knowledge14®

April 27–May 1, 2014 • Moscone Center • San Francisco, CA

Debugging 201: Practical Troubleshooting Concepts and Tools

Chris Maloy
Expert Services Team Lead
ServiceNow

Josh Nerius
Expert Services Consultant
ServiceNow

servicenow®

© 2014 ServiceNow All Rights Reserved

Agenda

General Debugging Concepts

Debugging Business Rules, Debug Log, Error Handling

Debugging Client Scripts, Client Script Performance

Field Watcher – **New!**

JavaScript Debugger – **New!**

© 2014 ServiceNow All Rights Reserved

Overview

- Disclaimer
  - Advanced topics and quick pace
  - Assumes some scripting experience and knowledge of ServiceNow concepts

- Objectives
  - Deal with the 3 areas where most issues occur
    - Something doesn't work
    - Something is slow
    - Don't know why something is occurring

© 2014 ServiceNow All Rights Reserved

Overview

- Setup
  - Lab instance
  - Reporting Lab Progress
- Optional (Install your Browser's JavaScript debugger)
  - Firefox and Firebug
  - Chrome (built-in)
  - Safari (built-in)
    - If the Develop menu does not appear in the menu bar, open Safari preferences, click Advanced, and select "Show Develop menu in menu bar."
  - IE 9 & 10 (built-in)

© 2014 ServiceNow All Rights Reserved

Quote

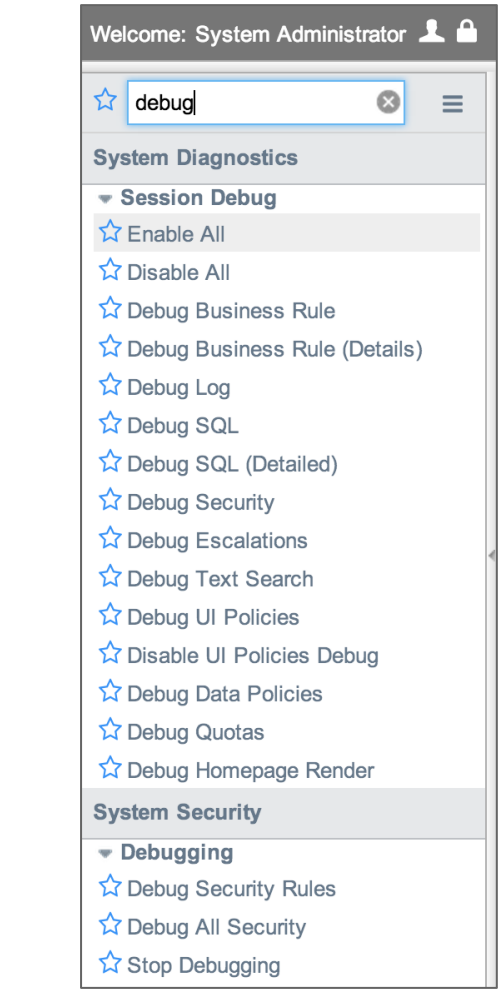
"If debugging is the process of removing software bugs, then programming must be the process of putting them in."

Edsger W. Dijkstra

© 2014 ServiceNow All Rights Reserved

Debugging Concepts

- Application
  - Session Debug modules
  - JavaScript Debugger (**New**)
- Application + Browser
  - Field Watcher (**New**)
- Browser
  - Developer Tools
  - Add-ons such as Firebug
  - JavaScript Log



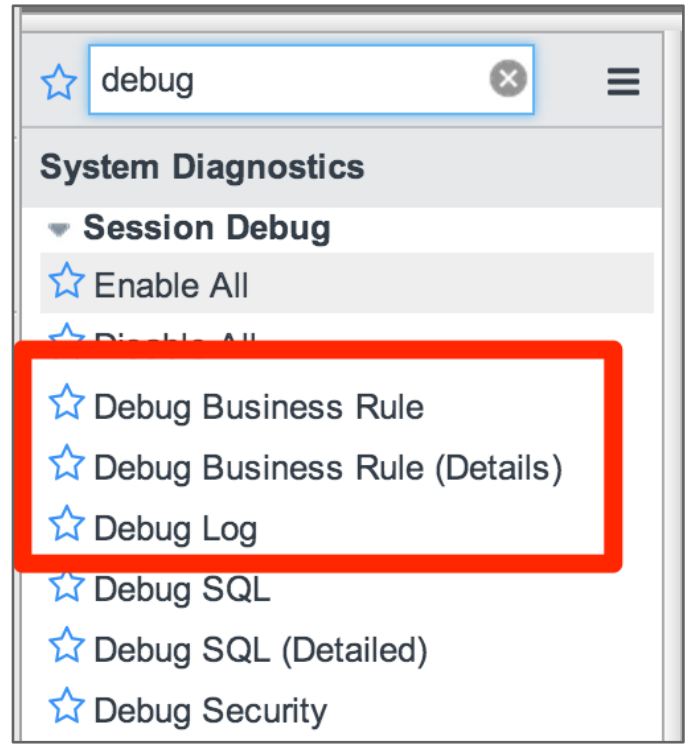
© 2014 ServiceNow All Rights Reserved

Debugging Business Rules

© 2014 ServiceNow All Rights Reserved

Debugging Business Rules

- Several options for debugging business rules
  - Debug Business Rule
  - Debug Business Rule (Details)
  - Debug Log is also useful during Business Rule debug session
- Enable at the beginning of development sessions



© 2014 ServiceNow All Rights Reserved

Debugging Business Rules: Output

```
08:03:21.227: Execute before business rules on sys_user:
08:03:21.227: === Skipping 'user query' on sys_user:; condition not satisfied: Condition: gs.getSession().isInteractive() && !gs.hasRole("admin")
08:03:21.248: Execute before business rules on sys_user_group:
08:03:21.249: ==> 'group query' on sys_user_group:
08:03:21.249: <== 'group query' on sys_user_group:
08:03:21.33[illegible]: Execute before business rules on incident:INC0010001 before engines (order <1000)
08:03:21.33[illegible]: ==> 'Copy State to Incident State' on incident:INC0010001
08:03:21.336: <== 'Copy State to Incident State' on incident:INC0010001
08:03:21.336: ==> 'Task Active State Management' on incident:INC0010001
08:03:21.364: <== 'Task Active State Management' on incident:INC0010001
08:03:21.366: === Skipping 'Change state on closed Insert' on incident:INC0010001; condition not satisfied: Filter Condition: incident_state=7^EQ
08:03:21.367: === Skipping 'Company Projection' on incident:INC0010001; condition not satisfied: Filter Condition: caller_idISNOTEMPTY^companyISEMPTY^EQ
08:03:21.368: === Skipping 'insert_incident' on incident:INC0010001; condition not satisfied: Condition: current.number.nil()
08:03:21.369: === Skipping 'mark_closed' on incident:INC0010001; condition not satisfied: Condition: current.incident_state == 7
08:03:21.378: === Skipping 'mark_resolved' on incident:INC0010001; condition not satisfied: Condition: current.incident_state == 6 || current.incident_state == 7
08:03:21.378: ==> 'Prevent Recursion of Parent Incident' on incident:INC0010001
08:03:21.383: <== 'Prevent Recursion of Parent Incident' on incident:INC0010001
08:03:21.384: ==> 'Stamp Approvals' on incident:INC0010001
08:03:21.395: <== 'Stamp Approvals' on incident:INC0010001
08:03:21.396: === Skipping '[DT] Exercise 2 - Incident Impact Change' on incident:INC0010001; condition not satisfied: Condition: current.category == "hardware"
08:03:21.397: === Skipping 'Caller Close' on incident:INC0010001; condition not satisfied: Condition: current.caller_id == gs.getUserID() && (current.incident_state.changesTo(6) ||
current.incident_state.changesTo(7))
08:03:21.398: === Skipping 'mark closed' on incident:INC0010001; condition not satisfied: Condition: current.state.changesTo(3) || current.state.changesTo(4) || current.state.changesTo(7)
08:03:21.399: === Skipping 'task closer' on incident:INC0010001; condition not satisfied: Filter Condition: activeCHANGESFROMtrue^active=false^state!=3^state!=4^state!=7^EQ
08:03:21.401: === Skipping 'task reopener' on incident:INC0010001; condition not satisfied: Filter Condition:
activeCHANGESFROMfalse^active=true^state=3^ORstate=4^NQactive=false^state=1^EQ
```

© 2014 ServiceNow All Rights Reserved

Debugging Business Rules: Output

08:03:21.227: Execute before business rules on sys_user:
08:03:21.227: === Skipping 'user query' on sys_user:; condition not satisfied: Condition: gs.getSession().isInteractive() && !gs.hasRole("admin")
08:03:21.248: Execute before business rules on sys_user_group:
08:03:21.249: ==> 'group query' on sys_user_group:
08:03:21.249: <== 'group query' on sys_user_group:
08:03:21.334: Execute before business rules on incident:INC0010001 before engines (order <1000)
08:03:21.336: ==> 'Copy State to Incident State' on incident:INC0010001
08:03:21.336: <== 'Copy State to Incident State' on incident:INC0010001
08:03:21.336: ==> 'Task Active State Management' on incident:INC0010001
08:03:21.364: <== 'Task Active State Management' on incident:INC0010001
08:03:21.366: === Skipping 'Change state on closed Insert' on incident:INC0010001; condition not satisfied: Filter Condition: incident_state=7^EQ
08:03:21.367: === Skipping 'Company Projection' on incident:INC0010001; condition not satisfied: Filter Condition: caller_idISNOTEMPTY^companyISEMPTY^EQ
08:03:21.368: === Skipping 'insert_incident' on incident:INC0010001; condition not satisfied: Condition: current.number.nil()
08:03:21.369: === Skipping 'mark_closed' on incident:INC0010001; condition not satisfied: Condition: current.incident_state == 7
08:03:21.378: === Skipping 'mark_resolved' on incident:INC0010001; condition not satisfied: Condition: current.incident_state == 6 || current.incident_state == 7
08:03:21.378: ==> 'Prevent Recursion of Parent Incident' on incident:INC0010001
08:03:21.383: <== 'Prevent Recursion of Parent Incident' on incident:INC0010001
08:03:21.384: ==> 'Stamp Approvals' on incident:INC0010001
08:03:21.395: <== 'Stamp Approvals' on incident:INC0010001
08:03:21.396: === Skipping '[DT] Exercise 2 - Incident Impact Change' on incident:INC0010001; condition not satisfied: Condition: current.category == "hardware"
08:03:21.397: === Skipping 'Caller Close' on incident:INC0010001; condition not satisfied: Condition: current.caller_id == gs.getUserID() && (current.incident_state.changesTo(6) || current.incident_state.changesTo(7))
08:03:21.398: === Skipping 'mark closed' on incident:INC0010001; condition not satisfied: Condition: current.state.changesTo(3) || current.state.changesTo(4) || current.state.changesTo(7)
08:03:21.399: === Skipping 'task closer' on incident:INC0010001; condition not satisfied: Filter Condition: activeCHANGESFROMtrue^active=false^state!=3^state!=4^state!=7^EQ
08:03:21.401: === Skipping 'task reopener' on incident:INC0010001; condition not satisfied: Filter Condition: activeCHANGESFROMfalse^active=true^state=3^ORstate=4^NQactive=false^state=1^EQ

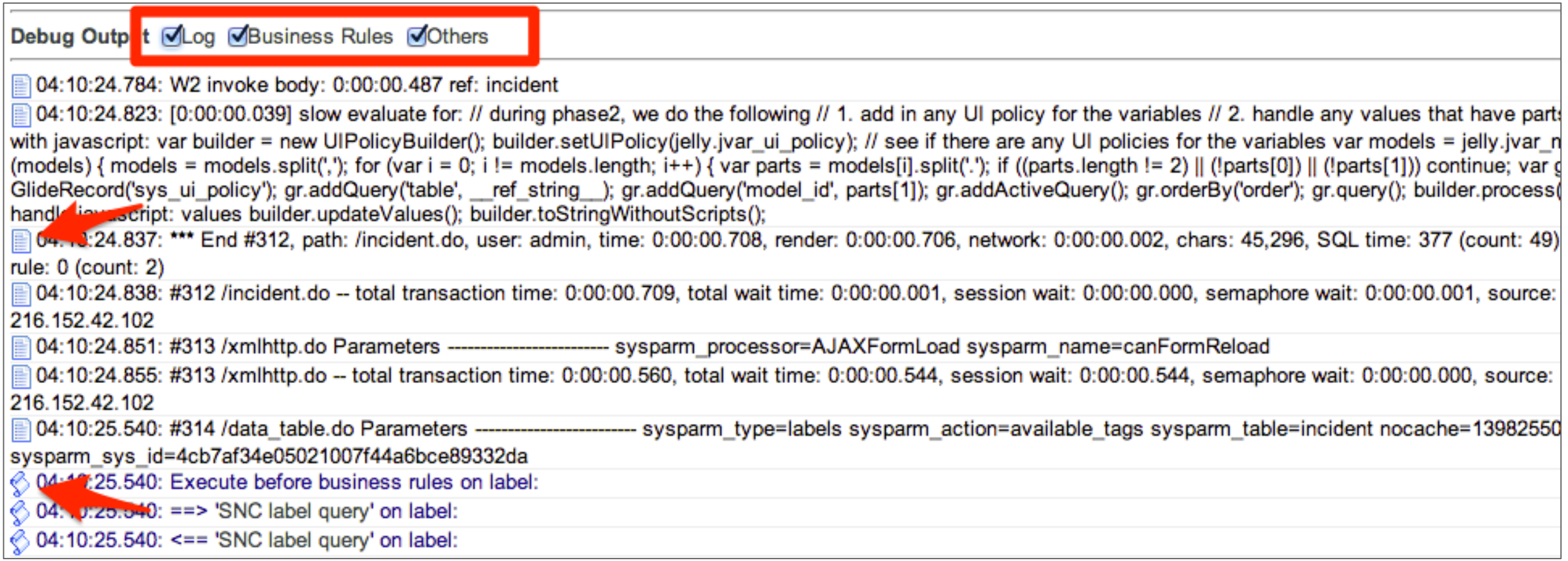
© 2014 ServiceNow All Rights Reserved

Debugging Business Rules: Output

```
08:03:21.227: Execute before business rules on sys_user:
08:03:21.227: === Skipping 'user query' on sys_user:; condition not satisfied: Condition: gs.getSession().isInteractive() && !gs.hasRole("admin")
08:03:21.248: Execute before business rules on sys_user_group:
08:03:21.249: ==> 'group query' on sys_user_group:
08:03:21.249: <== 'group query' on sys_user_group:
08:03:21.334: Execute before business rules on incident:INC0010001 before engines (order <1000)
08:03:21.336: ==> 'Copy State to Incident State' on incident:INC0010001
08:03:21.336: <== 'Copy State to Incident State' on incident:INC0010001
08:03:21.336: ==> 'Task Active State Management' on incident:INC0010001
08:03:21.364: <== 'Task Active State Management' on incident:INC0010001
08:03:21.366: === Skipping 'Change state on closed Insert' on incident:INC0010001; condition not satisfied: Filter Condition: incident_state=7^EQ
08:03:21.367: === Skipping 'Company Projection' on incident:INC0010001; condition not satisfied: Filter Condition: caller_idISNOTEMPTY^companyISEMPTY^EQ
08:03:21.368: === Skipping 'insert_incident' on incident:INC0010001; condition not satisfied: Condition: current.number.nil()
08:03:21.369: === Skipping 'mark_closed' on incident:INC0010001; condition not satisfied: Condition: current.incident_state == 7
08:03:21.378: === Skipping 'mark_resolved' on incident:INC0010001; condition not satisfied: Condition: current.incident_state == 6 || current.incident_state == 7
08:03:21.378: ==> 'Prevent Recursion of Parent Incident' on incident:INC0010001
08:03:21.383: <== 'Prevent Recursion of Parent Incident' on incident:INC0010001
08:03:21.384: ==> 'Stamp Approvals' on incident:INC0010001
08:03:21.395: <== 'Stamp Approvals' on incident:INC0010001
08:03:21.396: === Skipping '[DT] Exercise 2 - Incident Impact Change' on incident:INC0010001; condition not satisfied: Condition: current.category == "hardware"
08:03:21.397: === Skipping 'Caller Close' on incident:INC0010001; condition not satisfied: Condition: current.caller_id == gs.getUserID() && (current.incident_state.changesTo(6) ||
current.incident_state.changesTo(7))
08:03:21.398: === Skipping 'mark closed' on incident:INC0010001; condition not satisfied: Condition: current.state.changesTo(3) || current.state.changesTo(4) || current.state.changesTo(7)
08:03:21.399: === Skipping 'task closer' on incident:INC0010001; condition not satisfied: Filter Condition: activeCHANGESFROMtrue^active=false^state!=3^state!=4^state!=7^EQ
08:03:21.401: === Skipping 'task reopener' on incident:INC0010001; condition not satisfied: Filter Condition:
activeCHANGESFROMfalse^active=true^state=3^ORstate=4^NQactive=false^state=1^EQ
```

© 2014 ServiceNow All Rights Reserved

Debug Output: Visibility Options and Log Output



© 2014 ServiceNow All Rights Reserved

Lab 1

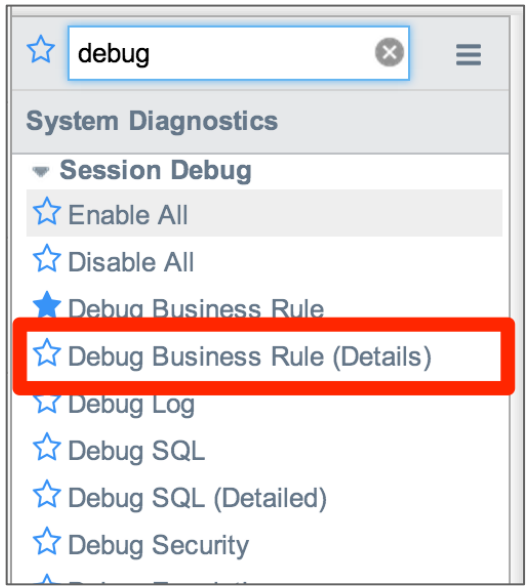
Debug Business Rule

- **Scenario:** we have a business rule that should add comments when an incident is closed.
- **Problem:** comments are not added when the incident is closed.
- **Lab Goal**: utilize the **Debug Business Rule** module to determine what is wrong.

© 2014 ServiceNow All Rights Reserved

Debug Business Rule (Details)

- Gives us additional debugging output
- Shows details of field value changes





© 2014 ServiceNow All Rights Reserved
© 2013 ServiceNow All Rights Reserved

Lab 2

Debug Business Rule (Details)

- **Scenario:** whenever a hardware incident is created, the **Impact** is set to 1.
- **Problem:** we do not know why.
- **Lab Goal**: utilize **Debug Business Rule (Details)** to determine what is modifying the **Impact** value.

© 2014 ServiceNow All Rights Reserved

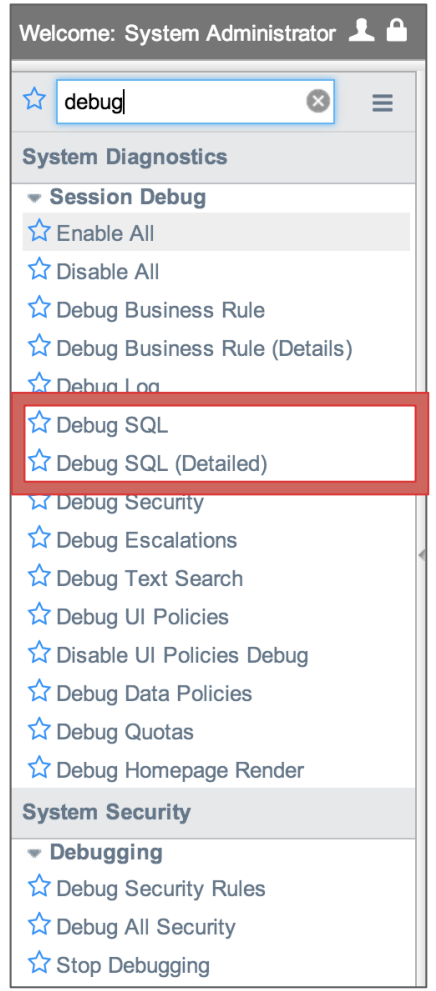
Other Debugging Output

© 2014 ServiceNow All Rights Reserved

Other Useful Server Side Debug Modules

- **Debug SQL** and **Debug SQL (Detailed)**

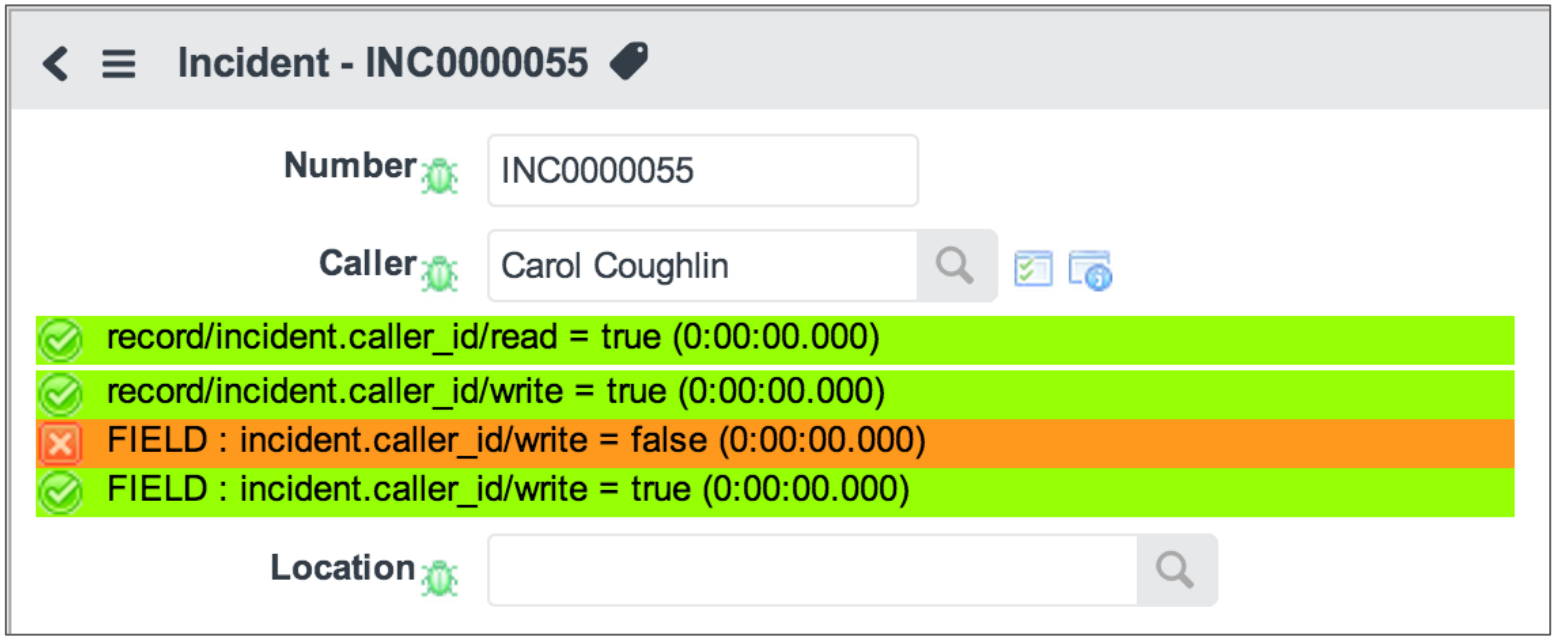




© 2014 ServiceNow All Rights Reserved

Other Useful Server Side Debug Modules

- Debug Security





© 2014 ServiceNow All Rights Reserved

Other Useful Server Side Debug Modules

- Debug Security

05:49:18.384: TIME = 0:00:00.000 PATH = ui_page/incident/read CONTEXT = sys_user.sys_id=null RC = true RULE =

05:49:18.391: TIME = 0:00:00.001 PATH = record/incident/write CONTEXT = INC0000055 RC = true RULE =

record/incident/write

record/incident/write

05:49:18.401: TIME = 0:00:00.000 PATH = record/incident/create CONTEXT = INC0000055 RC = true RULE =

record/incident/create

05:49:18.401: TIME = 0:00:00.000 PATH = record/incident/create CONTEXT = INC0000055 RC = true RULE = Evaluated from cache

record/incident/create

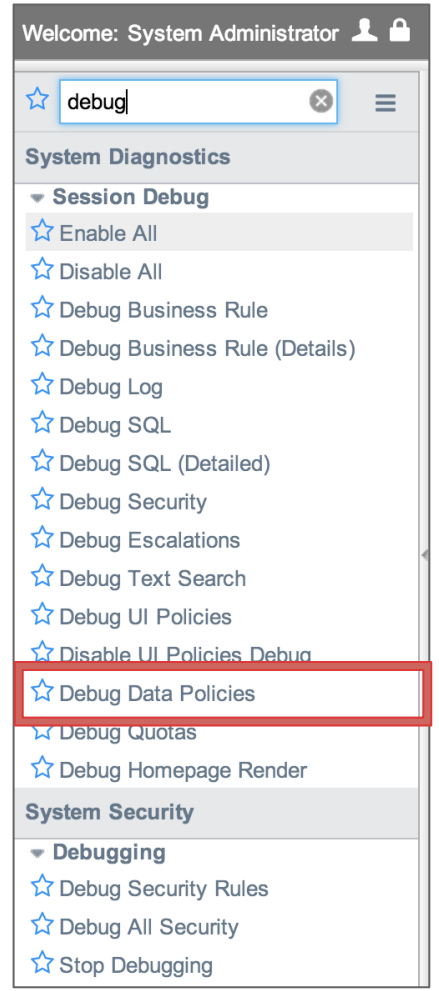


© 2014 ServiceNow All Rights Reserved

Other Useful Server Side Debug Modules

- Debug Data Policies

```
05:58:56.662: DataPolicies Found : 1
05:58:56.662: Running Data Policy on incident "Short Description is Mandatory"
05:58:56.662: Evaluating Condition
05:58:56.664: category=request^EQ - false
```



© 2014 ServiceNow All Rights Reserved

Error Handling

© 2014 ServiceNow All Rights Reserved

Error Handling

- Error handling is critical to ensuring predictable behavior
- Uncaught exceptions can have significant downstream impact
- Use try/catch to handle exceptions
  - Gives **you** control over code behavior when errors are encountered
  - Gives you the opportunity to “clean up” after the exception
  - Use when calling something you don’t have control over such as a Web Service
  - Allows code to continue executing after the exception is encountered

© 2014 ServiceNow All Rights Reserved

Error Handling: try / catch

Script:

```
81
82          // Parse FQDN with regex. By default, we take the last two separated by dot as
83          // the domain name and the rest as host name.
84          try {
85              var regexStr = gs.getProperty('glide.discovery.fqdn.regex', "^(.+)\\.(.+?\\..+?)$");
86              var fqdnRegex = new RegExp(regexStr);
87              var groups = fqdnRegex.exec(sysName);
88              if (groups.length != 3)
89                  return;
90
91              this.sysName = groups[1];
92              this.dnsDomain = groups[2];
93          } catch (e) {
94              this.log("FQDN parsing error! The regex " + regexStr + " was not able to parse '" + sysName + "'. Th
95          }
96      },
```

© 2014 ServiceNow All Rights Reserved

Lab 3

Try / Catch

- **Scenario:** a simple business rule should add three messages when saving a software incident.
- **Problem:** we are only seeing one of the three messages.
- **Lab Goal**: use try/catch to handle the exception thrown and allow code to continue executing.

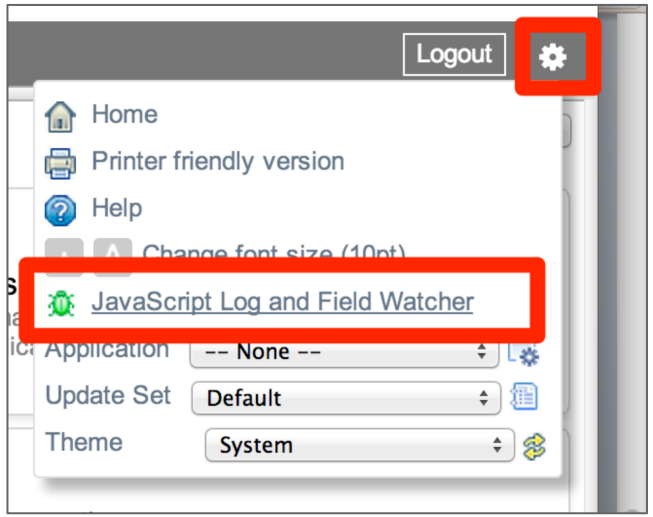
© 2014 ServiceNow All Rights Reserved

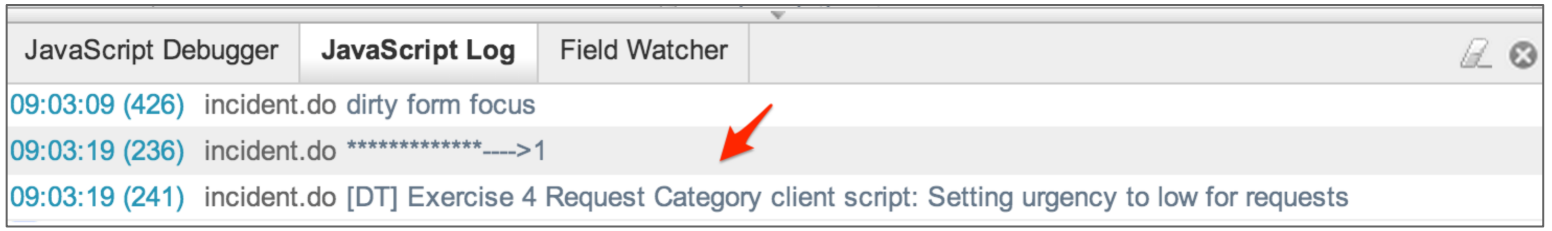
Client Script Debugging

© 2014 ServiceNow All Rights Reserved

Client Script Debugging

- Use **JavaScript Log and Field Watcher**
- Use native browser tools
- **Debugging 301** discusses native browser tools in more detail





© 2014 ServiceNow All Rights Reserved

Lab 4

Client Script Debugging

- **Lab Goal**: use the JavaScript Log and Field Watcher to observe client script output

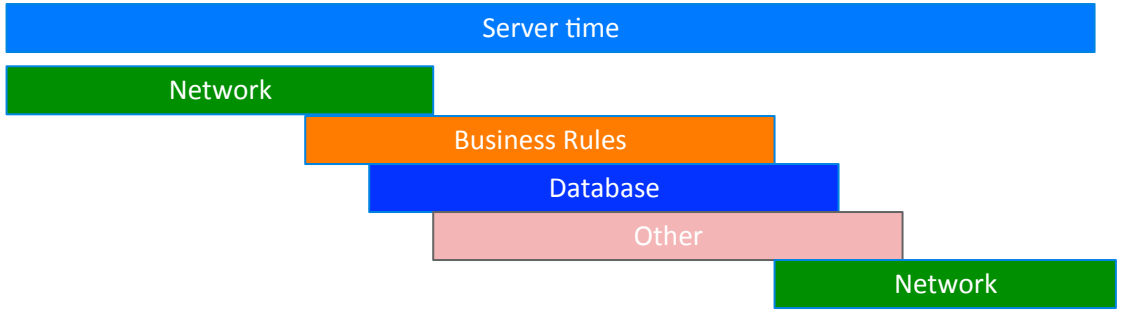
© 2014 ServiceNow All Rights Reserved

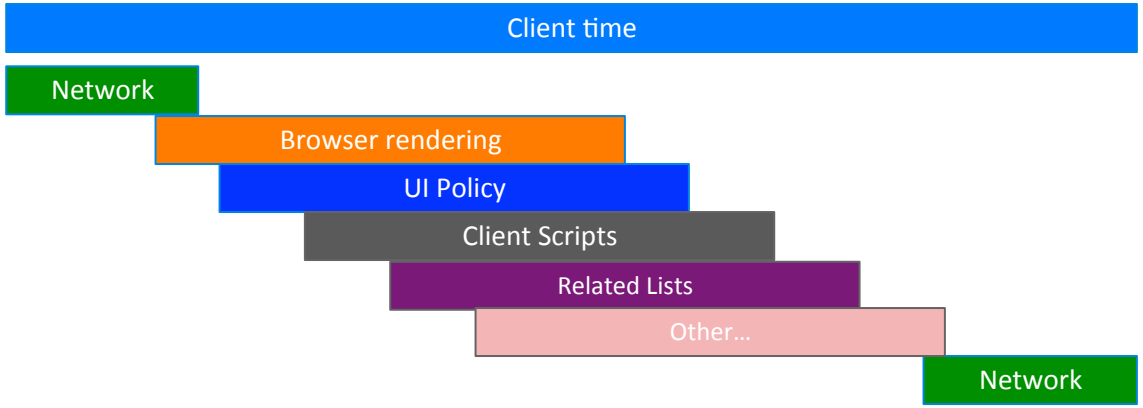
Performance

© 2014 ServiceNow All Rights Reserved

Client Performance

- If you cannot find the bottlenecks, you cannot fix the them
- Covered in more depth in **Debugging 301**
- The performance supply chain:

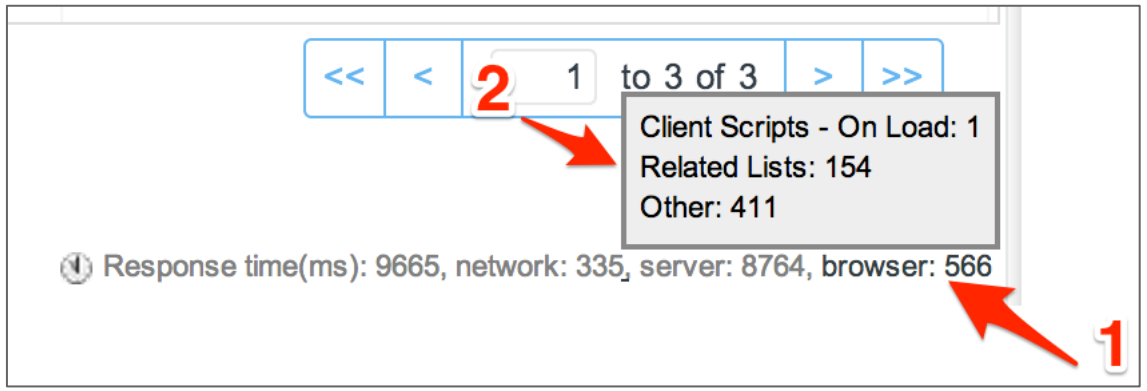


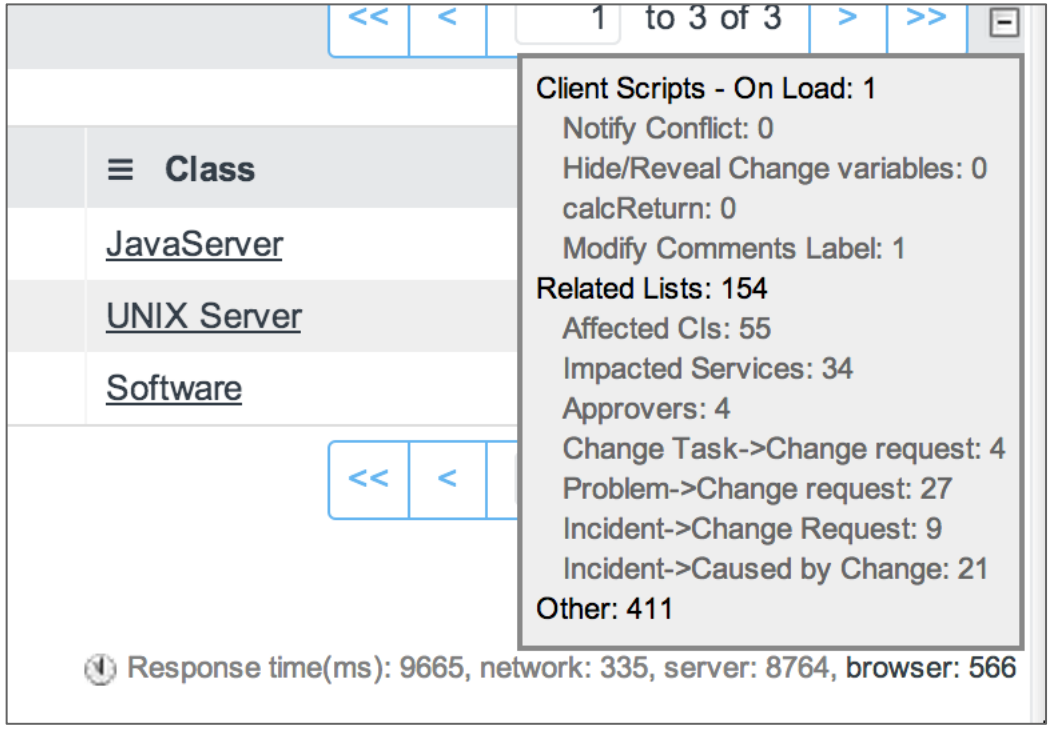


© 2014 ServiceNow All Rights Reserved
© 2013 ServiceNow All Rights Reserved

Client Performance

- Browser Timing Widget





© 2014 ServiceNow All Rights Reserved
© 2013 ServiceNow All Rights Reserved

Lab 5

Client Script Performance

- **Scenario:** we want to highlight VIP callers on Incident records.
- **Problem:** we notice that the browser hangs briefly when a VIP caller is selected.
- **Lab Goal**: use AJAX callbacks to optimize our client script.

© 2014 ServiceNow All Rights Reserved

Display Business Rules

- When retrieving data for use by the client, we have several options:
  - Add the field to the form and hide it using UI Policy
  - Use a client GlideRecord query (AJAX)
  - Use a display business rule to preload data and pass it to the client using g_scratchpad
- Avoid using costly client-side GlideRecord queries
- Use Display Business Rules in combination with g_scratchpad
- g_scratchpad
  - Provides a simple way for server-side code to send data to the client
  - Populated server-side, sent to client as an object

© 2014 ServiceNow All Rights Reserved

Lab 6

Display Business Rules

- **Scenario:** a client script needs to access data from a field that is not on the form.
- **Lab Goal**: examine existing display business rule and observe how it works.
- **Lab Challenge:** create your own display business rule that utilizes g_scratchpad.

© 2014 ServiceNow All Rights Reserved

Server Performance – GlideRecord setLimit()

- Always construct GlideRecord queries to return as little data as possible
- Use query conditions to limit result set
- If a query must return multiple records, consider using setLimit()
  - Ideal for situations where you must determine that “1 or more records exist”
  - **Example**: determine if a user has any open incident records. If the user has 100 open incident records, setLimit can reduce the amount of data returned without impacting the goal.

© 2014 ServiceNow All Rights Reserved

Lab 7

GlideRecord setLimit()

- **Lab Goal**: use setLimit() to reduce the number of records returned by a query.

© 2014 ServiceNow All Rights Reserved

Additional thoughts about performance

- Smaller forms
  - Less data over network and less to render
  - Few fields / fewer related lists / fewer client scripts
  - Consider using views
- Reduce round trips
  - Use Display Business Rules
  - Bundle multiple AJAX calls into one
- Use client scripts judiciously
- Avoid Blocking AJAX calls, use callbacks
- Be aware of the performance impact older browsers can have

© 2014 ServiceNow All Rights Reserved
© 2013 ServiceNow All Rights Reserved

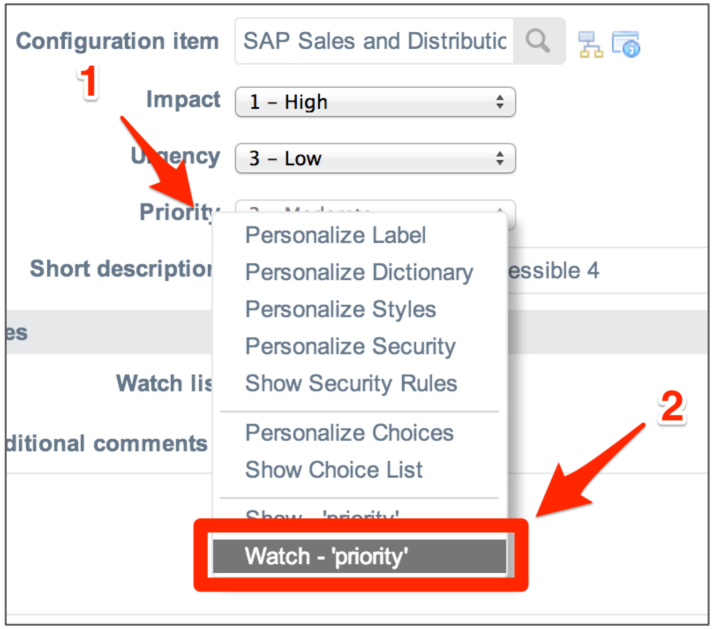
New!

Field Watcher and JavaScript Debugger

© 2014 ServiceNow All Rights Reserved

Field Watcher

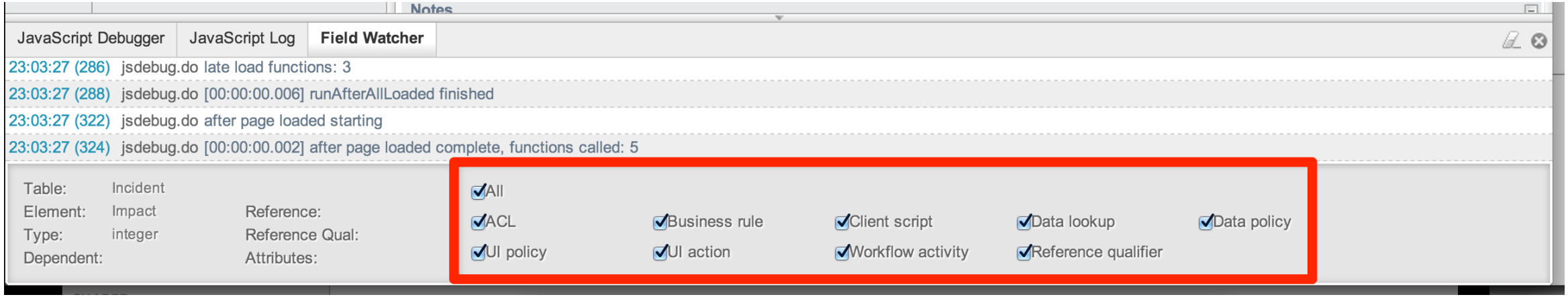
- **New** with the Dublin release
- Tracks and displays all actions on a selected form field
- Can be used to determine what happens to a field when events take place such as
  - Business rules firing
  - Enforcement of data policy
  - UI Action
  - Many more
- Only one field can be watched at a time
- Not available on password protected or encrypted fields



© 2014 ServiceNow All Rights Reserved

Field Watcher

- Select which types of events you want to watch



© 2014 ServiceNow All Rights Reserved

Field Watcher

- Output after saving a record



- **Orange** text = server-side changes such as ACLs
- **Blue** text = client-side changes such as client scripts

© 2014 ServiceNow All Rights Reserved

Lab 8

Field Watcher

- **Lab Goal**: use the Field Watcher to determine why a field value is changing.

© 2014 ServiceNow All Rights Reserved

Field Watcher – Additional Resources

- **ServiceNow Demo**: How to Debug with Field Watcher
  - https://www.youtube.com/watch?v=NEaahrwgqXM

© 2014 ServiceNow All Rights Reserved

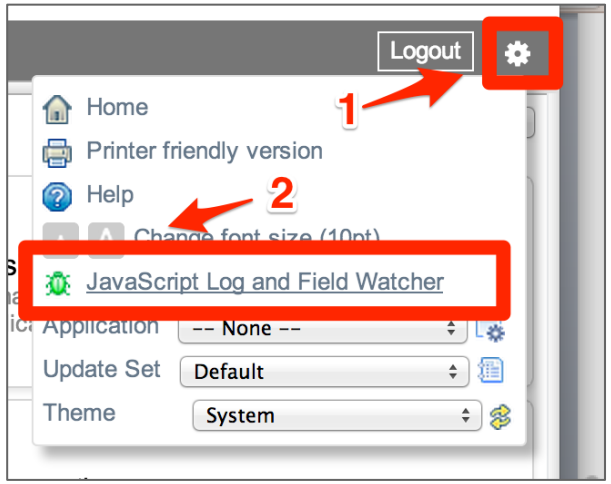
JavaScript Debugger

- **New** with the Dublin release
- **Server-side** JavaScript debugging
- Efficiently debug scripts and avoid gs.log statements
- Currently supports **Business Rules** and **Script Includes**
- Line-by-line control of code execution
- Allows inspection of server-side variables

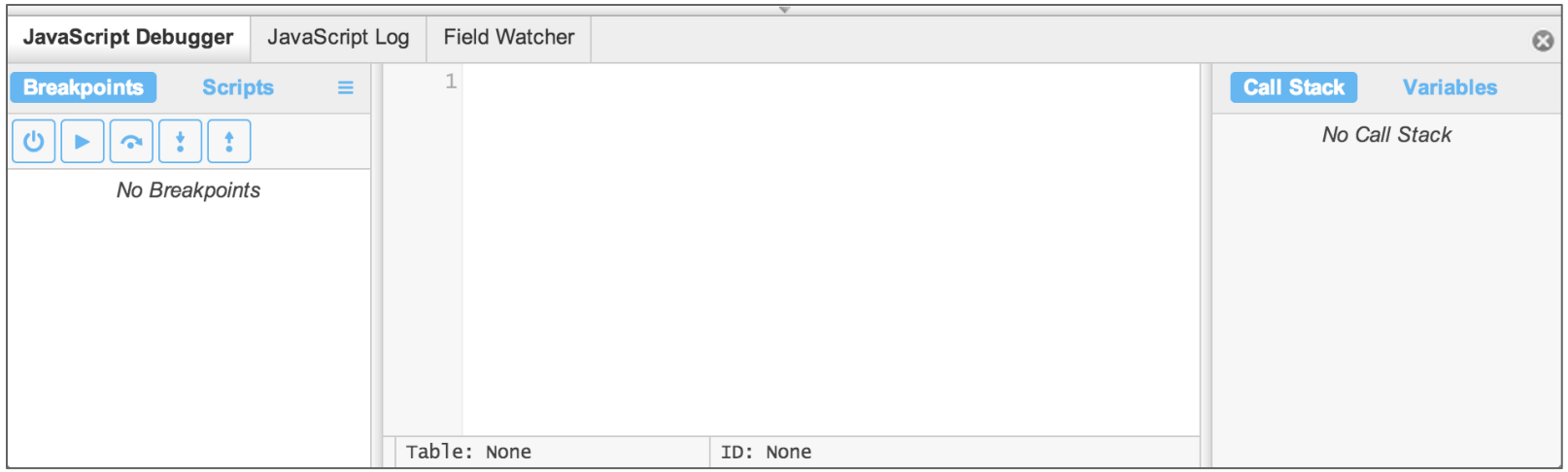
© 2014 ServiceNow All Rights Reserved

JavaScript Debugger

- Activating the Debugger
  - 1. Click the cog near the logout button
  - 2. Click “JavaScript Log and Field Watcher



- Debugger Pane



© 2014 ServiceNow All Rights Reserved

JavaScript Debugger – Debug Panel

Enable / Disable Debugging



© 2014 ServiceNow All Rights Reserved

JavaScript Debugger – Debug Panel

Continue



© 2014 ServiceNow All Rights Reserved

JavaScript Debugger – Debug Panel

Step over



© 2014 ServiceNow All Rights Reserved

JavaScript Debugger – Debug Panel



© 2014 ServiceNow All Rights Reserved

JavaScript Debugger – Debug Panel



© 2014 ServiceNow All Rights Reserved

JavaScript Debugger – Debug Panel

Shows current breakpoints



© 2014 ServiceNow All Rights Reserved

JavaScript Debugger – Debug Panel

Navigate / Search for Scripts



© 2014 ServiceNow All Rights Reserved

JavaScript Debugger – Debug Panel

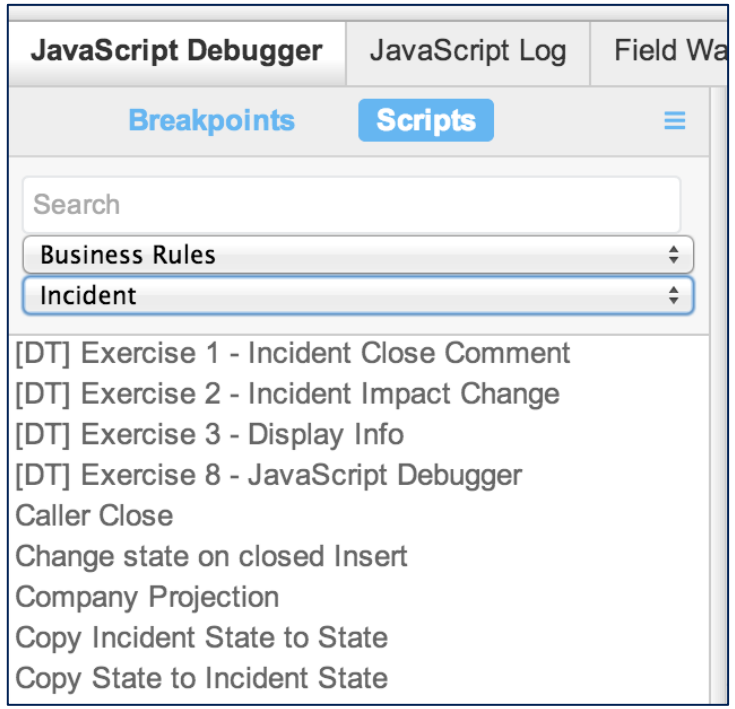
Variable Explorer



© 2014 ServiceNow All Rights Reserved

JavaScript Debugger – Setting Breakpoints

- Find the Script Include or Business rule to debug
  - Search by name
  - Filter by type and table



© 2014 ServiceNow All Rights Reserved

JavaScript Debugger – Setting Breakpoints

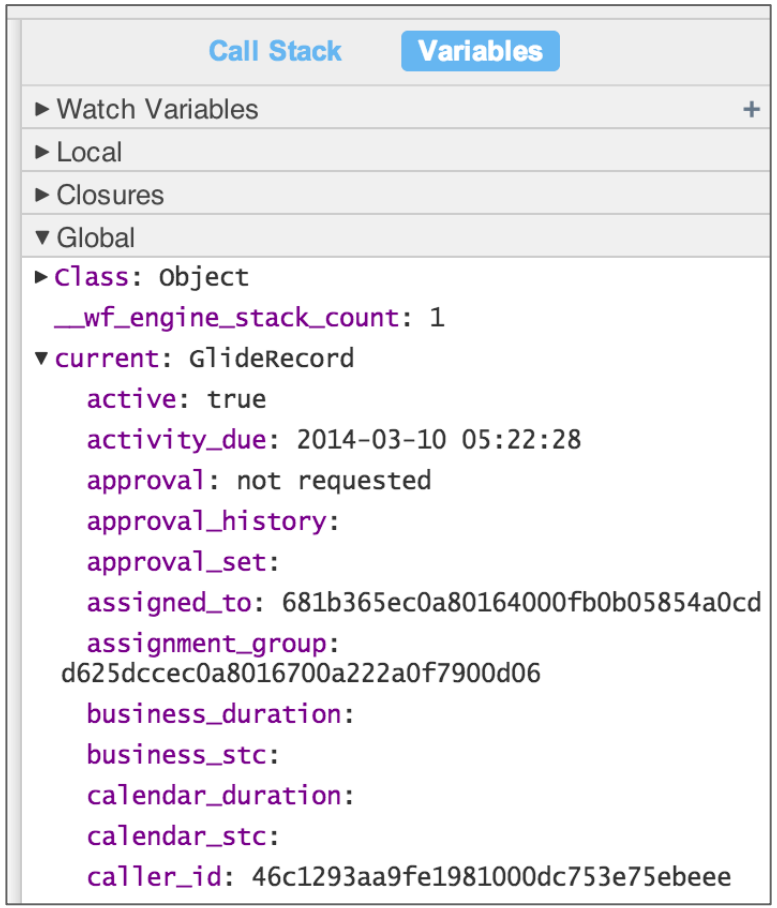
- Breakpoints are set from the code window
  - Click to the left of the line number you wish to break on
- Code changes can be saved directly from the debugger window



© 2014 ServiceNow All Rights Reserved

JavaScript Debugger – Variable Inspector

- View variables and values in real time
  - Watch variables
  - Local variables
  - Global variables
  - Closures



© 2014 ServiceNow All Rights Reserved

JavaScript Debugger – Additional Considerations

- Don’t use the debugger in Production
- Requires Internet Explorer 9 or higher, Firefox, Chrome or Safari
- Plugin may need to be manually activated on upgraded instances
  - Server-side JavaScript Debugger (**com.glide.debugger**)



© 2014 ServiceNow All Rights Reserved

Lab 9

JavaScript Debugger

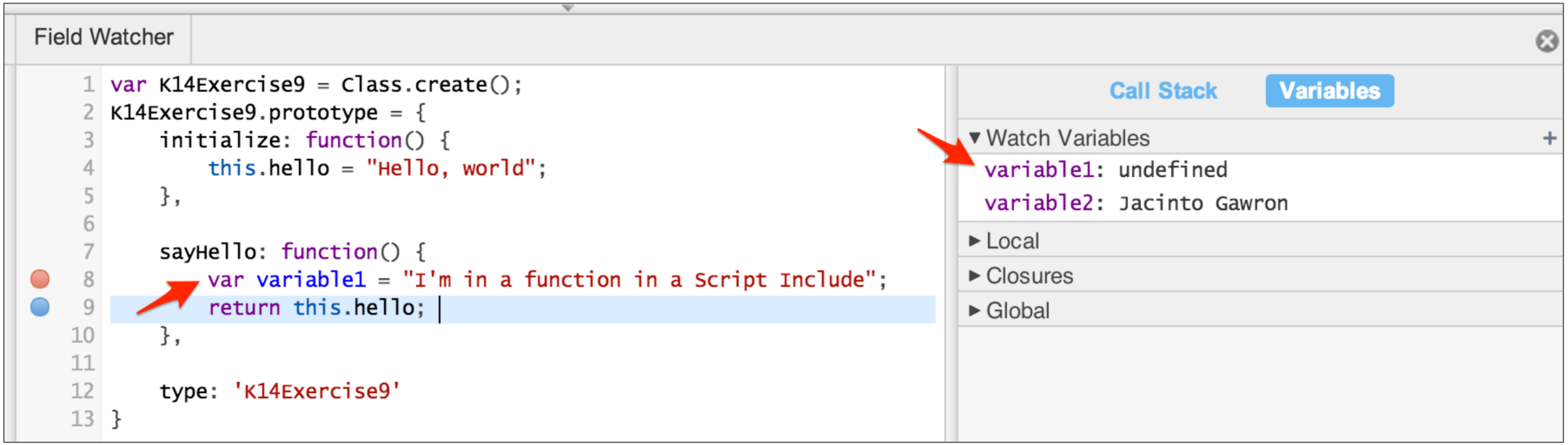
© 2014 ServiceNow All Rights Reserved

Watching Variables

© 2014 ServiceNow All Rights Reserved

Watching Variables

- Quick and easy way to keep track of specific variables
- Easier than drilling into the various variable scopes



© 2014 ServiceNow All Rights Reserved

Lab 9.1

Watching Variables

© 2014 ServiceNow All Rights Reserved

Quote

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

Brian Kernighan

© 2014 ServiceNow All Rights Reserved

Q & A

Questions?

© 2014 ServiceNow All Rights Reserved

Top Takeaways

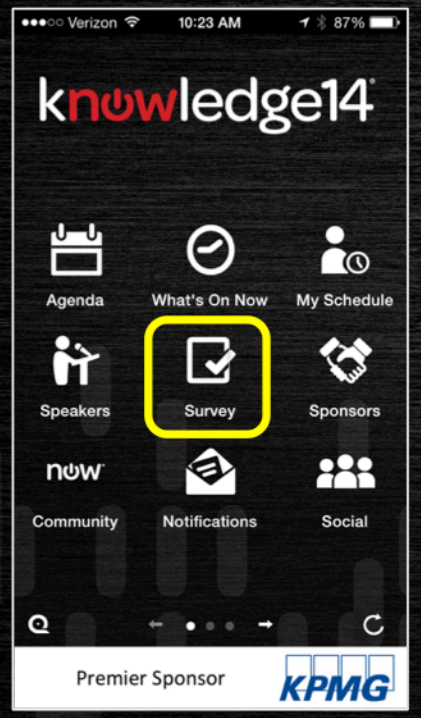
1. Use debugging tools early and often

2. Assess the performance implications of client scripts and avoid blocking code

3. Utilize the new JavaScript debugger and Field Watcher to improve debugging workflow

© 2014 ServiceNow All Rights Reserved

Tell us what you think!



Click on the survey link in your Knowledge14 mobile app. Help us make Knowledge 15 even more valuable!

Thank You

Chris Maloy

Expert Services Team Lead

ServiceNow

chris.maloy@servicenow.com

Josh Nerius

Expert Services Consultant

ServiceNow

josh.nerius@servicenow.com

Want a copy of this deck?

Sign into the BuzzHub to unlock all the slides and lab guides from Knowledge14 sessions

