

ServiceNow Script Include

Script Includes defines a function or class to be reusable server-side script logic that is executed only when explicitly called by other scripts.

Script Includes are called in Business Rule, Scheduled Job, Script Action, Reference Qualifier, UI Action, UI Pages, Workflow Script, Client Script (Glide Ajax) and in other script Include.

Script Includes are only loaded on request. This can be called by server-side scripts and client-side (Glide Ajax) scripts.

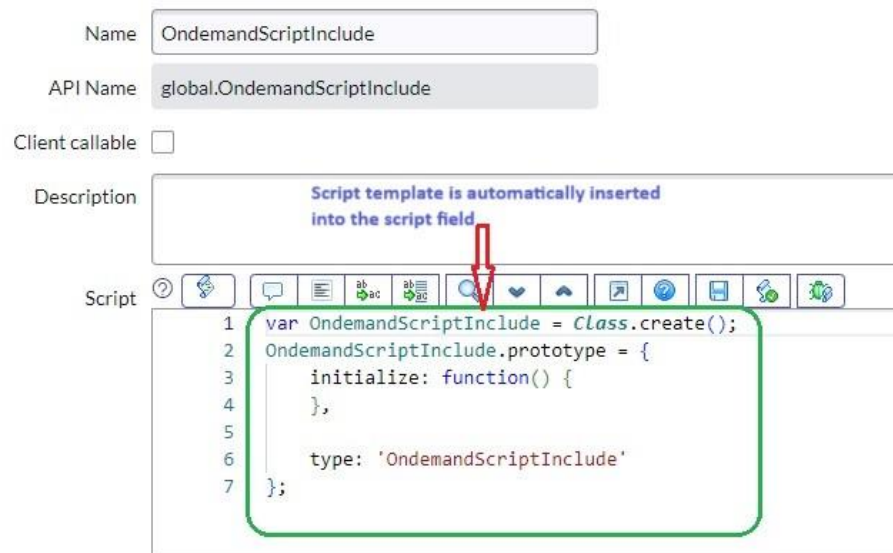
There are different types of Script Includes:

1. **On demand/classless**
2. **Extend an existing class**
3. **Define a new class**

1. On demand/classless:

On Demand or Classless Script Include stores only single function. The function is callable from other server-side scripts. On demand Script Includes can never be used client-side even if the Client callable option is selected.

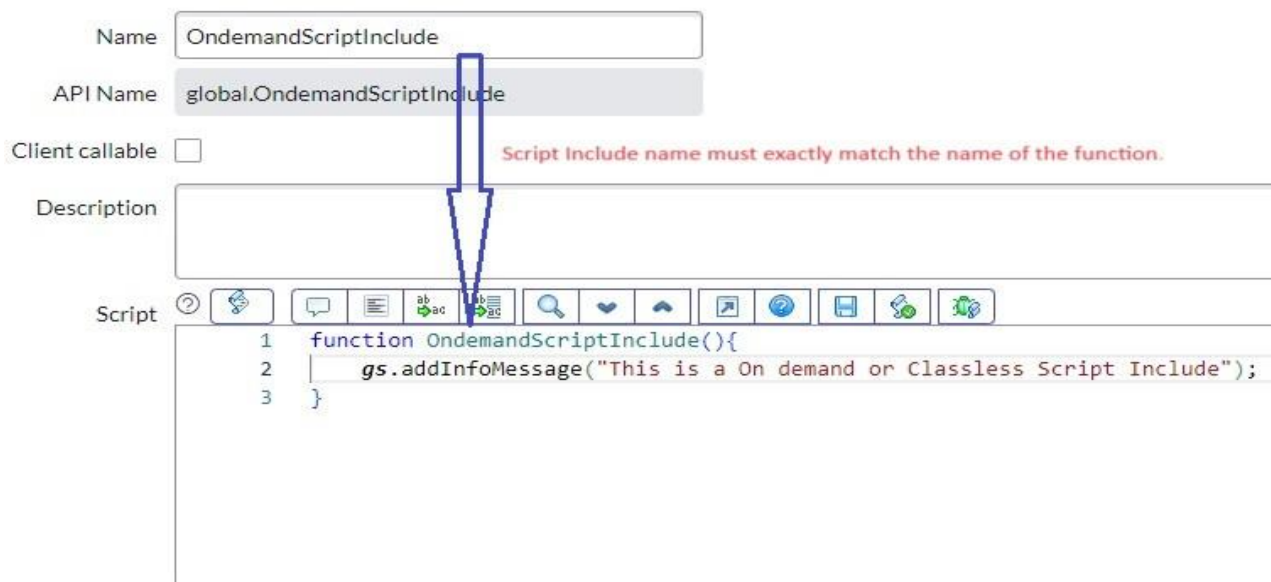
A script template is automatically inserted into the Script field



The screenshot shows the 'Script Include' form in ServiceNow. The 'Name' field is 'OndemandScriptInclude' and the 'API Name' is 'global.OndemandScriptInclude'. The 'Client callable' checkbox is unchecked. The 'Description' field contains the text 'Script template is automatically inserted into the script field'. The 'Script' field contains a JavaScript class template. A red arrow points to the top of the script field, and a green box highlights the template code.

```
1 var OndemandScriptInclude = Class.create();
2 OndemandScriptInclude.prototype = {
3     initialize: function() {
4     },
5
6     type: 'OndemandScriptInclude'
7 };
```

The template does not apply to on demand Script Includes. Delete the template and replace it with your function definition. The Script Include function is defined using standard JavaScript syntax.



The screenshot shows the configuration form for a Script Include named 'OndemandScriptInclude'. The 'Name' field is 'OndemandScriptInclude' and the 'API Name' is 'global.OndemandScriptInclude'. A red text annotation 'Script Include name must exactly match the name of the function.' points to the 'Name' field. The 'Client callable' checkbox is unchecked. The 'Description' field is empty. The 'Script' field contains the following code:

```
1 function OndemandScriptInclude(){
2   gs.addInfoMessage("This is a On demand or Classless Script Include");
3 }
```

On demand Script Includes are typically used when script logic needs to be reused. Examples include standardizing date formats, enabling/disabling logging, and validating email addresses.

2. Extend an existing class:

This creates a new class to store new functions.

This means creating a new Script Include/Class, reference an existing Class using the extendsObject() method to include all its functionality, add script logic.

Parent Script Include:



The screenshot shows the configuration form for a Script Include named 'parentScriptInclude'. The 'Name' field is 'parentScriptInclude' and the 'API Name' is 'global.parentScriptInclude'. The 'Application' dropdown is set to 'Global'. The 'Accessible from' dropdown is set to 'This application scope only'. The 'Client callable' checkbox is unchecked. The 'Active' checkbox is checked. The 'Description' field is empty. The 'Script' field contains the following code:

```
1 var parentScriptInclude = Class.create();
2 parentScriptInclude.prototype = {
3   initialize: function() {
4   },
5
6   showMessage: function(){
7     gs.addInfoMessage("Parent Script Include is Triggered");
8   },
9
10  type: 'parentScriptInclude'
11 };
```

Child Script Include:

After extending the class using Object.extendObject method, we expect all parent methods to be available to the child.

The screenshot shows the configuration page for a ServiceNow Script Include named 'childScriptInclude'. The API Name is 'global.childScriptInclude'. It is set to be 'Client callable' and 'Active'. The Application is 'Global' and it is 'Accessible from' 'This application scope only'. The Description field is empty. The Script field contains the following code:

```
1 var childScriptInclude = Class.create();
2 childScriptInclude.prototype = Object.extendsObject(parentScriptInclude, {
3
4     showChildMessage: function(){
5         gs.addInfoMessage("Child Script Include has been Triggered");
6     },
7
8     type: 'childScriptInclude'
9 });
```

A red arrow points from the text 'Parent Script Include' to the 'parentScriptInclude' parameter in the code.

Calling the script include functions:

```
new global.childScriptInclude().showChildMessage(); //showChildMessage()
function is present in child script include.
```

```
new global.childScriptInclude().showMessage(); //showMessage() function is
present in parent script include.
```

Output:

[0:00:00.027] Script completed in scope global: script

Script execution history and recovery [available here](#)

Background message, type:info, message: Child Script Include has been Triggered
Background message, type:info, message: Parent Script Include is Triggered

Although most ServiceNow classes are extensible, the most commonly extended classes are:

- GlideAjax: make AJAX calls from Client Scripts
- LDAPUtils: add managers to users, set group membership, debug LDAP
- Catalog*: set of classes used by the Service Catalog for form processing and UI building

3. Define a new class: This Script Include can be used for collection of functions. The standard practice is to include “Utils” in the name. Ex: ‘UserUtils’.

Utilities Script Includes typically define a new class and therefore use the automatically inserted script template.

The screenshot shows the configuration page for a Script Include named 'UserUtils'. The 'Name' field is 'UserUtils', 'API Name' is 'global.UserUtils', 'Application' is 'Global', and 'Accessible from' is 'This application scope only'. The 'Client callable' checkbox is unchecked, and the 'Active' checkbox is checked. The 'Description' field is empty. The 'Script' field contains the following JavaScript code:

```
1 var UserUtils = Class.create();
2 UserUtils.prototype = {
3   initialize: function() {
4   },
5
6   type: 'UserUtils'
7 };
```

The initialize function is automatically invoked when JavaScript objects are instantiated from the Script Include. Any variable defined as part of this object in the initialize function is known to all other functions in the Script Include.

This screenshot shows the same configuration page for 'UserUtils', but with a more complex script. The 'Script' field contains the following JavaScript code:

```
1 var UserUtils = Class.create();
2 UserUtils.prototype = {
3   initialize: function() {
4     this.number = 10;
5   },
6
7   testFunction: function(){
8     gs.addInfoMessage("The Number is - "+this.number);
9   },
10
11   type: 'UserUtils'
12 };
```