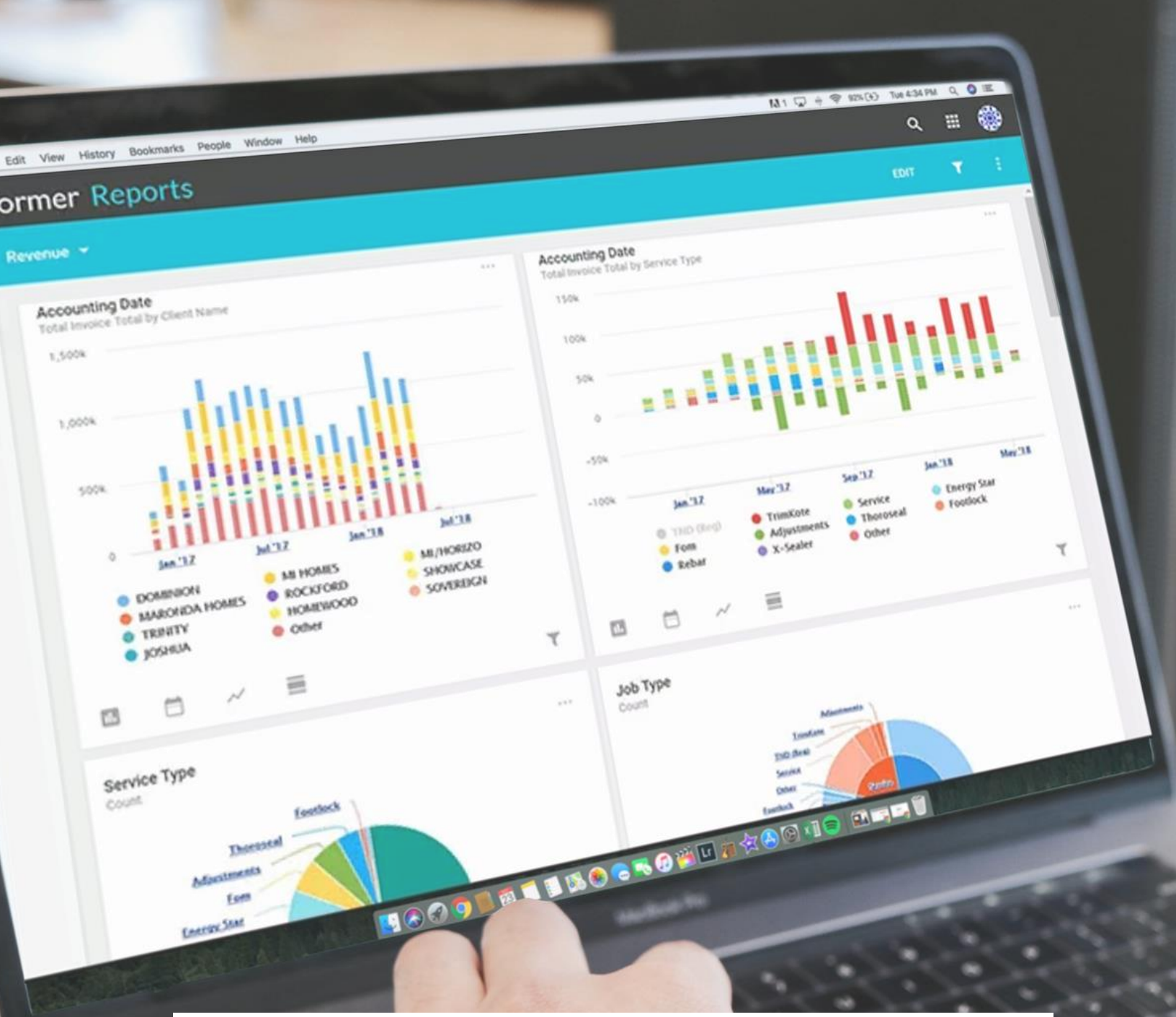


# informer

## JavaScript for Calculated Fields

Informer 5 Training

# Table of Contents


Overview ..... 2

Displaying a Value ..... 3

Variables ..... 3

Using Data in Calculations ..... 3

Data Types ..... 4

Operators ..... 4

Comments ..... 5

If, if else, and else ..... 5

Styling with CSS ..... 7

Using Data Type Functions ..... 7

Using Libraries ..... 7

Ordering Flow Steps around Merge Duplicates and Normalize ..... 8

Reusing Code ..... 9

Using Saved Functions ..... 9

Troubleshooting ..... 10

Finding More Information ..... 10

# Overview

Calculated Fields are a type of Flow Step in Informer 5 that manipulate the data returned by Queries. Calculated Fields are written in JavaScript and can be very simple or very complex. This training will cover the JavaScript elements most commonly used in Calculated Fields.

## Displaying a Value

The last value that the script sees is what displays as output. Numeric values do not need to be quoted, but text values do. A semicolon indicates the end of a line of code.

### Example:

```
"Hello World";
```

### Example:

```
15;
```

## Variables

When manipulating or creating data, variables can store that data for later use. This usually helps simplify the code and makes it easier to read. Use the keyword var to create a new variable.  Do not use the letter combination *let* or *const* to create variables in Informer.

### Example:

```
var myNewVariable= 'This is a variable!';
myNewVariable;
```

A variable's name must start with a letter or underscore (_), and it can include letters, numbers, and underscores (_). Typically, variables are named with camelCase (for instance, myNewVariable instead of mynewvariable or MyNewVariable). Variables are not restricted to any data type and can be changed from one type to another.

To display the contents of a variable, state the name of the variable at the end of the Calculated Field. As a best practice, create a variable for output and use that to track what value(s) will be displayed. This practice makes it easier to specify default values and it makes the code much easier to understand.

## Using Data in Calculations

The data from a Query is stored in predefined variables; each one named the alias of the Field. These aliases can be manually typed out or added with the "Insert Fields" button to the right of the expression window.

## Data Types

Every column in Informer has a specific data type, and each data type is handled differently by JavaScript expressions. For instance, the plus (+) operator behaves differently when used with numeric values versus text values. Below is a list of the types of Informer columns:

| Informer Type | JavaScript Type |
|---|---|
| **Text** | String |
| **Keyword** | String |
| **Full Text** | String |
| **Decimal Number** | Float |
| **Integer Number** | Integer or Int |
| **Date** | Date |
| **Timestamp** | Date |
| **Time** | Date |
| **Boolean** | Boolean |
| **Location** | Object |
| **Object** | Object |

One of the most common types of Objects is the Array, which is defined as an ordered collection of elements. Arrays in Informer most commonly take the form of multivalued data or data that has been processed by a Merge Duplicates Flow Step. There will be more on Arrays later on in the guide.

## Operators

Operators are symbols that do different actions in JavaScript. Here are some of the most common operators:

| Operator | Name | Notes |
|---|---|---|
| **+** | Addition | Adds numeric values or concatenates text values |
| **-** | Subtraction | |
| ***** | Multiplication | |
| **/** | Division | |
| **%** | Modulus | Returns the remainder after division |
| **++** | Increment | Adds 1 to the value |
| **--** | Decrement | Subtracts 1 from the value |
| **=** | Assignment | Evaluates the right side, then stores it in the left side |
| **+=** | Addition Assignment | Evaluates the right side, then adds it to the left side and stores the result there |
| **-=** | Subtraction Assignment | Evaluates the right side, then subtracts it from the left side and stores the result there |

| Operator | Name | Notes |
|---|---|---|
| **==** | Equality Comparison | Checks if one value is equal to another |
| **!=** | Inequality Comparison | Checks if one value is **not** equal to another |
| **<, <=** | Less than, Less than or equal to | Checks if left value is less (numerically or alphabetically) than the right value |
| **>, >=** | Greater than, Greater than or equal to | Checks if left value is greater (numerically or alphabetically) than the right value |
| **&&** | And | Combines two conditions and checks if both are true |
| **\|\| (pipes)** | Or | Combines two conditions and checks if at least one is true |

## Comments

Using comments is very helpful when writing JavaScript code. Leave comments to describe how something works, the purpose of the code, or to leave notes for editing something later. They are especially helpful when copying some code and pasting it in a different Flow Step. Comments are ignored when the code is run.

### Example:

```
// This entire line is a comment, but it does not continue to the next line

/*
This is also a comment
And this comment keeps going
For multiple lines
Until
*/
```

## If, if else, and else

Just like Excel, JavaScript provides tools for performing calculations based on a condition. One of the most common conditional syntaxes is the if/then statement.

```
if(condition){
//expression
}
```

For instance, display the word “ACTIVE” in a new Field if a status Field shows “a”.

### Example:

```
if(status == 'a') {
  'ACTIVE';
```

```
}
```

The `if` is always followed by a condition in parentheses. The condition typically uses one of the comparison operators, but it could be any piece of code that returns true or false.

## Example:

```
var myBoolean = true;
if(myBoolean) {
  'This will be displayed.';
}
```

If the condition in parentheses is true (the "`if`"), then the expression that follows (the "`then`") is evaluated. If the condition is not true, the expression is not evaluated. Usually, the "`then`" code block is enclosed in curly braces {}.

A common use case is to check if a variable has any value at all. The simplest way to check for blanks, null values, or just 0 is with a truthy condition.

## Example:

```
if(columnAlias) {
'Column Alias has a value';
}
if(!columnAlias) {
  'Column Alias is empty, zero, or null.'
}
```

In the past three examples, nothing is displayed when the condition is false. To determine what happens when the condition is false, include an "`else`" statement.

## Example:

```
if(status == 'a') {
  'ACTIVE';
} else if(status == 's') {
  'SUSPENDED';
}
else {
  'INACTIVE';
}
```

In JavaScript, the else statement essentially means 'if the immediately preceding if-elses are false, then do something'. Switch/case statements can be used as an alternative to if-statements in some cases. This training will not cover switch/case statements.

## Styling with CSS

When viewing data in Informer, it is displayed via a webpage. This means that any data with HTML code will be rendered as HTML, including Calculated Fields. A common use of this styling output is to use a **span** tag. See the example below.

### Example:

```
// CSS properties, formatted as Property1:value1;property2:value2 without any spaces
var myStyle = 'font-weight:bold;color:white;background-color:red';
var output = "<span style='" + myStyle + "'>Something highlighted!</span>";
output;
```

## Using Data Type Functions

JavaScript has a feature called functions that grants quick access to commonly used code. Functions are blocks of code that can be called or told to run. Functions might have optional *parameters* or inputs. Most data types have functions available to them, and these can be found online (https://www.w3schools.com/jsref/ is a great site for this). For example, any String data type has the function *toUpperCase():*

### Example:

```
var myText = "Some Text Here";
var output = myText.toUpperCase();
output; // This will display SOME TEXT HERE
```

Date variables have some particularly useful functions:

```
var myDate = someDateField;
myDate.getDate();
myDate.getDay();
myDate.getMonth();
myDate.getFullYear();
```

## Using Libraries

Informer Calculated Fields have access to two libraries that provide even more functions: Lodash (or _ when used in code) and Moment. Lodash handles very complicated functions and is covered in the advanced course. Moment takes a Date and gives it extra functions.

```
moment().now();
moment(d).startOf('day');
moment(d).add(5, 'days');
moment(d).format('YYYY MM DD ddd hh:mm:ss');
moment(someStart).isSameOrAfter(someEnd);
```

## Ordering Flow Steps around Merge Duplicates and Normalize

Flow Steps happen in the order that they appear on the left side of the Edit Screen. Sometimes the order of Flow Steps matters.

For instance, the Merge Duplicates Flow Step turns data into an array. Its opposite, the Normalize Flow Step, breaks an array of data into separate rows. Any Calculated Fields after a Merge Duplicates Flow Step must treat certain fields as an array. Any Calculated Fields after a Normalize Flow Step must treat certain fields as a non-array, or flat data. Typically, handling arrays is more complicated. Try to order the Flow Steps so that no Calculated Fields must handle arrays when possible.

An array (or a multivalued Field) is a collection of other values in a specific order. Each item has a position called an index, and you can refer to specific values by their indices. Indices always start at 0. Consider an array called firstNames:

| Index | Value |
|---|---|
| **0** | John |
| **1** | Kate |
| **2** | Mary |
| **3** | Sam |

Use firstNames to refer to the whole array, or firstNames[0] to refer to just John. The position in the square brackets can be a variable, as well.

### Example:

```
var firstNames = ['John', 'Kate', 'Mary', 'Sue'];
firstNames[0] + ' and ' + firstNames[1];
```

### Example:

```
var firstNames = ['John', 'Kate', 'Mary', 'Sue'];
var position = 3;
firstNames[position];
```

Use a variable and a for loop to iterate through all the values. A for loop creates a variable that starts with the value 0 and increments through each index. For example, to get a total of all the numbers in an array, see below.

### Example:

```
var numbers = [4, 10, 35, 17];
var total = 0;
for( var i in numbers ) {
  total = total + numbers[i]; //total += numbers[i]; -this would do the same
thing
```

```
}
total;
```

The above example uses a code block, just like with the if examples. At the start of the for loop, the variable i is created and given the value 0. At the end of the code block, i goes up by 1 and the loop checks if the array still has values remaining. If it does, the code block is run again. If the arrays are associated, the associated values share the same positions. In this case, use the same index variable to reference positions in those arrays.

### Example:

```
var firstNames = ['John', 'Kate', 'Mary', 'Sue'];
var lastNames = ['Smith', 'Smith', 'Anderson', 'Locke'];
var output = new Array(); // you don't have anything to put in it yet, so
just create a new array here
for( var i in firstNames) {
  output[i] = firstNames[i] + " " + lastNames[i];
}
output;
```

Using the same variable as the position for each of these eliminates mixing up the names and they will stay in the appropriate order.

## Reusing Code

It may be useful to copy and paste code from another Query. Keep in mind that even if all the Fields are the same, they may have different aliases from Query to Query. When copying code from somewhere else, such as a website or PDF (such as this guide), be sure to double check the punctuation. Certain characters such as quotes (" ") may need to be retyped.

To avoid needing to replace each location that a Field is referenced, shorthand the Field aliases as new variables.

### Example:

```
var field1 = insertFieldAliasHere;
var field2 = instertOtherAliasHere;
// do calculations with field1 and field2
```

## Using Saved Functions

Commonly repeated code can be saved into a Saved Function by an administrator. To use a Saved Function, click the Functions button to the right of the expression window and follow the prompts. It may require parameters, or inputs, to work correctly. Parameters are typically other Fields from the Query.

## Troubleshooting

If a Calculated Field fails, it may show a red exclamation point (!). Click the exclamation point to see what error occurred.

Other Calculated Field errors may cause the Field to simply show all blank values. Try printing a basic value at the very end, such as "output", to see if that displays correctly. This will indicate whether the Calculated Field is coming up with a blank value or if it fails to finish running.

Another method of testing Calculated Fields is replacing Field Aliases with test values (usually a literal number or a quoted text value). Use this to check the calculation's output for data that may not appear in your Query frequently.

## Finding More Information

There are plenty of resources for JavaScript references online. Keep in mind that most of them will be geared towards JavaScript for HTML and may include code to overwrite HTML elements. Calculated Fields in Informer cannot overwrite HTML.

You can find helpful documentation for Moment and Lodash at https://momentjs.com/docs/#/get-set/ and https://lodash.com/docs/3.10.1 respectively. Our Help center also includes helpful information on Calculated Fields.