JavaScript

JavaScript is available as of version 8.24.2 for scripts created as part of an operation (in other words, as a project item). At this time, JavaScript is not available for use within transformation mappings or the script pad. See Creating a Script for instructions on creating a new JavaScript and using the script builder in Jitterbit.

JavaScript Support in Jitterbit

Jitterbit's Javascript engine supports the ECMA-262 v5.1 standard as specified by ECMA International. This version of JavaScript has native support of JSON and the definition and use of functions inside scripts. Jitterbit's JavaScript conforms to standard JavaScript object manipulation and behavior.

WARNING: While Jitterbit supports JavaScript based on the standard, not all JavaScript functionality is available. Jitterbit does **not** support these JavaScript features:

- Document Object Model (DOM) web APIs
- Mozilla built-in functions and objects
- Certain JavaScript types such as `Set` and `Map`
- Access to Java objects

Simple data types, arrays, and JSON objects are fully supported. Jitterbit maps are also supported within JavaScript. JavaScript treats Jitterbit maps as JSON objects, and Jitterbit Scripts treat JSON objects as Jitterbit maps. JSON properties are accessed via map keys.

For example, given this JSON object defined in JavaScript:

```
var $myObj = {
 "name":"John",
 "age":30,
 "cars": {
   "car1":"Ford",
   "car2":"BMW",
   "car3":"Fiat"
   }
 };
```

In a Jitterbit Script, the object would be accessed by a map. Access the `"car3"` property like this:

```
$mycar = $myObj["cars"]["car3"];
```

Creating a New JavaScript

Within your project within Jitterbit Studio, there are several ways to create a new JavaScript as a project item:

- In the tree on the left, right-click on the **Scripts** section and select **New JavaScript**.
- Within an existing operation, right-click on any insertion point and choose **Insert** > **Script**. Or right-click on a source or target and choose **Insert Before This** > **Script** or **Insert After This** > **Script**. A script icon will appear within the operation graph. Then do one of the following:
  - Right-click on the script icon and choose **Create New JavaScript**.
  - Double-click on the script icon and click the button **Create New JavaScript**.

Using Script Builder

After you have created a new JavaScript in Jitterbit, you can enter your JavaScript directly within the script area.

NOTE: Unlike a Jitterbit Script, which requires that the script be wrapped within `<trans> ... </trans>` tags, there are no tags needed for scripts in JavaScript. Instead, simply enter your JavaScript directly within the script area.

On This Page

Jitterpak

Netsuite_to_Hubspot_with_JavaScript.jpk

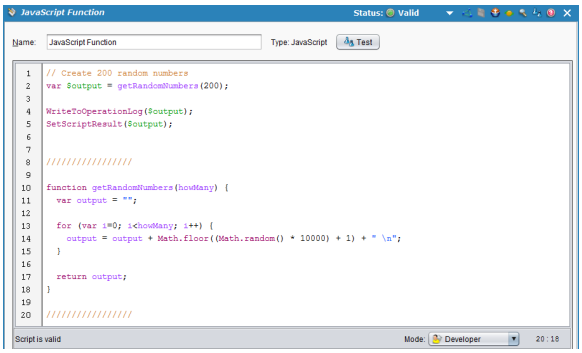
Related Articles

- JavaScript
- JavaScript Jitterbit and Common Functions
- JavaScript Standard Properties and Functions

Related Topics

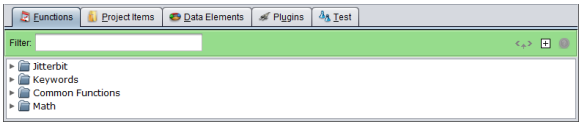
- Formula Builder
- Formula Builder Debugging
- Formula Builder Features
- Jitterbit Variables
- Project Variables
- Scripting
- Using the Formula Builder

Last updated: Feb 05, 2020



Functions

The functions available to use in a JavaScript are available under four categories.



Jitterbit

This category contains a list of functions specific to Jitterbit. These include a limited number of standard Jitterbit functions, as well as a number of JavaScript functions specific to Jitterbit.

Currently, a limited number of the Jitterbit Script functions are available for use in JavaScript. To access one of those Jitterbit function in your JavaScript, prefix the function with "`Jitterbit.`". These Jitterbit functions are available for use in JavaScript in Jitterbit:

- **`Jitterbit.ReadFile(string source_id[, string file_name])`**
  Reads a file from the specified source as described in Formula Builder File Functions. The returned value is a string. For example, in JavaScript, using this Jitterbit function to read a file:

  ```
  var MyData = Jitterbit.ReadFile("<TAG>Sources/myfile</TAG>");
  ```

- **`Jitterbit.WriteFile(string target_id, type file_contents[, string file_name])`**
  Writes to a file and automatically flushes file. No value is returned. See WriteFile under Formula Builder File Functions.
- **`Jitterbit.DbExecute(string database_id, string sql_str, ...)`**
  Executes a SQL statement on a database and returns the results. The returned value is a string. See DbExecute under Formula Builder Database Functions.
- **`Jitterbit.DbLookup(string database_id, string sql_str)`**
  Executes a SQL statement on a database and returns the first result matching the specified criteria. The returned value is a string. See DBLookup under Formula Builder Database Functions.
- **`Jitterbit.GetVar(string name)`** and **`Jitterbit.SetVar(string name, string value)`**

  > **WARNING:** The `Jitterbit.GetVar` and `Jitterbit.SetVar` functions are designed to allow the use of variables that contain periods within the variable name. However, using periods in a variable name is not recommended. As the value will be converted to a string when the variable is set, these functions cannot be used with complex data types such as arrays, dictionaries, or JSON objects. Instead, it is recommended that you create Jitterbit variables without periods and instead use underscores in place of periods and use the standard dollar sign $ syntax as described in Global Variables.

  The `Jitterbit.GetVar` function retrieves a previously assigned Jitterbit global variable (a variable prefixed with "`Jitterbit`"). The returned value is the appropriate data type.

  The `Jitterbit.SetVar` function assigns a value to a Jitterbit global variable (a variable prefixed with "`Jitterbit`"). The returned value is a string. This example uses a custom API that runs an operation containing a JavaScript using `Jitterbit.GetVar` and `Jitterbit.SetVar`:

```
var response="Hello World";

response += "\r\nProject name: " + Jitterbit.GetVar("$jitterbit.
operation.project_name");
response += "\r\nOperation Name: " + Jitterbit.GetVar("$jitterbit.
operation.name");
response += "\r\ntest_var: " + Jitterbit.GetVar("$jitterbit.api.
request.parameters.test_var");

Jitterbit.SetVar("$jitterbit.api.response.headers.content_type","text
/plain");
Jitterbit.SetVar("$jitterbit.api.response.headers.test_header",
"This is a header test");
Jitterbit.SetVar("$jitterbit.api.response",response);
```

Included in this category are several functions adapted for use with JavaScript in Jitterbit. These functions are not prefixed with "`Jitterbit.`".

- **`WriteToOperationLog`**
  Sends output to the current Jitterbit operation log. Created for use with JavaScript in Jitterbit, and works similarly to the Jitterbit function described in Formula Builder Logging and Error Functions.
- **`SetScriptOutput`** and **`SetScriptResult`**
  Sets the output for the script. These two functions are aliases, and either can be used depending on your preference. Unlike Jitterbit Scripts, which automatically return the value of the last line, Jitterbit JavaScripts do not automatically return a value. You can use either of these functions in your JavaScript to return the script result. For example:

```
var MyVariable = "Hello World";
WriteToOperationLog(MyVariable);
SetScriptResult(MyVariable);
```

Keywords

This category contains a list of JavaScript keywords in the ECMAScript 5.1 standard, and are included here for convenience.

Common Functions

This category contains a list of JavaScript functions in the ECMAScript 5.1 standard that may be relevant to Jitterbit users. This category is not comprehensive – you can use other JavaScript functions that are not listed here, such as JavaScript `Array`, `Date`, and `String` objects and their associated functions.

> **NOTE:** For a comprehensive list of JavaScript functions, refer to the ECMAScript 5.1 standard at https://www.ecma-international.org/ecma-262/5.1/.

Math

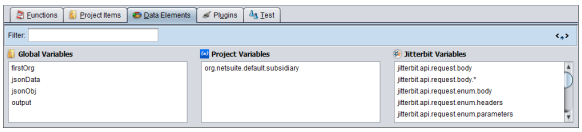
This category contains function properties available for the JavaScript `math` object as specified in the ECMAScript 5.1 standard. This category is provided for ease of reference; for the comprehensive documentation, see the ECMAScript 5.1 standard.

Project Items

In JavaScript, you can access project items such as sources and targets just as you would from a Jitterbit Script. Though Jitterbit Scripts can call Jitterbit JavaScripts, the reverse is not true at this time. Jitterbit JavaScripts cannot call other scripts or operations.

Data Elements

This tab provides access to data elements that are available globally to use throughout your project, including Global Variables, Variables, and Jitterbit Variables.



Global Variables

All Jitterbit variables can be accessed and updated from your JavaScript. Any newly defined JavaScript global variables will become Jitterbit global variables.

The syntax used for setting and retrieving a global variable depends on whether the global variable name contains a period.

> **WARNING:** The `Jitterbit.GetVar` and `Jitterbit.SetVar` functions are designed to allow the use of variables that contain periods within the variable name. However, using periods in a variable name is not recommended. As the value will be converted to a string when the variable is set, these functions cannot be used with complex data types such as arrays, dictionaries, or JSON objects. Instead, it is recommended that you create Jitterbit variables without periods and instead use underscores in place of periods and use the standard dollar sign $ syntax as described in Global Variables.

> **NOTE:** Additional information on the `Jitterbit.GetVar` and `Jitterbit.SetVar` functions is in the previous section on Functions.

Setting a Global Variable

- **Names without periods (recommended):** A global variable that does not contain any periods in its name can be created initially or updated using the command `var $`, or updated using a dollar sign $ without `var`.
  - **`var $:`** Using `var` and beginning with a dollar sign `$`, the code example `var $serverURL="https://www.example.com"` creates or updates a global variable called "`serverURL`" with a value of "`https://www.example.com`". New global variables that are being initialized must precede the $ with `var`.
  - **`$:`** Prefixed with a dollar sign `$`, the code example `$serverURL="https://www.example.com"` updates the same global variable called "`serverURL`" with the same URL. This works only for global variables that are already initialized.
- **Names with periods (not recommended):** A global variable that contains periods in its name can be created initially or updated only with the `Jitterbit.SetVar` function.
  - **`Jitterbit.SetVar:`** Using `Jitterbit.SetVar`, the code example `Jitterbit.SetVar("$server.URL", "https://www.example.com")` creates or updates a global variable called "`server.URL`" with a value of "`https://www.example.com`" that will be treated as a string. Note that the dollar sign $ **must** be included within the variable name, or the variable will not be global to the Jitterbit system.

Getting a Global Variable

- **Names without periods:** The value of a global variable that does not contain any periods in its name can be retrieved by prefixing with a dollar sign $.
  - **`$:`** Prefixed with a dollar sign $, the code example `$serverURL` retrieves the value of the global variable "`serverURL`".
- **Names with periods:** The value of a global variable that contains periods in its name can be retrieved only with the `Jitterbit.GetVar` function.
  - **`Jitterbit.GetVar:`** Using `Jitterbit.GetVar`, the code example `Jitterbit.GetVar("$server.URL")` returns the string value of the global variable called "`server.URL`". Note that the dollar sign $ **must** be included within the variable name to read the global value from the Jitterbit system.

Project Variables

Project variables are first created as a project item within Design Studio. Once a project variable is created, you may set values for them either through Design Studio, the Management Console, or Citizen Integrator. Learn more about creating and updating project variables under Project Variables.

In Jitterbit JavaScript, the syntax used for retrieving the value of a project variable depends on whether the project variable name contains a period.

- **Names without periods:** The value of a project variable that does not contain any periods in its name can be retrieved by beginning with a dollar sign $.
  - **`$:`** Prefixed with a dollar sign $, the code example `$org_netsuite_auth_username` retrieves the value of the project variable called "`org_netsuite_auth_username`".

- **Names with periods:** The value of a project variable that contains periods in its name can be retrieved only with the `Jitterbit.GetVar` function.

    - **`Jitterbit.GetVar:`** Using `Jitterbit.GetVar`, the code example `Jitterbit.GetVar("$server.URL")` returns the value of the project variable called "`server.URL`". Note that the dollar sign $ **must** be included within the variable name.

Jitterbit Variables

The Jitterbit system defines certain global variables that are always available throughout your project, known as Jitterbit variables (or known as predefined global variables). These can be used to fetch global information such as the name of the current source file or the name of the current operation. These are documented individually under the Jitterbit Variables section.

In Jitterbit JavaScript, Jitterbit variables predefined by Jitterbit are accessible only with the `Jitterbit.GetVar` function. This is because all Jitterbit variables predefined by Jitterbit contain periods within the variable name.

- **`Jitterbit.GetVar:`** Using `Jitterbit.GetVar`, the code example `Jitterbit.GetVar("$jitterbit.operation.error")` retrieves the value of the Jitterbit variable "`jitterbit.operation.error`". Note that the dollar sign $ **must** be included within the variable name.

Plugins

Currently, using plugins in a JavaScript is not supported. Instead, use a Jitterbit Script and call both the plugin and your JavaScript from it.

Testing and Debugging

In a JavaScript, you can see and validate the values of variables just as you would for a Jitterbit Script. However, the **Disable breakpoints** option is not applicable since debugging is not available in JavaScript. When testing, the results of the script (as set by either the **`SetScriptOutput`** or **`SetScriptResult`** functions) will appear in the "Result" field of the Design Studio "Test" tab.

Examples

The following JavaScript examples are provided for reference.

JavaScript File Functions

JavaScript File Functions

```
// This script will:
// * Generate some random numbers
// * Write them to a target file
// * Then read them back in
// * Write output to the Operation Log
// *********************************************

// Get 200 random numbers between 1 and 10000
var mystring = getRandomNumbers(200,1,10000);

// Write the data to a file
Jitterbit.WriteFile("<TAG>Targets/tmpdata</TAG>", mystring);

// Read the data back in from the file
var filedata = Jitterbit.ReadFile("<TAG>Sources/tmpdata</TAG>");

// Output to the Operation Log
WriteToOperationLog("Read file, output: " + filedata);

// Displays the data in the result of the Studio test script tab
SetScriptResult(filedata);

/////////////////

function getRandomNumbers(howMany,min,max) {
  var output = "";

  for (var i=0; i<howMany; i++) {
    output = output + getRandomNumber(min,max) + " \n";
  }

  return output;
}

function getRandomNumber(min,max) {
  return Math.floor((Math.random() * max) + min);
}

/////////////////
```

JavaScript Math Functions

JavaScript Math Functions

```
// Create 200 random numbers
var $output = getRandomNumbers(200);

WriteToOperationLog($output);
SetScriptResult($output);

/////////////////

function getRandomNumbers(howMany) {
  var output = "";

  for (var i=0; i<howMany; i++) {
    output = output + Math.floor((Math.random() * 10000) + 1) + " \n";
  }

  return output;
}

/////////////////
```

JavaScript Loop

JavaScript Loop

```
// Create 100 random numbers

var $output = "";

for (var i=0; i<100; i++) {
  $output = $output + Math.floor((Math.random() * 10000) + 1) + " \n";
}

SetScriptResult($output);
```

The maximum number of loop iterations allowed in Jitterbit Harmony is 50,000. The maximum number of loop iterations in JavaScript is per script, not per loop.

For example, one JavaScript script containing three loops, where each loop executes 25,000 iterations, would be a total of 75,000 iterations running within the one script.

To increase the maximum number of iterations allowed in any one JavaScript script, manually add `JavaScriptMaxIterations = X`, where `X` is greater than `50000`, to `[Settings]` in `jitterbit.conf`.

For more information, see Editing the Configuration File - jitterbit.conf.

JavaScript JSON Example 1

JavaScript JSON Example 1

```
WriteToOperationLog("\n\n Parsing JSON...");

var jsonData = Jitterbit.ReadFile("<TAG>Sources/JSON_Data</TAG>");
var $jsonObj = JSON.parse(jsonData);

WriteToOperationLog("Value of 'status' is: " + $jsonObj.status);
WriteToOperationLog("Value of 'operation' is: " + $jsonObj.operation);
WriteToOperationLog("Value of 'serverUrl' is: " + $jsonObj.serverUrl);

var $firstOrg = $jsonObj.orgAttrs[0];

WriteToOperationLog("First Org ID is: " + $firstOrg.orgId);
WriteToOperationLog("First Org Name is: " + $firstOrg.orgName);
```

JavaScript JSON Example 2

JavaScript JSON Example 2

```
WriteToOperationLog("\n\n Parsing JSON...");

var jsonData = Jitterbit.ReadFile("<TAG>Sources/JSON_Data</TAG>");
var $jsonObj = JSON.parse(jsonData);

WriteToOperationLog("Status: " + $jsonObj.status);
WriteToOperationLog("Operation: " + $jsonObj.operation);

var orgs = "";
var needComma = false;

for (var i=0; i<$jsonObj.orgAttrs.length; i++) {
  if (needComma) orgs = orgs + ",";
  orgs = orgs + $jsonObj.orgAttrs[i].orgId;
  needComma = true;
}

WriteToOperationLog("Org IDs: " + orgs);

// You can modify existing JSON values
// Any changes will be reflected in the Jitterbit system as a map variable
// Here we'll insert a random number as an authentication token
var randomNumber = Math.floor((Math.random() * 10000) + 1);
$jsonObj.authenticationToken = randomNumber;
```

JavaScript JSON Example 3

NOTE: The following example uses JSON stringify to easily create property value structures and then push them to an API.

To see this script in action using a Hubspot HTTP post as an example, check out the complete Jitterpak Netsuite_to_Hubspot_with_JavaScript.jpk.

JavaScript JSON Example 3

```
// This script uses JSON stringify
// to create a property value structure
// and then pushes it to an API

var $complexAPI = {
  "properties": [
    {
      "property": "email",
      "value": $email
    },
    {
      "property": "firstname",
      "value": $firstname
    },
    {
      "property": "lastname",
      "value": $lastname
    },
    {
      "property": "website",
      "value": $website
    },
    {
      "property": "phone",
      "value": $phone
    }
  ]
}

var $outputJSON = JSON.stringify($complexAPI);
Jitterbit.WriteFile("<TAG>Targets/Example HTTP Post</TAG>", $outputJSON);
WriteToOperationLog($outputJSON);
SetScriptResult($outputJSON);
```