Formulator
1.0

User Manual

License Agreement

Table of Contents

Introduction

Formulator is an application that provides the display and editing of data in electronic forms.

Formulator was initially designed to provide electronic character sheets for role playing games, with a scripting language powerful enough to perform the calculations required by different RPGs.

However, it turns out that the design of the forms is rather flexible and Formulator can be used for almost any sort of electronic form.

An electronic form in Formulator consists of:

- Static text and graphics,
- Data entry fields,
- Control buttons,
- Calculated fields,
- Custom drawn fields.

The form description and form data are held in separate files.

This allows a form description to be updated while still retaining all of the existing data that has been previously entered into the form.

If you have ever used Excel or PDF based character sheets and spent hours typing your character stats, equipment etc. into the character sheet only for you GM to say "Oh, I have updated the character sheet, you all need to update your characters to the new version.", then you will understand why I wrote Formulator. These programs do not provide any easy way to transfer data from one form to another as the form design and form data are intrinsically tied together. Entering the same data starts getting pretty old after the third or fourth time.

The form layout and behaviour is defined in a simple XML form description language.

A form description consists of:

- A set of resources that can be used in the form. This includes the set of fonts and font sizes, line styles and bitmap graphics available in the form.
- A description of the layout of the form.
- A set of Lua scripts that control the behaviour of the form.

Using This Manual

This manual is aimed at two target audiences:

- Someone who is completely unfamiliar with desktop applications and wants to learn to use Formulator with existing forms.
- Someone who is interested in creating their own forms and needs to know the form file format and scripting language.

To cater to these two audiences the manual is divided into two main sections:

- A User Manual,
- A Reference Section that describes the form definition file format and the form scripting language.

User Manual

Formulator Window Layout

The Formulator window consists of the following areas:

- Menu Bar
- Tool Bar
- Forms Tabs
- Form Display Area

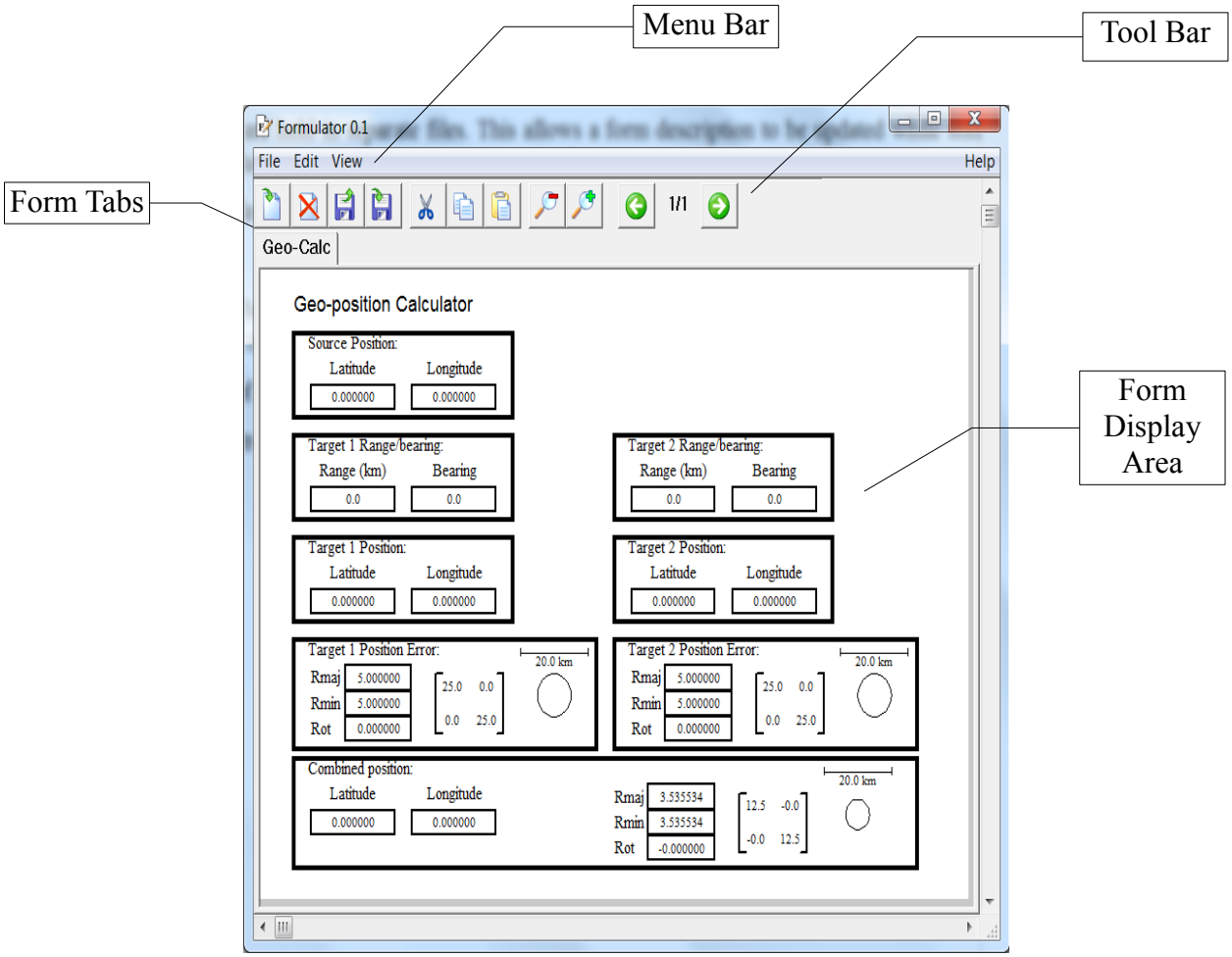


Illustration 1: Formulator Interface Elements

Menu Bar

The menu bar provides access to all file, editing and display actions available in Formulator.

Tool Bar

The tool bar provides quick access to the most commonly used actions.

The display of the button bar can be controlled through the menu.

Forms Tab

The forms tab allows the selection of the form to display from the set of currently loaded forms.

Form Display Area

This is where the contents of the currently selected form are displayed.

File Operations

Loading a Form

A new form may be loaded with either:

- The menu option File->Load Form, or
- The Load Form button on the tool bar.

Once loaded the new form will appear in the forms tab and will become the active form displayed in the form window.

The form will have all editable fields set to the default values specified by the form.

The same form may be loaded multiple times, displaying different data.

Any number of forms (subject to available memory) may be open at a time.

Loading Form Data

The data for a form that has previously been saved may be loaded by either:

- Selecting the menu option File->Load Data, or
- Clicking the Load Data button on the tool bar.

Note:

Unless the load and save method for the form has been overridden (see Form Scripting) then Formulator will save the field values in an XML file.

Formulator does not check that the loaded file contains data for the specific form. When form data is loaded, Formulator searches the file for all fields that match the field names of the form. This allows data from an old version of a form to be loaded into an updated version of the form.

Any fields in the data that are not understood by the current form are not loaded and will not be saved when saving the form data. Any fields in the form not specified in the file will remain with their current values.

Closing Forms

A form can be closed either by pressing the Close Form button on the Tool bar or from the menu option File->Close File.

The menu option File->Close All Files will close all currently open forms.

Saving Form Data

Save the current data in a form by either:

- Selecting the menu option File-> Save Data As, or
- Selecting the menu option File->Save Data, or
- Clicking the Save Data button on the tool bar.

Form File Sets

A form file set is a group of forms and their associated data files.

Form file sets provide a mechanism to quickly load a set of forms and data.

Selecting the menu option File->Save File Set will save the currently open forms and their associated data files as a file set.

The file set may now be reloaded by selecting the menu option File-> Load File Set.

Editing data in a Form

To edit a field in the form, move the pointer over the field to be edited and left click on the field. When the pointer is over an editable field the field will be highlighted in inverse colour.

A form may contain the following editable data fields:

- Text,
- Integer,
- Real Number,
- Boolean (used for on/off or yes/no options),
- Bitmap (used for selecting a bitmap image to be displayed)

Editing Text, Integer and Real Number fields

Text, Integer and Real Number fields are all edited in the same way.

Once the field has been activated by clicking on the field a cursor will be displayed in the field. The new field value can now be typed into the field.

The cursor keys can be used to move the cursor within the text being edited.

Double clicking the field will select all text in the field.

Holding shift while pressing the cursor keys will select a range of characters in the field.

When text in a field is selected any value typed will replace the selected text.

The menu item Edit->Copy or Ctrl+C will copy the currently highlighted text to the clipboard.

The menu item Edit->Paste or Ctrl+V will paste the current clipboard contents into the field, replacing any currently selected text.

The menu item Edit->Cut or Ctrl+X will delete the currently selected text and copy it to the clipboard.

Editing Boolean fields

Boolean fields act in a similar manner to checkboxes.

Once a boolean field has been selected by clicking on the field then its value can be toggled either by clicking the field again or by pressing the space bar.

Editing Bitmap fields

Once a Bitmap field has been selected by clicking on the field, clicking on the field again will bring up a file selection dialogue that allows a new bitmap image to be selected.

Interface Reference

Menu

File Menu

Menu Item	Description
Load Form	Load a form into a new tab.
Load Data	Load form data into the current form.
Save Data	Save the data entered into the current form.
Save Data As	Save the data entered into the current form to a new data file.
Close File	Close the current form.
Close All Files	Close all open forms.
Load File Set	Load a previously saved file set.
Save File Set	Save the currently open forms and associated data files as a file set.
Print	Print the current form.
Exit	Exit Formulator.

Edit menu

Menu Item	Description
Cut	Delete the currently selected text and copy the deleted text into the clipboard.
Copy	Copy the currently selected text into the clipboard.
Paste	Paste the text on the clipboard into the field currently being edited.
Delete	Delete the currently selected text.

View menu

Menu Item	Description
Next Page	Go to the next page of the form.
Previous Page	Go to the previous page of the form.
Show Toolbar	Show/hide the toolbar

Zoom	Change the zoom level. The zoom levels are selected from a sub-menu. Zoom options are: • 25% • 50% • 75% • 100% • 150% • 200% • Fit page width • Fit page height

Help Menu

Menu Item	Description
About	Display version and license information for Formulator.

Tool Bar

The tool bar provides a set of buttons to access commonly used menu items.

The buttons from left to right are:

- Load form.
- Close form.
- Load data.
- Save data.
- Cut.
- Copy.
- Paste.
- Decrease zoom level.
- Increase zoom level.
- Go to the previous page of the form.
- Go to the next page of the form.

Between the previous page and next page button the current page and page count for the current form is displayed.

Form File Reference

This section describes the format of the form definition file.

A form file specifies the layout and behaviour of a form.

The form file is an XML document that contains embedded Lua scripts to control the behaviour of the form.

It is assumed that anyone reading this section is interested in creating their own forms. Currently forms can only be created by direct editing of the XML file in a text or XML editor.

Formulator is based on the "enough rope" principle. It is very flexible, but you can get yourself into trouble as Formulator will not protect you from doing something silly.

However, to assist in the development of forms, Formulator supports the "-console" command line option. This option creates a console window when Formulator starts. Additional error messages about form load errors are displayed in the console. Also, the output for Lua print statements will be displayed in the console window. This can be quite handy for debugging the scripts in forms.

The following typographic conventions are used in the description of the form file format:

Form Element	References to an element definition
`Form Text`	Literal text in the form file.
<variable>	A variable value in the form file.

In addition the following are used to denote optional, selection and repeating of elements:

Element?	An optional element
Element+	One or more repeats of Element
Element*	Zero or more repeats of Element
{Element A | Element B}	A selection of either Element A or Element B

The form file can be edited in any text or XML editor. It is recommended that an XML editor is used so that the form schema can be used to assist in writing the form layout.

A DTD that specifies the structure of the form file is provided (form_schema.dtd). This DTD can be used by an XML editor to check the validity of the form file.

Each form file consists of:

- Form configuration information
- A list of resources used in the form
- A list of page descriptions

Each page contains consists of:

- Static elements on the page
- Editable fields on the page
- Calculated fields on the page
- Custom drawn elements on the page

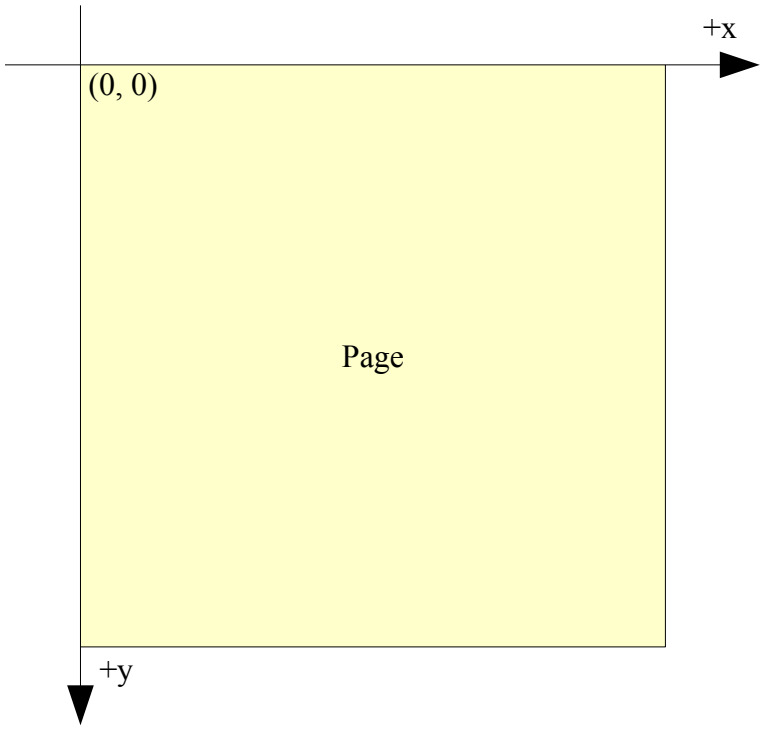
Form coordinate system

All form coordinates are specified in form measurement units (currently only millimetres are supported).

The top left corner of the page is (0, 0).

The X coordinate increases to the right.

The Y coordinate increases down the page.



File Format Reference

A form file has the following structure:

XML Header

Doctype

Element: form

A form file consists of the two header elements (the XML header and the Doctype) followed by a form element.

Form Header

XML Header

```
<?xml version="1.0" encoding="utf-8"?>
```

This is the standard XML file header. Formulator only supports UTF-8 XML documents.

Doctype

```
<!DOCTYPE form SYSTEM "form_schema.dtd">
```

This elements specifies the DTD that describes the form file layout. This is useful when editing a form file in an XML editor that understands DTD files. This element is optional.

Element: form

```
<form>
```

Element: tab_name

Element: units

Element: resources

Element: data_loadsave_scripts?

Element: page+

```
</form>
```

The form element is the top level XML element in the form file.

This element must contain a tab_name, units and resources element, plus one or more page elements.

This element may also contain an optional data_loadsave_scripts element.

Element: tab_name

```
<tab_name field="<field name>" format="<format string>"/>
```

This element specifies the text that appears on the form tab for the form.

The form tab text can be either a fixed string or a string based on the current value of a field in the form.

For a fixed string the field attribute must be set to “NULL”.

If a non “NULL” field is specified then the format string must contain a '%s' at the location to insert the text representation of the field value.

eg.

```
<tab_name field="player_name" format="Player: %s"/>
```

If the player_name field is currently “Fred” then the tab name for the form will be “Player: Fred”

Element: units

```
<units type="mm"/>
```

This element specifies the units used for all position values in the form. Currently the only supported units in “mm”.

Form Resource Specifications

Element: resources

```
<resources>
    Element: font_def*
    Element: pen_def*
    Element: bitmap_def*
</resources>
```

This element specifies the font, pen and bitmap resources that are available in the form.

Element: font_def

```
<font_def id="<resource id>">
  <name><font name></name>
  <size value="<font size>"/>
</font_def>
```

This element defines a font that is available for use in in the form.

The ***<resource id>*** is an integer. All resource ids must be unique but do not need to be contiguous.

The ***<font name>*** should match to a windows font name.

When designing a form for use by others you should consider which fonts are likely to be installed on other PCs. If you use an unusual font then consider providing the font with the form so it cam be installed. Formulator can only use installed fonts. Font files in the form directory cannot be used without being installed.

The ***<font size>*** is specified in form measurement units specified in the units element.

Element: pen_def

```
<pen_def id="<resource id>">
    Element: Colour
  <width value="<pen width>"/>
</pen_def>
```

The pen element defines a pen that can be used for drawing line based form elements.

A pen consists of a colour and a width in form measurement units.

Element: bitmap_def

```
<bitmap_def id="<resource id>">
  <file><bitmap file name></file>
</bitmap_def>
```

The bitmap_def element defines a bitmap that can be used for either static display or as the image for button form elements.

A bitmap consists of the path to a bitmap file. The file path may be either absolute or relative to the directory containing the form file. It is recommended to use relative file paths.

Remember that the bitmap files are not embedded in the form file. When distributing a form you must include all bitmap files used by the form file.

Element: data_loadsave_scripts

```
<data_loadsave_scripts>
   <Lua code>
</data_loadsave_scripts>
```

The data_loadsave_scripts element allows scripts to be defined to provide loading and saving of form data in a custom format.

The Lua code in this element must[1] define the following functions:

- form.load_data(filename)
- form.save_data(filename)

You have full access to the editable field contents from the 'form' table. You also have access to the Lua file I/O functions.

However, you are on your own from here.

The custom load/save scripts is a function for advanced users. Writing these functions requires more programming knowledge than I am willing to cover in this manual.

1 This is not entirely true. The presence of the data_loadsave_scripts element in the form tells Formulator to use “form.load_data” and “form.save_data” to load and save the form data respectively. However, these functions can be defined in any code block in the form. It is usually more convenient and logical to put the code for these function in the code block for this element.

Form structure elements

The following elements are used to define the structure of a form.

Element: page

<page width="***<page width>***" height="***<page height>***">

Element:

{ group | image | text | lines | rectangle | fill_rect | round_rect | fill_round_rect |

ellipse | fill_ellipse | edit_field | calc_field | custom_field | code | control_button | array }+

</page>

The page element defines a page in the form.

The ***<page width>*** and ***<page height>*** are specified in form measurement units.

Each page consists of a list of form elements that specify the page contents.

Element: group

<group x="***<group x offset>***" y="***<group y offset>***">

Element:

{ group | image | text | lines | rectangle | fill_rect | round_rect | fill_round_rect |

ellipse | fill_ellipse | edit_field | calc_field | custom_field | code | control_button | array }+

</group>

The group element is used to define a set of form elements that can be moved relative to the parent element (either a page, array or another group).

Element: array

<array index_name="***<name>***">

Element: index_list

Element:

{ group | image | text | lines | rectangle | fill_rect | round_rect | fill_round_rect |

ellipse | fill_ellipse | edit_field | calc_field | custom_field | code | control_button }+

</array>

The array element provides a method for defining a repeating set of form elements at different positions.

The index_name attribute of the array element may be any valid Lua identifier. Note: The index_name is currently unused, but may be used in a future version of Formulator.

The array element contains an index_list element followed by a list of form elements. The form elements are duplicated for each index in the index list at the x and y offsets specified in the index list.

Element: index_list

```
<index_list>
```

Element: index+

```
</index_list>
```

The index_list element specifies the list of elements in an array.

Element: index

```
<index value="<index name>" x="<offset x>" y="<offset y>" />
```

The index element specifies the name and location of an element in an array of form elements.

The value attribute must specify a valid Lua identifier (it must start with a letter).

The offset x and offset y attributes specify the location, relative to the parent element, of the copy of the form elements contained in the array element.

Form static elements

The static elements are used to describe the layout and appearance of the static parts of a form.

All static elements are drawn in the order in which they appear in the form file. This means then, where static elements overlap, static elements later in the form file will be drawn over the top of static elements earlier in the form file.

Element: text

```
<text>
```

Element: position

Element: align?

Element: font

Element: colour

Element: txt

```
</text>
```

The text element displays a fixed text string in the form.

The “position”, “align”, “font” and “colour” elements define where and how the text is displayed.

The text to be drawn is in the “txt” element.

Element: image

```
<image>
  <bitmap id="<resource id>"/>
```

Element: rect

```
</image>
```

The image element draws a bitmap resource at a fixed position on the form.

The ***<resource id>*** must be the id of a bitmap resource defined in the resources section of the form.

The rect element defines where the bitmap is to be drawn. The rectangle is specified relative to the parent element in the form. The bitmap is scaled to fit the specified rectangle. Aspect ratio of the original bitmap will not be preserved.

Element: round_rect

```
<round_rect>
```

Element: pen

Element: rect

Element: corner

```
</round_rect>
```

The round_rect element draws a rectangle with rounded corners on the form.

Only the outline of the rounded rect is drawn.

Element: fill_round_rect

```
<fill_round_rect>
```

Element: pen

Element: colour

Element: rect

Element: corner

```
</fill_round_rect>
```

The fill_round_rect element draws a rectangle with rounded corners filled with a solid colour.

Element: rectangle

```
<rectangle>
```

Element: pen

Element: rect

```
</rectangle>
```

The rectangle element draws a rectangle on the form.

Only the outline of the rectangle is drawn.

Element: fill_rect

```
<fill_rect>
```

Element: pen

Element: colour

Element: rect

```
</fill_rect>
```

The fill_rect element draws a rectangle filled with a solid colour.

Element: ellipse

```
<ellipse>
```

Element: pen

Element: rect

```
</ellipse>
```

The ellipse element draws an ellipse with the pen resource specified by the pen element.

The ellipse horizontal and vertical limits are specified by the rect element.

Element: fill_ellipse

```
<fill_ellipse>
```

Element: pen

Element: colour

Element: rect

```
</fill_ellipse>
```

The fill_ellipse element draws an ellipse with the pen resource specified by the pen element filled with the colour specified by the colour element.

The ellipse horizontal and vertical limits are specified by the rect element.

Element: Lines

```
<lines>
```

Element: pen

Element: point_list

```
</lines>
```

The lines element draws a set of lines specified by the point_list element using the pen resource specified by the pen element.

Form Field elements

The field elements are used to define the dynamic components of the form.

There are three primary types of fields:

- Edit fields
  These are fields that have user editable contents
- Calculated fields
  These are fields whose contents are calculated from either edit fields and/or other calculated fields.
- Custom Fields
  These are fields whose contents are custom drawn using a Lua script. These fields may be either static (useful for changing static text in arrays) or based on the data entered or calculated in other fields.

In addition Formulator forms also provide push button controls. These are buttons on the form that trigger a Lua script when pressed.

Element: edit_field

The edit_field has three variants depending on the type of data the field edits.
For text, int or real fields:

```
<edit_field name="<field_name>" type="<field_type>">
```

Element: rect

Element: justify

Element: font

Element: colour

Element: format?

Element: initial

```
</edit_field>
```

For boolean edit fields:

```
<edit_field name="<field_name>" type="bool">
```

Element: rect

Element: unchecked

Element: checked

Element: initial

```
</edit_field>
```

For bitmap fields:

```
<edit_field name="<field_name>" type="bitmap">
```

Element: rect

Element: initial

```
</edit_field>
```

The edit_field element defines an editable value in a form. The edit field element has two attributes:

- field_name
  This attribute specifies the name of the field in the form. Each edit field will create an entry in the Lua form table with the name specified by the field_name attribute. If the edit field is in an array then the form table entry will be a Lua table with its entries indexed by the table index values.
- type
  This attribute specifies the data type stored in the edit field. This must be one of: “text”, “int”, “real”, “bool” or “bitmap”.

There are three variations of the edit field depending on the method of editing the value in the edit field.

The first variant is for all fields whose values are edited as text. The “justify”, “font” and “colour” elements control the appearance of the text value displayed in the edit field. The format element is only used for real fields and must be a C format string for a floating point number. The default format string is “%1.1f”.

The second variant is for fields with boolean values. The initial value for this field must be either 0 (for false) or 1 (for true).

The third variant is for bitmap fields. The initial value for the bitmap field is the file name for the bitmap. Note that bitmaps in a bitmap edit field will be displayed scaled to fit the field rectangle, but the aspect ratio of the original bitmap image will be preserved. This may leave empty space left/right or above/below the image depending on the aspect ratio of the field relative to the aspect ratio of the bitmap. Note that this differs from the display of static bitmaps that are scaled to fill the rectangle of the static element.

Element: calc_field

<calc_field name="**<field_name>**" type="**<field type>**">

Element: rect

Element: justify

Element: font

Element: colour

Element: format?

Element: depend

Element: script

</calc_field>

The calc_field element defines a field whose value is calculated from the value of other fields in the form.

The ***<field_type>*** specifies the type of data for this field and must be one of:

- text
- int
- real

The “rect”, “justify”, “font”, “colour” and “format” elements define the position and appearance of the field in the form.

The “format” element is only used in fields of type “real”

The “depend” element must list all of the fields whose values are used in the calculation of this field. If the depend list is incorrect then this field may not updated as required. A calc_field may depend on any number of edit fields and/or calc fields.

The “script” element must define the function “form.calc_***<field_name>***”[2].

If this field is not a member of an array then this function takes no parameters.

If this field is an element of an array then this function takes the array index as a parameter.

The script must set the value of the ***<field_name>*** element in the form table.[3]

For a calc_field in an array the function must create the table for the array of calc fields if it does not already exist.

Example 1:

For a calc_field with the field_name “area” in a form with edit fields “width” and “height”, the script might be:

```
function form.calc_area()
  form.area = form.width * form.height
end
```

Example 2:

As above, but in an array.

```
function form.calc_area(idx)
  if (form.area == nil) then
    form.area = {}
  end
  form.area[idx] = form.width[idx] * form.height[idx]
end
```

Element: custom_field

<custom_field name="***<field_name>***">

Element: rect

Element: depend

Element: script

</custom_field>

The custom_field is used to draw data dependent imagery onto a form.

The “rect” element defines the bounding box of the field to be drawn.

2 This is not strictly true. While it is more logical to put the script for a calc_field in the script element of the calc_field this is not actually required. So long as the required function is defined somewhere in the form then Formulator will be happy.

3 This is also not strictly true. It usually makes sense to do it this way, but there are times where it may be convenient to calculate the values for multiple calc_fields in a single script. This can be done. Just set one calc_field to have the required dependencies and perform the calculations for all calc_fields that you want to calculate together. Then set the other calc fields to have a dependency on the calc field that does the calculations. These calc_fields can now have empty functions for their scripts.

The “depend” element lists the fields whose data is used in the drawing of the custom field.

The “script” element defines the script to draw the field.

The script for the custom field must define the function “form.draw_***<field_name>***”.

If this field is not a member of an array then this function takes no parameters.

If this field is an element of an array then this function takes the array index as a parameter.

Element: control_button

`<control_button name="`***<control_name>***`">`

Element: rect

Element: up_image

Element: down_image

Element: updates

Element: script

`</control_button>`

The control_button element is used to define a button control in the form.

The button control is a rectangular area of the form specified by the rect element that, when clicked, triggers the execution of a Lua script associated with the button.

The button control only supports drawing the button using a bitmap resource. A separate bitmap resource is used for the button up and button down image.

The button control script element must define the Lua function “form.***<control_name>***.pressed()”

If the button control is not in an array element then this function takes no parameters.

If the button control is in an array element then this function takes a single parameters, the array index.

The Lua script can update one or more editable fields. The list of fields updated by the button must be specified in the “updates” element.[4]

Element: code

`<code>`

`<Lua_code>`

`</code>`

The code element is used to define arbitrary Lua code that is executed when the form is loaded. This can perform initialisation of data or define functions to be used in the form.

4 Another statement that is not entirely true. The script can achieve the same effect by calling the FormUpdateField() function to notify Formulator that a field's value has been updated by the script.

Form Data Elements

This section describes the data elements that are used in the definition of the structure, static and field elements in the form.

Element: rect

`<rect left="<left>" top="<top>" right="<right>" bottom="<bottom>" >`

The rect element is used to define a rectangular region on the form.

The rectangle coordinates are always relative to the parent element.

Element: pen

`<pen id="<pen id>"/>`

The pen element is used to select the pen to use for drawing lines. Lines may only be drawn in lines styles defines in the resources section of the form.

The ***<pen id>*** must the the resource number of a pen resource defined in the resources section of the form.

Element: corner

`<corner width="<width>" height="<height>"/>`

The corner element is used to describe the size of corners of a round rectangle. The width and height specify the width and height of the ellipse that is used to draw the corners of the round rectangle.

Element: position

`<position x="<X pos>" y="<Y pos>"/>`

The position element is used to specify the position of text in a static text element.

Element: align

`<align h="<horizontal alignment>" v="<vertical alignment>"/>`

The align element is used to specify the alignment of text relative to the static text item's coordinates specified in the position element.

Allowed horizontal alignment values are:

- left
- centre
- right

Allowed vertical alignment values are:

- top
- centre

- bottom

Element: font

`<font id="<font_resource_id>"/>`

The font element is used to select a font resource from the resources in the form.

The ***<font_resource_id>*** must be the integer id of a font resource.

Element: txt

`<txt><text to display></txt>`

The txt element defines the text to display for a static text element.

Element: point_list

`<point_list>`

Element: (point+)

`</point_list>`

The “point_list” element is used to specify the points for the lines drawn by a “lines” element.

The “point_list” must contain at least two points.

Element: point

`<point x="<X pos>" y="<Y pos>"/>`

The point element is used to define a point in a point list.

Element: justify

`<justify h="<horizontal justification>" v="<vertical justification>"/>`

The justify element is used to specify the alignment of text relative the edit field item's bounding rectangle.

Allowed horizontal justification values are:

- left
- centre
- right

Allowed vertical justification values are:

- top
- centre
- bottom

Element: initial

`<initial value="<initial_value>"/>`

The initial element specifies the initial value for an edit field.

Element: depend

```
<depend>
```

*Element: field**

```
</depend>
```

The depend element defines a list of fields that another form element depends upon. The depend list is used to determine when fields need to be recalculated or redrawn.

The depend lists must be set correctly for a form to behave correctly.

Element: updates

```
<updates>
```

*Element: field**

```
</updates>
```

The updates element defines a list of fields that are updated by this element.

Element: field

```
<field name="<field_name>"/>
```

The field element specifies the name of a field in a dependency list or an updates list.

Element: script

```
<script>
<Lua script>
</script>
```

The script element defines the Lua script associated with the parent item.

The content requirement of the Lua script depends on the parent element.

Element: unchecked

```
<unchecked id="<bitmap_resource_id>"/>
```

The unchecked element specifies the bitmap resource to be drawn for a boolean edit field when its value is false.

Element: checked

```
<checked id="<bitmap_resource_id>"/>
```

The checked element specifies the bitmap resource to be drawn for a boolean edit field when its value is true.

Element: up_image

`<up_image id="`***<bitmap resource id>***`"/>`

The up_image is used to specify the button up image for a button control.

Element: down_image

`<down_image id="`***<bitmap resource id>***`"/>`

The down_image is used to specify the button down image for a button control.

Lua Scripting Reference

Formulator uses Lua 5.1 to provide scripting of form behaviour.

The following Lua standard libraries are available to Lua scripts in forms:

- base Provides basic Lua functions
- math Provides maths functions
- os Provides operating specific functions
- io Provides file input/output functions
- string Provides string manipulation functions

Details of the functions provided by these libraries can be found in the Lua 5.1 reference manual.

In addition to the library functions additional Formulator specific functions are available.

The functions are divided into the following groups:

- Drawing Functions
  These functions allow for drawing of content into custom fields
- XML file reading functions
  These functions allow for parsing an XML file and traversing the XML file as a tree
- Directory Functions
  These functions are for listing all files in a directory
- Form Control functions
  These functions provided for control over loading forms and loading and saving data.

Form Scripting Overview

Each form has its own Lua state.

Additional functions and variables may be added to the Lua state using the code elements of a form.

The form table

Each form defines a table in Lua at the global scope named 'form'.

The 'form' table contains both the scripts and values for fields in the form.

Each calculated field, custom field and button control must define one or more functions in the "form" table to handle its processing. The names and parameters of these functions must follow the convention require by the field type. See the form file definition section for details.

The current value of each field (editable or calculated) defined in the form is stored in an element of the 'form' table with the field name and has the same type as the field.

Formulator does not automatically create the for table entries for calculated fields. This must be done by the calculation scripts for the calculated fields.

If the value of a calculated field is not created by the script then the form will display "NIL" as the value for the field.

Special care must be taken with arrays of calculated fields to ensure that the table for the array of

calculated field values is created before setting the value of any element of the array.

All script functions and variables that are directly associated with form elements are elements of the 'form' table.

For example, if a form defines editable real number fields with the names “width” and “height” then a Lua script can calculate the area from the currently entered values as:

```
area = form.width * form.height
```

Drawing Functions

The drawing functions are provided to draw the contents of custom fields.

Drawing functions should only be called in the script for custom fields.

The drawing behaviour of these functions if called outside of a custom field script is undefined.

The coordinates of all drawing functions are in 0.001 mm units relative to the top left corner of the custom field.

The X coordinate increases to the right, the Y coordinate increases down the page.

Function: DrawLine

Draws a line between two points using the specified pen.

Syntax

```
DrawLine(PenId, StartX, StartY, EndX, EndY)
```

Parameters

PenId	The resource id of the pen to use.
StartX	The X coordinate for the start of the line.
StartY	The Y coordinate for the start of the line.
EndX	The X coordinate for the end of the line.
EndY	The Y coordinate for the end of the line.

Return Values

None

Function: DrawRect

Draw an unfilled rectangle using the specified pen.

Syntax

```
DrawRect(PenId, StartX, StartY, EndX, EndY)
```

Parameters

PenId	The resource id of the pen to use.
StartX	The X coordinate for the start of the rectangle.
StartY	The Y coordinate for the start of the rectangle.
EndX	The X coordinate for the end of the rectangle.
EndY	The Y coordinate for the end of the rectangle.

Returns

None

Function: DrawEllipse

Draw an unfilled ellipse inscribed within the specified bounding rectangle.

Syntax

DrawEllipse(PenId, StartX, StartY, EndX, EndY)

Parameters

PenId	The resource id of the pen to use.
StartX	The X coordinate for the start of the bounding rectangle.
StartY	The Y coordinate for the start of the bounding rectangle.
EndX	The X coordinate for the end of the bounding rectangle.
EndY	The Y coordinate for the end of the bounding rectangle.

Returns

None.

Function: FillRect

Draw a filled rectangle using the specified pen for the outline and specified fill colour.

Syntax

```
FillRect(PenId, FillColour, StartX, StartY, EndX, EndY)
```

Parameters

PenId	The resource id of the pen to use.
FillColour	The fill colour to use. Each of the Red, Green, Blue components of the colour must be in the range 0 .. 255. The FillColour value is Red + Green * 256 + Blue * 65536. The FillColour can be specified as a Hexadecimal value 0xBBGGRR.
StartX	The X coordinate for the start of the rectangle.
StartY	The Y coordinate for the start of the rectangle.
EndX	The X coordinate for the end of the rectangle.
EndY	The Y coordinate for the end of the rectangle.

Returns

None.

Function: FillEllipse

Draw a filled ellipse.

Syntax

```
FillEllipse(PenId, FillColour, StartX, StartY, EndX, EndY)
```

Parameters

PenId	The resource id of the pen to use.
FillColour	The fill colour to use. Each of the Red, Green, Blue components of the colour must be in the range 0 .. 255. The FillColour value is Red + Green * 256 + Blue * 65536. The FillColour can be specified as a Hexadecimal value 0xBBGGRR.
StartX	The X coordinate for the start of the rectangle.
StartY	The Y coordinate for the start of the rectangle.
EndX	The X coordinate for the end of the rectangle.
EndY	The Y coordinate for the end of the rectangle.

Returns

None.

Function: FillPie

Draw a filled pie segment.

The pie segment is of an ellipse defined be a bounding rectangle. The arc segment drawn is specified by a start angle and end angle in degrees. The angle 0 degrees is up, positive angles are clockwise and negative angles are counter clockwise. The arc is drawn in the clockwise direction from start angle to end angle.

Syntax

```
FillPie(PenId, FillColour, StartX, EndX, StartY, EndY, StartAngle, EndAngle)
```

Parameters

PenId	The resource id of the pen to use.
FillColour	The fill colour to use. Each of the Red, Green, Blue components of the colour must be in the range 0 .. 255. The FillColour value is Red + Green * 256 + Blue * 65536.

	The FillColour can be specified as a Hexadecimal value 0xBBGGRR.
StartX	The X coordinate for the start of the rectangle.
StartY	The Y coordinate for the start of the rectangle.
EndX	The X coordinate for the end of the rectangle.
EndY	The Y coordinate for the end of the rectangle.
StartAngle	The start angle of the ellipse in degrees.
EndAngle	The end angle of the ellipse in degrees.

Returns

None.

Function: DrawText

Draw a text string.

Syntax

```
DrawText(FontId, Colour, x, y, HorizontalAlign, VerticalAlign, Text)
```

Parameters

FontId	The resource id of the font to use.
Colour	The colour to draw the text. Each of the Red, Green, Blue components of the colour must be in the range 0 .. 255. The Colour value is Red + Green * 256 + Blue * 65536. The Colour can be specified as a Hexadecimal value 0xBBGGRR.
x	The x position for the text.
y	The y position for the text.
HorizontalAlign	This is either an integer or a string value indicating the horizontal alignment. If this value is an integer it must be either 0, 1 or 2. If this value is a string it must be either “LEFT”, “CENTRE” or “RIGHT”. • 0, “LEFT” = Align left. The x coordinate specifies the left edge of the text. • 1, “CENTRE” = Align centre. The x coordinate specifies the horizontal centre position of the text. • 2, “RIGHT” = Align right. The x coordinate specifies the right edge of the text.
VerticalAlign	This is either an integer or string value indicating the vertical alignment.

	This if this value is an integer it must be either 0, 1 or 2. If this value is a string it must be either “TOP”, “CENTRE” or “BOTTOM”. • 0, “TOP” = Align top. The y coordinate specifies the top edge of the text. • 1, “CENTRE” = Align centre. The y coordinate specifies the vertical centre position of the text. • 2, “BOTTOM” = Align bottom. The y coordinate specified the bottom edge of the text.
Text	This is the text string to be drawn.

Returns

None.

Function: DrawBitmap

Draws a bitmap resource. The bitmap is scaled to fit the specified rectangle. Aspect ratio is not preserved.

Syntax

```
DrawBitmap(BitmapId, Left, Top, Right, Bottom)
```

Parameters

BitmapId	The resource id of the bitmap to draw.
Left	The left edge of the destination area.
Top	The top edge of the destination area.
Right	The right edge of the destination area.
Bottom	The bottom edge of the destination area.

Returns

None

XML File Reading Functions

This group of functions provides an XML file parser to Lua functions in Formulator scripts.

The XML file method provide a stack based navigation of the XML file structure.

Function: XMLOpenFile

Open an XML file.

Syntax

```
FileHandle = XMLOpenFile(Filename)
```

Parameters

Filename	The name of the XML file to open.

Returns

Filehandle	int	The file handle of the opened file. The return value will be 0 if the file open failed.

Function: XMLCloseFile

Closes a previously opened XML file

Syntax

```
XMLCloseFile(FileHandle)
```

Parameters

FileHandle	The handle of the XML file to close.

Returns

None

Function: XMLFirstChildElement

Sets the current element to the first child element with the specified name contained within the current file element.

Syntax

```
Found = XMLFirstChildElement(FileHandle, ElementName)
```

Parameters

FileHandle	The file handle of the XML file as previously returned by XMLOpenFile().
ElementName	The name of the child element to find.

Returns

Found	bool	true if the element was found, otherwise false.

Function: XMLNextSiblingElement

Sets the file position to the next sibling element with the specified name.

Syntax

```
Found = XMLNextSiblingElement(FileHandle, ElementName)
```

Parameters

FileHandle	The file handle of the XML file as previously returned by XMLOpenFile().
ElementName	The name of the element.

Returns

Found	bool	true if the element was found, otherwise false.

Function: XMLGetText

Gets the element text of the current XML element.

Syntax

```
Text = XMLGetText()
```

Parameters

FileHandle	The file handle of the XML file as previously returned by XMLOpenFile().

Returns

Text	string	The element text. If the current element has no element text then this will be an empty string.

Function: XMLPop

Sets the current file position to the parent of the current element

Syntax

```
XMLPop()
```

Parameters

FileHandle	The file handle of the XML file as previously returned by XMLOpenFile().

Returns

None

Function: XMLGetAttribute

Gets the value of an attribute of the current XML element

Syntax

```
Found, Text = XMLGetAttribute(FileId, AttributeName)
```

Parameters

FileHandle	The file handle of the XML file as previously returned by XMLOpenFile().
AttributeName	The name of the attribute to read.

Returns

Found	bool	This will be true if the attribute was found in the current element, otherwise false.
Text	string	The text of the attribute.

Directory Functions

Function: OpenDir

Open a directory for scanning directory entries.

There is a limitation in the directory functions that one directory can be open at a time across all forms. This means that a form that uses the directory functions should open, read and process the directory, and close the directory all within a single event script.

If the directory calls are spread across multiple event scripts then the behaviour is undefined.

Syntax

```
OpenDir(DirectoryName)
```

Parameters

DirectoryName	The path of the directory to open.

Returns

None

Function: CloseDir

Close the currently open directory.

Syntax

```
CloseDir()
```

Parameters

None

Returns

None

Function: ReadDir

Reads then next entry in the open directory.

Syntax

```
EntryName, FileType = ReadDir()
```

Parameters

None

Returns

EntryName	string	Directory entry name
FileType	string	The file type. Will be one of: “file”, “dir”, “end”.

		• “file” indicates that the Entry Name is the name of a file • “dir” indicates that the Entry Name is the name of a directory. • “end” indicates that all entries in the directory have been read.

Function: SplitPath

Split a file path into the directory path and file name.

Syntax

```
DirPath, Filename = SplitPath(FilePath)
```

Parameters

FilePath	The file path to be split into the directory path and file name components.

Returns

DirPath	string	The directory path component of FilePath.
Filename	string	The file name component of FilePath.

Form Control Functions

The form control functions provide the ability to open forms and form data, manipulate form field status and control input handling.

Function: CloseForm

Close the current form.

Syntax

```
CloseForm()
```

Parameters

None

Returns

None.

Function: OpenForm

Open a Formulator form in a new tab.

Syntax

```
OpenForm(FormFilename)
```

Parameters

FormFilename	The name of the form file to load.

Returns

None

Function: OpenFormAndData

Open a form and load the form data.

Syntax

```
OpenFormAndData(FormFilename, DataFilename)
```

Parameters

FormFilename	The name of the form file to load.
DataFileName	The name of the form data file to load into the new form.

Returns

None

Function: LoadFormData

Load new form data into the current form.

Syntax

```
LoadFormData(FormDataFile)
```

Parameters

FormDataFile	The name of the form data file to load into this form.

Returns

None

Function: FormUpdateField

Trigger an update to the display of a field in the form.

For a calculated field this triggers an immediate re-calculation of the field.

For an edit field this causes the form to update the display of an edit field in the form to reflect the current value of the field in the Lua state. Normally Formulator updates the Lua state to reflect the value entered into the form. This triggers the reverse operation. The value in the form is updated to reflect the current value in the Lua state.

For a custom field this triggers a re-draw of the custom field.

Syntax

FormUpdateField(FieldName)

Parameters

FieldName	The name of the field to update.

Returns

None

Function: FormMarkFieldUpdated

This function marks another field in this form as updated.

This function is intended to be used on calculated fields but can be called for all field types.

For a calculated field, this will prevent the calculated field's script from being called, even if the current script has updated the dependencies of the calculated field.

For edit fields it will trigger a re-calculation of all fields the depend on that edit field without changing the value of the edit field.

For custom field this function has the same effect as FormUpdateField().

This can be used to prevent another calculated field's script from being triggered when the script updates the edit fields that the calculated field depends upon.

This can be used to implement complex form update effects such as bi-directional conversion (two edit fields, editing the value in one updates the value in the other).

To implement bi-directional conversion, create two edit fields and two hidden calculated fields, each dependent on one of the edit fields.

A hidden calculated field can be created by setting the field position off the page. Use of (-999, -999) is recommended.

Each calculated field updates the value of the edit field it does not depend upon based on the value of the edit field it does depend upon. Normally this will trigger the calculation script of the other calculated, leading to potential problems with rounding errors as both scripts are executed. To avoid this, call FormMarkFieldUpdated() on the other calculated field to prevent its script from being called, even though its dependencies have been updated.

Eg: Celsius/Fahrenheit conversion

Two real edit fields: Fahrenheit, Celsius.

Calculated field toFarenheit depends on Celsius and has the script:

```
form.toFahrenheit = 1
function form.calc_toFahrenheit()
  form.Fahrenheit = (form.Celsius * 9 / 5) + 32
  -- Tell Formulator we have updated Fahrenheit
  FormUpdateField("Fahrenheit")
  -- Prevent toCelsius from being triggered by the update to Fahrenheit as
  -- editing Celsius triggered this update in the first place
  FormMarkFieldUpdated("toCelsius")
end
```

Calculated field toCelsius depends in Fahrenheit and has the script:

```
form.toCelsius = 1
function form.calc_toCelsius()
  form.Celsius = (form.Fahrenheit - 32) * 5 / 9
  -- Tell Formulator we have updated Celsius
  FormUpdateField("Celsius")
  -- Prevent toFahrenheit from being triggered by the update to Celsius as
  -- editing Fahrenheit triggered this update in the first place
  FormMarkFieldUpdated("toFahrenheit")
end
```

Syntax

```
FormMarkFieldUpdated(FieldName)
```

Parameters

FieldName	The name of the field to mark as updated.

Returns

None

Function: FormMessageBox

Display a message box.

Syntax

FormMessageBox(Message, Title, Flags)

Parameters

Message	The message text to be display in the message box.
Title	The window title for the message box.
Flags	The message box flags. This is a string containing the message box flags. The flags string can contain one buttons control flag plus one icon display flag. The flags must be separated by a '|' character in the flags string. The possible button control flags are: • MB_OKCANCEL • MB_OK • MB_YESNOCANCEL • MB_YESNO The possible icon display flag are: • MB_ICONERROR • MB_ICONQUESTION • MB_ICONINFORMATION

Returns

None

Function: FormGrabKeyboard

Redirects all keyboard input to a Lua function.

This will disable editing of fields using the keyboard.

Syntax

```
FormGrabKeyboard(KeyboardHandler)
```

Parameters

KeyboardHandler	The name of the Lua function that is to handle key presses. This function takes a single integer parameter. The parameter is the windows virtual key code for the key that has been pressed.

Returns

None

Function: FormReleaseKeyboard

Releases the keyboard handling back the the standard form keyboard handler.

Syntax

```
FormReleaseKeyboard()
```

Parameters

None

Returns

None