ServiceNow Coding Best Practices

GLEN CHIASSON, MARCH 24TH, 2020

West Coast Workflow

What are coding best practices?

- A set of guidelines and standards that are recommended
- General term consisting of many different components such as:
  - Conventions
  - Programming styles
  - Appearance
  - Architecture
  - Code standards 

Why do best practices matter?

- Recommended guidelines to improve maintainability
- Especially beneficial for teams within an organization
- Equally beneficial if you are a solo developer

- Consistency
- Helps other maintain and contribute
- Instance performance

- Following examples are common. Certainly not a list of every standard
- Best practices can change and evolve

General Scripting

- Comment complex code

```
/***
  * getRiskEnvironment - determine the environment of the CIs to calculate the risk
  *
  * @param: chg - GlideRecord of change record
  * @return: result - Highest environment value of CIs: 1=production, 2=QA, 3=test, 4=dev, 0=error
  *
  ***/
```

General Scripting

- White Space – Which is easier to read?

```
// Example of poor white space usage
function createRelationship(relType,childCI,parentCI){
var newRel=new GlideRecord('cmdb_rel_ci');
var newRelType=new GlideRecord('cmdb_rel_type');
if(childCI==parentCI)
return;
var relTypeID;
if(newRelType.get(relType)){
relTypeID=newRelType.getValue('sys_id');
newRel.initialize();
newRel.setValue('type',relTypeID);
newRel.setValue('child',childCI.getValue('sys_id'));
newRel.setValue('parent',parentCI.getValue('sys_id'));
newRel.insert();
}
}
```

```
// Example of good white space usage
function createRelationship(relType, childCI, parentCI) {
// Put spaces between parameters and before curly brace

    var newRel     = new GlideRecord('cmdb_rel_ci');
// Put spaces around '='.
    var newrelType = new GlideRecord('cmdb_rel_type');
// Optionally, put additional spaces to align '=' w/previous line.
    var relTypeID;
// Group variables together where appropriate

    if (childCI == parentCI)
// Put a space after 'if' and around operators ('!=')
        return;

    if (newRelType.get(relType)) {
// Use extra blank lines to help break up chunks of code and improve readability

        relTypeID = newRelType.getValue('sys_id');

        newRel.initialize();
// Group similar statements together
        newRel.setValue('type', relTypeID);
        newRel.setValue('child', childCI.getValue('sys_id'));
        newRel.setValue('parent', parentCI.getValue('sys_id'));
        newRel.insert();
    }
}
```

General Scripting

- Descriptive variables and function names

```
// Poor function and variable names
function del(r, d, s) {

    var a=0;

    if (s == 13) // 13=cancel/delete
        r.deleteRecord();
    else
        a = d;

    return a;
}
```

```
function deleteIfCanceled(glideRec, defaultAnswer, stateValue) {

    var answer = 0;

    if (stateValue == 13)
        glideRec.deleteRecord();
    else
        answer = defaultAnswer;

    return answer;
}
```

General Scripting

- Variables for function results

```
if (gs.getUserID() == current.assigned_to ||
    gs.getUserID() == current.u_coordinator ||
    gs.getUserID() == current.caller_id ||
    gs.getUserID() == current.caller_id.manager) {
    // Do something important here
}
```

```
var currentUser = gs.getUserID();
var isOwner = (currentUser == current.assigned_to);
var isCoordinator = (currentUser == current.u_coordinator);
var isCaller = (currentUser == current.caller_id);
var isManager = (currentUser == current.caller_id.manager);

if (isOwner || isCoordinator || isCaller || isManager) {
   // Do some important processing here
```

General Scripting

- Coding Pitfalls
  - Use a demo instance to test new concepts
  - Write a few lines of code at a time and test as you go
  - Avoid hard coding

```
var taskID = gs.getProperty('acme.default.task');
var groupName = gs.getProperty('acme.group.name');
```

```
var taskID = '26c811f0607538806d07268c841dcd0';
var groupName = 'Service Desk';
```

General Scripting

- Avoid dot walking to a sys_id of a reference field
  - Adds an additional query to retrieve the entire caller record which can result in performance issues
  - Use getValue()

```
var id = current.caller_id.sys_id;
```

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

Script Includes

- Single spot for maintenance
- Reusable for common functionality
- Can be called from Business Rules, Workflow, Client Scripts, Reference Qualifiers
- When retrieving database values client side, always use a script include

```
current.watch_list = new AcmeIncidentFunctions().addGlideListElement(current.watch_list, gs.getUserID());
```

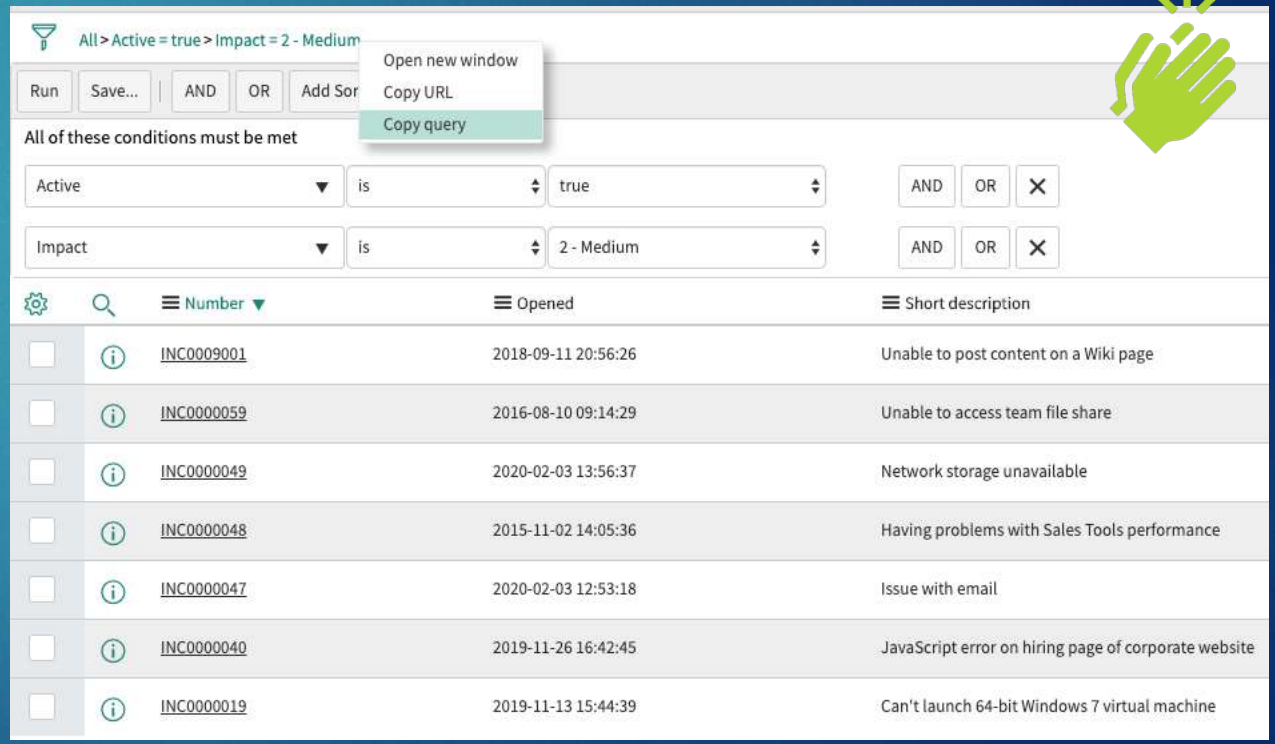
Database Interaction

- Avoid complex GlideRecord Queries
  - Use addEncodedQuery() to make the query easier to create and maintain
  - Create the query using a table record filter
  - Copy the query and assign it to a variable

```
var incQueryString = 'active=true^impact=2';
var incidentGR = new GlideRecord('incident');

incidentGR.addEncodedQuery(incQueryString);
incidentGR.query();

while (incidentGR.next()) {
    // do something
}
```



Database Interaction

- Counting records
  - Use GlideAggregate for simple record counting it is much more efficient due to built in functionality
  - Using getRowCount() can become troublesome as table size grows over time because every record is retrieved, then summed

```
/*
 * countInactiveIncidents - return the number of closed incidents
 *
 * @param - none
 * @returns integer - number of records found
 *
 */
function countInactiveIncidents() {

    var inc = new GlideRecord('incident');

    inc.addInactiveQuery();
    inc.query();

    var count = inc.getRowCount();
    gs.print(count + ' inactive incidents found');

    return count;
}
```

```
/*
 * countInactiveIncidents - return the number of closed incidents
 *
 * @param - none
 * @returns integer - number of records found
 *
 */
function countInactiveIncidents() {

    var inc = new GlideAggregate('incident');

    inc.addAggregate('COUNT')
    inc.addInactiveQuery();
    inc.query();

    var count = 0;
    if (inc.next())
        count = inc.getAggregate('COUNT');

    gs.print(count + ' inactive incidents found');

    return count;
}
```

Database Interaction

- Utilize setLimit() where possible
  - An unrestricted query will retrieve and process each record in a table
  - Ask yourself if you need to count ALL records? Or can I limit?

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

inc.addQuery('active', true);
inc.query();

if (inc.hasNext())
   // There is at least one active record
```

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

inc.addQuery('active', true);
inc.setLimit(1);  // Tell the database to only retrieve a maximum of 1 record
inc.query();

if (inc.hasNext())
   // There is at least one active record
```

Database Interaction

- When the record is known do not query the entire table
  - Retrieve the single record

```
var currentRecord = current.sys_id;

var incidentGR = new GlideRecord('incident');
incidentGR.query();

while (incidentGR.hasNext()) {
    if (incidentGR.sys_id == currentRecord) {
        //do something
    }
}
```

```
var incidentGR = new GlideRecord('incident');
incidentGR.get(sys_id_of_record_here);
//Do something with the record returned
if (incidentGR.active == true) {
    //do something
}
```

Client Scripts

- Best use to validate user input
  - Use UI policies where possible (read only fields)

```
if (g_form.getValue('impact') == '3' && g_form.getValue('priority') == '1') {
    g_form.showErrorBox('impact', 'Low impact not allowed with High priority');
}
```

Client Scripts

- Code in functions
  - When a variable or object is not enclosed, all client scripts have access to it
  - Can lead to unexpected results that can be extremely difficult to find

```
var state = "6";

function onSubmit() {

   if(g_form.getValue('incident_state') == state) {
        alert("This incident is Resolved");
   }
}
```

```
function onSubmit() {

   var state = "6";

   if(g_form.getValue('incident_state') == state) {
        alert("This incident is Resolved");
   }
}
```

Client Scripts

- Retrieving database values for the form
  - Use a Display Business Rule to send information
  - g_scratchpad object provides a mechanism for passing info from server to client

```
g_scratchpad.css = gs.getProperty('css.base.color');
g_scratchpad.hasAttachments = current.hasAttachments();
g_scratchpad.managerName = current.caller_id.manager.getDisplayValue();
```

```
var manager = g_scratchpad.managerName;
```

Client Scripts

- Retrieving database values for the form
  - Use asynchronous GlideAjax to avoid freezing the browser during data transfers
  - Never use GlideRecord in a client script

```
function onChange(control, oldValue, newValue, isLoading) {
    if (isLoading) {
        return;
    }
    if(newValue == ''){
        g_form.setValue('XXXXXXXXXX','');
        g_form.setValue('XXXXXXXXXX','');
        g_form.setValue('XXXXXXXXXX','');
        return;
    }
    if(newValue != oldValue){
        // refrence field
        var userSysId = g_form.getValue('XXXXXXXXXX');
        // Ajax include name below
        var ajax = new GlideAjax('XXXXXXXXXX');
        //Ajax function name below
        ajax.addParam("sysparm_name", "userValues");
        ajax.addParam('sysparm_sys_id', userSysId);
        ajax.getXML(doSomething);
    }
}
function doSomething(response){
    var values = response.responseXML.documentElement.getAttribute('answer').toString().split('|');
    //assign the values to the fields values[i] can as much as you pass the include function
    g_form.setValue('XXXXXXXXXX',values[0]);
    g_form.setValue('XXXXXXXXXX',values[1]);
    g_form.setValue('XXXXXXXXXX',values[2]);

}
```

```
userValues: function() {
    //assign the passing sys_id
    var sysID = this.getParameter('sysparm_sys_id');
    var gr = new GlideRecord('xxTABLE NAMExx');
    gr.addQuery('sys_id', sysID);
    gr.query();
    if (gr.next()) {
        return gr.xxfield - namexx + '|' + gr.xxfield - namexx + '|' + gr.xxfield - namexx;
    }
    return '';
},
```

Business Rules

- Know when they run and when to use them
  - Run server side so use where possible
  - Use conditions
  - Enclose in functions
  - Do not use current.update()
  - Avoid setAbortAction, use gs.addErrorMessage

display	Use to provide client-side scripts access to server-side data.
before	Use to update information on the current object. For example, a Business Rule containing current.state=3; would set the State field on the current record to the state with a value of 3.
after	Use to update information on related objects that need to be displayed immediately, such as GlideRecord queries.
async	Use to update information on related objects that do not need to be displayed immediately, such as calculating metrics and SLAs.

Health Scans

- ServiceNow, Vendors, Other Developers will be examining your code
- Security, upgradeability, performance, manageability and usability
  - Log statements
  - .getRowCount()
  - current.update()
  - gr = new GlideRecord()
  - isLoading() and return statements

Thank You!

- Initially appears like a hurdle, can be a struggle to get others onboard
  - Will ultimately save man hours and make code easier to manage and maintain for all

West Coast Workflow