thinkproject FORMWORKS

Formworks JavaScript Scripting Guide

Contents

Introduction to Formworks Scripting 9
Formworks scripting introduction 9
Common uses of scripting 9
Validation 9
Calculated fields 9
Date Arithmetic 9
JavaScript 9
Differences of JavaScript on iOS Devices/Web Forms 10
Testing and debugging your JavaScript scripts 10
Design Mode 10
Edit Mode 10
The Formworks Scripting Object Model 12
Similarities to other development environments 12
Elements, Events, Properties and Methods 12
Warning 13
Javascript Guidelines – The Basics 14
Case sensitivity 14
Syntax 14
Comment Operators (// and /* ... */) 14
Concatenation Operator (+) 15
Equals (=) and (==) 15
Terminating instructions (;) 15
Assigning and Reading values to Elements 16
User Defined Properties – App Only 16
Code structures 17
Switch / case structures 18
Relational Operators 18
"&&" and "||" comparison 18
Less than and Greater than comparisons 19

Negation (! And !=) 20
JavaScript – Advanced use 21
String Functions 21
Confirming that a value entered is a number 21
Converting between Strings and Numbers 21
Validating a range or individual characters in a Field 22
Replacing characters in a field 22
Character substitution within element names 22
Looping structures in JavaScript 24
Handling Dates 25
Handling Time 25
JavaScript – Global Functions 26
Introduction 26
Returning a value 26
Formworks Date Validation Properties and Functions – App Only 27
Date Properties 27
Date Functions 27
Time Properties 28
Time Functions 28
Setting Date Properties 28
Reading Date Properties 28
Common date formats 28
Formworks Time Validation Properties 28
Device User's Name, Email Address and Geo-Location 29
Device User's Name 29
Device User's Email Address 29
Geo-Location Data 29
Creating a Unique Reference 30
Renaming PDF Export Files 30
Formworks Alert Boxes, Notification Lists 31
Interactive alert Boxes 31
Notification Lists – App only 31
Scripting Specific Elements 32

Help Element....32
Single Selection (List Box)....32
Table Element....33
Filter Table Function – App Only....33
General Table Functions....33
Accessing Table Cells within Loops....35
Setting Row Visibility within Loops....35
Methods of Accessing Individual Table Cells....35
Forms....37
Automatically opening a new blank form – App only....37
Automatically closing and deleting a form – App only....37
Dynamically populating a Single Selection Dropdown List box....38
Dynamically populating a Single Selection List based on values in another....38
Elements....40
Containers....40
Fields....40
Other Elements....40
Events....42
Properties....43
Methods....43
The Script Template window....44
Introduction....44
OnValidate Event - Checking for valid input....44
Using the && "and" keyword and Alias names – validating multiple values....44
Checking for empty elements/fields....46
Device Submit Window....46
OnValueChange event....47
Intellisense....48
Calculated Fields....49
Introduction....49
Global Variables – App Only....51
Introduction....51
Custom Properties – App Only....52

Introduction 52
Features of Custom properties 52
Comparison chart 52
Date and Time Functions....54
Getting Current Date and Time 54
Webforms Only: 54
Calculating the difference in minutes between two times....55
Table Elements Scripting 56
Referencing Table cells in script 56
Looping through Table cells 56
Databases....57
Introduction 57
Quotations and calculation 57
Workflow and Project management 57
Maintaining your databases 58
Creating and Updating databases 58
Making a CSV data file 58
Uploading the CSV data file 58
Using databases on your form templates....59
Populating a basic list box 59
Populating a list box with different display and return values 60
The SQLite Select Statement 62
Querying using Wild Cards....62
Querying Multiple Columns 62
Combining And, Or and Where statements 63
Joining Databases....63
Types of Joins....63
Operators....63
Populating cascading list boxes 65
Local Databases on the device 65
Properties - General....66
valid Type: Boolean 66
warn Type: String 66

enabled Type: Boolean 66
visible Type: Boolean 66
title Type: String 67
prop 67
isIphone() Type: Boolean 67
Events - General 69
OnBlur 69
OnEnable 69
OnDisable 69
OnFocus 69
OnHide 69
OnShow 69
OnTap – button element specific 69
OnValidate 69
OnValueChange 70
Form Specific Events – in order of occurrence 71
OnStart 71
OnOpen 71
OnSave 71
OnClose 71
OnSubmitAndConfirm 71
Globals – App Only 71
Form Specific Methods and Properties 72
Audio Functions – App Only 72
Miscellaneous Functions 72
gotoPage("AliasName/PageName" / Page Number) 72
openURLPage(strVar) – App Only 72
Enabled() Enable() Disable() 74
Visible() Show() Hide() 74
Appendix I 75
Elements - their properties, events and methods 75
Buttons 75
Properties 75

Events....75
Methods....75
Checkboxes....75
Properties....75
Events....75
Methods....75
Date elements....76
Properties....76
Events....76
Methods....76
Functions....76
Forms....77
Events – in order of occurrence.....77
Methods - General....77
Methods – Audio (App Only)....77
Related functions....77
Groups and Sections....79
Properties....79
Events....79
Methods....79
Images....80
Properties....80
Events....80
Labels....80
Properties....80
Events....80
Line....80
Properties....80
Events....80
Multi-Select....81
Properties....81
Events....81
Methods....81

Pages....82
Properties....82
Events....82
Methods....82
Photos, Signatures and Sketches....83
Properties....83
Events....83
Methods....83
Single Selects....84
Properties....84
Events....84
Methods....84
Filter Table Function – App Only....85
General Table Functions....85
Text Boxes and Paragraph elements....86
Properties....86
Events....86
Methods....86
Time elements....87
Properties....87
Events....87
Methods....87
Appendix II....88
Regular Expression Types....88

Introduction to Formworks Scripting

Formworks scripting introduction

Scripting provides Form Authors with the ability to extend the functionality of their forms. Anyone familiar with scripting languages will be able to create powerful validation scripts and trigger actions driven by data entered in a form. JavaScript is supported in Formworks for Webforms as well as the iOS app. There may be some differences in functions between Webforms and the App, these will be pointed out within the guide.

Common uses of scripting

There are many reasons for using scripting. Detailed below are some of the more common uses.

Validation

Script can be used to define specific fields as being mandatory/required and specify error messages to be displayed to the device user if they fail to complete these fields. Validation can be as simple as enforcing a single key reference field, or much more complex. You can create conditional logic that stipulates that if one field has been completed, another must also be. Or if a certain value has been entered in one field, another field must also be completed. An example would be a salutation field, containing Mr, Mrs, Ms and Other. If 'Other' is selected, a text box could be enabled that the device user must complete.

You could move the 'focus' of input to a specific field, or page, based on what options a user selects from say, a drop down box. If the user selects 'Private residence', the focus could be moved to page 2, with details of private dwelling types. If they select 'Commercial', the focus could shift to page 3, containing commercial properties.

Calculated fields

Using script you can sum the values of text boxes, placing the total in another text box. These calculations could range from adding a few fields together and calculating VAT, to complex formulas. Values can be allocated to non-numeric fields, like checkboxes, and totals stored in variables that whilst invisible to the device user, would appear in the output. An example usage would be health and safety scores, based on checkboxes that represent risk factors. Calculations can be performed both dynamically, as data is entered, and on demand, for example on the click of a button.

Date Arithmetic

Formulas can be applied to dates and times – both those entered by the user in date and time elements, and the system date and time. For example, you could compare a date entered by a user to the system date, to check if the date was within an acceptable range, or subtract one date from another.

JavaScript

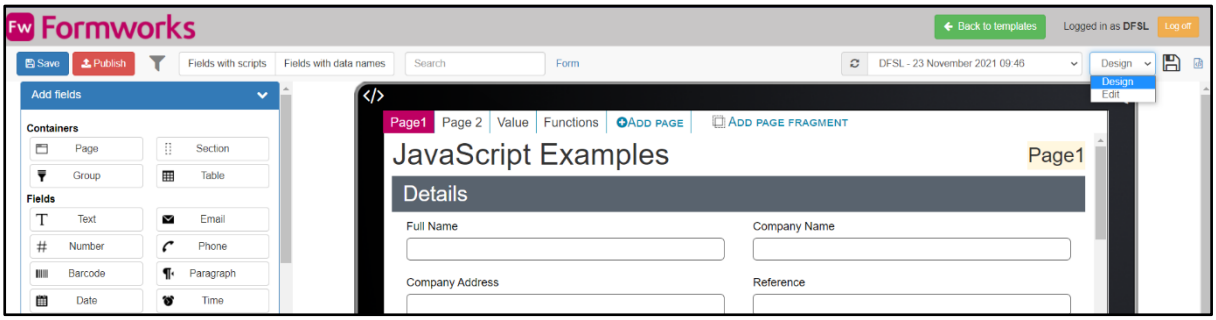
Scripts are written in the popular JavaScript language, though its implementation within Formworks would require some familiarisation. For more information on JavaScript, visit the JavaScript website at https://www.javascript.com/.

Differences of JavaScript on iOS Devices/Web Forms

Typically in JS, elements are referenced using the document.getElementById function. Formworks does not utilise this method, however you can refer to an element using the dollar function, $(). An example of this would be $("Name"), which will reference a field given the Alias 'Name'.
There are slight differences between JavaScript on the iOS App and Webforms, these will be stated throughout the document.

Testing and debugging your JavaScript scripts

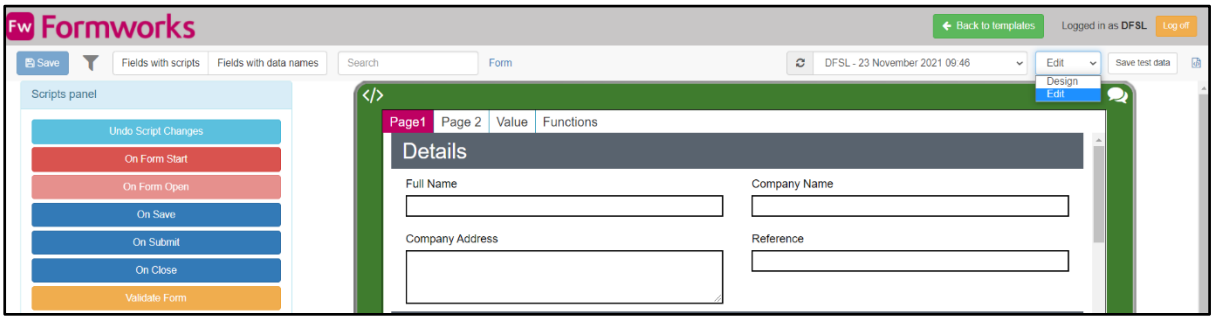
The Formworks Template Designer provides tools to develop and test your JavaScript. These features will be covered in detail in later chapters, but as an introduction, the Template Designer has two modes, Design and Edit.



Design Mode

The design mode is used to design, layout and build your template. Element properties are set in this mode, and scripting entered into the events of the various elements on your template.

Edit Mode



The Edit mode gives you the ability to test the scripting functionality you have built into your template. For example, any script entered into the OnValueChange event of an element will be executed when you enter text into a text box, or select an option from a single select element.

In addition, you can test form event related script. To test mandatory fields, you could enter values into the forms elements and select the Validate Form button on the left of the screen. This triggers all the form elements OnValidate events and highlights those that have failed validation for whatever reason.

The Formworks Scripting Object Model

Similarities to other development environments

Scripting in Formworks employs a common object model, similar to development environments such as Visual Studio. The principles of developing script using Formworks follows many of the basics of object oriented programming. This involves using objects referred to in Formworks as elements. We set element properties and values using method calls when an event occurs.

Elements, Events, Properties and Methods

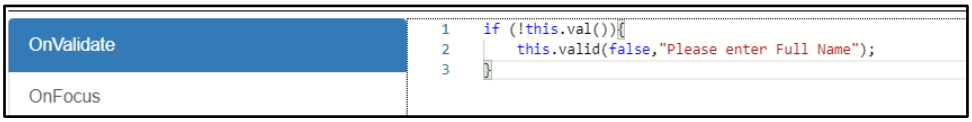
The objects such as a form's pages, text boxes and radio buttons that you insert onto the form in the Form Designer window are called Elements.

Events are actions that can happen to an element, such as when a user touches a text box (OnFocus), or moves from an element that has the focus to a different element, (OnBlur) or when the value within an element is changed, (OnValueChange).

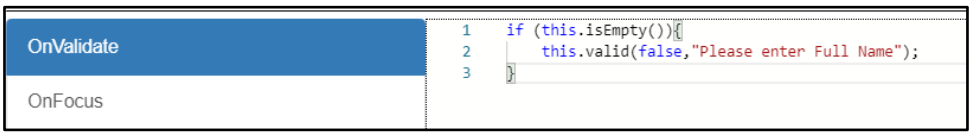
Most of these elements possess properties, such as Message or 'warn', Value and Valid. Usually, the purpose of writing script is to change the value of an element's properties.

You can read and set properties using method calls. For example, the method call this.valid(false, "Please describe category.") would set the Valid property of an element to false, as well as the message to 'Please describe category'.

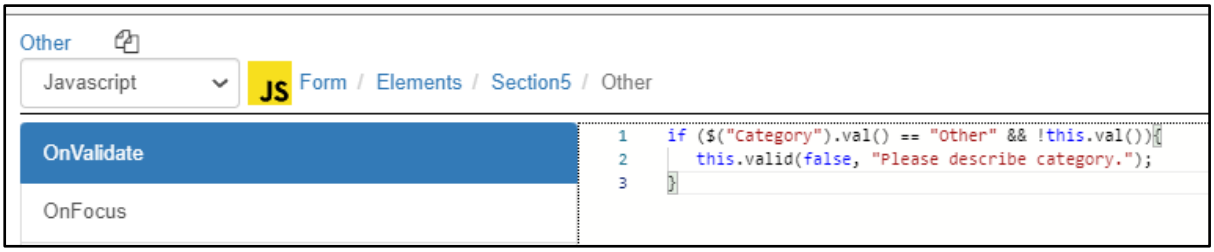
A common example would be making a text element mandatory for input. When the device user tries to submit the form, the OnValidate Event for each element is run in the order that they appear on the Form Designer, checking the valid property for every element. If any valid properties have been set to false, the form will not submit. The method val() returns the text value of the element, whereas the isEmpty() method simply checks if there is a value in a field and can be used for all elements, including Multiselect lists.



Note: isEmpty() can be used on Webforms only.



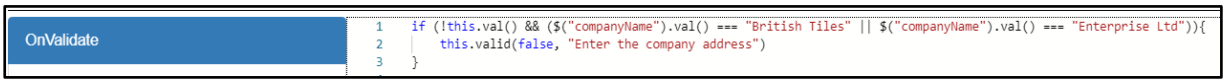
Taking a real world example to demonstrate the use of properties, methods and events, you might wish to make a text box element mandatory for input if another field, say a drop down has a certain value. To do this create a script that uses a method to set the element's Valid property to false if the drop down field has a value of 'Other' and no input has been entered into the text field. Note that you can only set the valid and message properties of an element from within the elements OnValidate event. You can refine this example by using another method to set the element's message property to display a meaningful message to the user.



The element, Elements.Section5.Other can be considered a basic object. When entering code directly against the element's template, the keyword 'this' represents the element's entire name. The term 'val' represents the element's Value property. The Method call, 'this.valid(false, "Please describe category")' sets the element's Valid property to false and the message 'Please describe category', is a meaningful message to display to the device user if they fail to complete the field before they try to submit the form.

The following example is slightly more complex in that it compares to values to decide if an element should be considered mandatory. This is known as conditional logic. The && characters represent "and" in the logic. The element to be evaluated, companyName (an alias), is enclosed in the $() search function. Whilst either two or three "=" characters would work.

The final example adds the || "or" logical operator to the formula. The two parts of the "and" section have been enclosed in braces () to ensure they are evaluated together, though this may not strictly be necessary.



Warning

To provide a message, while leaving the valid property set to true, the warn method call can be used. The syntax for this is:

```
this.warn("This is a warning");
```

Javascript Guidelines – The Basics

Case sensitivity

JavaScript instructions are NOT case sensitive when referring to field names, therefore the below statements are both valid:

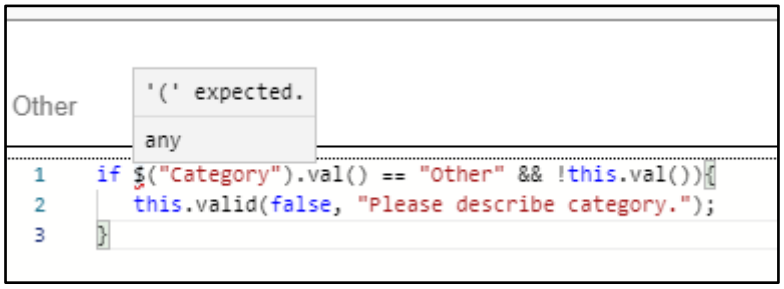
```
$("textField").val("Test value");
```
And
```
$("textfield").val("Test value");
```

However, when comparing one string to another, the case of the value to be compared is relevant. For example, these two statements are not the same, because of the different case used for James.

```
if ($("Account").val() == "James"){
   $("OutputValue").val("James & Co");
}
```
And
```
if ($("Account").val() == "james"){
   $("OutputValue").val("James & Co");
}
```

Syntax

You can create and save JavaScript scripts in the Form Designer, using the Scripting window. It is possible to save incorrect instructions, such as using capital letters on JavaScript keywords or missing the closing "}" instruction on an "if" code structure, but incorrect instructions will be highlighted with a red underline. Hovering over the underline will display a reason for the error. In the below example a '(' is expected to enclose the condition for the if statement.



Comment Operators (// and /* … */)

Use two forward slash characters to create a single line comment in JavaScript (//). To comment multiple lines of code use /* text */

```
1  //Single line comment - Description of script
2  $("outputvalue").val("Test value");
3
4  /*
5  Multiple line comments. I don't want any of this
6  description included in my code.
7  */
```

Variable name restrictions

In JavaScript, variables must not contain the hyphen character (-). This same restriction applies to element names and aliases.

Concatenation Operator (+)

To concatenate string values, use the plus sign (+). For example:

```
var x = "Test" + " Case"; //The variable x now contains "Test Case".
```

The below example demonstrates the concat() function, the outputValue would be 'Now is the time for all good men':

```
var a = "Now is the time ";
var b = "for all good men";
$("outputValue").val(a.concat(b));
```

To place a carriage return or line feed between variables in script, you would use \r or \r\n. For example;

```
var x = "Test \r" + " Case";
```

However, if you wish to place a carriage return or line feed between characters in a paragraph field, when prefilling a form, you would use the \n character.

Equals (=) and (==)

When you are assigning a value to a variable, you use a single equals character, for example:

```
stringVariable = "Test";
```

When you are testing that one value equals another, you use two, or three equals characters, for example:

```
if (variable1 == variable2){... etc.  or if (variable1 === variable2){... etc.
```

Terminating instructions (;)

Each instruction must be terminated with the semi-colon character (;).

Assigning and Reading values to Elements

The most common actions in scripting are allocating values to elements, reading values and setting element properties. For example;

- $("textBox").val("Test"); //setting a val property
- variable = $("textBox").val(); //reading an element's value
- $("textBox").visible(); //reading an element's visible property
- Web only: $("Table").row(1).visible = true/false; //some elements such as Table rows require the visible property to be set with an '=' sign

Most elements have a value property, though not all. For example, neither line nor label elements can contain a 'val'.
You read the value of an element, say a text box, by accessing its val property:

```
variable = $("textBox").val()
```

However, there are a number of different ways to refer to an element within script:

- Fully Qualified Name:
  - variable = $("Page1.Section1.Group1.textBox").val();
- Alias property:
  - variable = $("AliasName").val();
- Dataname - App only:
  - Variable = $("DataName").val();

In addition to the above, as a two-step process, you can create an object reference to an element, and then refer to the reference object's value property:

```
var tempField = $("Page1.Section1.Value/Alias");
$("TargetField").val(tempField.val());
```

User Defined Properties – App Only

In addition to the standard element properties, that are pre-defined within the application, you can create your own element properties, allocate a value to them, and read these values to use within your script. This can be done by utilising the prop function. There are a limitless number of uses for User Defined Properties, but obvious examples are workflow management and simply to hold multiple readable values against an element, for example:

Creating a new User Defined Property:

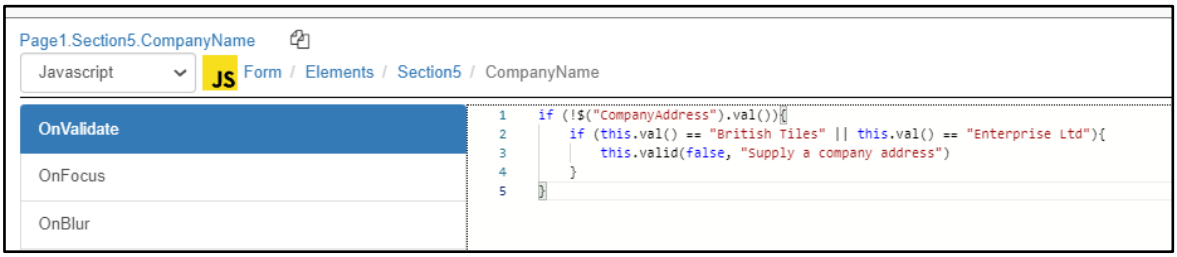
```
$("textBoxWorkflow").prop("Notification", "Initial Client Contact");

$("textBoxReference").prop("ClientRef", 12345);
```

Reading the value of a User Defined Property:

```
$("textBoxOutput").val($("textBoxWorkflow").prop("Notification"));
$("textBoxOutput").val($("textBoxWorkflow").prop("ClientRef").toString());
```

Code structures



Basic structures such as, "if (), else if (), else" are supported. These must include both the "{" and "}" instructions.

```
if (first check) {
        first instruction;
}else if (second check) {
        second instruction;
}else {
        final instruction;
}
```

Where more than one option is possible, you can use an "if, else if, else" structure:

```
1   if ($("Account").val() == "James"){
2       $("OutputValue").val("James & Co");
3   }else if ($("Account").val() == "Martin"){
4       $("OutputValue").val("Martin & Sons");
5   }else{
6       $("OutputValue").val("Another Client");
7   }
8
```

As above, the clause to be evaluated must be enclosed in braces () and compared using two equals characters, as in line 1.

The else if clause on line 3 will execute if the first comparison on line 1 is false.
If the comparisons on both line 1 and line 3 are false, the 'else' clause on line 5 will be used.
The curly braces are used to group script together.

A variant of the "if" statement can be expressed on a single line. This variation will be familiar to both JavaScript and C# users:

```
1   $("outputValue").val($("Account").val() == "James"? "James & Co" : "Another Client");
```

You would interpret this line as:
If the value of the Account text box equals "James", enter "James & Co" in the outputValue text element, otherwise enter "Another Client".

Switch / case structures

```
1   switch ($("Account").val()){
2       case "James":
3       case "James & Co":
4       case "James & Sons":
5           $("OutputValue").val("James & Co");
6           break;
7       case "Martin":
8           $("OutputValue").val("Martin & Sons");
9           break;
10      default:
11          $("OutputValue").val("Another Client");
12          break;
13  }
```

Switch structures are an alternative to if, then, else structures. In this example, if the value in the Account text box = James or James & Co or James & Sons, the first case clause will be valid and execute. This is represented by lines 2,3 and 4. Where the same script is to be executed, you do not need to repeat it, simply place the case statements together as shown.

It is important to place a colon character (:) after each case option.

After the script to be run for each option, a break statement, like the ones on lines 6, 9 and 12 are required to prevent "fall through", where subsequent code can be run in error.

The default instruction on line 10 indicates that if no other condition is considered true, then the script starting at line 11 to the break statement at line 12, should be run.

Relational Operators

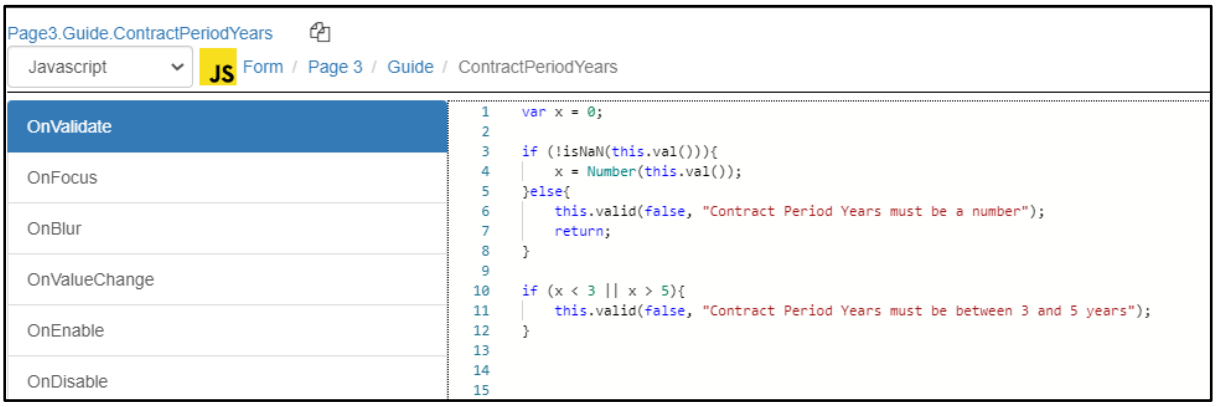
These include:

Operator	Meaning
&&	and
||	or
< >	less than, greater than
<= >=	less than or equal to, greater than or equal to
!	Not
!=	Not equal to

"&&" and "||" comparison

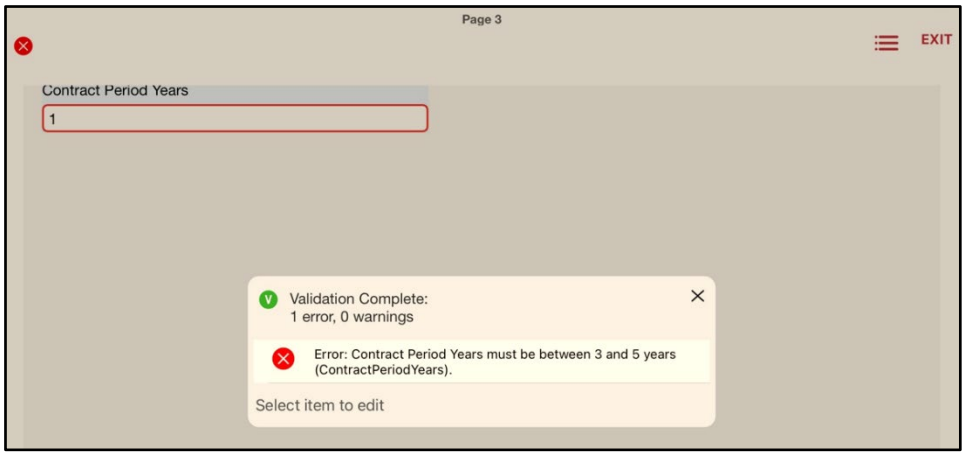
You can also compare values using && for 'and' and || for 'or', as in the above example, under Code structures.

Less than and Greater than comparisons



To ensure that a numeric value is within a specified range, use the Less than (<), Greater than (>) and Equals operators. Digits should not be enclosed in quotes.

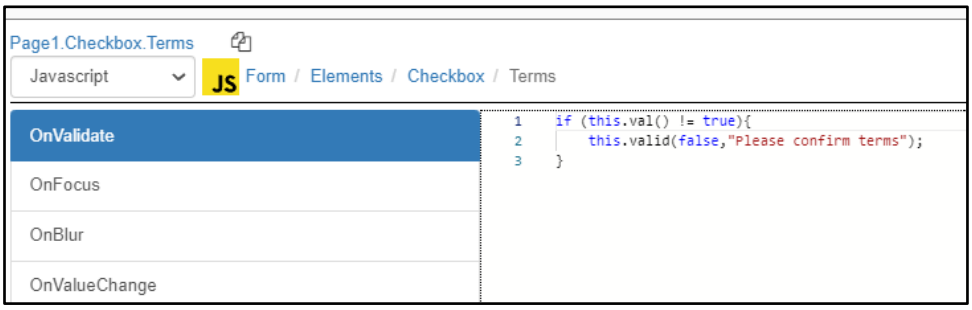
If the value 1 is entered in the ContractPeriodYears text element, a warning message will be displayed in the Form Submission window, when the device user tries to submit the form.



If the ContractPeriodYears text element is left empty, the message "Contract Period Years must be a number" will be displayed.

Negation (! And !=)

The characters != can be used to test if a value does not equal another value. See figure below:



The character ! Can preface a field or variable to check if a value exists. The below example will return true if the Account field does not have a value:

```
if (!$("Account").val()){..
```

JavaScript – Advanced use

String Functions

Confirming that a value entered is a number

Frequently you will need to confirm that a user has entered a number in a text element. The simplest way to achieve this is the Is Not a Number function. The logic here is, If NOT is Not a Number. In other words, it IS a number.

```
if (!isNaN($("AnyNumber").val())){
   $("outputValue").val("is a number");
}else{
   $("outputValue").val("Not a number")
}
```

This function simply checks that your test value doesn't contain any alphabetic characters. It will also check if there are spaces INSIDE the test value. But it will return true if the value starts with either a space of 0.

Converting between Strings and Numbers

There are several functions that can perform the conversion between numbers and strings. The easiest to remember is the Number() function.

This example first checks that the value in the AnyNumber text element is a number, then adds it to the value in the AnotherNumber text element, before placing the sum in the outputValue element.

```
if (!isNaN($("AnyNumber").val()))
   $("outputValue").val((Number($("AnyNumber").val()) + Number($("AnotherNumber").val())).toString());
```

Note: when assigning integer values to a text field on the app, the .toString() function must be used, as in the above example.

Another way to do this would be to use the valueAsNumber() function, see below:

```
var int = $("AnyNumber").valueAsNumber()
```

Note: valueAsNumber() will work on Webforms only.

Concatenating a literal string to a numeric value using the concatenate instruction:

```
var1 = "35";
var1 = var1 + 1;
```

Following the concatenation instruction "+" the variable var1 now contains the string value "351".

You can perform arithmetic functions on string variables that contain numbers, with conversion:

```
var1 = Number(var1) + 1;
```

The variable var1 now contains 352.

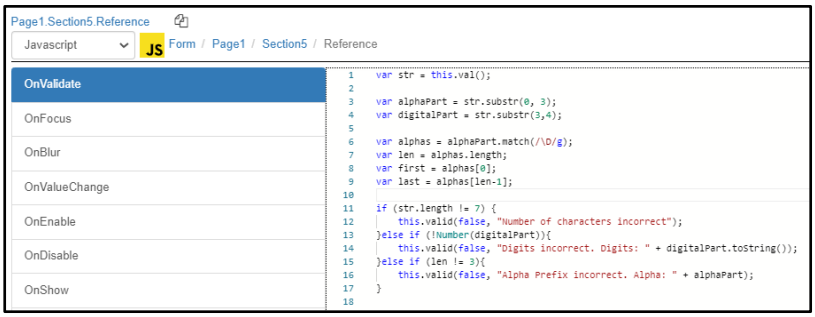
Finding the length of an element's value

You can determine the length of a field's value, for example, to confirm that the user has entered the correct number of characters, using the length function. The format for this function is:

```
var vLength = stringVariable.length;
```

Validating a range or individual characters in a Field

If you need to know that part of a field is alpha and part is numeric, you can use the match() function. For clarity, I have also used the substr() and length function to confirm that the Reference field is seven characters long and contains three alpha characters, followed by four digits.



In this example, assume the Reference field contains the value "ABC1234". The two components, "ABC" and "1234" have first been stored in two local variables using the substr() function. Indexing in JavaScript is based on the first character equalling 0.

The first line instructs JavaScript to, convert the value of the field to a variable named 'str', to make referencing the value easier. Line 3 reads the characters in the Reference field, from position 0, for 3 characters long, and stores them in the local alphaPart variable. The 4th line of the function stores the characters from position 3, for 4 characters long in the digitalPart variable.

The str.match() function, matches all the non numerical characters of the alphaPart variable and stores it in a variable called alphas. The next lines stores the first occurrence of a string, and the last. The search string could be a literal value, such as "Ref", or as in this instance, any alpha characters, both upper and lower case. This is performed using the "/\D/g" section of the function. "\D" is a character class that indicates any non digit characters, and the "g" character indicates that we are searching for more than one character. Appendix II at the end of this guide contains all the supported character classes. Using these classes you can search for either alphanumeric characters, digits, or a range of combinations and other characters.

To confirm that the digital part of the Reference field is correct, you can simply try to convert it to a number, and see if this fails. In the example, using:

```
}else if (!Number(digitalPart)){
```

Replacing characters in a field

```
$("elementName").val().replace("original string", "replacement string");

var changed = $("elementName").val().replace("original string", "replacement string");
```

Character substitution within element names

There are three ways to refer to an element within script:

Its fully qualified name, for example:

```
$("Page1.Item1.Q").val($("anotherElement").val());
```

By an alias allocated to the element, in addition to its name, i.e.,

```
$("Q1").val($("anotherElement").val());
```

Or by replacing characters within the element name with the values of variables. In this example, counter = 1, therefore the result of this substitution is the same as example 1, `$("Page1.Item1.Q").val()`.

```
var counter = 1;
$("Page1.Item" + counter + ".Q").val($("anotherElement").val());
```

Use standard JavaScript string concatenation to place the value of the variable within the 'string' of the element name that you are referring to.

Looping structures in JavaScript

The use of looping structures follows on logically from the previous section, Character Substitution within Element Names. By substituting parts of element names with variables, and placing them within a looping structure, it is possible to perform many actions with very few lines of code.

The basic structure of a for loop in JavaScript is:

```
for (var index = startNumber; index <= endNumber; stepsIncrement/Decrement){
        actions;
}
```

In the next example, the character i will first contain the number 1, then for every loop iteration, it will increment by 1, until it reaches 10. Then the loop will terminate. The steps parameter is required, and can be positive (++ etc) or negative (-- etc).

Here we are iterating through the actions inside the loop 10 times. The character i will always contain the number of the iteration. Then we use simple string concatenation, so that we refer to 10 different text box elements, all named Quantity, but in ten different sections. The sections are named Item1, Item2, Item3 etc. In the example, when an item with an empty value is found, the variable "counter" will be set to the value in i, and the break instruction will cause the loop to terminate. The code will continue execution from the first instruction outside of the for loop.

```
for (i = 1; i <= 10; i++){
        if (!$("Page1.Item" + i + ".Quantity").val()){
                counter = i;
                break;
        }
}
```

The alternative to the above code, would be something like:

```
if (!$("Page1.Item1.Quantity").val()) {
        some action
}
if (!$("Page1.item2.Quantity").val()) {
        some action
}
if (!$("Page1.item3.Quantity").val()) {
        some action
}
```

etc.. Repeated 10 times.

Handling Dates

The JavaScript today function returns a string of today's date. For example:

App:

```
$("DateField").val(today());
Or
var date = today();
```

Web:

```
$("DateField").val(today);
or
var date = today;
```

When placed in a variable, the today function will output in the following format 'Tue Nov 02 2021 12:08:01 GMT+0000 (Greenwich Mean Time)', depending on the users location.

Assigning a date field with the today function will input the current date in whichever date format the Date field has been set to ie DD/MM/YYYY

Handling Time

To place the current system time into a time element, use:

App:

```
$("TimeField").val(timeNow());
```

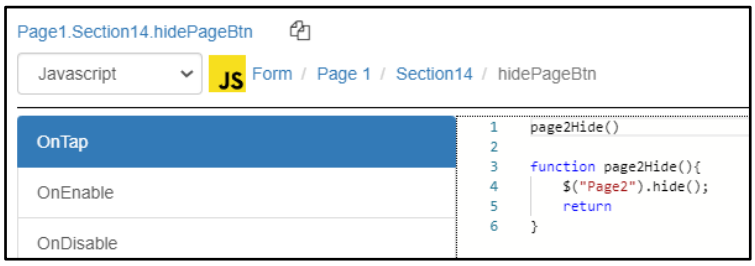
Web:

```
var time = today.getHours() + ":" + today.getMinutes();
```

JavaScript – Global Functions

Introduction

JavaScript and Formworks support Global Functions. On Webforms and the App these will need to be placed in the same event as where the function is being called. Functions can either perform simple actions that require no parameters, or quite complex actions that require one or more parameters being passed to the function.



A simple example of a JavaScript function would be to hide a template page. The script has been placed in a buttons OnTap event and called from the same place.

Even though no parameters are being passed to the function, the braces () are still required after the function name and when it is called.

Returning a value

You can both pass a value to a function and return a value from a function. In this example we are checking if the string value of the 'current' text box is 4 characters long. On line 2, we are calling a Global Function in the templates OnOpen event. The function validString returns true if the string is 4 characters long. Any other length returns false.

You don't need to return a value and instead, as in the first example, simply issue a 'return' statement.

Formworks Date Validation Properties and Functions – App Only

There are eight built in date properties, one date function, two time properties and one time function in Formworks. These are designed to simplify working with dates. The days property returns the number of days since 1st January 1900 (previous dates are returned as negative values).

In addition, it is possible to set both the maximum and minimum acceptable dates for a date field. These values can be set either as a number or using any of the standard date formats below. Most of these properties are both read and write, other than those marked as read only.

Date Properties

- $("DateField").val()
- $("DateField").prop("days");
- $("DateField").prop("minimumDate");
- $("DateField").prop("minimumDateDays");
- $("DateField").prop("minimumDate");
- $("DateField").prop("maximumDateDays");
- $("DateField").prop("seconds"); (read only)
- $("DateField").prop("weekday"); (read only)

Date Functions

- var dob = $("DateOfBirth").age();

Time Properties

- $("TimeField").val()
- var secondsValue = $("TimeField").prop("seconds");

Time Functions

- var differenceInMin = $("endTime").subtractTime($("startTime").prop("minutes"));

Setting Date Properties

The following are examples of setting the Formworks date properties. The same rules apply to both the minimumDate property and the maximumDate property

```
$("DateField").val("07/08/18")                    Quotes required.
$("DateField").prop("days", "44382");             Quotes required.
$("DateField").prop("minimumDate", "2021/07/14");     Quotes required.
$("DateField").prop("minimumDateDays", "44391");      Quotes required.
$("DateField").prop("minimumDate", "2021/07/14");     Quotes required.
$("DateField").prop("maximumDateDays", "44391");      Quotes required.
```

Reading Date Properties

All properties can be read using the prop() function, using the below format:

```
var nVar = $("DateField").prop("days");
```
nVar could contain say 46201 – a numeric value.

```
var nVar = $("DateField").prop("maximumDate");
```
nVar would again contain a number.

When returning values on the app, like say the days property, you may need to use the .toString() function. Eg. `$("DaysField").val(nVar.toString());`

Common date formats

These include: ddMMyy, dd/MM/yy, ddMMyyyy, dd/MM/yyyy, dd-MM-yyyy, yyyy-MM-dd, yyyy/MM/dd.

Formworks Time Validation Properties

Time elements also have a seconds property. This returns the number of seconds since 00:00. For example: `var nVar = $("TimeElement").prop("seconds");`

You can take one time field away from another time field, using the following syntax:

```
var difference = $("endTime").subtractTime($("startTime").val());
$("Difference").val(difference);
```

Normally the first time element, Time1, will be lower than the second, Time2. For example, start and end times for work on site might be 09:30 until 11:30. In which case the function will return 120.

However if the period on site goes over midnight, the end time value might appear to be less than the start time. For example, start at 23:30 and end at 01:30. This will also return 120.

Device User's Name, Email Address and Geo-Location

In addition to basic JavaScript functionality, it is also possible to gather certain information within script, such as the user's name, email address and the current geo-location, if location services are enabled on the iPad or web browser.

Device User's Name

To acquire the device user's name in script, use the userName() function. For example:

```
App: $("userName").val(userName());

Webforms: $("userName").val($.userName());
```

Device User's Email Address

To acquire the email address in script, use the userEmail() function. For example:

```
App: $("userEmail").val(userEmail());

Webforms: $("userEmail").val($.userEmail());
```

Note: The device user's email address is taken from the email address entered when they logged into the device, NOT the email address as contained in the Formworks portal Manage Users & Roles screen.

Geo-Location Data

To acquire the Geo-location of the device in script, use the location() function. For example:

App:

```
$("textField").val(location());
```

Webforms:

```
$.location().then();
 $.location().then(function (pos){
   $("location").val(pos.latitude + ' ,' + pos.longitude)
})
```

This returns a comma separated latitude/longitude pair. This is returned as a single string, example: 51.432100, 0.397287.

Note: If being pulled from a browser (Webforms), the location can be delayed so the above function is used to get the location and load it in a timely fashion.

Creating a Unique Reference

Generating a unique reference is a common requirement. The following example takes the first and second (if available) initials of the user and combines them with the current day, month, and year. The below example can be placed in the OnValidate event of a text field element.

```
var fullDate = $("DateField").val();
if (fullDate){var date = fullDate.replace(/-/g,"");} //To replace a forward slash - .replace(/\//g,"");

//will take first and surname initial only if there are middle names
var name = $("FullName").val();
if (name){
  var names = name.split(" ");
  initials = names[0].substr(0,1);
  if (names.length > 1){
     initials += names[names.length -1].substr(0,1);
  }
}
this.val(initials + date);
```

Renaming PDF Export Files

You can change the name of the PDF files that Formworks exports. To do this, place a Text field named __FileName on your template. This can be set to hidden.

The easiest way to use this function is to place script in the __FileName Text field OnValidate property. You can include literal text and the value from other elements on the template. For example, the following script could be placed in the OnValidate event:

```
this.val("Job Form " + $("JobNumber").val() + " " + $("VisitDate").val());
```

In this example, the literal text "Job Form " would be included, then the value from the JobNumber element, followed by a space and then the VisitDate value.

Formworks Alert Boxes, Notification Lists

Interactive alert Boxes

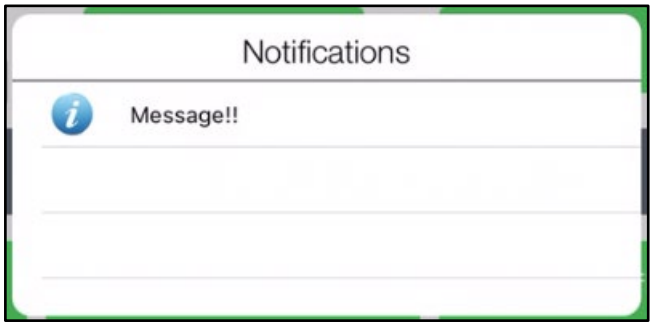
You can generate an alert box to immediately warn a device user that they may have entered an incorrect value. For example, when the device user leaves a field (OnBlur event), an alert box with the heading "Warning Message" and the message content, "Are you certain that the date is correct?" could display, as in the example below. You could include scripting that tests the values of fields, and displays alert messages if appropriate. The alert box has an OK button to close it.

```
displayAlert("Warning Message", "This is an alert");
alert("This is an alert");
```

Note: this option is case sensitive. If you spell alert with a capital 'D' or 'A', it will not work.

Notification Lists – App only

An alternative to Alert boxes is the Notify instruction set. These instructions construct a list of notices that can be displayed to the user.




```
notify(message, autoShowNotificationList,
showIcon)

clearAllNotifications()

showNotificationList()

hideNotificationList()
```

These methods allow messages to be added to, and managed within, the notification area (accessed by tapping on the "i" icon in the top bar). Notifications are displayed with the most recent one at the top of the list. If the optional parameter autoShowNotificationList is included and set to "1", the notification list will be shown. Two examples of the notify instruction would be:

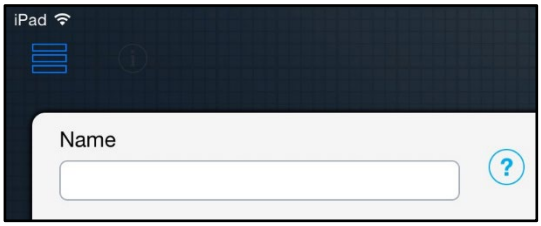
```
notify("Ordinary Message",1, 1);
        // Places text Ordinary Message into the notify list and displays list.

notify("Ordinary Message",0, 1);      // Places text into list, without displaying.
```

There is no limit to the number of messages that can be added to the notification area. All notifications are removed when a form is closed or submitted.

Scripting Specific Elements

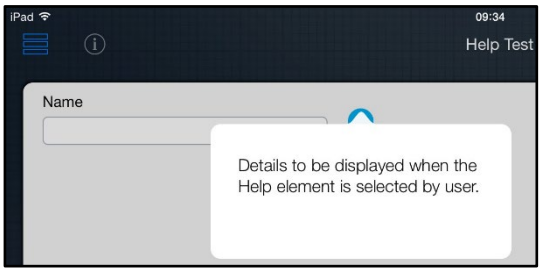
Help Element



The Help element allows you to place a help icon on a template. The help message can be entered into the help element using the Template Designer, or it can be updated dynamically using the element's value property, via script. For example:

```
$("helpElement").val("New help content");
```

The benefit of using script, is that the help text can be tailored to reflect choices that have been made by the device user.



In addition, you can change the visibility of the help (?) icon, so that it only displays where relevant.

But probably the most powerful feature of the help element, is that you can include hypertext links in the text. When selected by the device user, the hypertext links will load a web page, and display it in the default browser. Using this method, you could even display a PDF document in the default reader. When you combine the ability to change the help text dynamically, with the ability to place hypertext links within the text, you can achieve quite powerful functionality with this element.

Single Selection (List Box)

At its simplest, you can populate a Single Selection list box in script using the addOption() method. For example:

```
$("ListBox").addOption("Option 1");
```

To include a different return value to the value being displayed, place the return value first, separated with a comma:

```
$("ListBox").addOption("JBH","John B Henry");
```

Table Element

Filter Table Function – App Only

The Filter Table Function is the most powerful of the table functions and is described here in detail:

```
$("TableName/Alias").filterTable(Column, "title/value", Operator, Value, hidden Rows)
```

Parameter	Description
Column	Number representing the column to be used in the filter operation. Columns are based on 1 as the first column.
Title/Value	String representing the basis of the filter. Its value can be either "title" or "value". If you select "title", the value within the element is ignored and only the title property of the element is evaluated. If you select "value", the value content property of the element is evaluated.
Operator	String representing the operator used to filter the table "==" ">" "<" ">=" "<=" "contains" "doesNotContain"
Value	String/decimal.
Hidden Rows	String choosing whether hidden rows are included in the filter.

General Table Functions

`$("Table").enable(); or $("Table").disable();`	Sets / reads enabled status.
`$("Table").show(); or $("Table").hide();`	Sets / reads visibility status.
`$("Table").repeatRow()`	Will repeat first row in table
`$("Table").removeRows(int)`	Will remove any rows more than the int

The below is available on Webforms only:

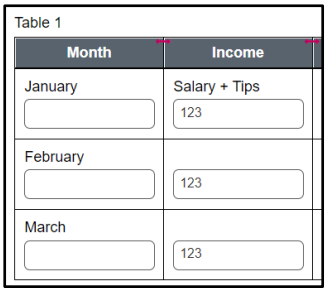
`$("Table").col(int).hide(); and $("Table").col(int).show();`	Hide / Reveal columns.
`$("Table").col(int).visible(true/false);`	Sets column visibility.
`$("Table").row(int).visible = true/false;`	Sets row visibility.

The below is available on the App only:

`$("Table1").hideColumn("string/int") and $("Table1").showColumn("string/int")`	Hide / Reveal columns.
`$("Table1").hideRow("string/int") and $("Table1").showRow("string/int")`	Hide / Reveal rows.
`$("Table1").enableColumn("string/int") and $("Table1").disableColumn("string/int")`	Enable / Disable columns.
`$("Table1").enableRow("string/int") and $("Table1").disableRow("string/int")`	Enable / Disable rows.
`$("Table1").hideColumnHeaders(); and $("Table1").showColumnHeaders();`	Hides / Reveals Headers.
`$("Table1").hideBorders(); and $("Table1").showBorders();`	Hides / Reveals grid lines.
`var rows = $("Table1").rowCount();`	Returns the table row count

Note: when returning the rowCount value to a text field, the toString() function must be used. Eg. `$("textField").val(rows.toString());`

Accessing Table Cells within Loops



Individual cells within a table can take their value from an element within the cell. In this example, the table contains two columns and three rows. Column two contains number elements that are to be summed.

In the same way as you would loop through any set of text boxes that have similar names, you can loop through the cells of the table, extracting the values from each cell. The name of the text boxes is relevant and they will therefore need to be included to work in JavaScript.

```
var x = 0;

for (i = 1; i <= 3; i++){
    if (Number($("Page1.Section1.Table1.cell_2_" + i + ".Text1").val())){
        x = x + Number($("Page1.Section1.Table1.cell_2_" + i + ".Text1").val());
    }
}

$("OutputField").val(x.toString());

```

```
var rows = $("Table1").rowCount();

for (i = 1; i <= rows; i++){
```

If you don't wish to hard code the number of rows into the for loop, you could substitute the number 3 in the example with the $("Table1").rowCount() function.

Note: this is only available on the App.

Setting Row Visibility within Loops

To be able to hide rows, say on the OnOpen or submission is a common requirement. The following script demonstrates how to do this:

```
for (i = 1; i <= 10; i++) {

$("Page1.Section1.Table1").row(i).visible = $("Page1. Section1.Table1.cell_1_" + i + ".Date1").val(); //This line will work on Webforms only

//The below can be used on the App only
  var dVar = $("Page2.Section2.testTable.cell_1_" + i + ".Date1");
  if (dVar.val()){
    $("Page1.Section1.Table1").showRow(i.toString());
  }else{
    $("Page1.Section1.Table1").hideRow(i.toString());
  }
}
```

When run, this script will keep rows 1 to 10 visible if the Date1 field in the first column of that row has a value, otherwise the row will be hidden. The Table's Alias can also be used in place of the full path name.

Methods of Accessing Individual Table Cells

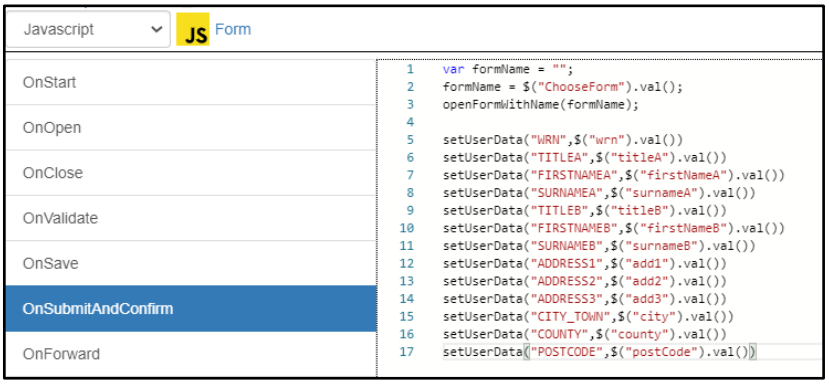
You can refer to the value of the first element in a table cell, simply by referring to the cell itself. As you can see from the above example, the fully qualified name of say, the first cell in the table would be:

```
$("Page1.Section1.Table1.cell_1_1.Text1").val();
```

Forms

Automatically opening a new blank form – App only

You can request that the device automatically loads a new blank form template, when the user submits the current form. This may be useful for example, where a continuation, (child) form will always follow a parent form, or when the device user will always be working with the same type of form. To do this, use the openFormWithName() function. For example:



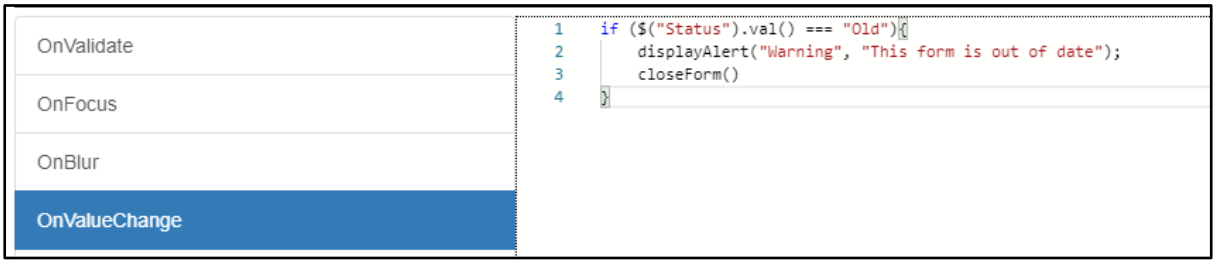

```
openFormWithName("Project Management Schedule");
```

Place this statement in the forms OnSubmitAndConfirm event, and when the device user submits the current form, the screen will clear, and a new blank form of the type specified will load.

A common scenario would be to place a list of 'child' or related forms in a drop down list box within the parent form. When a child form is selected, you could place values that you wish passed to the child form, within global variables. Script could be placed in the child forms OnOpen event, to read the global variables, and place their values into reference fields for example, as shown in the screen capture.

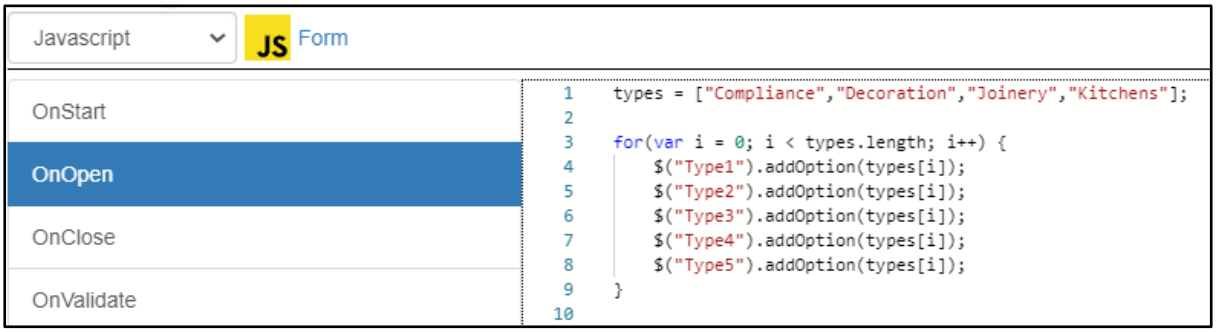
Automatically closing and deleting a form – App only

You can place code in a forms events, for example the OnValueChange event, that can test the value of a field and based on the result, force a form to close, saving any changes. The script in the following screen capture tests the value of the Status field, and if this contains the string "Old", a message box is displayed to the user. When the user selects the "OK" button, the form closes, saving it.



Dynamically populating a Single Selection Dropdown List box

Whilst you can populate Single Selection elements, by entering the values against them in the Template Designer, this may not always be convenient. For example, to maintain a large number of single selection lists, each containing exactly the same values, would entail changing every list element, each time there was a change to the list content. An alternative would be to store the contents of the list in a JavaScript table, and dynamically load this table into each of the single selection elements when the form opens. In the example, the same five items are placed into seven separate single selection elements when the form opens. These seven lists can all be maintained from this one location in this fashion.



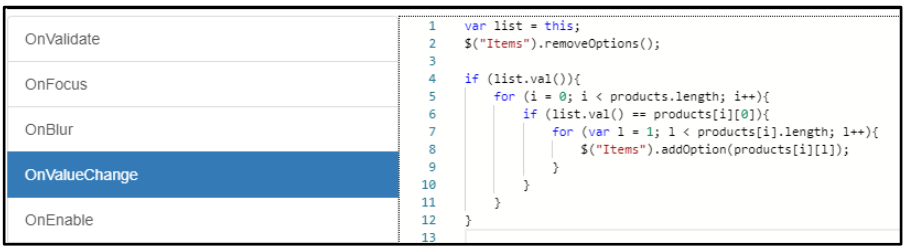
Note: Populating a list box from a table is handled far more efficiently and simpler in Formworks by using Local Databases. These give you the option of changing the values without republishing the template. However, Formworks Databases are subject to licence restrictions.

Dynamically populating a Single Selection List based on values in another

In the form's OnOpen event, create a JavaScript nested table using the following syntax, where the first values, in this instance Clothing, Curtains, Linen and Miscellaneous are the options in the first single selection element.

Note: As mentioned above, tables are handled better in the latest release of Formworks by using the Local Databases option.

In the OnValueChange event of the first single selection element, place the script based on this capture, where Items is the alias for the second single selection element – the one to be populated based on the selections from the first element.



You MUST include the check "if (list.val()){", to confirm that there is a value in the first single selection element. If you omit this check, the form will run correctly in Design/Test mode, but the template will fail to load on the iPad, with a scripting error when the template is subsequently published.

No script is required in the second (Items) single selection element.

Elements

All elements support script and have properties you can change using script. Elements are explained more fully in the Formworks General Guide, and are just mentioned here for completeness. These elements include:

Containers

Page	The main element container. You can have multiple pages.
Section	A container element that horizontally spans a page.
Group	A container to group elements without horizontally spanning the page.
Table	A container element with cells across rows and columns. Can contain other elements

Fields

Text	Can contain alpha, numeric, email address data etc.
Paragraph Text	Contains multiple lines of free format text. You specify the number of lines.
Date	Date element, that can display a calendar to the device user.
Time	Time element. Output is formatted as a time.
Single Selection	Mutually exclusive options, like Windows radio buttons.
Multiple Selection	Multiple selection options, like a Windows check box list.
Checkbox	Single selection, true or false element.
Signature	This element displays an area where someone can sign.
Photo	Provides the option either taking a photo, or using one already captured.
Sketch	Provides a sketching area. The background can be taken from an image file.

Other Elements

Label	Simple descriptive element. The contents can be exported with other data.
Image	Embedded image that can be displayed to the user.
Button	Provides the option of running script on demand.
Line	Provides a simple separator between elements. Can be used for page breaks in PDF exports.
Files	Webforms Only.
Help	Places a help (?) icon on the template. The help text can be entered on the Template designer, or set dynamically in script. The visibility of the help icon can also be changed in script. The help text can contain hypertext links.

The usual method of entering script would be to:

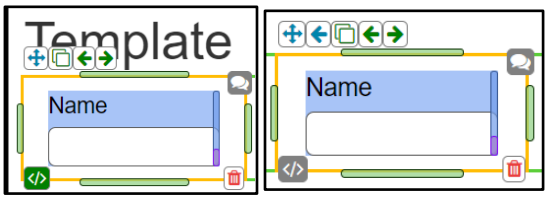
- Select the element on the form
- Select the script '</>' button, ensure the correct scripting language has been chosen.
- Select the event from the left side of the Script Template window

- Enter your script against the event in the Script Template window

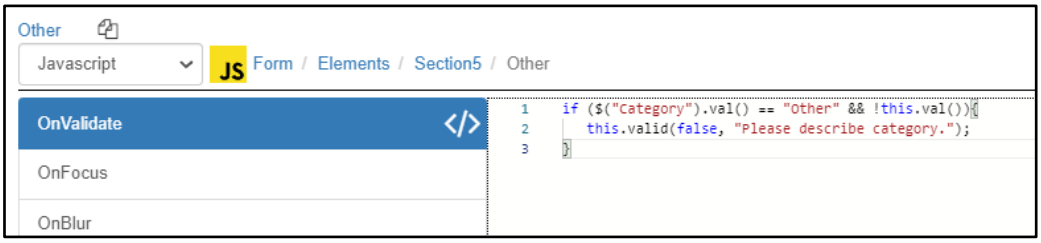
Events

When you select an element, a script button appears which looks like '</>'. Once selected the events that can be used in script are listed on the left hand side of the Script Template window. These events will differ depending on whether you select a Form, Page, Section, or an element that can accept input, such as a text box. Button elements have a unique event called 'OnTap' that is activated when the button is pressed.

Form	Page	Section	Element
OnStart OnOpen OnClose OnValidate OnSave OnSubmitAndConfirm OnForward OnReturn OnWorkflow Globals	OnValidate OnEnable OnDisable OnShow OnHide	OnValidate OnEnable OnDisable OnShow OnHide	OnValidate OnFocus OnBlur OnValueChange OnEnable OnDisable OnShow OnHide



You can quickly check if any scripting has been entered against an element's events, by selecting the element and viewing the script button ('</>'). If any scripts exist against events for the element the button will appear green, whereas if the element does not have any script against it, the button will appear grey. Commented out scripting will still show as green.



Selecting the script button will open the Script Template window and display the script for the highlighted event. If no code has been entered, a blank template will display with no events highlighted.

Properties

Properties are values that an element possesses. These properties include, "enabled", "message", "valid" and "visible". When you use script to change these properties, you can change the way an element behaves or appears on the device.

Methods

Methods are instructions that you use to change an element's properties. Normally you would use the Script Template window to place these method 'calls' in an element's events. Most elements support the OnValidate() and focus() methods. The OnValidate method will run any code in the target element's OnValidate event. The focus method will set the focus to the target element.

To set the focus to an element that is on a different page to that being viewed on the device, first use the gotoPage() method to select the required page. The format for using this is:

App: `gotoPage("PageName"/pageNumber);`

Webforms: `$.gotoPage("PageName"/pageNumber);`

You can use the Page name or Alias. This must be enclosed in quotes, as in the example. As always, the instructions are case sensitive.

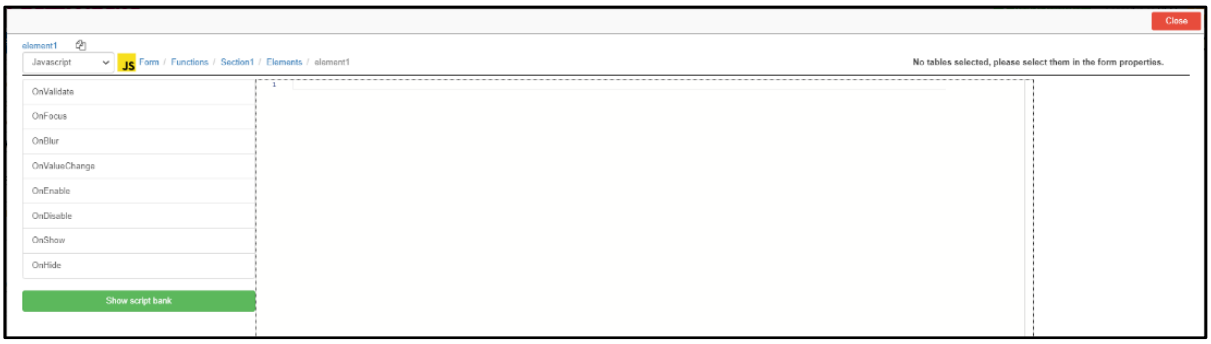
The format to use of the OnValidate() and focus() methods is:

```
$("element").fire("OnValidate") and $("element").focus()

$("element").validate() // The validate() method is available on Webforms only
```

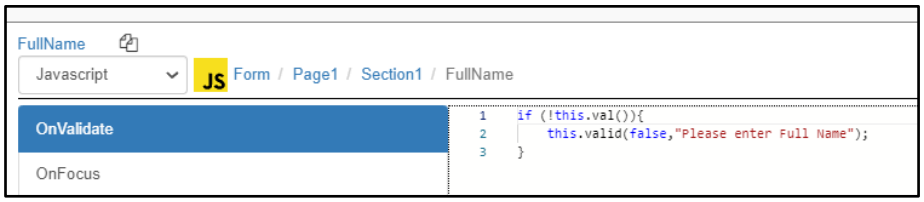
The Script Template window

Introduction



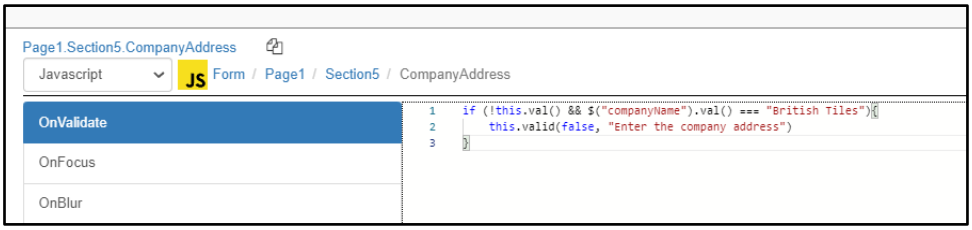
Script can be entered against any of an element's events, but the Warn and Valid properties can only be set in the OnValidate event. To place script against an event, select the element, then the script button. The Script Template window will open with the element name selected in the top left, and the possible events underneath. Then select which event you would like to place the script in, this will highlight it in blue. You can change the element selected by hovering over the element path separated by '/' to enter additional script against any other element or events.

OnValidate Event - Checking for valid input



The most common use of scripting is to ensure that valid entries have been made. You can perform this by selecting the element, then selecting the OnValidate event. You test the value for the element and if this is incorrect, you set the element's Valid property to false and display an appropriate error message. The term 'this' refers to the selected element, so the line 'this.valid(false,"Please enter Full Name");' is setting the Page1.Section1.FullName text box element's Valid property to false. This will prevent the form from submitting and cause the value in the element's Message property to be displayed. In this instance, the message property has also been set to "Please enter Full Name".

Using the && "and" keyword and Alias names – validating multiple values



It is possible to check the values from multiple elements in script. To achieve this, place the validation code in the OnValidate event of one of the elements and refer to the other elements either by their full names, or by their Aliases. Aliases are unique names

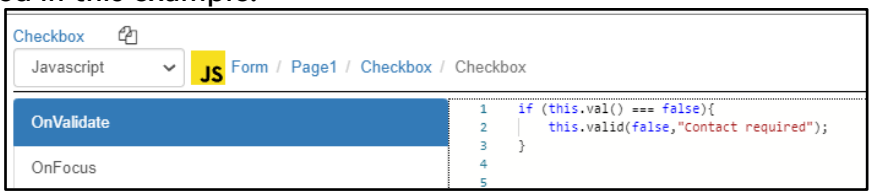
supplied when the elements are named in the Form Designer. Obviously employing aliases entails less typing. You use the "&&" keyword to combine the fields of the query.

Checking for empty elements/fields

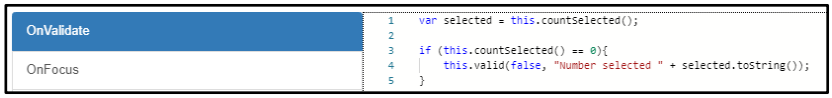
You can check if most of the elements have been left empty by testing whether they do not have a value ('!' for App and Webforms, or isEmpty() for Webforms). You can also check if they have the value of an empty string (""). The elements that can be tested this way include:

- Text elements (including text, number, email, phone etc).
- Paragraph text
- Date
- Time
- Single-select
- Photo
- Sketch
- Signature

To confirm if a checkbox has been selected, you use the true and false keywords. As demonstrated in this example.



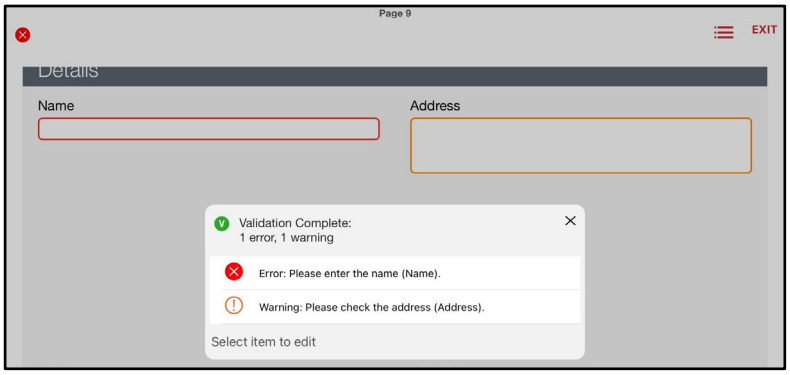
To confirm if a selection has been made from a multiple selection element, you can use the countSelected() and getOption() functions. CountSelected() returns the number of items selected and getOption() will return true or false, depending on whether the option specified within the brackets has been selected.



Note: these functions are only available on the App. To achieve this on Webforms, you can check if the multiselect field has a value (eg: if ($("multiSelect").val()){..) or use the hasChosen function (eg: $("OtherText").visible(this.hasChosen("Other"));).

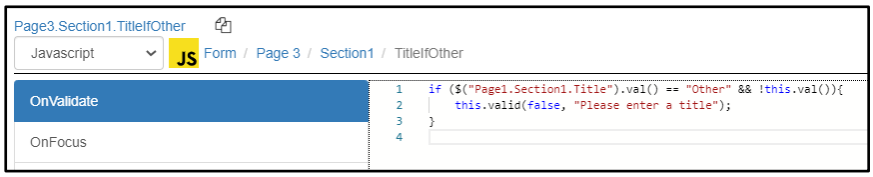
Device Submit Window

When a device user attempts to submit a form, and the script for a required element sets the element's Valid property to false, a message will appear on the Form Submission window informing the user that there is an error in their input. They will be prevented from submitting the form. Multiple elements can have their Valid properties controlled in this way, and an individual line of text will appear on the device for each element that has its Message property set to a value.

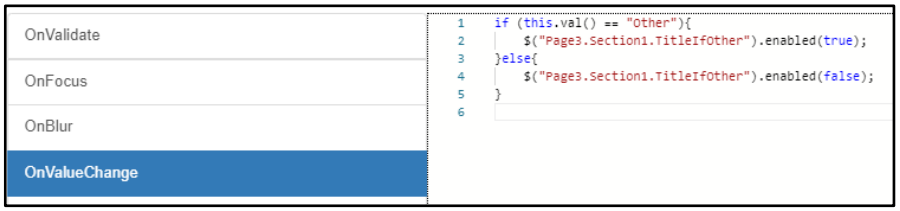


You can set the Message property for an element without setting its Valid property to false. In this way you can warn a user that their input may be incorrect, without preventing them from submitting the form. When the Message property is set in this way, it is indicated on the device by an amber circle containing an exclamation mark.

OnValueChange event

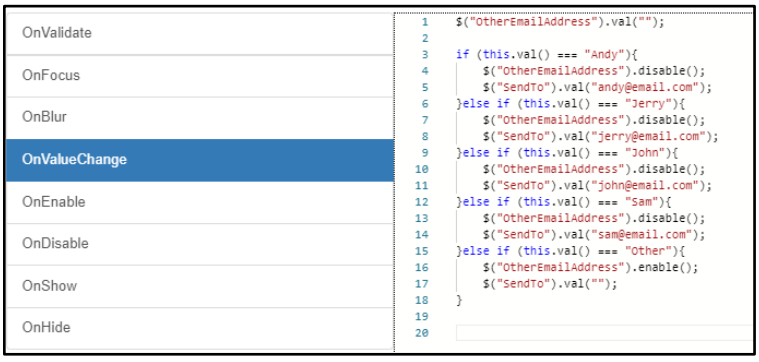


The OnValidate event is used to test data when the device user submits a form. But you may wish to change the value or condition of an element prior to this point – for example when the user enters data in a text box, or selects a value from a drop down list. In this case, you could use the OnValueChange event.



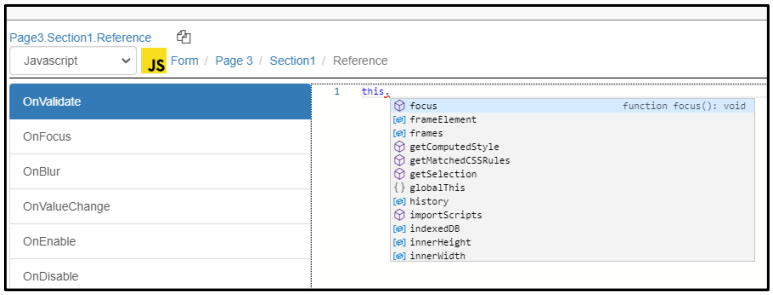
When a user makes a change, such as selecting a value from a single selection element, it is possible to change the values of another element's properties. For example, if the user selected 'House' from a single selection element, you could disable other elements that relate to flats. In this example, if the "Other" option in a salutation single selection element is selected, the element's OnValueChange event is used to enable a text element (TitleIfOther), to permit the user to enter an alternative title.

The OnValidate event of the TitleIfOther element then ensures that if "Other" has been selected, text has been entered.



A more complex example could be to check the values from a list of names, and use them to populate an email address field. The script in the following example will populate the hidden field, "SendTo" with a value depending on the selection from a single selection element. If "Other" is selected, then the "OtherEmailAddress" field is enabled, so it can be used instead. It is also possible to directly acquire the device user's email address, using the u*serEmail()* function, which returns a string value of the logged in user's email address.

Intellisense



Formworks supports rudimentary Intellisense. Currently this works using either the "this" keyword, or element alias names, prefaced with '$'. To use Intellisense, you need to use the fully qualified element name or Alias, enter a full-stop, then select the Ctrl and Space keys simultaneously. A drop down list of available properties will appear.

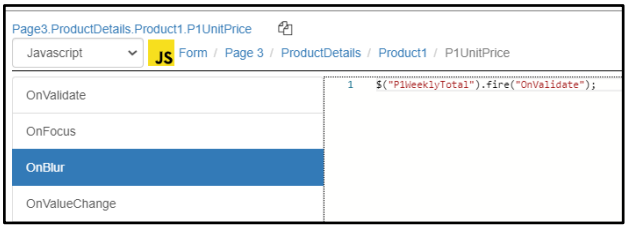
Calculated Fields

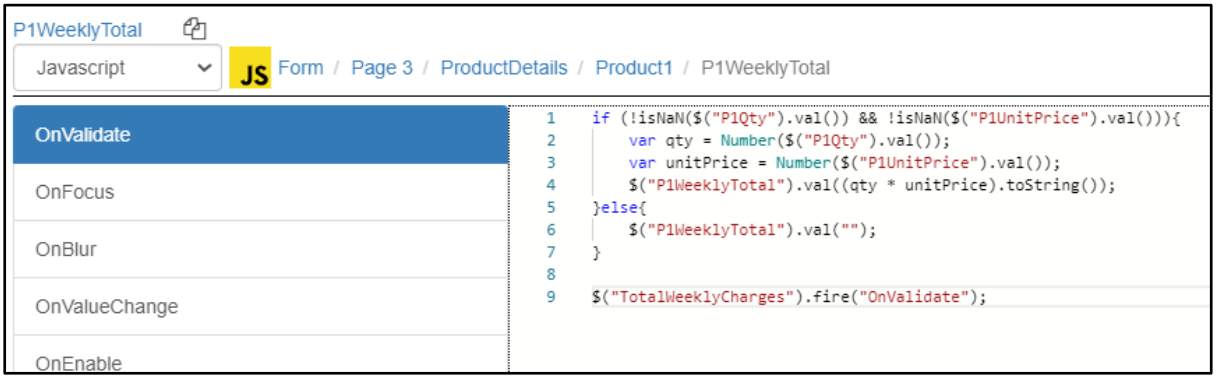
Introduction

Using the value from one field to populate another is covered throughout this guide. But because performing calculations and displaying the result is so basic to many user's requirements, it is covered specifically here.

Before performing a calculation based on the contents of a field, you need to confirm that the field is not empty and contains a value. Otherwise, your script will generate an error. You do this by testing the field's value with the isNAN() function.

The following example is quite typical. Changing the values within a number of fields, such as quantity and price, can cause the total to automatically update. To achieve this, you place the calculating code in the OnValidate event of the total field, and call the OnValidate event from the OnBlur event of the quantity and price fields. This method causes the value of the total to be updated as the user exits the fields that are involved in the calculation.





You should also check in case the user goes back to the field and subsequently removes a value. That is why in the example, the code not only enters the multiplication of the price by the quantity if values have been entered, but also places an empty string in the total field if no value is found.

You may wish to allocate Alias names to the fields involved in the calculations, to reduce the complexity of the script. It may also be useful to make the total fields 'Read Only' to prevent users from entering values directly.

Where possible, combine your script with the same events of an element. For example, if you have two pieces of script, one within the OnBlur event, and one in the OnValidate event, and these can both be effectively contained in the OnValidate event, then do so. Events have a processing overhead and reducing the number employed can speed up device input.

Global Variables – App Only

Introduction

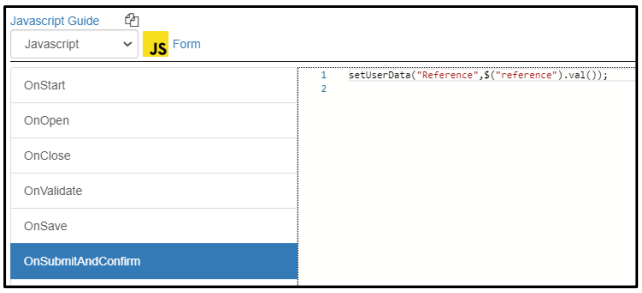
Global Variables are variables that are entered in one form, and can then be retrieved by the same user from the device's memory and used on subsequent forms. This data is set on a per-user / device basis. Global variables retain their value for 90 days from the point where they are set. Following this point, they must be re-saved to retain their values.

Two methods are employed to work with Global Variables - setUserData and getUserData. The format for their use is:

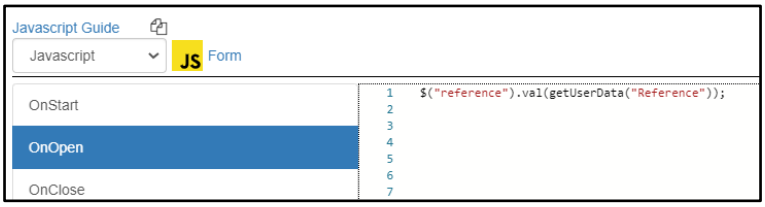
```
setUserData("Key",value);
getUserData("Key");
```

For example, to save the value from a text box with an alias of reference, into a Global Variable named Reference, the two methods would be coded as follows:

```
setUserData("Reference", $("reference").val());     -- Store the value of the reference element.
var ref = getUserData("Reference");                  -- Retrieve the value from "Reference".
```



In this example, the OnSubmitAndConfirm event of a form is used to capture the value from a text element with an alias of reference, and store it in a global variable named Reference. You can then employ the OnOpen event of a different form to retrieve the value from the Global Variable Reference, and place it into the Value property of the second forms Reference field.



Custom Properties – App Only

Introduction

Custom Properties are used to store temporary values that are globally available to any page on a form. Custom Properties are similar to Global variables and hidden fields, but they differ in a number of important ways. Unlike Hidden fields, they are not exported and will not appear in CSV or XML output. Unlike Global variables, the values in Custom properties will not be saved when a form is closed.

Custom properties can be used instead of Hidden fields, in a situation where the value doesn't need to (or mustn't) be submitted and exported along with the rest of the field data. They can contain other data types in addition to text. See the comparison chart below for a comparison between the features of Custom properties, Hidden fields and Global variables.

Custom properties can have any name except the reserved property names of "title", "visible", "enabled", "valid", "color", and "message". Custom properties are global to a form, and therefore available from any page. They can be set and retrieved in the same way as a standard variable – for example:

```
$("field").prop("roomsWithSmokeAlarm", 4);
```

The above example could be used on house inspection forms where there are sections for each room and each section contains a check box indicating if a smoke alarm is present in the room. The OnValueChange event for each of these check boxes could update the `$("field").prop("roomsWithSmokeAlarm")` Custom property and the value of this property then used elsewhere on the form.

Features of Custom properties

- Globally available to any page within a form.
- Temporary storage. Their value is lost when the form is saved or submitted.
- Does not become part of the output, and will not appear on CSV, PDF or XML data.
- Must not be named the same as standard properties.
- Values are set using the same notation as standard variables.
- Can contain any data type.

Comparison chart

Feature	Custom Props	Hidden Fields	Global Fields
Globally available in form	Yes	Yes	Yes
Saved on device after submission	No	No	Yes
Forms part of the output data	No	Yes	No
Part of the User Interface	No	Yes	No

Can contain any data type	Yes	Yes	Yes

Date and Time Functions

Getting Current Date and Time

To insert today's date or time into a field, prior to the forms submission, place the following script into the elements OnValidate event:

App Only:

```
this.val(today());
```

or

```
this.val(timeNow());
```

Web Only:

```
this.val(today);
```

or

```
var time = today.getHours() + ":" + today.getMinutes();
this.val(time);
```

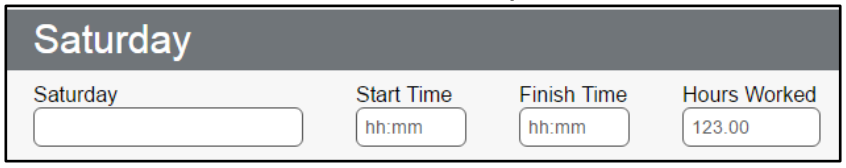
Webforms Only:

Webforms utilises the following Get Date methods:

getFullYear()	Get the year as a four digit number (yyyy)
getMonth()	Get the month as a number (0-11)
getDate()	Get the day as a number (1-31)
getHours()	Get the hour (0-23)
getMinutes()	Get the minute (0-59)
getSeconds()	Get the second (0-59)
getTime()	Get the time (milliseconds since January 1, 1970)
getDay()	Get the weekday as a number (0-6)

Calculating the difference in minutes between two times

Calculating the difference between two times is a common requirement. In the example, the user would enter the time they started and when they finished. These times could also be entered by tapping on a button element.



The system automatically performs the calculations to place the total in hours, and fractions of hours in the Hours Worked field. Care has to be taken here, as the employee may work over the midnight period.

To calculate the difference in time the 'subtractTime()' function can be utilised. The below example is placed within a button field and initially checks that the two time fields (startTime and endTime) both have a value. Depending on how you want to display the total time there are a couple of ways to do this. If you need the total time in hours as a decimal, the code on line 6 will give you that result. If you need the total in minutes, line 8 will achieve this.

```
1   var sTime = $("startTime");
2   var sMin = sTime.prop("minutes");
3   var eTime = $("endTime");
4
5   if (sTime.val().length == 5 && eTime.val().length == 5){
6       var differenceHrs = (eTime.subtractTime(sMin))/60;
7       $("resultInHours").val(differenceHrs.toString());
8       var differenceMin = eTime.subtractTime(sMin);
9       $("resultInMinutes").val(differenceMin.toString());
10  }
```

The above can be achieved in less lines, however it can appear quite crowded

```
if ($("startTime").val().length == 5 && $("endTime").val().length == 5){
    $("resultInHours").val(($("endTime").subtractTime($("startTime").prop("minutes"))/60).toString());
    $("resultInMinutes").val($("endTime").subtractTime($("startTime").prop("minutes")).toString());
}
```

Table Elements Scripting

Referencing Table cells in script

```
if ($("Page1.Section1.BudgetTable.cell_1_1.Text1").val() && $("Page1.Section1.BudgetTable.cell_1_2.Text1").val()){
    this.val((Number($("Page1.Section1.BudgetTable.cell_1_1.Text1").val()) + Number($("Page1.Section1.BudgetTable.cell_1_2.Text1").val()).toString()));
}

```

You cannot refer to the value property of an element in a Table cell, without using the elements name. This method of referencing can only be used for elements that have a value property, for example, text boxes.

Looping through Table cells

As you need to refer to the elements within a table cell by name, you can create a looping structure very simply as long as the field names within a column or row as needed are the same. This example uses the third column of a table and loops through the first three rows, placing the value 10 in an element named Quantity.

```
for (i = 1; i <= 3; i++){
        $("Page1.Audit.MajorRisks.cell_3_" + i + ".Quantity").val("10");
}
```

Databases

Introduction

Creating JavaScript tables within the form template has the limitation that the table items, which normally contain the contents of a dropdown list box, are hard coded into the template. Although dropdown lists are editable after a template is published, if there are many dropdown fields to update, this can be a big job.

Databases offer a convenient alternative to this process, whereby tables of data in the shape of CSV (Comma Separated Value) files, can be uploaded to the Formworks portal, and synchronised to the user's device. There are many possible uses for databases within a form, and examples could include:

List boxes

- Populating a drop down list box with client names, or branch addresses and contact details, rather than hard coding them into the template.
- Populating a second list box (or text box), based on the value selected in another list box. Multi-tiered list box population is supported in this fashion.

Quotations and calculation

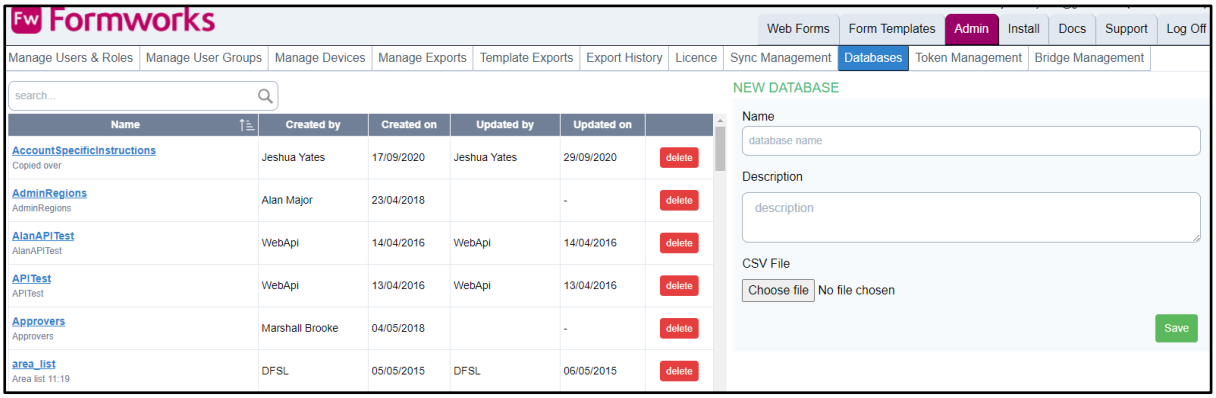
- Providing a look-up database of cost values for products selected on a form. Databases can contain all the product cost information necessary to provide up to date quotes as a form is completed.

Workflow and Project management

- Returning an email address or URL for a contact/employee selected on a form. A form could be automatically emailed to a contact or group of contacts, depending on information within the form.
- As a form is aware of the user's name when it opens, list boxes could be populated with information specific to that user. Using databases the system could populate list boxes with only the projects or clients that are applicable.
- You could automatically provide the device user with a list of additional forms that should be completed, based on the data entered within the current form. For example, a certain client may always require an additional 'Notes' document.
- The 'rules' regarding specific clients could be represented within databases which are subsequently interpreted within script to guide user input

Maintaining your databases

When you have prepared the CSV file, select Admin, then Databases.

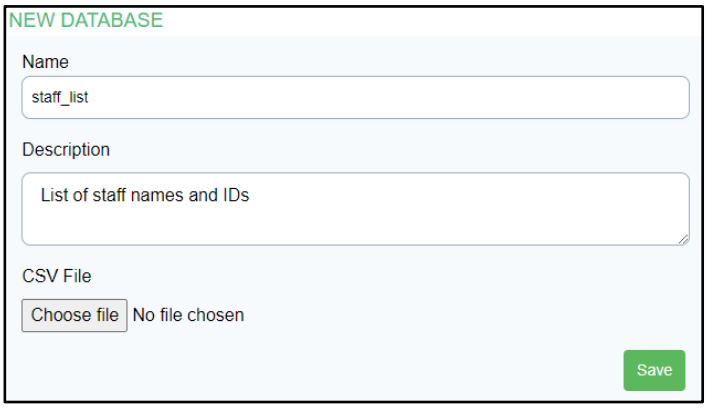


Creating and Updating databases

Making a CSV data file

To create a Formworks database, you need to save the database to a CSV file. The easiest way to do this is probably to create an Excel worksheet, placing your database data in columns, then save it as a CSV file instead of a workbook, using the File, Save As option, then change the file type to CSV.

Uploading the CSV data file



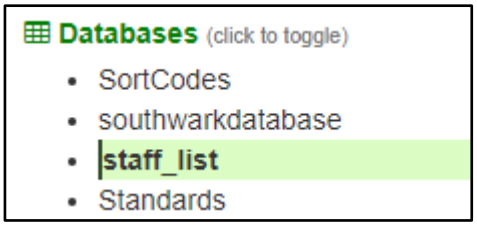
- From the databases tab, enter the name of the new database in the Name field on the right of the screen. The name must not contain any spaces, but you can use underscores, for example, staff_list.
- Enter a description.
- Browse to the CSV file and select it.
- Select Save.
    - A message indicating that the file has been saved will display.
    - The file will appear in the Saved Databases section. The name would have been converted to lowercase.
    - The same process is used both to create a new database and to update an existing one. To update an existing database, simply use the same name.

Note:

You cannot delete a database that is in use by a published template. The template must be retired first. If you delete a database in use by a template in Design state, the database will need to be added via the Template Designer.

Using databases on your form templates

Once you have uploaded your database, use the Template Designer screen to attach it to your template. On the Form Properties section of the designer above Template Icon, use the Database drop down list to select the database you wish to use in your template.



Populating a basic list box

The script example here is the basic code to populate a list box from a database column. The database is called listdatabase, and we are populating the 'dropdown' list box with the values in the Nationality column. Both the display and return (key) value of the single select element will be the same. It is common to place such basic list box populating code in the form's OnOpen event. However, it can also be placed in the OnValueChange events of other list boxes etc.

Lines 5 and 11 create the query and extract the Nationality column from the listdatabase database. Line 1 to 3 will check if the list box has a value and then clears the contents of the list box if it is empty, ready for new values. Lines 6 and 19 are functions contained within the executeQuery method, which checks that records have been returned from the database that match your query. Lines 11 and 17 processes each record extracted. Lines 14 and 17 are a nested for loop that processes every column in the 'current' record. Line 16 populates the Nationality list box with the value of the column. Line 18 is available for use on Webforms only, the refresh() method will refresh the list box to load the new values as browsers can incur a delay in reading and writing the list from the database.

Note: Populating a basic list box in the Formworks XD App can be done much simpler as it is now a built in function. This has been reduced down to one line, which can be placed in the OnOpen event of a template or in the OnValueChange event of a dropdown field. This new function requires 4 parameters, the database name, the column you wish to pull data from, the keyValue and the displayValue respectively. The below example will populate a list box given the alias ProductList from two database columns. The database is called Products, and the column it looks for is called Unit. The key value will come from the Rate column and the display value from the Unit column. The key value will always be displayed within the '[]' brackets.

```
$("ProductList").setDatabaseConfiguration("Products", "Unit", "[Rate]", "Unit");
```

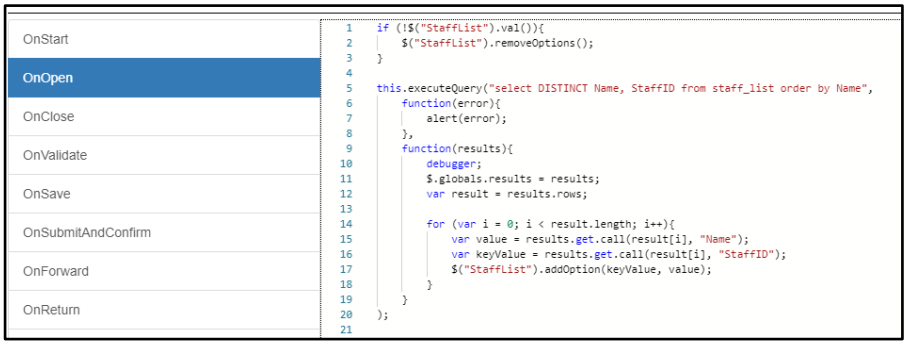
To display more than one column value in the list box, columns can be concatenated using '||'. In the below example, the Rate and Unit columns will be displayed in the list box.

```
$("ProductList").setDatabaseConfiguration("Products", "Unit", "[Rate]", "Rate || ', ' || Unit");
```

Note: the setDatabaseConfiguration function is available on the App only.

Populating a list box with different display and return values

List boxes can contain both display and return (key) values that are hidden from the iPad device user. For example, a list box can display the name, "Alan L Major", but return the code "ALM" when selected. The next example demonstrates how to query the index variable to tell which database column is currently being processed within the do, end loop, so you can use both a display and return value.



In this example, if the column being processed is Name, the value from that column is placed in the 'value' variable. If the column is StaffID, the value is placed in the 'keyValue' variable. After processing the current record, (Lines 15 and 16), we populate the StaffList list box on line 17. The 'keyValue' forms the hidden, return value, and 'value' is the value that will be displayed in the list box. Then the processing continues to the next record extracted from the database, and the cycle (Lines 14 and 18) continues.

Note: To populate a listbox with different display and return values in the Formworks XD App is similar to the example shown in the 'Populating a basic list box' section. In the below example the return value will be from the County column, while the display value will be the Postcode.

```
$("County").setDatabaseConfiguration("counties", "County", "[County]", "Postcode");
```

Note: the setDatabaseConfiguration function is available on the App only.

The SQLite Select Statement

Within your script code, you use SQLite select statements to retrieve data rows from your databases. There are many sources of information on the Internet regarding SQLite and how to format your queries, and one good source of information is:

https://sqlite.org/lang_select.html

http://www.tutorialspoint.com/sqlite/sqlite_select_query.htm

These can range from relatively simples queries that return all the databases columns (* character):

```
select * from Staff_list
```

Where you simply wish to populate the text of a list box, to more complex queries where the values in more than one column of the database must meet certain criteria. For example, if you were searching for both an area and skill set match. Where you do not require all the columns of a database to be returned, you can specify the column names:

```
select StaffID, Name from Staff_list
```

Querying using Wild Cards

If you wish to search a database with only a partial value, for example, you may wish to retrieve all clients with DA contained within their postcodes, then to accomplish this, you would use wild cards. There are two wild card characters, the underscore (_) and the percent character (%). The underscore represents a single character, whilst the % represents any number of characters. To return a results set that contained all clients with surnames beginning with "smi", the query might be:

```
select ClientName, ContactNumber from Client_List where Surname like 'smi%'
```

This would return all records with any number of characters following the smi. To return all client records that contained smi anywhere within the surname name field:

```
select ClientName, ContactNumber from Client_List where Surname like '%smi%'
```

The percent characters representing any number of characters both before and after the smi characters.

Querying Multiple Columns

The 'and' clause can be added to your queries to query multiple database columns. For example, to query the Staff_list database by both part of the name, and an exact area match, the query could be:

```
"select StaffID, Name from Staff_list where Name like '%smi%' and Area = 'SE'"
```

And if you wished to substitute the values in text box elements for the staff name and area:

```
"select StaffID, Name from Staff_list where Name like '%" + $("StaffName").val() + "%' and Area = '" + $("Area").val() + "' order by Name"
```

Combining And, Or and Where statements

You can combine the key words, 'and' and 'or' into your query. The following example shows how to do this. If you look closely, you will notice that numeric values do not need to be enclosed within single quotes, in the way text does.

```
"select StaffID, Name from Staff_list where name like '%" + $("StaffName").val() + "%' and Area = '" + $("Area").val() + "' or Salary > " + $("Salary").val() + " order by Name"
```

When the values from your forms elements have been inserted into the query, it could look like this:

```
"Select StaffID, Name from Staff_List where Name like '%Alan%' and Area = 'NE' or Salary > 30000 order by Name"
```

Note: SQLite uses a single = character to represent an exact match, whilst JavaScript uses two == or three === to indicate that one value equals another.

Joining Databases

Types of Joins

There are different types of joins available in SQLite:

- INNER JOIN: returns rows when there is a match in both databases.
- LEFT JOIN: returns all rows from the left database, even if there are no matches in the right database.
- RIGHT JOIN: returns all rows from the right database, even if there are no matches in the left database.
- FULL JOIN: returns rows when there is a match in one of the databases.
- SELF JOIN: is used to join a database to itself as if the database were two databases, temporarily renaming at least one database in the SQL statement.
- CARTESIAN JOIN: returns the Cartesian product of the sets of records from the two or more joined databases.

Operators

Different operators can be used to join databases, i.e., such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT. However, the most common operator is the equal symbol.

The example below demonstrates the use of an INNER JOIN. This has been used to join two normalised databases, a situation common with relational databases. The area has been represented as a code,(N,S,E,W) in the Staff_list database, whilst the Area_Code database contains both the codes and description of the area. The two databases are joined in the query by the Area_Code field, so staff details can be extracted from the Staff_list database, and the description of the area retrieved from the Area_list

database. Where a field name is present in both databases, it is good practice to precede its name with the database name, to avoid confusion.

```
"select StaffID, Name from staff_list INNER JOIN Area_list On staff_list.Area_Code = Area_list.Area_Code wh
ere Name like '%" + $("StaffName").val() + "%' and Area_list.Area_Code = '" + $("Area").val() + "' order by N
ame"
```

Populating cascading list boxes

To populate a second list box, based on the selection of an entry in the first list box, you would introduce a "where" clause in the SQL query. For example:

```
"select ClientName, ClientCode from Clients where area = '" + $("Areas").val() + "'";
```

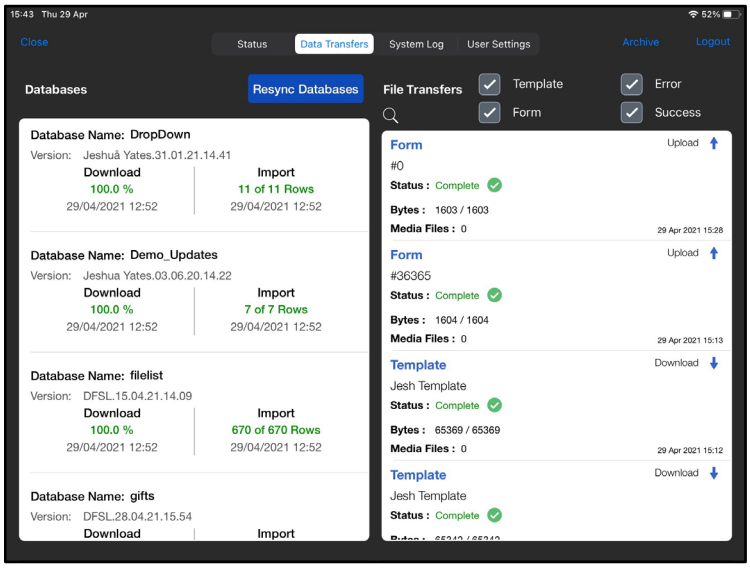
This query would take the value from the Areas element, (either a list box or text box), and when executed against the Client database, only extract clients that matched the area code. The + characters are used to concatenate the value from the Areas element into the query, and the apostrophe character (') that is required to define the appended value. So the actual query string should look something like this:

```
"select ClientName, ClientCode from Clients where area = 'Area1'"
```

Once you have extracted the results set from the database, you can use the above code to iterate through the data records, using the index value to identify the field and the value variable to retrieve the column value. Using this method, you would normally never require to extract more than two columns at a time from a database; one to display on a list box, and one as the return value.

Local Databases on the device

You can monitor the status of the databases on the device, from the Settings screen. Databases are displayed on the left side of the Data Transfers tab of the Settings screen.



The database name, and date and time the database 'version' was uploaded to the portal is displayed here.

Databases are Downloaded to the device, then Imported for use by the form templates. The status of both of these stages is indicated real-time as they load.

If a database fails to load for any reason, the Resync Databases option will completely delete the databases from the device, and download them again.

Properties - General

valid Type: Boolean

Where a "valid" property is present, setting its value to false will prevent the form from submitting. The valid property, like the message property, can only be set in the OnValidate event of an element. Setting it in any other event will not cause an error, but it will be ignored by the application. You set the message property with the valid property.

The format for setting a valid property to false is:
```
$("element").valid(false, "This is an invalid field");
```

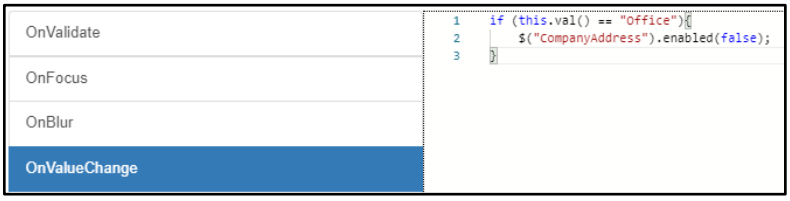
warn Type: String

The warn property of an element is used to display warning messages in the Submit window, when the device user selects Submit Form. The most common usage of the warn property is to deliver a meaningful message to the device user when incorrect data has been entered on a form, or when a field has not been completed, but is not required. The warn type is a warning and displays a red icon. The warn property, like the Valid property, can only be set in the OnValidate event of an element.

The format to set the Warn property for an element is:
```
$("element").warn("This is a warning message");
```

enabled Type: Boolean

The Enabled property allows you to enable/disable an element, or read an element's current state. This can be based for example on the value from another element. In this example, if the value of the text box element, CompanyName1 is changed to "Office", the text box element (alias) companyAddress will be disabled.



You can enable or disable an entire page, by setting its Enabled property to true/false. Setting a page's Enabled property to false will disable all the elements on the page. The page will still be accessible from the pages drop down navigation control, and the 'next page' and 'previous page' buttons.

The format to set the Enabled property of an element is: `$("element").enabled()` or `$("element").enabled(true/false);`

visible Type: Boolean

The Visible property allows you to read or set the visibility of an element to true/false and effectively make the element disappear from the form. This could be based on another value, as with the Enabled property. You could check the value of a text box, and if it were not appropriate to accept input into another text box, you could make it disappear. Most users find the appearance and disappearance of fields on a screen unsettling, and it is normally better practise to disable fields, rather than making them disappear.

The format for using the Visible property is: $("element").visible() or $("element").visible(true/false);

title Type: String

The title property of an element is the text name that displays against the element, not its value. On a button element, this would be the text on the button. For a check box, it would be the text description that displays by the check box and for a text box, the text that displays above the text box.

You can read the value of the title property, or write to it. To do this you need to use the prop() function. The prop function of an element enables access to read and write other properties of an element and is not limited to the title property.

To read the title property, you can use:

```
var fieldTitle = $("textField").prop("title");
```

To change the title property of an element, you can use:

```
$("textField").prop("title","New Title");
```

Note: The title properties of the container elements Page, Section and Group, are read only on the App. It is possible to change these on Webforms. They cannot be changed within script. Page titles can be changed within script in the Formworks XD App Only.

prop

The prop property of an element enables access to read and write other properties of the element.
The prop function can be used to read or write the following properties:

- Web only: color (read & write), textColor (read & write)
- App only: valid (will return true, until form has been validated), color (write only), textColor (write only), message (after form has been validated only), custom defined properties
- All platforms: value, title, enabled, visible

Read property: `var x = $("field").prop("value");`

Write to property: `$("field").prop("color", "green");`

There are also a range of date/time properties also available, these have been covered in the relevant chapters.

isIphone() Type: Boolean

The isIphone property can be used to check the device type being used and is read only. This will return true if the device being used is an iPhone and false if the device is an iPad.

```
var device = isIphone();
if (device) {
  $("Device").val("iPhone");
}else{
   $("Device").val("iPad");
}
```

Note: isIphone() is available on the Formworks XD App only and not on Webforms.

Events - General

OnBlur

This is the 'Lost Focus' event. When the device user exits one element by selecting another, the OnBlur event of the first element is fired. You could use this event to check the value of an element and then enable or disable other elements based on the value.

OnEnable

An element's enabled property can be used to enable or disable the element. When an element is enabled, the OnEnable event will fire.

OnDisable

An element's enabled property can be used to enable and disable the element. When an element is disabled, the OnDisable event will fire.

OnFocus

The OnFocus event is triggered when the device user accesses an element, for example by selecting a text box element or drop down list box.

OnHide

An element's Visible property can be used to make the element visible or invisible. When the element's Visible property is set to false, the element's OnHide event is fired.

OnShow

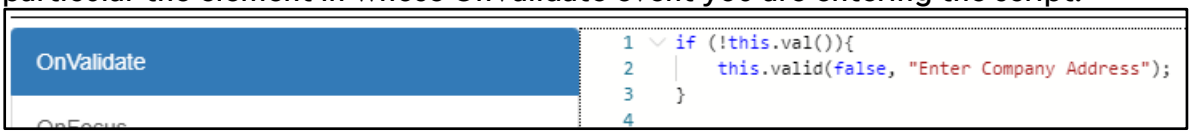
An element's visible property can be used to make the element visible or invisible. When the element's visible property is set to true, the element's OnShow event is fired.

OnTap – button element specific

This event is fired when you touch a button element. You can create button elements and place script in their OnTap events to perform actions when they are selected.

OnValidate

The OnValidate event is where you place script that is run when the device user selects to submit a form. You can perform checks on the values of elements on the form, in particular the element in whose OnValidate event you are entering the script.



The normal process is to check the value of various elements, then set the element's "valid" property to false if you wish to prevent the form from being submitted. You should also enter a meaningful message to be displayed in the Form Submit window when they select to submit the form. In this way the device user knows why they cannot submit the form. The valid property can only be set in the OnValidate event. Fields can be set to mandatory or required within the Form Designer under the selected elements properties. If this method is used instead of the above scripting, then the error message displayed will be 'Required:' followed by the fields name or alias.

If you need to enter the same script against a number of fields within a container, such as a page, section or group, you could place the code within the container's OnValidate event and call it using the container's fire("OnValidate") method. In this way, instead of entering the same script in a number of locations, it could be centralised. Then if changes need to be made, they would only need to be made once.

Even though Forms, Pages, Sections and Groups have an OnValidate method and event handler, calling OnValidate on a page for example, does not trigger the OnValidate() methods of the fields inside that page. It is only when the form is submitted that each element on the form has its OnValidate() method called.
In Webforms however, there is a 'validate()' function that can be called at any time to run the scripting within a field's OnValidate event. The format for this is:

```
$("element").validate();
```

OnValueChange

This event is similar to the OnBlur event, in that if you change the value in a date element, it would fire as you leave the date field. The event will also fire when a field element has been changed, for example in a text field this will fire after every key press. Note: In the Formworks XD App this event will fire after 2000mseconds of the last keypress for Time and Number elements and 400mseconds for normal text fields or dates.

Form Specific Events – in order of occurrence

OnStart

This event is fired before the OnOpen event and only once in the life cycle of a form.

OnOpen

This event fires every time a form is opened and is useful for setting up variables, calling a database etc.

OnSave

This event was designed to prevent the loss of data, for example should the device's battery expire unexpectedly. The event fires automatically at various points during completion of a form - when values change, particularly sketches, signatures and photo fields, but also when other field values like text boxes change. The OnValidate event is NOT fired for this option, and the form will not be validated until Submit or Validate Form is selected. This event has virtually no use as far as user developed scripting is concerned and does NOT relate to the Save And Close option. Because the event fires so often during device user input, placing code here will greatly reduce the response times moving between fields.

OnClose

This event is fired when a device user selects the "Save And Close" option. This is the event to place script that is read directly before a form is saved onto the device to be completed and submitted at a later time.

OnSubmitAndConfirm

This event fires after the device user select the Submit option from the Submit Form window. This event could be used in conjunction with the Global methods getUserData() and setUserData() to pass information between forms.

Globals – App Only

Available for use in the Formworks XD App Only. This event runs initially before the OnStart or OnOpen events. This event was added to be used as a space to put all global functions and variables. Script to populate dropdown fields from databases can also be placed in this event. Functions placed within this event that also need to be run within the original Formworks app or Webforms will need to be placed in the forms OnOpen event as well.

Form Specific Methods and Properties

Audio Functions – App Only

startRecording()

Toggles recording. First time this instruction is issued, recording will start, second time it pauses recording, third time it starts recording again. Format: `startRecording();`

pauseRecording()

Pauses an ongoing recording. In the Formworks XD App this will toggle the recording once a recording has already been started. Recording will pause, second time it will continue the recording, third time will pause again. Format: `pauseRecording();`

stopRecording()

Stops a recording and an alert to save and or rename the audio file will appear. The filename can be set within scripting, however this is mandatory. Format:
`stopRecording();` or `stopRecording("recording1");`

recordingStatus()

Returns the status of any active or inactive recordings. Recording Status has three possible values:

Active	Recording is currently in progress
Paused	Recording has been paused
Inactive	Recording has been stopped. There is no current recording.

Format: `$("status").val(recordingStatus());`

Miscellaneous Functions

gotoPage("AliasName/PageName" / Page Number)

This form level method moves the focus to the page specified within the quotes. Either the page name or an alias can be provided, but this must be enclosed in **straight** quotes, as in the example. To set the focus on a specific field on a different page, first issue the `gotoPage()` instruction, followed by the `$("element").focus()` method. It is not possible to move directly to a field on a different page without first issuing the `gotoPage()` instruction. See below for the format:

App: `gotoPage("Page2");` or `gotoPage(2);`

Web: `$.gotoPage("Page2");` or `$.gotoPage(2);`

openURLPage(strVar) – App Only

This method opens the input device's default web browser, normally Safari, with an iPad, and opens the URL specified in the method's string parameter. For example, `openUrlPage("https://www.bbc.co.uk/weather/2643743");` would open the BBC weather website for London. You can move back to the Formworks input screen by selecting Formworks, from the top right hand side of the screen.

Enabled() Enable() Disable()

The enabled() function reads the enabled property of an element. This will return true or false. The property can also be set using this function. Both examples below will effectively achieve the same outcome of toggling the enabled state of an element.

```
if ($("AliasName").enabled()) {
   $("AliasName").enabled(false);
} else {
   $("AliasName").enabled(true);
}
```

Or:

```
if ($("AliasName").enabled()) {
   $("AliasName").disable();
} else {
   $("AliasName").enable();
}
```

.enable() and .disable() will work in the same way on Webforms and the App. The enable() function will enable an element, while the disable() function will disable an element. Format:

```
$("AliasName").enable();
$("AliasName").disable();
```

Visible() Show() Hide()

The visible() function will read the visible property state of an element. This function can be used to read the elements state or change it.

```
if ($("Page1.Section1.Signature1").visible()) {
   $("Page1.Section1.Signature1").visible(false);
} else {
   $("Page1.Section1.Signature1").visible(true);
}
Or:
if ($("Page1.Section1.Signature1").visible()) {
   $("Page1.Section1.Signature1").hide();
} else {
   $("Page1.Section1.Signature1").show();
}
```

.show() and .hide() will work in the same way on Webforms and the App. The show() function will change an element's visible property to visible, while the hide() function will change the visible property to hidden. Format:

```
$("AliasName").show();
$("AliasName").hide();
```

Appendix I

Elements - their properties, events and methods

Buttons

Properties

enabled	Boolean, read or write. Format: $("element").enabled(true/false);
enable	Format: $("element").enable();
disable	Format: $("element").disable();
visible	Boolean, read or write. Format: $("element").visible(true/false);
show	Format: $("element").show();
hide	Format: $("element").hide();
prop	String. Example: $("element").prop("title","changed");

Events

OnDisable, OnEnable, OnHide, OnShow and OnTap

Methods

OnTap	Used to active the script behind the button

Checkboxes

Properties

val	Boolean. The value contained within the field, true/false. Read only on the App. Format: $("element").val(true/false); or var x = $("element").val();
enabled	Boolean, read or write. Format: $("element").enabled(true/false);
enable	Format: $("element").enable();
disable	Format: $("element").disable();
valid	Boolean. Format: this.valid(true/false,"message");
warn	String. Format: this.warn("message");
visible	Boolean, read or write. Format: $("element").visible(true/false);
show	Format: $("element").show();
hide	Format: $("element").hide();
prop	String. Example: $("element").prop("value", true);

Events

OnValidate, OnFocus, OnBlur, OnValueChange, OnEnable, OnDisable, OnShow and OnHide

Methods

focus	Sets focus to a specified field. Format: $("element").focus();
fire	Fires a given element's event. Format: $("element").fire("OnValidate");
validate	Web Only. Fires an element's OnValidate event. Format: $("element").validate();

Date elements

Properties

val	String. The value contained within the field.
enabled	Boolean, read or write. Format: $("element").enabled(true/false);
enable	Format: $("element").enable();
disable	Format: $("element").disable();
valid	Boolean. Format: this.valid(true/false,"message");
warn	String. Format: this.warn("message");
visible	Boolean, read or write. Format: $("element").visible(true/false);
show	Format: $("element").show();
hide	Format: $("element").hide();
prop	String. Example: $("element").prop("title", "New Title"); The following properties are available: days, minimumDate, mimimumDateDays, maximumDate, maximumDateDays, seconds and weekday

Events

OnBlur, OnDisable, OnEnable, OnFocus, OnHide, OnShow, OnValidate and OnValueChange

Methods

focus	Sets focus to a specified field. Format: $("element").focus();
fire	Fires a given element's event. Format: $("element").fire("OnValidate");
validate	Web Only. Fires an element's OnValidate event. Format: $("element").validate();

Functions

today	Sets value of Date field to today's date. Format App: $("element").val(today()); Format Webforms: $("element").val(today);

Forms

Events – in order of occurrence.

For details see "Form Specific Events" above.
OnStart, OnOpen, OnClose, OnValidate, OnSave, OnSubmitAndConfirm and Globals

Methods - General

gotoPage	Changes the page within the form. Format on Webforms: $.goToPage("Page2"/int); Format on App: goToPage("Page2"/int);
save	Webforms only. Saves the form. Format: $.save();
submit	Validates the form and shows the Submission window. Webforms Format: $.submit(); App Format: submit();
App Only:	
closeForm()	Closes the form. Saves form by default. Format: closeForm();
openFormWithName	Can be used to automatically open another form when current one closes. Format: openFormWithName("template name");
deleteForm	Deletes the form from the device, if permitted by form template settings. Format: deleteForm();
validate	Web Only. Fires an element's OnValidate event. Format: $("element").validate();

Methods – Audio (App Only)

startRecording	Toggles between recording and pausing. First time starts recording, second time pauses and third time resumes recording. Format: startRecording();
pauseRecording	Pauses an active recording. Note: Formworks XD App will toggles between recording and pausing. First time pauses recording, second time starts and third time pauses again. Format: pauseRecording();
stopRecording	Stop an active recording. Can also set the filename within parentheses. Format: stopRecording(); or stopRecording("recording1");
recordingStatus	Returns the state of an active or inactive recording. Format: $("status").val(recordingStatus());

Related functions

userEmail()	Gets the users name as set on the Formworks portal. App Format: userEmail(). Webforms Format: $.userEmail();
userName()	Gets the users name as set on the Formworks portal. App Format: userName(). Webforms Format: $.userName();
location()	Gets the users current location (lat & long). Format: location();. Webforms Format: $.location()
App Only:	
setUserData("Key",value)	
getUserData("Key")	
isIphone()	

Groups and Sections

Properties

enabled	Boolean, read or write. Format: `$("element").enabled(true/false);`
enable	Format: `$("element").enable();`
disable	Format: `$("element").disable();`
valid	Boolean. Format: `this.valid(true/false,"message");`
warn	String. Format: `this.warn("message");`
visible	Boolean, read or write. Format: `$("element").visible(true/false);`
show	Format: `$("element").show();`
hide	Format: `$("element").hide();`
prop	String. Example: `$("element").prop("title");` **Note:** The title property of these container elements are read only on the App.

Events

OnDisable, OnEnable, OnHide, OnShow and OnValidate

Methods

focus	Sets focus to a specified field. Format: `$("element").focus();`
fire	Fires a given element's event. Format: `$("element").fire("OnValidate");`
validate	Web Only. Fires an element's OnValidate event. Format: `$("element").validate();`

Images

Properties

visible	Boolean, read or write. Format: `$("element").visible(true/false);`
show	Format: `$("element").show();`
hide	Format: `$("element").hide();`

Events

OnHide and OnShow

Labels

Properties

visible	Boolean, read or write. Format: `$("element").visible(true/false);`
show	Format: `$("element").show();`
hide	Format: `$("element").hide();`
prop	String. Example: `$("element").prop("color", "green");` to set the text colour, use `prop("textColor");`

Events

OnHide and OnShow

Line

Properties

visible	Boolean, read or write. Format: `$("element").visible(true/false);`
show	Format: `$("element").show();`
hide	Format: `$("element").hide();`

Events

OnHide and OnShow

Multi-Select

Properties

val	String. The value contained within the field.
enabled	Boolean, read or write. Format: `$("element").enabled(true/false);`
enable	Format: `$("element").enable();`
disable	Format: `$("element").disable();`
valid	Boolean. Format: `this.valid(true/false,"message");`
warn	String. Format: `this.warn("message");`
visible	Boolean, read or write. Format: `$("element").visible(true/false);`
show	Format: `$("element").show();`
hide	Format: `$("element").hide();`
prop	String. Example: `$("element").prop("title", "New Title");`

Events

OnBlur, OnDisable, OnEnable, OnFocus, OnHide, OnShow, OnValidate and OnValueChange

Methods

focus	Sets focus to a specified field. Format: `$("element").focus();`
fire	Fires a given element's event. Format: `$("element").fire("OnValidate");`
validate	Web Only. Fires an element's OnValidate event. Format: `$("element").validate();`
countSelected()	Return the number of items selected. Format `var selected = $("multi").countSelected();`

getOption()	Returns true or false, depending on whether the option specified within the brackets has been selected. Currently only working if stringify method is used. Format:
	`var mValue = $("multiSelect").getOption("Option1");`
	`var jSONString = JSON.stringify(mValue);`
	`$("Result").val(jSONString);` would return true if "Option1" had been selected

hasChosen(Option)	Webforms only. Will return true or false, similar to getOption. Format: `var other = this.hasChosen("Other");`

Pages

Properties

enabled	Boolean, read or write. Format: `$("element").enabled(true/false);`
enable	Format: `$("element").enable();`
disable	Format: `$("element").disable();`
valid	Boolean. Format: `this.valid(true/false,"message");`
warn	String. Format: `this.warn("message");`
visible	Boolean, read or write. Format: `$("element").visible(true/false);` • Hidden pages do not show on apps Page List. • Hidden pages are skipped over when using apps Page List. • You cannot set the visibility for all pages to hidden in script. But this is possible in the Template designer. • If the current page is set to hidden, the next visible page is displayed. • If the last visible page is set to hidden, the previous visible page is displayed.
show	Format: `$("element").show();`
hide	Format: `$("element").hide();`
prop	String. Example: `$("element").prop("title", "New Title");` **Note:** The title property of the page container element is read only except in the Formworks XD App.

Events

OnDisable, OnEnable and OnValidate

Methods

fire	Fires a given element's event. Format: `$("element").fire("OnValidate");`
validate	Web Only. Fires an element's OnValidate event. Format: `$("element").validate();`

Photos, Signatures and Sketches

Properties

val	String. The value contained within the field.
enabled	Boolean, read or write. Format: $("element").enabled(true/false);
enable	Format: $("element").enable();
disable	Format: $("element").disable();
valid	Boolean. Format: this.valid(true/false,"message");
warn	String. Format: this.warn("message");
visible	Boolean, read or write. Format: $("element").visible(true/false);
show	Format: $("element").show();
hide	Format: $("element").hide();
prop	String. Example: $("element").prop("title", "New Title");

Events

OnDisable, OnEnable, OnFocus, OnHide, OnShow, OnValidate and OnValueChange

Methods

focus	Sets focus to a specified field. Format: $("element").focus();
fire	Fires a given element's event. Format: $("element").fire("OnValidate");
validate	Web Only. Fires an element's OnValidate event. Format: $("element").validate();
clear() $("Photo").clear();	The clear method removes the contents of element. i.e., Note: Sketch fields can only be cleared within script in the Formworks XD App or Webforms.

Single Selects

Properties

val	String. The value contained within the field. This can also be used to clear the selected value, i.e. $("element").val(""); or $("element").val("No");
enabled	Boolean, read or write. Format: $("element").enabled(true/false);
enable	Format: $("element").enable();
disable	Format: $("element").disable();
valid	Boolean. Format: this.valid(true/false,"message");
warn	String. Format: this.warn("message");
visible	Boolean, read or write. Format: $("element").visible(true/false);
show	Format: $("element").show();
hide	Format: $("element").hide();
prop	String. Example: $("element").prop("title", "New Title");

Events

OnBlur, OnDisable, OnEnable, OnFocus, OnHide, OnShow, OnValidate and OnValueChange

Methods

addOption	Adds a new option to the end of the list. Dropdowns only. Format: $("element").addOption("New option");
clear	Clear all options. Dropdowns only. Format: $("element").clear(); Use $("element").val("") to clear selection.
focus	Sets focus to a specified field. Format: $("element").focus();
fire	Fires a given element's event. Format: $("element").fire("OnValidate");
validate	Web Only. Fires an element's OnValidate event. Format: $("element").validate();
removeOption	Remove the option with the given value. Dropdowns only. Format: $("element").removeOptions();
setDatabaseConfiguration()	Formworks XD App only. Populates a Dropdown single selection field from a database. Format: $("County").setDatabaseConfiguration("databaseName", "FilterByColoumnHeader", "[keyValue/columnHeader]", "displayValue/coloumnHeader");

Table

Filter Table Function – App Only

$("TableName/Alias").filterTable(Column, "title/value", Operator, Value, hidden Rows)

Parameter	Description
Column	Number representing the column to be used in the filter operation. Columns are based on 1 as the first column.
Title/Value	String representing the basis of the filter. Its value can be either "title" or "value". If you select "title", the value within the element is ignored and only the title property of the element is evaluated. If you select "value", the value content property of the element is evaluated.
Operator	String representing the operator used to filter the table "==" ">" "<" ">=" "<=" "contains" "doesNotContain"
Value	String/decimal.
Hidden Rows	String choosing whether hidden rows are included in the filter.

General Table Functions

$("Table").enable(); or $("Table").disable();	Sets / reads enabled status.
$("Table").show(); or $("Table").hide();	Sets / reads visibility status.
$("Table").repeatRow()	Will repeat first row in table
$("Table").removeRows(int)	Will remove any rows more than the int

The below is available on Webforms only:

$("Table").col(int).hide(); and $("Table").col(int).show();	Hide / Reveal columns.
$("Table").col(int).visible(true/false);	Sets column visibility.
$("Table").row(int).visible = true/false;	Sets row visibility.

The below is available on the App only:

$("Table1").hideColumn("string/int") and $("Table1").showColumn("string/int")	Hide / Reveal columns.
$("Table1").hideRow("string/int") and $("Table1").showRow("string/int")	Hide / Reveal rows.
$("Table1").enableColumn("string/int") and $("Table1").disableColumn("string/int")	Enable / Disable columns.
$("Table1").enableRow("string/int") and $("Table1").disableRow("string/int")	Enable / Disable rows.
$("Table1").hideColumnHeaders(); and $("Table1").showColumnHeaders();	Hides / Reveals Headers.
$("Table1").hideBorders(); and $("Table1").showBorders();	Hides / Reveals grid lines.
var rows = $("Table1").rowCount();	Returns the table row count

Note: when returning the rowCount value to a text field, the toString() function must be used. Eg. $("textField").val(rows.toString());

Text Boxes and Paragraph elements

Properties

val	String. The value contained within the field.
enabled	Boolean, read or write. Format: $("element").enabled() or .enabled(true/false);
enable	Format: $("element").enable();
disable	Format: $("element").disable();
valid	Boolean. Format: this.valid(true/false,"message");
warn	String. Format: this.warn("message");
visible	Boolean, read or write. Format: $("element").visible(true/false);
show	Format: $("element").show();
hide	Format: $("element").hide();
prop	String. Example: $("element").prop("color", "red"); Note: The color property can only be updated for Paragraph fields within script in the Formworks XD App

Events

OnBlur, OnDisable, OnEnable, OnFocus, OnHide, OnShow, OnValidate and OnValueChange

Methods

focus	Sets focus to a specified field. Format: $("element").focus();
fire	Fires a given element's event. Format: $("element").fire("OnValidate");
validate	Web Only. Fires an element's OnValidate event. Format: $("element").validate();

Time elements

Properties

val	String. The value contained within the field.
enabled	Boolean, read or write. Format: `$("element").enabled()` or `.enabled(true/false);`
enable	Format: `$("element").enable();`
disable	Format: `$("element").disable();`
valid	Boolean. Format: `this.valid(true/false,"message");`
warn	String. Format: `this.warn("message");`
visible	Boolean, read or write. Format: `$("element").visible(true/false);`
show	Format: `$("element").show();`
hide	Format: `$("element").hide();`
prop	String. Example: `$("element").prop("title", "New Title");`

Events

OnBlur, OnDisable, OnEnable, OnFocus, OnHide, OnShow, OnValidate and OnValueChange

Methods

focus	Sets focus to a specified field. Format: `$("element").focus();`
fire	Fires a given element's event. Format: `$("element").fire("OnValidate");`
validate	Web Only. Fires an element's OnValidate event. Format: `$("element").validate();`
subtractTime()	Subtracting time elements: `$("resultInMinutes").val($("endTime").subtractTime($("startTime").prop("minutes")).toString());` Normally the first time element, Time1, will be lower than the second, Time2. For example, start and end times for work on site might be 09:30 until 11:30. In which case the function will return 120.
seconds()	App only. Can be used to compare or add times together. Format `prop("seconds");`

```
var time1 = $("timeSpent1").prop("seconds");
var time2 = $("timeSpent2").prop("seconds");

var total = (Number(time1) + Number(time2))/60;
```

Appendix II

Regular Expression Types

.	all characters
\d	digits
\D	non-digits
\s	space symbols, tabs, newlines
\S	all but \s
\w	latin letters, digits, underscore '_'
\W	all but \w
g	finds all matching ie. /\d/g matches all digits
^	matches beginning of input
$	matches end of input