Formworks XD Lua Scripting Guide

Contents

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 into a form.

Formworks provides an object model that will be familiar to developers who have used Visual Studio or similar object oriented programming environments.

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. When checking if the date entered by the user is within a specific range, Formworks has its own unique date validation functions that are covered later in this guide.

Lua

Scripts are written in a common scripting language called Lua. The version of Lua implemented within Formworks is 5.1.4. For more information on Lua, visit the Lua website at http://www.lua.org/. Full documentation on the Lua language can be found here: http://www.lua.org/manual/5.2/.

Limitations of Lua on iOS Devices

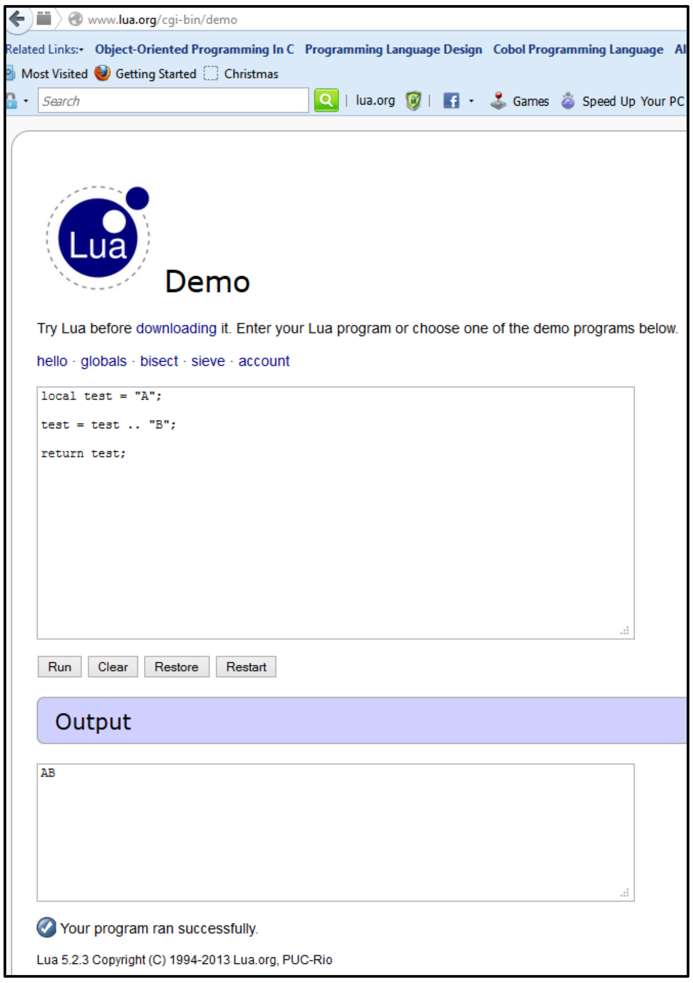
Some Lua functions are not available on iOS. These are the "file:" functions (e.g. file:close, file:flush etc.) and the "io" functions (e.g. io.close, io.flush etc.).

Testing and debugging your Lua scripts

Whilst the Formworks application includes simple debugging facilities, you may wish to test more complex scripts prior to employing them within a template, or simply experiment with code. You can do this very simply at the Lua demo web site:

http://www.lua.org/cgi-bin/demo

Using this site, you can enter Lua code in the first panel, and select the Run option to confirm that the code runs successfully. If you wish to see the value returned by your code, use the "return" instruction, as in the screen capture.


www.lua.org/cgi-bin/demo
Related Links: Object-Oriented Programming In C Programming Language Design Cobol Programming Language
Most Visited Getting Started Christmas
Search
lua.org
Games Speed Up Your PC
Lua
Demo
hello · globals · bisect · sieve · account
Run Clear Restore Restart
Output

Try Lua before downloading it. Enter your Lua program or choose one of the demo programs below.

```
local test = "A";
test = test .. "B";
return test;
```

```
AB
```

Your program ran successfully.

Lua 5.2.3 Copyright (C) 1994-2013 Lua.org, PUC-Rio

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 using method calls when an event occurs.

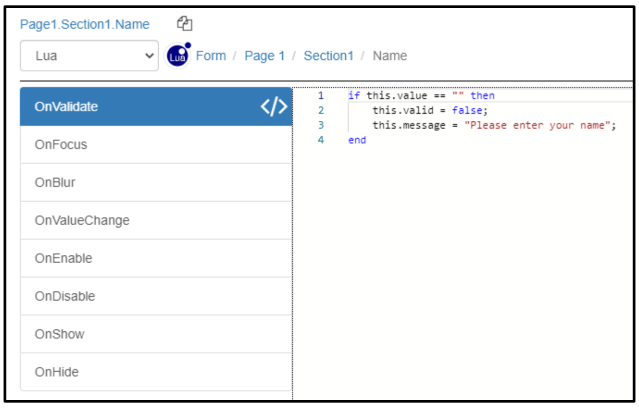
Elements, Events, Properties and Methods

The objects such as a form's pages, text boxes and radio buttons that you drag 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, 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 textbox.valid = false would set the Valid property of a text box element named textbox to false.



```
if this.value == "" then
    this.valid = false;
    this.message = "Please enter your name";
end
```

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. To do this create a script that uses a method to set the element's Valid property to false if no input has been entered. 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. 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, Page1.Section1.Name 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 'value' represents the element's Value property. The Method call, "this.valid = false" sets the element's Valid property to false, whilst the method call, "this.message = "Please enter your name", is a meaningful message to display to the device user if they fail to complete the field before they try to submit the form. We will now cover elements, properties, methods and events in more detail.

Lua Guidelines – The Basics

Case sensitivity

All Lua instructions are case sensitive. You can create and save Lua scripts in the Form Designer, using the Scripting window. It is possible to save incorrect instructions, such as using capital letters on Lua keywords or missing the "end" instruction on an "if, then, end" code structure, but incorrect instructions will either be ignored, or could cause a fatal error on the device when the user attempts to access the form.

All element names and properties are considered case sensitive when used in script, and must be represented in script exactly as they are in the Form Designer.

Comment Operator (--)

Use two hyphen characters to create a comment in Lua (--). You can comment an entire line, or place a comment at the end of a line of script. See Example 1.

Variable name restrictions

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

Concatenation Operator (..)

To concatenate string values, use two full stops (..). For example:

local var = "Test" .. " Case"; The variable var now contains "Test Case".

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

```
local var = "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 or an element's property, you use a single equals character, for example:

stringVariable = "Test";

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

```
if variable1 == variable2 then... 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.value = "Test"; -- setting a value property
- variable = textBox.value; -- reading an element's value
- textBox.visible = true/false; -- setting an element's property value.

Most elements have a value property, though not all. For example, neither line nor label elements can contain a 'value'.

You read the value of an element, say a text box, by accessing its value property:

Variable = textBox.value.

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

- Fully Qualified Name:
  - Variable = Page1.Section1.Group1.textBox.value;
- Alias property:
  - Variable = getField("AliasName").value;
- Dataname:
  - Variable = getField("DataName").value;

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:

```
local tempField = getField("Page1.Section1.Values/Alias/Dataname");

TargetField.value = tempField.value;
```

User Defined Properties

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. 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:

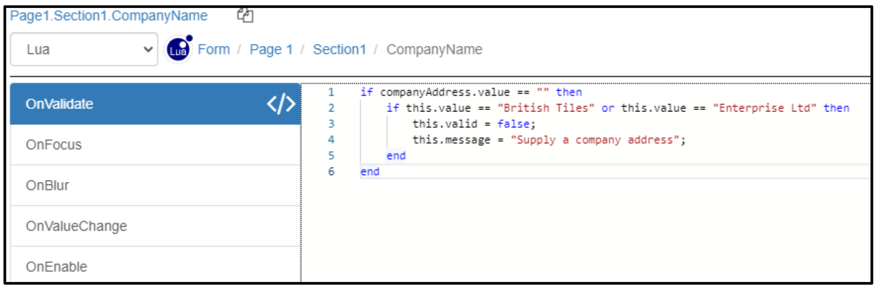
```
textBoxWorkflow.Notification = "Initial Client Contact";

textBoxReference.ClientRef = 12345;
```

Reading the value of a User Defined Property:

```
textBoxOutput.value = textBoxWorkflow.Notification;
```

Code structures

Basic structures such as, "if, then, else, elseif and end" are supported. These must include both the "then" and "end" instructions.


Page1.Section1.CompanyName
Lua
Form / Page 1 / Section1 / CompanyName
OnValidate
OnFocus
OnBlur
OnValueChange
OnEnable

```lua
if companyAddress.value == "" then
    if this.value == "British Tiles" or this.value == "Enterprise Ltd" then
        this.valid = false;
        this.message = "Supply a company address";
    end
end
```

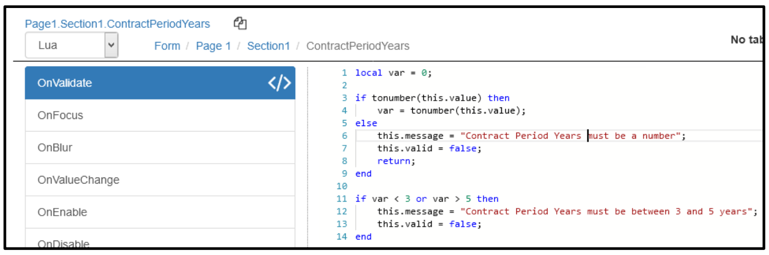
```
if first check then
  first instruction;
elseif second check then
  second instruction;
else
  final instruction;
end
```

"and" and "or" comparison

You can also compare values using the 'and' and 'or' keywords, as in the above example.

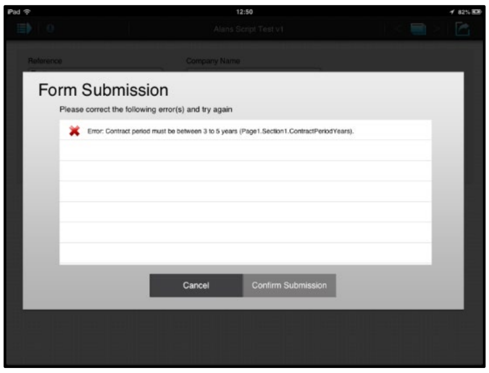
Less than and Greater than comparisons

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.


Page1.Section1.ContractPeriodYears
Lua
Form / Page 1 / Section1 / ContractPeriodYears
No tab
OnValidate
OnFocus
OnBlur
OnValueChange
OnEnable
OnDisable

```lua
local var = 0;

if tonumber(this.value) then
    var = tonumber(this.value);
else
    this.message = "Contract Period Years must be a number";
    this.valid = false;
    return;
end

if var < 3 or var > 5 then
    this.message = "Contract Period Years must be between 3 and 5 years";
    this.valid = false;
end
```

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.



Form Submission

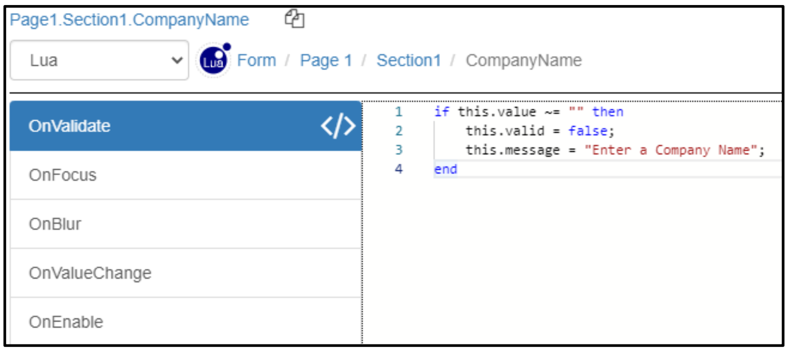
Please correct the following error(s) and try again

Error: Contract period must be between 3 to 5 years (Page1.Section1.ContractPeriodYears).

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

Negation (~=)

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



Page1.Section1.CompanyName

Lua | Form / Page 1 / Section1 / CompanyName

```
if this.value ~= "" then
    this.valid = false;
    this.message = "Enter a Company Name";
end
```

Lua – Advanced use

String Functions

Converting between Strings and Numbers

Lua will automatically convert strings to their numeric value when required, or treat them as strings when you use string functions, such as the concatenate operator.

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 without prior conversion:

```
Var1 = Var1 + 1;
```

The variable Var1 now contains 352.

Usually the automatic conversion between numbers and strings is sufficient, but there are some instances as you will see in the following examples, where you may need to issue explicit conversion instructions. To check if a value can be used as a number, use the tonumber(value) function.

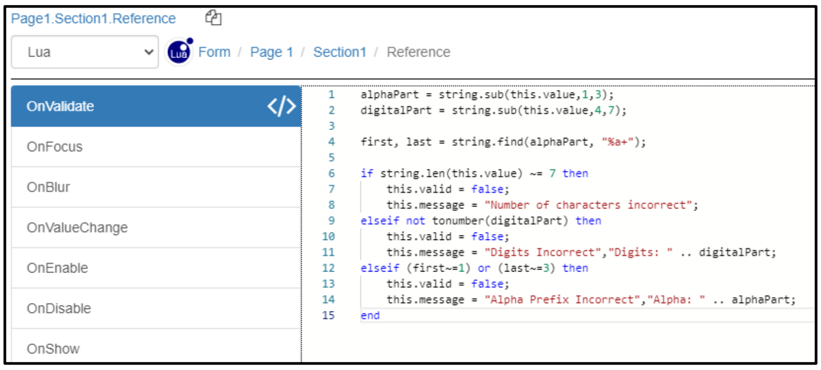
Finding the length of an element's value

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 string.len() function. The format for this function is:

```
varLen = string.len(stringVariable);
```

Validating a range or individual characters in a Field

Lua does not support 'real expressions'. If you need to know that part of a field is alpha and part is numeric, you can use the string.find() function. For clarity, I have also used the string.sub() function to confirm that the Reference field is seven characters long and contains three alpha characters, followed by four digits.



```
alphaPart = string.sub(this.value,1,3);
digitalPart = string.sub(this.value,4,7);

first, last = string.find(alphaPart, "%a+");

if string.len(this.value) ~= 7 then
    this.valid = false;
    this.message = "Number of characters incorrect";
elseif not tonumber(digitalPart) then
    this.valid = false;
    this.message = "Digits Incorrect","Digits: " .. digitalPart;
elseif (first~=1) or (last~=3) then
    this.valid = false;
    this.message = "Alpha Prefix Incorrect","Alpha: " .. alphaPart;
end
```

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 string.sub() function. Indexing in Lua is based on the first character equalling 1, unlike many languages that assume the first character is in the 0 position.

The first line instructs Lua to, "read the characters in the Reference field, from positions 1 to 3 and store them in the local alphaPart variable. The second line of the function stores the characters from positions 4 to 7 in the digitalPart variable.

The string.find() function returns two values – 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 "%a+" section of the function. "%a" is a character class that indicates all lower case characters, and the "+" character indicates that we are searching for more than one character. Appendix IV at the end of this guide contains all the supported character classes. Using these classes you can search for either upper or lower case 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:

```
elseif not tonumber(digitalPart)
```

Replacing characters in a field

```
string.gsub(elementName.value, "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:

```
Page2.Item1.Q.value = anotherElement.value;
```

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

```
Q1.value = anotherElement.value;
```

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, Page2.Item1.Q.value.

```
local counter = 1;
Page2["Item" .. counter]["Q"].value = anotherElement.value;
```

You must use fully qualified names and enclose each part of the element naming path in square braces. Use standard Lua string concatenation to place the value of the variable within the 'string' of the element name that you are referring to.

Looping structures in Lua

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 loop in for Lua is:

```
for index = startNumber, endNumber, stepsIncrement do
    actions;
end
```

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 optional, and can be positive (1,2,3 etc) or negative (-2 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/end loop.

```
for i=1,10 do
  if Page1["Item" .. i]["Quantity"].value == "" then
    counter = i;
    break;
  end
end
```

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

```
if Page1.Item1.Quantity.value == "" then
  some action
end
if Page1.item2.Quantity.value== "" then
  some action
end
if Page1.item3.Quantity.value== "" then
  some action
end
```

etc.. Repeated 10 times.

Handling Dates

The Lua today() function returns a string of today's date. For example:

```
stringVariable = today(); -- stringVariable contains "14/01/2013", or to place today's date into a date element:

elementName.value = today();
```

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

```
elementName.value = os.date("%H:%M");
```

A full list of possible os.date return values is included below:

Alternatively, the os.time() function returns the current Operating System date as a number of seconds. Passing parameters to the os.time() function, returns a number of seconds representing the date passed to it. For example:

```
Start = os.time{year=YearValue, month=MonthValue, day=DayValue, hour=0};
```

Using the os.time() function, either with or without parameters, allows you to perform date arithmetic against the current date, or in fact any two dates. This is useful when you want to ensure that one date is before another. For example, the following code finds the seconds value for today's date and the seconds value for the Date of Birth date field, then compares them. If the Date of Birth date is greater than today, validation is set to false.

```
todayNumber = os.time{year=string.sub(today(),7,10), month=string.sub(today(),4,5),
day=string.sub(today(),1,2)};
dobNumber = os.time{year=string.sub(this.value,1,4), month=string.sub(this.value,6,7),
day=string.sub(this.value,9,10)};

if dobNumber == nil then
  this.valid = false;
  this.message = "Invalid Date of Birth";
elseif dobNumber >= todayNumber then
  this.valid = false;
  this.message = "Date of Birth must be in past";
end
```

This is only an example of Lua date arithmetic. Formworks has its own functions which handle the above scenario much simpler. For example, to ensure that the value of a date of birth field is in the past, place the following script in the fields OnValidate event:

```
this.miniumumDate = "today";
```

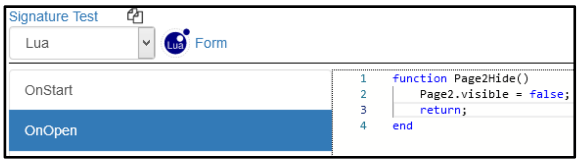
The next section deals with the built in date functions included with the latest version of Formworks.

For further information about handling dates and times in Lua, refer to the Date and Time Functions chapter of this guide.

Lua – Global Functions

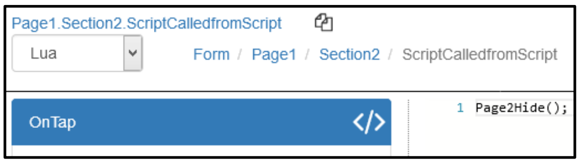
Introduction

Lua and Formworks support Global Functions. These can be called from any place in a template and can either perform simple actions that require no parameters, or quite complex actions that require one or more parameters being passed to the function.



```
function Page2Hide()
    Page2.visible = false;
    return;
end
```

The function should be placed in the template's OnOpen event. But Global Functions do not 'run' when this event runs, only when they are specifically called. A simple example of a Lua function would be to hide a template page. The script has been placed in the templates OnOpen event and can be called from anywhere, though in the example, a button is used.

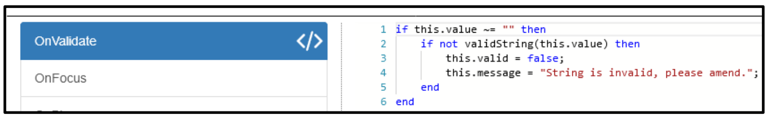

Page1.Section2.ScriptCalledfromScript
Lua
Form / Page1 / Section2 / ScriptCalledfromScript
OnTap

```
Page2Hide();
```

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

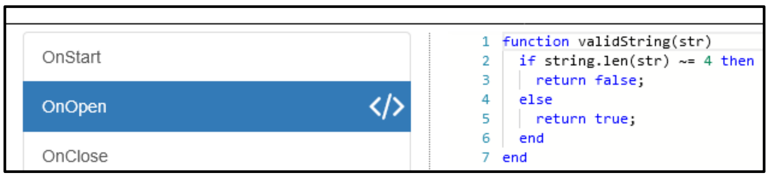
Returning a value

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.



```
1 if this.value ~= "" then
2     if not validString(this.value) then
3         this.valid = false;
4         this.message = "String is invalid, please amend.";
5     end
6 end
```

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



```
function validString(str)
  if string.len(str) ~= 4 then
    return false;
  else
    return true;
  end
end
```

Formworks Date Validation Properties and Functions

There are eight built in date properties in Formworks, four date functions, two time properties and one time function. These are designed to simplify working with dates and provide a convenient alternative to the preceding chapter. The usual method of comparing date values would entail using Lua functions to convert the dates to seconds, then compare the seconds value. The days property returns the number of days since 1st January 1900 (previous dates are returned as negative values). This makes comparing and setting date elements much easier.

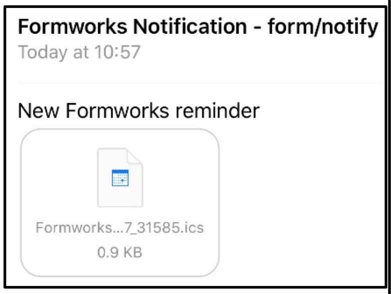
In addition, it is possible to set both the maximum and minimum acceptable dates for a date field. These values can be set either as a number, or using any of the standard date formats below. Most of these properties are both read and write, other than those marked as read only.

Date Properties

- dateElement.value
- dateElement.days
- dateElement.minimumDate
- dateElement.minimumDateDays
- dateElement.maximumDate
- dateElement.maximumDateDays
- dateElement.seconds (read only)
- dateElement.weekday (read only)

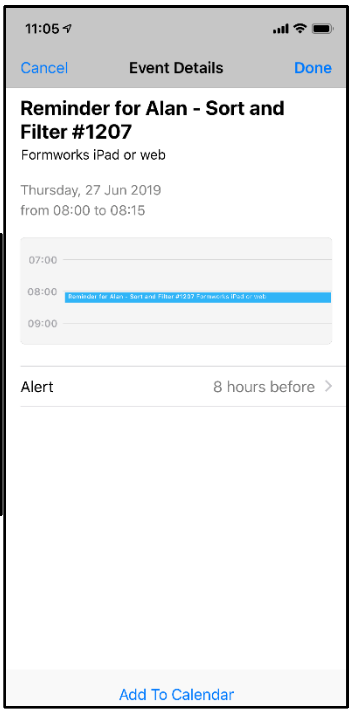
Date Functions

- dateElement1.equalTo(dateElement2.value)
- FirstDate.lessThan(SecondDate.value)
- dateElement1.greaterThan(dateElement2.value)
- dateElement.age()
- date.notification(this.value, "This is your reminder ", "-PT8H");
    - This function is somewhat different to the other date functions. The notification function generates an email upon submission of the form, containing a reminder, .ics attachment. To add the reminder to a calendar, you open the ics attachment and select the Add To Calendar button. At present, this function is configured to automatically set meeting durations to 15 minutes.


Formworks Notification - form/notify
Today at 10:57

New Formworks reminder

Formworks...7_31585.ics
0.9 KB


11:05

Cancel Event Details Done

Reminder for Alan - Sort and Filter #1207
Formworks iPad or web

Thursday, 27 Jun 2019
from 08:00 to 08:15

07:00
08:00
09:00

Alert 8 hours before

Add To Calendar

Time Properties

- timeElement.value;
- timeElemement.seconds;

Time Functions

- Time2.subtractTime(Time1.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

dateElement.value = "07/08/18" Quotes required.

dateElement.days = 17110; Quotes optional.

dateElement.minimumDate = "today"; Quotes required.

dateElement.minimumDate = "2016/12/01"; or "14/12/2016" etc. Common date formats accepted.

dateElement.minimumDate = "formStartDate"; Quotes required.

dateElement.minimumDate = "formStartDate+20"; Quotes required.

dateElement.minimumDateDays = "17110" Quotes required.

dateElement.minimumDateDays = tostring(secondDateElement.days + 5);

dateElement.maximumDate = "today-5480" for example, to ensure a date of birth is 16 years ago, or "18/10/2018" etc.

dateElement.maximumDateDays = 43953; for example would set the maximum date selectable to 4th May 2020.

dateElement.seconds can only be read in script, not set (read only).

Reading Date Properties

nVar = dateElement.days; nVar could contain say 46201 – a numeric value.

nVar = dateElement.maximumDate nVar would again contain a number.

strVar = dateElement.maximumDate strVar would contain say "today", or "today+5" etc.
The point being that minimumDate and maximumDate returns the string that was originally used to set the property's value, not an actual date. The property would only return a date string, if that was used to set its value, for example if you set the maximumDate, or minimumDate, using an instruction like: dateElement.maximumDate = "11/12/2016". This value would then become the return value.

dateElement.days returns the number of days since 1st January 1900. The Linux epoch date.
dateElement.days = 17800 would set the date element to that date.
nVar = dateElement.seconds. nVar would equal a value such as 1531353600.
nVar = dateElement.weekday. nVar would equal the day number of the week, with 0 being Sunday. Therefore Wednesday would equal 3.
nVar = dateElement.age. nVar equals the number of years between current date and dateElement.

Using Date Functions

boolVar = dateElement1.equalTo(dateElement2.value). Function returns true if two dates are the same.

boolVar = FirstDate.lessThan(SecondDate.value); boolVar returns true if the first date is less than the second date.

boolVar = dateElement1.greaterThan(dateElement2.value); boolVar is true if the first date is greater than the second date.

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: nVar = timeElemement.seconds;

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

Difference.value = Time2.subtractTime(Time1.minutes);

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 Lua 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.

Device User's Name

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

stringVariable = getCurrentUserName();

Device User's Email Address

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

stringVariable = getCurrentUserEmail();

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. This may cause some problems when comparing email addresses in script, due to the case sensitive nature of Lua. As you cannot enforce the case employed when a device user logs into Formworks, you may wish to force the return value of this function to upper or lower case.

Geo-Location Data

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

stringVariable = getCurrentLocation();

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

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, year, hours, minutes and seconds.

```
local initial1 = "";
local initial2 = "";
local user = getCurrentUserName();

initial1 = string.sub(user,1,1);

first, last = string.find(user," ");

if (first > 1 and string.len(user) >= first+1) then
   initial2 = string.sub(user,first+1,first+1);
end

dateString = os.date("%d%m%y%H%M%S");

UniqueReference.value = initial1 .. initial2 .. dateString;
```

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.value = "Job Form " .. JobNumber.value .. "  " ..  VisitDate.value .. "  " .. "{formid}";
```

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

Please note: The unique Formworks ID is only available when the form is exported, it cannot be used in script.

Formworks Alert Boxes, Notification Lists

Interactive alert Boxes

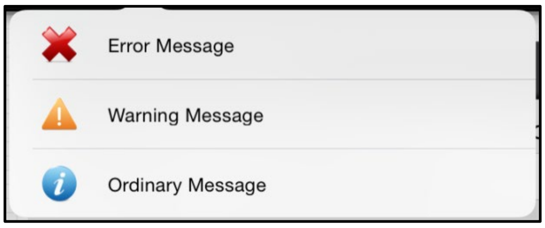
You can generate an alert box to immediately warn a device user that they may have entered an incorrect value. For example, when the device user leaves a field (OnBlur event), an alert box with the heading "Warning Message" and the message content, "Are you certain that the date is correct?" could display, 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.

alert("Are you certain that the date is correct?", "Warning Message");

Please note that this option is case sensitive. If you spell alert with a capital 'A', it will not work.

Notification Lists

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])

warning(message [, autoShowNotificationList])

error(message [, autoShowNotificationList])

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). The different methods each display the appropriate type of icon next to the message (e.g. blue circle for notification, yellow triangle for warning, red stop sign for error, etc.). 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:

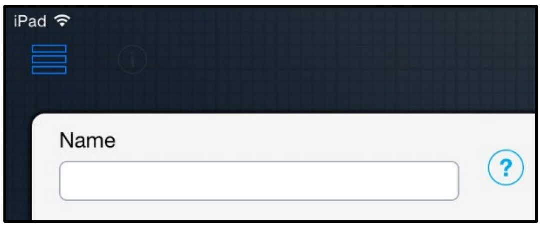
notify("Ordinary Message",1); -- Places text Ordinary Message into the notify list and displays list.

notify("Ordinary Message"); -- 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

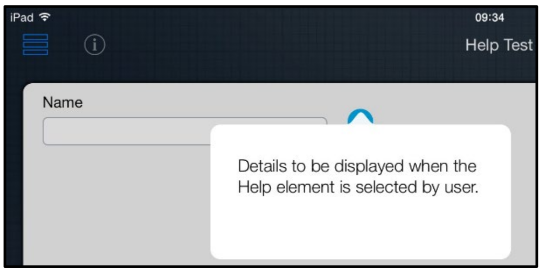
Help Element



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.value = "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.



Details to be displayed when the Help element is selected by 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 .add method. For example:

```
listBox.add("Option 1");
```

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

```
listBox.add("JBH","John B Henry");
```

Table Element

Filter Table Function

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)

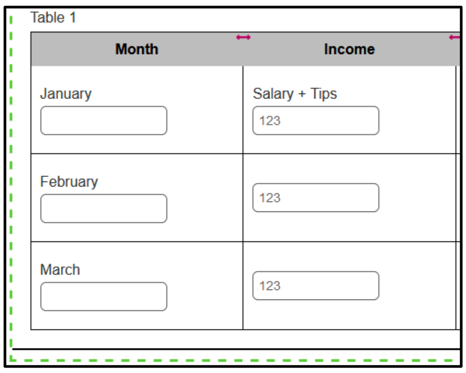
Parameter	Description
Column	Number representing the column to be used in the filter operation. Columns are based on 1 as the first column.
Title/Value	String representing the basis of the filter. Its value can be either "title" or "value". If you select "title", the value within the element is ignored 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

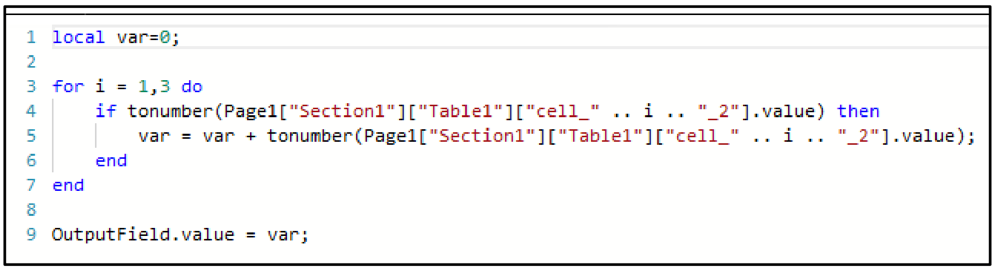
Table.enabled = true/false;	Sets / reads enabled status.
Table.hideColumn(string/int) and Table.showColumn(string/int)	Hide / Reveal columns.
Table.hideRow(string/int) and Table.showRow(string/int)	Hide / Reveal rows.
Table.enableColumn(string/int) and Table.disableColumn(string/int)	Enable / Disable columns.
Table.enableRow(string/int) and Table.disableRow(string/int)	Enable / Disable rows.
Table.hideColumnHeaders() and Table.showColumnHeaders();	Hides / Reveals Headers
Table.hideBorders() and Table.showBorders()	Hides / Reveals grid lines.
Rows = Table.rowCount()	Returns the table row count
Table.column_3.visible = true/false;	Sets column visibility.

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 text box 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. However, as the individual cells are only identified by their place within the table, the name of the text boxes is irrelevant and they can therefore be named more meaningfully.



```
local var=0;

for i = 1,3 do
    if tonumber(Page1["Section1"]["Table1"]["cell_" .. i .. "_2"].value) then
        var = var + tonumber(Page1["Section1"]["Table1"]["cell_" .. i .. "_2"].value);
    end
end

OutputField.value = var;
```

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

Setting Row Visibility within Loops

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

```
for i = 1,10 do
        Page1["SectionName"]["Table1"]["row_" .. i].visible =
        Page1["SectionName"]["Table1"]["cell_1_" .. i].value ~= "";
end
```

Methods of Accessing Individual Table Cells

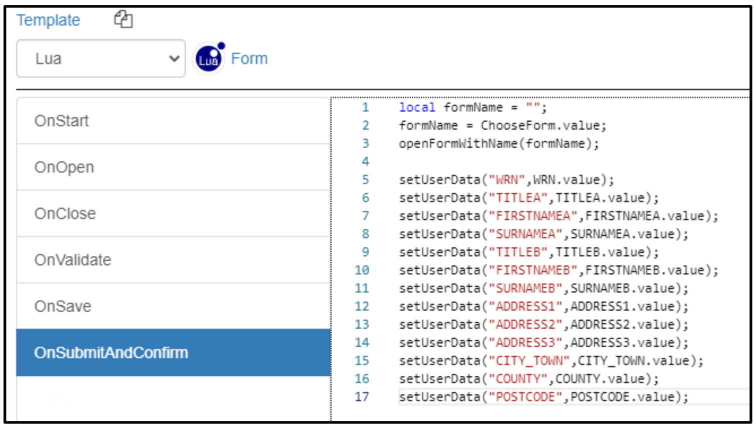
You can refer to the value of the first element in a table cell, simply by referring to the cell itself. As you can see from the above example, the fully qualified name of say, the first cell in the table would be:

```
Page1.Section1.Table1.cell_1_1.value;
```

Forms

Automatically opening a new blank form

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:



```lua
local formName = "";
formName = ChooseForm.value;
openFormWithName(formName);

setUserData("WRN",WRN.value);
setUserData("TITLEA",TITLEA.value);
setUserData("FIRSTNAMEA",FIRSTNAMEA.value);
setUserData("SURNAMEA",SURNAMEA.value);
setUserData("TITLEB",TITLEB.value);
setUserData("FIRSTNAMEB",FIRSTNAMEB.value);
setUserData("SURNAMEB",SURNAMEB.value);
setUserData("ADDRESS1",ADDRESS1.value);
setUserData("ADDRESS2",ADDRESS2.value);
setUserData("ADDRESS3",ADDRESS3.value);
setUserData("CITY_TOWN",CITY_TOWN.value);
setUserData("COUNTY",COUNTY.value);
setUserData("POSTCODE",POSTCODE.value);
```

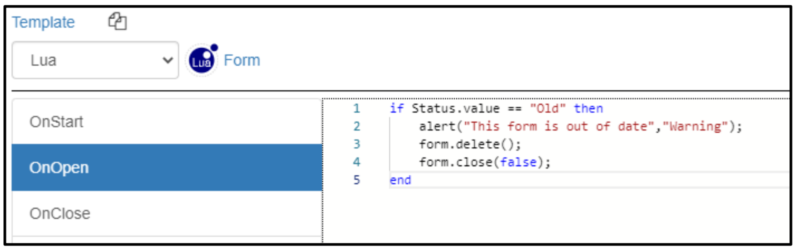
```
openFormWithName("Project Management Schedule");
```

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.

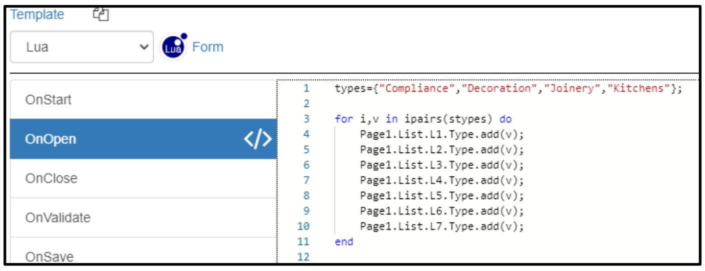
Automatically closing and deleting a form

You can place code in a forms events, for example the OnOpen event, that can test the values of fields and based on the result, force a form to close, with or without saving, and even to delete the form from the iPad. 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, without saving, and is deleted from the device. Changing the false instruction in the form.close instruction to true, will cause the form to be saved prior to closing.



```lua
if Status.value == "Old" then
    alert("This form is out of date","Warning");
    form.delete();
    form.close(false);
end
```

Dynamically populating a Single Selection Dropdown List box

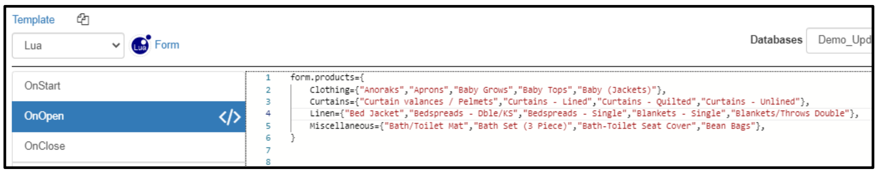


```lua
types={"Compliance","Decoration","Joinery","Kitchens"};

for i,v in ipairs(stypes) do
    Page1.List.L1.Type.add(v);
    Page1.List.L2.Type.add(v);
    Page1.List.L3.Type.add(v);
    Page1.List.L4.Type.add(v);
    Page1.List.L5.Type.add(v);
    Page1.List.L6.Type.add(v);
    Page1.List.L7.Type.add(v);
end
```

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 Lua 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

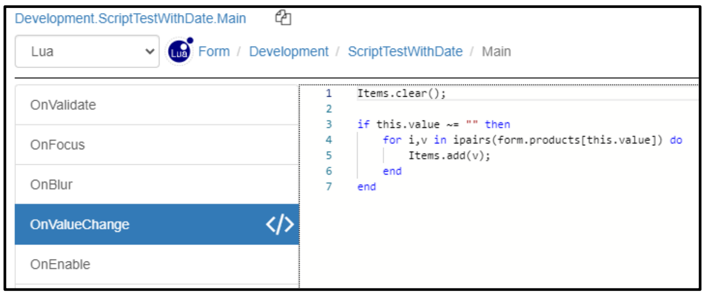


```lua
form.products={
    Clothing={"Anoraks","Aprons","Baby Grows","Baby Tops","Baby (Jackets)"},
    Curtains={"Curtain valances / Pelmets","Curtains - Lined","Curtains - Quilted","Curtains - Unlined"},
    Linen={"Bed Jacket","Bedspreads - Dble/KS","Bedspreads - Single","Blankets - Single","Blankets/Throws Double"},
    Miscellaneous={"Bath/Toilet Mat","Bath Set (3 Piece)","Bath-Toilet Seat Cover","Bean Bags"},
}
```

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

Note: Only single word selections can be used. For example, the term "Clothing and Shoes" would fail.

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


Development.ScriptTestWithDate.Main
Lua
Form / Development / ScriptTestWithDate / Main
OnValidate
OnFocus
OnBlur
OnValueChange
OnEnable

```lua
Items.clear();

if this.value ~= "" then
    for i,v in ipairs(form.products[this.value]) do
        Items.add(v);
    end
end
```

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 this.value ~= "" then", 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

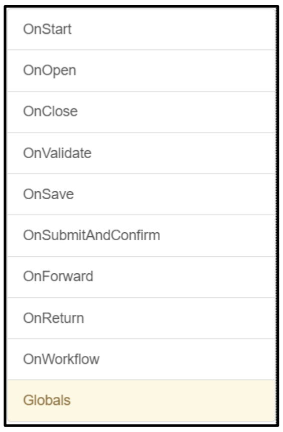
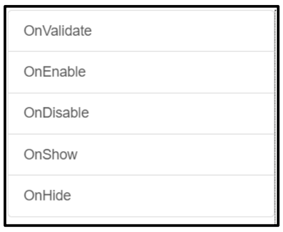
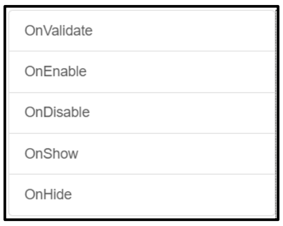
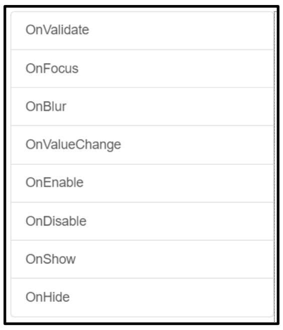
Label	Simple descriptive element. The contents can be exported with other data.
Image	Embedded image that can be displayed to the user.
Button	Provides the option of running script on demand.
Line	Provides a simple separator between elements. Can be used for page breaks in PDF exports.
Help	Places a 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:

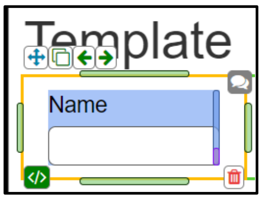
- Select the element on the form
- Select the script '</>' button
- Select the event from the left side of the Script Template window
- Enter your script against the event in the Script Template window

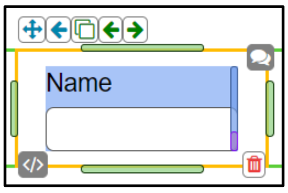
Events

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.

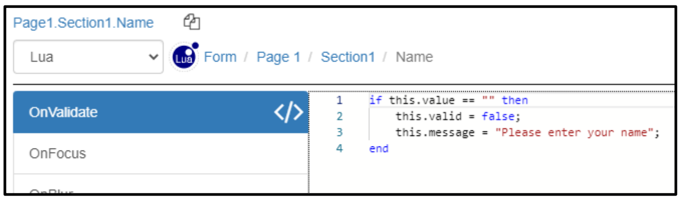
Form	Page	Section	Element
			
OnStart	OnValidate	OnValidate	OnValidate
OnOpen	OnEnable	OnEnable	OnFocus
OnClose	OnDisable	OnDisable	OnBlur
OnValidate	OnShow	OnShow	OnValueChange
OnSave	OnHide	OnHide	OnEnable
OnSubmitAndConfirm			OnDisable
OnForward			OnShow
OnReturn			OnHide
OnWorkflow			
Globals			

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.



Page1.Section1.Name

Lua | Form / Page 1 / Section1 / Name

OnValidate

OnFocus

```
if this.value == "" then
    this.valid = false;
    this.message = "Please enter your name";
end
```

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 form.changePage() method to select the required page. The format for using this is:

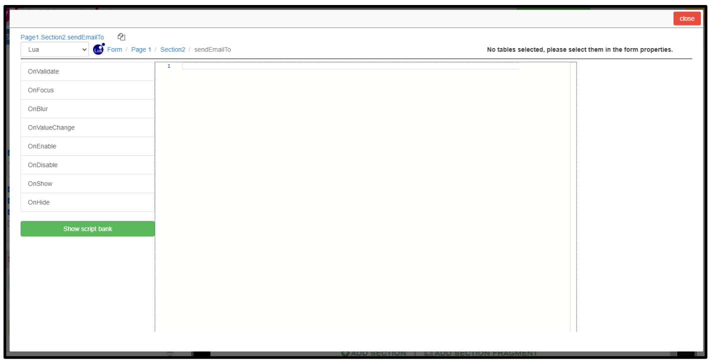
`form.changePage("aliasName")`, if an alias has been allocated to the page, otherwise, use the Page name. This must be enclosed in quotes, as in the example. As always, the instructions are case sensitive.

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

`element.OnValidate();` and `element.focus();`

The Script Template window

Introduction



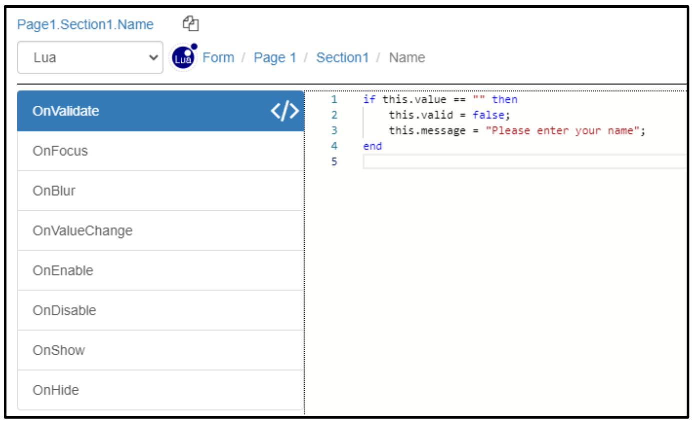
Page1.Section2.sendEmailTo

No tables selected, please select them in the form properties.

Script can be entered against any of an element's events, but the Message 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 commonest 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" is setting the Page1.Section1.Name 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 been set to "Please enter your name".

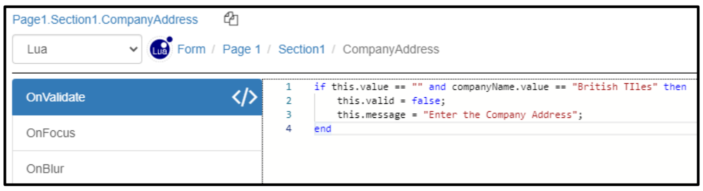


Page1.Section1.Name

```
if this.value == "" then
    this.valid = false;
    this.message = "Please enter your name";
end
```

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 "and" keyword to combine the fields of the query.



Page1.Section1.CompanyAddress

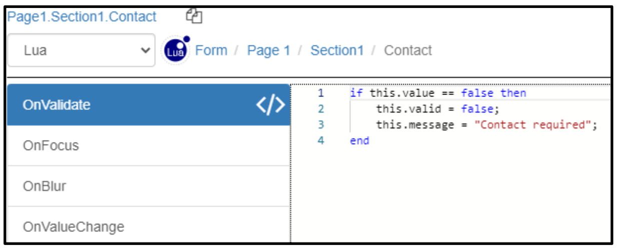
```
if this.value == "" and companyName.value == "British TIles" then
    this.valid = false;
    this.message = "Enter the Company Address";
end
```

Checking for empty elements/fields

You can check if most of the elements have been left empty by testing whether they have a value of an empty string (""). The elements that can be tested this way include:

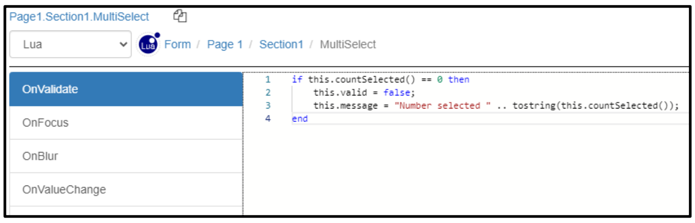
- Text elements (including text, number, email, phone etc).
- Paragraph text
- Date
- Time
- Single-select
- Photo
- Sketch
- Signature

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



```
if this.value == false then
    this.valid = false;
    this.message = "Contact required";
end
```

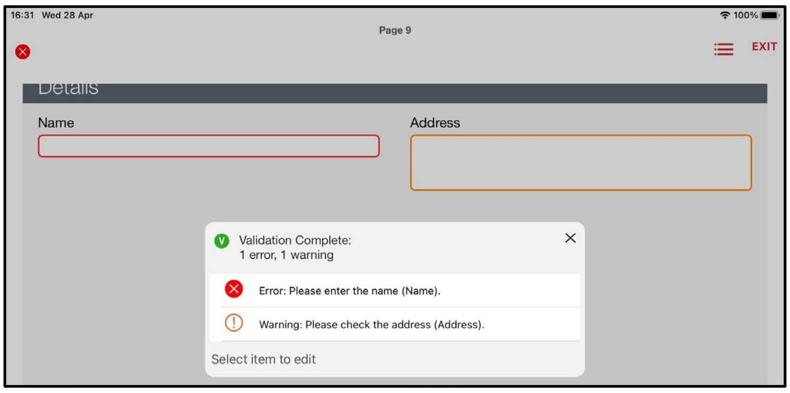
To confirm if a selection has been made from a multiple selection element, you can use the count(), countSelected() and getOption() functions. Count() returns the number of items in a multi-select element, countSelected() returns the number of items selected and getOption().value will return true or false, depending on whether the option specified within the brackets has been selected. For example, this.getOption("Option 1").value would return true if "Option 1" had been selected.


Page1.Section1.MultiSelect
Lua
Form / Page 1 / Section1 / MultiSelect
OnValidate
OnFocus
OnBlur
OnValueChange

```
if this.countSelected() == 0 then
    this.valid = false;
    this.message = "Number selected " .. tostring(this.countSelected());
end
```

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.



Validation Complete:
1 error, 1 warning

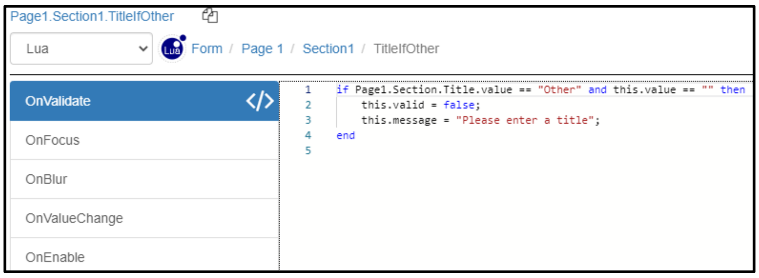
Error: Please enter the name (Name).

Warning: Please check the address (Address).

Select item to edit

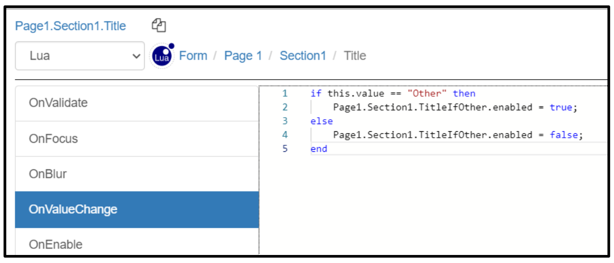
You can set the Message property for an element without setting its Valid property to false. In this way you can warn a user that their input may be incorrect, without preventing them from submitting the form. When the Message property is set in this way, it is indicated on the device by an amber circle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

OnValueChange event


Page1.Section1.TitleIfOther
Lua
Form / Page 1 / Section1 / TitleIfOther
OnValidate
OnFocus
OnBlur
OnValueChange
OnEnable

```
if Page1.Section.Title.value == "Other" and this.value == "" then
    this.valid = false;
    this.message = "Please enter a title";
end
```

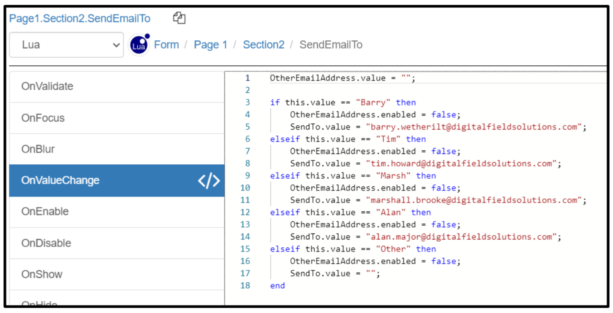
The OnValidate event is used to test data when the device user submits a form. But you may wish to change the value or condition of an element prior to this point – for example when the user enters data in a text box, or selects a value from a drop down list. In this case, you could use the OnValueChange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



```
if this.value == "Other" then
    Page1.Section1.TitleIfOther.enabled = true;
else
    Page1.Section1.TitleIfOther.enabled = false;
end
```

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.

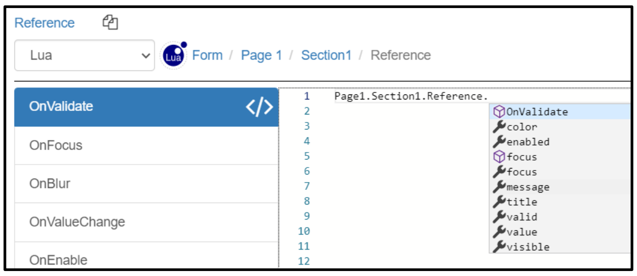

Page1.Section2.SendEmailTo
Lua
Form / Page 1 / Section2 / SendEmailTo
OnValidate
OnFocus
OnBlur
OnValueChange
OnEnable
OnDisable
OnShow

```
OtherEmailAddress.value = "";

if this.value == "Barry" then
    OtherEmailAddress.enabled = false;
    SendTo.value = "barry.wetherilt@digitalfieldsolutions.com";
elseif this.value == "Tim" then
    OtherEmailAddress.enabled = false;
    SendTo.value = "tim.howard@digitalfieldsolutions.com";
elseif this.value == "Marsh" then
    OtherEmailAddress.enabled = false;
    SendTo.value = "marshall.brooke@digitalfieldsolutions.com";
elseif this.value == "Alan" then
    OtherEmailAddress.enabled = false;
    SendTo.value = "alan.major@digitalfieldsolutions.com";
elseif this.value == "Other" then
    OtherEmailAddress.enabled = false;
    SendTo.value = "";
end
```

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 *getCurrentUserEmail()* function, which returns a string value of the logged in user's email address.

Intellisense

Formworks supports rudimentary Intellisense in version 2. Currently this works using either the "this" keyword, or element alias names. 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.



```
Page1.Section1.Reference.
```

```
OnValidate
color
enabled
focus
focus
message
title
valid
value
visible
```

Calculated Fields

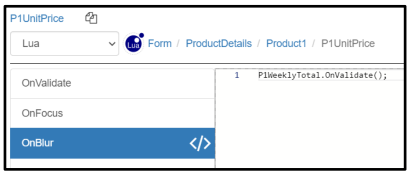
Introduction

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 actually contains a value. Otherwise your script will generate an error. You do this by testing the field's value with the tonumber() function.

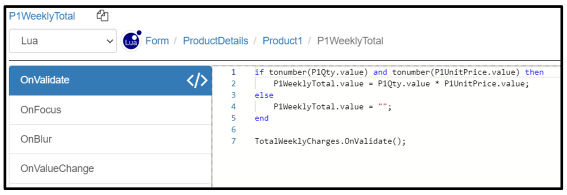
The following example is quite typical. Changing the values within a number of fields, such as quantity and price, can cause the total to automatically update. To achieve this, you place the calculating code in the OnValidate event of the total field, and call the OnValidate event from the OnBlur event 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.



```
P1WeeklyTotal.OnValidate();
```

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.



```
if tonumber(P1Qty.value) and tonumber(P1UnitPrice.value) then
    P1WeeklyTotal.value = P1Qty.value * P1UnitPrice.value;
else
    P1WeeklyTotal.value = "";
end

TotalWeeklyCharges.OnValidate();
```

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

Introduction

Version II of Formworks has introduced the concept of Global Variables. 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.

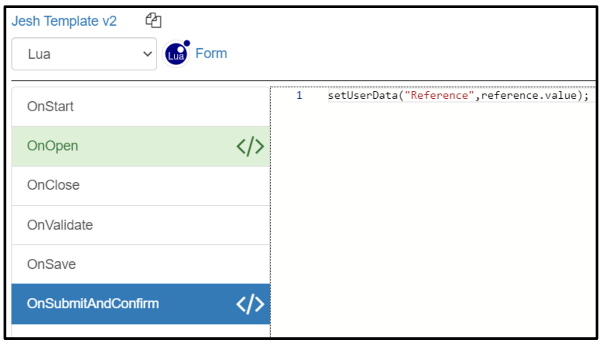
Two methods are employed to work with Global Variables - setUserData and getUserData. The format for their use is:

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

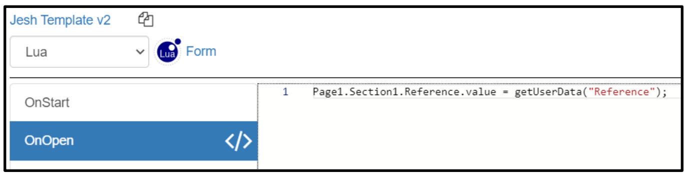
For example, to save the value from a text box with an alias of reference, into a Global Variable named Reference, the two methods would be coded as follows:

```
setUserData("Reference",reference.value);     -- Store the value of the reference element.
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.



```
setUserData("Reference",reference.value);
```



```
Page1.Section1.Reference.value = getUserData("Reference");
```

Custom Properties

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", 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:

```
form.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 OnValueChanged event for each of these check boxes could update the form.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.

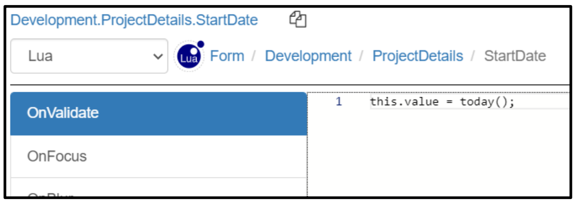
Comparison chart

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

Calculating the days between two dates

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


Development.ProjectDetails.StartDate
Lua
Form / Development / ProjectDetails / StartDate
OnValidate
OnFocus

```
1    this.value = today();
```

this.value = today();

or

this.value = os.date("%H:%M");

Descriptive text can also be included with the os.date function, for example:

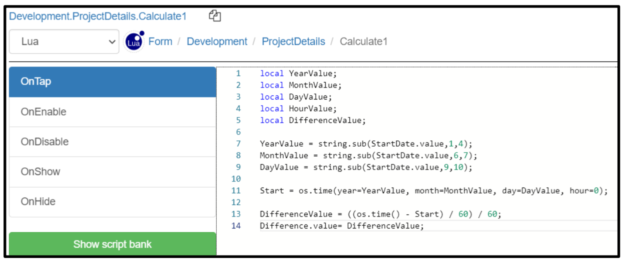
os.date("today is %A, in %B")

this script will return: today is Tuesday, in May

Because this script has been entered against the element's OnValidate event, it will not run until the form is submitted, and therefore, the date will not be visible to the user of the form.

Lua also supports the os.time function. This function can be used both with and without parameters. Without parameters, it returns a number of seconds, representing the current Operating System date and time. The returned value can be stored to a variable, or used in calculations against the current date. When used with parameters, the os.time function can return the number of seconds represented by a passed date/time.

- In the example below, lines 1 to 5 are used to create the variables used in the code that follows.
- Lines 7 to 9 split the date entered in the date element, StartDate into its year, month and date values, using the string.sub() function, storing them in the variables created above.
- Line 11 calls the os.time function, passing the year, month and day values to the os.time function as a Lua table, and stores the seconds value returned in the Start variable. The first three parameters are required. Subsequent parameters, hours etc., are assumed 0 if not supplied.


Development.ProjectDetails.Calculate1
Lua
Form / Development / ProjectDetails / Calculate1
OnTap
OnEnable
OnDisable
OnShow
OnHide
Show script bank

```
1    local YearValue;
2    local MonthValue;
3    local DayValue;
4    local HourValue;
5    local DifferenceValue;
6
7    YearValue = string.sub(StartDate.value,1,4);
8    MonthValue = string.sub(StartDate.value,6,7);
9    DayValue = string.sub(StartDate.value,9,10);
10
11   Start = os.time(year=YearValue, month=MonthValue, day=DayValue, hour=0);
12
13   DifferenceValue = ((os.time() - Start) / 60) / 60;
14   Difference.value= DifferenceValue;
```

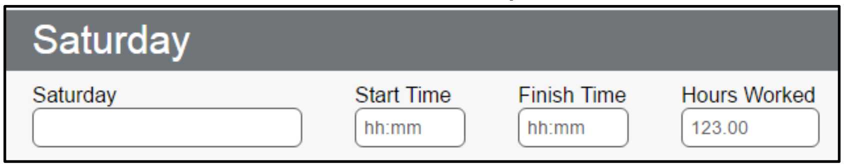
- Line 13 takes the seconds value, calculated from the StartDate date element and subtracts it from the current operating system time. Then divides it by 60, to calculate the minutes difference, then divides it by 60 again, to calculate the hours difference.

- Line 14 places the number of hours calculated in the previous line, and places them in the Difference text box.

Whilst fewer lines could be used to achieve the same result, I have used this example to highlight the individual steps required to calculate the difference between two dates.

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.



Saturday

Saturday	Start Time	Finish Time	Hours Worked
	hh:mm	hh:mm	123.00

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 2 will give you that result. If you need the total in minutes, line 3 will achieve this.

```
1   if string.len(startTime.value) == 5 and string.len(endTime.value) == 5 then
2       resultInHours.value = (endTime.subtractTime(startTime.minutes))/60;
3       resultInMinutes.value = endTime.subtractTime(startTime.minutes);
4   end
```

Operating System Time Function

It is also possible to use the operating system time (os.time) function, with formatting characters, to extract say the current hour and minutes.

Note 1: Presently, date fields that have been completed by hand, return a date string, in the format yyyymmdd. Prepopulated date elements return a number of seconds date representation.

Note 2: Values are written to date elements in the format dd/mm/yyyy, but when read in script, a date element returns its value in the format yyyymmdd.

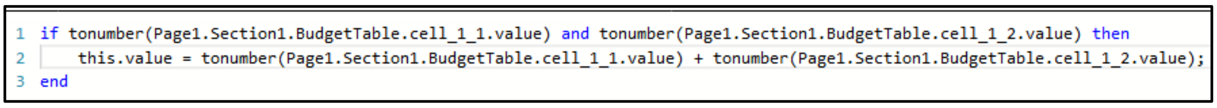
The following is a table of the formatting characters that can be used with the os.time() function.

%a	abbreviated weekday name (e.g., Wed)
%A	full weekday name (e.g., Wednesday)
%b	abbreviated month name (e.g., Sep)
%B	full month name (e.g., September)
%c	date and time (e.g., 09/16/98 23:48:10)
%d	day of the month (16) [01-31]
%H	hour, using a 24-hour clock (23) [00-23]
%I	hour, using a 12-hour clock (11) [01-12]
%M	minute (48) [00-59]
%m	month (09) [01-12]
%p	either "am" or "pm" (pm)
%S	second (10) [00-61]
%w	weekday (3) [0-6 = Sunday-Saturday]
%x	date (e.g., 09/16/98)
%X	time (e.g., 23:48:10)
%Y	full year (1998)
%y	two-digit year (98) [00-99]
%%	the character `%'

Table Elements Scripting

Referencing Table cells in script

You can refer to the value property of an element in a Table cell, without using the elements name, by referencing the value property of the cell in which the element is located. Only one element, the first encountered in the grid cell, can be referenced using this syntax. This method of referencing can only be used for elements that have



```
if tonumber(Page1.Section1.BudgetTable.cell_1_1.value) and tonumber(Page1.Section1.BudgetTable.cell_1_2.value) then
    this.value = tonumber(Page1.Section1.BudgetTable.cell_1_1.value) + tonumber(Page1.Section1.BudgetTable.cell_1_2.value);
end
```

a value property, for example, text boxes.

Looping through Table cells

As you don't need to refer to the elements within a table cell by name, you can create a looping structure very simply. This example uses the third column of a table and loops through the first three rows, placing the value 10 in an element.

```
for i = 1,3 do
  Page1["Audit"]["MajorRisks"]["cell_3_" .. i].value = "10";
end
```

Databases

Introduction

Creating Lua tables within the form template has the limitation that the table items, which normally contain the contents of a down list box, are hard coded into the template. This means that should an item need deleting or replacing, the template would need to be duplicated, amended, published, and the original template eventually retired.

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

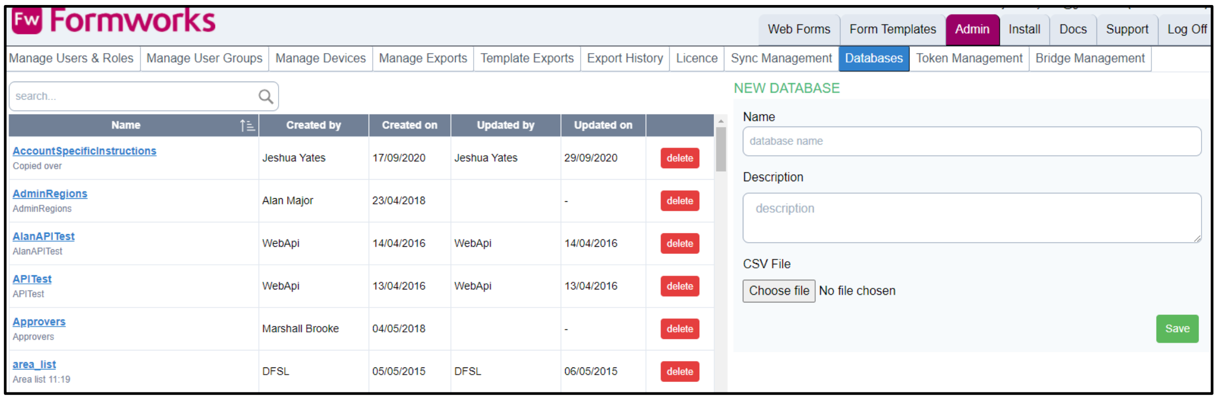
- Providing a look-up database of cost values for products selected on a form. Databases can contain all the product cost information necessary to provide up to date quotes as a form is completed.

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.



Name	Created by	Created on	Updated by	Updated on	
AccountSpecificInstructions Copied over	Jeshua Yates	17/09/2020	Jeshua Yates	29/09/2020	delete
AdminRegions AdminRegions	Alan Major	23/04/2018		-	delete
AlanAPITest AlanAPITest	WebApi	14/04/2016	WebApi	14/04/2016	delete
APITest APITest	WebApi	13/04/2016	WebApi	13/04/2016	delete
Approvers Approvers	Marshall Brooke	04/05/2018		-	delete
area_list Area list 11:19	DFSL	05/05/2015	DFSL	06/05/2015	delete

NEW DATABASE

Name: database name

Description: description

CSV File: Choose file No file chosen

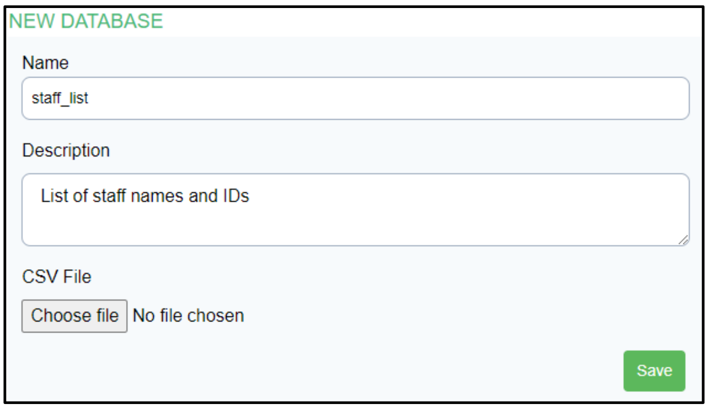
Save

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


NEW DATABASE

Name: staff_list

Description: List of staff names and IDs

CSV File: Choose file No file chosen

Save

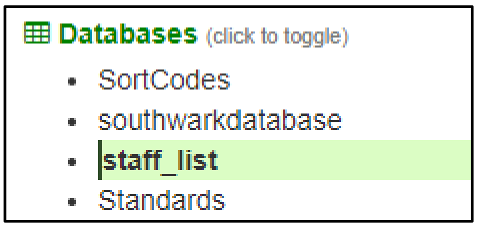
- From the databases tab, enter the name of the new database in the Name field on the right of the screen. The name must not contain any spaces, but you can use underscores, for example, staff_list.
- Enter a description.
- Browse to the CSV file and select it.
- Select Save.
  - A message indicating 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.

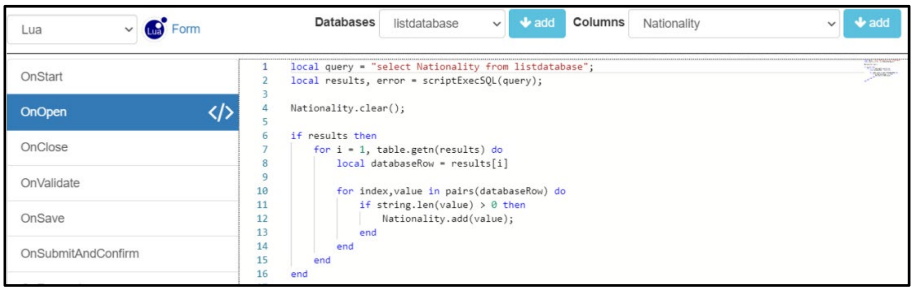


Databases (click to toggle)

- SortCodes
- southwarkdatabase
- staff_list
- Standards

Populating a basic list box

The Script Template area lists which databases are available for use within your code, and if you select a database, the columns drop down list will inform you of the column names, so you can include them in your queries. The most common mistake users make with databases is using column names in script that aren't in the database. You can easily check column names with the columns drop down list. As you can see, if and do statements must always have a matching end statement.



```
1   local query = "select Nationality from listdatabase";
2   local results, error = scriptExecSQL(query);
3
4   Nationality.clear();
5
6   if results then
7       for i = 1, table.getn(results) do
8           local databaseRow = results[i]
9
10          for index,value in pairs(databaseRow) do
11              if string.len(value) > 0 then
12                  Nationality.add(value);
13              end
14          end
15      end
16  end
```

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 Nationality 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 1 and 2 create the query and extract the Nationality column from the listdatabase database. Line 4 clears the contents of the Nationality list box, ready for new values. Lines 6 and 16 are an if, end test, which checks that records have been returned from the database that match your query. Lines 7 and 15 are a do, end loop that processes each record extracted. Lines 10 and 14 are a nested do, end loop that processes every column in the 'current' record. Line 11 checks the length of the value in the Nationality column is greater than 0. This has been placed here as a safeguard. You could also have amended the query to: "select distinct Nationality from listdatabase where length(Nationality) > 0"; to achieve the same result. Line 12 populates the Nationality list box with the value of the column.

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");
```

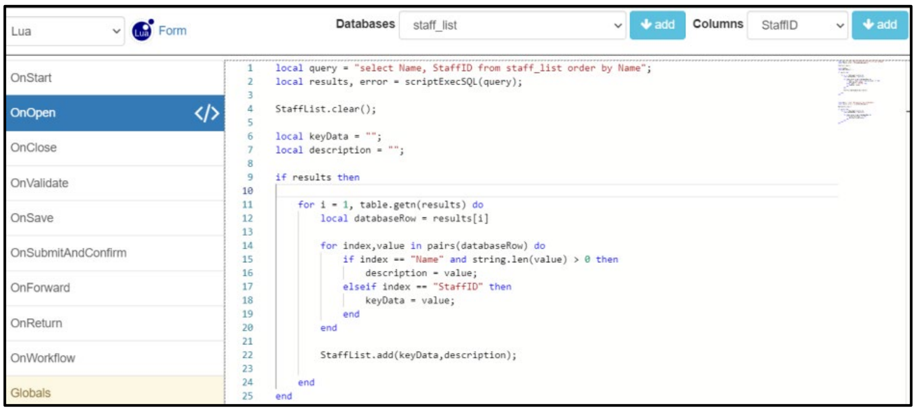
To display more than one column value in the list box, columns can be concatenated using '||'. In the below example, the Rate and Unit columns will be displayed in the list box.

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

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.

The important thing to note, is that as per line 14, the column headings will always be stored in the "index" variable, and the value of the column will always be in the "value" variable. This permits script like this example, where we are testing the index to see which column is currently being processed.



```
1   local query = "select Name, StaffID from staff_list order by Name";
2   local results, error = scriptExecSQL(query);
3
4   StaffList.clear();
5
6   local keyData = "";
7   local description = "";
8
9   if results then
10
11      for i = 1, table.getn(results) do
12          local databaseRow = results[i]
13
14          for index,value in pairs(databaseRow) do
15              if index == "Name" and string.len(value) > 0 then
16                  description = value;
17              elseif index == "StaffID" then
18                  keyData = value;
19              end
20          end
21
22          StaffList.add(keyData,description);
23
24      end
25  end
```

You will notice that whilst the column heading Staff Code needs to be placed in square brackets for the SQL query on line 1, it must not be in brackets for the test of the index variable on line 17. Placing square brackets in the test on Line 17 will mean that the column will never be found and the test never true.

In this example, if the column being processed is StaffName, the value from that column is placed in the description variable. If the column is Staff Code, the value is placed in the keyData variable. After processing leaves the do, end loop that processes the current record, (Lines 14 and 20), we populate the StaffList list box on line 22. The keyData forms the hidden, return value, and description 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 11 and 24) continues.

So, basically you have two loops, the first cycles through every individual record extracted from the database (Lines 11,24), and the second cycles through the column headings of the current record being processed (Lines 14, 20).

It may help you visualise the if / end, and do / end constructs by indenting them as I have in these examples. Otherwise it is easy to miss a matching end statement.

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");
```

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

The above script works fine if you know exactly what value you are using the query your databases. But you may only have a partial value. For example, you may wish to retrieve all clients with DA contained within their postcodes. 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%'

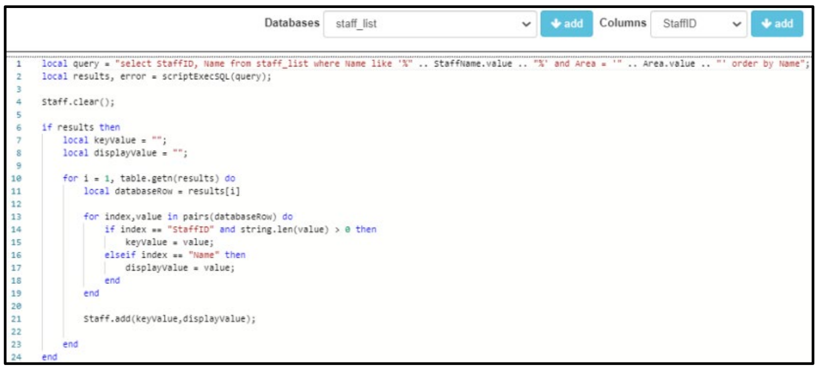
This would return all records with any number of characters following the smi. To return all client records that contained smi anywhere within the surname name field:

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



```
local query = "select StaffID, Name from staff_list where Name like '%" .. StaffName.value .. "%' and Area = '" .. Area.value .. "' order by Name";
local results, error = scriptExecSQL(query);

Staff.clear();

if results then
    local keyValue = "";
    local displayValue = "";

    for i = 1, table.getn(results) do
        local databaseRow = results[i]

        for index,value in pairs(databaseRow) do
            if index == "StaffID" and string.len(value) > 0 then
                keyValue = value;
            elseif index == "Name" then
                displayValue = value;
            end
        end

        Staff.add(keyValue,displayValue);

    end
end
```

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.value .. "%' and Area = '" .. Area.value .. "' 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.value .. "%' and Area = '" .. Area.value .. "' or Salary > " .. Salary.value .. " 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 Lua uses two == to indicate that one value equals another.

Joining Databases

Types of Joins

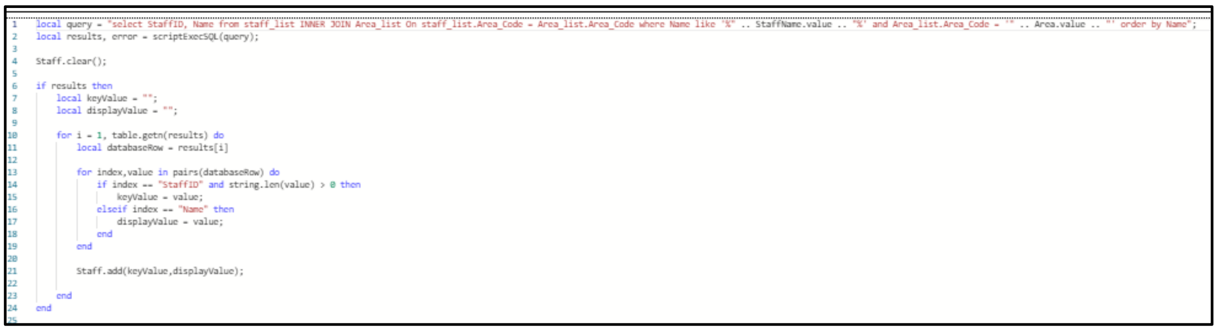
There are different types of joins available in SQLite:

- 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. In this instance; Area_list.Area_Code. In the example, the description of the area is placed in the



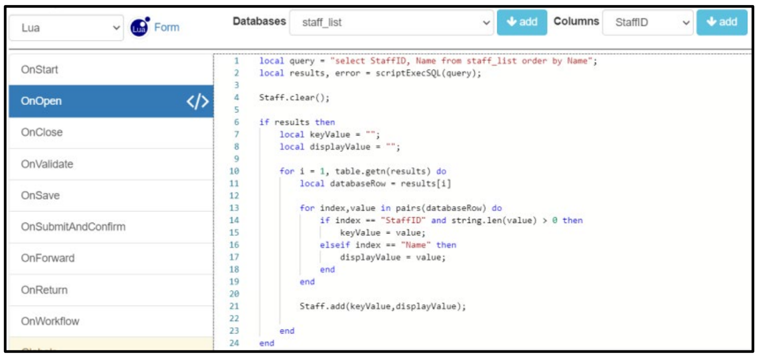
```
1   local query = "select StaffID, Name from staff_list INNER JOIN Area_list On staff_list.Area_Code = Area_list.Area_Code where Name like '%" .. StaffName.value .. "%' and Area_list.Area_Code = '" .. Area.value .. "' order by Name";
2   local results, error = scriptExecSQL(query);
3
4   Staff.clear();
5
6   if results then
7       local keyValue = "";
8       local displayValue = "";
9
10      for i = 1, table.getn(results) do
11          local databaseRow = results[i]
12
13          for index,value in pairs(databaseRow) do
14              if index == "StaffID" and string.len(value) > 0 then
15                  keyValue = value;
16              elseif index == "Name" then
17                  displayValue = value;
18              end
19          end
20
21          Staff.add(keyValue,displayValue);
22
23      end
24  end
25
```

Areaname textbox in line 18.

Populating a List box from a Database

Populating a drop down list box is probably the most common use of database and has been used for most of the screen captures on this subject. If you hard code a list of say clients or staff names using the Template Designer, you risk one or more of the names changing. To amend the list on a published form would entail creating a duplicate of the form, amending the duplicate, publishing it and retiring the original template. But with databases you can simply upload a replacement CSV file.

In this example we are populating a list box with an alias name of Staff, with a list of staff names and staff id's. The script has been placed in the form's OnOpen event, so the list box is populated as soon as the user opens it.



```
1   local query = "select StaffID, Name from staff_list order by Name";
2   local results, error = scriptExecSQL(query);
3
4   Staff.clear();
5
6   if results then
7       local keyValue = "";
8       local displayValue = "";
9
10      for i = 1, table.getn(results) do
11          local databaseRow = results[i]
12
13          for index,value in pairs(databaseRow) do
14              if index == "StaffID" and string.len(value) > 0 then
15                  keyValue = value;
16              elseif index == "Name" then
17                  displayValue = value;
18              end
19          end
20
21          Staff.add(keyValue,displayValue);
22
23      end
24  end
```

- Line 1: We store the SQL query in a local variable, "query". We are selecting two fields from the Staff_list database, StaffID and Name. To select all the columns, we would use the * character instead of listing them.
- Line 2: We use the scriptExecSQL() function to execute the query against the Staff_list database. The scriptExecSQL function returns two parameters, an array of Lua databases, and a string indicating if an error arose during the execution of the query. If the query executes correctly, the error parameter will equal "".
- Line 4: We clear any values from the list box, preparing it for the values from the Staff_list database.
- Line 6: Tests the results variable, to see if any values have been returned by the query.
- Lines 7 and 8 set up the local variables to hold the key and display values for the list box.
- Line 10: Starts to iterate through the data rows returned by the query.
- Line 11: Places the entire data row in the databaseRow variable.
- Lines 13 through 19 iterate through the columns:
  - The index value will always return the column name. You should NEVER rely on the order of the columns to retrieve data.
  - In this instance, the column value will be stored in the value variable.
  - Using this method, you can retrieve the value from any column name/value pair.
- Line 21: Pushes the StaffID and name into the Staff list box.
  - The name will be displayed, and the staff id returned in exports of say CSV or XML.

- If you only require a description in the list box, with no specific code returned, i.e., if the staff name is sufficient without the staff id, then you would just use Staff.add(value);

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.value .. "'";
```

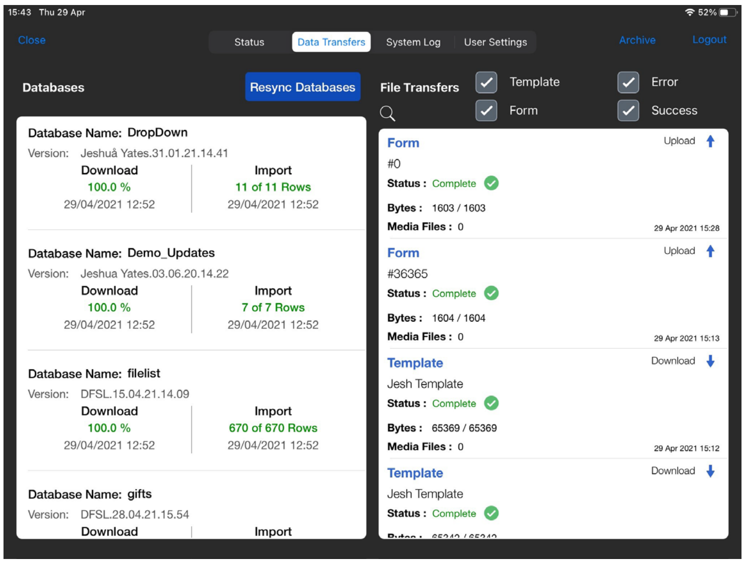
This query would take the value from the Areas element, (either a list box or text box), and when executed against the Client database, only extract clients that matched the area code. The .. characters are used to concatenate the value from the Areas element into the query, and the apostrophe character (') 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(i) screen. Databases are displayed on the left side of the Data Transfers tab of the Settings screen.



Database Name: DropDown
Version: Jeshuå Yates.31.01.21.14.41
Download 100.0 % 29/04/2021 12:52 | Import 11 of 11 Rows 29/04/2021 12:52

Database Name: Demo_Updates
Version: Jeshua Yates.03.06.20.14.22
Download 100.0 % 29/04/2021 12:52 | Import 7 of 7 Rows 29/04/2021 12:52

Database Name: filelist
Version: DFSL.15.04.21.14.09
Download 100.0 % 29/04/2021 12:52 | Import 670 of 670 Rows 29/04/2021 12:52

Database Name: gifts
Version: DFSL.28.04.21.15.54
Download | Import

Form — Upload
#0
Status : Complete
Bytes : 1603 / 1603
Media Files : 0
29 Apr 2021 15:28

Form — Upload
#36365
Status : Complete
Bytes : 1604 / 1604
Media Files : 0
29 Apr 2021 15:13

Template — Download
Jesh Template
Status : Complete
Bytes : 65369 / 65369
Media Files : 0
29 Apr 2021 15:12

Template — Download
Jesh Template
Status : Complete

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.

Post Code Lookup

Introduction

Formworks provides three functions to integrate with web services. The example provided below details how to integrate a Formworks application with the Post Code Anywhere post code lookup service.

http.checkConnected(), http.getStringFromUrl(url) and http.getTableFromJSON(url)

The http.getTableFromJSON() function is the main one involved in post code lookups.. This function passes a URL to the post code lookup web service, and receives a string of address data back. The returned string is in the JSON format. This function automatically converts this string into a Lua table. This table structure simplifies the display of the address data. There is no comparable Formworks function to work with XML data, so it is recommended that a post code service that provides JSON formatted data, such as Post Code Anywhere is used.

The steps involved in retrieving and displaying post code data are:

1. Format a string that includes your account key, the post code to be queried and your account key, and user name, to use as a URL which is passed to the post code lookup service provided.
2. Use the http.getTableFromJSON() function to pass the URL to the web service, and store the string of address data returned by the lookup service, to a Lua table.
3. Display the returned partial address data in a drop down list box.
4. The device user selects the partial address from the drop down list, which provides your script with a unique address ID code.
5. You can use this unique ID to format a second URL, which when passed to the web service, will provide detailed address data.

In script, format a string with the URL for the web service that includes the post code to be queried, your account key and user name. Remove any spaces from the post code before inserting in the URL. This should look similar to the following example:

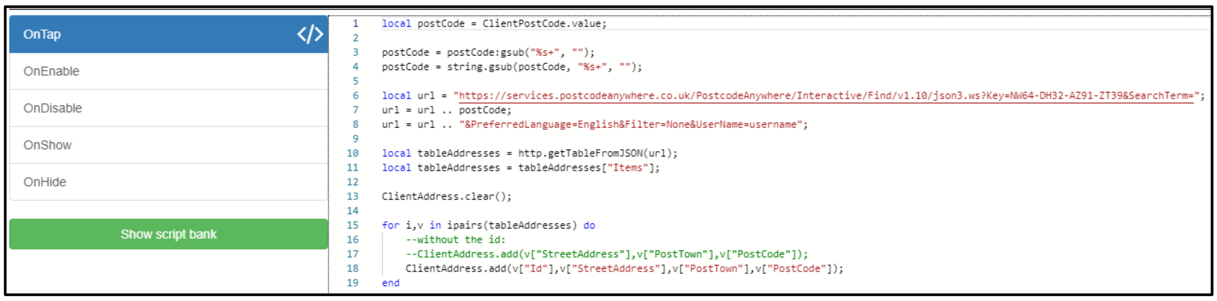
http://services.postcodeanywhere.co.uk/PostcodeAnywhere/Interactive/Find/v1.10/json3.ws?Key=AA33-AA33-AA33-AA33&SearchTerm=DA122NG&PreferredLanguage=English&Filter=None&UserName=username";

In this example, AA33-AA33-AA33-AA33 is the account key, DA122NG is the post code to be queried, and username is the user name.

Call the post code lookup web service, using the http.getTableFromJSON() function, passing the URL that you have constructed. Save the string of partial address data returned by the function, to a Lua table. This is handled for you automatically by the function.

Ensure that the drop down list box used to display the address is clear then populate the list box with the table, by iterating over the tables contents.

The script to achieve this is included in the example below, which has been placed in the OnTap event of a button.



```lua
local postCode = ClientPostCode.value;

postCode = postCode:gsub("%s+", "");
postCode = string.gsub(postCode, "%s+", "");

local url = "https://services.postcodeanywhere.co.uk/PostcodeAnywhere/Interactive/Find/v1.10/json3.ws?Key=NW64-DH32-AZ91-ZT39&SearchTerm=";
url = url .. postCode;
url = url .. "&PreferredLanguage=English&Filter=None&UserName=username";

local tableAddresses = http.getTableFromJSON(url);
local tableAddresses = tableAddresses["Items"];

ClientAddress.clear();

for i,v in ipairs(tableAddresses) do
    --without the id:
    --ClientAddress.add(v["StreetAddress"],v["PostTown"],v["PostCode"]);
    ClientAddress.add(v["Id"],v["StreetAddress"],v["PostTown"],v["PostCode"]);
end
```

It may be that the partial address data supplied above is sufficient for your requirements. If so, you should substitute the line:

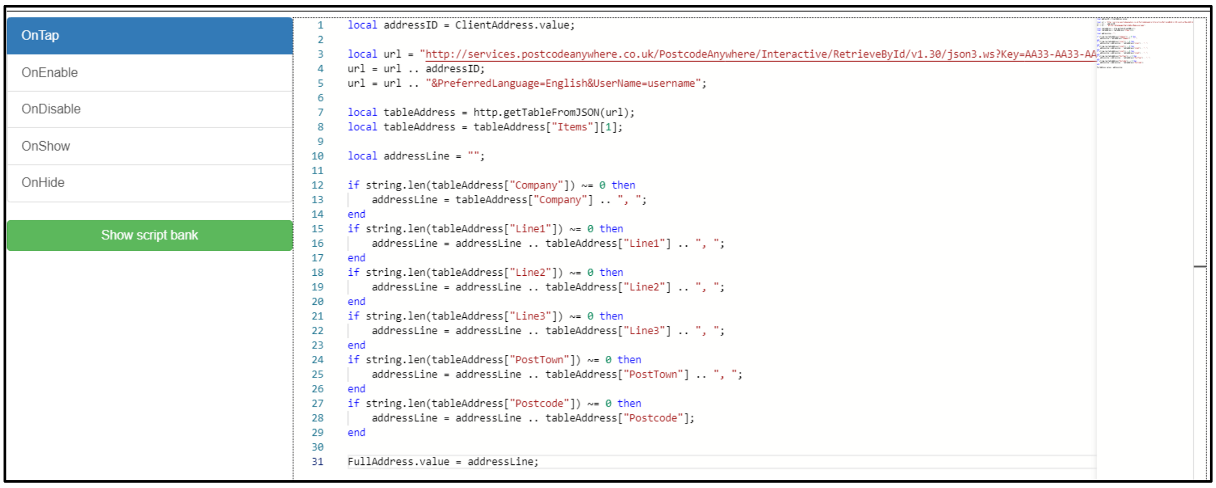
```
ClientAddress.add(v["StreetAddress"],v["PostCode"]);
```

For the appropriate line in the above example. The list boxes value property will then contain the address details instead of a unique property ID.

If not, then you will need to make a second call to the post code lookup service to obtain detailed address data. You do this by acquiring the unique ID from the list box's value property, when the device user selects the address from the drop down list. Place the id in a string, using the following format:

http://services.postcodeanywhere.co.uk/PostcodeAnywhere/Interactive/RetrieveById/v1.30/json3.ws?Key=AA33-AA33-AA33-AA33&Id=23747212.00&PreferredLanguage=English&UserName=username

Again, the string returned by the http.getTableFromJSON() function is stored in a Lua table. You can check if the fields in the table are empty by using the string.len() function. In the example below, an address field has been populated, by checking if table fields are empty, and if not, their contents are concatenated (..).



```lua
local addressID = ClientAddress.value;

local url = "http://services.postcodeanywhere.co.uk/PostcodeAnywhere/Interactive/RetrieveById/v1.30/json3.ws?Key=AA33-AA33-AA
url = url .. addressID;
url = url .. "&PreferredLanguage=English&UserName=username";

local tableAddress = http.getTableFromJSON(url);
local tableAddress = tableAddress["Items"][1];

local addressLine = "";

if string.len(tableAddress["Company"]) ~= 0 then
    addressLine = tableAddress["Company"] .. ", ";
end
if string.len(tableAddress["Line1"]) ~= 0 then
    addressLine = addressLine .. tableAddress["Line1"] .. ", ";
end
if string.len(tableAddress["Line2"]) ~= 0 then
    addressLine = addressLine .. tableAddress["Line2"] .. ", ";
end
if string.len(tableAddress["Line3"]) ~= 0 then
    addressLine = addressLine .. tableAddress["Line3"] .. ", ";
end
if string.len(tableAddress["PostTown"]) ~= 0 then
    addressLine = addressLine .. tableAddress["PostTown"] .. ", ";
end
if string.len(tableAddress["Postcode"]) ~= 0 then
    addressLine = addressLine .. tableAddress["Postcode"];
end

FullAddress.value = addressLine;
```

Address fields available

Appendix V contains a listing of all the available address fields returned by the Post Code Anywhere post code lookup service. The following are the main ones that you are likely to require are; Company, Line1, Line2, Line3, Line4, Line5, PostTown, County, Postcode.

The easiest way to tell what address fields you may be interested in, is to use the http.getStringFromUrl(url) function with your URL and save the string returned to a paragraph element's value property, so you can inspect it.

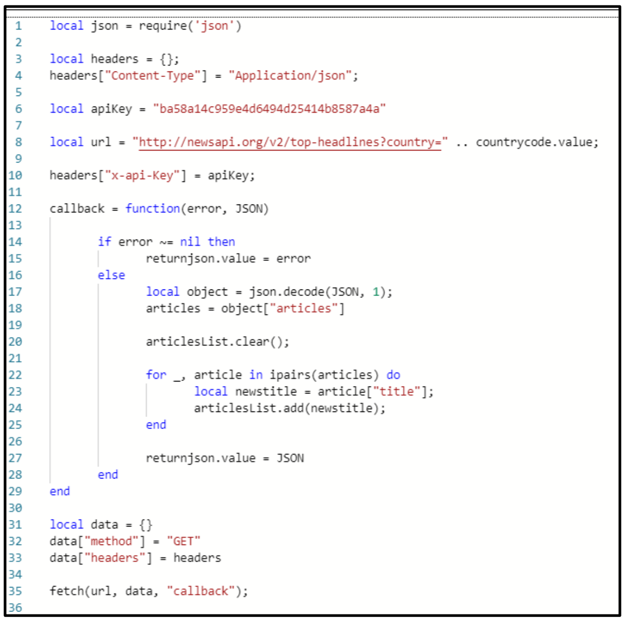
Other Network Functions

Formworks XD brings along a new network function to retrieve information using the GET method.

The fetch function requires three parameters, a url, data (ie the method) and a function. The format is: fetch(url, data, "functionName");

The example to the right demonstrates how to use the fetch function and a created function 'callback' to pull news articles from a news API using a country code value to populate a drop down field.

The headers array is created on line 3 and will hold the Content-Type and API key for the News API. The country code used to complete the url is derived from a user input text field on line 8.

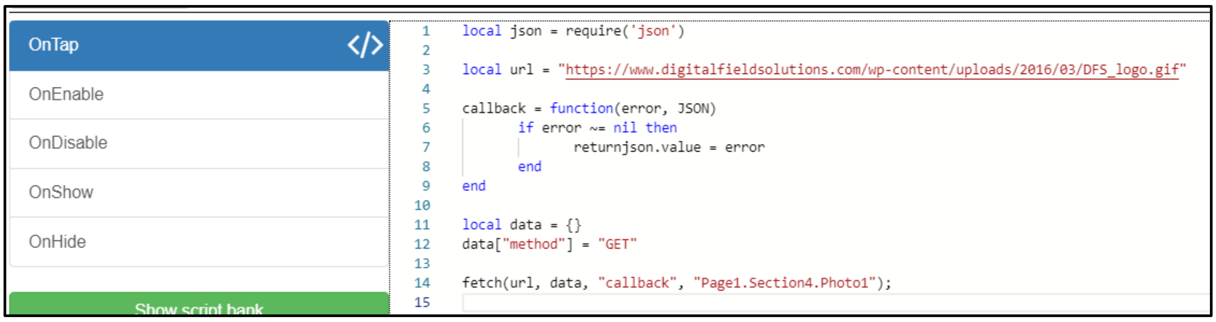


```
1   local json = require('json')
2
3   local headers = {};
4   headers["Content-Type"] = "Application/json";
5
6   local apiKey = "ba58a14c959e4d6494d25414b8587a4a"
7
8   local url = "http://newsapi.org/v2/top-headlines?country=" .. countrycode.value;
9
10  headers["x-api-Key"] = apiKey;
11
12  callback = function(error, JSON)
13
14          if error ~= nil then
15                  returnjson.value = error
16          else
17                  local object = json.decode(JSON, 1);
18                  articles = object["articles"]
19
20                  articlesList.clear();
21
22                  for _, article in ipairs(articles) do
23                          local newstitle = article["title"];
24                          articlesList.add(newstitle);
25                  end
26
27                  returnjson.value = JSON
28          end
29  end
30
31  local data = {}
32  data["method"] = "GET"
33  data["headers"] = headers
34
35  fetch(url, data, "callback");
36
```

The callback function will return any article results from the url and populate the articleList dropdown.

The data array consists of the GET method and the header array.

The fetch function can also be used to populate a photo field from a url. This requires an additional parameter after the function parameter to name the photo field. The below example demonstrates this.



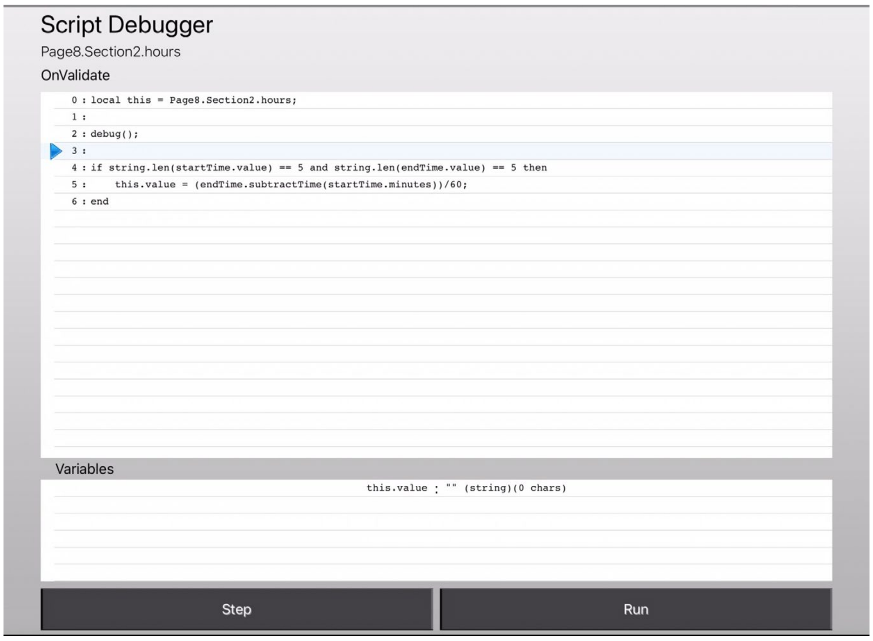
```
local json = require('json')

local url = "https://www.digitalfieldsolutions.com/wp-content/uploads/2016/03/DFS_logo.gif"

callback = function(error, JSON)
       if error ~= nil then
              returnjson.value = error
       end
end

local data = {}
data["method"] = "GET"

fetch(url, data, "callback", "Page1.Section4.Photo1");

```

Debugging

Introduction

Debug support within Formworks is currently in Beta. It is intended as a step towards helping with script debugging, rather than a robust, fully-featured debugging environment.

Calling the debug() method in a script will show a debug window, giving the user the ability to step through the script and see any local variable values. The debug window will close when the script has finished running.



```
local this = Page8.Section2.hours;

debug();

if string.len(startTime.value) == 5 and string.len(endTime.value) == 5 then
    this.value = (endTime.subtractTime(startTime.minutes))/60;
end
```

```
this.value : "" (string)(0 chars)
```

If the script is not already debugging, and the form is in test (i.e., not published), the script is broken down line-by-line, with each line being run individually, one line at a time on a separate thread.

When a form has been either published or saved, all calls to debug() are removed.

Note: Current limitations to the debugging process are:

- Unable to step into Lua functions called within functions
- Scripts with more than 99,999 lines of code may not debug correctly
- Adding debug() to an OnValidate() event handler will prevent the Submission window from showing , so the form cannot be submitted.
- The Debug window does not appear on forms that have been previously saved.

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.

The format for setting a valid property to false is: element.valid = false;

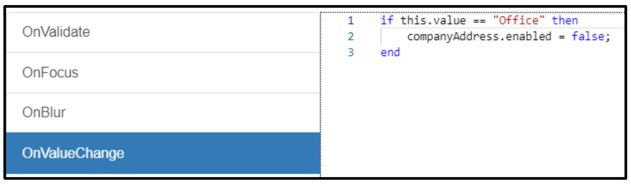
message Type: String

The message property of an element is used to display messages in the Submit window, when the device user selects Submit Form. The most common usage of the Message property is to deliver a meaningful message to the device user when incorrect data has been entered on a form, or when a required field has not been completed. The Message property works in combination with the Valid property. If the Valid property of the element has been set to false, the message will be an error type, and display an amber triangle. If the Valid property is true, the message type will be a warning and display a red icon. The Message property, like the Valid property, can only be set in the OnValidate event of an element.

The format to set the Message property for an element is: element.message = "Hello World";

enabled Type: Boolean

The Enabled property allows you to enable/disable an element. 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.



```
if this.value == "Office" then
    companyAddress.enabled = false;
end
```

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 = true/false;

visible Type: Boolean

The Visible property allows you to 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 = 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. So, both the following statements are valid uses for the title property: element.title = "Company Name"; or stringVariable = element.title;

Note: The title properties of the container elements Page, Section and Group, are read only. They cannot be changed within script. Page titles can be changed within script in the Formworks XD App Only.

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.

```
local device = isIphone();
if device == true then
  Device.value = "iPhone";
else
  Device.value = "iPad";
end
```

Note:Formworks XD App only.

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

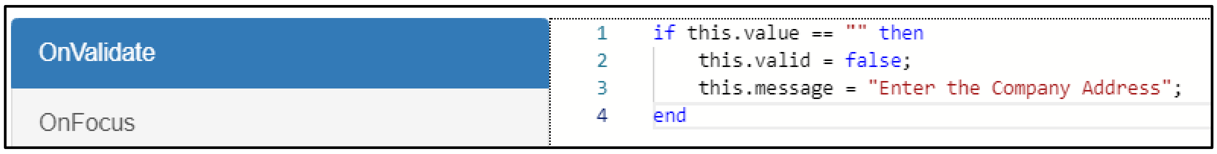
An element's visible property can be used to make the element visible or invisible. When the element's visible property is set to true, the element's OnShow event is fired.

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.



```
if this.value == "" then
    this.valid = false;
    this.message = "Enter the Company Address";
end
```

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 against the element's Message property, 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 message and valid properties can only be set in the OnValidate event. If you set an element's message property and leave it's valid property as true, a message can be displayed on the Form Submit window, for example as a warning.

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 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.

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 200mseconds of the last keypress.

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 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

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 will need to be placed in the forms OnOpen event as well.

Form Specific Methods and Properties

Audio Functions

form.startRecording()

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

form.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: form.pauseRecording();

form.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: form.stopRecording(); or form.stopRecording("recording1");

form.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.value = form.recordingStatus();

Miscellaneous Functions

form.changePage("aliasName / pagename")

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 form.changePage() 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 form.changePage() instruction.

form.openURL(strVar)

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, form.openURL("https://www.bbc.co.uk/news") would open the BBC news website. You can move back to the Formworks input screen by selecting Formworks, from the top right hand side of the screen.

form.templateId

This property is read only and returns the unique template id for the form in use. An example would be: stringVar = form.templateId; stringVar would contain the GUID value of the template.
As always, these instructions are case sensitive.

Appendix I

Elements - their properties, events and methods

Buttons

Properties

enabled	Boolean. Format: element.enabled = true/false;
title	String. Format: element.title = "New Title";
valid	Boolean. Format: element.valid = true/false;
visible	Boolean. Format: element.visible = true/false;
color	String. Example element.color = "red"; Note: Formworks XD App Only.

Events

OnDisable	Fired when the enabled property is set to false
OnEnable	Fired when the enabled property is set to true
OnHide	Fired when the visible property is set to false
OnShow	Fired when the visible property is set to true
OnTap	Fired when a button element is touched

Methods

OnTap	Used to active the script behind the button

Checkboxes

Properties

title	String. Format: element.title = "New Title";
value	String. The value contained within the field.
enabled	Boolean. Format: element.enabled = true/false;
message	String. Format: element.message = "New Message";
valid	Boolean. Format: element.valid = true/false;
visible	Boolean. Format: element.visible = true/false;

Events

OnBlur	Fired when the element loses focus
OnDisable	Fired when the enabled property is set to false
OnEnable	Fired when the enabled property is set to true
OnFocus	Fired when the element gains focus
OnHide	Fired when the visible property is set to false

OnShow	Fired when the visible property is set to true
OnValidate	OnValidate events are fired in turn when a user submits a form
OnValueChange	Fired when the value for an element changes

Methods

focus	Sets focus to a specified field. Format: element.focus();
OnValidate	Fires an element's OnValidate event. Format: element.OnValidate();

Date elements

Properties

enabled	Boolean. Format: element.enabled = true/false;
message	String. Format: element.message = "New Message";
title	String. Format: element.title = "New Title";
valid	Boolean. Format: element.valid = true/false;
value	String. The value contained within the field.
visible	Boolean. Format: element.visible = true/false;
color	Not supported.
days	Number. Format: element.days = 17110.
minimumDate	String input. Format: "today" or "14/12/2018". See date functions.
minimumDateDays	String input. Format "17110".
maximumDate	String input. Example "today-5480". See date functions.
maximumDateDays	Number input. Format "43953".
seconds (read only)	
weekday (read only)	

Events

OnBlur	Fired when the element loses focus
OnDisable	Fired when the enabled property is set to false
OnEnable	Fired when the enabled property is set to true
OnFocus	Fired when the element gains focus
OnHide	Fired when the visible property is set to false
OnShow	Fired when the visible property is set to true
OnValidate	OnValidate events are fired in turn when user submits a form
OnValueChange	Fired when the value for an element changes

Methods

focus	Sets focus to a specified field. Format: element.focus();
OnValidate	Fires an element's OnValidate event. Format: element.OnValidate();

Functions

dateElement1.equalTo(dateElement2.value)
FirstDate.lessThan(SecondDate.value);
dateElement1.greaterThan(dateElement2.value)
dateElement.age()

Forms

Properties

templateId	Usage: form.templateId. This property is read only and returns the unique template id for the form in use. An example would be: stringVar = form.templateId; stringVar would contain the GUID value of the template.

Events – in order of occurrence.

For details see "Form Specific Events" above.

OnStart OnOpen OnClose
OnValidate OnSave
OnSubmitAndConfirm Globals

Methods - General

changePage	Changes the page within the form. Format: form.changePage("Page2");
save	Saves the form. Format: form.save();
close	Closes the form. Saves first by default, unless the save parameter is set to false. Format: form.close(); or close(true/false) to control saving of the form;
submit	Validates the form and shows the Submission window. Format: form.submit();
submitWithoutConfirm	Submits a form, but if there are no validation errors, the validation screen does not display. Format: form.submitWithoutConfirm();
openFormWithName	Can be used to automatically open another form when current one closes.
delete	Deletes the form from the device, if permitted by form template settings. Format: form.delete();
OnValidate	Triggers the form.OnValidate event.

Methods - Audio

startRecording	Toggles between recording and pausing. First time starts recording, second time pauses and third time resumes recording. Format: form.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: form.pauseRecording();
stopRecording	Stop an active recording. Can also set the filename within parentheses. Format: form.stopRecording();
recordingStatus	Returns the state of an active or inactive recording. Format: form.recordingStatus();

Related functions

getCurrentUserName()

getCurrentUserEmail()
getCurrentLocation()
setUserData("Key",value)
getUserData("Key")
isIphone()

Groups and Sections

Properties

enabled	Boolean. Format: element.enabled = true/false;
message	String. Format: element.message = "New Message";
title	String. Format: stringVariable = element.title; **Note:** The title property of these container elements are read only.
valid	Boolean. Format: element.valid = true/false;
visible	Boolean. Format: element.visible = true/false;

Events

OnDisable	Fired when the enabled property is set to false
OnEnable	Fired when the enabled property is set to true
OnHide	Fired when the visible property is set to false
OnShow	Fired when the visible property is set to true
OnValidate	OnValidate events are fired in turn when user submits a form

Methods

OnValidate	Fires an element's OnValidate event. Format: element.OnValidate();
color("colour1","colour2");	Sets the foreground / background colour on sections. Groups and tables are not currently supported. Example: section.color("red","green");
enabled	Enables or disables every element with a section or group. Format: element.enabled = true/false;
visible	Sets the visibility of a section or group. Format: element.visible = true/false;

Images

Properties

visible	Boolean. Format: element.visible = true/false;

Events

OnHide	Fired when the visible property is set to false
OnShow	Fired when the visible property is set to true

Methods

set	Format: element.set("url");

Labels

Properties

title	String. Format: element.title = "New Title";
visible	Boolean. Format: element.visible = true/false;
color	String. Example element.color = "red"; Note: Formworks XD App Only.

Events

OnHide	Fired when the visible property is set to false
OnShow	Fired when the visible property is set to true

Line

Properties

visible	Boolean. Format: element.visible = true/false;

Events

OnHide	Fired when the visible property is set to false
OnShow	Fired when the visible property is set to true

Multi-Select

Properties

enabled	Boolean. Format: element.enabled = true/false;
message	String. Format: element.message = "New Message";
title	String. Format: element.title = "New Title";
valid	Boolean. Format: element.valid = true/false;
value	String. The value contained within the field.
visible	Boolean. Format: element.visible = true/false;

Events

OnBlur	Fired when the element loses focus
OnDisable	Fired when the enabled property is set to false
OnEnable	Fired when the enabled property is set to true
OnFocus	Fired when the element gains focus
OnHide	Fired when the visible property is set to false
OnShow	Fired when the visible property is set to true
OnValidate	OnValidate events are fired in turn when a user submits a form
OnValueChange	Fired when the value for an element changes

Methods

Focus	Sets focus to a specified field. Format: element.focus();
OnValidate	Fires an element's OnValidate event. Format: element.OnValidate();
count()	Returns the number of items available. Format element.count();
countSelected()	Return the number of items selected. Format element.countSelected();
getOption()	Returns true or false, depending on whether the option specified within the brackets has been selected. Format: element.getOption("Option 1").value would return true if "Option 1" had been selected
setOption("Option 1", false)	Use this method to set the values to true or false within a multiselect element. E.g., Page1.Section1.ElementName.setOption("Option 1", false)

Pages

Properties

enabled	Boolean. Format: element.enabled = true/false;
message	String. Format: element.message = "New Message";
title	String. Format: stringVariable = element.title; **Note:** The title property of the page container element is read only except in the Formworks XD App.
valid	Boolean. Format: page.valid = true/false;
visible	Boolean. Format: pageName.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.
- It is not possible to hide a page from the onBeforeOpen() and onBeforeStart() events. The form is not in scope at these points.

Events

OnDisable	Fired when the enabled property is set to false
OnEnable	Fired when the enabled property is set to true
OnValidate	OnValidate events are fired in turn when users selects to submit form

Methods

OnValidate Fires an element's OnValidate event. Format: element.OnValidate();

Photos, Signatures and Sketches

Properties

enabled	Boolean. Format: element.enabled = true/false;
message	String. Format: element.message = "New Message";
title	String. Format: element.title = "New Title";
valid	Boolean. Format: element.valid = true/false;
value	String. The value contained within the field.
visible	Boolean. Format: element.visible = true/false;

Events

OnDisable	Fired when the enabled property is set to false
OnEnable	Fired when the enabled property is set to true
OnFocus	Fired when the element gains focus
OnHide	Fired when the visible property is set to false
OnShow	Fired when the visible property is set to true
OnValidate	OnValidate events are fired in turn when a user submits a form
OnValueChange	Fired when the value for an element changes

Methods

OnValidate	Fires an element's OnValidate event. Format: element.OnValidate();
clear()	The clear method removes the contents of element. i.e., Photo1.clear(); Note: Sketch fields can only be cleared within script in the Formworks XD App.

Single Selects

Properties

enabled	Boolean. Format: element.enabled = true/false;
message	String. Format: element.message = "New Message";
title	String. Format: element.title = "New Title";
valid	Boolean. Format: element.valid = true/false;
value	String. The value contained within the field. This can also be used to clear the selected value, i.e., listBox.value = "";, or set the value, i.e. listBox.value = "Yes";
visible	Boolean. Format: element.visible = true/false;

Events

OnBlur	Fired when the element loses focus
OnDisable	Fired when the enabled property is set to false
OnEnable	Fired when the enabled property is set to true
OnFocus	Fired when the element gains focus
OnHide	Fired when the visible property is set to false
OnShow	Fired when the visible property is set to true
OnValidate	OnValidate events are fired in turn when a user submits a form
OnValueChange	Fired when the value for an element changes

Methods

add	Adds a new option to the end of the list. Dropdowns only. You must use the element's fully qualified name, NOT the alias.
clear	Clear all options. Dropdowns only. Use listBox.value = "" to clear selection.
focus	
remove	Remove the option with the given value.
Dropdowns only.	
OnValidate	Triggers the OnValidate event.

setDatabaseConfiguration() Formworks XD App only. Populates a Dropdown single selection field from a database. Format:

```
County.setDatabaseConfiguration("databaseName",
"FilterByColoumnHeader", "[keyValue/columnHeader]",
"displayValue/coloumnHeader");
```

Filter Table Function

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.enabled = true/false;	Sets / reads enabled status.
Table.hideColumn(string/int) and Table.showColumn(string/int)	Hide / Reveal columns.
Table.hideRow(string/int) and Table.showRow(string/int)	Hide / Reveal rows.
Table.enableColumn(string/int) and Table.disableColumn(string/int)	Enable / Disable columns.
Table.enableRow(string/int) and Table.disableRow(string/int)	Enable / Disable rows.
Table.hideColumnHeaders() and Table.showColumnHeaders();	Hides / Reveals Headers
Table.hideBorders() and Table.showBorders()	Hides / Reveals grid lines.
Rows = Table.rowCount()	Returns the table row count
Table.column_3.visible = true/false;	Sets column visibility.

Text Boxes and Paragraph elements

Properties

enabled	Boolean. Format: element.enabled = true/false;
message	String. Format: element.message = "New Message";
title	String. Format: element.title = "New Title";
valid	Boolean. Format: element.valid = true/false;
value	String. The value contained within the field.
visible	Boolean. Format: element.visible = true/false;
color	String. Example element.color = "red"; Note: The color property can only be updated for Paragraph fields within script in the Formworks XD App

Events

OnBlur	Fired when the element loses focus
OnDisable	Fired when the enabled property is set to false
OnEnable	Fired when the enabled property is set to true
OnFocus	Fired when the element gains focus
OnHide	Fired when the visible property is set to false
OnShow	Fired when the visible property is set to true
OnValidate	OnValidate events are fired in turn when user submits a form
OnValueChange	Fired when the value for an element changes, with a delay of 200mseconds in the Formworks XD App.

Methods

focus	Sets focus to a specified field. Format: element.focus();
OnValidate	Fires an element's OnValidate event. Format: element.OnValidate();

Time elements

Properties

enabled	Boolean. Format: element.enabled = true/false;
message	String. Format: element.message = "New Message";
title	String. Format: element.title = "New Title";
valid	Boolean. Format: element.valid = true/false;
value	String. The value contained within the field.
visible	Boolean. Format: element.visible = true/false;
color	Not supported.
seconds	Number. Format: nvar = element.seconds;

Events

OnBlur	Fired when the element loses focus
OnDisable	Fired when the enabled property is set to false
OnEnable	Fired when the enabled property is set to true
OnFocus	Fired when the element gains focus
OnHide	Fired when the visible property is set to false
OnShow	Fired when the visible property is set to true
OnValidate	OnValidate events are fired in turn when user submits a form
OnValueChange	Fired when the value for an element changes

Methods

focus	Sets focus to a specified field. Format: element.focus();
OnValidate	Fires an element's OnValidate event. Format: element.OnValidate();
subtractTime()	Subtracting time elements: `Difference.value = Time2.subtractTime(Time1.minutes);` 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()	Can be used to compare or add times together. `Total.value = (tonumber(Time2.seconds) - tonumber(Time1.seconds))/60;`

Appendix IV

Character classes

.	all characters
%a	letters
%c	control characters
%d	digits
%l	lower case letters
%p	punctuation characters
%s	space characters
%u	upper case letters
%w	alphanumeric characters
%x	hexadecimal digits
%z	the character with representation o

Appendix V

Post Code Anywhere Address Fields

Company
Line1
Line2
Line3
Line4
Line5
PostTown
County
Postcode
Mailsort
Barcode
Type
DeliveryPointSuffix
SubBuilding
BuildingName
BuildingNumber
PrimaryStreet
SecondaryStreet
DoubleDependentLocality
DependentLocality
PoBox
PrimaryStreetName
PrimaryStreetType
SecondaryStreetName
SecondaryStreetType
CountryName
CountryISO2
CountryISO3