Debugging Business Rules

Lab 5.1 30-35 minutes

Lab objective

You will perform the following in this lab:

- Debug Business Rules using several different strategies.

> **Note:** *Should the Script Debugger lose connection to the script you are debugging at any time, simply close and re-open the window.*

A. Preparation

1. Navigate to **System Definition > Business Rules**.
2. Locate and open the **Lab 5.1 Business Rule Debugging** Business Rule.
3. Select the **Active** checkbox to make the Business Rule active.
4. On the Advanced Tab, overwrite the string **<your_initials>** with **your personal initials** in the Script. For example:

```
catch(err){
    gs.error("jb: a JavaScript runtime error occurred - " + err);
}
```

5. **Save** the record to remain on the form.
6. Review the Business Rule and read the script so you understand:
   - **when** it triggers
   - **what** it executes
7. Consider saving this Business Rule as a Favorite as you will be opening the record often throughout the lab. Select **Create Favorite** on the form's Context menu.

B. Practice Using the Script Debugger – Breakpoints and Variables

1. In the *Script* field, set breakpoints by clicking the gutter to the left of the lines beginning with:

   - **var myNum**
   - **var priorityValue**
   - **var createdValue**

```
1  current.short_description = "This text set by the Lab 5.1 Business Rule Debugging BR";
2  var myNum = current.state;
3
4  //Advise logged in user when Incident was created
5  var priorityValue = current.priority;
6  var createdValue = current.sys_created_on;
7
8  SlaTargetNotification(priorityValue,createdValue);
```

2. Select the **Open Script Debugger** ( ) icon on the Syntax Editor toolbar or the **System Diagnostics > Script Debugger** module on the Application Navigator. *The Script Debugger opens in another browser window.*

3. Notice the list of all breakpoints set in the platform by you appears on the left-side of the code pane.

4. Set another breakpoint by clicking the gutter to the left of the line beginning with **SlaTargetNotification**.

```
1  current.short_description = "This text set by the Lab 5.1 Business Rule Debugging BR";
2  var myNum = current.state;
3
4  //Advise logged in user when Incident was created
5  var priorityValue = current.priority;
6  var createdValue = current.sys_created_on;
7
8  SlaTargetNotification(priorityValue,createdValue);
```

   What can you conclude about where breakpoints can be set? Record your answer here. _____

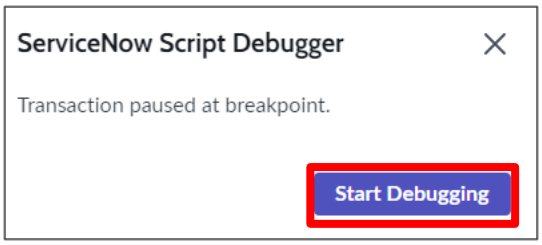
5. **Keep the Script Debugger window open**. Force the *Lab 5.1 Business Rule Debugging* Business Rule to execute.

   a) Open any active Incident and change the value of **State** to anything except Closed.

   b) **Save** the record to remain on the form.

6. Select **Start Debugging**.

> **Note:** *Had the Script Debugger window not been open, the script would have executed normally without stopping at any breakpoints.*

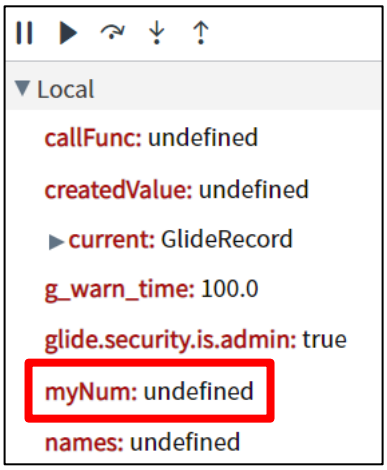


7. Script execution is paused at the first breakpoint. How can you tell? Explain your reasoning:

   _____

8. Examine the value of the **myNum** variable in the *Local* variables section on the right-side of the window. Does it contain the value you expected?

   _____



9. Select the **Next Breakpoint** button ( ) to resume script execution and pause at the next breakpoint.

10. Re-examine the value of the **myNum** variable in the *Local* variables section. What value does it contain now? _____

11. Explain why the **myNum** variable value is sometimes *undefined* and other times contains a value:

    _____

    _____

12. Repeat steps 9-11 to watch the values of the **priorityValue** and **createdValue** variables update as you step through the code.

> **Note:** *Specifically notice the value of* ***createdValue****. This type of output is extremely valuable when you are scripting dates. For example, you may write a script, such as:*
>
> `if (current.sys_created_on <= 'some value')…`
>
> *and it does not work. This output will quickly confirm the syntax/value in the [sys_created_on] field and you can then adjust the value of 'somevalue' accordingly.*

13. When you reach the breakpoint on the line beginning with *SlaTargetNotification*, select the **Next Breakpoint** button one last time to complete the execution of the script.

14. Think of an example in the past where you declared variables in a script and they did not return the values you were expecting. Would it have been helpful to have variable value output available like this?

    _____

C. Practice Using the Script Debugger – Current and Previous Values

1. **Keep the Script Debugger window open**. Open any Incident with a *Short description* value other than "This text set by the Lab 5.1 Business Rule Debugging BR".

2. Force the *Lab 5.1 Business Rule Debugging* Business Rule to execute:

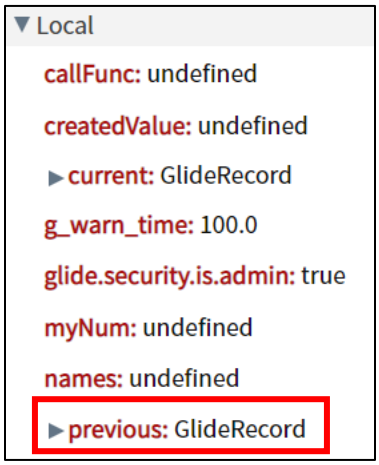
   a) Change the **State** value to anything except Closed.

   b) **Save** the record to remain on the form.

3. Select **Start Debugging**.

4. Document the **previous** object's *short_description* field value.

   a) Expand the **previous: GlideRecord** object in the *Local* variables list.

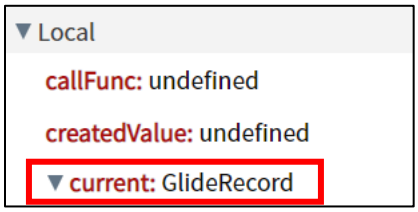


   b) Record the previous value of the short_description field here:

   _____

5. Document the **current** object's *short_description* field value.

   a) Expand the **current: GlideRecord** object in the *Local* variables list.



   b) Record the current value of the **short_description** field here:

   _____

6. Why are they different? Explain your reasoning:

   _____

   _____

7. Select the **Next Breakpoint** button as many times as needed to complete the execution of the script.

8. Close the Script Debugger window.

D. Practice Using the Script Debugger – Call Stack

In this section, you will practice using the Script Debugger when one server-side script calls another server-side script. This feature can help you pinpoint exactly which script needs fixing when multiple scripts are executing.

1. Navigate to **System Definition > Script Includes**.

> **Note:** *You will learn about Script Includes in an upcoming module. For now, you are only adding a breakpoint to the script.*

2. Locate and open the **SlaTargetNotification** Script Include.

3. Set a breakpoint in this script by clicking in the gutter to the left of the line beginning with **gs.info**.

```
1  function SlaTargetNotification(priorityLevel, sysCreatedOn) {
2      var loggedInUser = gs.getUserDisplayName();
3
4      gs.info(loggedInUser + ", this Priority-" + priorityLevel + " Incident has been open since "
   + sysCreatedOn);
5  }
```

4. Select the **Open Script Debugger** button on the Syntax Editor toolbar. *The Script Debugger opens in another window.*

5. Notice the list of breakpoints now includes the one you just set in step 3.

6. Select any one of the *Lab 5.1 Business Rule Debugging* lines to load that script in the code pane.

7. Update the breakpoins so only the line of code beginning with **var createdValue** and **try** are set.

Business Rule > **Lab 5.1 Business Rule Debugging**

```
1   current.short_description = "This text set by the Lab 5.1 Business Rule Debugging BR";
2   var myNum = current.state;
3
4   //Advise logged in user when Incident was created
5   var priorityValue = current.priority;
6   var createdValue = current.sys_created_on;
7
8   SlaTargetNotification(priorityValue,createdValue);
9
10
11  // The function in this try/catch is not defined
12  try{
```

8. **Keep the Script Debugger window open**. Force the *Lab 5.1 Business Rule Debugging* Business Rule to execute:

   a) Open any active Incident and change the **State** value to anything except Closed.

   b) **Save** (not Submit) the record to remain on the form.

9. Select **Start Debugging**.

10. You are stopped at the first breakpoint in the *Lab 5.1 Business Rule Debugging* script. Review the information displayed in the **Call Stack** and **Transaction Detail** on the left-side of the screen.

11. How can you be sure which script you are currently debugging? What does the **Code Pane Header** read? Record your answer here:

    _____

12. Notice the very next line in the script after the breakpoint you are currently paused at calls the *SlaTargetNotification* Script Include. Select the **Next Breakpoint** button one time.

13. Notice the information in the **Call Stack** now includes the Script Includes breakpoint details.

14. How can you be sure which script you are currently debugging? What does the **Code Pane Header** read? Record your answer here:

    _____

15. Select the **Next Breakpoint** button one time.

16. Notice the information in the **Call Stack** on the left-side of the screen changed.

17. What does the **Code Pane Header** read? Record your answer here:

    _____

18. Select the **Next Breakpoint** button one last time to complete the execution of the script.

19. Close the Script Debugger window.

E. Practice Debugging Using GlideSystem Logging Methods

1. Open the **Lab 5.1 Business Rule Debugging** Business Rule.

2. Select the breakpoints to remove them. There should be no breakpoints set in the script after this step is complete.

3. Save the record. Notice two undefined functions are called in this script:

   - **thisFunctionDoesNotExist()** is in a try/catch.
   - **thisFunctionAlsoDoesNotExist()** is NOT in a try/catch.

```
// The function in this try/catch is not defined
try{
    thisFunctionDoesNotExist();
}
catch(err){
    gs.error("<jb!!!: a JavaScript runtime error occurred - " + err);
}

// This function is not defined and is not part of a try/catch
thisFunctionAlsoDoesNotExist();
```

   Predict what will occur when the Business Rule executes. Record your answer here:

   _____

4. Force the *Lab 5.1 Business Rule Debugging* Business Rule to execute:

   a) Open any active Incident and change the **State** value to anything except Closed.

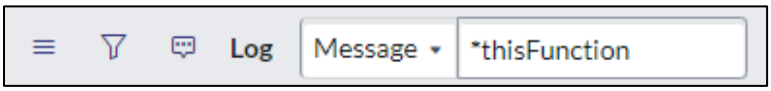
   b) Select **Update**.

5. Open **System Logs > System Log > Script Log Statements**. Which undefined function produced an error and why?

   _____

6. Was a log message produced for the undefined thisFunctionAlsoDoesNotExist() function? _____

7. Search the log for records where **Message** contains **thisFunction**.



8. Did you find the log message advising *thisFunctionAlsoDoesNotExist()* is not defined?

_____

> **Note:** *If you are debugging using the Script Debugger, note that undefined functions not wrapped in a* ***try/catch*** *stop script execution and the remaining breakpoints are ignored. If this happens to you, check this list instead to get more information about why a function call may have failed.*

F. Practice Debugging Using Script Tracer

1. Navigate to **System Diagnostics > Script Tracer**.

2. Select **Start Tracer**.

3. Force the *Lab 5.1 Business Rule Debugging* Business Rule to execute:

   a) Create a new Incident with 'Joe Employee' in the **Caller** field.

   b) Select 'Hardware' in the **Category** field.

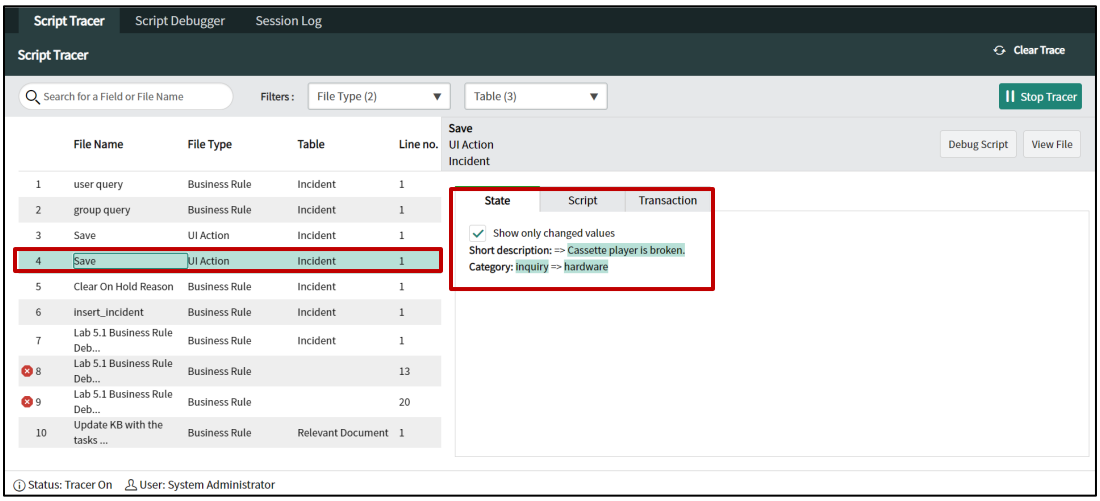
   c) Enter "Cassette player is broken" in the **Short Description** field.

   d) Select **Save** from the Context menu.

4. On the Script Tracer window, select the second occurrence of *Save - UI Action* (here, row four) under **File Name**.

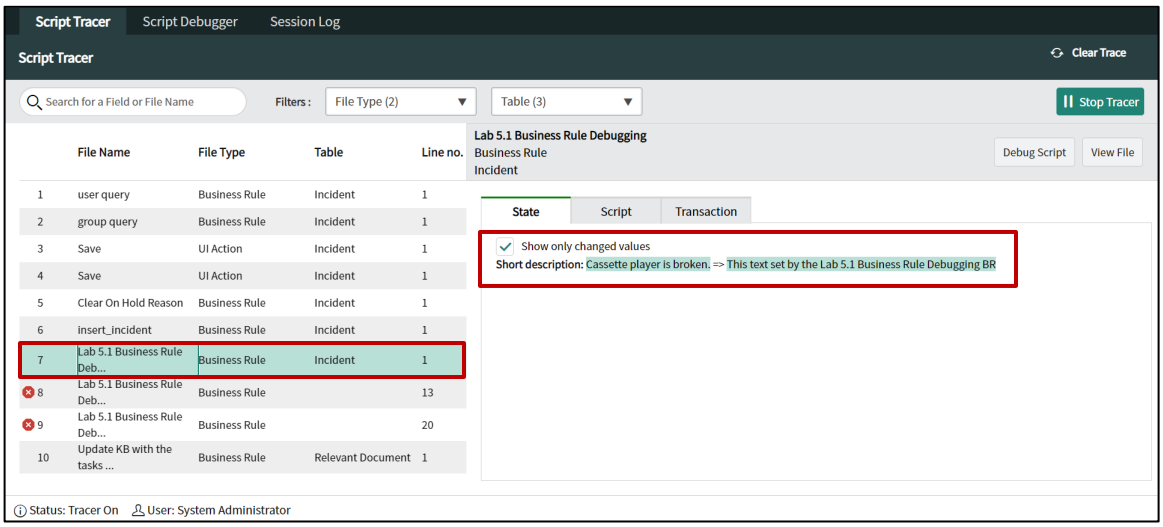
> **Note:** *Your row numbers may vary from those shown in this Lab.*

   a) Review each Script Tracer tab (State, Script, and Transaction) contents.

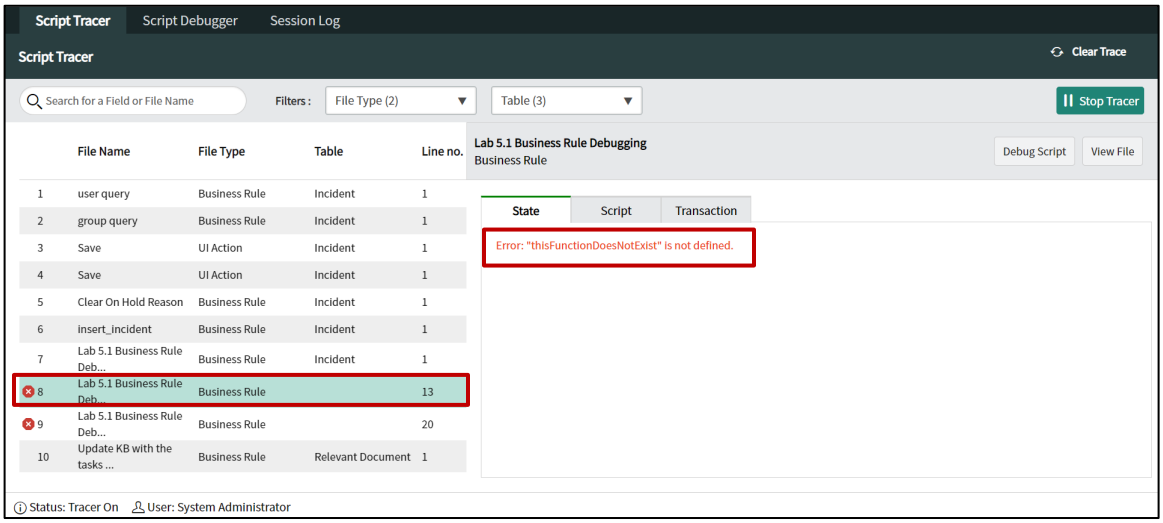


5. Verify that the category and short description match what was initially on the Incident.

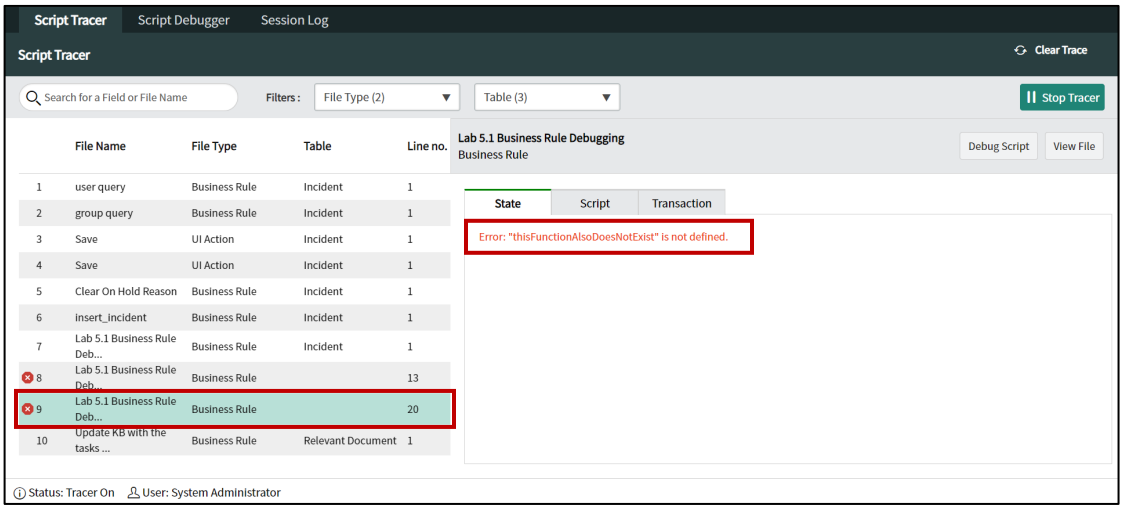
6. Select the first occurrence of *Lab 5.1 Business Rule Debugging* (here, row seven), under **File Name**. Did the short description change according to the Business Rule? If not, debug and retest.



7. Select the first error message on row eight and review the **State**, **Script,** and **Transaction** tab contents.



8. Select the second error message on row nine and review the **State**, **Script,** and **Transaction** tab contents.



9. How do these debugging results compare to the GlideSystem logging method results? _____

10. Close the Script tracer window and move to the next step.

G. Practice Debugging Using the Debug Business Rule Feature

1. Select **System Diagnostics > Session Debug > Debug Business Rule**. The Script Debugger and Session Log open in another browser window.

2. The condition for the execution of *Lab 5.1 Business Rule Debugging* Business Rule is current.state !=7. Update a record that will not trigger the Business Rule.

   a) Open a closed Incident and make any change to the record.

   b) **Save** the record to remain on the form.

   Select the Session Log tab and search for the execution of the Lab 5.1 Business Rule Debugging Business Rule by searching for the string ==> **Lab 5.1 Business Rule Debugging**. Did your test meet the Business Rule's Condition criteria? How can you tell? _____

3. Disable Business Rule Debugging: **System Security > Debugging > Stop Debugging**.

4. Close the Script Debugger and Session Log window and make the *Lab 5.1 Business Rules Debugging* Business Rule **inactive**.

Lab Completion

Well done! You have successfully practiced testing scripts using several different server-side debugging techniques and freed the sock monkeys.