Genesys Designer Help

Segmentation Block

5/8/2025

Segmentation Block

Contents

You can use a **Segmentation** block to take a different path depending on the specific values of application variables. A valid ECMAScript expression containing application variables, ECMAScript operators, and Designer functions can be used to define a Segmentation Option. If this condition is evaluated to a *true* (Boolean) value while the application executes, the application flow takes the path of that Segmentation Option.

You can define multiple Segmentation Options, each with their own conditions. For example, the condition can be a variable with a Boolean value, a call to a function that returns a Boolean, or a combination of variables with logic operators that evaluates to a Boolean.

The first condition that evaluates successfully is selected as the segmentation path, and any blocks under that Segmentation Option are executed. If no condition expression evaluates successfully, none of the Segmentation Options execute, and the application executes the block that follows the **Segmentation** block.

Application variable values can be set based on logic in the application, by querying external data sources from blocks (such as the HTTP REST block), or by collecting input from a caller in the User Input block.

Conditions are ordered and exclusive, which means:

- Condition expressions are evaluated in the order they are defined.

- If one condition evaluates to true and the corresponding path is selected, then the following condition expressions are not tested. After executing this segment path, the application executes the block that follows the **Segmentation** block.

> ## Tip
>
> If the same logic needs to be executed in multiple segmentation paths, Genesys recommends that you keep the paths for each option independent and avoid using **GoTo** blocks to jump between paths. The common logic can be moved into a Shared Module, which can then be called from multiple paths. This improves the structure and reliability of your application.

The **Segmentation** block selects the first segment whose condition is a valid ECMAScript expression that evaluates to *true* (Boolean). If none of the conditions evaluate to *true*, no segment is executed, and processing moves on to the next sibling of the **Segmentation** block.

> ## Warning
>
> You must use condition expressions that evaluate to a Boolean value. Expressions that evaluate to a different data type can result in errors.

The following are valid expressions:

- Using a variable whose value is *true* or *false* and comparing it to a Boolean value, such as the variable used to hold the result of a **Special Days** block:

```
isSpecialDayVar == true
```

or:

```
isSpecialDayVar == false
```

- Using a Boolean property of an object stored in a variable, such as the **Route Call** block outcome variable:

```
routeCallOutcomeVar.success == true
```

- An expression using Boolean variables and logical operators (&&, ||):

```
var1 == false || (var2 == true && var3 == true))
```

- An expression using comparison operators (==, ===, !=, !==, >, <, >=, <=):

```
var1.length > 3 || var2 === 'stop'
```

Important

Do not use an expression that does not have explicit comparison operators (such as `varIsHoliday`).

When using condition expressions that do not evaluate to a Boolean value, the following rules apply:

Important

- These rules are general ECMAScript/Javascript rules, and apply as-is to Designer. (This not the regular "flavor" of Javascript that runs in a browser. Instead, this Javascript is executed by Genesys platform components and has certain restrictions.)

- It is mandatory to ensure these expressions evaluate to a Boolean value and not to other data types or values, such as *undefined*.

- The condition expression evaluates to an object => the condition is considered *true* (this applies to arrays, even if they are empty). Instead, use the following:

```
typeof myVar === 'object'
```

- The condition expression evaluates to *undefined* => the condition is considered *false.* Instead, use the following:

```
myVar !== undefined
```

- The condition expression evaluates to *null* => the condition is considered *false*. Instead, use the following:

```
myVar !== null
```

- The condition expression evaluates to a number => the condition is considered *false* if the value is +0, -0, or NaN; otherwise, the condition is considered *true*. Instead, use the following:

```
myVar === 3
```

- The condition expression evaluates to a string => the condition is considered *false* if the string is empty; otherwise, the condition is considered *true*. Instead, use the following:

```
myVar.length > 0
```

Conditions tab

Click **Add Condition** and type the condition to evaluate in the **Condition Expression** field. The value can be a simple Boolean value, a variable with some Boolean content, or any valid JavaScript expression that evaluates to a Boolean. The condition expression can refer to other variables.

You can edit the **Segment Label** field to give a meaningful label to your segment. The child segment block will be named accordingly.

To remove a condition, click the trash icon for that condition in the **Segmentation** block or click the trash icon on the related child block.

> ### Important
> Always make sure the condition evaluates to a Boolean value at runtime.

Properties - Segmentation - decide how to route call

This block is used to evaluate expressions and take different paths in the application based on the outcome. E.g varZipCode==94014 can be used to take a different path vs varZipCode==95125.

·)) **Conditions**　　🔖 **Milestone**

+ Add Condition

Segment Label	Condition Expression	Delete
Sales Call	varServiceType == ''	🗑
Battery help	varServiceType == 'battery'	🗑
Electronic help	varServiceType == 'electronic'	🗑
All other cases	varServiceType == 'other'	🗑

Milestone tab

Add a Milestone to mark this key moment while the application is running, similar to within the Milestone block.