

Debugging

This page shows how to inspect values while a script runs, how to read error messages, and how to find slow dataflow blocks.

Topics on this page:

- [print](#)
- [printConsole](#)
- [Errors](#)
- [Profiling: startProfile, printProfile, stopProfile, profileModelTree](#)

print

`print(value1, value2, ...)`

Adds text to the **print** property of the block that runs the script.

- **value1, value2, ...**: any values. Each value is converted to text and written directly after the previous one. `print` does **not** add spaces between values or a line break at the end. Add `"\n"` yourself if you want separate lines.
- **Returns** null.

Lists print as `[1, 2, 3]`, objects as `{a: 1}`, and null as `null`.

The **print** property is updated once, when the script finishes. If the script stops with an error, the **print** property shows the error message instead, and the text printed before the error is lost.

```
print("2 + 2 = ", 2 + 2);
print("done");
return "ok";
// print property: 2 + 2 = 4done
```

```
var items = ["a", "b", "c"];
for (var i = 0; i < items.length; i++) {
  print(i, ": ", items[i], "\n");
}
return items.length;
```

`print` also works in the scripts of the [Filter](#), [Column Mapping](#), [Edit Rows](#), and [Remove Rows](#) blocks. The text of all rows is collected in the **print** property of that block.

When a Script block is [called as a function](#), `print` does nothing. Use `printConsole` instead.

printConsole

```
printConsole(value1, value2, ...)
```

Writes a line to the browser's developer console (open it with F12 in most browsers).

- **value1, value2, ...**: any values. They are joined with " , " (space, comma, space).
- **Returns** null.

`printConsole` works everywhere, including scripts that are called as functions, so it is the best way to debug a reused script.

```
/* Prints the following lines to the browser console:  
1  
2  
3  
*/  
var json = JSON.parse("[1,2,3]");  
for (var i = 0; i < json.length; i++) {  
    printConsole(json[i]);  
}
```

```
printConsole("width", @parent.width); // prints: width , 100
```

Errors

DG Script has no `try/catch`. An error stops the script.

Errors While Running

When a Script block stops with an error:

- **output** becomes null;
- **print** shows the error message;
- the error is also written to the browser console.

When a Script block is called as a function and stops with an error, the call returns null and the error is written only to the browser console.

```
var x = null;
return x.length;
// output: null
// print: can not access length of null
```

Syntax Errors

If the script cannot be parsed, it does not run at all. **print** shows the error, and **output** keeps its previous value. The number in parentheses is the position of the problem, counted in characters from the start of the script (the first character is 0).

```
return 1 +;
// print: Unexpected token: SEMICOLON (10): ;
```

Common Messages

Message	Meaning
Unexpected token: ...	Syntax error. Check for unsupported syntax (see What is not supported), a missing bracket, a reserved word used as a name, or a missing semicolon after return.
Unexpected character ...	A character that DG Script does not accept, such as a backquote, #, or a non-English letter in a name.
Unterminated string ...	A string has no closing quote before the end of the line.
can not access X of null	You read property X of a value that is null, for example a component name that does not exist.
can not access 5 of [1, 2]	You read a list index that is past the end of the list.
invalid call to ...	You called something that is not a function, for example a misspelled function name, or a function before its declaration.
~= not supported for this type	You used ~= on something that is not a page model property.
Program is empty	The script property is empty.

```
var list = [1, 2];
return list[5];
// print: can not access 5 of [1, 2]
```

Tip

To find which line fails, add print calls. Because an error replaces the print text, use `printConsole` when you need to see the output from before the error.

Profiling

Use these functions to find dataflow blocks that take a long time to run. Profiling measures every dataflow block that runs while profiling is on, including Script blocks, table blocks, and other logic blocks.

startProfile

`startProfile()`

Starts recording how long each dataflow block takes to run. **Returns** null.

printProfile

`printProfile(count)`

Returns a report of the count blocks that used the most total time since profiling started, and also writes it to the browser console.

- **count**: number of blocks to list. Required.
- **Returns** a string. The first line is `profile result:`. Each following line has the form `t:<time> c:<runs> <path> (<type>)`, where `<time>` is the total time in **microseconds**, `<runs>` is the number of runs, and `<path>` is the page model path of the block. Blocks inside symbols start with the symbol name and `::`. Returns null if count is not a number.

stopProfile

`stopProfile()`

Stops profiling and **deletes** the recorded data. Call `printProfile` before `stopProfile`. **Returns** null.

Example

Create two Script blocks. In the first, start profiling:

```
startProfile();
```

Use the page for a while, then run the second:

```
var report = printProfile(10);
stopProfile();
return report;
```

A result might look like this:

```
profile result:
t:15300 c:12 page.dataflow.filter1 (tableFilter)
t:4200 c:40 page.dataflow.script1 (script)
```

profileModelTree

profileModelTree(object)

Returns an object that describes the page model tree under object. Use it to find parts of a page that contain many objects.

- **object**: a page model object, such as @stage or @parent.
- **Returns** an object with:
 - @type: the type of the object;
 - @count: the number of objects in this subtree, including the object itself;
 - one key for each child object, holding the same kind of description. Children of repeaters and groups use the keys .0, .1, and so on.
- Returns null if object is not a page model object.

```
var tree = profileModelTree(@stage);
return JSON.stringify(tree);
```

```
var tree = profileModelTree(@);
return tree["@type"]; // "script"
```

[Previous: Page Model](#)

[Next: Strings](#)

From:

<http://wiki.dglogik.com/> - **DGLogik**

Permanent link:

http://wiki.dglogik.com/dglux5_wiki:dgscript:debugging

Last update: **2026/10/22 17:28**

