# Xaml Form Creator

## ****General Description:****

This custom activity receives XAML mark-up and launches a WPF form.  Input values passed to the activity are used to initialize the elements in the form and, on submit, the data is scraped and passed on as an OutArgument.

The user has control over which event in the page will trigger the submit of the form and over the information retrieved in the Output.

## ****Parameters:****

### Input:

* FormXAMLPath - **Path to the file containing the XAML Markup used in rendering the form.**
  + Required Argument
  + This is a XAML file with a Grid as its parent element which will contain all the elements of our current page
  + This form needs to follow this format:

**XAML File Format**

<Grid xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:xctk="http://schemas.xceed.com/wpf/xaml/toolkit">

<!-- Custom XAML Markup goes here -->

</Grid>

* StyleSheetPath - **Path to the file containing the Resource Dictionary used in styling the form.**
  + Optional
  + This is a XAML file with a ResourceDictionary as its parent element which will contain all the globally applied styles
  + Referencing a style needs to be done in the main form with **Style="{DynamicResource KeyName}"  (**check Example 2 for details)
  + This form needs to follow this format:

**StyleSheet Format**

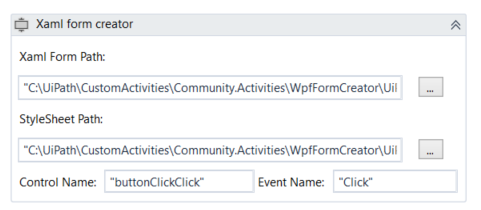
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

 <!-- Custom XAML Markup goes here -->

</ResourceDictionary>

* SubmitElementName (string) - **Name of the Control on which we perform the submit event.**
  + Required Argument
* SubmitEventName  (string) - **Name of the Event on which we perform the submit (e.g. click).**
  + Required Argument
* InputDictionary  (Dictionary<string, Dictionary<string, object>>) - **Initialization values for all the framework elements in the form. By default, this also describes the properties which will be retrieved at submit.**
  + Optional
  + If GetAllProperties is left unchecked, the InputDictionary is a required argument
* GetAllProperties  (CheckBox)- **If this is checked, the submit event will return all non-null properties of all the framework elements mentioned in the Elements To Retrieve array.**
  + Optional
* ElementsToRetrieve  (string[])- **Array with the names of the elements for which, at submit, we should retrieve the all the non-null properties.**
  + Optional
  + This property should only be used when GetAllProperties is checked, in this case, the ElementsToRetrieve is a **Required Argument**



### Output:

* OutputDictionary  (Dictionary<string, Dictionary<string, object>>)  - **The values returned when the form is submitted**
  + if **GetAllProperties**is unchecked, the Output Dictionary will have the same structure as the Input dictionary, so only the properties that were specified in the **InputDictionary** will be retrieved
  + if **GetAllProperties**is checked, then all the properties from the elements named in **ElementsToRetrieve**will be retrieved instead

## ****General Description:****

The Input dictionary has the following structure :



Basically, for each control, we have a dictionary with the values for each property (like **Text** for Textboxes, **IsChecked** for CheckBoxes, Visibility, etc.) and all these values will be set to their corresponding elements at the launch of the form.

The form will be closed either when pressing the red "X" button or when activating the **Submit** event. The Submit event is configured by the user and it can be picked out of all the events which could normally happen inside a WPF form. The activity receives as an input parameter the name of the control on which this event will occur (for example: we can give the name of a button, a checkbox or any of the elements present in the page) and the Event Name (for example: "Click"). Therefore, when the specified event occurs on the control the form is closed and the custom activities returns the **OutputDictionary**to the workflow and the execution is resumed.

For the content of the **OutputDictionary,** there are 2 posiblilties:

* by default, the Output Dictionary will have the exact same structure and properties as the Input dictionary, so it will contain the exact same properties and will retrieve their values at submit (so if the user changes the value of a text property, the output dictionary will contain the updated value). One key thing to note is that since the output and input dictionaries have the same structure, it is easy to reopen the form and directly use the dictionary obtained from the previous form as input
* if **GetAllProperties**is checked, then all the properties from the elements named in **ElementsToRetrieve** array will be retrieved instead, so for each of the elements named there we will have a dictionary with all their properties and their values. (This means that a lot of data that might not be necessary will also be retrieved, this is why this feature is disabled by default)

## ****Observations and limitations:****

* If you want to include the XAML files used to create the forms in an UiPath project, please  change their extension from .xaml to .xml, otherwise you won't be able to publish the project
* The output dictionary will only be filled if the user presses the submit button, otherwise, the OutputDictionary will be null
* Referencing a style needs to be done in the main form with **Style="{DynamicResource KeyName}"**

## ****Examples:****

**Download the following project in order to better see the examples:**



### Example #1 - Simple Form:

Let us assume that were building a front-office robot and that we want to launch a form in which the current user will fill in details about a certain issue. We want to assign an owner for the issue (which by default should be the current user) fill in a ticket number, a field where the issue is described and a field for detailed comments.

#### ****Solution:****

**Open the Example1.xaml in order to see the complete solution. Read the indications below for the detailed explanations.**

##### Mark-up for the form

For Starters, lets design our WPF form. After creating some basic XAML mark-up, we'll end up with something like this:

**Example 1 - XAML Mark-up**

<Grid xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:xctk="http://schemas.xceed.com/wpf/xaml/toolkit"

Margin="10" Width="300" Height="300" >

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto" />

<ColumnDefinition Width="\*" />

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="\*" />

<RowDefinition Height="Auto" />

</Grid.RowDefinitions>

<Label>Owner:</Label>

<TextBox Grid.Column="1" Margin="0,0,0,10" Name="owner"/>

<Label Grid.Row="1">Ticket Number:</Label>

<TextBox Grid.Row="1" Grid.Column="1" Margin="0,0,0,10" Name="ticketnumber" />

<Label Grid.Row="2">Issue:</Label>

<TextBox Grid.Row="2" Grid.Column="1" Margin="0,0,0,10" Name="issue" />

<Label Grid.Row="3">Comment:</Label>

<TextBox Grid.Row="3" Grid.Column="1" AcceptsReturn="True" Name="comment" />

<Button Grid.Row="4" Grid.Column="1" Margin="5" HorizontalAlignment="Center" Name="submitButton" Width="100" Height="30" xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">Submit</Button>

</Grid>

Now we'll save this as an xml file inside our UiPath project ("Example1\TestFiles\ParentForm\_5.xml"). Note that we added all our content inside a <Grid> element. Also note that all the textboxes have names which the robot will use to idetify them in the page when assigning them values and when retrieving data from them.

We also added a button that has the name "submitButton", we'll configure our activity so that when the user clicks on this button, the form is submitted.

As input dictionary, we'll pass the following Dictionary:

**Input Dictionary**

New Dictionary( Of String, Dictionary( Of String, Object)) From {

{ "owner", New Dictionary( Of String, Object) From { { "Text", "Radu Bucur" } } },

{ "ticketnumber", New Dictionary( Of String, Object) From { { "Text", "" } } },

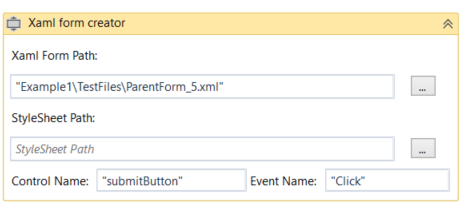
{ "issue", New Dictionary( Of String, Object) From { { "Text", "" } } },

{ "comment", New Dictionary( Of String, Object) From { { "Text", "" } } }}

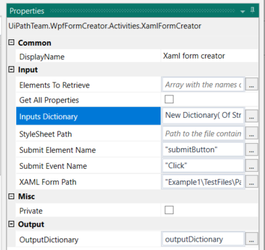
Right now only the "owner" textbox has a hard-coded value (in a real-world scenario this would be a dynamic value) and the other fields are empty. We could omit these fields from the dictionary, but we want them there because we want the **OutputDictionary** to contain the Text property for all those textboxes.

##### Activity

After all these configurations, our Xaml Form Creator will look like this:

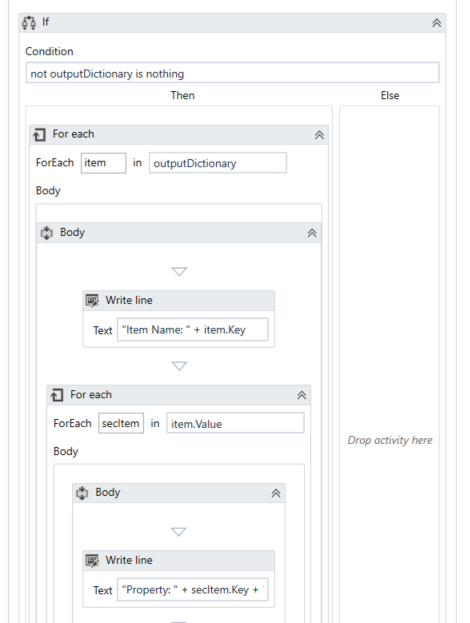


And our properties panel will look like this:



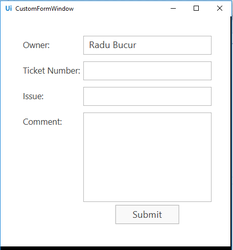
##### Testing the Activity

In order to display the Output of the activity, we'll use the following code which loops through the controls inside the OutputDictionary and their properties and outputs them to the Output Window:

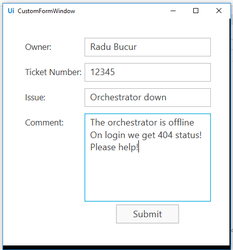


##### End Result

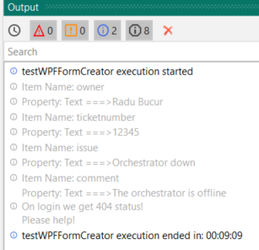
If you start the process **Example\Example1.xaml**, the form will look something like this:



Now, let us Start the process and fill in the form:



After we press submit, the process will end and if we check the Output Window, we can actually see that the OutputDictionary actually contains the text values inside all the textboxes in our form:



### Example #1.b - Adding a style sheet:

Let us use the code from example #1 and add a new xml document containing a <ResourceDictionary> element and all our style:

**StyleSheet File**

<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

<Style TargetType="TextBox">

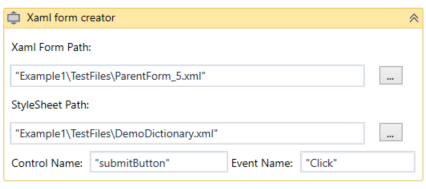
<Setter Property="Foreground" Value="Red" />

</Style>

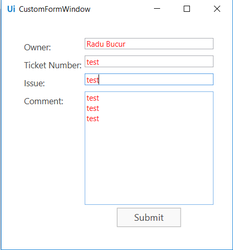
</ResourceDictionary>

This file is saved under the following path: "Example1\TestFiles\DemoDictionary.xml"

Now, if we simply set the StyleSheet property with this path, our activity will look something like this:



If we run the process again, we can see that all the text inside our textboxes is red:



### Example #2 - Creating a more complex Form

Now, lets assume that we're trying to create a workflow for an attended robot in which the current user is trying to modify some data inside the employee database. Naturally, we'll want to provide him with a WPF form in order to perform this task.

Here we'll have quite a few types of controls. Maybe we'll want a custom text box dedicated to entering a date (like the hire Date) or a text box that accepts only numbers and a ComboBox with from which our user will only be able to select a list of predefined values. We might even want to display a picture of the employee in the current form, just for the sake of it!

#### ****Solution:****

**Open the Example2.xaml in order to see the complete solution. Read the indications below for the detailed explanations.**

It is easy to create finely tuned controls once we start leveraging the custom controls provided by the **Extended WPF Toolkit**. With this library, we can add a **DateTimePicker**which we'll use to edit the employee's hire date and an **IntegerUpDown**control in order to input the total number of years of experience the employee has.

For creating check boxes and combo boxes we can use the  **ComboBox** and **CheckBox** standard WPF controls. The same thing goes for the image we want to add, it can be added to the form by using the **Image**control and we can even dynamically set the **Source** property if we don't want a static image.

##### Mark-up for the form

For this example, lets check the mark-up for this form: "Example2\TestFiles\ParentForm\_6.xml"

This is how the mark-up looks like:

**XAML Mark-up**

<Grid xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:xctk="http://schemas.xceed.com/wpf/xaml/toolkit"

Margin="10" Width="800" Height="375" >

<Grid.ColumnDefinitions>

<ColumnDefinition Width="350" />

<ColumnDefinition Width="450" />

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

</Grid.RowDefinitions>

<StackPanel Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="2" Orientation="Horizontal">

<Image

Name="employee"

Grid.Row="0" Grid.Column="0"

Margin="2" Stretch="Fill" Width="96" HorizontalAlignment="Center" >

<Image.ToolTip>

<TextBlock>

<Run>Employee:</Run>

<TextBlock Margin="4,0,0,0" Text="Camus" />

</TextBlock>

</Image.ToolTip>

</Image>

<Label HorizontalAlignment="Center" Style="{DynamicResource HeaderStyle}" VerticalAlignment="Center">

Employee Details Form

</Label>

</StackPanel>

<Grid Grid.Row="1" Grid.Column="0" Margin="5">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto" />

<ColumnDefinition Width="\*" />

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="\*" />

</Grid.RowDefinitions>

<Label>First Name:</Label>

<TextBox Grid.Column="1" Margin="0,5,0,10" Name="firstName"/>

<Label Grid.Row="1">Last Name:</Label>

<TextBox Grid.Row="1" Grid.Column="1" Margin="0,5,0,10" Name="lastName" />

<Label Grid.Row="2">Title:</Label>

<TextBox Grid.Row="2" Grid.Column="1" Margin="0,5,0,10" Name="title" />

<Label Grid.Row="3">Manager's Note:</Label>

<TextBox Grid.Row="3" Grid.Column="1" AcceptsReturn="True" Name="managerNote" Height="100" />

</Grid>

<Grid Grid.Row="1" Grid.Column="1" >

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto" />

<ColumnDefinition Width="\*" />

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

</Grid.RowDefinitions>

<Label>Hire Date:</Label>

<xctk:DateTimePicker Grid.Column="1" Width="120" HorizontalAlignment="Left" x:Name="hireDate" Format="Custom" FormatString="dd-MM-yyyy" Margin="0,5,0,10" />

<Label Grid.Row="1">Years of Experience:</Label>

<xctk:IntegerUpDown HorizontalAlignment="Left" Width="40" Grid.Row="1" Grid.Column="1" FormatString="N0" Value="3" Increment="1" Minimum="0" Maximum="50" Margin="0,5,0,10"/>

<Label Grid.Row="2">Bachelor's Degree:</Label>

<CheckBox Grid.Row="2" Grid.Column="1" Name="bachelorsDegree" Margin="0,5,0,10" ></CheckBox>

<Label Grid.Row="3">Master's Degree:</Label>

<CheckBox Grid.Row="3" Grid.Column="1" Name="mastersDegree" Margin="0,5,0,10" ></CheckBox>

<Label Grid.Row="4">Background:</Label>

<ComboBox Grid.Row="4" Grid.Column="1" Width="Auto" Name="Background" >

<ComboBoxItem Content="Mobile Development" />

<ComboBoxItem Content="Web Development" />

<ComboBoxItem Content="Data Scientist" />

<ComboBoxItem Content="Cloud Computing" />

</ComboBox>

</Grid>

<StackPanel Grid.Row="2" Grid.Column="0" Grid.ColumnSpan="2">

<Button Style="{DynamicResource SubmitButton}" Name="submitButton" xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">Submit Changes</Button>

</StackPanel>

</Grid>

The style dictionary is inside this file: "Example2\TestFiles\DemoDictionary.xml"

**StyleSheet File**

<ResourceDictionary

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >

<Style TargetType="TextBox">

<Setter Property="Foreground" Value="Black" />

<Setter Property="FontSize" Value="14" />

</Style>

<Style TargetType="Label">

<Setter Property="HorizontalAlignment" Value="Right"/>

<Setter Property="Margin" Value="0,5,0,10"/>

<Setter Property="FontSize" Value="14" />

</Style>

<Style x:Key="HeaderStyle" TargetType="Label">

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="Foreground" Value="Black" />

<Setter Property="FontSize" Value="20" />

<Setter Property="FontWeight" Value="Bold"/>

</Style>

<Style x:Key="SubmitButton" TargetType="Button">

<Setter Property="FontSize" Value="20" />

<Setter Property="Margin" Value="5"/>

<Setter Property="HorizontalAlignment" Value="Center"/>

<Setter Property="Width" Value="180"/>

<Setter Property="Height" Value="45"/>

<Setter Property="Background" Value="White"/>

</Style>

</ResourceDictionary>

##### Activity

Finally, lets see how the **Xaml form creator** activity is configured.

The path to our current form and to the StyleSheet are added as well as the name of the Control for the submit event, in this case, **"submitButton"**and the event name: **"Click".**

The input Dictionary also needs to be configured. In this example, we'll use some hard-coded data for it, but in a production scenario, those properties will most likely be initialized using variables.

In order to convert the Source of the image, we pass a BitmapImage object to the Source property

**Input Dictionary**

New Dictionary( Of String, Dictionary( Of String, Object)) From { { "firstName", New Dictionary( Of String, Object) From { { "Text", "Albert" } } },

{ "employee", New Dictionary( Of String, Object) From { { "Source", New BitmapImage(New Uri(Environment.CurrentDirectory + "\Example2\TestFiles\albertcamus.jpg", UriKind.Absolute)) } } },

{ "hireDate", New Dictionary( Of String, Object) From { { "Value", DateTime.Now } } },

{ "lastName", New Dictionary( Of String, Object) From { { "Text", "Camus" } } },

{ "title", New Dictionary( Of String, Object) From { { "Text", "RPA Dev" } } },

{ "managerNote", New Dictionary( Of String, Object) From { { "Text", "" } } },

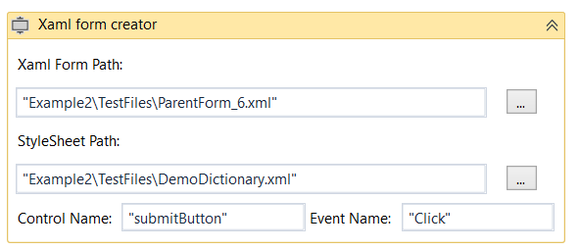
{ "bachelorsDegree", New Dictionary( Of String, Object) From { { "IsChecked", True } } },

{ "mastersDegree", New Dictionary( Of String, Object) From { { "IsChecked", True } } },

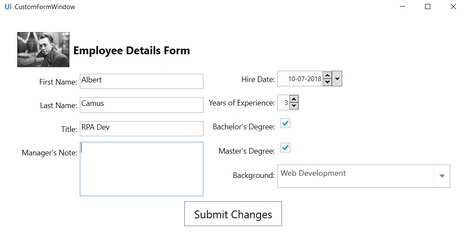
{ "Background", New Dictionary( Of String, Object) From { { "SelectedValuePath", "Content" },

{ "SelectedValue", "Web Development" } } }}

So, the activity looks like this:



##### End Result



Now, if we will fill in the form and we submit the changes, the output window will look like this:

