class="label_left" width="150px">
                    <label style="margin-left: 10px">$
{gs.getMessage('Total')}:</label>
                </td>
                <td>
                    <g:currency_format value="${sc_ta
sk.request_item.request.price}" />
                </td>
            </tr>
        </table>
    </td>
</tr>
<j:set var="jvar_line_num" value="0" />
```

```
<tr>
    <td width="100%">
        <table width="100%">
            <tr class="header">
                <td colspan="2">
                    ${sc_req_labels.number.sys_meta.label}
                </td>
                <td>
                    ${sc_req_labels.description.sys_meta.label}
                </td>
                <td>
                    ${sc_req_labels.price.sys_meta.label}
                </td>
                <td>
                    ${sc_req_labels.quantity.sys_meta.label}
                </td>
                <td>
                    ${gs.getMessage('Total')}
                </td>
            </tr>
            <j:set var="jvar_item_price" value="${sc_task.request_item.price * sc_task.request_item.quantity}" />
            <j:set var="jvar_overall_total" value="${jvar_overall_total + jvar_item_price}" />
            <j:set var="jvar_line_color" value="odd" />
            <j:set var="jvar_line_num" value="${jvar_line_num + 1}" />
            <j:if test="${jvar_line_num % 2 == 0}">
                <j:set var="jvar_line_color" value="even" />
            </j:if>
            <g:evaluate var="ni" expression="
                var smart_description = sc_task.request_item.cat_item.short_description;
                if (smart_description == null || smart_description == '' || smart_description == 'undefined')
                    smart_description = sc_task.request_item.cat_item.name;
            "/>
            <tr class="${jvar_line_color}">
```

```
<td valign="top">
    <g2:evaluate var="jvar_pop_target
" expression="${ref}.getRecordClassName()" />
    <a class="linked" target="gsft_main"
n">
        href="sc_req_item.do?sys_id=${sc_task.request_item.sys_id}"
        onmousemove="popListDiv(event
, 'sc_req_item', '${sc_task.request_item.sys_id}', '${sysparm_view}')"
        onmouseout="lockPopup(event)"
        
    </a>
</td>
<td valign="top">
    <a class="linked" target="gsft_main"
n">
        href="sc_req_item.do?sys_id=${sc_task.request_item.sys_id}"
        ${sc_task.request_item.number}
    </a>
</td>
<td valign="top" width="50%">
    <g:call function="variable_summary
_approval.xml"
        question_name="${sc_task.reque
st_item.sys_id}"
        question_help_tag="${smart_des
cription}"
        sc_req_item="${sc_task.request
_item.sys_id}"
        help_class="${jvar_line_color}
"/>
    </td>
    <td valign="top"> <g:currency_format v
alue="${sc_task.request_item.price}" /> </td>
    <td valign="top"> ${sc_task.request_it
em.quantity}</td>
    <td valign="top"> <g:currency_format v
alue="${jvar_item_price}" /></td>
</tr>
</table>
</td>
```

```
</tr>
</j:jelly>
```

Display field messages

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.

These methods are used to change 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 work as expected.

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

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

Method Detail

Method Detail	Parameters	Example
showFieldMsg(input, message, type, [scrollform])	<ul style="list-style-type: none">• input — name of the field or control• message — message you would like to appear• type — 'info', 'error', or 'warning'; defaults to info if not supplied• scroll form — (optional) Set	<p>Error Message</p> <pre>g_form.showFieldMsg('impact','Low impact not allowed with High priority','error');</pre> <p>Impact: 3 - Low Low impact not allowed with High priority</p> <p>Informational Message</p>

Method Detail	Parameters	Example
	<p>scrollForm to false to prevent scrolling to the field message offscreen</p>	<pre>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');</pre> 
hideFieldMsg(input)	<ul style="list-style-type: none"> input — name of the field or control clearAll — (optional) boolean parameter indicating whether to clear all messages. If true, all messages for the field are cleared. If false or empty, only the last message is removed 	<p>Removing a Message</p> <pre>//this will clear the last message printed to the field g_form.hideFieldMsg('impact');</pre>

Legacy support

The `showErrorBox()` and `hideErrorBox()` are still available but simply call the new methods with type of error. You should use the new methods.

Log output

GSLog is a script include that simplifies script logging and debugging by implementing levels of log output, selectable by per-caller identified sys_properties values.

Log level

Logs can be at the level of Debug, Info, Notice, Warning, Err, or Crit (after BSD syslog.h and followers). The default logging level is Notice, so levels should be chosen accordingly.

Where to use

Use for any server-side script where you want to implement event logging.

For the API reference, see [GSLog\(\)](#).

For more information, see [Debugging scripts](#)

Modify a GlideDateTime field value

This example demonstrates how to modify a GlideDateTime field value using a script.

Name: Modify a GlideDateTime Field Value

Type: A server side script that accesses a GlideDateTime field.

Table: N/A

Description: Given a GlideDateTime field or script object, show a variety of ways to easily modify value. The same concept also applies to the GlideDate object.

Parameters: N/A

Script:

Note: This script is applicable to global applications only.

```
//You first need a GlideDateTime object
//this can be from instantiating a new object "var gdt = n
ew GlideDateTime()"
//or getting the object from a GlideDateTime field
//getting the field value (for example: var gdt = current.
start_date) only returns the string value, not the object
//to get the object use var gdt = current.start_date.getGl
ideObject();
//now gdt is a GlideDateTime object
var gdt = current.start_date.getGlideObject();

//All methods can use negative values to subtract interval
s

//add 1 hour (60 mins * 60 secs)
gdt.addSeconds(3600);

//add 1 day
gdt.addDaysLocalTime(1);

//subtract 1 day
gdt.addDaysLocalTime(-1);

//add 3 weeks
gdt.addWeeksLocalTime(3);

//subtract 6 months
gdt.addMonthsLocalTime(-6);

//add 1 year, representing the date and time using the UT
C timezone instead of the local user's timezone.
gdt.addYearsUTC(1);

//set the value of the GlideDateTime object to the curren
t session timezone/format
GlideSession.get().setTimeZoneName('US/Eastern');
gdt.setDisplayValue('2018-2-28 00:00:00');
gs.info('In ' + GlideSession.get().getTimeZoneName() + ":" +
" " + gdt.getDisplayValue());
```

Sample ASP.NET with C Sharp redirect with cookies

This sample ASP .NET code creates a simple authentication portal and passes an unencrypted HTTP header as a cookie.

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

Note: Cookies are domain specific and cannot be used across different network domains. The only domain that can read a cookie is the domain that sets it. It does not matter what domain name you set. If you do not have an option of your SSO portal being in the same network domain as your ServiceNow instance (for example, in an on-premises deployment), an alternative is to pass the SSO token using URL GET or POST parameters.

This sample assumes:

- The web server supports ASP .NET and C#
- The target ServiceNow instance is `https://<instance name>.service-now.com/`
- SiteMinder or another single sign-on application has pre-authenticated the user
- The target ServiceNow instance expects an HTTP header of `SM_USER`

Change the ASP code to redirect users to the proper ServiceNow instance.

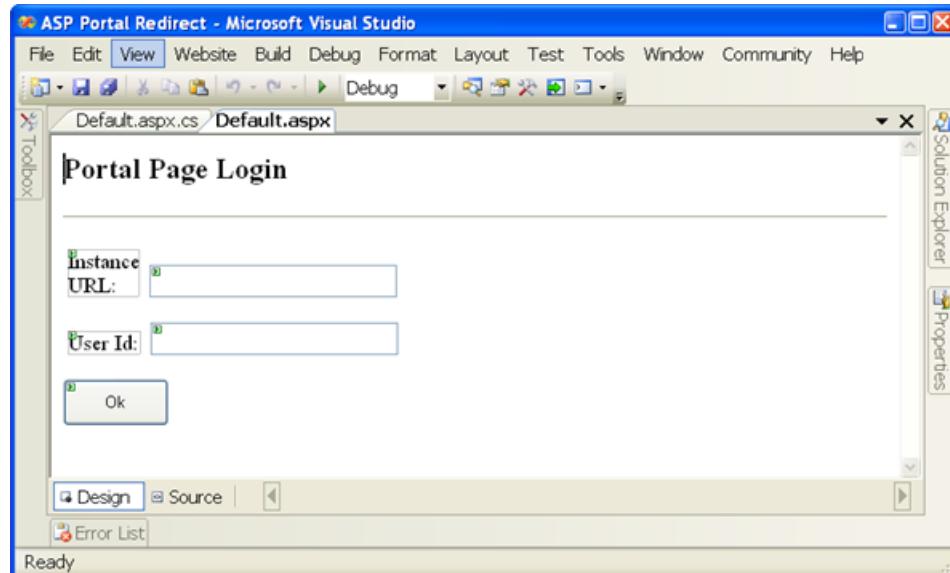
```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Portal Page Login</title>
<%--    <meta http-equiv="REFRESH" content="0;url=https://<instance name>.service-now.com/">--%>

</head>
<body>
    <form id="form1" runat="server">
        <h2><b> Portal Page Login </b></h2>
        <hr style="position: static" />
        <br />
```

```
<asp:Label ID="Label2" runat="server" Font-Size="Larger" Height="21px" Style="position: static" Text="Instance URL:" Width="113px"></asp:Label>
<asp:TextBox ID="urlBox" runat="server" Font-Size="Large" Style="position: static"></asp:TextBox><br />
<br />
<asp:Label ID="Label1" runat="server" Font-Size="Larger" Height="17px" Style="position: static;" Text="User Id:" Width="113px"></asp:Label>
<asp:TextBox ID="userNameBox" runat="server" Font-Size="Large" Style="position: static;"></asp:TextBox>

<br />
<br />
<asp:Button ID="Button1" runat="server" Height="39px" Style="position: static;" Text="Ok" Width="88px" OnClick="Button1_Click" />
</form>
</body>
</html>
</asp>
```

ASP Portal Redirect



The following C# code handles the OnClick button event for the form.

The code:

- Creates the cookie "SM_USER"
- Performs a redirect to the URL specified on the ASP form.

Change the C# code to create the proper cookie name.

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        try
        {
            HttpCookie myCookie = new HttpCookie("SM_USER");
            myCookie.Value = userNameBox.Text;
            Response.Cookies.Add(myCookie);
            Response.Redirect(urlBox.Text);

        }catch { }
    }
}
```

Useful approval assignment scripts

This is a searchable version of the useful approval and assignment scripts.

Warning: The customization described here was developed for use in specific instances, and is not supported by Now Support. This method is provided as-is and should be tested thoroughly before implementation. Post all questions and comments regarding this customization to our community [forum](#).

For an easy-to-navigate version, visit the Useful Scripts portal.

Assign a group for ESS requests

Type: Assignment Rule

Description: This script automatically assigns a group for all ESS Requests.

Script example:

```
if(current.opened_by.roles=="") {
    current.assignment_group.setDisplayValue('Network');
    current.update(); }
```

Assign Catalog Item to Group Based on Delivery Plan Task

Type: Assignment Rule

Description: This assignment rule assigns a service catalog item to the database group if it uses a delivery plan that has a catalog task assigned to the desktop group.

```
//Return catalog items that have no group but do have a delivery plan assigned
var ri =new GlideRecord("sc_cat_item");
ri.addQuery("group","=",null);
ri.addQuery("delivery_plan","!=",null);
ri.query();
while(ri.next()){
    gs.log("Found an item");
    //Return tasks that point to the same delivery plan as the above item
    var dptask =new GlideRecord("sc_cat_item_delivery_task");
    dptask.addQuery("delivery_plan","=",ri.delivery_plan);
    dptask.query();while(dptask.next()){
        gs.log("Found a task");var gp = dptask.group.getDi
```

```
splayValue();
    gs.log(gp); //If the task is assigned to desktop, assign the item's group to desktop
    if(dptask.group.getDisplayValue() == "Desktop") {
        ri.group.setDisplayValue("Desktop");
        gs.log("updating "+ ri.getDisplayValue());
        ri.update(); break;}}
```

Assign items with one task

Type: Assignment Rule

Description: Automatically assigns any catalog items with only one task associated to a particular group.

```
//Get the catalog item for the current requested item
var scCatItem =new GlideRecord("sc_cat_item");
if(scCatItem.get('sys_id', current.cat_item)){
// If the catalog item already has an assignment group or
if using workflow we don't need to make an assignment
    if(!scCatItem.delivery_plan.nil() && scCatItem.group.nil(
)) {
        var dpTask =new GlideRecord("sc_cat_item_delivery_
task");
        dpTask.addQuery("delivery_plan", "=", scCatItem.deli
very_plan);
        dpTask.query();
        if(dpTask.getRowCount() == 1 && dpTask.next()) {
// Check that there is only 1 record in the GlideR
ecord
            dpTask.group; } }}
```

Assignment based on workload

Type: Business Rule

Description: Populate the assigned_to based on the assignment group member who has the least amount of active incidents.

Parameters:

- order: >1000 if you want to execute after assignment rules
- condition: current.assigned_to == " && current.assignment_group != "

- when: before, insert/update

```
var assignTo = getLowestUser();
gs.addInfoMessage("assigning to is "+ assignTo);
current.assigned_to= assignTo;

function getLowestUser(){
    var userList =new Array();
    var cg =new GlideRecord('sys_user_grmember');
    cg.addQuery('group', current.assignment_group);
    cg.query();
    while(cg.next()) {
        var tech = cg.user.toString();
        var cnt = countTickets(tech);
        gs.addInfoMessage("Tech counts "+ cg.user.name+' '+ cnt +" "+ tech);
        userList.push({ sys_id: tech, name: cg.user.name, count : cnt });
    }

    for(var i=0; i < userList.length; i++){
        gs.addInfoMessage(userList[i].sys_id+" "+ userList[i].name+" "+ userList[i].count);
        userList.sort(function(a, b){
            gs.addInfoMessage("Sorting: "+ a.sys_id+"("+ a.count+
        );
        "+ b.sys_id+"("+ b.count+ ")");
        return a.count- b.count;
    });

    if(userList.length<=0) return "";
    return userList[0].sys_id;
}

function countTickets(tech){
    var ct =new GlideRecord('incident');
    ct.addQuery('assigned_to',tech);
    ct.addQuery('active',true);
    ct.query();
    return ct.getRowCount();
}
```

Run assignment rules when category is changed

Type: Client script

Table: Incident

Description: This example is an onChange client script on the category field within Incident. Note: this script used to use synchronous AJAX (asynchronous behavior is specified by the third parameter of the ajaxRequest call). The implementation below uses asynchronous AJAX. The drawback of using the synchronous version is that a network response problem could cause the browser to hang.

```
// Make an AJAX request to the server to get who this incident would be
// assigned to given the current values in the record. This runs the assignment
// rules that have been defined in System Policy and returns the assigned_to and
// the assignment_group

function onChange(control, oldValue, newValue, isLoading){
    if(isLoading){return;
    // No change, do not do anything
    }

    // Construct the URL to ask the server for the assignment
    var url ="xmlhttp.do?sysparm_processor=AJAXAssignment&sys_target=incident";
    var uv = g_form('sys_uniqueValue');
    if(uv){
        url += "&sys_uniqueValue=" + uv.value;
    }
    // Make the AJAX request to the server and get the response
    var serial = g_form.serialize();
    // get all values currently assigned to the incident
    var response = ajaxRequest(url, serial,true, responseFunc);
}

// This callback function handles the AJAX response.
function responseFunc(response){
    var item= response.responseXML.getElementsByName("item")[0];
    // Process the item returned by the server
    if(item){
        // Get the assigned_to ID and its display value and put them on the form
        varname=item.getAttribute("name");
        var name_label =item.getAttribute("name_label");
```

```
if(name_label &&name){  
    g_form.setValue('assigned_to',name, name_label);}  
else{  
    g_form.setValue('assigned_to','','');}  
// Get the assignment_group ID and its display value a  
nd put then on the form  
var group =item.getAttribute("group");  
var group_label =item.getAttribute("group_label");  
if(group_label && group){  
    g_form.setValue('assignment_group', group, group_lab  
el);}  
else{  
    g_form.setValue('assignment_group','','');}}}
```

Custom approval UI macro

Type: UI macro

The following option illustrates how to obtain more detail from the **My Approvals** view of an Execution Plan by creating a new UI Macro.

- Navigate to **System UI** and click **UI Macros**.
- Rename the existing "approval_summarizer_sc_task" to something like "approval_summarizer_sc_task_old" and deactivate it.
- Create a new one using the same name ("approval_summarizer_sc_task"). The name should basically tell you what the macro does and to what it applies. In this case, we're replacing an existing one so we decided to re-use the existing name.

Macros		Description	Active
<input type="checkbox"/>	approval_summarizer		true
<input type="checkbox"/>	approval_summarizer_change_request		true
<input type="checkbox"/>	approval_summarizer_default		true
<input type="checkbox"/>	approval_summarizer_no_request		true
<input checked="" type="checkbox"/>	approval_summarizer_sc_task		true
<input type="checkbox"/>	approval_summarizer_sc_task_old		false
<input type="checkbox"/>	approval_variable_summary		true
<input type="checkbox"/>	catalog_cart_default		true
<input type="checkbox"/>	catalog_model_info		true
<input type="checkbox"/>	catalog_user_info		true
<input type="checkbox"/>	context_form_header	Context menu for a form header	true
<input type="checkbox"/>	context_list_header	Context menu for a list header	true
<input type="checkbox"/>	decorate_list_actions	List actions are the column above the ch...	true
<input type="checkbox"/>	decorate_welcome_header Stripe	Provide content on the right side of the...	true
<input type="checkbox"/>	decorate_welcome_header_left	Provide content on the left side of the ...	true
<input type="checkbox"/>	diagrammer	Includes for the Visual Diagrammer	true
<input type="checkbox"/>	diagram_scale	Provides scale drop down select for diag...	true
<input type="checkbox"/>	dialog_button	Create a button that performs an arbitra...	true
<input type="checkbox"/>	dialog_buttons_ok_cancel	OK and Cancel buttons for a dialog Defa...	true

Then copy the xml script at the bottom of this article into the xml code window in the new UI Macro. This is great way to give some detail to an approver when you are doing line item approvals using approval tasks within the Service Catalog Execution Plans.

Different ways

Old way

This is the view you see in **My Approvals** when using an approval task using the old method.

Approver:	Ted Lucci	Approval for:	TASK10041	Update	Approve	Reject	Delete												
State:	Requested																		
Comments:																			
<p>Summary of Task being approved:</p> <table border="1"> <tr> <td>Short Description:</td> <td>Get the Department head approval for this test item</td> </tr> <tr> <td>Assigned To:</td> <td></td> </tr> <tr> <td>Priority:</td> <td>4</td> </tr> <tr> <td>Requested By:</td> <td>Jannie Caruvana</td> </tr> <tr> <td>Requested For:</td> <td>Christen Mitchell</td> </tr> <tr> <td>Description:</td> <td></td> </tr> </table>								Short Description:	Get the Department head approval for this test item	Assigned To:		Priority:	4	Requested By:	Jannie Caruvana	Requested For:	Christen Mitchell	Description:	
Short Description:	Get the Department head approval for this test item																		
Assigned To:																			
Priority:	4																		
Requested By:	Jannie Caruvana																		
Requested For:	Christen Mitchell																		
Description:																			
<input type="button" value="Update"/> <input type="button" value="Approve"/> <input type="button" value="Reject"/> <input type="button" value="Delete"/>																			

Notice there is not much detail telling the approver what they are actually approving. You can see the short description of the task but not much information about the item .

New way

This is the view you will see if you use the xml script below in place of the OOB (out-of-box) UI Macro.

The screenshot shows a ServiceNow approval interface. At the top, there are fields for 'Approver' (Fred Lucy) and 'State' (Requested). To the right, there are buttons for 'Update', 'Approve', 'Reject', and 'Delete'. Below these, there is a 'Comments' area and a summary section. The summary includes details about the item being approved: opened by Janie Carvana, requested for Christen Mitchell, and a total amount of \$50.00. A table below shows the item's details: number RITM10012, description 'Test Item Short Description', price \$25.00, quantity 2, and total \$50.00. At the bottom of the interface are buttons for 'Update', 'Approve', 'Reject', and 'Delete'.

Using this method you can see details much like the request approval. You have a link into the item ordered, a short description (which contains the ability to expand the variables from the item), price, quantity and the total price. This helps the approver in that it shows more detail. They can now see what they are actually approving.

Useful field scripts

Common use cases for field customization scripts.

Warning: The customization described here was developed for use in specific instances, and is not supported by Now Support. This method is provided as-is and should be tested thoroughly before implementation. Post all questions and comments regarding this customization to our community [forum](#).

For more information, see [Server API reference](#).

Automatically populate a field

The following example shows how to use a client script to auto-fill a **Short Description** based on the selected **Subcategory**.

In this case, if the table has a record with **Subcategory** = Password and **Short Description** = Password Reset. When the user selects the **Subcategory** of Password on the Incident form, a client script looks up the matching record and sets **Short Description** equal to Password Reset.

Client script settings:

- Type = onChange
- Table name = incident
- Field name = Subcategory

```
function onChange(control, oldValue, newValue, isLoading){  
    if(isLoading){return;}  
    var newrec = get('sys_row');  
    //Check if new record  
    if (newrec.value == -1) {  
        var lookup = new GlideRecord('u_short_desc_lookup');  
        lookup.addQuery('u_subcategory', g_form.getValue('subcategory'));  
        lookup.query();  
        var temp; //temp var - reusable  
        if(lookup.next()){  
            temp = lookup.u_short_description;  
            if(null != temp){  
                //Set the form value from lookup if there  
                is a lookup value  
                g_form.setValue('short_description', temp)  
            }  
            } else {  
                g_form.setValue('short_description', "");  
            }  
        } else {  
            //If a lookup record does not exist based o  
            n lookup.addQuery  
            //Then set to UNDEFINED or NULL depending o  
            n type  
            g_form.setValue('short_description', "");  
        }  
    }  
}
```

Disable HTML tags in descriptions

This code disables HTML tags in **Description** and **Short Description** fields by substituting the tags with non-executing versions.

```
doit();
```

```
function doit(){
    var desc = current.description.toString();
    var shdesc = current.short_description.toString();
    if(desc.indexOf('script')>-1|| shdesc.indexOf('script')>-1) {
        desc = desc.replace(/<script>/g,"(script)");
        current.description = desc.replace(/<\/script>/g,"(\!/sc
ript)");
        shdesc = shdesc.replace(/<script>/g,"(script)");
        current.short_description = shdesc.replace(/<\/script>/
g,"(\!/script)");
    }
}
```

Eliminate leading and trailing spaces in fields

This example of the script trims trailing and leading spaces in the FirstName and LastName fields of the sys_user.

```
doit();

function doit(){
    var now_GR =new GlideRecord('sys_user');
    gr.query();
    while(gr.next()){
        if((gr.first_name.toString().length!= gr.first_name.t
oString().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();}}
    }
```

Make a field label flash

The following client script example is for the number field on incident. The label flashes for two seconds.

```
g_form.flash("incident.number", "#FFFACD", 0);
```

The arguments for the flash method are as follows:

- tablename.fieldname
- RGB color or acceptable CSS color like "blue" or "tomato"

- Integer that determines how long the label flashes:
 - 2 for a 1-second flash
 - 0 for a 2-second flash
 - -2 for a 3-second flash
 - -4 for a 4-second flash

Note: Do not specify this argument if you want the field label colored the specified color.

Make a field label bold

This client script makes the label of a particular field bold. In this case, the field is the **Short Description** on the **Incident Table**.

```
function onLoad() {
    var l = g_form.getLabel('incident.short_description');
    l.style.fontWeight = 'bold';}
```

Make fields read-only

This onLoad client script makes the following fields on the Incident [incident] table read-only:

- **Incident state**
- **Impact**
- **Urgency**
- **Priority**
- **Configuration item**
- **Assigned to**

The script also removes the magnifying glass for the read-only Reference Fields (**Configuration item** and **Assigned to**).

```
function onLoad() {
var incidentState = g_form.getValue('incident_state');
if( incidentState == '6' || incidentState == '7'){
    g_form.setReadonly('incident_state',true);
```

```
g_form.setReadonly('impact',true);
g_form.setReadonly('urgency',true);
g_form.setReadonly('priority',true);
g_form.setReadonly('cmdb_ci',true);
g_form.setReadonly('assigned_to',true);}}
```

Set current date/time in field

You can set date and time values in client scripts and script includes.

Client script

You can use the following two lines to set the current date and time in a date/time field. This approach bypasses the problem of getting the value into the proper format and proper time zone.

```
var ajax = new GlideAjax('MyDateTimeAjax');
ajax.addParam('sysparm_name','nowDateTime');
ajax.getXML(function(){
    g_form.setValue('put your field name here', ajax.getAnswer());});
```

For more information on running server-side scripts with the client, refer to [GlideAjax](#).

System script include

```
// Be sure the "Client callable" checkbox is checked

var MyDateTimeAjax = Class.create();
MyDateTimeAjax.prototype = Object.extendsObject(AbstractAjaxProcessor, {
    nowDateTime:function(){
        return gs.nowDateTime();}});
```

Toggle the timer field by field name

The following client script toggles the timer field based on a particular field name.

```
function toggleTimerByFieldName(fieldName) {
    //Step 1: Find the timer object
    //timeObjectName: the timer objects name as it would normally be referenced
    //timeObjectHidden: the hidden input node in the field td
```

```
//timeObjectParent: the parent td node containing the file  
ld and it's constituent nodes  
//timeObjectFields: anchor tag with onclick to stop timer  
  
var timeObjectName = fieldName;  
var timeObjectHidden = get(timeObjectName);  
  
//Step 2: simulate click stop button  
var timeObjectParent;  
var timeObjectFields;  
  
//verify that we got the correct object  
if(timeObjectHidden.type=="hidden") {  
  
    //Get Parent td node  
    timeObjectParent = timeObjectHidden.parentNode;  
  
    //Get input fields  
    timeObjectFields = timeObjectParent.getElementsByTagName("input");  
  
    //simulate click of stop button  
    var timerTestString ="paused";  
    var timerImg;  
  
    //loop through input objects looking for the pause timer  
    // object  
    for(var elIt=0; elIt < timeObjectFields.length; elIt++) {  
        if(timeObjectFields[elIt].id.match(timerTestString)) {  
            if(timeObjectFields[elIt].value=="false") {  
                timeObjectFields[elIt].value="true";  
                timerImg = timeObjectParent.getElementsByTagName("img")[0];  
                timerImg.src="images/timer_start.gifx";}  
            elseif(timeObjectFields[elIt].value=="true") {  
                timeObjectFields[elIt].value="false";  
                timerImg = timeObjectParent.getElementsByTagName("img")[0];  
                timerImg.src="images/timer_stop.gifx";}}}}}
```

Modify GlideDateTime field value

The following example uses a server-side script to access a GlideDateTime field.

The following server-side script example shows how to modify values using the GlideDateTime API. The same concept also applies to the GlideDate object.

```
//You first need a GlideDateTime object
//this can be from instantiating a new object "var gdt = new GlideDateTime()"
//or getting the object from a GlideDateTime field
//getting the field value (for example: var gdt = current.start_date)
//only returns the string value, not the object
//to get the object use var gdt = current.start_date.getGlideObject();
//now gdt is a GlideDateTime object
var gdt = current.start_date.getGlideObject();

//All methods can use negative values to subtract intervals
//add 1 hour (60 mins * 60 secs)
gdt.addSecondsLocalTime(3600);

//add 1 day
gdt.addDaysLocalTime(1);

//subtract 1 day
gdt.addDaysLocalTime(-1);

//add 3 weeks
gdt.addWeeksLocalTime(3);

//subtract 6 months.
gdt.addMonthsLocalTime(-6);

//add 1 year, representing the date and time using the UTC timezone instead of the local user's timezone.
gdt.addYearsUTC(1);
```

Useful scheduling scripts

A business rule script specifies the actions that the business rule takes. Scripts commonly include predefined global variables to reference items in your system, such as the current record. Global variables are available to all business rules.

Warning: The customization described here was developed for use in specific instances, and is not supported by Now Support. This method is provided as-is and should be tested thoroughly before implementation. Post all questions and comments regarding this customization to our community [forum](#).

Calculate duration given a schedule

Type: Before update/insert business rule

Table: Incident

Description: A Business Duration calculates the Open to Close duration on an incident based on the particular [Creating and using schedules](#). If there is no schedule specified, the script will simply use the first schedule returned by the query.

Script example:

The example below sets the resolved duration when the incident state moves to resolved.

```
if(current.incident_state==6){  
    var dur = calcDurationSchedule(current.opened_at,current  
.sys_updated_on);  
    current.u_resolved_duration= dur;  
  
    function calcDurationSchedule(start, end) {  
        // Get the user  
        var usr =new GlideRecord('sys_user');  
        usr.get(gs.getUserID());  
        // Create schedule - pass in the sys_id of your standar  
d work day schedule and pass in the users timezone  
        var sched =new GlideSchedule('08fcd0830a0a0b2600079f56b1  
adb9ae',usr.time_zone);  
        // Get duration based on schedule/timezone
```

```
    return(sched.duration(start.getGlideObject(), end.getGlideObject())); }
```

Check upcoming termination dates

Type: Scheduled script

Description: This script checks nightly for termination dates on contracts coming up in 90, 50, or 10 days (depending on the contract duration field).

Script example:

```
function contractNoticeDue() {
    var now_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";
        } elseif((gr.u_termination_date<= gs.daysAgo (-50) && (gr.u_contract_duration=="Medium")){
            gr.u_contract_status="In review";
        } elseif((gr.u_termination_date<= gs.daysAgo (-10))&&(gr.u_contract_duration=="Short")){
            gr.u_contract_status="In review";
        }
        gr.update();
    }
}
```

Use scripts in business rules to accomplish common tasks such as:

- Comparing two date fields.
- Parsing XML payloads.
- Aborting a database action in a business rule.

With scripts, you can also:

- Specify the operation that triggers the business rule.
- Use the scratchpad with display business rules to change form values just before a user loads the form.

- Use the OR condition like you would in a condition builder.

You can also utilize the system's scripting functionality available for server-side scripts.

You can use options on the Business Rules form to build conditions, set field values, and display alert messages without needing to write a script.

Abort a database action in a business rule

During a before business rule script, you can cancel or abort the current database action using the current.setAbortAction(true) method.

For example, if the before business rule is executed during an insert action, and you have a condition in the script that calls current.setAbortAction(true), the new record stored in current is not created in the database.

Add autofill functionality

Add autofill functionality is also called incident template, auto assignments, quick calls, call script, or auto populate.

Let's say you want to auto-fill your **Short Description** based on the **Subcategory** selected. First, create a lookup table, then populate the key field, in this case **Subcategory** and the auto-filled field, **Short Description**. So let's say your table had a record with **Subcategory** = Password and **Short Description** = Password Reset. When the user selects the subcategory of Password on the Incident form a client script looks up the matching record and sets short description equal to Password Reset. Client script settings... **Type** = onChange, **Table name** = incident, **Field name** = Subcategory.

```
function onChange(control, oldValue, newValue, isLoading)
{
    if (isLoading) { return; }
    var newrec = get('sys_row');
    //Check if new record
    if (newrec.value == -1) {
        var lookup = new GlideRecord('u_short_desc_lookup');
        lookup.addQuery('u_subcategory', g_form.getValue('subcategory'));
        lookup.query();
        var temp; //temp var - reusable
```

```
    if (lookup.next()) {
        temp = lookup.u_short_description;
        if (null != temp) { //Set the form value from look
up if there is a lookup value
            g_form.setValue('short_description', temp); }
        else {
            g_form.setValue('short_description', ""); } }
    else {
        //If a lookup record does not exist based on lookup.a
ddQuery
        //Then set to UNDEFINED or NULL depending on type
        g_form.setValue('short_description', ""); } }

}
```

You could populate many fields or even pull in call script questions into the **Comments** field so call center personnel gather good information to pass on to a technician. There are already Assignment Rule, Templates and Wizards built in that perform similiar functions.

Example script: A default before-query business rule

You can use a query business rule that executes before the database query is made to prevent users from accessing certain records.

Warning: The customization described here was developed for use in specific instances, and is not supported by Now Support. This method is provided as-is and should be tested thoroughly before implementation. Post all questions and comments regarding this customization to our community [forum](#).

Consider the following example from a default business rule that limits access to incident records.

Default business rule limits access to incident records

Name	Table	When
incident query	Incident	before, query

Example script

This example prevents users from accessing incident records unless they have the itil role listed in the Caller or Opened by field. So, for example, when self-service users open a list of incidents, they can only see the incidents they submitted.

```
if(!gs.hasRole("itil")&& gs.isInteractive()) {
    var u = gs.getUserID();
    var qc = current.addQuery("caller_id", u).addOrCondition
("opened_by", u).addOrCondition("watch_list", "CONTAINS", u
);
    gs.print("query restricted to user: "+ u);}
```

Note: You can also use access controls to restrict the records that users can see.

Schedule script for weekdays

Type: Business Rules/Client Scripts

This script schedules the script for weekdays. Insert any script where it says "Your Script Here."

```
var go ='false';
var now =new Date();

// Correct time zone, which is by default GMT -7
now.setHours(now.getHours()+8);
var day = now.getDay();

// No go on Saturday or Sunday
if(day !=0&& day !=6){

// (your script here)

}
```

Set date field according to current date

This script sets a date field depending on the current day of the week. In this example, if the day is Monday through Wednesday, it sets the date to this coming Monday; otherwise it sets the date field to next Monday.

```
function setCabDate() {
var today =new Date();
```

```
var thisDay = today.getDay();
//returns 0 for Sunday, 1 for Monday, etc. thru 6 for Saturday.
var thisMon =new GlideDateTime();
thisMon.setDisplayValue(gs.beginningOfThisWeek());
var nextMon = thisMon.getNumericValue();
nextMon +=(1000*60*60*24*7);

if((thisDay <4)&&(thisDay >0))
    //if today is Mon thru Wed (thisDay = 1, 2, or 3), set cab to this coming Monday.
    current.u_req_cab_rev_date.setDateNumericValue(thisMon.getNumericValue());
elseif((thisDay >=4) || (thisDay ==0))
    //if today is Thurs thru Sun (thisDay = 4, 5, 6, or 0), set cab to next Monday.
    current.u_req_cab_rev_date.setDateNumericValue(nextMon);
}
```

To validate the input of all date/time fields, you can use the following in a validation script (**System Definition > Validation Scripts**). Because the date/time format is hard coded in this script, it must match your instance's date/time format. If your instance's date/time format changes, you must update your validation script.

Set the validation script's type to Date/Time. Then, with this validation script, if a user enters an incorrect format in a date/time field, they will receive an error message.

```
function validate(value){
// empty fields are still valid dates
if(!value) return true;

// We "should" have the global date format defined always
defined. but there's always that edge case...
if(typeof g_user_date_time_format !=='undefined')return is
Date(value, g_user_date_time_format);

// if we don't have that defined, we can always try guessing
return parseDate(value)!==null;}
```

Date/time validation

The screenshot shows the ServiceNow Validation Script interface. At the top, there are buttons for 'Update' and 'Delete'. Below that, the 'Type' is set to 'glide_date_time' and 'Active' is checked. The 'Description' field contains the text 'Validates format of date/time fields'. The 'Validator' section contains the following JavaScript code:

```
function validate(value) {
    if (!value) {
        return true;
    }
    return (getDateFromFormat(value, 'yyyy-MM-dd HH:mm:ss') != 0);
}
```

At the bottom, there are 'Update' and 'Delete' buttons.

Using client and server code in a UI action

You can use a script to validate input upon a UI Action click on the client side before updating the record on the server side. The user will not have to click the button twice to validate the required fields and update the record.

The script calls the client function for client-side validation, and the UI action completes the task if it passes. The code that runs without `onclick` statement ensures that the server-side function does not run until the client function is no longer running. If successful, the server-side function runs and updates the record.

```
// Client-side onclick function
function resolveIncident() {
    // Set the 'Incident state' and 'State' values to 'Resolved', and display mandatory fields
    g_form.setValue('incident_state', 6);
    g_form.setValue('state', 6);
    g_form.setValue('resolved_by', g_user.userID);

    // Call the UI action and skip the 'onclick' function
    gsftSubmit(null, g_form.getFormElement(), 'resolve_incident'); //MUST call the 'Action name' set in this UI Action
}
```

```
}

// Code that runs without 'onclick'
// Ensure call to server-side function with no browser errors
if (typeof window == 'undefined')
    serverResolve();

// Server-side function
function serverResolve() {
    current.incident_state = IncidentState.RESOLVED;
    current.state = IncidentState.RESOLVED;
    current.resolved_by = gs.getUserID();
    current.update();
}
```

gsftSubmit(String control, Object form, String action_name)

Submits a form as if the user had clicked a UI Action after checking for required fields and executing onSubmit() client scripts. Enables calling client-side code and server-side code and in a single UI action.

Parameters

Name	Type	Description
control	String	Name of a form button on which you want to simulate a user click. Use null otherwise.
form	Object	Optional. The form element that contains the submitted input. You can retrieve this value by calling the g_form.getFormElement() method.
action_name	String	Action name. This value is provided in the record listed in the UI Actions [sys_ui_action] table. For example, 'resolve_incident' is the action name of the Resolve button in the Incident [incident] table.

Related concepts

- [GlideUser - Client](#)

Related reference

- [GlideForm - getFormElement\(\)](#)
- [Scoped GlideSystem - eventQueue\(String name, Object instance, String parm1, String parm2, String queue\)](#)

Sample ASP Script for unencrypted single sign-on

This sample ASP .NET code creates a simple authentication portal and passes an unencrypted HTTP header as a URL parameter.

This sample assumes:

- The web server supports ASP .NET
- The target instance is `https://<instance name>.service-now.com/`
- SiteMinder or another single sign-on application has pre-authenticated the user
- The target instance expects an HTTP header of `SM_USER`

Change the ASP code to redirect users to the proper instance and create the proper HTTP header.

```
<html xmlns = "http://www.w3.org/1999/xhtml" > <head run
at = "server" > <title>Portal Page Login </ title > <%
--> <meta http-equiv = "REFRESH" content = "0;url=https:
//<instance name>.service-now.com/">--%>

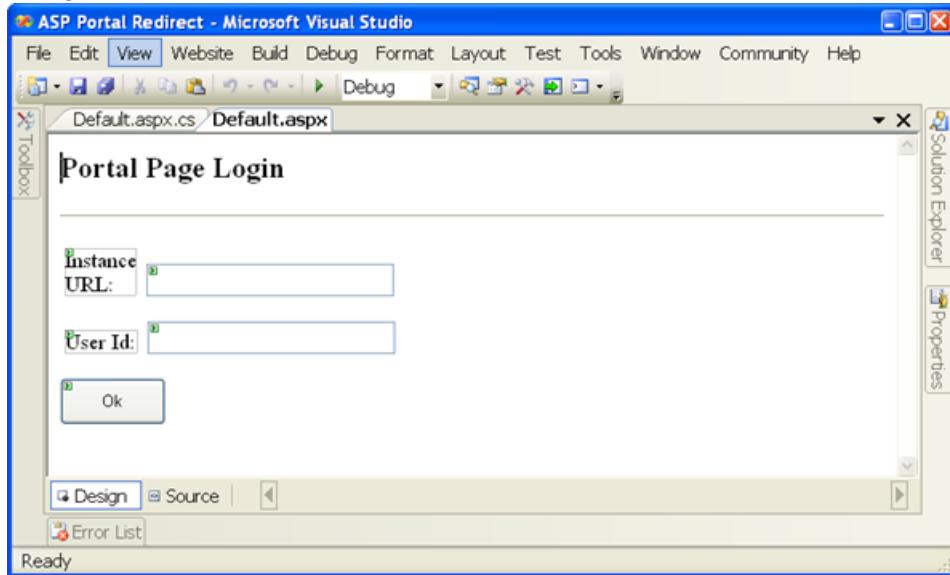
<script runat = "server" >

    public void go_to(object sender, EventArgs e)
    {
        //Send URL parameters
        String URL = urlBox.Text + "?SM_USER=" + userNameBox.Text;
        Response.Redirect(URL);
    }
}
```

```
</ script >
```

```
</ head > <body > <form id = "form1" runat = "server"
> <h2 >< b > Portal Page Login </ b >< / h2 > <hr style
= "position: static" /> <br /> <asp:Label ID = "Label2
" runat = "server" Font-Size = "Larger" Height = "21px
" Style = "position: static" Text = "Instance URL:" Width
= "113px" >< / asp:Label> <asp:TextBox ID = "urlBox" runa
t = "server" Font-Size = "Large" Style = "position: stat
ic" >< / asp:TextBox> <br /> <br /> <asp:Label ID = "Lab
el1" runat = "server" Font-Size = "Larger" Height = "17
px" Style = "position: static;" Text = "User Id:" Width =
"113px" >< / asp:Label> <asp:TextBox ID = "userNameBox" ru
nat = "server" Font-Size = "Large" Style = "position: st
atic;" >< / asp:TextBox> <br /> <br /> <asp:Button ID =
"Button1" runat = "server" Height = "39px" Style = "posit
ion: static;" Text = "Ok" Width = "88px" OnClick = "go_to"
/ > </ form > </ body > </ html >
```

Designer



Validate date and time

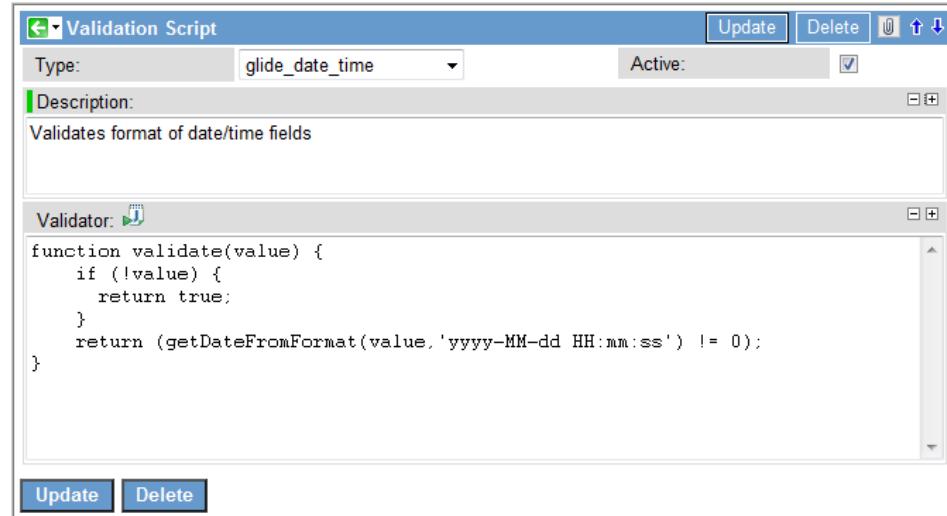
To validate the input of all date/time fields, you can use the following in a validation script (**System Definition > Validation Scripts**).

Because the date/time format is hardcoded in this script, it must match your instance's date/time format. If your instance's date/time format changes, you must update your validation script.

Set the validation script's type to "glide_date_time". Then, with this validation script, if a user enters an incorrect format in a date/time field, they will receive an error message.

```
function validate (value ) { if ( !value ) { return true ; } return (getDateFromFormat (value , 'yyyy-MM-dd HH:mm:ss' ) != 0 ) ; }
```

Date/time validation



Calculating durations

Often you may need to provide users with a way to specify when a task or process is due. Using the DurationCalculator script include, you can calculate the due date using either a simple duration or relative duration.

Typically, setting a due date requires that you calculate work time rather than the total time. Only the part of the day when work is performed is considered when determining the due date. For example, a task is due in 10 hours, but is restricted to a business day schedule. If the work starts at 10am on Monday, it is due on Tuesday at 12pm as calculated below.

10am-5pm on Monday (6 hours) + 8am-12pm on Tuesday (4 hours)

For information on schedules, which you can use as inputs to DurationCalculator methods, see [Creating and using schedules](#).

This script demonstrates how to use DurationCalculator to compute a due date.

```
/**  
 * Demonstrate the use of DurationCalculator to compute a  
due date.  
 *  
 * You must have a start date and a duration. Then you can  
compute a  
* due date using the constraints of a schedule.  
*/  
  
gs.include('DurationCalculator');  
executeSample();  
  
/**  
 * Function to house the sample script.  
 */function executeSample(){  
  
    // First we need a DurationCalculator object.var dc =  
new DurationCalculator();  
  
    // ----- No schedule examples -----  
----  
  
    // Simple computation of a due date without using a sc  
hedule. Seconds// are added to the start date continuously  
to get to a due date.  
    dc.setStartTime("5/1/2012");if(!dc.calcDuration(2*  
24*3600)){// 2 days  
        gs.log("*** Error calculating duration");return;  
    gs.log("calcDuration no schedule: "+ dc.getEndDateTim  
e());// "2012-05-03 00:00:00" two days later  
  
    // Start in the middle of the night (2:00 am) and comp  
ute a due date 1 hour in the future// Without a schedule t  
his yields 3:00 am.
```

```
    dc.setStartTime("5/3/2012 02:00:00");if(!dc.calcDuration(3600)){  
        gs.log("*** Error calculating duration");return;  
        gs.log("Middle of night + 1 hour (no schedule): "+ dc.  
getEndTime());// No scheduled start date, just add 1 h  
our  
  
        // ----- Add a schedule to the date calculato  
r -----  
        addSchedule(dc);  
  
        // Start in the middle of the night and compute a due  
date 1 hour in the future.// Since we start at 2:00 am the  
computation adds the 1 hour from the start// of the day,  
8:00am to get to 9:00am  
        dc.setStartTime("5/3/2012 02:00:00");if(!dc.calcDuration(3600)){//  
            gs.log("*** Error calculating duration");return;  
            gs.log("Middle of night + 1 hour (with 8-5 schedule): "  
"+ dc.getEndTime());// 9:00 am  
  
            // Start in the afternoon and add hours beyond quitting  
time. Our schedule says the work day// ends at 5:00pm, if  
the duration extends beyond that, we roll over to the nex  
t work day.// In this example we are adding 4 hours to 3:0  
0pm which gives us 10:00 am the next day.  
            dc.setStartTime("5/3/2012 15:00:00");if(!dc.calcDuration(4*3600)){//  
                gs.log("*** Error calculating duration");return;  
                gs.log("Afternoon + 4 hour (with 8-5 schedule): "+ dc.  
getEndTime());// 10:00 am.  
  
                // This is a demo of adding 2 hours repeatedly and exa  
mine the result. This// is a good way to visualize the res  
ult of a due date calculation.  
                dc.setStartTime("5/3/2012 15:00:00");// for(var i=  
2; i<24; i+=1){if(!dc.calcDuration(i*3600)){//  
                    gs.log("*** Error calculating duration");retur  
n;  
                    gs.log("add "+ i +" hours gives due date: "+ dc.ge  
tEndTime());}  
  
                // Setting the timezone causes the schedule to be inte
```

```
rpreted in the specified timezone.// Run the same code as
above with different timezone. Note that the 8 to 5 workda
y is// offset by the two hours as specified in our timezon
e.
    dc.setTimeZone("GMT-2");
    dc.setStartTime("5/3/2012 15:00:00");for(var i=2;
i<24; i+=1){if(!dc.calcDuration(i*3600)){//
        gs.log("*** Error calculating duration");retur
n;}
    gs.log("add "+ i +" hours gives due date (GMT-2):
"+ dc.getEndTime());}}
```

```
/**
 * Add a specific schedule to the DurationCalculator objec
t.
 *
 * @param durationCalculator An instance of DurationCalcul
ator
 */
function addSchedule(durationCalculator){// Load the "8-5
weekdays excluding holidays" schedule into our duratio
n calculator.var scheduleName ="8-5 weekdays excluding hol
idays";var grSched =new GlideRecord('cmn_schedule');
    grSched.addQuery('name', scheduleName);
    grSched.query();if(!grSched.next()){
        gs.log('*** Could not find schedule "'+ scheduleNa
me +'');return;}
    durationCalculator.setSchedule(grSched.getUniqueValue(
), "GMT");}
```

Simple duration vs relative duration

How much work is required to complete a task can be expressed as a "relative duration".

Relative duration determines the expected due date and time relative to the starting time. Examples of relative durations include "Next business day by 4pm," or "2 business days by 10:30am."

To calculate a relative duration, the calendar and time zone must be considered to determine what "next business day" means since it is the calendar that defines which days are valid work days and the time zone will affect the result as well. As an example, consider "Next business day by 4pm":

- If it is Monday at 12pm: Next business day by 4pm => Tuesday at 4pm
- If it is Friday at 2pm: Next business day by 4pm => the following Monday at 4pm

Note: Next business day is often defined by a starting day and time. For example, "next business day at 4pm if before 2pm" indicates that if the current time is after 2pm on a business day, then "Next business day" really means 2 business days since today does not count.

Calculating a simple duration

This business rule and script example demonstrate how to calculate a simple duration.

```
var dur =new DurationCalculator();
dur.setSchedule(current.schedule);
dur.setStartTime("");

if(current.duration_type==""){
    dur.calcDuration(current.duration.getGlideObject()
).getNumericValue()/1000);}else{
    dur.calcRelativeDuration(current.duration_type);}

current.end_date_time= dur.getEndDateTime();
current.work_seconds= dur.getSeconds();
```

This script demonstrates how to use DurationCalculator to calculate a simple duration.

```
/**
 * Sample script demonstrating use of DurationCalculator to
 * compute simple durations
 *
 */

gs.include('DurationCalculator');
executeSample();

/**
 * Function to house the sample script.
 */
```

```
function executeSample() {  
  
    // First we need a DurationCalculator object.  
    var dc =new DurationCalculator();  
  
    // Compute a simple duration without any schedule. Th  
e arguments  
    // can also be of type GlideDateTime, such as fields f  
rom a GlideRecord.  
    var dur = dc.calcScheduleDuration("5/1/2012","5/2/2012  
");  
    gs.log("calcScheduleDuration no schedule: "+ dur);  
    // 86400 seconds (24 hours)  
  
    // The above sample is useful in limited cases. We alm  
ost always want to  
    // use some schedule in a duration computation, let's  
load a schedule.  
    addSchedule(dc);  
  
    // Compute a duration using the schedule. The schedule  
    // specifies a nine hour work day. The output of this  
is 32400 seconds, or  
    // a nine hour span.  
    dur = dc.calcScheduleDuration("5/23/2012 12:00","5/24/  
2012 12:00");  
    gs.log("calcScheduleDuration with schedule: "+ dur);  
    // 32400 seconds (9 hours)  
  
    // Compute a duration that spans a weekend and holiday  
. Even though this  
    // spans three days, it only spans 9 work hours based  
on the schedule.  
    dur = dc.calcScheduleDuration("5/25/2012 12:00","5/29/  
2012 12:00");  
    gs.log("calcScheduleDuration with schedule spaning hol  
iday: "+ dur);  
    // 32400 seconds (9 hours)  
  
    // Use the current date time in a calculation. The out  
put of this is  
    // dependent on when you run it.  
    var now =new Date();  
    dur = dc.calcScheduleDuration("5/15/2012",new GlideDat
```

```
eTime());
    gs.log("calcScheduleDuration with schedule to now: "+
dur);
    // Different on every run.}

/**
 * Add a specific schedule to the DurationCalculator object.
 *
 * @param durationCalculator An instance of DurationCalculator
 */
function addSchedule(durationCalculator) {
    // Load the "8-5 weekdays excluding holidays" schedule into our duration calculator.
    var scheduleName = "8-5 weekdays excluding holidays";
    var grSched =new GlideRecord('cmn_schedule');
    grSched.addQuery('name', scheduleName);
    grSched.query();if(!grSched.next()){
        gs.log('*** Could not find schedule "'+ scheduleName +'"');
        return;
    }
    durationCalculator.setSchedule(grSched.getUniqueValue());
}
```

Calculating a relative duration

An example of a relative duration calculation script.

This script calculates the relative duration for "Next day at 4pm if after 10am":

```
// Next day at 4pm if before 10am
var days =1;
if(calculator.isAfter(calculator.startDateTime,"10:00:00"))
)
    days++;

calculator.calcRelativeDueDate(calculator.startDateTime, days,"16:00:00");
```

This script demonstrates how to use DurationCalculator to calculate a relative duration.

```
/**  
 * Sample use of relative duration calculation.  
 *  
 */  
  
gs.include('DurationCalculator');  
executeSample();  
  
/**  
 * Function to house the sample script.  
 */  
function executeSample(){  
  
    // First we need a DurationCalculator object. We will  
    also use  
    // the out-of-box relative duration "2 bus days by 4pm  
    "  
    var dc =new DurationCalculator();  
    var relDur ="3bf802c20a0a0b52008e2859cd8abcf2";  
    // 2 bus days by 4pm if before 10am  
    addSchedule(dc);  
  
    // Since our start date is before 10:00am our result i  
    s two days from  
    // now at 4:00pm.  
    dc.setStartTime("5/1/2012 09:00:00");  
    if(!dc.calcRelativeDuration(relDur)){  
        gs.log("*** calcRelativeDuration failed");  
        return;  
    }  
    gs.log("Two days later 4:00pm: "+ dc.getEndTime())  
;  
  
    // Since our start date is after 10:00am our result i  
    s three days from  
    // now at 4:00pm.  
    dc.setStartTime("5/1/2012 11:00:00");  
    if(!dc.calcRelativeDuration(relDur)){  
        gs.log("*** calcRelativeDuration failed");  
        return;  
    }  
    gs.log("Three days later 4:00pm: "+ dc.getEndTime()  
});}  
  
/**
```

```
* Add a specific schedule to the DurationCalculator object.  
*  
* @param durationCalculator An instance of DurationCalculator  
*/  
function addSchedule(durationCalculator){  
    // Load the "8-5 weekdays excluding holidays" schedule  
    // into our duration calculator.  
    var scheduleName ="8-5 weekdays excluding holidays";  
    var grSched =new GlideRecord('cmn_schedule');  
    grSched.addQuery('name', scheduleName);  
    grSched.query();  
    if(!grSched.next()){  
        gs.log('*** Could not find schedule "'+ scheduleName +'");  
        return;  
    }  
    durationCalculator.setSchedule(grSched.getUniqueValue(),  
    "GMT"); }
```

How to implement a relative duration

You can implement a relative duration by creating the cmn_relative_duration table and the DurationCalculator script include.

Before you begin

Role required: admin

Procedure

1. Create the cmn_relative_duration table.
2. Create the DurationCalculator script include.
3. Create a sample relative duration entry (for example, "Next business day by 4pm").
4. Add the needed fields to SLA tables to support relative durations.
5. Modify duration calculation for SLAs.

6. Modify SLA Percentage timer calculation for SLAs (this must use work_seconds).
7. Add schedule fields to the Workflow: Schedule and Timezone (selected based on the field from workflow table).
8. Add duration support fields to the Workflow Task activity.
9. Implement duration calculation script for the task activity.

The relative duration table and the DurationCalculator methods

The cmn_relative_duration table supports the definition of a due date as either a duration of time or a relative duration.

This table consists of two fields: "name" and "script." The "script" field contains the relative duration calculation script. This script includes the "calculator" variable, which is used to calculate the due date.

The DurationCalculator script include can be used to perform the duration calculations. The following are methods that are available in this script include.

DurationCalculator script include table

Method	Description
setSchedule(String schedID, [String timezone])	Sets the schedule and time zone to be used for calculating the due date.
setStartTime(GlideDateTime start)	Sets the start time for the duration calculations. If 'start' is blank, uses current date/time.
calcDuration(int seconds)	Calculates the end date and time. Upon completion the this.endDateTime and this.seconds properties will be set to indicate the results of the calculation.
calcRelativeDuration(String relativeDurationID)	Calculates the duration using the specified relative duration script. Upon completion the

Method	Description
	this.endDateTime and this.seconds properties will be set to indicate the results of the calculation.
getEndDateTime()	Gets the this.endDateTime property that was set by calcDuration/calcRelativeDuration indicating the end date and time for the duration.
getSeconds()	Gets the this.seconds property that was set by calcDuration/ calcRelativeDuration indicating the total number of seconds of work to be performed for the duration. Note: This is the total work time, not the total time between start and end times and may be used to determine percentages of the work time
getTotalSeconds()	Gets the this.totalSeconds property that was set by calcDuration/ calcRelativeDuration indicating the total number of seconds between the start and end times of the duration.

The following functions are used in relative duration scripts:

Relative duration script functions

Function	Description
boolean isAfter(GlideDateTime dt, String time)	Is 'time' of day after the time of day specified by 'dt'? dt, if blank,

Function	Description
	uses current date/time. time is in "hh:mm:ss" in 24-hour format.
calcRelativeDueDate(GlideDateTime start, int days, String endTime)	Calculates the due date starting at 'start' and adding 'days' using the schedule and time zone. When we find the day that the work is due on, set the time to 'endTime' of that day. Upon completion, this.endDate and this.seconds properties will be set to indicate the results of the calculation. If endTime is blank, use end of the ending work day.

Creating custom UI Pages and UI macros

Use UI pages to create custom pages for an application and UI macros for custom controls or interfaces.

Every UI Page is a [Jelly](#) template. Jelly turns XML into executable code. A UI Page works similar to how an index.html file is used in an AngularJS application. Jelly tags in the HTML field of the UI Page form contain AngularJS logic.

Creating [UI Macros](#) requires knowledge of Jelly script. Review the existing UI macros for examples and suggested approaches. Those who want to build custom interfaces with JavaScript technologies should consider Service Portal as an alternative.

- [UI pages](#)

UI pages can be used to create and display forms, dialogs, lists and other UI components.

- [UI Macros](#)

UI macros are discrete scripted components administrators can add to the user interface.

- [Jelly tags](#)

Use Jelly to turn XML into HTML.

UI pages

UI pages can be used to create and display forms, dialogs, lists and other UI components.

Use UI pages as widgets on dashboards. To find the UI pages, navigate to **System UI > UI Pages**.

This functionality requires a knowledge of HTML or Jelly. You can also create simple AngularJS applications using UI pages. To learn more, see the [Building Apps with AngularJS developer training](#).

The UI page form provides the following fields:

UI page

Field	Input Value
Name	Name used to invoke the page via a URL (must not contain spaces).
Direct	Select this check box for a direct UI page [sys_ui_page]. A direct UI page doesn't include the common HTML, CSS, and scripts. This setting requires adding custom CSS and JavaScript to use in the page.
HTML	Main component of the page, and it defines what will be rendered when the page is shown. It can contain either static XHTML or dynamically generated content defined as Jelly, and it can call script includes and UI Macros.
Client Script	Client-side JavaScript that runs in the browser (e.g. functions called by buttons, etc.). It is intended to handle any client-side processing needed, for example setting focus to a field, or other interactive

Field	Input Value
	DHTML features after a page is loaded. Ultimately, a UI page's Client Scripts are deployed to the browser within a <script/> tag, so it could be defined within the page's HTML field to achieve the same effect. Using the Client Script field instead to define these scripts makes things much more tidy and readable though, and it keeps the Jelly and HTML from becoming unmanageable.
Processing Script	Script that runs on the server when the page is submitted. This is useful if your page has a form (defined with the <g:ui_form/> or <g:form/> tags).
obsolete-custom-processors	Note: This feature is deprecated. While legacy, custom processors will continue to be supported, creating new custom processors has been deprecated. Instead, please use the Scripted REST APIs . The following information is left in the documentation for existing processors only.
Related lists on the form view:	
Versions	Shows all versions of the UI page. Use this list to compare versions or to revert to a previous version.

A UI page can be secured by creating an ACL with the following parameters:

- **Type:** ui_page
- **Operation:** read
- **Name:** name of the UI page to be protected

For details on creating an ACL rule, see [Create an ACL rule](#).

UI page access

Each UI page has a URL computed from the application scope, page name, and the .do file extension.

For example, to display the page called glidewindow_example on the demo system, you would navigate to `https://<instance name>.service-now.com/glidewindow_example.do`. If the page was part of a custom application called example_app, you would instead navigate to `https://<instance name>.service-now.com/x_example_app_glidewindow_example.do`.

You can also add additional parameters to a URL that can be accessed within a page's HTML section as jelly variables. That is, appending arguments to the URL as follows: `/my_test_page.do?sysparm_verbose=true` creates jelly variables called verbose that can be accessed as follows:

```
<j2:if test="$![empty(sysparm_verbose)]"> <span>show extra stuff </span> </j2:if >
```

A common practical example of this might be retrieving a database record for display. To build a list of a user's roles, pass in a parameter with the user's sys_id. Invoke the following UI page to display a list of roles for that user with Jelly code:

```
role_select.do?  
sysparm_user=5137153cc611227c000bbd1bd8cd2007  
  
<j:set var = "jvar_user_id" value = "${sysparm_user}" />  
  
    <g:evaluate> var userRoles = new GlideRecord('sys_user_has_role');  
        userRoles.addQuery('user', '${jvar_user_id}');  
        userRoles.query();  
    </g:evaluate>
```

```
<select id='select_role'>
    <j:while test = "${userRoles.next()}">
        <option value = "${userRoles.sys_id}"> ${userRoles.role.name} </option>
    </j:while>
</select>
```

An exception to be careful of, though, is the reserved variable name `sys_id`. This variable always contains the ID of the UI page itself, regardless of what is specified in the URL. A common substitute variable name is `sysparm_id`.

Do not use URL parameters to load client scripts in UI pages. The system no longer evaluates scripts that are passed by URL parameter. If your implementation depends on this behavior, you can [add the system property](#) [glide.security.disable_ui_pages_sysparm_client_script] and set it to **false** to temporarily allow the evaluation of URL parameters passing scripts in UI pages.

UI page process scripts

If your UI page contains a form (uses the `<g:form>` tag), you can submit the form and have the process 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="49
9836460a0a0b1700003e7ad950b5da" />
</g:ui_form>
```

You can access the field using `application_sys_id`:

```
var application = new GlideRecord('hr_application');
application.get(application_sys_id);
application.status = "Rejected";
application.update();
var urlOnStack = GlideSession.get().getStack().bottom();
response.sendRedirect(urlOnStack);
```

Important: The preceding script is usable only with Global applications.

If you are using the UI page for a dialog, you can also reference the most recent URL on the stack using the code above and then send the response to that location. This is useful if you want to have the dialog's processing script update something and then redisplay the screen that brought up the dialog.

UI Macros

UI macros are discrete scripted components administrators can add to the user interface.

UI macros are typically controls that provide inputs or information not provided by existing field types. By default, the system provides UI macros for a variety of user interface elements such as:

- All formatters
- The Service Catalog cart
- The action icons next to fields
- The action icons on forms and lists
- The widgets of a content management system
- The Orchestration activity designer

Administrators can create their own UI macros to provide custom controls or interfaces. Creating UI macros requires knowledge of [Jelly script](#). Review the existing UI macros for examples and suggested approaches. Those who want to build custom interfaces with JavaScript technologies should consider Service Portal as an alternative.

Calling UI macros

Administrators can call UI macros from certain record types associated with the user interface.

Calling UI Macros by record type

Record type	Example
Dictionary attribute	Display an icon for a reference field: <pre>ref_contributions=ui_macro_name</pre>
UI page	Display something on a UI page: <pre><g:macro_invoke macro="ui_macro_name" /></pre>
UI macro	Call a UI macro from another UI macro: <pre><?xml version="1.0" encoding="utf-8"?> <j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null"> <g:ui_macro_name /> <g:ui_macro_name_2 /> </j:jelly></pre>

UI macro form

Each UI macro record consists of a name and an XML document written in Jelly code.

UI macro fields

Field	Description
Name	A unique and descriptive name for this macro.
Active	Select the check box to render the element as defined. Clear the check box to disable the element without deleting the code. For example, the email_reply macro is inactive by default.
Description	Describe the purpose of the macro and parameters passed to it.
XML	Jelly script that defines the macro.

Jelly tags

Use Jelly to turn XML into HTML.

Watch these introductory to learn about using Jelly in the Now Platform.

- [Introducing Jelly Scripting - Part 1 \(Video\)](#)
- [Introducing Jelly Scripting - Part 2 \(Video\)](#)
- [Introducing Jelly Scripting - Part 3 \(Video\)](#)

Jelly Tags

if

- **Description:** The `if` tag is just what it looks like, an `if` tag. This is like an `if` statement in any programming language, but keep in mind that there is no `elseif` tag and no `else` tag. If you want to create that kind of structure, try the `choose/when/otherwise` syntax.
- **Parameters:** `test` - The condition to evaluate in order to determine if the block will execute.

- Example:

```
<g:evaluate var="jvar_now_GR" object="true">
    var now_GR = new GlideRecord("incident");
    now_GR.addQuery("active", true);
    now_GR.query();
    now_GR;
</g:evaluate>

<j:if test="${!jvar_now_GR.hasNext()}">
    We did not find any active incidents.
</j:if>
<j:if test="${jvar_now_GR.next()}">
    We found ${jvar_now_GR.getRowCount()} active incidents
.
</j:if>
```

while

- Description: The `while` tag does a while loop.
- Parameters: `test` - The condition to evaluate in order to determine if the statement will loop through. This should be an expression enclosed in `${}` or `$[]` that evaluates to true or false.
- Example:

```
<g:evaluate var="jvar_now_GR" object="true">
    var now_GR = new GlideRecord("incident");
    now_GR.addQuery("active", true);
    now_GR.query();
    now_GR;
</g:evaluate>

<j:while test="${jvar_now_GR.hasNext()}">
    <a href="incident.do?sys_id=${jvar_now_GR.getValue('sys
_id')}">${jvar_now_GR.getValue('number')}
```

set

- Description: The `set` tag sets a variable.
- Parameters:

- **var** - The variable to set. Often the system prefixes these variables with `jvar_` for consistency.
- **value** - The value to set `var` to. This is often an expression enclosed in `${}` or `$[]`.
- **defaultValue** - If the value results to null or empty, this value is put into the `var`.
- Example:

```
<j:set var="jvar_incident_number" value="${jvar_now_GR.getValue('number')}"/>
```

set_if

- Description: The `set_if` tag sets a variable based on a test. This tag is similar to the `ternary` operator in other programming languages (`var = <test> ? <if_true> : <if_false>`).
- Parameters:
 - **var** - The variable to set. Often the system prefixes these variables with `jvar_` for consistency.
 - **test** - The condition to evaluate in order to determine if the statement will evaluate the true value or the false value. This should be an expression enclosed in `${}` or `$[]` that evaluates to true or false.
 - **true** - The value to set the variable to if `test` evaluates to true. This parameter is optional, so if the field is blank, and if `test` evaluates to true, the variable is left blank.
 - **false** - The value to set the variable to if `test` evaluates to false. This parameter is optional, so if the field is blank, and if `test` evaluates to false, the variable will be left blank.

choose

- Description: The `choose` tag starts a choose block of code. This is similar to the `if-elseif-else` kind of syntax in most programming languages. With a `choose` tag, you can use `when` and `otherwise` tags to specify other blocks of code.

- Parameters: None

- Example:

```
<j:choose>
    <j:when test="${jvar_now_GR.getRowCount() &lt; 1}>
        We found multiple records!
    </j:when>
    <j:when test="${jvar_now_GR.next()}>
        We found record ${jvar_now_GR.getValue('number')}
    </j:when>
    <j:otherwise>
        Sorry, we could not find the record you specified.
    </j:otherwise>
</j:choose>
```

when

- Description: The `when` tag is used within a `choose` block to indicate a condition. This tag is similar to an `if` or an `elseif` in that it specifies a condition, executes the inner content, and then implies a break at the end to leave the `if-elseif` construct.
- Parameters: `test` - The condition to evaluate in order to determine if the statement will loop through. This should be an expression enclosed in `${}` or `$[]` that evaluates to true or false.

- Example:

```
<j:choose>
    <j:when test="${jvar_now_GR.getRowCount() &lt; 1}>
        We found multiple records!
    </j:when>
    <j:when test="${jvar_now_GR.next()}>
        We found record ${jvar_now_GR.getValue('number')}
    </j:when>
    <j:otherwise>
        Sorry, we could not find the record you specified.
    </j:otherwise>
</j:choose>
```

otherwise

- Description: The `otherwise` tag is used within a `choose/when/otherwise` block, and is like the `else` or `default` case.
- Parameters: None

- Example:

```
<j:choose>
    <j:when test="${jvar_now_GR.getRowCount() < 1}">
        We found multiple records!
    <j:when test="${jvar_now_GR.next()}>
        We found record ${jvar_now_GR.getValue('number')}
    <j:otherwise>
        Sorry, we could not find the record you specified.
    </j:otherwise>
</j:choose>
```

Glide Tags

evaluate

- Description: The `evaluate` tag evaluates JavaScript code (server side), and makes variables visible to future code. Unlike other tags, the `evaluate` tag evaluates the content that is inside the tag as server side JavaScript.

The context is the same as that of `script` includes in the system. Other script includes, global business rules, `GlideRecord`, `GlideSystem`, and Jelly variables (prefixed with `jelly`. if the parameter `jelly="true"` is set) are available.

- Parameters:
 - `var` - The name of the variable that will be set to the result of the script.
 - `object` - If set to `true`, the result of the expression is treated as an object instead of a primitive variable (string or integer variable values).
 - `jelly` - If set to `true`, allows Jelly context variables to be referenced in the script.
 - `expression` - This is an expression to be executed for the value to put in `var`. The expression can be either of two places. First, it can be an attribute on the `evaluate` tag itself. Otherwise, the content between the beginning tag and ending tag is the expression. The last line of the expression is the actual value passed into `var`.
- Example:

```
<g:evaluate var="jvar_now_GR" object="true">
    var now_GR = new GlideRecord("incident");
    now_GR.addQuery("active", "true");
    now_GR.query();
    now_GR; // this is the variable put into the variable j
var_now_GR
</g:evaluate>
```

```
<g:evaluate var="jvar_now_GR" object="true" expression="">
    var now_GR = new GlideRecord('incident');
    now_GR.addQuery('active', 'true');
    now_GR.query();
    now_GR; // this is the variable put into the variable j
var_now_GR" />
```

messages

- Description: The `messages` tag helps with translation. When `gs.getMessage()` is called anywhere on a page, there are two possible places where the translation is found. First, the page checks a local cache of translations. Second, the page makes an AJAX call to the server to find the translation. What `g:messages` does is allow pages to cache certain messages.
- Parameters: None
- Example:

```
<g:messages>
    Yes
    No
    Cancel
</g:messages>
```

breakpoint

- Description: When the `breakpoint` tag is called, it prints a list of all the variables in Jelly at the current moment, with their respective values. If a variable is specified, it prints the requested variable and its value. The output is placed in the system log.
- Parameters: `var` - (Optional) The variable to log the value for. If `var` is not specified, then all variables are dumped into the log.

- Example:

```
<g:breakpoint />  
<g:breakpoint var="sysparm_view"/>
```

no_escape

- Description: The system, by default, uses escaped output as a security measure. Output placed inside of no_escape tags is not escaped before output. Be careful when using these tags, because if user input is displayed here it can open a security vulnerability on the page.
- Parameters: None
- Example (phase 1) – Disables automatic output escaping of all contained \${} expressions:

```
<g:no_escape>  
  ${jvar_raw_html_data}  
</g:no_escape>
```

- Example (phase 2) – Use NOESC to disable escaping for the specified string. This implies the expression must evaluate to a string.

```
<g:no_escape>$ [NOESC:jvar_expr]</g:no_escape>
```

For information on phase 1 and phase 2 evaluation, refer to the Jelly introduction videos listed at the beginning of this section.

macro_invoke

- Description: The macro_invoke tag calls a UI macro that you have specified in the database. You may also call a UI macro by specifying it in the tag name. For example, if you had a UI macro named my_macro, you could call that macro with the tag <g:my_macro/>.
- Parameters:
 - macro - The name of the UI macro to execute. If your tag name is g:macro_invoke, then the macro attribute specifies the name of the macro. If the tag name includes the name of the macro, then there is no need to include a macro attribute.

- Other attributes - For each attribute you specify, a variable with that name will be available in the context of the UI macro, prefixed with `jvar_`.
- Example:

```
<!-- Will invoke the contents of the UI macro named "sample_macro", which will have the variable jvar_message available within it-->
<g:macro_invoke macro="sample_macro" message="This is a sample macro variable." />
```

```
<!-- Will invoke the contents of the UI macro named "sample_macro", which will have the variable jvar_message available within it-->
<g:sample_macro message="This is a sample macro variable." />
```

if_polaris

- Description: The `if_polaris` tag checks if Next Experience is enabled for the current page. It must include at least one of the child tags `<g:then />` or `<g:else />`.

- Parameters: None

- Example:

```
<g:if_polaris>
    <g:then><g:inline template="polaris_nav"/></g:then>
    <g:else><g:inline template="classic_nav"/></g:else>
</g:if_polaris>

<g:if_polaris>
    <g:then><a href="...">Click here to see what's new!</a>
</g:then>
</g:if_polaris>

<g:if_polaris>
    <g:else>Core UI only code here!</g:else>
</g:if_polaris>
```

then

- Description: The `then` tag is used within an `if_polaris` block to set the page content when Next Experience is enabled.
- Parameters: None
- Example:

```
<g:if_polaris>
    <g:then><g:inline template="polaris_nav"/></g:then>
    <g:else><g:inline template="classic_nav"/></g:else>
</g:if_polaris>

<g:if_polaris>
    <g:then><a href="...">Click here to see what's new!</a>
</g:then>
</g:if_polaris>

<g:if_polaris>
    <g:else>Core UI only code here!</g:else>
</g:if_polaris>
```

else

- Description: The `else` tag is used within an `if_polaris` block to set the page content when Next Experience isn't enabled.
- Parameters: None
- Example:

```
<g:if_polaris>
    <g:then><g:inline template="polaris_nav"/></g:then>
    <g:else><g:inline template="classic_nav"/></g:else>
</g:if_polaris>

<g:if_polaris>
    <g:then><a href="...">Click here to see what's new!</a>
</g:then>
</g:if_polaris>

<g:if_polaris>
    <g:else>Core UI only code here!</g:else>
</g:if_polaris>
```

- [Jelly escaping types](#)

You use different methods when escaping characters in JavaScript and HTML. JavaScript uses the backslash character, and HTML uses the ampersand character.

- [Extensions to Jelly syntax](#)

Apache's Jelly syntax is used to render forms, lists, UI pages, and many other things rendered in ServiceNow.

Jelly escaping types

You use different methods when escaping characters in JavaScript and HTML. JavaScript uses the backslash character, and HTML uses the ampersand character.

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).

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)

Type	From	To
	NL (newline)	\n ('\ followed by 'n')
HTML	& (ampersand)	&
	< (less than)	<
	> (greater than)	>

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

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

Add escaping to a Jelly replacement

You can handle character escaping in Jelly files. XML escaping behavior can be modified only by users with the `security_admin` role.

About this task

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).

Procedure

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}
```

Extensions to Jelly syntax

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.

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

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

Namespaces

Jelly often includes multiple namespaces when invoking tags.

The "j" namespaces are standard Jelly whereas the "g" namespaces are unique to ServiceNow scripts. 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

You can use if statements in Jelly scripts.

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 statement 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 statement works is that the JEXL empty function returns true if its argument is:

1. null
2. an empty string
3. a zero length collection

4. a map with no keys
5. an empty array

Note: You cannot mix javascript and jvar variables in a JEXL expression. They must be broken into separate expressions.

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>

This page provides a comparative explanation of three tags: `<g:insert>`, `<g:inline>`, and `<g:call>`.

<g:insert>

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.

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

<g:inline>

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: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:

```

```

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="syspa
rm_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.opCurrent()"/>
```

object="true"

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

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

jelly="true"

If you would like to access Jelly variables inside an evaluate, include jelly="true" in the evaluate 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_fixEscaping" 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: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");
```

You can 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

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 &&. For example:

```
<j:if test="${jvar_form_name == 'sys_form_template' & &
!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.

```
<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 (), 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="t
ext-decoration: none; color: white">
        ${gs.getMessage("Loading...") }
    </span>
</span>
```

Tracing Jelly

ServiceNow has a feature that 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:

```
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**.

Debugging scripts

Debug scripts using session logs and Now Platform debugging tools such as a walk-through script debugger and error messages that display in the UI.

Debugging server-side scripts

Use the Script Debugger and session logs to debug server-side code. For more information, see [Script Debugger and Session Log](#).

You can also use session debug to display error messages related to a server-side script that runs as a result of a client-side change. For more information, see [Session debug](#).

GSLog is a script include that simplifies script logging and debugging by implementing levels of log output, selectable by per-caller identified sys_properties values. For more information, see [GSLog API](#).

Debugging client-side scripts

Use session debug to display debugging messages in the user interface. For more information, see [Session debug](#). Use the session log to view logging information for script includes and custom UIs, such as Agent Workspace.

You can also debug client-side scripts using browser-based developers tools.

Debugging applications and scopes

Use the application debugging options to understand how a script's application scope might affect your application, table, or record. You may need to update cross-scope privileges to troubleshoot scope access issues. See [Debugging applications](#).

- [Script Debugger and Session Log](#)

The Script Debugger enables users with the `script_debugger` role to debug server-side JavaScript, while the Session Log enables you to view and download required logs.

- [Session debug](#)

Enable session debugging to display debugging messages in the user interface.

- [Debugging applications](#)

Application developers can display debug messages about configuration records to help them troubleshoot issues. The Debug Scopes module provides information about the system switching between custom applications to run server-side scripts.

- [Debugging business rules](#)

Debugging business rules can be achieved with resources available in the ServiceNow product.

- [Debugging classifications](#)

You must add a system property to enable classification debugging.

- [Field watcher](#)

The field watcher tool tracks and displays all actions that the system performs on a selected form field.

- [Writing to the debug log](#)

To write to the debug log in your client-side JavaScript, or UI policies, make a call to the global function `jslog()`.

- [JavaScript debug window](#)

The JavaScript debug window appears in a bottom pane of the user interface when an administrator turns on debugging.

- [JS Code Coverage Debug](#)

The JS Code Coverage Debug application allows administrators and application developers to log the scripts triggered during a user session and then review which lines of code the system ran.

Script Debugger and Session Log

The Script Debugger enables users with the `script_debugger` role to debug server-side JavaScript, while the Session Log enables you to view and download required logs.

Users with the `script_debugger` role can perform these actions using Script Debugger:

- Have a dedicated debug transaction, which applies only to the current session.
- Set and remove breakpoints.
- Pause the current session at a breakpoint.
- Evaluate expressions during runtime.
- Step through code line-by-line.
- Step into and out of function and method calls.
- View the value of local and global variables.
- View the value of private variables from function closures.
- View the call stack.
- View the transaction that the system is processing.
- Turn off the script debugger to resume running paused scripts.

Use the Session Log tab to retrieve the session log for business rules, script includes, and a custom UI such as ServiceNow® Agent Workspace that has a GraphQL component. Users with the `script_debugger` role can:

- View session logs in a separate tab.
- Download a log.
- View logs for Agent Workspace.
- Specify debug options to view or download only the required logs.

By default, 100 transactions and 10,000 messages appear on the Session Log tab. If the transaction or message count exceeds the default value, the session log is cleared and the next transactions or messages appear. You can configure this transaction and message count using the `glide.debugger.log.transaction.count` and `glide.debugger.log_messages_limit` user preferences respectively. For more information about the `glide.debugger.log.transaction.count` and `glide.debugger.log_messages_limit` user preferences, see [User preference settings](#).

Note: Enable Session Log as a separate tab with Script Debugger using the `glide.debugger.log.ui` system property.

- The **Page** option displays logs under forms and lists and on the **Session Log** tab.
- The **Session** option displays logs only on the **Session Log** tab.

For more information about the `glide.debugger.log.ui` system property, see [Available system properties](#).

When you execute a statement in the Console, the executed statement is stored in the browser cache. You can use the up arrow key to get the previous statement and down arrow key to get the next statement from the browser cache. The user preference setting, `glide.debugger.console.cached_stmt_limit`, defines the number of statements cached in a browser session. The default statement cache value is 20 and the maximum value is 100. You can configure the statement cache value from user preferences.

Note: The cached statements are not available when the browser cache is cleared or when you log in from a different browser or a different computer.

The Script Debugger can pause any server-side script that runs in an interactive transaction such as business rules, script includes, script actions, or UI actions that require a response to proceed. If the `GlideSystem` method `isInteractive()` returns **True** when running the script in context, then the Script Debugger can pause it.

Note: Some script objects, such as script includes, can be called from multiple contexts. For example:

- when a business rule runs a script include on a form submit that is an interactive transaction waiting on the form data to change before continuing.
- when a scheduled job runs the same script include that is a non-interactive background transaction that can also run other scripts simultaneously.

To debug client-side scripts, you can use browser-based developers tools.

A debugger transaction remains open as long as the user session is valid. If a user logs out or their session times out, the system closes the debugger transaction.

To view debug logs, see [Display debugging logs](#).

Note: When the Script Debugger is enabled, code is executed in interpreted mode. If parts of the script are set to run in strict mode, the debugger is not able to find the correct objects and the debugger fails. The Script Debugger must run on scripts outside of strict mode.

- [Script Tracer and debugging scripts](#)

The Script Tracer can help you filter your debugging search to quickly narrow down script problems. You can identify lines of scripts in the Glide record that have undergone change during execution. Finding those specific lines of scripts rather than doing a wide search helps save time and improves productivity.

- [Parts of the user interface](#)

The Script Debugger interface displays information about breakpoints set, the call stack and line number of the currently executing script line, details about variables and transactions, and status of console.

- [Script Debugger step-through and console controls](#)

After the Script Debugger pauses a script, use the step-through controls to move between script lines and move between scripts in the call stack. Use the Console controls to expand console, collapse console, clear console, and rerun expressions.

- [Evaluate expressions in runtime using Console](#)

Define, declare, and verify new variables and functions while you debug a script in runtime using Console. The script execution must be paused in order to use Console.

- [Launch the Script Debugger](#)

Developers can launch the Script Debugger from the application navigator, Studio, or from the syntax editor.

- [Set or remove breakpoints](#)

Set breakpoints or conditional breakpoints to pause scripts at specific lines, and remove breakpoints when you are done debugging them.

- [Set or remove logpoints](#)

Set breakpoints or conditional logpoints to log messages to the console at specific lines, and remove logpoints when you are done debugging them.

- [Script Debugger status](#)

The Script Debugger status determines what debugging actions are available and what information it can display.

- [Transaction details](#)

The Script Debugger displays transaction details for the current paused user session.

- [Available transaction details](#)

The Script Debugger provides a standard set of transaction details for developers to debug and troubleshoot scripts.

- [Script Debugger multiple developer support](#)

The Script Debugger allows multiple developers to debug their own transactions without affecting each other.

- [Script Debugger impersonation support](#)

You can use the Script Debugger while impersonating another user, but only if the impersonated user has the script_debugger role and has read access to the target script.

- [Script Debugger Scripts - Background support](#)

The Scripts - Background module does not support setting breakpoints directly in the script field. You can however, set breakpoints in the script objects called or triggered by the Scripts - Background module.

- [Domain separation and Script Debugger](#)

Domain separation is supported in Script Debugger. Domain separation enables you to separate data, processes, and administrative tasks into logical groupings called domains. You can control several aspects of this separation, including which users can see and access data.

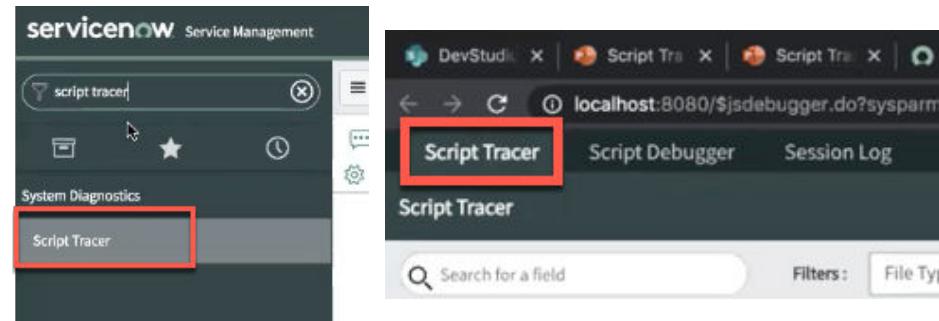
Script Tracer and debugging scripts

The Script Tracer can help you filter your debugging search to quickly narrow down script problems. You can identify lines of scripts in the Glide record that have undergone change during execution. Finding those specific lines of scripts rather than doing a wide search helps save time and improves productivity.

Overview

Use the Script Tracer to narrow your search so you can debug scripts and business rules more efficiently. You can find the Script Tracer by searching in the left navigation pane.

Note: To use the Script Tracer, your role must be admin.



Once you enable Script Tracer and execute a UI transaction, the Tracer searches through all the scripts being executed. The following filters are available:

- **File type:** Search for a specific file type
- **Table:** Look in the specific table for the script being executed

The Script Tracer searches for changes in the script during execution and presents them in a list for you to examine. When you click **Start Tracer**, the Tracer begins searching for changes in the Glide record. You can click the **Debug Script** button at any time to see the script itself.

A screenshot of the Script Tracer results page. At the top, there are tabs for 'State', 'Script', and 'Transaction', with 'State' highlighted by a red box. To the right of the tabs are 'Debug Script' and 'View File' buttons, both highlighted with red boxes. The main content area displays a list of script changes for a record named 'percentage'. The list includes fields like Social: 87, Maths: 96, Updated: 2020-10-15 15:26:58, Percentage: 0 => 84, Updated by: jithamanyu.manne@snc, Science: 67, Evaluated: true, Grade: A, Sys ID: 7cdf7d361b3f1010265f986b234bcb8d, Next Level:, Created: 2020-10-15 15:26:58, Name: John, Updates: 0, Total: 252, Choice Of Field:, Created by: jithamanyu.manne@snc, and Tags:.

Use the tabs to see specific information from the Tracer.

The **State** tab displays the differences between the old and new scripts.

- By default, the **Show only changed values** check box is enabled, so you can avoid fields that have not changed.
- To view all the fields (changed or not), you can clear that check box.

Note: If the file is not reflected in the trace statement, it means the changes in the Glide record is not recognized by the system.

If there are any errors, they display at the top of the State tab, with their line numbers and error message displayed in order of occurrence.

File Name	File Type	Table	Line no.	Run Point Scan UI Action
1 Set System Flag	Business Rule	User Preference	1	
2 Run Point Scan	UI Action		1	
3 Run Point Scan	UI Action		1	Error: "sn_health_scan" is not defined.

- **Script:** Displays the line of changed scripts that the Glide record has undergone during execution. You can view the entire line of script by clicking the **Show Script** button.
- **Transaction:** Shows all transaction records of the trace
- **Debug Script:** Opens the script in Debugger to debug the script
- **View File:** Opens the script in the ServiceNow platform for editing
- **Clear trace:** Clears the trace when you are finished.

Limiting the tracer

You may want to set a limit for your trace so that you don't generate too many returns. By default there will be up to 1,000 lines of script traced. Once this number is reached you must clear the trace and start tracing again. If you want to change the maximum number of lines for tracing you can configure your limit using the property `glide.debug.trace_line_limit`.

Since each trace you run is new, make sure you're finished reading the results of one trace before clearing it and beginning another one.

To learn more, see [Debugging scripts](#).

Parts of the user interface

The Script Debugger interface displays information about breakpoints set, the call stack and line number of the currently executing script line, details about variables and transactions, and status of console.

Script Debugger Not Paused

```

Business Rule > GTD Tour Name and Page Name Validation
Script type and name

Breakpoints
Variables
Console pane
Status bar

```

```

Business Rule > GTD Tour Name and Page Name Validation
Script type and name

Breakpoints
Variables
Console pane
Status bar

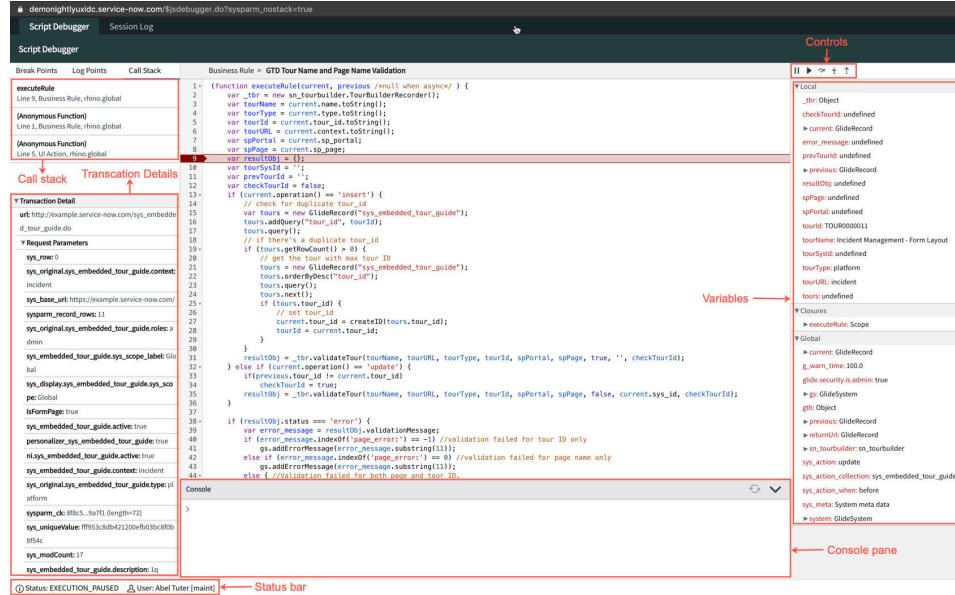
```

```

1+ (function executeRule(current, previous /*null when async*/ ) {
2+   var tourName = current.name.toDateString();
3+   var tourType = current.type.toDateString();
4+   var tourURL = current.url.toDateString();
5+   var tourURL = current.context.toDateString();
6+   var spPortal = current.sp_portal;
7+   var spPage = current.sp_page;
8+   var sysId = current.sys_id;
9+   var resultObj = {};
10+   var tourSysId = '';
11+   var tourId = '';
12+   var checkTourId = false;
13+   if (current.operation) { "insert" } {
14+     // check if there's a duplicate tour_id
15+     var tours = new GlideRecord("sys_embedded_tour_guide");
16+     tours.query("tour_id", tourId);
17+     tours.query();
18+     // if there's a duplicate tour_id
19+     if (tours.hasNext()) {
20+       // get the tour with max tour ID
21+       tours = new GlideRecord("sys_embedded_tour_guide");
22+       tours.query("tour_id");
23+       tours.query();
24+       tours.next();
25+       if (tours.tour_id) {
26+         // set tour_id
27+         current.tour_id = createID(tours.tour_id);
28+         tourId = current.tour_id;
29+       }
30+     }
31+     resultObj = _trv.validateTour(tourName, tourURL, tourType, tourId, spPortal, spPage, true, "", checkTourId);
32+   } else if (current.operation) { "update" } {
33+     if (current.tour_id != current.tour_id)
34+       checkTourId = true;
35+     resultObj = _trv.validateTour(tourName, tourURL, tourType, tourId, spPortal, spPage, false, current.sys_id, checkTourId);
36+   }
37+   if (resultObj.status == "error") {
38+     var error_message = resultObj.validationMessage;
39+     if (error_message.indexOf("page_error") == -1) //validation failed for tour ID only
40+       gs.addErrorMessage(error_message.substring(1));
41+     else if (error_message.indexOf("tour_error") == -1) //validation failed for page name only
42+       gs.addErrorMessage(error_message.substring(1));
43+     else //Validation failed for both page and tour ID
44+   }

```

Script Debugger Paused



Parts of the Script Debugger

User interface element	Description
Breakpoints	Displays a list of breakpoints set by script type, script name, and line number. The debugger updates this list as you add and remove breakpoints.
Call stack	Displays a list of script calls that preceded or invoked the current line number. This information is only visible when the debugger pauses on a breakpoint.
Transaction details	Displays information about the current transaction. This information is only visible when the debugger pauses on a breakpoint.
Status	Displays if the debugger is waiting for a breakpoint, paused on a

User interface element	Description
	breakpoint, or has encountered an exception.
User	Displays the name of the user who is running the current debugger session.
Coding pane header	Displays the script type and name of the script in the coding pane.
Breakpoint icon	Indicates the line number where the debugger pauses when evaluating the current script.
Pause debugging button	Stops any current debugging session, and disables the Script Debugger for the current user. The Script Debugger doesn't pause on breakpoints for the current user until it's restarted.
Console	Displays a command line interface used for evaluating expressions during runtime. The console is available only when the script execution is paused.
Resume script execution button	Advances from the current breakpoint to the next breakpoint. If there are no other breakpoints, the script runs to completion.
Step over next function call button	Advances past the method that's about to be called, executing the method as a single step.
Step into next function call	Advances to the first line of executed code within a method call. Stepping into a method updates the current position within the call stack. If the user doesn't

User interface element	Description
	have read access to the method call, then this control acts like step over instead.
Step out of current function	Exits from current method call and returns to the calling script from the call stack. If the user isn't within a method call, then this control acts like step over instead.
Local	Displays a list of local scope JavaScript variable names and their values. This information is only visible when the debugger pauses on a breakpoint.
Closures	Displays a list of global scope JavaScript variable names and their values set by function closure. This information is only visible when the debugger pauses on a breakpoint.
Global	Displays a list of global scope JavaScript variable names and their values. This information is only visible when the debugger pauses on a breakpoint.

Session Log

Session Log

Session log options: Clear log, Download Log, Settings

Filters: Debug Output (2), Apps (1), Message Type (1)

Transactions

Transaction ID	Number of messages	Messages
32805 \$uxpage	0	32805 \$uxpage.do
32806 /api/now/graphql	2	06:21:34.284 SchemaBuilder, total time for building schemas 2 milliseconds 85 Schema stats: Schema Prefix: GlideInteraction_ Top Level Operations: [Query, Mutation] Schema Size: 6
32807 /api/now/graphql	1	06:21:34.286 SchemaBuilder, total time for building schemas 1 milliseconds
32812 /api/now/graphql	2	06:21:34.286 Schema stats: Schema Prefix: GlideUIAction_ Top Level Operations: [Query] Schema Size: 39
32813 /api/now/graphql	1	06:21:34.292 SchemaBuilder, total time for building schemas 4 milliseconds
32814 /api/now/graphql	2	06:21:34.294 Schema stats: Schema Prefix: GlideLayout_ Top Level Operations: [Query] Schema Size: 263
32815 /api/now/graphql	1	06:21:34.295 SchemaBuilder, total time for building schemas 1 milliseconds
32818 /api/now/graphql	2	06:21:34.296 Schema stats: Schema Prefix: GlideAggregate_ Top Level Operations: [Query] Schema Size: 22
32819 /api/now/graphql	2	06:21:34.306 SchemaBuilder, total time for building schemas 5 milliseconds 06:21:34.306 Schema stats: Schema Prefix: GlideContextualSearch_ Top Level Operations: [Query] Schema Size: 259
		06:21:34.307 SchemaBuilder, total time for building schemas 0 milliseconds
		06:21:34.308 SchemaBuilder, total time for building schemas 0 milliseconds
		06:21:34.309 Schema stats: Schema Prefix: GlideConnect_ Top Level Operations: [Query, Mutation] Schema Size: 72
		06:21:34.309 SchemaBuilder, total time for building schemas 0 milliseconds
		06:21:34.310 Schema stats: Schema Prefix: GlideDataLookupQuery_ Top Level Operations: [Query] Schema Size: 8
		06:21:34.313 SchemaBuilder, total time for building schemas 2 milliseconds

Session Log user interface elements

User interface element	Description
Transactions	Transaction ID. Displays information about the current transaction.
Filter for log text	Field to enter text to filter the logs that contain a specific text.
Debug Output	Option to filter logs based on the dynamically loaded debug output types. For example, Security Rule .
Apps	Option to filter logs based on the dynamically loaded apps. For example, Service Management Integrations .

User interface element	Description
Message Type	Option to filter logs based on the dynamically log levels. For example, Info .
Clear log	Clears all logs.
Download log	Download the logs in HTML file format.
Settings	Session debug options. For information about the debug options, see Session debug .

Script Debugger step-through and console controls

After the Script Debugger pauses a script, use the step-through controls to move between script lines and move between scripts in the call stack. Use the Console controls to expand console, collapse console, clear console, and rerun expressions.

Step-through controls

Control	Icon	Description
Stop debugging SHIFT+F2		Stops any current debugging session, and disables the Script Debugger for the current user. The Script Debugger does not pause on breakpoints for the current user until it is restarted.
Start debugger - F2	⊕	Enables the Script Debugger for the current user. The Script Debugger pauses on breakpoints.

Control	Icon	Description
Resume script execution - F9	▶	Advances from the current breakpoint to the next breakpoint. If there are no other breakpoints, the script runs to completion.
Step over next function call - OPTION+F9	↷	Advances to the next evaluated line of script based on current conditions. The Script Debugger skips any lines of code that do not need to run because their conditions are not met. For example, when the condition of an <code>if</code> statement is not true, the debugger skips the code block for the condition.
Step into next function call - OPTION+F10	↓	When the Script Debugger pauses on a method call, this control allows the user to advance to the first line of executed code within the method call. Stepping into a method updates the current position within the call stack. If the user does not have read access to the method call, then this control acts like step over instead.

Control	Icon	Description
Step out of current function - OPTION+F11		When the Script Debugger pauses within a method call, this control allows the user to exit the current method call and return to the calling script from the call stack. If the user is not within a method call, then this control acts like step over instead.

Console controls

Control	Icon	Description
Open Console		Expands the Console.
Close Console		Collapses the Console.
Clear expressions		Clears all the expressions in the Console.
Re-execute expression		Re-executes the expression which is already executed.

Related tasks

- [Set or remove breakpoints](#)

Evaluate expressions in runtime using Console

Define, declare, and verify new variables and functions while you debug a script in runtime using Console. The script execution must be paused in order to use Console.

Before you begin

- Review [Limitations with using Console](#)
- Role required: script_debugger, admin

Procedure

1. Launch Script Debugger in one of the following ways:

Application	Navigation path
Application navigator	Navigate to System Diagnostics > Script Debugger .
Studio	Navigate to File > Launch Script Debugger .
Syntax Editor	Click the Script Debugger icon  .

The Script Debugger modal is displayed.

2. Trigger the script.

For example, create a record to trigger an insert business rule script. The Script Debugger pauses the script on the first line that contains a breakpoint, and then you see the ServiceNow Script Debugger confirmation window.



3. Click **Start Debugging**.

The focus shifts to the Script Debugger window and you see the target script that paused at the first breakpoint.

Note: Make sure that the status of Script Debugger is EXECUTION_PAUSED. You can use Console only when the script execution is paused during debugging.

4. Click the Open Console icon to expand the Console pane. To start evaluating expressions, enter one or more expressions in the Console and press Enter. For example, enter `var x = 10;` and press Enter. To enter multiple lines of expressions, press Shift + Enter after each line and press Enter after the last expression. To clear all the expressions in the Console, click the clear console icon . For more information on Console controls, see [Script Debugger step-through](#) and [console controls](#).

```

Business Rule = GTD Tour Name and Page Name Validation
Call Stack
1+ (function executeRule(current, previousResult, when, async) {
2+   var _tbr = new sys_tourbuilder.TourBuilderRecorder();
3+   var tourName = current.name.toString();
4+   var tourId = current.id.toString();
5+   var tour = current.tour;
6+   var tourType = current.tour_type.toString();
7+   var spPortal = current.sp.portal;
8+   var spPage = current.sp.page;
9+
10+   var tourTypeId = '';
11+   var previousTourId = '';
12+   var tourCreated = false;
13+   if (current.operation == 'insert') {
14+     var tours = new GlideRecord("sys_embedded_tour_guide");
15+     var tourId = new GlideRecord("sys_tour");
16+     tour.addQuery("tour_id", tourId);
17+     tour.query();
18+     // If there's a duplicate tour_id
19+     if (tourId.next() != -1) {
20+       // get the tour with max tour ID
21+       tours = new GlideRecord("sys_embedded_tour_guide");
22+       tours.addQuery("tour_id", tourId);
23+       tours.query();
24+       if (tourId.next() != -1) {
25+         // set tour_id
26+         current.tour_id = createID(tours, tourId);
27+         tourId = current.tour_id;
28+       }
29+     }
30+   }
31+   resultObj = _tbr.validateTour(tourName, tourId, tourType, tourId, spPage, true, '', checkTourId);
32+   else if (current.operation == 'update') {
33+     if (previousTourId != current.tour_id) {
34+       checkTourId = true;
35+     }
36+   }
37+   if (resultObj.status === 'error') {
38+     var error_message = resultObj.validationMessages;
39+     if (error_message.indexOf('page_name') == -1) //validation failed for tour ID only
40+       gs.addErrorMessage(error_message.substring(1));
41+     else if (error_message.indexOf('tour_id') == -1) //validation failed for page name only
42+       gs.addErrorMessage(error_message.substring(1));
43+     else //Validation failed for both page and tour ID.
44+   }
45+
Console
> var x = 10;
< undefined
> x
< 10
> 10
< 10
0f54c

```

Status: EXECUTION_PAUSED User: Abel Tuter (main)

After a statement is executed, it is stored in the browser cache. You can use the up arrow key to get the previous statement and down arrow key to get the next statement from the browser cache. You can configure the number of cached statements for a session from user preferences. For more information about user preferences settings, see [Script Debugger and Session Log](#).

- [Limitations with using Console](#)

You need to be aware of a few limitations when you use Console to evaluate expressions while debugging a script in runtime.

Related reference

- [Limitations with using Console](#)

You need to be aware of a few limitations when you use Console to evaluate expressions while debugging a script in runtime.

- The properties and values of an object don't display in Console. When you try to display an object to Console, only the string value of the object appears.
- Console doesn't support GlideSystem printing methods, such as info() and print().

- You can't use the this keyword in Console.
- A script debugger timeout occurs when you evaluate expressions in Console.
- While executing long scripts, if you see the response Awaiting response from server, you can't resume debugging or stop debugging using the resume or stop controls.

Launch the Script Debugger

Developers can launch the Script Debugger from the application navigator, Studio, or from the syntax editor.

Before you begin

Role required:

- admin
- script_debugger

Procedure

Select a path based on your starting point:

Starting point	Navigation path
Application navigator	Navigate to System Diagnostics > Script Debugger .
Studio	Navigate to File > Launch Script Debugger .
Syntax Editor	Click the Script Debugger icon.

The system opens the Script Debugger in a new window.

Set or remove breakpoints

Set breakpoints or conditional breakpoints to pause scripts at specific lines, and remove breakpoints when you are done debugging them.

Before you begin

Role required:

- admin
- script_debugger

About this task

Breakpoints belong to the developer who sets them. Developers must set and remove their own breakpoints.

Note: At a specific line, you can set either a logpoint or breakpoint but not both.

Procedure

1. Navigate to the server script to debug. For example, navigate to **All > System Definition > Business Rules**.
2. From the Syntax Editor, click the gutter next to a script line.

Action	Description
Set a breakpoint	Click a blank line to set a breakpoint.
Set a conditional breakpoint	Right-click a blank line and click Add conditional breakpoint to set a conditional breakpoint.
Remove a breakpoint	Click a breakpoint to remove it.
Remove a conditional breakpoint	Right-click a conditional breakpoint and select Remove breakpoint to remove it.

3. From the Syntax Editor toolbar, click the **Open Script Debugger** icon. The system opens a Script Debugger window.
4. Trigger the script. For example, create a record to trigger an insert business rule script.

The Script Debugger pauses the script on the first line containing a breakpoint, and the system displays a confirmation window.



5. Click **Start Debugging**.

The system switches focus to the Script Debugger window and displays the target script paused at the first breakpoint. Console pane is enabled.

6. When debugging is complete, remove breakpoints from the script.

Related reference

- [Script Debugger step-through and console controls](#)

Set or remove logpoints

Set breakpoints or conditional logpoints to log messages to the console at specific lines, and remove logpoints when you are done debugging them.

Before you begin

- Set the `glide.debug.log_point` system property to `true`. See [Available system properties](#) for more information.
- Role required: admin or script_debugger

About this task

Logpoints belong to the developer who sets them. Developers must set and remove their own logpoints.

Note: At a specific line, you can set either a logpoint or breakpoint but not both.

Procedure

1. Navigate to the server script to debug. For example, navigate to **All > System Definition > Business Rules**.
2. From the Syntax Editor, click the gutter next to a script line.

Action	Description
Set a logpoint	Right-click a blank line and click Add logpoint to set a logpoint.
Remove a logpoint	Right-click a logpoint and select Remove logpoint to remove it.
Edit a logpoint	Right-click a logpoint and select Edit logpoint to edit it.

Note: The script entered for the logpoint must have the same format as that of the script in GSLog and GSInfo script includes.

3. From the Syntax Editor toolbar, click the **Open Script Debugger** icon .

4. On the Script Debugger window, trigger the script. For example, create a record to trigger an insert business rule script.

Note: You can also add logpoints in the Script Debugger window.

5. In the Script Debugger window, click **Session Log** to view the logpoints.
6. When debugging is complete, remove logpoints from the script.

Script Debugger status

The Script Debugger status determines what debugging actions are available and what information it can display.

The Script Debugger displays its status at the bottom left of the user interface.

Sample Script Debugger status

 Status: EXECUTION_PAUSED

Possible Script Debugger status values

Status	Occurs when	Description	Actions available
WAITING_FOR_FIRST_BREAKPOINT	The user opens a Script Debugger window or tab.	The Script Debugger is ready to pause script and display debugging information.	Pause script at the first breakpoint in the call stack.
EXECUTION_PAUSED	<ul style="list-style-type: none">The Script Debugger pauses on a breakpoint.The user steps over, steps into, or steps out to the next line of evaluated code.	The Script Debugger has paused on a line of code, and the user can debug the script. Console is enabled.	<ul style="list-style-type: none">Resume processing until the Script Debugger reaches the next breakpoint.Step through a script.Display the call stack.Display transaction information.

Status	Occurs when	Description	Actions available
			<ul style="list-style-type: none"> Display variable values. Evaluate expressions in Console during runtime.
WAITING_FOR_BREAKPOINT	<ul style="list-style-type: none"> The user resumes processing until the Script Debugger reaches the next breakpoint. The user steps through a script until the Script Debugger reaches the next line of code to evaluate or the transaction completes. 	<p>The Script Debugger is searching for the next line of code at which to pause. Users will typically never see this status because the Script Debugger changes status after it locates the next breakpoint or script line to evaluate.</p>	<ul style="list-style-type: none"> Pause script at the next breakpoint. Pause script at the next script line requiring evaluation.
OFF	<ul style="list-style-type: none"> The user pauses the Script Debugger. The user closes the Script 	<p>The Script Debugger is inactive and does not pause scripts or display debugging information.</p>	<ul style="list-style-type: none"> Start the Script Debugger. Open a Script Debugger window or tab.

Status	Occurs when	Description	Actions available
	<p>Debugger window or tab.</p> <ul style="list-style-type: none">The user session ends for any reason.The administrator resets all Script Debugger instances by navigating to the debugger_res et.do page.		

Log entries

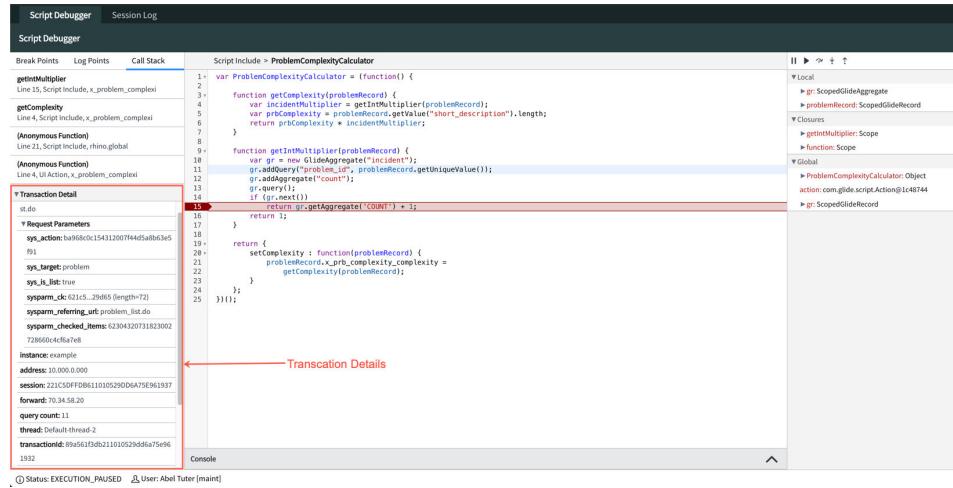
Every time a debug transaction finishes executing, the system creates a log entry for it with a DEBUGGED prefix. For example:

```
2016-08-15 15:57:32 (197) Default-thread-3 900F51016711220
0C4098C7942415A75 *** End
#39, path: /my-app.do, user: admin, DEBUGGED total transaction time: 0:00:11.010,
transaction processing time: 0:00:11.010, network: 0:00:00
.000, chars: 6,058, uncompressed
chars: 20,731, SQL time: 50 (count: 34), business rule: 0
(count: 0), phase 1
form length 56,464, largest chunk written: 10,428, request parms size: 40, largest input read: 0
```

Transaction details

The Script Debugger displays transaction details for the current paused user session.

Transaction details are available in a dedicated resizable section underneath the Call Stack on the bottom left of the Script Debugger.



The Script Debugger only displays transaction details when it pauses on a script. Developers can use transaction details to:

- Inspect the URL of the currently paused transaction.
- Inspect the request parameters for the currently paused transaction.
- Inspect network information about the current transaction.
- Inspect the user and session ID that initiated the debug transaction.

Related reference

- [Available transaction details](#)

Available transaction details

The Script Debugger provides a standard set of transaction details for developers to debug and troubleshoot scripts.

Available transaction details

Transaction element	Description
url	The URL of the currently paused transaction.
Request parameters	The list of request parameters for this transaction. Each transaction has its own list of request parameters, but record transactions typically include the field values used to insert, update, or delete a record.
instance	The instance name.
address	The IP address of the end-user client system.
session	The user session ID.
forward	The IP address of the load balancer.
query count	The number of database queries the Script Debugger has made.
thread	The name of the thread running the Script Debugger instance.
transactionid	The Sys ID of the current transaction.
token	The Script Debugger token of the currently paused transaction. The system uses this token to identify different Script Debugger instances.

Transaction element	Description
name	The name of the currently paused transaction. You can use this name to identify transactions in the logs.
processor	The name of the processor processing the current transaction, if present.
method	The HTTP request method the currently paused transaction uses.
startTime	The date-time stamp when the Script Debugger instance started.
page	The current table or UI page associated with the transaction.
user	The user who triggered the debug transaction.
nodeid	The Sys ID of the node running the Script Debugger instance.

Related concepts

- [Transaction details](#)

Script Debugger multiple developer support

The Script Debugger allows multiple developers to debug their own transactions without affecting each other.

The Script Debugger only allows developers to see and interact with items related to their current debugging session such as:

- Breakpoints
- Call stack
- Console

- Transactions
- Status

The Script Debugger prevents one developer from seeing or modifying another debug session. Administrators, however, can impersonate another user, open the Script Debugger, and debug transactions generated by the impersonated user.

The Script Debugger displays the debug session user at the bottom left of the user interface.

Sample Script Debugger user

 User: System Administrator

Concurrent Script Debugger usage

By default, the system supports debugging [(The number of semaphores on the instance) / 4] concurrent transactions. Administrators can specify the number of concurrent transactions the system can debug by setting the `glide.debugger.config.max_node_concurrency` system property. The system can debug up to [(The number of semaphores on the instance) - 2] concurrent transactions.

Administration of debugging sessions

Debugging sessions can remain actively debugging (in the EXECUTION_PAUSED or WAITING_FOR_BREAKPOINT statuses) until:

- The user pauses the Script Debugger.
- The user closes the Script Debugger.
- The user session ends.

Administrators can view the currently running debugger sessions by navigating to the page `xmlstats.do`.

Administrators can stop all currently running debugging sessions by navigating to the page `debugger_reset.do`. Only users with the admin role can access this page.

Related concepts

- [Script Debugger impersonation support](#)

Script Debugger impersonation support

You can use the Script Debugger while impersonating another user, but only if the impersonated user has the script_debugger role and has read access to the target script.

While impersonating another user, you can:

- See and change breakpoints that belong to the impersonated user.
- View and pause on scripts that the impersonated user has read access to.
- Evaluate expressions in Console on behalf of the impersonated user.

The Script Debugger step-through controls also use the read access of the impersonated user. For example, if the impersonated user does not have read access to a function in the call stack, any **Step into** action instead becomes a **Step over** action.

The impersonated debugging session lasts until:

- You stop impersonating the user.
- You log out or the user session ends.
- You pause the Script Debugger.
- You close the Script Debugger.

Related concepts

- [Script Debugger multiple developer support](#)

Script Debugger Scripts - Background support

The Scripts - Background module does not support setting breakpoints directly in the script field. You can however, set breakpoints in the script objects called or triggered by the Scripts - Background module.

While running arbitrary JavaScript code in the **Scripts - Background** module, the Script Debugger can only pause scripts when you:

- Call a script include containing breakpoints.
- Trigger a business rule containing breakpoints.
- Trigger a script action containing breakpoints.

Domain separation and Script Debugger

Domain separation is supported in Script Debugger. Domain separation enables you to separate data, processes, and administrative tasks into logical groupings called domains. You can control several aspects of this separation, including which users can see and access data.

Support level: Basic

- Business logic: Ensure that data goes into the proper domain for the application's service provider use cases.
- The application supports domain separation at run time. The domain separation includes separation from the user interface, cache keys, reporting, rollups, and aggregations.
- The owner of the instance must set up the application to function across multiple tenants.

Sample use case: When a service provider (SP) uses chat to respond to a tenant-customer's message, the customer must be able to see the SP's response.

For more information on support levels, see [Application support for domain separation](#).

How domain separation works in Script Debugger

Script Debugger is not a full application but rather, a feature in the Platform suite, meaning it works alongside other features, including domain separation.

Session debug

Enable session debugging to display debugging messages in the user interface.

You can enable all areas for abundant logging on the bottom of each page load, or you can enable each module one by one, for more specific information about what is happening during this session, and specifically, for the previous transaction. Select session debug options under **System Diagnostics > Session Debug**. When enabled, session debugging is active during the user session or until disabled. To view debug logs, see [Display debugging logs](#).

The system provides the following session debugging options.

Session debug options

Debug option	Description
Enable All	Displays all available debugging messages.
Disable All	Stops displaying all debugging messages.
Debug Business Rule	Displays debugging messages for business rules. If there are business rules from multiple applications affecting a table or record, the system displays which application the business rule comes from.
Debug Upgrade	Displays detailed information logged for records processed during the last family-to-family or patch version upgrade session. See Debug Upgrade .
Debug Business Rule (Details)	Displays debugging messages for business rules and any changes made by business rules. If there are business rules from multiple applications affecting a table

Debug option	Description
	or record, the system displays which application the business rule comes from.
Debug Log	Displays all log entries.
Debug NLQ	Displays debugging messages for Natural Language Query (NLQ) queries.
Debug Date/Time	Displays Date/Time failures when inputs do not match required formats.
Debug SQL	Displays debugging messages for SQL queries.
Debug SQL (Detailed)	Displays debugging messages for SQL statements and any changes made by SQL statements.
Debug Security	Displays debugging messages for access controls. If there are access controls from multiple applications affecting a table or record, the system displays which application the access controls comes from.
Debug Escalations	Displays debugging messages for SLA and SLO escalations.
Debug AI Search	Displays debugging messages for AI Search.
Debug Metric Statistics	Displays an aggregate view of performance data (slow transactions, scripts, queries, events, and mutexes). These aggregate metrics are sorted by transaction, to help identify items that affect page performance.

Debug option	Description
Debug Text Search	Displays debugging messages for search result relevance and indexing.
Debug UI Policies	Displays debugging messages for UI policies.
Disable UI Policies Debug	Stops displaying debugging messages for UI policies.
Debug UI Macro	Displays the start and end of the UI Macro in the DOM as HTML comments. The comments consist of table name and UI macro name.
Disable Debug UI Macro	Stops displaying the start and end of the UI Macro in the DOM as HTML comments.
Debug Data Policies	Displays debugging messages for data policies.
Debug Quotas	Displays debugging messages for transaction quotas.
Debug Homepage Render	Displays debugging messages for homepages.
Debug Scopes	Displays debugging messages for entering or exiting application scopes when running script objects.

Display debugging logs

Display session debug logs to help diagnose script and application problems.

Before you begin

Role required: none

Procedure

1. Navigate to **All > Session Debug** and select **Enable All**.
2. Under **Session Debug**, select **Debug Log**.
The Debug log displays.

Debugging applications

Application developers can display debug messages about configuration records to help them troubleshoot issues. The Debug Scopes module provides information about the system switching between custom applications to run server-side scripts.

The system offers the following debugging options to help application developers determine how applications affect configuration records.

Application debug options

Debugging option	Description
Debug Business Rule	Use this module to determine which application's business rules are running against tables. The system only displays application information if business rules from different application scopes run on the same table.
Debug Business Rule (Details)	Use this module to determine the results of running business rules against tables. The system only displays application information if business rules from different application scopes run on the same table.
Debug Security	Use this module to determine which application's access

Debugging option	Description
	controls apply to a given table or record.
Debug Scopes	Use this module to determine the application scope context in which a script runs. Since one script can call another script it is possible to have multiple application scope context changes while running a series of scripts.
Enable Session Debug	Use this related link to enable the generation of log messages for a particular application. Application scripts that use GlideSystem logging methods will generate output to the log at the indicated verbosity level.

When multiple applications contribute to the debug output, the system adds a new section called **Apps** to the display a list of the applications writing to the session log. Clicking on the check box next to the application name hides or displays the application's associated debug messages.

Sample application debug output of business rules

The screenshot shows the ServiceNow Incident view for incident INC001.0011. The top navigation bar includes links for Create Security Incident, Resolve Incident, and Delete. Below the header, there are two sections: "Debug Output" and "Apps". Under "Debug Output", checkboxes are selected for "Business Rules" and "Others". Under "Apps", checkboxes are selected for "Global", "AppOne", and "Security Incident". The main content area displays a list of debug log entries:

```

15:03:25.150: Execute before query business rules on sys_attachment;
15:03:25.158: App:Security Incident === Skipping 'Prevent non-security roles reading' on sys_attachment; condition not satisfied: Condition: !new sn_sir_core.SecurityIncident().canAccessSecureRecords();
15:03:25.158: Finished executing before query business rules on sys_attachment;
15:03:25.160: Execute before query business rules on sys_attachment;
15:03:25.170: App:Security Incident === Skipping 'Prevent non-security roles reading' on sys_attachment; condition not satisfied: Condition: !new sn_sir_core.SecurityIncident().canAccessSecureRecords();
15:03:25.170: Finished executing before query business rules on sys_attachment;
15:03:27.564: >>> Preceding lines from previous transaction

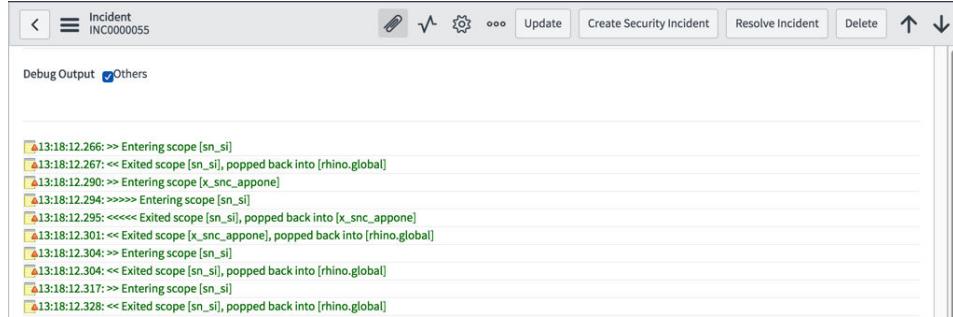
```

Debugging scopes

Application developers can use the **Debug Scopes** module to display information about when the system switches between custom applications to run server-side scripts.

When enabled, the system displays a message whenever the system switches to a custom application to run a server-side script.

Sample debug scopes output from the incident table



The screenshot shows a ServiceNow interface with the title bar "Incident" and ID "INC0000055". Below the title bar is a toolbar with icons for edit, search, update, create security incident, resolve incident, and delete. To the right of the toolbar are sorting arrows. The main area is titled "Debug Output" with a checkbox for "Others" which is checked. The content area contains several lines of log messages:

```
13:18:12.266: >> Entering scope [sn_si]
13:18:12.267: << Exited scope [sn_si], popped back into [rhino.global]
13:18:12.290: >> Entering scope [x_snc_appone]
13:18:12.294: >>>> Entering scope [sn_si]
13:18:12.295: <<<< Exited scope [sn_si], popped back into [x_snc_appone]
13:18:12.301: << Exited scope [x_snc_appone], popped back into [rhino.global]
13:18:12.304: >> Entering scope [sn_si]
13:18:12.304: << Exited scope [sn_si], popped back into [rhino.global]
13:18:12.317: >> Entering scope [sn_si]
13:18:12.328: << Exited scope [sn_si], popped back into [rhino.global]
```

Every time the system runs a server-side script object it enters the script's scope context. When the script finishes running, the script exits the scope context. The debugging messages track changes to the script scope context.

The debugging message displays a greater than character > each time the system enters a script object's context, and displays a less than character < every time the system exits a script object's context. In cases where one script calls another the debugging message adds another greater than character to the path for each call. For example, if a business rule calls a script include, which in turn calls another script object there would three characters in the path such as:

```
> Entering scope [x_app_one]
>> Entering scope [x_app_two]
>>> Entering scope [x_app_three]
```

Note: The system does not display entering or exiting messages for script objects in the global scope.

Application developers may want to enable other debugging options to in conjunction with this option to see information about the possible source of the server-side script such as Debug Business Rule.

Debugging business rules

Debugging business rules can be achieved with resources available in the ServiceNow product.

1. Tools

The first step in the process is to identify tools which will help you figure out what's wrong.

Debugging tools

Debugging tool	Description
System Dictionary	Navigate to System Definition > Dictionary . The dictionary provides a list of all tables within your instance and can be invaluable when trying to locate information.
System Log	Navigate to System Logs > System Log . You can place alert statements in your business rule which can write information to the log.
Debug Business Rule (Details)	Navigate to System Diagnostics > Session Debug > Debug Business Rule (Details) . This debugging module displays the results business rules. Use this module to see if conditions are being met and values are being set as expected.
Alert Messages	There are several system functions that allow you to print messages to the page, the field or the log file.

Debugging tool	Description
	See Scripting alert, info, and error messages .
Business Rule Examples	Sometimes you can find what you're looking for in scripts others have written, including business rule error messages, or by building an OR query.
GlideRecord Information	This is the basic syntax used to query the database for information. See Querying tables in script . GlideRecord also includes aggregation support.

2. Variables

The next step is to gain some insight into the behavior of your business rule. For every action except an insert, you will more than likely use a query to get your record(s).

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

To verify whether your query is actually returning records you can use `gs.addInfoMessage` to display information at the top of the screen.

```
var rec = new GlideRecord('incident');
rec.addQuery('active',true);
rec.query();
gs.addInfoMessage("This is rec.next: " + rec.next());
while (rec.next()) {
  gs.print(rec.number + ' exists');
}
```

If your query returns no records you see the following:

```
This is rec.next: false
```

Use this technique to verify every variable within your business rule contains expected values.

Tip: If necessary, break your script down into individual pieces and verify each piece works separate from the whole and then put them all back together one step at a time.

3. Locating information

The last step is to make sure you know where to find the information your rule is looking for.

In the ServiceNow application, one table can extend another table. This means when searching for information, you might need to query the parent table for the extended table's sys_id to find what you seek.

A good example is the sc_task table, which extends the task table. The script below queries the extended table (sc_task) for the current sys_id and then query the parent table (task) for records with the matching sys_id, and then prints out the work notes field.

```
var kids = new GlideRecord('sc_task');
kids.query();

gs.addInfoMessage("This is requested item number: " + current.number);
gs.print("This is the requested item number: " + current.number);

while (kids.next()) {
    var parents = new GlideRecord('task');
    parents.addQuery('sys_id', '=', kids.sys_id);
    parents.query();

    while(parents.next()) {
        gs.addInfoMessage("This is task number: " + parents.number);
        gs.print("This is task number: " + parents.number);
        gs.addInfoMessage("These are the work notes: " + parents.work_notes);
```

```
    gs.print("These are the work notes: " + parents.work_notes);
}
}
```

Debugging classifications

You must add a system property to enable classification debugging.

Debugging classifications

The resulting log entries list the name of each classifier that runs, along with all the names and values that are available to the criteria in the classifier. To log debugging information about classifications, add the following system property.

System Property	Description
glide.discovery.debug.ci_identification	<p>Enables debugging information for process classification.</p> <ul style="list-style-type: none">Type: true falseDefault Value: falseLocation: Add to the System Properties [sys_properties] table

Field watcher

The field watcher tool tracks and displays all actions that the system performs on a selected form field.

Note: Field watcher is not supported with Next Experience in Tokyo. For more information about supported features in Next Experience, see [Considerations for activating Next Experience](#).

Administrators can use the field watcher to figure out what happens to the field and how the value of the field changes when an event such

as the firing of a business rule or enforcement of a data policy, takes place. Administrators can also impersonate non-admin users to debug what happens when those users make changes on an instance. Only one field can be watched at a time. Non-admin users with the impersonator role have access to the field watcher feature.

How the field watcher works

The Field Watcher tool logs activity when any of the following events occur on a field:

- The default value is set on the field.
- User access rights for the field change due to an ACL or dictionary setting.
- A data policy prevents the value from being set.
- A reference qualifier query of the field value executes.
- A UI policy changes a field to or from read-only, visible, mandatory, or editable.
- A dependent value in another field restricts field choices.
- The value of the field is set or changed based on:
 - Assignment rules
 - Actions from an engine, such as the workflow engine
 - Business rules
 - User entries
 - Client scripts
 - UI actions

Note: The field watcher works only on form fields. It cannot be used on list fields. Also, field watcher is not available on password-protected fields or encrypted fields. Field watcher is only available within the UI frame. The option to watch a field does not appear in the context menu if you open a record outside of the UI frame, for example, in a new tab.

Use field watcher

Access field-level debugging information using the field watcher.

Before you begin

Role required: none

Procedure

1. Navigate to the form for which you want to view field-level debugging information.
2. Activate field watcher by right-clicking any field label on a form and select **Watch - <field name>**.

The debug icon () appears next to the field label. From this point on, the field watcher records every action taken on the selected field. For example, if you are watching a Priority field, if the priority is changed from Moderate to Low and the record is updated, the field watcher will display information about that change.

3. View the field watcher log by clicking the debug icon.
A new pane opens at the bottom of the screen, showing a field watcher tab. It may also show tabs for [JavaScript Logging](#) and [JavaScript Debugger](#).
4. Click the **Field Watcher** tab, if needed.
5. Stop watching a field by right-clicking the field and selecting **Unwatch - <field name>**. To watch another field, right-click that field and select **Watch - <field name>**.
6. Clear the field watcher log by clicking the clear log button ().
7. Resize the field watcher pane by dragging the splitter bar up or down. Dragging the splitter bar to the bottom of the screen closes the field watcher pane. Reopen the pane by clicking the debug icon again.

Field watcher tab details

The field watcher displays field information and configuration options.

The left-side of the Field Watcher tab shows the following information for the watched field.

- **Table:** table to which the field belongs.
- **Element:** field label.
- **Type:** type of data stored in the field.
- **Dependent:** field on which the current field depends.
- **Reference:** table from which the field's value originates, if applicable.
- **Reference Qual:** reference qualifiers that may be restricting data on the field.
- **Attributes:** attributes on the field as specified in the dictionary entry for that field.

On the right-side of the Field Watcher tab, select the types of activity information you want to see for the selected field. Clear the check box for any type of information that is not needed.

Watching a hidden field

Administrators may need to watch a hidden field.

1. Use the dictionary to determine the column name of the field.
2. Elevate privileges to the security_admin role.
3. Navigate to **System Definition > Scripts Background**.
4. In Run script (JavaScript executed on server), enter the following command:

```
gs.getSession ( ) . setWatchField ( "hidden_field" ) ;
```

Replace hidden_field with the column name of the hidden field.

5. Navigate to the form containing the missing field.

The Field Watcher tab output contains information about the hidden field.

Viewing information for the watched field

When information for a watched field is changed and the record is updated, the field watcher tab displays relevant information at the bottom.

Field watcher viewing data

Table: Incident	Reference:	<input checked="" type="checkbox"/> All	<input checked="" type="checkbox"/> Business rule	<input checked="" type="checkbox"/> Client script
Element: State	Reference Qual:	<input checked="" type="checkbox"/> ACL	<input checked="" type="checkbox"/> Data policy	<input checked="" type="checkbox"/> UI policy
Type: integer	Attributes:	<input checked="" type="checkbox"/> Data lookup	<input checked="" type="checkbox"/> Workflow activity	<input checked="" type="checkbox"/> Reference qualifer
Dependent:		<input checked="" type="checkbox"/> UI action		

ⓘ 12:30:32 (939) REQUEST ACTION - [Save](#)	Value received from client is: 2	
ⓘ 12:30:32 (945) ACL - record/incident.state/write	true	
ⓘ 12:30:33 (117) BUSINESS RULE - [Run SLAs](#)	Active → New	
ⓘ 12:30:33 (118) BUSINESS RULE - [Run SLAs](#)	Active → New	
ⓘ 12:30:33 (118) BUSINESS RULE - [Run SLAs](#)	New → Active	
Timestamp	**Type of item that changed and associated name**	**Old and new values**

Field watcher information includes:

- **Timestamp:** time the field was changed using the HH:MM:SS (ms) format.
- **Orange text:** server-side changes, such as ACLs.
- **Blue text:** client-side changes, such as client scripts.
- **Type of object that changed the field and its associated name:** The type of item that changed on the field; for example, **CLIENT SCRIPT**, **BUSINESS RULE**, or **ACL**. In the case of scripts, business rules, or other configuration-type fields, field watcher displays the name of the script or business rule that changed the field, if any. Click the name to go directly to the record for that item.
- **Old and new values:** The old and new values for the field, if the value changed. Field watcher does not record the value if it was inserted in the form by default at the time the record was created.
- **Additional information:** Call tracing information, such as the name of the script engine or workflow that changed the field. Click the plus icon to expand the selection.
 - **Orange text:** Indicates server-side activity.
 - **Blue text:** Indicates client-side activity.

Example: Watching the incident priority

The following example shows what happens to the **Priority** field on the incident form when both the **Impact** and **Urgency** fields change.

The Incident form has two client-side data lookups change the priority. Additionally, server-side ACLs and the data lookup engine fire when the record is saved. Finally, a client-side UI policy sets the **Priority** field back to read-only, which is the default setting.

Watching the incident priority

Original values
<ul style="list-style-type: none">Priority:1 - CriticalImpact:1 - HighUrgency:1 - High
First Change
<ol style="list-style-type: none">1. The user changes the Impact value to 3 - Low.2. The priority automatically changes to 3 - Moderate based on the Priority Lookup data lookup definition used by default in ServiceNow incidents.
Note:
At this point, the record has not been saved.
Second Change
<ol style="list-style-type: none">1. The user changes the Urgency value to 2 - Medium.2. The priority automatically changes to 4 - Low based on the same Priority Lookup data lookup definition.

Original values

3. The user saves the record by right-clicking the form header and choosing **Save**.

Field watcher example

Table:	Incident	<input checked="" type="checkbox"/> All	<input checked="" type="checkbox"/> ACL	<input checked="" type="checkbox"/> Business rule	<input checked="" type="checkbox"/> Client script	
Element:	Priority	<input checked="" type="checkbox"/> Data lookup	<input checked="" type="checkbox"/> Data policy	<input checked="" type="checkbox"/> UI policy	<input checked="" type="checkbox"/> UI action	
Type:	integer	<input checked="" type="checkbox"/> Workflow activity	<input checked="" type="checkbox"/> Reference qualifier			
Dependent:	Attributes:					
					1 → 3	
(i)	14:26:06 (062) DATA LOOKUP - onchange of incident.impact					
(i)	14:26:26 (458) DATA LOOKUP - onchange of incident.urgency					
					3 → 4	
(i)	13:26:38 (590) REQUEST ACTION - Save	Value received from client is: 4				
(i)	13:26:38 (598) UI ACTION - Save					
(i)	13:26:38 (603) ACL - record/incident.priority/write	true				
(i)	13:26:38 (603) ACL - record/incident.priority/create	true				
(i)	13:26:38 (659) SCRIPT ENGINE - com.glide.data.lookup.DataLookupScriptEngine	4 - Low → 4 - Low				
(i)	13:26:38 (805) ACL - record/incident.priority/read	true				
(i)	13:26:38 (805) ACL - record/incident.priority/write	true				
(i)	14:26:39 (284) UI POLICY - Priority is managed by Data Lookup - set as read-only	ReadOnly set to true				
(i)	14:26:39 (285) UI POLICY - Priority is managed by Data Lookup - set as read-only	Setting disabled to true				

Note: The values that change from 1 to 3, and then from 3 to 4, refer to the numerical values in the choice list.

Writing to the debug log

To write to the debug log in your client-side JavaScript, or UI policies, make a call to the global function `jslog()`.

An example of using `jslog()` in JavaScript:

```
function logData (r) {  
    lastLogDate = r.responseXML.documentElement.getAttribute ("last_log_entry") ; var items = r.responseXML.getElementsByName ("log") ;  
    jslog ("response=" + r.responseText) ; }
```

Additionally, when client scripts run, the name of the client script and timing information is displayed. This can be useful in determining which scripts are running and whether they are impacting performance.

Debug UI policies

Enabling the `glide.ui.ui_policy_debug` property lets you monitor the processing of UI actions.

Here are some sample log events from an incident policy that sets fields to read-only if the incident_state is closed.

```
GlideFieldPolicy: Evaluating condition
GlideFieldPolicy:      incident_state (7) = 7 -> true
GlideFieldPolicy: --->>> TRUE
GlideFieldPolicy:      Setting opened_at disabled to true
GlideFieldPolicy:      Setting opened_by disabled to true
GlideFieldPolicy:      Setting closed_at disabled to true
GlideFieldPolicy:      Setting closed_by disabled to true
GlideFieldPolicy:      Setting company disabled to true
```

Access the 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. Users with the admin role can access this log.

Before you begin

Role required: admin

About this task

The steps to access the JavaScript debug window depend on which UI version you are using.

Note: The JavaScript debug window is not supported with Next Experience in Tokyo. For more information about supported features in Next Experience, see [Considerations for activating Next Experience](#).

Procedure

1. Open the JavaScript log by navigating to the appropriate location for your version of the UI.

Core UI

a. Click the gear icon in the banner frame.

b. Click the **Developer** section.

c. Toggle the **JavaScript Log and Field Watcher** switch.

UI15

- a. Click the gear icon in the banner frame.
 - b. Click **JavaScript Log and Field Watcher**.
-

UI11

Click the debug icon () in the banner frame.

A new pane opens at the bottom of the screen. It shows the JavaScript Log tab and may also show the Field Watcher tab.

2. If needed, select the **JavaScript Log** tab.

3. Click the clear icon () to clear the contents of the log, as needed.

JavaScript debug window

The JavaScript debug window appears in a bottom pane of the user interface when an administrator turns on debugging.

Note: The JavaScript debug window is not supported with Next Experience in Tokyo. For more information about supported features in Next Experience, see [Considerations for activating Next Experience](#).

Use the debug window to access these tools.

- **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.

JavaScript debug window



Using the JavaScript debug window

The JavaScript debug window enables access to the JavaScript Log and the Field Watcher tools.

Before you begin

Role required: admin

About this task

The steps to access the JavaScript debug window depend on which UI version you are using.

Note: The JavaScript debug window is not supported with Next Experience in Tokyo. For more information about supported features in Next Experience, see [Considerations for activating Next Experience](#).

Procedure

1. Open the JavaScript debug window by navigating to the appropriate location for your version of the UI.

Core UI

- a. Click the gear icon in the banner frame.
- b. Click the **Developer** section.
- c. Toggle the **JavaScript Log** and **Field Watcher** switch.

UI15

a. Click the gear icon in the banner frame.

b. Click **JavaScript Log and Field Watcher**.

UI11

Click the debug icon () in 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 Log
- Field Watcher

Related concepts

- [Writing to the debug log](#)
- [Field watcher](#)

JS Code Coverage Debug

The JS Code Coverage Debug application allows administrators and application developers to log the scripts triggered during a user session and then review which lines of code the system ran.

Users with the js_coverage_debugger role can debug scripts without having to set breakpoints or review onscreen debug messages. Instead, the system saves script usage data in the JavaScript Code Coverage [sys_js_code_coverage] table. Each JavaScript Code Coverage record contains:

- The user session that called the script.
- The script record the system called identified by table, sys_id, and script field.
- The script record the system called identified by type and name.

- The transaction that called the script.
- The start time of the transaction.
- The contents of the script field highlighted to indicate which lines the system ran.

Sample code coverage highlighting

Script	Code
1	<code>var build = gs.getProperty("glide.buildname");</code>
2	<code>if (GlideStringUtil.notNil(build))</code>
3	<code> build = build.substring(0,1).toLowerCase();</code>
4	
5	<code>var collision = new GlideRecord("sys_embedded_help_content");</code>
6	<code>collision.addActiveQuery();</code>
7	<code>collision.addQuery("page", current.page);</code>
8	<code>collision.addQuery("modifier", current.modifier);</code>
9	<code>collision.addQuery("product", current.product);</code>
10	<code>if (current.isValidRecord() && GlideStringUtil.notNil(current.sys_id))</code>
11	<code> collision.addQuery("sys_id", "!=" , current.sys_id);</code>
12	
13	<code>if (GlideStringUtil.notNil(current.qualifier))</code>
14	<code> collision.addQuery("qualifier", current.qualifier);</code>
15	<code>else</code>
16	<code> collision.addNullQuery("qualifier");</code>
17	
18	<code>if (current.version == "all" && GlideStringUtil.notNil(build))</code>
19	<code> collision.addQuery("version", build).addOrCondition("version", "all");</code>
20	<code>else</code>
21	<code> collision.addQuery("version", current.version);</code>
22	
23	<code>collision.query();</code>
24	<code>if (collision.next()) {</code>
25	<code> if (collision.canWrite())</code>
26	<code> gs.addErrorMessage(gs.getMessage("Embedded Help Composite Key Validation"));</code>

JS Code Coverage highlighting

The JS Code Coverage application highlights script fields to indicate whether the system ran or skipped each line.

Sample code highlighting

Script

```
1  var build = gs.getProperty("glide.buildname");
2  if (GlideStringUtil.notNil(build))
3      build = build.substring(0,1).toLowerCase();
4
5  var collision = new GlideRecord("sys_embedded_help_content");
6  collision.addActiveQuery();
7  collision.addQuery("page", current.page);
8  collision.addQuery("modifier", current.modifier);
9  collision.addQuery("product", current.product);
10 if (current.isValidRecord() && GlideStringUtil.notNil(current.sys_id))
11     collision.addQuery("sys_id", "!=" , current.sys_id);
12
13 if (GlideStringUtil.notNil(current.qualifier))
14     collision.addQuery("qualifier", current.qualifier);
15 else
16     collision.addNullQuery("qualifier");
17
18 if (current.version == "all" && GlideStringUtil.notNil(build))
19     collision.addQuery("version", build).addOrCondition("version", "all");
20 else
21     collision.addQuery("version", current.version);
22
23 collision.query();
24 if (collision.next()) {
25     if (collision.canWrite())
26         gs.addErrorMessage(gs.getMessage("Embedded Help Composite Key Validation"));
```

The color of the highlight indicates how the system evaluated the code line.

Meaning of code highlighting

Highlight color	Meaning
Green	This is an executable line of code that the system ran during the session.
Red	This is an executable line of code that the system skipped for some reason. The system may have skipped an executable line of code because the necessary script conditions were not met or because the script function was never called. You may want to use the Script Debugger to determine

Highlight color	Meaning
	why the system skipped the line of executable code.
Gray	This is a non-executable line of code such as white space, code comment, or a portion of an expression split across multiple lines that cannot run on its own.

Administrators and application developers can use this information to conduct more targeted debugging activities such as using the Script Debugger to determine why script conditions are not being met.

Activate JS Code Coverage Debug

You can activate the JS Code Coverage Debug plugin (com.glide.js.coverage) if you have the admin role.

Before you begin

Role required: admin

Procedure

1. Navigate to **All > System Applications > All Available Applications > All**.
2. Find the plugin using the filter criteria and search bar.

You can search for the plugin by its name or ID. If you cannot find a plugin, you might have to request it from ServiceNow personnel.

3. Select **Install**, and then in the Activate Plugin dialog box, select **Activate**.

Note: When domain separation and delegated admin are enabled in an instance, the administrative user must be in the **global** domain. Otherwise, the following error appears: Application installation is unavailable because another operation is running: Plugin Activation for <plugin name>.

What to do next

To see the components the plugin installed, refresh the plugin form and select the **Plugin Files** related list.

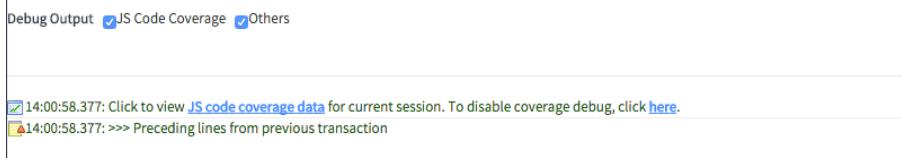
Debug with JS Code Coverage Debug

Use JS Code Coverage Debug to record a user session and then review which scripts and lines of code the system ran.

Before you begin

Role required: admin or js_coverage_debugger

Procedure

1. Navigate to **All > JS Code Coverage Debug > Enable Coverage**.
The system logs which scripts and code lines the system runs as well as displays session debug messages in the JS Code Coverage namespace.


The screenshot shows the 'JS Code Coverage Debug' page with the 'Enable Coverage' tab selected. At the top, there's a checkbox for 'JS Code Coverage' which is checked. Below it is a link 'Click to view JS code coverage data for current session. To disable coverage debug, click here.' Underneath, there's a note 'Preceding lines from previous transaction' with a small orange icon. The main area shows two log entries:

 - [14:00:58.377] Click to view JS code coverage data for current session. To disable coverage debug, click [here](#).
 - [14:00:58.377] >>> Preceding lines from previous transaction
2. Navigate to the table or page whose logic you want to test.
For example, navigate to **Incident > Create New**.
3. Trigger the script or scripts you want to test.
For example, create an incident with an associate CI item to test several business rules.
4. When you have completed testing, navigate to **JS Code Coverage Debug > Disable Coverage**.
The system stops logging script and code lines run.
5. Navigate to **JS Code Coverage Debug > Coverage Data**.
The system displays the list of coverage data associated with the current user session.

Javascript Code Coverages			
	Script Name	Script Reference	Transaction Name
All > Session = B61FFF2B4F20720082A074828110C793			
<input type="checkbox"/>	sys_script_include.d2426c9ec0a8016501958...	Script Include: JSON	#3894 /cmdb_ci_service_list.do
<input type="checkbox"/>	sys_script_include.d65f78c40a0a0b6900196...	Script Include: AbstractAjaxProcessor	#3921 /incident.do
<input type="checkbox"/>	sys_script.78e8bbe70a00070479138c7302ad6...	Business Rule: Update Parent Incident Count	#3921 /incident.do
<input type="checkbox"/>	sys_script_include.2e59987d0a0a2c3946f71...	Script Include: GSLog	#3921 /incident.do
<input type="checkbox"/>	sys_ui_action.7ca0a8d60a0a0b340080d6f48c...	UI Action: Edit...	#3894 /cmdb_ci_service_list.do
<input type="checkbox"/>	sys_script.62a7bfaf0a0a0a6500c49682bd823...	Business Rule: user query	#3887 /xmlhttp.do
<input type="checkbox"/>	sys_script.6cdfa0737f0000016fb2c23171995...	Business Rule: Run SLAs	#3921 /incident.do
<input type="checkbox"/>	sys_script.28beab035f201000b12e357f2b47...	Business Rule: Caller Close	#3921 /incident.do
<input type="checkbox"/>	sys_script.33b21f640a0a3c7400f6acob7d43f...	Business Rule: mark_resolved	#3921 /incident.do
<input type="checkbox"/>	sys_script.9782b8dac0a80a675bf2167f4ec8...	Business Rule: Task Active State Management	#3921 /incident.do
<input type="checkbox"/>	sys_script_include.61c188b30a0a0bb900dc6...	Script Include: SysMessageAjax	#3888 /xmlhttp.do
<input type="checkbox"/>	sys_script.d5e2b86cc0a80009011d75b919052...	Business Rule: insert_incident	#3921 /incident.do
<input type="checkbox"/>	sys_script.26e463f9ac100b0e5748246a0674...	Business Rule: Process SLAs	#3921 /incident.do
<input type="checkbox"/>	sys_script_include.4d9e146f9fa302000391b...	Script Include: IncidentState	#3921 /incident.do
<input type="checkbox"/>	sys_script.475ef3c5c611228401440a7a5f29a...	Business Rule: task closer	#3921 /incident.do
<input type="checkbox"/>	sys_script_include.7de297a8c0a8016400d5b...	Script Include: SysRefList	#3894 /cmdb_ci_service_list.do
<input type="checkbox"/>	sys_script_include.fb32a2d8c0a80a6000e90...	Script Include: ArrayUtil	#3921 /incident.do
<input type="checkbox"/>	sys_ui_action.42da42d00a0a0b340066377beb...	UI Action: Submit	#3921 /incident.do
<input type="checkbox"/>	sys_script.245748ecc61122aa016c02dfbf7f...	Business Rule: Incident Create Knowledge	#3921 /incident.do
<input type="checkbox"/>	sys_script.2bc2f9b1c0a801640199f9eb00673...	Business Rule: incident_query	#3922 /xmlhttp.do
Actions on selected rows...			
Actions on selected rows...			

6. Select the script or transaction you want to review.

JavaScript Code Coverage fields

Field	Description
Script Name	Displays the script run by table name, sys_id value, and script field.
Script Reference	Displays the script run by script type and name.
Transaction Name	Displays the transaction that called the script by thread ID and URI.

For example, select the **Script Reference** Business Rule: incident events.

The system displays the JS Code Coverage Debug record.

The screenshot shows a 'JS Code Coverage Debug Section' window with the following details:

- Script Name: sys_script.d56b5d71c0a80164019d0e0be2cf784f.script
- Script Reference: Business Rule: incident.events
- Transaction Name: #3921 /incident.do
- Start Time: 2017-02-01 10:55:44
- Script content (lines 1-23):

```
1 if (current.operation() != 'insert' && current.comments.changes()) {  
2     gs.eventQueue("incidentcommented", current, gs.getUserID(), gs.getUserName());  
3 }  
4  
5 if (current.operation() == 'insert') {  
6     gs.eventQueue("incident.inserted", current, gs.getUserID(), gs.getUserName());  
7 }  
8  
9 if (current.operation() == 'update') {  
10    gs.eventQueue("incident.updated", current, gs.getUserID(), gs.getUserName());  
11 }  
12  
13 if (!current.assigned_to.nil() && current.assigned_to.changes()) {  
14     gs.eventQueue("incident.assigned", current,  
15     current.assigned_to.getDisplayValue(), previous.assigned_to.getDisplayValue());  
16 }  
17  
18 if (!current.assignment_group.nil() && current.assignment_group.changes()) {  
19     gs.eventQueue("incident.assigned_to.group", current,  
20     current.assignment_group.getDisplayValue(),  
21     previous.assignment_group.getDisplayValue());  
22 }  
23 if (current.priority.changes() && current.priority == 1) {  
24     gs.eventQueue("incident.priority.1", current, current.priority,  
25     previous.priority);  
26 }
```

7. Review the **Script** field to determine which lines of code the system ran.

For example, the business rule added the incident.inserted event to the event queue.

Result

You determine which lines of code the system ran.

What to do next

Use the code coverage information to do more targeted debugging activities such as set breakpoints and review variable values with the Script Debugger.

Packages Call Removal tool

The Packages Call Removal Tool provides modules to identify fields that might contain scripts, find scripts that contain Packages calls to

ServiceNow Java classes, and to examine proposed script changes that eliminate those Packages calls.

Note: This tool is not activated by default, and is only available to users with the admin role.

Packages calls to ServiceNow Java classes will be prevented in a future release. The Packages Call Removal tool helps prepare your instance to use the new API and includes the following scripts and pages:

- The Find Packages Fields script scans scripts for Packages calls to ServiceNow Java classes.
- The Find Packages Calls script proposes changes that remove Packages calls or replaces them with GlideScriptable names.
- The Packages Call Items page lists and enables you to work on proposed changes to scripts.

The tool might generate errors as it tries to generate preferred, scriptable alternatives for Packages calls to ServiceNow Java classes.

Note: Create an update set before migrating the changes that result from running the Packages Call Removal Tool. For information, see [Get started with update sets](#).

Activate the Packages Call Removal Tool

You must activate the Packages Call Removal Tool plugin to access the tool.

Before you begin

Role required: admin

Procedure

1. Navigate to **All > System Applications > All Available Applications > All**.
2. Find the Packages Call Removal Tool plugin using the filter criteria and search bar.

You can search for the plugin by its name or ID. If you cannot find a plugin, you might have to request it from ServiceNow personnel.

3. Select **Install**, and then in the Activate Plugin dialog box, select **Activate**.

Note: When domain separation and delegated admin are enabled in an instance, the administrative user must be in the **global** domain. Otherwise, the following error appears: Application installation is unavailable because another operation is running: Plugin Activation for <plugin name>.

Find a Packages call

After you run the Find Packages Fields script to define the list of fields to search for Packages calls, run the Find Packages Calls script to generate the list of fields with Packages calls to ServiceNow Java classes. The script also proposes changes.

1. Click **(3) Find Packages Calls (script)** to execute a script that searches the fields for packages calls and populates the Packages Call Items list with proposed changes.

Find Packages Calls

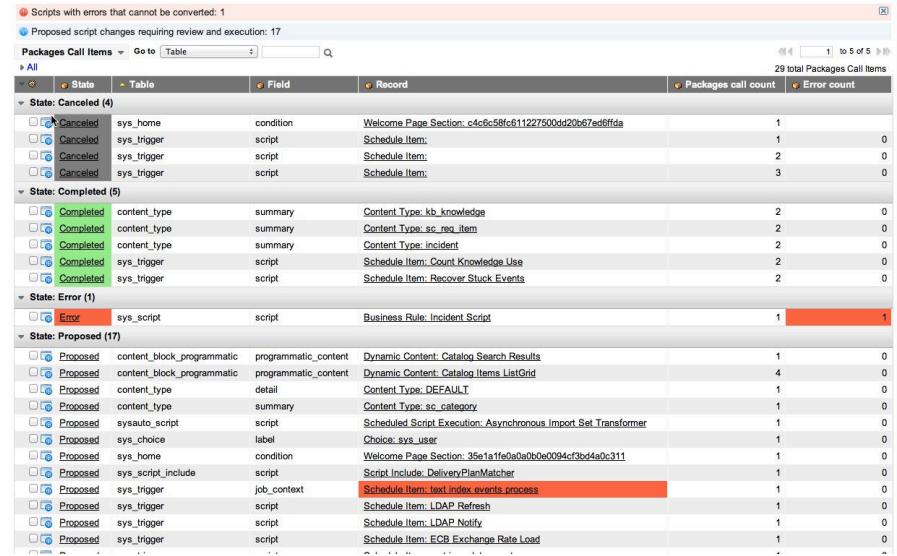
```
[0:00:01.349] Script completed: (3) Find Packages Calls (script)

*** Script: Searching Packages Call Fields for Packages calls in script, populating Packages Call Items
*** Script: Next, go to the 'Packages Call Items' module
*** Script: Script output below is only relevant to interested admins
*** Script:
*** Script: Searching aim_asset.acquisition_method for Packages calls in script
*** Script: Searching aim_license.assigned_condition for Packages calls in script
*** Script: Searching aim_timeline.condition for Packages calls in script
*** Script: Searching cim_contract_terms_and_conditions for Packages calls in script
*** Script: Searching catalog_ui.policy.catalog.condition for Packages calls in script
*** Script: Searching ci_identifier.script for Packages calls in script
*** Script: Searching cim_condition_checker.condition for Packages calls in script
*** Script: Searching cim_contract_history.terms_and_conditions for Packages calls in script
*** Script: Searching clone_cleanup_script.script for Packages calls in script
*** Script: Searching cmdb_assignment.condition for Packages calls in script
*** Script: Searching cmdb_baseline.condition for Packages calls in script
*** Script: Searching cmdb_depreciation.script for Packages calls in script
*** Script: Searching cmdb_sv_license_calculation.calulation for Packages calls in script
*** Script: Searching cmn_map_page.script for Packages calls in script
*** Script: Searching cmn_notify_message.condition for Packages calls in script
*** Script: Searching cmn_notify_service_provider.construction_script for Packages calls in script
*** Script: Searching cmn_notify_supplier.construction_script for Packages calls in script
*** Script: Searching cmn_relative_duration.script for Packages calls in script
*** Script: Searching cmn_schedule.condition.condition for Packages calls in script
*** Script: Searching cmn_schedule_page.client_script for Packages calls in script
*** Script: Searching cmn_schedule_page.server_script for Packages calls in script
*** Script: Searching cmn_timeline_page.condition for Packages calls in script
*** Script: Searching cmn_timeline_page.style.condition for Packages calls in script
*** Script: Searching content_block.condition for Packages calls in script
*** Script: Searching content_block_detail.script for Packages calls in script
*** Script: Searching content_block_lists.script for Packages calls in script
*** Script: Searching content_block_programmatic_content for Packages calls in script
*** Script: Searching content_page_rule.advanced_condition for Packages calls in script
*** Script: Searching content_page_rule.condition for Packages calls in script
*** Script: Searching content_type.condition for Packages calls in script
*** Script: Searching content_type.summary for Packages calls in script
*** Script: Searching content_type_detail.condition for Packages calls in script
*** Script: Searching content_type_detail.script for Packages calls in script
*** Script: Searching contract.sla.pause.condition for Packages calls in script
*** Script: Searching contract.sla.reset.condition for Packages calls in script
*** Script: Searching contract.sla.start.condition for Packages calls in script
*** Script: Searching contract.sla.stop.condition for Packages calls in script
*** Script: Searching diagrammer_action.condition for Packages calls in script
*** Script: Searching diagrammer_action.script for Packages calls in script
*** End of searching diagrammer_action.script for Packages calls in script
```

2. Click **(4) Packages Call Items** to display the list of affected fields on the Packages Call Items page.

The Packages Call Items page lists the items with Packages calls, shows each item's current state, the table that contains the field, the affected record, the number of Packages calls contained in the field's script, and the number of errors that occurred when the proposed script was generated.

Packages Call Items



The screenshot shows a grid of 29 total Packages Call Items. The items are grouped by State:

- State: Canceled (4)**: Includes sys_home, sys_trigger, sys_trigger, and sys_trigger.
- State: Completed (5)**: Includes content_type, content_type, content_type, sys_trigger, and sys_trigger.
- State: Error (1)**: Includes sys_script.
- State: Proposed (17)**: Includes content_block_programmatic, content_block_programmatic, content_type, content_type, sysauto_script, sys_choice, sys_home, sys_script_include, sys_trigger, sys_trigger, sys_trigger, sys_trigger, sys_trigger, sys_trigger, sys_trigger, sys_trigger, and sys_trigger.

Each row in the grid displays the following columns: State, Table, Field, Record, Packages call count, and Error count. The 'Error' row has a red background.

- On the Packages Call Items page, click a record for any of the listed items to open the form and revise as described in Replacing Packages Calls.

As you work through the list, the **State** of the items are updated and the items are grouped by State:

- Proposed**: A proposed revision exists for the field.
- Error(red)**: One or more errors occurred when the proposed script was generated.
- Rejected(gray)**: A proposed change has been rejected.
- Completed(green)**: The field has been successfully revised to remove Packages calls to ServiceNow Java classes.
- Canceled (gray)**: Since the proposed change was generated, either the original script has been changed to no longer require modification, or the original record no longer exists.

Glide object name

The names of the new Glide objects that replace Packages calls are derived from the Java package name used in the Packages call.

About this task

Although the tool automatically substitutes the appropriate new scriptable name for each Packages call it encounters, in some circumstances it can be useful to know how to manually replace a Packages call with its new scriptable object equivalent. Use the steps below to determine the new script object name from the Packages call name. The replacement objects include the same methods and properties as the objects they replace.

To determine the new object name:

Procedure

1. Note the third term in the Java package name. This is usually glide, but is sometimes snc or glideapp.
2. Drop all of the prefix terms, leaving only the last.
For example, Packages.com.glide.monitor.AbstractBucketCollector becomes AbstractBucketCollector.
3. Capitalize the first letter of the term noted in the first step above and add it to the front of the term defined in step 2.
For example, Packages.com.glide.monitor.AbstractBucketCollector becomes GlideAbstractBucketCollector and Packages.com.snc.cmdb.BaselineCMDB becomes SncBaselineCMDB.
4. Verify that the name is valid by executing a gs.print() command in Scripts Background, specifying only the name with no quotes. For example:

```
gs.print( SncBaselineCMDB );
```

```
*** Script: [JavaClass com.snc.cmdb.BaselineCMDB]
```

Note that there are exceptions to this rule, such as "Glide", which replaces "Packages.com.glide.Glide", "TestExtension", which replaces

"Packages.com.glide.junit.misc.TestExtension"; and "UINotification", which replaces "Packages.com.glide.ui.UINotification".

Glide object replacement list

This table lists the Glide classes and the Packages calls they replace.

Note: The publication of this list does not imply that these scriptable objects are for use by customers, consultants, and partners. The use of the Glide prefix does not imply that these scriptable objects are in the same category as or have the same status as objects such as GlideRecord. Except where documentation is provided in the API Reference, these undocumented APIs are not intended for general use, and ServiceNow, Inc., does not make any commitment to document them, answer questions about them, or maintain them indefinitely in their current form. Over time, ServiceNow, Inc., intends to migrate a subset of the functionality represented by these objects into the documented API and remove the rest.

GlideScriptable object replacement list

GlideScriptable Class	Packages Call
Glide	Packages.com.glide.Glide
GlideAbstractBucketCollector	Packages.com.glide.monitor.AbstractBucketCollector
GlideAbstractDomainProvider	Packages.com.glide.db.domain.AbstractDomainProvider
GlideAbstractExecutionPlan	Packages.com.glide.execution_plan.AbstractExecutionPlan
GlideAbstractListener	Packages.com.glide.listener.AbstractListener
GlideAbstractRenderer	Packages.com.glide.ui.portal.AbstractRenderer
GlideAction	Packages.com.glide.script.Action

GlideScriptable Class	Packages Call
GlideActionManager	Packages.com.glide.ui.action.ActionManager
GlideAJAXScheduleItem	Packages.com.glide.schedules.AJAXScheduleItem
GlideAJAXSchedulePage	Packages.com.glide.schedules.AJAXSchedulePage
GlideAlertActions	Packages.com.glide.alerts.AlertActions
GlideappAbstractChoiceListQuestion	Packages.com.glideapp.questionset.AbstractChoiceListQuestion
GlideappADSLILoader	Packages.com.glideapp.ecc.ADSDLILoader
GlideappAJAXMapPage	Packages.com.glideapp.google_maps.AJAXMapPage
GlideappCalculationHelper	Packages.com.glideapp.servicelog.CalculationHelper
GlideappCart	Packages.com.glideapp.servicelog.Cart
GlideappCartItem	Packages.com.glideapp.servicelog.CartItem
GlideappCatalogCategoryBatcher	Packages.com.glideapp.servicelog.CatalogCategoryBatcher
GlideappCatalogItem	Packages.com.glideapp.servicelog.CatalogItem
GlideappCategory	Packages.com.glideapp.servicelog.Category

GlideScriptable Class	Packages Call
GlideappCategoryPopper	Packages.com.glideapp.servicecatalog.CategoryPopper
GlideappCatItemPopper	Packages.com.glideapp.servicecatalog.CatItemPopper
GlideappChartParameters	Packages.com.glideapp.chart.ChartParameters
GlideappChatRoom	Packages.com.glideapp.live.db.ChatRoom
GlideappChatRoom\$Error	Packages.com.glideapp.live.db.ChatRoom.Error
GlideappCheckBoxQuestion	Packages.com.glideapp.questionset.CheckBoxQuestion
GlideappCMDBHelper	Packages.com.glideapp.ecc.CMDBHelper
GlideappCMDBSoftwareHelper	Packages.com.glideapp.ecc.CMDBSoftwareHelper
GlideappContainerAwareQuestionSet	Packages.com.glideapp.questionset.ContainerAwareQuestionSet
GlideappContextDiagramProcessor	Packages.com.glideapp.workflow.ui.ContextDiagramProcessor
GlideappDateQuestion	Packages.com.glideapp.questionset.DateQuestion
GlideappDateTimeQuestion	Packages.com.glideapp.questionset.DateTimeQuestion
GlideappDeliveryPlan	Packages.com.glideapp.servicecatalog.DeliveryPlan

GlideScriptable Class	Packages Call
GlideappECCInputMessage	Packages.com.glideapp.ecc.ECCInputMessage
GlideappECCOutputMessage	Packages.com.glideapp.ecc.ECCOutputMessage
GlideappECCQueueConnector	Packages.com.glideapp.ecc.ECCQueueConnector
GlideappECCQueueProcessor	Packages.com.glideapp.ecc.ECCQueueProcessor
GlideappECCResponseMessage	Packages.com.glideapp.ecc.ECCResponseMessage
GlideappExpandableText	Packages.com.glideapp.live_feed.HTMLTransformers.ExpandableText
GlideappExpertPanelCatalogOrder	Packages.com.glideapp.servicecatalog.ExpertPanelCatalogOrder
GlideappFixes	Packages.com.glideapp.servicecatalog.Fixes
GlideappHome	Packages.com.glideapp.home.Home
GlideappHomePage	Packages.com.glideapp.home.HomePage
GlideappHomePageFactory	Packages.com.glideapp.home.HomePageFactory
GlideappleCC	Packages.com.glideapp.ecc.IECC
GlideapplOUpgrade	Packages.com.glideapp.servicecatalog.IOUpgrade
GlideapItemOptionsQuestionSet	Packages.com.glideapp.servicecatalog.ItemOptionsQuestionSet

GlideScriptable Class	Packages Call
GlideappJMSECCReceiver	Packages.com.glideapp.jms.JMSECCReceiver
GlideappJMSECCSender	Packages.com.glideapp.jms.JMSECCSender
GlideappKBIncludes	Packages.com.glideapp.knowledge.KBIncludes
GlideApplication	Packages.com.glide.sys.Application
GlideApplicationModule	Packages.com.glide.processors.ApplicationModule
GlideappListCollectorQuestion	Packages.com.glideapp.questionset.ListCollectorQuestion
GlideappLiveFeedEventHandler	Packages.com.glideapp.live_feed.LiveFeedEventHandler
GlideappLiveFeedJournalWriter	Packages.com.glideapp.live_feed.LiveFeedJournalWriter
GlideappLiveFeedUIAction	Packages.com.glideapp.live_feed.LiveFeedUIAction
GlideappLiveProfile	Packages.com.glideapp.live_common.LiveProfile
GlideappLiveUtils	Packages.com.glideapp.live.LiveUtils
GlideappLookupSelectQuestion	Packages.com.glideapp.questionset.LookupSelectQuestion
GlideappMessageTag	Packages.com.glideapp.live_feed.MessageTag

GlideScriptable Class	Packages Call
GlideappOrderGuide	Packages.com.glideapp.servicecatalog.OrderGuide
GlideappProcessQueue	Packages.com.glideapp.ecc.ccmdb.ProcessQueue
GlideappQuestion	Packages.com.glideapp.questionset.Question
GlideappQuestionChoice	Packages.com.glideapp.questionset.QuestionChoice
GlideappQueueHelper	Packages.com.glideapp.ecc.QueueHelper
GlideappQueueReader	Packages.com.glideapp.ecc.QueueReader
GlideappReferenceQuestion	Packages.com.glideapp.questionset.ReferenceQuestion
GlideappRequestItemWorkflow	Packages.com.glideapp.servicecatalog.RequestItemWorkflow
GlideappRequestNew	Packages.com.glideapp.servicecatalog.RequestNew
GlideappScriptHelper	Packages.com.glideapp.servicecatalog.ScriptHelper
GlideappSecurityMask	Packages.com.glideapp.servicecatalog.SecurityMask
GlideappSequencedQuestionSet	Packages.com.glideapp.questionset.SequencedQuestionSet
GlideappTaskApprovalHelper	Packages.com.glideapp.servicecatalog.TaskApprovalHelper

GlideScriptable Class	Packages Call
GlideappTimeAgo	Packages.com.glideapp.live_feed.TimeAgo
GlideappUpdateVersion	Packages.com.glideapp.version.UpdateVersion
GlideappUpgradeQuestions	Packages.com.glideapp.survey.UpgradeQuestions
GlideappValveProcessor	Packages.com.glideapp.servicecatalog.valve.ValveProcessor
GlideappVariable	Packages.com.glideapp.servicecatalog.variables.Variable
GlideappVariablePoolQuestionSet	Packages.com.glideapp.servicecatalog.variables.VariablePoolQuestionSet
GlideappWizardIntercept	Packages.com.glideapp.wizard.WizardIntercept
GlideappWMILoader	Packages.com.glideapp.ecc.WMI Loader
GlideappWorkflow	Packages.com.glideapp.workflow.Workflow
GlideappWorkflowHelper	Packages.com.glideapp.workflow.WorkflowHelper
GlideappYesNoQuestion	Packages.com.glideapp.questions.YesNoQuestion
GlideAQueryExplanation	Packages.com.glide.db.explain.AQueryExplanation
GlideArchiver	Packages.com.glide.db.auxiliary.Archiver

GlideScriptable Class	Packages Call
GlideArchiveRecord	Packages.com.glide.db.auxiliary.ArchiveRecord
GlideArchiveRestore	Packages.com.glide.db.auxiliary.ArchiveRestore
GlideArchiveStatus	Packages.com.glide.db.auxiliary.ArchiveStatus
GlideArchiveTable	Packages.com.glide.db.auxiliary.ArchiveTable
GlideARecurrence	Packages.com.glide.schedule.recurrence.ARoccurrence
GlideAttachmentIndexDocument	Packages.com.glide.lucene.attachments.AttachmentIndexDocument
GlideAttachmentIndexTypes	Packages.com.glide.lucene.attachments.AttachmentIndexTypes
GlideAttributes	Packages.com.glide.util.GlideAttributes
GlideAuditDelete	Packages.com.glide.audit.AuditDelete
GlideAuditor	Packages.com.glide.script.Auditor
GlideAutomationEncrypter	Packages.com.glide.util.AutomationEncrypter
GlideBaseTag	Packages.com.glide.ui.jelly.tags.BaseTag
GlideBootstrap	Packages.com.glide.db.impex.Bootstrap

GlideScriptable Class	Packages Call
GlideBoundedIntProperty	Packages.com.glide.util.BoundedIntProperty
GlideCacheManager	Packages.com.glide.sys.cache.CacheManager
GlideCalendar	Packages.com.glide.schedule.GlideCalendar
GlideCalendarWeekEntry	Packages.com.glide.calendar.GlideCalendarWeekEntry
GlideCanceledUITransaction	Packages.com.glide.ui.CanceledUITransaction
GlideCascadeFromDelete	Packages.com.glide.db.CascadeFromDelete
GlideCatalogCloneWorker	Packages.com.glide.catalog.clone.CatalogCloneWorker
GlideChannel	Packages.com.glide.channel.Channel
GlideChannelManager	Packages.com.glide.channel.ChannelManager
GlideChannelMessage	Packages.com.glide.channel.ChannelMessage
GlideChartFieldColors	Packages.com.glide.ui.chart.dataSet.ChartFieldColors
GlideChartGeneratorFactory	Packages.com.glide.ui.chart.ChartGeneratorFactory
GlideChartUtil	Packages.com.glide.ui.chart.dataSet.ChartUtil

GlideScriptable Class	Packages Call
GlideChartValue	Packages.com.glide.ui.chart.dataSet.ChartValue
GlideChecksum	Packages.com.glide.util.Checksum
GlideChoice	Packages.com.glide.choice.Choice
GlideChoiceList	Packages.com.glide.choice.ChoiceList
GlideChoiceListGenerator	Packages.com.glide.choice.ChoiceListGenerator
GlideChoiceListSet	Packages.com.glide.choice.ChoiceListSet
GlideChoiceListUpdateSaver	Packages.com.glide.update.saver.ChoiceListUpdateSaver
GlideClientBrowserTimes	Packages.com.glide.client_transaction.ClientBrowserTimes
GlideClientNetworkTimes	Packages.com.glide.client_transaction.ClientNetworkTimes
GlideClusterMessage	Packages.com.glide.cluster.ClusterMessage
GlideClusterState	Packages.com.glide.cluster.ClusterState
GlideClusterSynchronizer	Packages.com.glide.cluster.ClusterSynchronizer
GlideCMSLinkHelper	Packages.com.glide.cms.CMSLinkHelper

GlideScriptable Class	Packages Call
GlideCMSPageLink	Packages.com.glide.cms.CMSPageLink
GlideCollectionEnumerator	Packages.com.glide.util.CollectionEnumerator
GlideCollectionQueryCalculator	Packages.com.glide.ui.CollectionQueryCalculator
GlideCollisionDetector	Packages.com.glide.update.collisions.CollisionDetector
GlideColumnAttributes	Packages.com.glide.db.impex.ColumnAttributes
GlideCompanyResolver	Packages.com.glide.misc.CompanyResolver
GlideCompiler	Packages.com.glide.script.Compiler
GlideCompositeElement	Packages.com.glide.db.CompositeElement
GlideConfiguration	Packages.com.glide.notification.Configuration
GlideContentConfig	Packages.com.glide.cms.ContentConfig
GlideContentPage	Packages.com.glide.cms.ContentPage
GlideContentSite	Packages.com.glide.cms.ContentSite
GlideContentType	Packages.com.glide.cms.ContentType

GlideScriptable Class	Packages Call
GlideContextMenu	Packages.com.glide.db_context_menu.ContextMenu
GlideContextMenuItem	Packages.com.glide.db_context_menu.ContextMenuItem
GlideContextualSecurityManager	Packages.com.glide.sys.security.ContextualSecurityManager
GlideController	Packages.com.glide.script.GlideController
GlideConverter	Packages.com.glide.currency.Converter
GlideCookieMan	Packages.com.glide.ui.CookieMan
GlideCounter	Packages.com.glide.util.Counter
GlideCredentials	Packages.com.glide.communications.crypto.Credentials
GlideCryptoService	Packages.com.glide.security.CryptoService
GlideCSVExporter	Packages.com.glide.generators.CSVExporter
GlideCustomerScriptFixer	Packages.com.glide.script.api.CustomerScriptFixer
GlideDatabaseVerifier	Packages.com.glide.db.DatabaseVerifier
GlideDatabaseViewLink	Packages.com.glide.database_views.DatabaseViewLink
GlideDataSource	Packages.com.glide.db.impex.datasource.DataSource

GlideScriptable Class	Packages Call
GlideDate	Packages.com.glide.glideobject.GlideDate
GlideDateTime	Packages.com.glide.glideobject.GlideDateTime
GlideDateUtil	Packages.com.glide.util.DateUtil
GlideDBAction	Packages.com.glide.db.DBAction
GlideDBAggregateQuery	Packages.com.glide.db.DBAggregateQuery
GlideDBAggregateUtil	Packages.com.glide.db.DBAggregateUtil
GlideDBCategoryDebug	Packages.com.glide.secondary_db_pools.DBCategoryDebug
GlideDBChangeManager	Packages.com.glide.db.change.DBChangeManager
GlideDBCompositeAction	Packages.com.glide.db.DBCompositeAction
GlideDBConfiguration	Packages.com.glide.db.DBConfiguration
GlideDBConfigurationManager	Packages.com.glide.db.DBConfigurationManager
GlideDBConfigurationManagerEventHandler	Packages.com.glide.db.DBConfigurationManagerEventHandler
GlideDBConfigurationParms	Packages.com.glide.db.DBConfigurationParms
GlideDBConfigurationV2Migrator	Packages.com.glide.db.DBConfigurationV2Migrator

GlideScriptable Class	Packages Call
GlideDBConnection	Packages.com.glide.db.pool.DBConnection
GlideDBConnectionPool	Packages.com.glide.db.pool.DBConnectionPool
GlideDBConnectionPooler	Packages.com.glide.db.pool.DBConnectionPooler
GlideDBDelete	Packages.com.glide.db.DBDelete
GlideDBI	Packages.com.glide.db.DBI
GlideDBImageProvider	Packages.com.glide.db_image.DBImageProvider
GlideDBIMySQL	Packages.com.glide.db.rdbms.mysql.DBIMySQL
GlideDBIndex	Packages.com.glide.db.DBIndex
GlideDBKeyStoreFactory	Packages.com.glide.certificates.DBKeyStoreFactory
GlideDBMacro	Packages.com.glide.ui.jelly.tags.form.DBMacro
GlideDBMicroStats	Packages.com.glide.db.DBMicroStats
GlideDBMultiTargetAction	Packages.com.glide.db.DBMultiTargetAction
GlideDBObjectManager	Packages.com.glide.db.DBObjectManager
GlideDBObjectToken	Packages.com.glide.db.DBObjectToken

GlideScriptable Class	Packages Call
GlideDBPoolTest	Packages.com.glide.secondary_db_pools.DBPoolTest
GlideDBPropertiesConfig	Packages.com.glide.db.DBPropertiesConfig
GlideDBQuery	Packages.com.glide.db.DBQuery
GlideDBTypes	Packages.com.glide.db.DBTypes
GlideDBUpdate	Packages.com.glide.db.DBUpdate
GlideDBUtil	Packages.com.glide.db.DBUtil
GlideDBView	Packages.com.glide.db.DBView
GlideDebugEvaluator	Packages.com.glide.jsdebug.DebugEvaluator
GlideDefaultUpdateSaver	Packages.com.glide.update.saver.DefaultUpdateSaver
GlideDiagram	Packages.com.glide.diagrammer.Diagram
GlideDiagramAction	Packages.com.glide.diagrammer.DiagramAction
GlideDiagramEdge	Packages.com.glide.diagrammer.DiagramEdge
GlideDiagramElement	Packages.com.glide.diagrammer.DiagramElement
GlideDiagramNode	Packages.com.glide.diagrammer.DiagramNode
GlideDistUpgradeRunner	Packages.com.glide.dist.upgrade.runner.DistUpgradeRunner

GlideScriptable Class	Packages Call
GlideDocument	Packages.com.glide.util.GlideDocument
GlideDomain	Packages.com.glide.db.domain.Domain
GlideDomainDisplay	Packages.com.glide.db.domain.DomainDisplay
GlideDomainHierarchy	Packages.com.glide.db.domain.DomainHierarchy
GlideDomainNumberProvider	Packages.com.glide.db.domain.DomainNumberProvider
GlideDomainPathDisplay	Packages.com.glide.db.domain.DomainPathDisplay
GlideDomainPathProvider	Packages.com.glide.db.domain.DomainPathProvider
GlideDomainSpoolProvider	Packages.com.glide.db.domain.DomainSpoolProvider
GlideDomainSupport	Packages.com.glide.db.domain.DomainSupport
GlideDomainTree	Packages.com.glide.db.domain.DomainTree
GlideDomainUtil	Packages.com.glide.db.domain.DomainUtil
GlideDomainValidator	Packages.com.glide.db.domain.DomainValidator
GlideDuration	Packages.com.glide.glideobject.GlideDuration

GlideScriptable Class	Packages Call
GlideECBDownloader	Packages.com.glide.currency.ECBDownloader
GlideECCQueueTransformer	Packages.com.glide.db.impex.ECQueueTransformer
GlideElement	Packages.com.glide.script.GlideElement
GlideElementDescriptor	Packages.com.glide.db.ElementDescriptor
GlideElementIterator	Packages.com.glide.util.ElementIterator
GlideElementUserImage	Packages.com.glide.script.glide_elements.GlideElementUserImage
GlideElementXMLSerializer	Packages.com.glide.script.GlideElementXMLSerializer
GlideEmail	Packages.com.glide.notification.Email
GlideEmailAction	Packages.com.glide.notification.outbound.EmailAction
GlideEmailFormatter	Packages.com.glide.notification.outbound.EmailFormatter
GlideEmailInbound	Packages.com.glide.notification.inbound.EmailInbound
GlideEmailOutbound	Packages.com.glide.notification.outbound.EmailOutbound
GlideEmailReader	Packages.com.glide.notification.inbound.EmailReader

GlideScriptable Class	Packages Call
GlideEmailSender	Packages.com.glide.notification.outbound.EmailSender
GlideEmailWatermark	Packages.com.glide.notification.EmailWatermark
GlideEmitter	Packages.com.glide.ui.jelly.Emitter
GlideEncrypter	Packages.com.glide.util.Encrypter
GlideEncryptionContext	Packages.com.glide.sys.EncryptionContext
GlideEncryptionContextCipher	Packages.com.glide.sys.security.EncryptionContextCipher
GlideEncryptionWrapperDB	Packages.com.glide.sys.security.EncryptionWrapperDB
GlideEncryptionWrapperDBAdmin	Packages.com.glide.sys.security.EncryptionWrapperDBAdmin
GlideEncryptionWrapperNAE	Packages.com.glide.sys.security.EncryptionWrapperNAE
GlideEncryptionWrapperNAEAdmin	Packages.com.glide.sys.security.EncryptionWrapperNAEAdmin
GlideEscalationManager	Packages.com.glide.escalation.EscalationManager
GlideEscalationTimerJobMarkII	Packages.com.glide.job.EscalationTimerJobMarkII
GlideEvaluator	Packages.com.glide.script.Evaluator
GlideEvent	Packages.com.glide.policy.Event

GlideScriptable Class	Packages Call
GlideEventManager	Packages.com.glide.policy.EventManager
GlideExcelExporter	Packages.com.glide.generators.ExcelExporter
GlideExcelLoader2	Packages.com.glide.db.impex.ExcelLoader2
GlideExecutionPlan	Packages.com.glide.execution_plan.ExecutionPlan
GlideExpressionWrapper	Packages.com.glide.ui.jelly.GlideExpressionWrapper
GlideExtensionPoint	Packages.com.glide.sys.ExtensionPoint
GlideFieldList	Packages.com.glide.processors.FieldList
GlideFile	Packages.com.glide.script.proxy.File
GlideFileUtil	Packages.com.glide.util.FileUtil
GlideFilter	Packages.com.glide.script.Filter
GlideFilterList	Packages.com.glide.script.FilterList
GlideFixCatalogPlans	Packages.com.glide.fixes.FixCatalogPlans
GlideFixDeliveryPlans	Packages.com.glide.fixes.FixDeliveryPlans
GlideFixGroups	Packages.com.glide.fixes.FixGroups

GlideScriptable Class	Packages Call
GlideFixItemOptionsAgain	Packages.com.glide.fixes.FixItemOptionsAgain
GlideFixRules	Packages.com.glide.fixes.FixRules
GlideFixSpellCheck	Packages.com.glide.fixes.FixSpellCheck
GlideFixStuff	Packages.com.glide.fixes.FixStuff
GlideFixUsers	Packages.com.glide.fixes.FixUsers
GlideForm	Packages.com.glide.ui.GlideForm
GlideFormCommon	Packages.com.glide.ui.GlideFormCommon
GlideFormulator	Packages.com.glide.ui.GlideFormulator
GlideGauge	Packages.com.glide.report.Gauge
GlideGovernor	Packages.com.glide.sys.util.Governor
GlideGregorianCalendar	Packages.com.glide.util.GlideGregorianCalendar
GlideGroup	Packages.com.glide.sys.Group
GlideGroupByListTag	Packages.com.glide.ui.jelly.tags.form.GroupByListTag
GlideGuid	Packages.com.glide.util.Guid
GlideHierarchicalReference	Packages.com.glide.glideobject.HierarchicalReference
GlideHistorySet	Packages.com.glide.audit.HistorySet

GlideScriptable Class	Packages Call
GlideHistoryTag2	Packages.com.glide.ui.jelly.tags.mergedata.HistoryTag2
GlideHostUtil	Packages.com.glide.util.HostUtil
GlideHTTPClient	https://www.youtube.com/watch?feature=player_detailpage&v=9qYI_pBZej
GlideHTTPRequest	Packages.com.glide.communications.HTTPRequest
GlideHTTPResponse	Packages.com.glide.communications.HTTPResponse
GlideI18NStyle	Packages.com.glide.ui.I18NStyle
GlideICALUtil	Packages.com.glide.policy.ICALUtil
GlideIConstants	Packages.com.glide.util.IConstants
GlideIGlideRecord	Packages.com.glide.util.IGlideRecord
GlideImageLoader	Packages.com.glide.script.ImageLoader
GlideImpersonate	Packages.com.glide.sys.Impersonate
GlideImportLog	Packages.com.glide.db.impex.ImportLog
GlideImportMap	Packages.com.glide.db.impex.ImportMap
GlideImportMapField	Packages.com.glide.db.impex.ImportMapField

GlideScriptable Class	Packages Call
GlideImportSet	Packages.com.glide.system_import_set.ImportSet
GlideImportSetLoader	Packages.com.glide.system_import_set.ImportSetLoader
GlideImportSetRun	Packages.com.glide.system_import_set.ImportSetRun
GlideImportSetTransformer	Packages.com.glide.system_import_set.ImportSetTransformer
GlideImportSetTransformerWorker	Packages.com.glide.system_import_set.ImportSetTransformerWorker
GlideIndexDescriptor	Packages.com.glide.db.IndexDescriptor
GlideIndexUtils	Packages.com.glide.db.IndexUtils
GlideIntegerTime	Packages.com.glide.glideobject.IntegerTime
GlideInternalElementTypeChoiceList	Packages.com.glide.script.InternalElementTypeChoiceList
GlideInternalMonitor	Packages.com.glide.ui.monitor.InternalMonitor
GlideIOMonitor	Packages.com.glide.ui.monitor.IOMonitor
GlideIOStats	Packages.com.glide.db.IOStats
GlideIPAddressUtil	Packages.com.glide.util.IPAddressUtil
GlideIQueryCondition	Packages.com.glide.util.IQueryCondition

GlideScriptable Class	Packages Call
GlideIRow	Packages.com.glide.db.meta.IRow
GlideIRQuerySummary	Packages.com.glide.db.ir.IRQuerySummary
GlideIRQuerySummarySimple	Packages.com.glide.db.ir.IRQuerySummarySimple
GlideISecurityManager	Packages.com.glide.sys.security.ISecurityManager
GlideITableIterator	Packages.com.glide.db.access.ITableIterator
GlideJDBCLoader	Packages.com.glide.db.impex.JDBCLoader
GlideJDBCProbeTestWorker	Packages.com.glide.db.impex.JDBCProbeTestWorker
GlideJellyContext	Packages.com.glide.ui.jelly.GlideJellyContext
GlideJellyRunner	Packages.com.glide.ui.jelly.JellyRunner
GlideJID	Packages.com.glide.xmpp.JID
GlideJSTestUtil	Packages.com.glide.autotester.JSTestUtil
GlideJSUtil	Packages.com.glide.script.JSUtil
GlideLabelEventHandler	Packages.com.glide.labels.LabelEventHandler
GlideLabelGenerator	Packages.com.glide.db.LabelGenerator

GlideScriptable Class	Packages Call
GlideLabelUtil	Packages.com.glide.labels.LabelUtil
GlideLDAP	Packages.com.glide.sys.ldap.LDAP
GlideLDAPConfig	Packages.com.glide.sys.ldap.LDAPConfig
GlideLDAPConfigurations	Packages.com.glide.sys.ldap.LDAPConfigurations
GlideLDAPErrorAnalyzer	Packages.com.glide.sys.ldap.LDAPErrorAnalyzer
GlideLDAPGroups	Packages.com.glide.sys.ldap.LDAPGroups
GlideLDAPRefresh	Packages.com.glide.sys.ldap.LDAPRefresh
GlideLDAPResult	Packages.com.glide.sys.ldap.LDAPResult
GlideLDAPTarget	Packages.com.glide.sys.ldap.LDAPTarget
GlideLDAPTransformQueue	Packages.com.glide.sys.ldap.LDAPTransformQueue
GlideLDAPUsers	Packages.com.glide.sys.ldap.LDAPUsers
GlideLDAPUserUpdate	Packages.com.glide.sys.ldap.LDAPUserUpdate
GlideList	Packages.com.glide.glideobject.GlideList
GlideListGroupProperties	Packages.com.glide.list_v2.ListGroupProperties

GlideScriptable Class	Packages Call
GlideListLabel	Packages.com.glide.ui.LabelControl
GlideListM2MBacking	Packages.com.glide.glideobject.GlideListM2MBacking
GlideListProperties	Packages.com.glide.list_v2.ListProperties
GlideListSearchQuery	Packages.com.glide.ui.ListSearchQuery
GlideLoader	Packages.com.glide.db.impex.Loader
GlideLoadTestDirector	Packages.com.glide.load_test.LoadTestDirector
GlideLocale	Packages.com.glide.sys.GlideLocale
GlideLocaleLoader	Packages.com.glide.currency.LocaleLoader
GlideLock	Packages.com.glide.script.Lock
GlideLog	Packages.com.glide.util.Log
GlideLogCleanup	Packages.com.glide.util.LogCleanup
GlideLogFileReader	Packages.com.glide.log_file.LogFileReader
GlideLRUCache	Packages.com.glide.sys.cache.LRU Cache
GlideLuceneTextIndexEvent	Packages.com.glide.lucene.TextIndexEvent

GlideScriptable Class	Packages Call
GlideMarkupWriter	Packages.com.glide.util.MarkupWriter
GlideMemoryActive	Packages.com.glide.ui.monitor.MemoryActive
GlideMemoryCache	Packages.com.glide.ui.monitor.MemoryCache
GlideMemoryRecord	Packages.com.glide.script.GlideMemoryRecord
GlideMemorySwap	Packages.com.glide.ui.monitor.MemorySwap
GlideMemoryTable	Packages.com.glide.util.MemoryTable
GlideMemoryTotal	Packages.com.glide.ui.monitor.MemoryTotal
GlideMetaData	Packages.com.glide.script.MetaData
GlideMIDServerInfoAccessor	Packages.com.glide.script.MIDServerInfoAccessor
GlideMobileExtensions	Packages.com.glide.ui.MobileExtensions
GlideModule	Packages.com.glide.sys.Module
GlideMultipleAction	Packages.com.glide.db.MultipleAction
GlideMultipleDelete	Packages.com.glide.db.MultipleDelete
GlideMultipleInsert	Packages.com.glide.db.MultipleInsert

GlideScriptable Class	Packages Call
GlideMultipleUpdate	Packages.com.glide.db.MultipleUpdate
GlideMutex	Packages.com.glide.sys.lock.Mutex
GlideMySQLWatch	Packages.com.glide.sys.stats.MySQLWatch
GlideNumber	Packages.com.glide.script.glide_elements.GlideNumber
GlideNumberManager	Packages.com.glide.db.NumberManager
GlideObjectManager	Packages.com.glide.glideobject.GlideObjectManager
GlideObjectUtil	Packages.com.glide.util.ObjectUtil
GlideOrderingDefinitionCreator	Packages.com.glide.sorting.OrderingDefinitionCreator
GlideOrderingManager	Packages.com.glide.sorting.OrderingManager
GlideOutputWriter	Packages.com.glide.ui.io.GlideOutputWriter
GlideOutputWriterFactory	Packages.com.glide.ui.io.GlideOutputWriterFactory
GlideOverLoadedChoices	Packages.com.glide.script.OverLoadedChoices
GlidePartitionMonitor	Packages.com.glide.ui.monitor.PartitionMonitor
GlidePivotTableSummaryTableWriter	Packages.com.glide.ui.chart.dataSet.PivotTableSummaryTableWriter

GlideScriptable Class	Packages Call
GlidePlugin	Packages.com.glide.sys.Plugin
GlidePluginManager	Packages.com.glide.sys.PluginManager
GlidePluginManagerWorker	Packages.com.glide.sys.PluginManagerWorker
GlidePluginUtils	Packages.com.glide.sys.PluginUtils
GlidePOP3Reader	Packages.com.glide.notification inbound.POP3Reader
GlidePOP3ReaderJob	Packages.com.glide.job.POP3ReaderJob
GlidePopup	Packages.com.glide.ui.Popup
GlidePriceGenerator	Packages.com.glide.currency.PriceGenerator
GlidePriceLoader	Packages.com.glide.currency.PriceLoader
GlideProcessor	Packages.com.glide.processors.Processor
GlideProcessRunner	Packages.com.glide.util.ProcessRunner
GlideProgressMonitor	Packages.com.glide.worker.ProgressMonitor
GlideProgressWorker	Packages.com.glide.worker.ProgressWorker
GlideProperties	Packages.com.glide.util.GlideProperties

GlideScriptable Class	Packages Call
GlidePropertiesDB	Packages.com.glide.util.GlidePropertiesDB
GlideProperty	Packages.com.glide.util.GlideProperty
GlidePublicPage	Packages.com.glide.ui.PublicPage
GlideQueryBreadcrumbs	Packages.com.glide.misc.QueryBreadcrumbs
GlideQueryCondition	Packages.com.glide.db.conditions.QueryCondition
GlideQueryFormatter	Packages.com.glide.ui.jelly.tags.form.QueryFormatter
GlideQueryString	Packages.com.glide.db.QueryString
GlideQueryTerm	Packages.com.glide.db.QueryTerm
GlideRecord	Packages.com.glide.script.GlideRecord
GlideRecordCache	Packages.com.glide.sys.RecordCache
GlideRecordEnsurer	Packages.com.glide.misc.RecordEnsurer
GlideRecordFactory	Packages.com.glide.script.GlideRecordFactory
GlideRecordKeySetLoader	Packages.com.glide.script.GlideRecordKeySetLoader
GlideRecordLock	Packages.com.glide.script.RecordLock

GlideScriptable Class	Packages Call
GlideRecordPopupGenerator	Packages.com.glide.calendar.RecordPopupGenerator
GlideRecordRollback	Packages.com.glide.script.GlideRecordRollback
GlideRecordSet	Packages.com.glide.script.GlideRecordSet
GlideRecordSimpleSerializer	Packages.com.glide.script.GlideRecordSimpleSerializer
GlideRecordXMLSerializer	Packages.com.glide.script.GlideRecordXMLSerializer
GlideReferenceField	Packages.com.glide.script.ReferenceField
GlideRegexUtil	Packages.com.glide.util.RegexUtil
GlideRegisterEscalationEvents	Packages.com.glide.fixes.RegisterEscalationEvents
GlideRelatedListReconciler	Packages.com.glide.misc.RelatedListReconciler
GlideRelationship	Packages.com.glide.sys.Relationship
GlideRelationships	Packages.com.glide.db.Relationships
GlideRelationshipUtil	Packages.com.glide.sys.RelationshipUtil
GlideRemoteGlideRecord	Packages.com.glide.communications.RemoteGlideRecord
GlideRenderProperties	Packages.com.glide.ui.RenderProperties

GlideScriptable Class	Packages Call
GlideReplaceUpdateFiles	Packages.com.glide.util.ReplaceUpdateFiles
GlideReplicationEngine	Packages.com.glide.replicator.ReplicationEngine
GlideReport	Packages.com.glide.report.Report
GlideReportChoiceList	Packages.com.glide.script.ReportChoiceList
GlideReportViewManagement	Packages.com.glide.report.ReportViewManagement
GlideRequestMap	Packages.com.glide.util.RequestMap
GlideRevertToOutOfBox	Packages.com.glide.update.RevertToOutOfBox
GlideRhinoEnvironment	Packages.com.glide.script.GlideRhinoEnvironment
GlideRhinoHelper	Packages.com.glide.script.GlideRhinoHelper
GlideRhinoScope	Packages.com.glide.script.RhinoScope
GlideRhinoScopeHandler	Packages.com.glide.script.GlideRhinoScopeHandler
GlideRhinoTestCase	Packages.com.glide.autotester.RhinoTestCase
GlideRRDBAlertProcessor	Packages.com.glide.rrdb.alerts.RRDBAlertProcessor
GlideRRDBDefinition	Packages.com.glide.rrdb.RRDBDefinition

GlideScriptable Class	Packages Call
GlideRunScriptJob	Packages.com.glide.job.RunScriptJob
GlideSchedule	Packages.com.glide.schedules.Schedule
GlideScheduleDateTime	Packages.com.glide.glideobject.ScheduleDateTime
GlideScheduleDateTimeSpan	Packages.com.glide.schedules.ScheduleDateTimeSpan
GlideScheduleItem	Packages.com.glide.schedules.ScheduleItem
GlideScheduler	Packages.com.glide.schedule.GlideScheduler
GlideScheduleTimeMap	Packages.com.glide.schedules.ScheduleTimeMap
GlideScheduleTimeSpan	Packages.com.glide.schedules.ScheduleTimeSpan
GlideScriptChoiceList	Packages.com.glide.script.ChoiceList
GlideScriptedProgressWorker	Packages.com.glide.worker.ScriptedProgressWorker
GlideScriptEvaluator	Packages.com.glide.script.ScriptEvaluator
GlideScriptGlobals	Packages.com.glide.script.GlideScriptGlobals
GlideScriptListener	Packages.com.glide.listener.ScriptListener

GlideScriptable Class	Packages Call
GlideScriptProcessor	Packages.com.glide.processors.ScriptProcessor
GlideScriptRecordUtil	Packages.com.glide.script.GlideRecordUtil
GlideScriptSystemUtilDB	Packages.com.glide.script.GlideSystemUtilDB
GlideScriptViewManager	Packages.com.glide.ui.ViewManager
GlideScriptWriter	Packages.com.glide.script.ScriptWriter
GlideSearchQueryFormatter	Packages.com.glide.text_search.SearchQueryFormatter
GlideSecondaryDatabaseBehindnessChecker	Packages.com.glide.secondary_db_pools.SecondaryDatabaseBehindnessChecker
GlideSecondaryDatabaseConfiguration	Packages.com.glide.secondary_db_pools.SecondaryDatabaseConfiguration
GlideSecurityManager	Packages.com.glide.sys.security.GlideSecurityManager
GlideSecurityQueryCalculator	Packages.com.glide.sys.security.SecurityQueryCalculator
GlideSecurityUtils	Packages.com.glide.util.SecurityUtils
GlideSelfCleaningMutex	Packages.com.glide.sys.lock.SelfCleaningMutex
GlideServiceAPIWrapper	Packages.com.glide.service_api.ServiceAPIWrapper

GlideScriptable Class	Packages Call
GlideServlet	Packages.com.glide.ui.GlideServlet
GlideServletRequest	Packages.com.glide.ui.GlideServletRequest
GlideServletResponse	Packages.com.glide.ui.GlideServletResponse
GlideServletStatus	Packages.com.glide.ui.ServletStatus
GlideSession	Packages.com.glide.sys.GlideSession
GlideSessionDebug	Packages.com.glide.sys.SessionDebug
GlideSessions	Packages.com.glide.ui.Sessions
GlideSessionSandbox	Packages.com.glide.script.GlideSessionSandbox
GlideShellCommand	Packages.com.glide.util.ShellCommand
GlideSimmerDown	Packages.com.glide.db.change.command.SimmerDown
GlideSimmerUp	Packages.com.glide.db.change.command.SimmerUp
GlideSimpleDateFormatEx	Packages.com.glide.util.SimpleDateFormatEx
GlideSimpleHTTPClient	Packages.com.glide.communications.SimpleHTTPClient
GlideSimpleScriptListener	Packages.com.glide.listener.SimpleScriptListener

GlideScriptable Class	Packages Call
GlideSMTPConnection	Packages.com.glide.notification.outbound.SMTPConnection
GlideSMTPSender	Packages.com.glide.notification.outbound.SMTPSender
GlideSMTPSenderJob	Packages.com.glide.job.SMTPSenderJob
GlideSOAPDocument	Packages.com.glide.communications.soap.SOAPDocument
GlideSOAPRequest	Packages.com.glide.communications.soap.SOAPRequest
GlideSOAPResponse	Packages.com.glide.communications.soap.SOAPResponse
GlideSOAPSecurity	Packages.com.glide.processors.soap.SOAPSecurity
GlideSOAPSigner	Packages.com.glide.communications.soap.SOAPSigner
GlideSocket	Packages.com.glide.script.proxy.Socket
GlidesoftGlideAttributesImpl	Packages.com.glidesoft.util.xml.GlideAttributesImpl
GlidesoftXMLMemoryTable	Packages.com.glidesoft.util.xml.XMLMemoryTable
GlideSQLChildMonitor	Packages.com.glide.monitor.sql.SQLChildMonitor
GlideSQLDebug	Packages.com.glide.ui.diagnostics.SQLDebug

GlideScriptable Class	Packages Call
GlideSQLDeleteMonitor	Packages.com.glide.monitor.sql.SQLDeleteMonitor
GlideSQLInsertMonitor	Packages.com.glide.monitor.sql.SQLInsertMonitor
GlideSQLResponseMonitor	Packages.com.glide.monitor.sql.SQLResponseMonitor
GlideSQLSelectMonitor	Packages.com.glide.monitor.sql.SQLSelectMonitor
GlideSQLUpdateMonitor	Packages.com.glide.monitor.sql.SQLUpdateMonitor
GlideSSHClient	Packages.com.glide.communications.SSHClient
GlideStack	Packages.com.glide.sys.GlideStack
GlideStatistician	Packages.com.glide.sys.stats.Statistician
GlideStatsInfo	Packages.com.glide.monitor.StatsInfo
GlideStatus	Packages.com.glide.util.GlideStatus
GlideStopWatch	Packages.com.glide.util.StopWatch
GlideStorageUtils	Packages.com.glide.db.meta.StorageUtils
GlideStringCache	Packages.com.glide.sys.cache.StringCache

GlideScriptable Class	Packages Call
GlideStringInputStream	Packages.com.glide.util.StringInputStream
GlideStringList	Packages.com.glide.collections.StringList
GlideStringUtil	Packages.com.glide.util.StringUtil
GlideSubQuery	Packages.com.glide.db.conditions.SubQuery
GlideSubstituteURL	Packages.com.glide.notification.substitution.SubstituteURL
GlideSummaryTableGroupReader	Packages.com.glide.ui.chart.dataSet.SummaryTableGroupReader
GlideSummaryTableOrderedReader	Packages.com.glide.ui.chart.dataSet.SummaryTableOrderedReader
GlideSummaryTableReader	Packages.com.glide.ui.chart.dataSet.SummaryTableReader
GlideSummaryTableWriter	Packages.com.glide.ui.chart.dataSet.SummaryTableWriter
GlideSynchronizedLRUCache	Packages.com.glide.sys.cache.SynchronizedLRUCache
GlideSysAttachment	Packages.com.glide.ui.SysAttachment
GlideSysAttachmentInputStream	Packages.com.glide.ui.SysAttachmentInputStream
GlideSysBRThreadMonitor	Packages.com.glide.monitor.threads.SysBRThreadMonitor
GlideSysChoice	Packages.com.glide.script.SysChoice

GlideScriptable Class	Packages Call
GlideSysConcurrencyMonitor	Packages.com.glide.monitor.threads.SysConcurrencyMonitor
GlideSysCPUThreadMonitor	Packages.com.glide.monitor.threads.SysCPUThreadMonitor
GlideSysDateUtil	Packages.com.glide.sys.util.SysDateUtil
GlideSysDBThreadMonitor	Packages.com.glide.monitor.threads.SysDBThreadMonitor
GlideSysField	Packages.com.glide.db.SysField
GlideSysFileUtil	Packages.com.glide.sys.util.SysFileUtil
GlideSysForm	Packages.com.glide.ui.SysForm
GlideSysForms	Packages.com.glide.ui.SysForms
GlideSysList	Packages.com.glide.ui.SysList
GlideSysListControl	Packages.com.glide.ui.SysListControl
GlideSysLog	Packages.com.glide.sys.SysLog
GlideSYSMany2Many	Packages.com.glide.db.SYSMany2Many
GlideSysMessage	Packages.com.glide.ui.SysMessage
GlideSysNetThreadMonitor	Packages.com.glide.monitor.threads.SysNetThreadMonitor
GlideSysRelatedList	Packages.com.glide.ui.SysRelatedList

GlideScriptable Class	Packages Call
GlideSysSection	Packages.com.glide.ui.SysSection
GlideSysSemaphore	Packages.com.glide.sys.util.SysSemaphore
GlideSystem	Packages.com.glide.script.GlideSystem
GlideSystemDateUtil	Packages.com.glide.script.system.GlideSystemDateUtil
GlideSystemUtil	Packages.com.glide.util.SystemUtil
GlideSystemUtilDB	Packages.com.glide.script.system.GlideSystemUtilDB
GlideSystemUtilScript	Packages.com.glide.script.system.GlideSystemUtilScript
GlideSysThreadMonitor	Packages.com.glide.monitor.threads.SysThreadMonitor
GlideSysUserList	Packages.com.glide.ui.SysUserList
GlideTable	Packages.com.glide.db.meta.Table
GlideTableChoiceList	Packages.com.glide.script.TableChoiceList
GlideTableCleaner	Packages.com.glide.misc.TableCleaner
GlideTableCleanerJob	Packages.com.glide.job.TableCleanerJob
GlideTableCreator	Packages.com.glide.db.impex.TableCreator

GlideScriptable Class	Packages Call
GlideTableDescriptor	Packages.com.glide.db.TableDescriptor
GlideTableGroupMover	Packages.com.glide.db.auxiliary.TableGroupMover
GlideTableManager	Packages.com.glide.db.TableManager
GlideTableMover	Packages.com.glide.db.auxiliary.TableMover
GlideTableParentChange	Packages.com.glide.db.table.TableParentChange
GlideTableParentColumnChange	Packages.com.glide.db.table.TableParentColumnChange
GlideTaskToken	Packages.com.glide.execution_plan.TaskToken
GlideTemplate	Packages.com.glide.script.Template
GlideTestAgent	Packages.com.glide.autotester.GlideTestAgent
GlideTextIndexEvent	Packages.com.glide.ts.event.TextIndexEvent
GlideThreadAttributes	Packages.com.glide.ui.GlideThreadAttributes
GlideThreadUtil	Packages.com.glide.util.ThreadUtil
GlideTime	Packages.com.glide.glideobject.GlideTime
GlideTimelineFrameSeparator	Packages.com.glide.schedules.TimelineFrameSeparator

GlideScriptable Class	Packages Call
GlideTimelineItem	Packages.com.glide.schedules.TimelineItem
GlideTimelineSpan	Packages.com.glide.schedules.TimelineSpan
GlideTomcatHelper	Packages.com.glide.startup.TomcatHelper
GlideTransaction	Packages.com.glide.sys.Transaction
GlideTransactionManager	Packages.com.glide.sys.TransactionManager
GlideTransferAuditDataHelper	Packages.com.glide.audit.TransferAuditDataHelper
GlideTransformer	Packages.com.glide.db.impex.Transformer
GlideTreePicker	Packages.com.glide.ui.TreePicker
GlideTSAccumulator	Packages.com.glide.ts.cluster.TSAccumulator
GlideTSAccumulatorProcessor	Packages.com.glide.ts.cluster.TSAccumulatorProcessor
GlideTSChainsHandler	Packages.com.glide.ts.trends.TSChainsHandler
GlideTSChainsLoader	Packages.com.glide.ts.trends.TSChainsLoader
GlideTSChainsPusher	Packages.com.glide.ts.trends.TSChainsPusher
GlideTSChainsSummarizer	Packages.com.glide.ts.trends.TSChainsSummarizer

GlideScriptable Class	Packages Call
GlideTSClusterDefinitions	Packages.com.glide.ts.cluster.TSClusterDefinitions
GlideTSDebug	Packages.com.glide.ts.util.TSDebug
GlideTSDidYouMean	Packages.com.glide.ts.util.TSDidYouMean
GlideTSGlobalKeywordSummarizer	Packages.com.glide.ts.trends.TSGlobalKeywordSummarizer
GlideTSIndexStatistician	Packages.com.glide.ts.stats.TSIndexStatistician
GlideTSIndexStopGenerator	Packages.com.glide.ts.stats.TSIndexStopGenerator
GlideTSIndexTables	Packages.com.glide.ts.indexer.TSIndexTables
GlideTSKeywordHandler	Packages.com.glide.ts.trends.TSKeywordHandler
GlideTSKeywordLoader	Packages.com.glide.ts.trends.TSKeywordLoader
GlideTSKeywordPusher	Packages.com.glide.ts.trends.TSKeywordPusher
GlideTSMoversViewer	Packages.com.glide.ts.cluster.TSMoversViewer
GlideTSSearchStatistician	Packages.com.glide.ts.stats.TSSearchStatistician
GlideTSSearchSummary	Packages.com.glide.ts.trends.TSSearchSummary

GlideScriptable Class	Packages Call
GlideTSTopSearches	Packages.com.glide.ts.util.TSTopSearches
GlideTSUtil	Packages.com.glide.ts.util.TSUtil
GlideTSVersion	Packages.com.glide.ts.TSVersion
GlideUIAction	Packages.com.glide.ui.action.UIAction
GlideUISession	Packages.com.glide.ui.Session
GlideUnloader	Packages.com.glide.db.impex.Unloader
GlideUpdateManager2	Packages.com.glide.update.UpdateManager2
GlideUpdateSet	Packages.com.glide.update.UpdateSet
GlideUpdateSetController	Packages.com.glide.system_update_set.UpdateSetController
GlideUpdateSetPreviewer	Packages.com.glide.update.UpdateSetPreviewer
GlideUpdateSetWorker	Packages.com.glide.update.UpdateSetWorker
GlideUpdateSyncher	Packages.com.glide.policy.UpdateSyncher
GlideUpdateTableChoiceList	Packages.com.glide.script.UpdateTableChoiceList
GlideUpgrade	Packages.com.glide.sys.Upgrade
GlideUpgradeArtifactManager	Packages.com.glide.misc.UpgradeArtifactManager

GlideScriptable Class	Packages Call
GlideUpgradeLog	Packages.com.glide.update.UpgradeLog
GlideUpgradeMonitor	Packages.com.glide.update.UpgradeMonitor
GlideURI	Packages.com.glide.ui.GlideURI
GlideURL	Packages.com.glide.util.GlideURL
GlideURLUTF8Encoder	Packages.com.glide.util.URLUTF8Encoder
GlideURLUtil	Packages.com.glide.util.URLUtil
GlideUser	Packages.com.glide.sys.User
GlideUserAuthenticator	Packages.com.glide.sys.UserAuthenticator
GlideUserGroup	Packages.com.glide.sys.UserGroup
GlideUtil	Packages.com.glide.util.GlideUtil
GlideViewRuleNavigator	Packages.com.glide.ui.ViewRuleNavigator
GlideWarDeleter	Packages.com.glide.misc.WarDeleter
GlideWarDownloader	Packages.com.glide.misc.WarDownloader
GlideWhiteListManager	Packages.com.glide.script.api.GlideWhiteListManager
GlideWikiModel	Packages.com.glide.wiki.GlideWikiModel

GlideScriptable Class	Packages Call
GlideWorkerThread	Packages.com.glide.worker.WorkerThread
GlideWorkerThreadManager	Packages.com.glide.sys.WorkerThreadManager
GlideWSClient	Packages.com.glide.util.WSClient
GlideWSDefinition	Packages.com.glide.wsdlreader.WSDefinition
GlideWSDLReader	Packages.com.glide.wsdlreader.GlideWSDLReader
GlideXMLChoiceListSerializer	Packages.com.glide.choice.XMLChoiceListSerializer
GlideXMLDocument	Packages.com.glide.util.XMLDocument
GlideXMLElementIterator	Packages.com.glide.util.XMLElementIterator
GlideXMLGlideRecordSerializer	Packages.com.glide.processors.xmlhttp.XMLGlideRecordSerializer
GlideXMLParameters	Packages.com.glide.util.XMLParameters
GlideXMLStats	Packages.com.glide.ui.XMLStats
GlideXMLSysMetaSerializer	Packages.com.glide.processors.xmlhttp.XMLSysMetaSerializer
GlideXMLUtil	Packages.com.glide.util.XMLUtil
GlideZipUtil	Packages.com.glide.util.ZipUtil
RhinoEnvironment	Packages.com.glide.script.RhinoEnvironment

GlideScriptable Class	Packages Call
RhinoHelper	Packages.com.glide.script.RhinoHelper
SecurelyAccess	Packages.com.glide.sys.util.SecurelyAccess
ServiceAPI	Packages.com.glide.service_api.ServiceAPI
SncAddress32Bit	Packages.com.snc.common.net.works.Address32Bit
SncAliasApplier	Packages.com.snc.field_normalization.AliasApplier
SncAppFiles	Packages.com.snc.apps.api.AppFiles
SncApplicationFileListener	Packages.com.snc.apps.db.ApplicationFileListener
SncAppsAccess	Packages.com.snc.apps.api.AppsAccess
SncAppsUI	Packages.com.snc.apps.api.AppsUI
SncASensor	Packages.com.snc.discovery.sensor.ASensor
SncAuthentication	Packages.com.snc.authentication.digest.Authentication
SncBaselineCMDB	Packages.com.snc.cmdb.BaselineCMDB
SncBulkCopy	Packages.com.snc.ha.clone.BulkCopy

GlideScriptable Class	Packages Call
SncClassifiedProcess	Packages.com.snc.discovery.sens or.ClassifiedProcess
SncClassify	Packages.com.snc.discovery.sens or.snmp.Classify
SncCloneController	Packages.com.snc.ha.clone.Clon eController
SncCloneInstance	Packages.com.snc.ha.clone.instan ce.Instance
SncCloneLogger	Packages.com.snc.ha.clone.Clon eLogger
SncCloneTask	Packages.com.snc.ha.clone.Clon eTask
SncCloneUtils	Packages.com.snc.ha.clone.Clon eUtils
SncConfiguration	Packages.com.snc.field_norma lization.db.Configuration
SncConfigurations	Packages.com.snc.field_norma lization.Configurations
SncConnectionTest	Packages.com.snc.ha.connectiv ity.ConnectionTest
SncDBChangeManagerFactoryHA	Packages.com.snc.db.clone.chan ge.DBChangeManagerFactoryHA
SncDeviceHistory	Packages.com.snc.discovery.loggi ng.DeviceHistory
SncDiscoveryCancel	Packages.com.snc.discovery.Disco veryCancel

GlideScriptable Class	Packages Call
SncDiscoveryClassification	Packages.com.snc.discovery.DiscoveryClassification
SncDiscoveryLog	Packages.com.snc.discovery.logging.DiscoveryLog
SncDiscoveryRanges	Packages.com.snc.commons.networks.DiscoveryRanges
SncDiscoveryRangesDB	Packages.com.snc.discovery.DiscoveryRangesDB
SncDiscoveryReconciler	Packages.com.snc.discovery.dataBase.DiscoveryReconciler
SncDiscoverySNMPClassification	Packages.com.snc.discovery.sensor.snmp.DiscoverySNMPClassification
SncDiscoveryUtils	Packages.com.snc.discovery.DiscoveryUtils
SncDropBackupTablesTask	Packages.com.snc.ha.clone.instance.DropBackupTablesTask
SncDropTablesTask	Packages.com.snc.ha.clone.DropTablesTask
SncEC2Properties	Packages.com.snc.ec2.EC2Properties
SncECMDBUtil	Packages.com.snc.cmdb.ECMDBUtil
SncElrondClient	Packages.com.snc.customer_logo.n.ElrondClient
SncExpert	Packages.com.snc.expert.Expert

GlideScriptable Class	Packages Call
SncExpertInstance	Packages.com.snc.expert.ExpertInstance
SncExpertPanel	Packages.com.snc.expert.ExpertPanel
SncExtantDataJob	Packages.com.snc.field_normalization.db.ExtantDataJob
SncExtantDataJobState	Packages.com.snc.field_normalization.ExtantDataJobState
SncFailoverController	Packages.com.snc.da.gateway.replication.FailoverController
SncFileTree	Packages.com.snc.apps.file.FileTree
SncGatewayCache	Packages.com.snc.da.gateway.GatewayCache
SncGatewayClone	Packages.com.snc.da.gateway.clone.GatewayClone
SncGatewayConnectivity	Packages.com.snc.da.gateway.GatewayConnectivity
SncGatewayPluginStartup	Packages.com.snc.da.gateway.replication.GatewayPluginStartup
SncGatewayTruncateHierarchy	Packages.com.snc.da.gateway.clone.GatewayTruncateHierarchy
SncGlideGateways	Packages.com.snc.da.gateway.GlideGateways
SncHAClone	Packages.com.snc.ha.clone.HAClone

GlideScriptable Class	Packages Call
SncHAConnectionTest	Packages.com.snc.ha.connectivity.HAConnectionTest
SncHADatabaseCheck	Packages.com.snc.ha.tablecheck.HADatabaseCheck
SncHAPairingUtils	Packages.com.snc.ha.HAPairingUtils
SncHAReplicationController	Packages.com.snc.ha.HAReplicationController
SncHAReplicationQueueSnapshotBuilder	Packages.com.snc.ha.tablecheck.HAReplicationQueueSnapshotBuilder
SncHTableCheck	Packages.com.snc.ha.tablecheck.HATableCheck
SncHTableCheckThread	Packages.com.snc.ha.tablecheck.HATableCheckThread
SncHTableQuickCheck	Packages.com.snc.ha.tablecheck.HATableQuickCheck
SncHTableRepair	Packages.com.snc.ha.tablecheck.HATableRepair
SncHAUtils	Packages.com.snc.ha.HAUtils
SncHostname	Packages.com.snc.discovery.utils.Hostname
SncInstanceClone	Packages.com.snc.ha.clone.instance.InstanceClone
SncInstanceConnectionTest	Packages.com.snc.ha.connectivity.InstanceConnectionTest

GlideScriptable Class	Packages Call
SncInstanceRollback	Packages.com.snc.ha.clone.instance.InstanceRollback
SncIPAddressV4	Packages.com.snc.common.net.works.IPAddressV4
SncIPAddressV6	Packages.com.snc.common.net.works.IPAddressV6
SncIPAuthenticator	Packages.com.snc.ipauthenticator.IPAuthenticator
SncIPIterator	Packages.com.snc.common.net.works.IPIterator
SncIPList	Packages.com.snc.common.net.works.IPList
SncIPMetaCollection	Packages.com.snc.common.net.works.IPMetaCollection
SncIPNetmaskV4	Packages.com.snc.common.net.works.IPNetmaskV4
SncIPNetworkV4	Packages.com.snc.common.net.works.IPNetworkV4
SncIPRangeV4	Packages.com.snc.common.net.works.IPRangeV4
SncJRobinGraphDef	Packages.com.snc.jrobin.JRobinGraphDef
SncLayer7Connections	Packages.com.snc.discovery.Layer7Connections
SncMACAddress	Packages.com.snc.common.net.works.MACAddress

GlideScriptable Class	Packages Call
SncMACAddressMfr	Packages.com.snc.common.net works.MACAddressMfr
SncMakeAndModel	Packages.com.snc.cmdb.MakeAn dModel
SncMIDConfigParameter	Packages.com.snc.common.MID ConfigParameter
SncMIDServerRangesDB	Packages.com.snc.discovery.MIDS erverRangesDB
SncNormalCoalescer	Packages.com.snc.field_norma lization.NormalCoalescer
SncNormalizer	Packages.com.snc.field_norma lization.Normalizer
SncNormalValueChanger	Packages.com.snc.field_norma lization.NormalValueChanger
SncNotifySNC	Packages.com.snc.system.NotifyS NC
SncOnCallRotation	Packages.com.snc.on_call_rotatio n.OnCallRotation
SncPendingValueCollector	Packages.com.snc.field_norma lization.PendingValueCollector
SncPreferences	Packages.com.snc.field_norma lization.Preferences
SncPrintServerHelper	Packages.com.snc.discovery.data base.PrintServerHelper
SncProbe	Packages.com.snc.discovery.Pro be

GlideScriptable Class	Packages Call
SncProbeRunTime	Packages.com.snc.discovery.perfmon.ProbeRunTime
SncRBSSensorProcessor	Packages.com.snc.discovery_auto_mation.RBSSensorProcessor
SncReadTest	Packages.com.snc.ha.ReadTest
SncReclassifyCI	Packages.com.snc.cmdb.ReclassifyCI
SncRelationships	Packages.com.snc.cmdb.Relationships
SncReplicationAdvisor	Packages.com.snc.db.replicate.ReplicationAdvisor
SncReplicationEngine	Packages.com.snc.db.replicate.ReplicationEngine
SncReplicationQueue	Packages.com.snc.db.replicate.ReplicationQueue
SncRequestCredentials	Packages.com.snc.customer_logo_n.RequestCredentials
SncRrdGlideBackendFactory	Packages.com.snc.jrobin.RrdGlideBackendFactory
SncRuleApplier	Packages.com.snc.field_normalization.RuleApplier
SncRuleToPending	Packages.com.snc.field_normalization.RuleToPending
SncSAMCounter	Packages.com.snc.software_asset_management.SAMCounter

GlideScriptable Class	Packages Call
SncScheduleDropBackupTablesTask	Packages.com.snc.ha.clone.instance.ScheduleDropBackupTablesTask
SncScrapeIANAEnterpriseNumbers	Packages.com.snc.discovery.data.base.ScrapeIANAEnterpriseNumbers
SncScrapeIEENICCodes	Packages.com.snc.discovery.data.base.ScrapeIEENICCodes
SncSendNotificationTask	Packages.com.snc.ha.clone.instance.SendNotificationTask
SncSensorProcessor	Packages.com.snc.discovery.SensorProcessor
SncSerialNumber	Packages.com.snc.discovery.SerialNumber
SncSerialNumberList	Packages.com.snc.discovery.SerialNumberList
SncSessionMate	Packages.com.snc.discovery.SessionMate
SncSimmerControl	Packages.com.snc.ha.clone.instance.SimmerControl
SncTableEditor	Packages.com.snc.apps.api.TableEditor
SncTableRotation	Packages.com.snc.db.replicate.TableRotation
SncTableRotationExtension	Packages.com.snc.db.replicate.TableRotationExtension
SncTableRotationExtensions	Packages.com.snc.db.replicate.TableRotationExtensions

GlideScriptable Class	Packages Call
SncTableRotationWatcher	Packages.com.snc.db.replicate.TableRotationWatcher
SncTransformApplier	Packages.com.snc.field_normalization.TransformApplier
SncTreeBuilder	Packages.com.snc.apps.tree.TreeBuilder
SncTriggerSynchronizer	Packages.com.snc.automation.TriggerSynchronizer
SncValue	Packages.com.snc.field_normalization.db.Value
TestExtension	Packages.com.glide.junit.misc.TestExtension
UINotification	Packages.com.glide.ui.UINotification

Packages Call Removal Tool error types

Possible error types generated by the Packages Call Removal Tool.

Error types

Error message	Description	Solution
Class Name Could Not BeParsed	A line of script has what appears to be a Packages call (for example, the line contains Packages.com.glide) but on examination there is insufficient text to parse out a class name that corresponds to a	This error type generally does not present a problem, but is nevertheless flagged as an error in case the script references a valid class that the parser, for whatever reason, cannot parse.

Error message	Description	Solution
	scriptable object name.	
Class Does Not Exist	A line of script has an identifiable Packages call to a Java class that does not exist. This may be because it was incorrectly typed, or because the Java class no longer exists. Either way, the line of script is currently not doing anything and is potentially generating errors whenever it is run.	This error type usually identifies a script that is not doing what its author intended, if it ever did. The original script should be revisited, and the invalid Packages call removed.
Class Is Not a Scriptable Class	A line of script has an identifiable Packages call to an existing ServiceNow Java class that is not currently marked as scriptable. The call cannot be replaced, because there is currently no corresponding scriptable name.	To convert this script, either rewrite the script so that it does not use the Java class or wait until ServiceNow, Inc., provides a scriptable name for the class.
Class Does Not Have a Scriptable Name	A line of script has an identifiable Packages call to an existing ServiceNow Java class that does not currently have a scriptable name.	This error is less likely to occur than a Class Is Not a Scriptable Class error; however, as with that error, to convert this script, either rewrite the script so that it does not use the Java class or wait until ServiceNow, Inc.,

Error message	Description	Solution
		provides a scriptable name for the class.
Variable Name Being Assigned Conflicts With a Scriptable Name	A variable name has the same name as a scriptable name. For example, var GlideSession = Packages.com.glide.sys.GlideSession.get(); would generate this error because the variable name GlideSession is the same as the scriptable name for the GlideSession Java class.	Before this script can be converted, the variable name must be changed wherever it occurs in the script. For this example, the line could be changed to var gsession = Packages.com.glide.sys.GlideSession.get(); to remove the conflict. Also, be sure to change the variable name elsewhere in the script wherever it is used.
Internal Error - Unable To Find String To Be Replaced	The tool is unable to find the script text it is currently evaluating for Packages call replacement.	This error type is very unlikely. If it does occur, examine your code and correct any errors. If the error message still occurs, open an incident with Technical Support.