



This PDF is generated from authoritative online content, and is provided for convenience only. This PDF cannot be used for legal purposes. For authoritative understanding of what is and is not supported, always use the online content. To copy code samples, always use the online content.

Designer User's Guide

ECMAScript Expressions

5/18/2022

Contents

- 1 Before you start
 - 1.1 A few notes on syntax
- 2 Examples of how ECMAScript can be used
 - 2.1 Building a Dynamic TTS Prompt
 - 2.2 Controlling the Application Flow
- 3 Advanced scripting in Designer
- 4 Built-in user functions
 - 4.1 setAttributes
 - 4.2 isDataTableValueValid



- Administrator

In certain blocks, you can use ECMAScript expressions to perform dynamic operations while an application is running. For example, you can use ECMAScript expressions to assign values to variables or perform certain functions, like sorting an array.

Related documentation:

-

Before you start

To use this feature, you must have a basic level of familiarity and understanding of ECMAScript syntax and rules.

Although the terms ECMAScript and JavaScript are often used interchangeably, Designer technically supports ECMAScript and does not support JavaScript functions that are typically used for web-browser based applications, such as pop-up windows, alerts, and so on.

Designer blocks that support the use of ECMAScript expressions include:

- Assign Variables Block
- Menu Option Block
- Return Block
- Shared Module Block
- Data Tables
- Activity Block
- Milestone Block

In general, block properties do not support ECMAScript expressions unless otherwise stated. If ECMAScript expressions are not supported, you must enter a value (a string that is taken as a *literal*). This value is not evaluated at runtime. For example, in the Play Message block, the value of a TTS prompt is interpreted as a literal string.

Warning

Due to certain platform dependencies and limitations, not all ECMAScript functions are supported when used in Designer. For example, the **toLocaleString** function is not supported at this time.

A few notes on syntax

Some rules and guidelines to follow when using ECMAScript in Designer:

- Strings must be quoted ('). For example, 'hello' is a string, whereas hello is a reference to a variable called **hello**.
- Single quotes (') are recommended, as opposed to double quotes (").
- When specifying an object in JSON notation, surround the JSON with parentheses. For example: `{'abc': 'def'}`.

Use caution! Any errors in your script can cause erratic behavior, so test your changes to make sure that your script works correctly before running it in your production environment. Designer can check your script for syntax errors, but cannot validate it nor check for runtime errors that might occur when the script is executed.

Examples of how ECMAScript can be used

Below are examples of how ECMAScript expressions can be used in Designer applications.

Building a Dynamic TTS Prompt

You can use the Assign block to concatenate a string to be spoken by the application. For example, the expression below reads the caller's phone number or ID:

```
'You are calling from ' + ANI
```

Controlling the Application Flow

A Segmentation block can take ECMAScript expressions that evaluate to a Boolean value, and thus control the flow of the application.

For example, you might want to inform your customers about upcoming seasonal events and you need a way to determine the current season and whether the event is occurring within the coming week. The expression below determines whether the call was received within seven days of the event, and whether the current season is summer or autumn:

```
numDays > 7 && (isSummer | | isAutumn)
```

Advanced scripting in Designer

In blocks where it is supported, you can use the **Advanced Scripting** tab to compose more complex ECMAScript constructs, such as loops or multiple nested conditions.

Let's take a look at how advanced scripting could be used to assign a value to a variable. Here, some script has been added to the **Advanced Scripting** tab of an Assign Variables Block:

Properties - Prepare order details



This block can assign values of expressions to variables. Define a variable in the Initialize phase or block and select it in this block to assign it values or results of ECMAScript expressions. You can also call ECMAScript utility functions, such as sorting an array, and provide an input to be run through the function.

Assignments Sort Function **Advanced Scripting**

Write your ECMAScript here. Be careful - don't burn yourself!

```
1 //assume this data was retrieved from an external system using HTTP REST
2 varOrderDetails = [
3   { "item" : "Laptop bag",    quantity : 3, backordered : false },
4   { "item" : "Phone charger", quantity : 2, backordered : false },
5   { "item" : "Super rare fish", quantity : 1, backordered : true }
6 ];
7
8 var i; // a local variable that exists only in this script
9 varOrdersPrompt = ""; // use a variable defined in Initialize phase
10
11 for ( i = 0; i < varOrderDetails.length; ++i ) {
12   // 3 laptop bags .... give a space between quantity and item name
13   varOrdersPrompt += varOrderDetails[ i ].quantity + ' ' + varOrderDetails[ i ].item;
14   // its odd to hear 2 of phone charger (not chargers) - lets fix that
15   varOrdersPrompt += varOrderDetails[ i ].quantity > 1 ? 's' : '';
16
17   if ( i < varOrderDetails.length - 1 ) {
18     varOrdersPrompt += ', '; // add a comma to give TTS a short pause
19   }
20 }
```

Store the outcome of the advanced scripting evaluation in this variable

varDidScriptHaveErrors

The variable will be set to false if an error is thrown during advanced script evaluation, and true otherwise.

In this example, the script sets the variable `varOrdersPrompt` to 3 Laptop bags, 2 Phone chargers, 1 Super rare fish.

Here's how it works:

The sample script first initializes JSON data in `varOrderDetails` so that it becomes an array of three JSON objects. Each JSON object has properties — **item**, **quantity**, and **backordered**. The script then proceeds to loop through orders and forms a string to play back to the caller to notify them of their order status.

The script uses variables in two scopes:

- A scope exclusive or local to this script itself (`i`). This variable remains available only while this script runs, and then it disappears.
- Top-level variables that were defined in the **Initialize** phase remain available throughout this application flow, but not in any modules this application calls (such as `varOrdersPrompt`).

Important

Advanced Scripting is an optional feature and might not be enabled on your system. To enable this functionality, contact Genesys.

Built-in user functions

Designer also has built-in ECMAScripts that you can invoke from a Designer application, such as from an Assign or Segmentation block, to perform certain functions at runtime.

setAttributes

You can use this function to add arbitrary key-value pairs to the Session Detail Record (SDR):

```
setAttributes(key, value)
```

- If *key* is a string, a property with `key = value` is added to the attributes object in the SDR.
- If *key* is an object, all of its properties are added as individual properties in the attributes object in the SDR. In this case, *value* is ignored.

Example

```
// set first key value pair
setAttributes('key1', 'Success')

// add a second one using variables in the application
var varKeyName = "key3"
var varKeyValue = "value3"
setAttributes( varKeyName, varKeyValue)

// lets add an object which will overwrite "key1" and add "key2"
var myObject = { "key1" : "value1", "key2" : "value2" }
setAttributes( myObject )

// All these statements will finally generate this data:
attributesList {
  "key1" : "value1",
  "key2" : "value2",
  "key3" : "value3"
}
```

Usage and limitations

You can use this function in all phases of Default and Digital applications, in both Self Service and Assisted Service. However, you must adhere to the following rules:

- Do not log Personally Identifiable Information (PII).

-
- Do not log secure variables.
 - Keep the data size less than 25Kb.

Warning

Reading is not supported. Use application variables when both read and write access is required.

isDataTableValueValid

You can use this function to determine if a value returned from a data table query is valid. For example, you might use the following function in an Assign block:

```
isDataTableValueValid(value, datatype)
```

This function has two arguments:

- *value* is a single value returned from a data table query
- *datatype* is the data type of the data table column, such as 'string', 'boolean', 'integer', 'announcement', or 'numeric' (this argument is optional)

If the data table value is valid, the script returns `true`. Here is a list of values that this function can return:

- `isDataTableValueValid(varStr, 'string')` on a valid (or empty) string returns `true`. Anything else returns `false`.
- `isDataTableValueValid(varNum, 'numeric')` on a valid number or 0 returns `true`. Anything else returns `false`.
- `isDataTableValueValid(varNum, 'integer')` on a valid integer or 0 returns `true`. Anything else returns `false`.
- `isDataTableValueValid(varBool, 'boolean')` on `true` or `false` returns `true`. Anything else returns `false`.
- `isDataTableValueValid(varAudio, 'announcement')` on a valid (or null) announcement returns `true`. Anything else returns `false`.