ServiceNow

Technical Best Practices

Make code Easy to read:

- **Comment your code** – So when colleagues refer the script, they will take less time to work with it.

```
var gr = new GlideRecord('sys_user');
gr.addQuery('sys_id', sys);
gr.query();
if (gr.next()) {
    var obj =
        {
            "email": gr.email.toString(),
            "userid": gr.userid.toString(),
            "location": gr.location.toString()
        };
    return JSON.stringify(obj); //storing UserInfo in JSON format
```

- **Use White space** – Leave space in between to make it more user friendly.

```
userdata.prototype = Object.extendsObject(AbstractAjaxProcessor, {

    getuserinfo: function() {
        var sys = this.getParameter('sysparm_sysid');

        var gr = new GlideRecord('sys_user');
        gr.addQuery('sys_id', sys);
        gr.query();
        if (gr.next()) {

            var obj =
                {
                    "email": gr.email.toString(),
                    "userid": gr.userid.toString(),
                    "location": gr.location.toString()
                };
            return JSON.stringify(obj); //storing UserInfo in JSON format

        }

    },
```

- **Write simple statement** – Using JSON is pretty straightforward and easy to understand.

```
var gr = new GlideRecord('sys_user');
gr.addQuery('sys_id', sys);
gr.query();
if (gr.next()) {

    var obj =
        {
            "email": gr.email.toString(),
            "userid": gr.userid.toString(),
            "location": gr.location.toString()
        };
    return JSON.stringify(obj); //storing UserInfo in JSON format

}
```

Construct reusable functions - We use User information multiple times while scripting, so its better to write a reusable function in script include and use it whenever needed instead of writing the logic all over again.

```
var gr = new GlideRecord('sys_user');
gr.addQuery('sys_id', sys);
gr.query();
if (gr.next()) {

    var obj =
        {
            "email": gr.email.toString(),
            "userid": gr.userid.toString(),
            "location": gr.location.toString()
        };
    return JSON.stringify(obj); //storing UserInfo in JSON format

}
```

Variables

- **Use descriptive variable and function** names - Using the meaningful variable name and function name will help the readers to understand the purpose of code.

```
getUserInfo: function() {
    var sysId = this.getParameter('sysparm_sysid');

    var grUser = new GlideRecord('sys_user');
    grUser.addQuery('sys_id', sysId);
    grUser.query();
    if (grUser.next()) {

        var obj =
            {
                "email": grUser.email.toString(),
                "userid": grUser.userid.toString(),
                "location": grUser.location.toString()
            };
        return JSON.stringify(obj); //storing UserInfo in JSON format

    }

},
```

- **Use Variables to store function results** – When possible, use a descriptive variable to store a value and refer to the variable instead of calling the same function repeatedly

```
getGroup: function() {

    var GroupName =  this.getParameter('sys_gname');

    var grGroup = new GlideRecord('sys_user_group');
    grGroup.addQuery('name', GroupName);
    grGroup.query();
    if (grGroup.next()) {

        var assignGroup = grGroup.sys_id;
        return assignGroup;
    }

}
```

- **Verify value exist before using them** – To avoid unpredictable results and warning messages, verify that variables and fields have a value before using them.

```
if (newValue) {

    var ga = new GlideAjax('userdata');
    ga.addParam('sysparm_name', 'getUserInfo');
    ga.addParam('sysparm_sysid', newValue);
    ga.getXMLAnswer(answer);
    alert('ans: ' + answer);

}
```

- **Return Values** - Get in the practice of returning some type of value when you create new functions. The value returned can tell the calling code useful information about how the function executed.

Avoid Complex GlideRecord Queries – Instead of creating a lot of addQuery(), can use addEncodedQuery() to get the desire result without any complication.

```
var gaInc = new GlideRecord('incident');
gaInc.addEncodedQuery('active=true^category=inquiry^sys_created_on<=javascript:gs.endOfToday()');
gaInc.query();
while(gaInc.next()){

gs.print('Number: ' + gaInc.number);

}
```

Interacting with the Database

- **Use GIldeAggregate for simple record** counting – Instead of using GlideRecord for counting, can use GlideAggregate function which is meant mathematical queries.

```
var gaInc = new GlideAggregate('incident');
gaInc.addAggregate('COUNT');
gaInc.addInactiveQuery();
gaInc.query();
if(gaInc.next()){

    var totalCount = gaInc.getAggregate('COUNT');
    gs.print('Number of Inactive Incident: ' + totalCount);

}
```

- **Let the database do the work** - Whenever possible, leverage the power of the database to retrieve the proper records. Whenever using query() method, it will retrieve all those records

which can take time. But instead use setLimit() to return if only some certain number of records needed.

```
var gaInc = new GlideRecord('incident');
gaInc.addInactiveQuery();
gaInc.setLimit(1);
gaInc.query();
if(gaInc.next()){

    var incNo = gaInc.number;
    gs.print('Number: ' + incNo);

}
```

Avoid Coding Pitfalls

- **Experiment in Sandbox Instance** - Trying out new coding ideas or concepts in a development instance can lead to messy results later on. If you make changes, then alter your approach while using an update set, you may find unwanted changes getting in the update set and being promoted to other instances. If you do not use an update set for your proof of concept, your development instance may behave differently than your other instances, making it difficult to verify defects and fixes.
- **Code in stages** - Do not attempt to write hundreds of lines of code in one sitting. This is especially true if you are learning a new technology. Write a few lines of code at a time and test as you go to ensure it works as expected.
- **Do not use Hard-coded Values** - Avoid using hard-coded values in scripts, as they can lead to unpredictable results and can be difficult to track down later. Instead, try looking up a value by reference or by creating a property and retrieving the value with gs.getProperty().

```
var gname = gs.getProperty('incident.assignment.group');

if(current.assignment_group == gname){

gs.eventQueue('testing.testing',current, current.category, current.caller_id, "Customer Service");

}
```

- **Avoid Dot-Walking to the sys_id of a reference Field** - The value of a reference field is a sys_id. When you dot-walk to the sys_id, the system does an additional database query to retrieve the caller_id record, then retrieves the sys_id. This can lead to performance issues. Instead, use the statement.

```
var id = current.getValue('caller id');
```

- **Use getDisplayValue() Effectively** - Instead of using name, number, or other explicit field name used for the display value, use the method getDisplayValue().

```
var id = current.caller_id.getDisplayValue();
```