CREATORCON™

enable the service revolution

Advanced GlideRecord Scripting

Steven Bell
SR. IMPLEMENTATION SPECIALIST
Accenture Technology

Mark Amann
SR. IMPLEMENTATION SPECIALIST
Accenture Technology

accenture technology
A ServiceNow Master Partner

servicenow

Agenda

Introduction

GlideRecord Scripting Intro

Advanced GlideRecord Scripting

Advanced Techniques

Conclusion

Speaker Intros

Steven Bell

Application Architect

Author on ServiceNow Community and CloudSherpas sites

Title: Senior Implementation and Training Specialist, CloudSherpas - ServiceNow Master Partner

Accomplishments:

Over 30 years development and architecture experience, focus includes CMDB, Discovery, Orchestration, WebServices, and scripting.

Education/Certifications: ServiceNow Certified Instructor, Application Developer, Implementation Specialist, Administrator. Microsoft Certified Solutions Developer. BS in Computer Science

Contact: steven.bell@cloudSherpas.com

Speaker Intros

Mark Amann

Title: Senior Implementation and Training Specialist, CloudSherpas - ServiceNow Master Partner

Accomplishments: Over 20 years experience in the computer industry. 5 years Service Now experience. Primary focus on complex integrations, Discovery, Orchestration and scripting.

Education/Certifications: Service Now Certified System Admin, Service Now Certified Implementation Specialist, ServiceNow Certified Application Developer, Service Now Certified Trainer, ITIL v3 Certified.

Contact: mark.amann@cloudsherpas.com

GlideRecord Scripting Intro

- What is a GlideRecord?
  - Way of accessing data in a table via code
  - Tables make up a Database
  - Used for database operations instead of writing SQL Queries
  - GlideRecord is a object array with a sealed structure (immutable)
  - What we will be teaching works in Fuji, Geneva, and Helsinki
  - **Our focus: Programming the GlideRecord Object itself**

Definitions

- Method – a function inside an object
- Property – a variable inside an object that is externally available
- Instantiate – bring into existence. New!
- Immutable – cannot ever be changed. String, Integer, Boolean, *GlideRecord*!
- Mutable – can be changed. Object
- Refactor – Restructuring the internal functionality without changing the external behavior

Advanced GlideRecord Scripting Intro

- Loop Refactoring
- Dot Notation – Maintainability
- Unit Testing Techniques
- Encoded Queries
- Extending the GlideRecord Object

Best Practices – Loop Refactoring

- Bad Practice

  Executing a GlideRecord inside of a loop! Creates a multitude of calls to the database!

```
1  var incidents = new GlideRecord('incidents');
2  incidents.query();
3
4  while (incidents.next()) {
5      var cmdb_ci = new GlideRecord('cmdb_ci');
6      cmdb_ci.addQuery('sys_id', incidents.cmdb_ci);
7      cmdb_ci.query();
8
9      while (cmdb_ci.next()) {
10         gs.info('---> {0}', cmdb_ci.name);
11     }
12 }
```

Best Practices – Loop Refactoring

- Best Practice

  Refactoring – consolidating the work down to two calls to the database

```
1  var incidents = new GlideRecord('incident');
2  incidents.query();
3
4  incidentsList = [];
5  while (incidents.next()) {
6      incidentsList.push(incidents.cmdb_ci + '');
7  }
8
9  var cmdb_ci = new GlideRecord('cmdb_ci');
10 cmdb_ci.addQuery('sys_id', 'IN', incidentsList);
11 cmdb_ci.query();
12
13 while (cmdb_ci.next()) {
14     gs.info('---> {0}', cmdb_ci.name);
15 }
```

Best Practices – Dot Notation

- Bad Practice - Complex OR GlideRecord Construction

```
1   var incidentRecords = new GlideRecord('incident');
2
3   var incQuery = incidentRecords.addQuery('location.name', 'LIKE', 'San Diego');
4   incQuery.addOrCondition('location.name', 'LIKE', 'Salt Lake City');
5   incQuery.addOrCondition('location.name', 'LIKE', 'New York');
6
7   var incQuery2 = incidentRecords.addQuery('priority', 1);
8   incQuery2.addOrCondition('priority', 3);
9
10  var incQuery3 = incidentRecords.addQuery('state', 2);
11  incQuery3.addOrCondition('state', 4);
12
13  incidentRecords.orderBy('number');
14  incidentRecords.query();
15
```

Best Practices – Dot Notation

- Dot Notation

```
1   var incidentRecords = new GlideRecord('incident');
2
3   incidentRecords.addQuery('location.name', 'LIKE', 'San Diego')
4       .addOrCondition('location.name', 'LIKE', 'Salt Lake')
5       .addOrCondition('location.name', 'LIKE', 'New York');
6   incidentRecords.addQuery('priority', 1).addOrCondition('priority', 3);
7   incidentRecords.addQuery('state', 2).addOrCondition('state', 4);
8
9   incidentRecords.orderBy('number');
10  incidentRecords.query();
```

Best Practices – Dot Notation

- Further Refactoring

```
1   var incidentRecords = new GlideRecord('incident');
2
3   incidentRecords.addQuery('location.name', 'CONTAINS', 'San Diego')
4       .addOrCondition('location.name', 'CONTAINS', 'Salt Lake')
5       .addOrCondition('location.name', 'CONTAINS', 'New York');
6   incidentRecords.addQuery('priority', 'IN', '1, 3');
7   incidentRecords.addQuery('state', 'IN', '2, 4');
8
9   incidentRecords.orderBy('number');
10  incidentRecords.query();
11
```

Best Practices – Unit Testing

- Bad Practice
  - Not testing an Update/Insert/Delete GlideRecord by doing a SELECT first!
  - The side-effect – Throws out query elements!!!

Best Practices – Unit Testing

- OOB – A bad GlideRecord .addQuery / .orCondition will be ignored!
- If you are doing an Update or a Delete this could lead to unintended consequences!

```
var incidents = new GlideRecord('incident');
incidents.addQuery('statsu', '7');
incidents.query();

while (incidents.next()) {
    gs.info('---> {0}', incidents.number);
    incidents.status = '3';
    incidents.update();
}
```

BAD!

WILL UPDATE ALL INCIDENT RECORDS!

Best Practices – Unit Testing

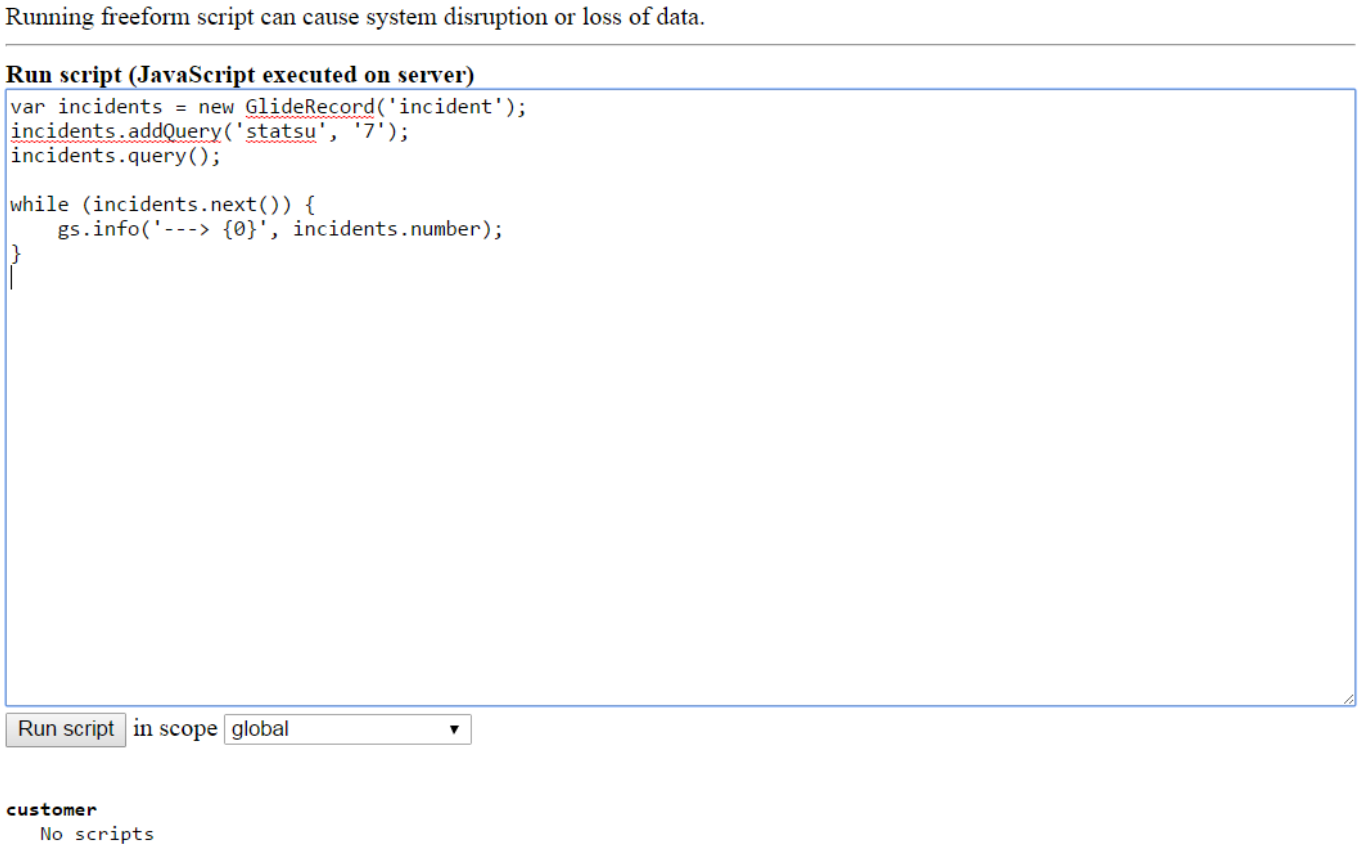
- Problems with GlideRecord errors – the default setting

Some Techniques:

- Turning on **glide.invalid_query.returns_no_rows**
- Try / Catch DOES NOT WORK with GlideRecord errors
- Extend the GlideRecord object to fail if an error is detected (complex)
- Try / Fail – Manual testing approach…ugh!
- Turn off Business Rule activation using .setWorkflow
- gs.trace – to measure execution time
- .getRowCount
- .setLimit

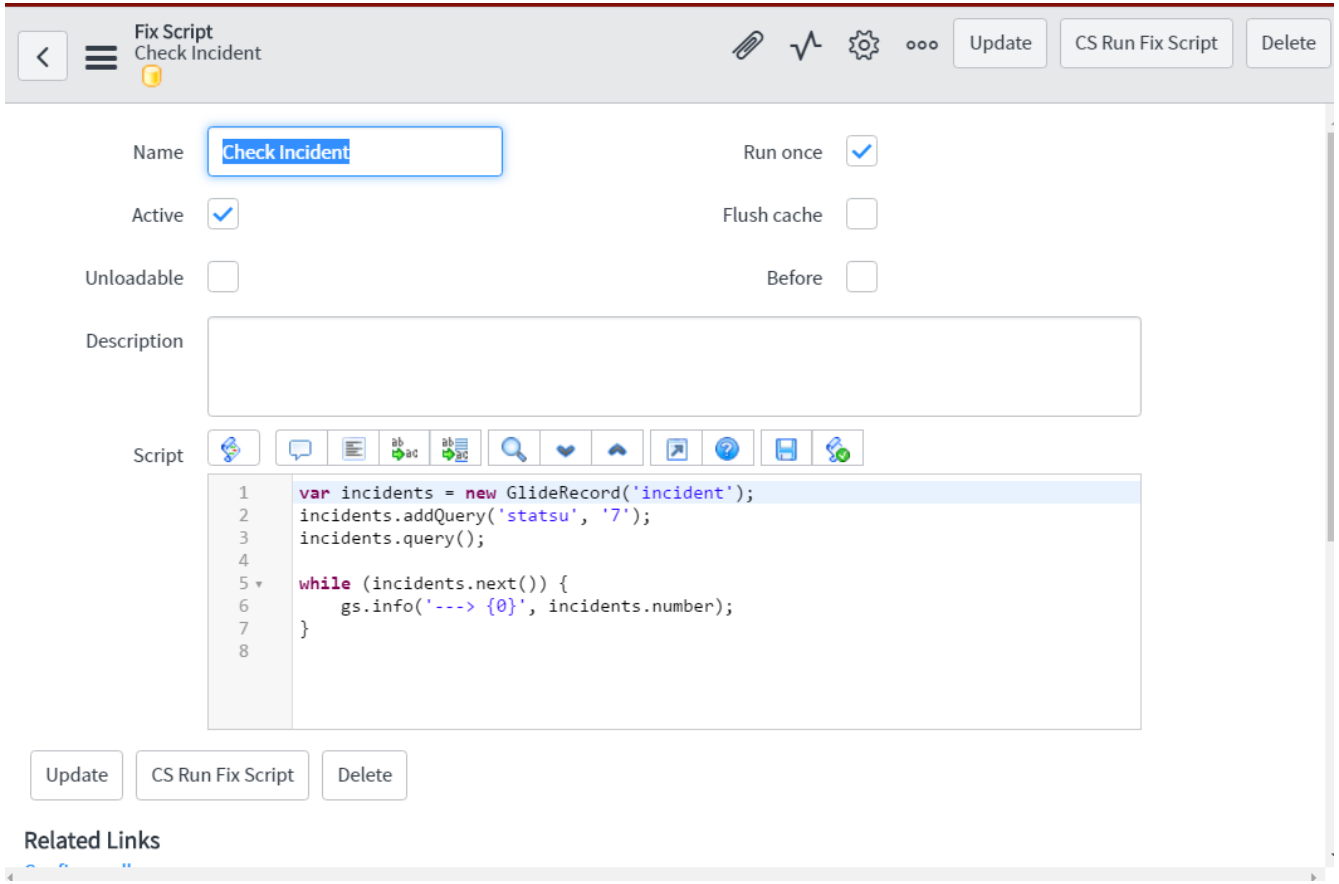
Best Practices – Unit Testing

- Testing techniques – Scripts – Background
  - Requires raising security level to Security Admin
  - No Auditing
  - No Versioning
  - Primitive Editor
  - Invisible to Update Sets



Best Practices – Unit Testing

- Testing techniques – Fix Scripts
  - Runs a normal Admin security
  - Versioning
  - Full Editor
  - Inclusion in Update Sets



Encoded Queries Intro

- What are they?
  - .addEncodedQuery
  - A type of short-hand to allow for complex queries to be described in a minimum of space
  - Can be obtained from the breadcrumbs of a list view, and from the list view of a condition builder column (UI Policies for example)

- Limitations
  - No user designated order of precedence! Parenthesis do NOT work!
  - No XOR function

Encoded Queries Intro

- Binary Operations
  - AND

FALSE	FALSE	**FALSE**
FALSE	TRUE	**FALSE**
TRUE	FALSE	**FALSE**
TRUE	TRUE	**TRUE**

  - OR

FALSE	FALSE	**FALSE**
FALSE	TRUE	**TRUE**
TRUE	FALSE	**TRUE**
TRUE	TRUE	**TRUE**

Encoded Queries Intro

- Binary Operations – Test!

  - AND

TRUE	FALSE	TRUE	**FALSE**
FALSE	FALSE	TRUE	**FALSE**

  - OR

TRUE	FALSE	TRUE	**TRUE**
FALSE	TRUE	FALSE	**TRUE**

Encoded Queries

- Before

```
var incidentRecords2 = new GlideRecord('incident');

incidentRecords2.addQuery('location.name', 'CONTAINS', 'San Diego')
    .addOrCondition('location.name', 'CONTAINS', 'Salt Lake')
    .addOrCondition('location.name', 'CONTAINS', 'New York');
incidentRecords2.addQuery('priority', 'IN', '1,3');
incidentRecords2.addQuery('state', 'IN', '2,4');

incidentRecords2.orderBy('number');
incidentRecords2.query();

while (incidentRecords2.next()) {
    gs.info('---> Number: {0}, {1}, {2}, {3}',
        incidentRecords2.number,
        incidentRecords2.priority,
        incidentRecords2.state,
        incidentRecords2.location.name);
}
```

- After

```
var sql = 'location.nameCONTAINSSan Diego' +
    '^ORlocation.nameCONTAINSSalt Lake' +
    '^ORlocation.nameCONTAINSNew York' +
    '^priorityIN1,3' +
    '^stateIN2,4';

var incidentRecords = new GlideRecord('incident');
incidentRecords.addEncodedQuery(sql);
incidentRecords.orderBy('number');
incidentRecords.query();

while (incidentRecords.next()) {
    gs.info('---> Number: {0}, {1}, {2}, {3}',
        incidentRecords.number,
        incidentRecords.priority,
        incidentRecords.state,
        incidentRecords.location.name);
}
```

Prototypes Introduction

- Prototype

```
function animal(legs, shell) {
    this.hasLegs = legs,
    this.hasShell = shell
}
animal.prototype.checkLegs = function() {
    return this.hasLegs;
};
var elephant = new animal(true, false);
if (elephant.checkLegs()) {
        alert("The elephant does have legs!");
}
```

Our original function: Animal with two properties and no methods.

Now we add a new method to our object (this is immediately available).

When we instantiate the object we get the new method!

Prototypes Introduction

- Prototype

```
animal.prototype.checkShell = function() {
    return this.hasShell;
}

if (!elephant.checkShell()) {
        alert("The elephant does not have a shell!");
}
```

Add yet another method. Is this immediately available even though it was added later?

Yes! But please don't do this kind of coding! ☺

Extending GlideRecord

- So what exactly is the GlideRecord object to JavaScript?
  - It is a sealed object in that the OOB GlideRecord cannot be directly modified, or even observed
  - You cannot directly add new fields or methods to it
  - It CAN be extended using a JavaScript Prototype!

```
1  GlideRecord.prototype.addBetweenQuery = function(fieldToCheck, beginDate, endDate) {
2      var beginDateCheck = gs.dateGenerate(beginDate); // convert to the appropriate string
3      var endDateCheck = gs.dateGenerate(endDate); // ibid.
4
5      this.addQuery(fieldToCheck, '>=', beginDateCheck);
6      this.addQuery(fieldToCheck, '<=', endDateCheck);
7  };
```

Useful Extras

- .orderBy
- .groupBy
- .autoAssistFields
- .setForceUpdate
- .glideRecordSecure
- gs.getSession().setStrictQuery
- global.JSUtil – listObject, nil, notNil
- global.GlideUtil - unique

Conclusion (Workshop)

- What will you be creating in the workshop?
  - Taking an inefficient GlideRecord in a loop example and refactoring it to make it faster – includes the techniques used to measure the difference in speed
  - Taking a difficult to maintain complex GlideRecord query and using dot notation making it easier to use
  - Taking the dot notation GlideRecord query and turning it into an encoded query
  - Extending the GlideRecord Object to include a new “Between Dates” functionality

Conclusion

- Update Set
  - Create at the beginning of your lab
  - At the end of the lab complete and export

Conclusion

- Resources and Useful Links

  Developer Community!

  Geneva Wiki!

  Community Code Snippets: Articles List to Date

  www.codeschool.com <- Javascript 3rd course, and Javascript Best Practices course.

  Wiki: Application Scoping

  Community: Application Scoping

  Books:
  Javascript the Good Parts by Douglas Crockford
  Secrets of the JavaScript Ninja by John Resig
  The Principles of Object-Oriented JavaScript by Nicholas Zakas
  The entire “You Don’t Know JS” series by Kyle Simpson

Q & A

Demo

Workshop

- Get you lab here: http://klabs.link/ags-b
  - Password: **Knowledge16**

- Lab Guide:
  - **https://github.com/sabell2016/creatorcon**

- All other content from CreatorCon 16:
  - https://community.servicenow.com/community/on-demand-library

- CreatorCon 15 Content:
  - https://community.servicenow.com/docs/DOC-3051
  - https://community.servicenow.com/community/on-demand-library/knowledge15-exclusive-content
- https://community.servicenow.com

Take the Survey

Please take a moment to complete a session survey in the Knowledge16 app.

Thank You

Steven Bell

Sr. Implementation Specialist

Accenture Technology

steven.bell@accenture.com

Mark Amann

Sr. Implementation Specialist

Accenture Technology

mark.amann@accenture.com