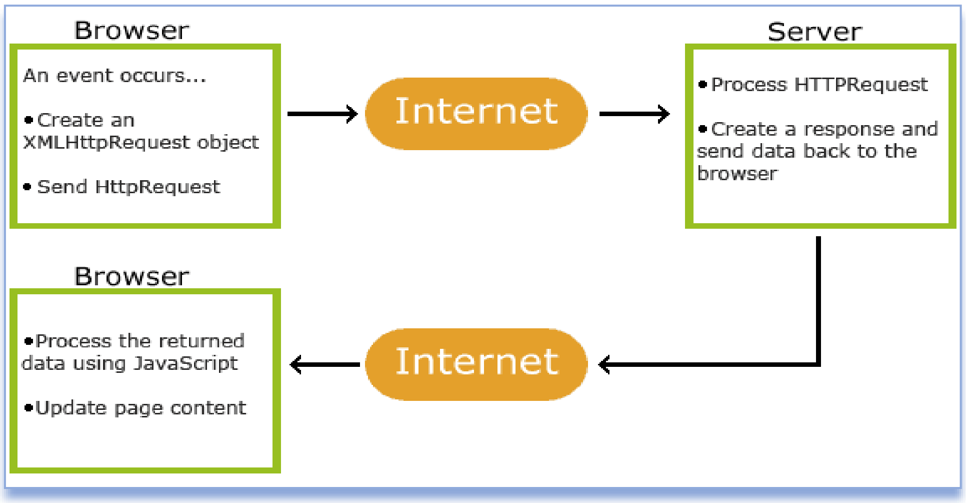
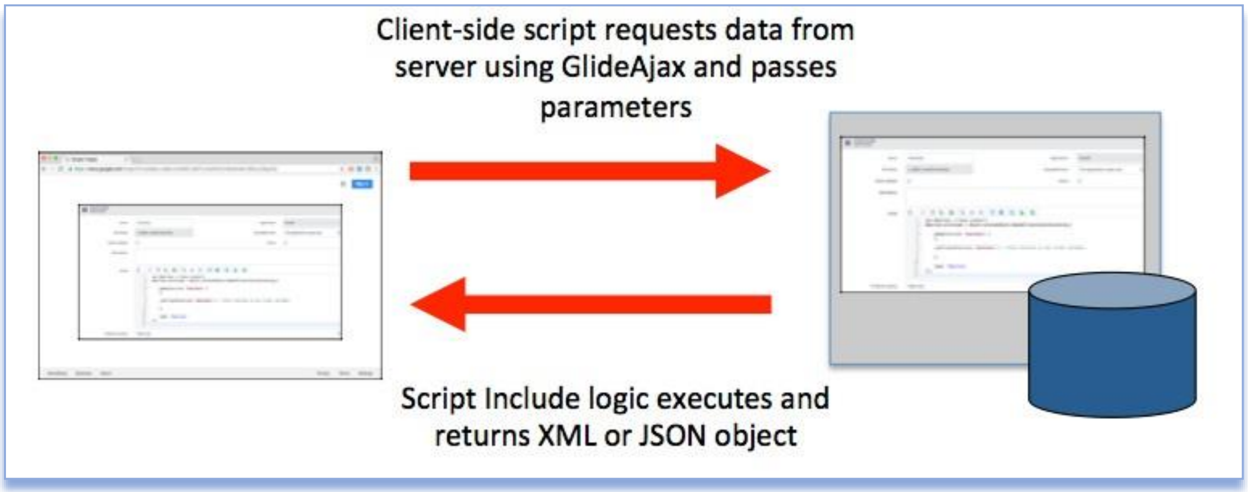
SERVICENOW - GLIDEAJAX

As you know **AJAX** means, Asynchronous JavaScript and XML. AJAX allows us to update part of our web page asynchronously by exchanging small amounts of data with the server, that's exchange occurs behind the scene instead of reloading the whole form.


Browser
An event occurs...
• Create an XMLHttpRequest object
• Send HttpRequest
Internet
Server
• Process HTTPRequest
• Create a response and send data back to the browser
Browser
• Process the returned data using JavaScript
• Update page content
Internet

What Glide Ajax is, essentially just a way to construct an AJAX call which triggers a client-callable Script Include on the ServiceNow servers, which then returns some value(s) back to the client.

I say Values, because the response will be in XML/ JSON format and can contain multiple values.


Client-side script requests data from server using GlideAjax and passes parameters
Script Include logic executes and returns XML or JSON object

GlideAjax is **not** just a replacement for the GlideRecord API. However, it can replace that functionality (and more) on the client, by **asking the server to do most of the work**. In this way, GlideAjax can be a **massive boon to your client-side performance**

There is some parameter which are predefined.

Sysparam_name: GlideAjax uses this to find which function to use.

1. Sysparam_function
2. Syparam_value
3. Sysparam_type

Avoid these when you are passing any extra parameters.

The Code is then executed with getXML () or getXMLWait () functions.

There are 2 Types of GlideAjax,

1. Synchronous GlideAjax
2. Asynchronous GlideAjax

Synchronous GlideAjax

Use synchronous GlideAjax only when you cannot continue without the GlideAjax Response. This will stop the session till response is received.

You can use getXMLWait(), if you need to wait till you receive the response and this function does not need a separate callback function.

But Recommendation will be to use getXML() with a callback function to improve the responsiveness of the application.

Note : getXMLWait() method is not available in scoped applications.

```
Client Script :


var ga = new GlideAjax('HelloWorld') ;
ga.addParam('sysparm_name','helloWorld');
ga.addParam('sysparm_user_name',"Bob");
ga.getXMLWait();
alert(ga.getAnswer());
```

```
 Script Include :
var HelloWorld = Class.create();
HelloWorld.prototype = Object.extendsObject(AbstractAjaxProcessor, {
helloWorld: function() { return "The Server Says Hello " + this.getParamet
er('sysparm_user_name') + "!"; } } );
```

ASynchronous GlideAjax

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.

Example :

```
Client Script :


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); }
```

```
Script Include :
var HelloWorld = Class.create();
HelloWorld.prototype = Object.extendsObject(AbstractAjaxProcessor, {
   helloWorld:function() { return "Hello " + this.getParameter('sysparm_us
er_name') + "!"; } ,
//_privateFunction: function() { // this function is not client callable

    }
 });
```

What you have seen till now, server is processing and returning a single value. Now we will look into an example to retrieve multiple data from server and returning them as a JavaScript (JSON) Object.

Script Include:

```
var MyClass=Class.create();
MyClass.prototype = Object.extendsObject(global.AbstractAjaxProcessor, {
getRecord : function() {
var res = {};
res.number = "INC101";
res.description = "Demo testing";
var result = new JSON().encode(res); // convert object to JSON string
return result;
},
 type: 'MyClass'
});
```

Client Script:

```
function onLoad() {
 getDetails();
//local function ==> calling GlideAjax
function getDetails(){
var _grInc = new GlideAjax("MyClass");
_grInc.addParam("sysparm_name","getRecord");
_grInc.getXML(getDetailCallBack);
}
function getDetailCallBack(glideAjax){
var res = glideAjax.responseXML.documentElement.getAttribute("answer");
//Processing data
var result = JSON.parse(res);
var _number = result.number;
var _description = result.description;
alert("Number is : "+_number+"& Description is : "+_description);}}
```

The combination of GlideAjax and JSON is transforming JavaScript objects between server and client very efficiently.

We use only two basic JSON function.

1. JSON.encode() → To encode the string into JSON format
2. JSON.parse() → To decode the JSON format string into its native data type

Troubleshooting Steps

SCRIPT INCLUDE

- ✓ Make sure that , Object.extendsObject(AbstractAjaxProcessor) this is there in your script include.

  The AbstractAjaxProcessor Script Include provides a standard set of functions that every GlideAjax Script Include must implement. The interface basically acts as a mediator that allows extracting data from the XML request and building the XML response that GlideAjax relies upon for messaging between client and server.

  No AbstractAjaxProcessor, no XML messages, no GlideAjax. Make sure a line like this exists in your Script Include.

- ✓ Do not use an initialize function. This may be less obvious, but do you notice that there is no initialize function in the Script Include examples above. This is actually important. The AbstractAjaxProcessor has its own initialize function and creating this function on your inheriting Script Include overrides the base class initialize function. It has a similar result to not inheriting the AbstractAjaxProcessor at all.
  Normally, when using class inheritance like this, you would explicitly call the base class function.

- ✓ Make sure that you have selected the checkbox "Client Callable". If there is not a check in the Client callable checkbox, then GlideAjax doesn't know the Script Include exists. This checkbox is a security measure and prevents malicious users from being able to call any Script Include from the client using browser dev tools.
  As a bonus, go ahead and check Client callable as your first step. When you follow up by tying in a Name, ServiceNow will automatically script a basic GlideAjax Script Include. This will take care of both the previously mentioned initialize function and AbstractAjaxProcessor issues.

- ✓ Use Public Function. GlideAjax can only use public functions from the client script. The distinguishing feature is the presence or absence of the underscore (_) at the front of the function name. Public functions will start with a letter, private functions will start with an underscore.

- ✓ Set Return Values . GlideAjax has two ways of setting the answer values in the message that gets returned to the client. The first way is to simply use a "return" statement as done in the helloWorld function above on line 4. This will set the "answer" parameter for the Client Script to consume.
  The other approach is to use the this.newItem and the setAttribute function calls in order to set multiple answer values. You can see this approach in the below Script Include from the ServiceNow Wiki.

```
/*
 * MyFavoritesAjax script include Description - sample AJAX processor retu
rning multiple value pairs
 */
var MyFavoritesAjax = Class.create();
MyFavoritesAjax.prototype = Object.extendsObject(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 meth
ods of AbstractAjaxProcessor
    },

    _addFavorite : function(name, value) {
        var favs = this.newItem("favorite");
        favs.setAttribute("name", name);
        favs.setAttribute("value", value);
    },
    type : "MyFavoritesAjax"});
```

Either way, it is important to set a return value in your publicly accessed function. In most cases, the simple return value is all you need.

✓ Mark script Include Public if needed. When accessing a GlideAjax Script Include from a public UI Page or Public CMS page, you also need to make your Script Include public. This is accomplished by setting the isPublic attribute on the Script Include object as shown below. This is another one of those security features to protect your Script Includes from malicious users. Without it, any user on the internet could use your GlideAjax script. In other words, be careful about which Script Includes you make public and test the security of those scripts and the tables that they access.

```
var HelloWorld = Class.create();
HelloWorld.prototype = Object.extendsObject(AbstractAjaxProcessor, {
   helloWorld: function() {
      return "Hello " + this.getParameter('sysparm_user_name') + "!";
   },
   _privateFunction: function() { // this function is not client callable

   },
   isPublic: true
});
```

✓ Include logging in script includes for better troubleshooting . I would like you to follow WIKI on this .

CLIENT SCRIPT

```
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);
}
```

- ✓ Correct Script Include Name. make sure you use the correct Script Include name on the first line of your GlideAjax call in your Client Script. In line 1 above, you see 'HelloWorld' in the first and only parameter to the GlideAjax constructor function. This is used by GlideAjax to find the correct Script Include server side so if you spell it wrong, you won't get your desired results.
- ✓ Correct Function Name as sysparm_name. Likewise, you have to have something like line 2 in the above Client Script. That is, you need to add a parameter called sysparm_name which should map to the public function you want to call on the Script Include. In this case we are calling helloWorld on the Script Include. GlideAjax has been fairly silent when it fails to find the correct Script Include or function on that Script Include. It just assumes you meant to do that and keeps on going with an empty or undefined response. So, double check the function call and spelling of the parameters passed, it can be as simple as that.
- ✓ Remember Call Back. you can do GlideAjax synchronously or asynchronously. Async GlideAjax is what you will want to use in most circumstances and to do that you have to pass a callback function. You can see this in the above Client Script in both the getXML function and the function named HelloWorldParse. When getXML retrieves the information from the server, HelloWorldParse will be called with the results. Notice the parameter "response" in the HelloWorldParse function. Don't forget that.
- ✓ Treat your results as a primitive data type. on the line where you see response.responseXML.documentElement.getAttribute('answer'), remember that the result of that function is not a complex object. You may have a GlideRecord server side, but in the Client Script you have a string, number, Boolean, null, or undefined value.
  That means there is no reason to do getAttribute('answer').doSomething(). There is no doSomething, it doesn't exist. The reason is the hidden XML message under the hood of GlideAjax. To communicate between the Client Script and Script Include, an XML message is used. Since XML doesn't understand GlideRecord or other complex objects, any value you pass will be converted to a primitive type.