Dialog Validation

A WPF Custom control

Summary

This document introduces a new custom control and describes its features. Code sample is provided to demonstrate typical usage.

Introduction

The Dialog Validation custom control is a customized control for the WPF (Windows Presentation Framework) environment[1]. It aims at providing the same buttons you can find at the bottom of a dialog box but in a simple and localized way.

The control is provided as a signed 64-bits assembly at www.numbatsoft.com. An unsigned (and untested) 32-bits version is also available.

How to use the control

You need to add the following line in your xaml file:

```
xmlns:ctrl="CustomControlsLibrary"
```

Alternatively, if the name “CustomControlLibrary” is already used, you can try the following syntax:

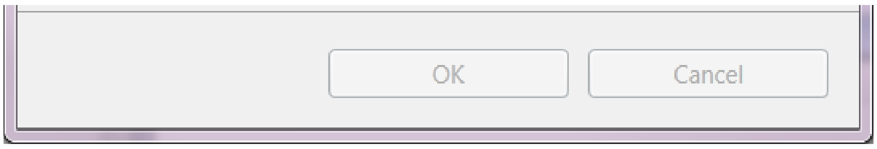
```
xmlns:ctrl="clr-namespace:CustomControls;assembly=DialogValidation"
```

You can then add the control in your UI:

```
<ctrl:DialogValidation/>
```

This will display the OK and Cancel buttons (English), right aligned, wherever you put the control:

1- For more information about WPF, look at the Microsoft .NET Framework documentation. The control has been designed for .NET version 4.5.



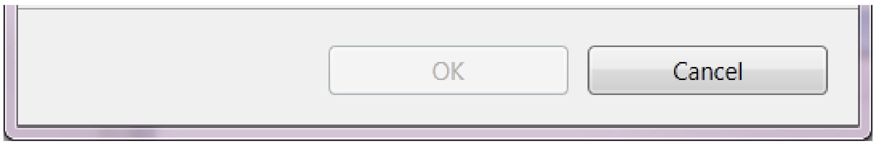
The buttons are disabled because they must be connected to your dialog box through command binding:

```
<Window.CommandBindings>
    <CommandBinding Command="{x:Static ctrl:DialogValidation.DefaultCommandOk}"
Executed="OnOk" CanExecute="OnCanExecuteOk"/>
    <CommandBinding Command="{x:Static ctrl:DialogValidation.DefaultCommandCancel}"
Executed="OnCancel"/>
</Window.CommandBindings>
```

In the code above, OnOk, OnCanExecuteOk and OnCancel are methods you implemented in the dialog box. You are of course free to choose any name you want.

"{x:Static ctrl:DialogValidation.DefaultCommandOk}" and "{x:Static ctrl:DialogValidation.DefaultCommandCancel}" are objects declared in the custom control, that you can use to activate the binding. You can also use your own commands (more on that later).

Assuming OnCanExecuteOk disallows executing OnOk yet, the dialog box then looks like this:



Clicking the Cancel button will execute the code of OnCancel.

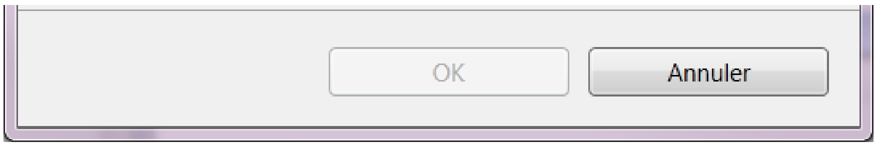
Customization

The control is not limited to displaying OK and Cancel. This section demonstrates the various customization options available.

Localization

If you set the *IsLocalized* property to true, buttons will display a localized text instead of the default English text. Localized strings are taken from OS language, more specifically from the good old user32.dll file.

Here is a screenshot taken on a French Windows 7 OS:



Choice of buttons

You select which buttons you want to see by changing the value of the *ActiveCommands* property. The full syntax is as follow:

```
<ctrl:DialogValidation>
    <ctrl:DialogValidation.ActiveCommands>
        <ctrl:ActiveCommandYes/>
        <ctrl:ActiveCommandNo/>
    </ctrl:DialogValidation.ActiveCommands>
</ctrl:DialogValidation>
```

The *ActiveCommands* property is the default content property, so you can use a shortcut version of the same syntax:

```
<ctrl:DialogValidation>
    <ctrl:ActiveCommandYes/>
    <ctrl:ActiveCommandNo/>
</ctrl:DialogValidation>
```

And finally, the custom control provides a converter from a comma-separated list of strings, so you can use this even shorter syntax:

```
<ctrl:DialogValidation ActiveCommands="Yes, No"/>
```

(The whitespace character is ignored).

The custom control supports the following buttons and their localized versions:

- OK
- Cancel
- Abort
- Retry
- Ignore
- Yes
- No
- Close
- Help
- Try Again
- Continue

Note how some letters are underlined. The localized buttons will have the proper letter underlined as well.

You specify in the list how the buttons are ordered. The same button can appear multiple times if that makes sense for your application.

Orientation

The custom control is implemented as a Stackpanel, and you can specify its orientation with the *Orientation* property.

This allows you to show buttons horizontally (the default) or vertically.

Alignment

By default, buttons are aligned on the right, but you can change that with the standard *HorizontalContentAlignment* property. Vertically displayed buttons are aligned to Bottom and you can also change the default with the *VerticalContentAlignment* property.

Customizing commands

In case you want to use the same command for one of the custom control buttons and some other command source in your application, you can either use (taking the example of the OK button) "{x:Static ctrl:DialogValidation.DefaultCommandOk}" for both, or use your own command. In that case, simply specify which object to use in the *CommandOk* (or any of the *CommandXXX*) property. For instance:

```
<ctrl:DialogValidation CommandOk="{StaticResource MyCommand}"/>
```

Customizing content

The content of each button can be customized with the *ContentOk* (or any of the *ContentXXX*) property.

Important: customized content is active only if *IsLocalized* is set to true, and then every button is displayed with its localized or customized content. The point of this is to use a different string in peculiar circumstances where you need text different than the Windows text.

Typically, if you start customizing buttons with something other than text, you probably want to create your own control anyway, so no support for more customization is offered here.

Default and Cancel buttons

By default, the OK button has IsDefault=True, and the Cancel button IsCancel=True.

To avoid this behavior, use either ActiveCommandOkBase or ActiveCommandCancelBase. For example, the example below specifically keeps the IsDefault behavior of the OK button, and removes the IsCancel behavior. As a result, pressing the Escape key does not close the dialog box.

```
<ctrl:DialogValidation>
    <ctrl:ActiveCommandOk/>
    <ctrl:ActiveCommandCancelBase/>
</ctrl:DialogValidation>
```