ApplFLab

Application Framework Laboratory

Custom-made User Interface Components in VisualWorks

Tutorial Exercises

Koen De Hondt

Style conventions used in this text:

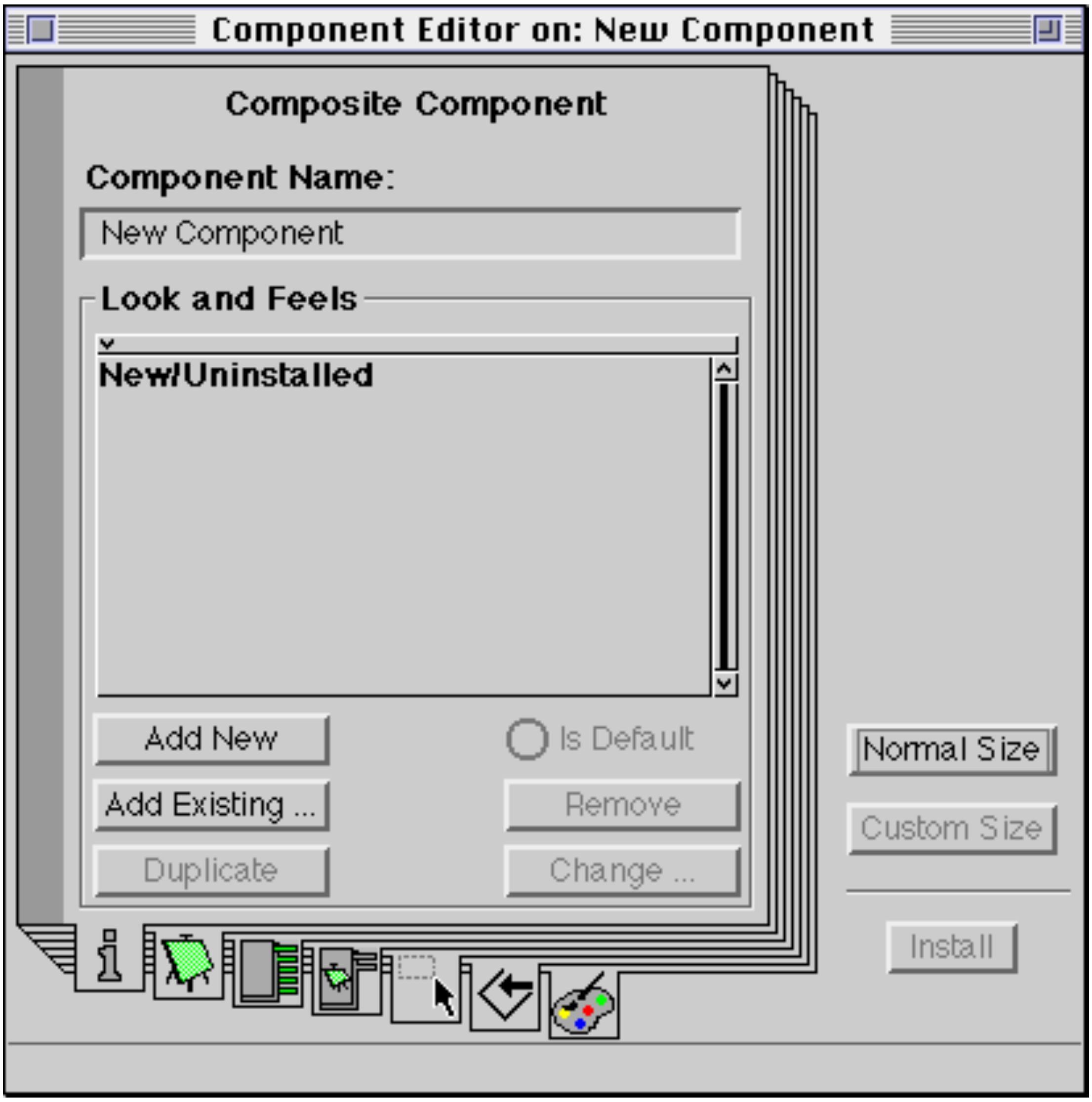
Text in *italic*:	text that you read on screen.
Text in `fixed size font`:	text you type into fields.
<u>Underlined</u> text:	to indicate a choice.
Capitalized words	concepts/tools in ApplFLab and class names.

EXERCISE 1 TURNING AN EXISTING APPLICATION MODEL INTO A COMPONENT

The goal of this exercise is to create a component that uses an already existing application model, TutorialIconEditor, an icon editor such as the one embedded in the Component Editor. This exercise shows how to use the Component Editor. Advanced features wrt. properties are deferred until later.

Step 1 Via the launcher, open a Component Editor on a new composite component.

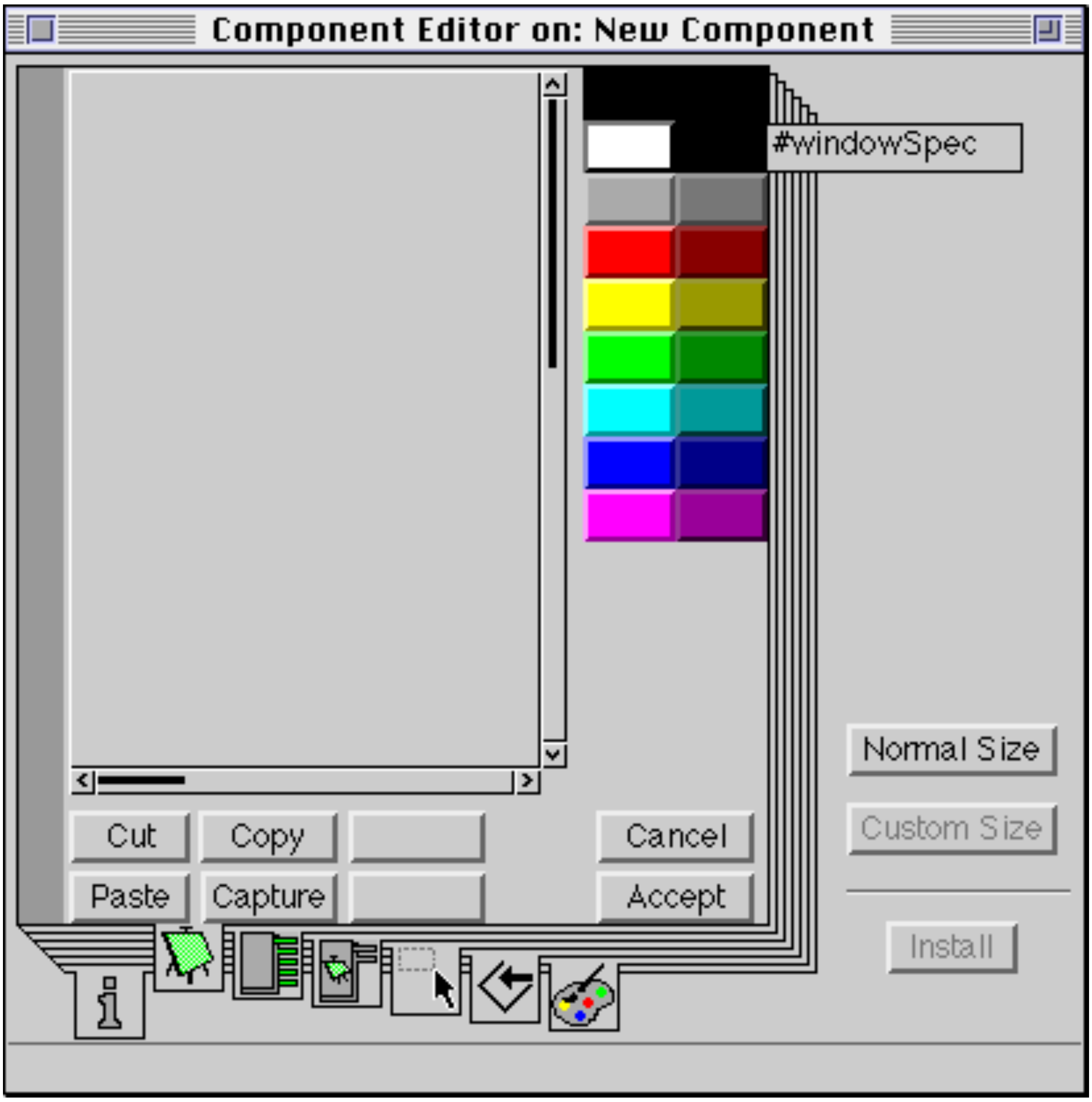
Step 2 The first page of the Component Editor is displayed: the general information page.



Enter the component name: `Icon Editor`.
Select the only item in the *Look and Feels* list and press the *Change...* button. In the dialog that appears, find and select class TutorialIconEditor

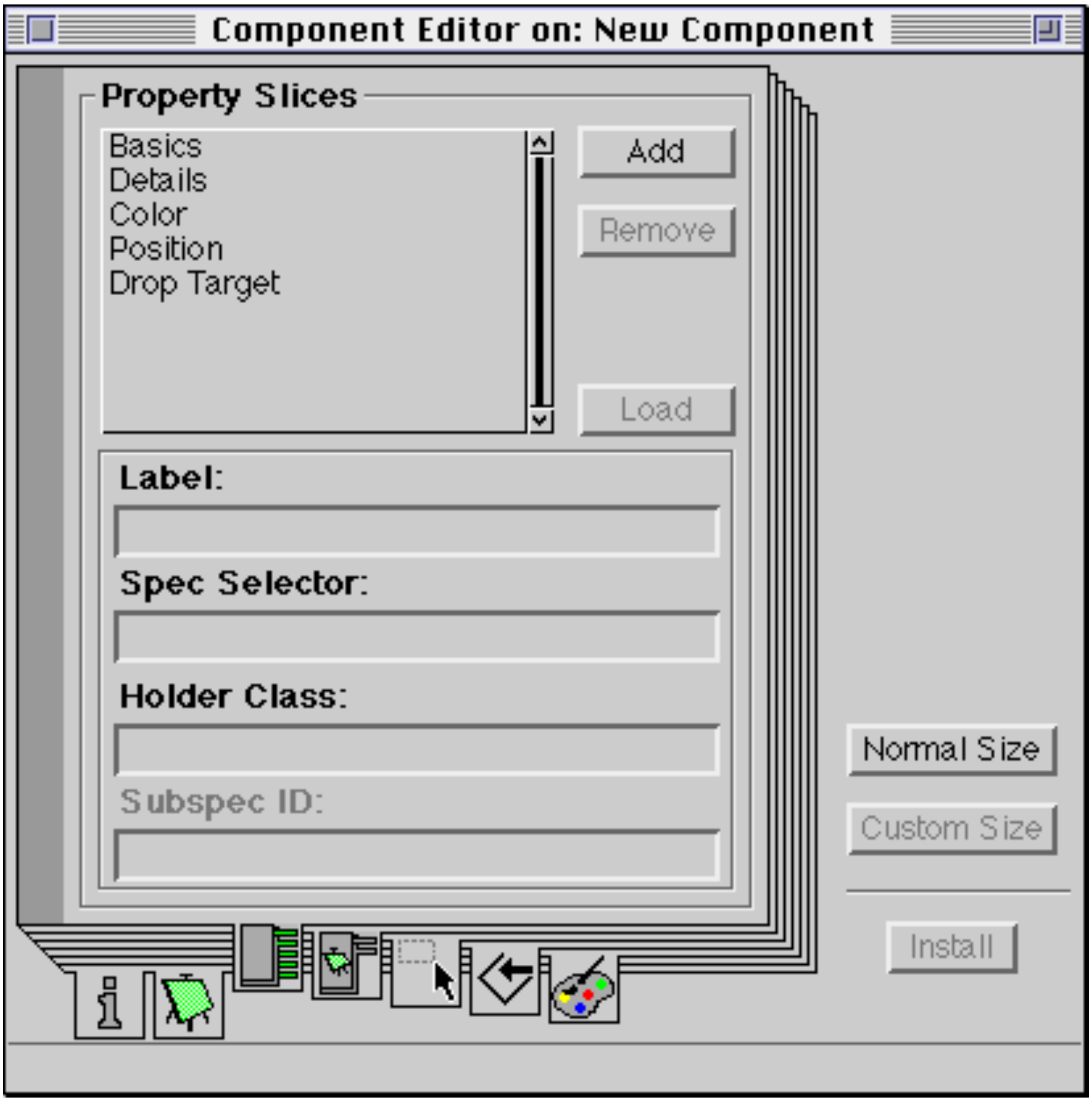
and resource windowSpec. Press the *OK* button.
The only item in the *Look and Feels* list changes corresponding to the selections in the dialog window.

Step 3 Turn to the next page of the Component Editor's notebook: the user interface painter page.



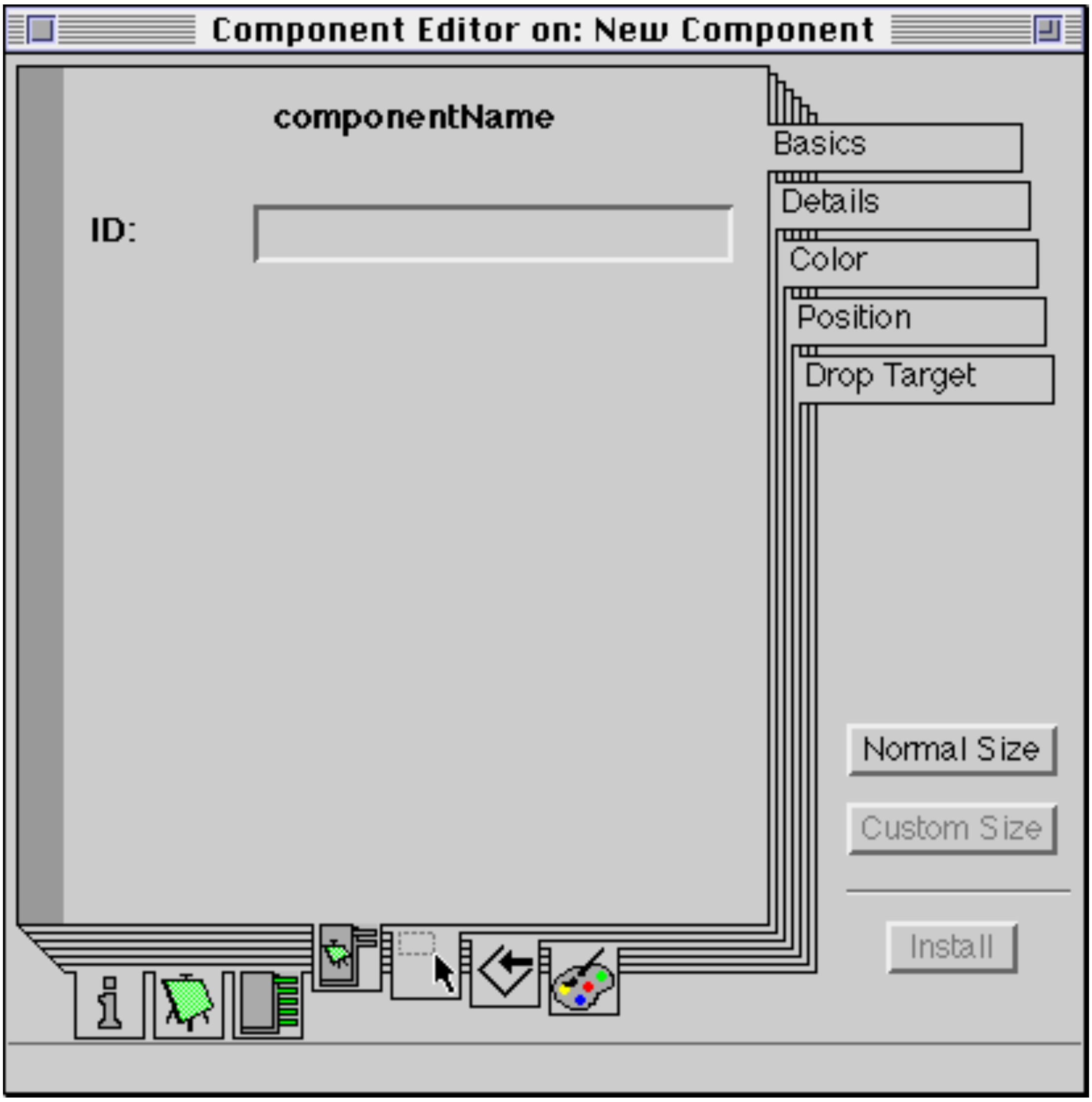
The user interface specification selected in the previous step is now displayed.
Note that this user interface can be edited. Leave it as it is for now.

Step 4 Turn to the next page of the Component Editor's notebook: the slice configuration page.



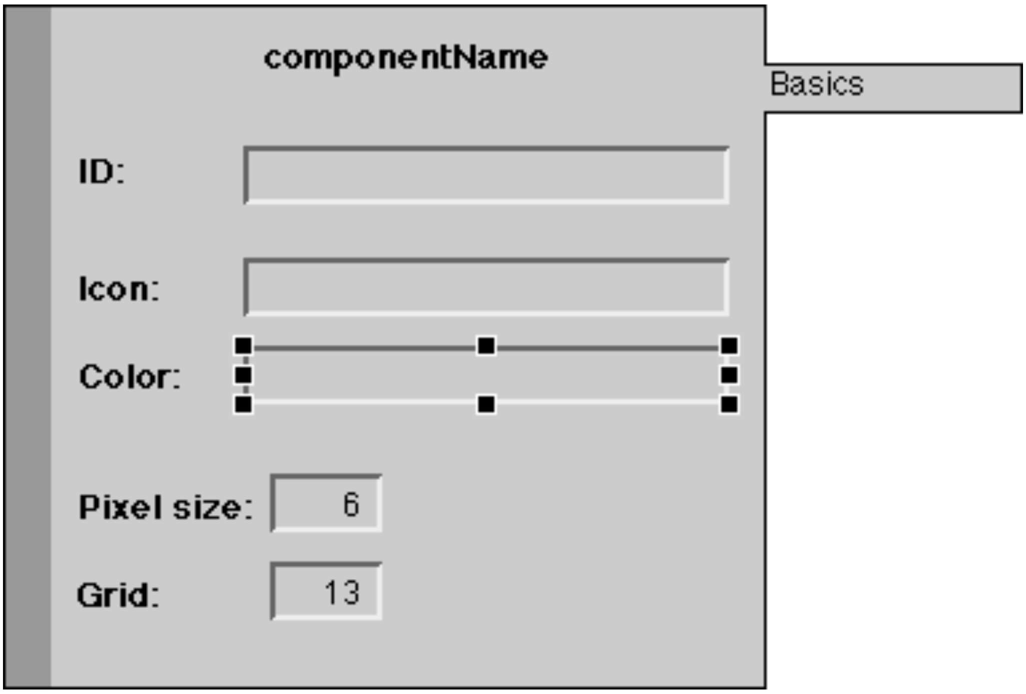
Leave the property slices as they are.

Step 5 Turn to the next page of the Component Editor's notebook: the properties slice painter page.

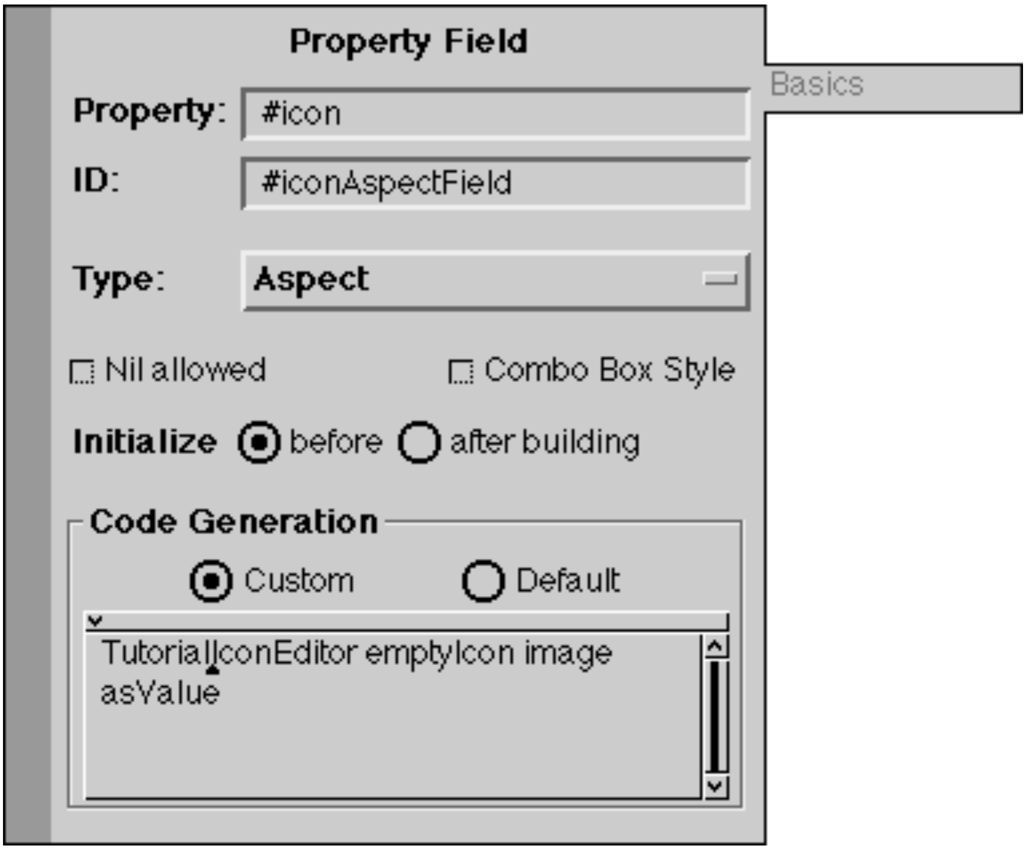


Open the *Properties* palette via the Launcher and position it next to the Component Editor.
By default, the Basics page contains a label for the component's name and an input field for the component's ID.

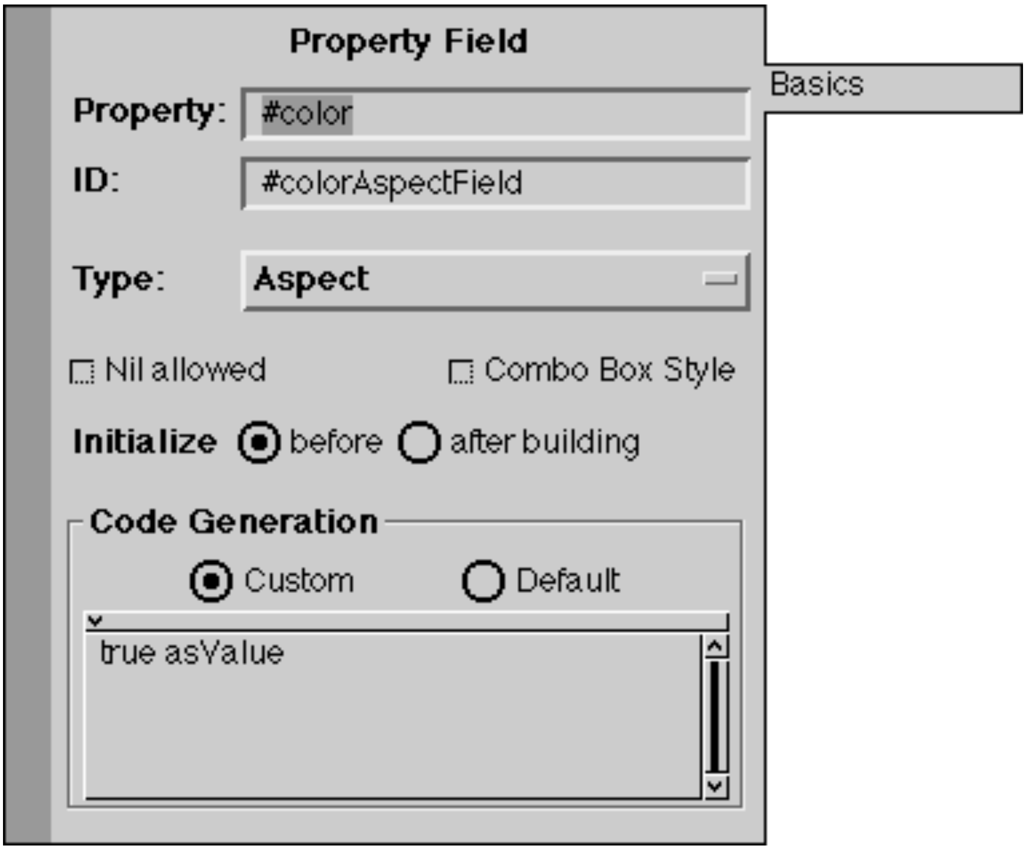
After completing step 5, the Basics slice of our component should look like this:



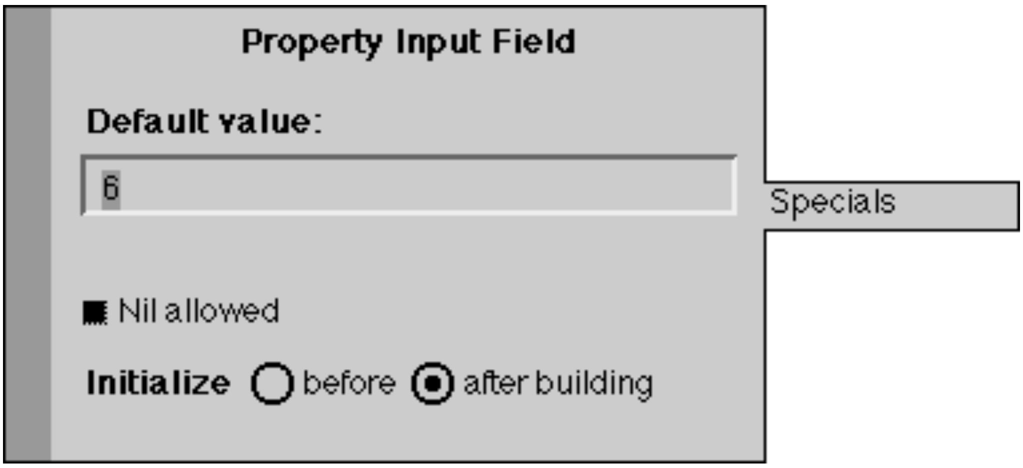
Add a label `Icon` and put a Property Field (from the *Properties* palette) next to it. Configure the latter as depicted below.



Add a label `Color` and put a Property Field next to it. Configure the latter as depicted below.

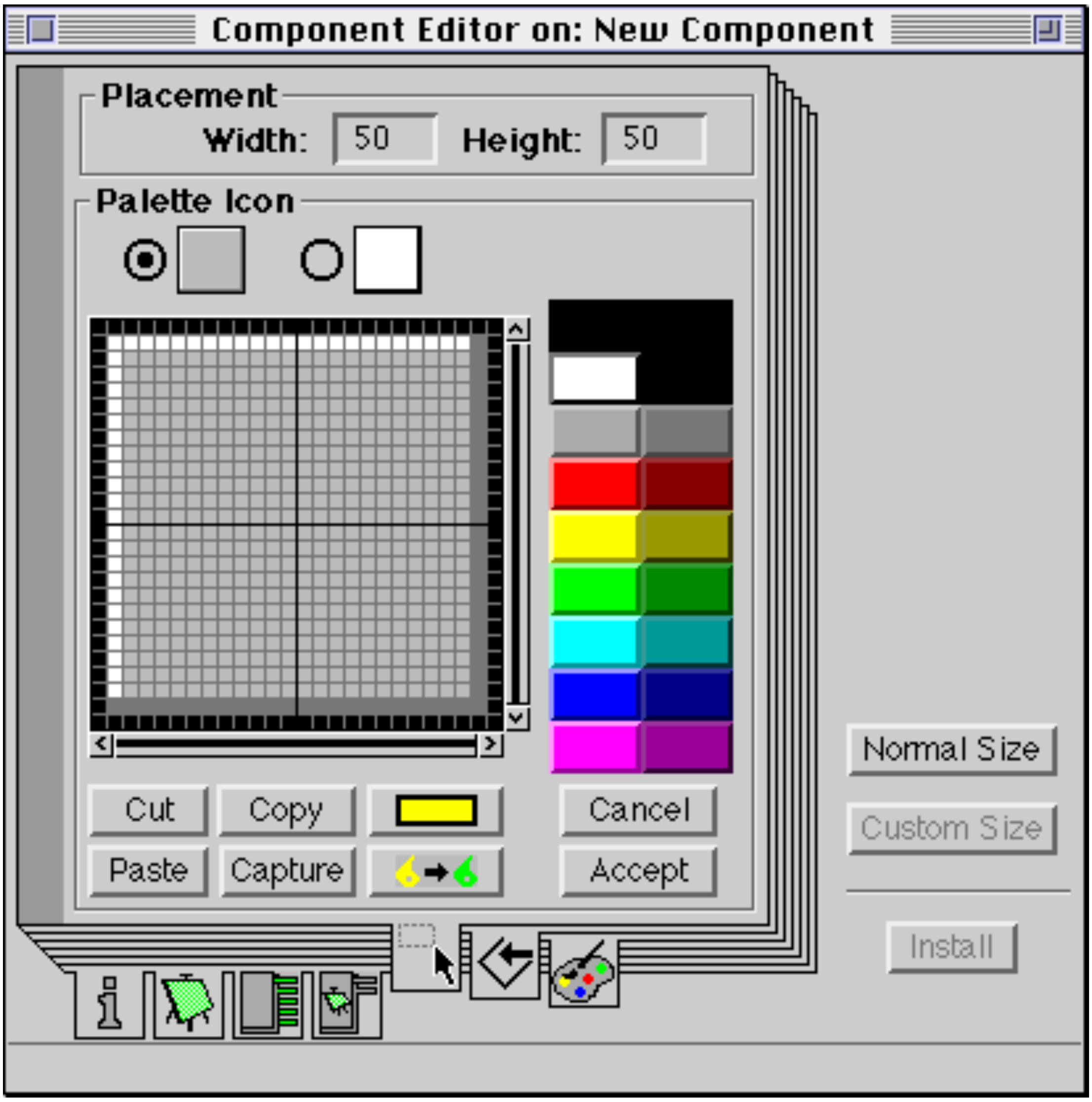


Add a label `Pixel size` and put a Property <u>Input</u> Field (from the *Properties* palette) next to it. On the *basics* slice in the properties tool, enter the aspect `scale` and select type `Number`. On the *details* slice, enter right alignment. On the *specials* slice, enter `6` as the *default value*, check the *Nil allowed* flag, and select *after building*, as shown in the picture below.



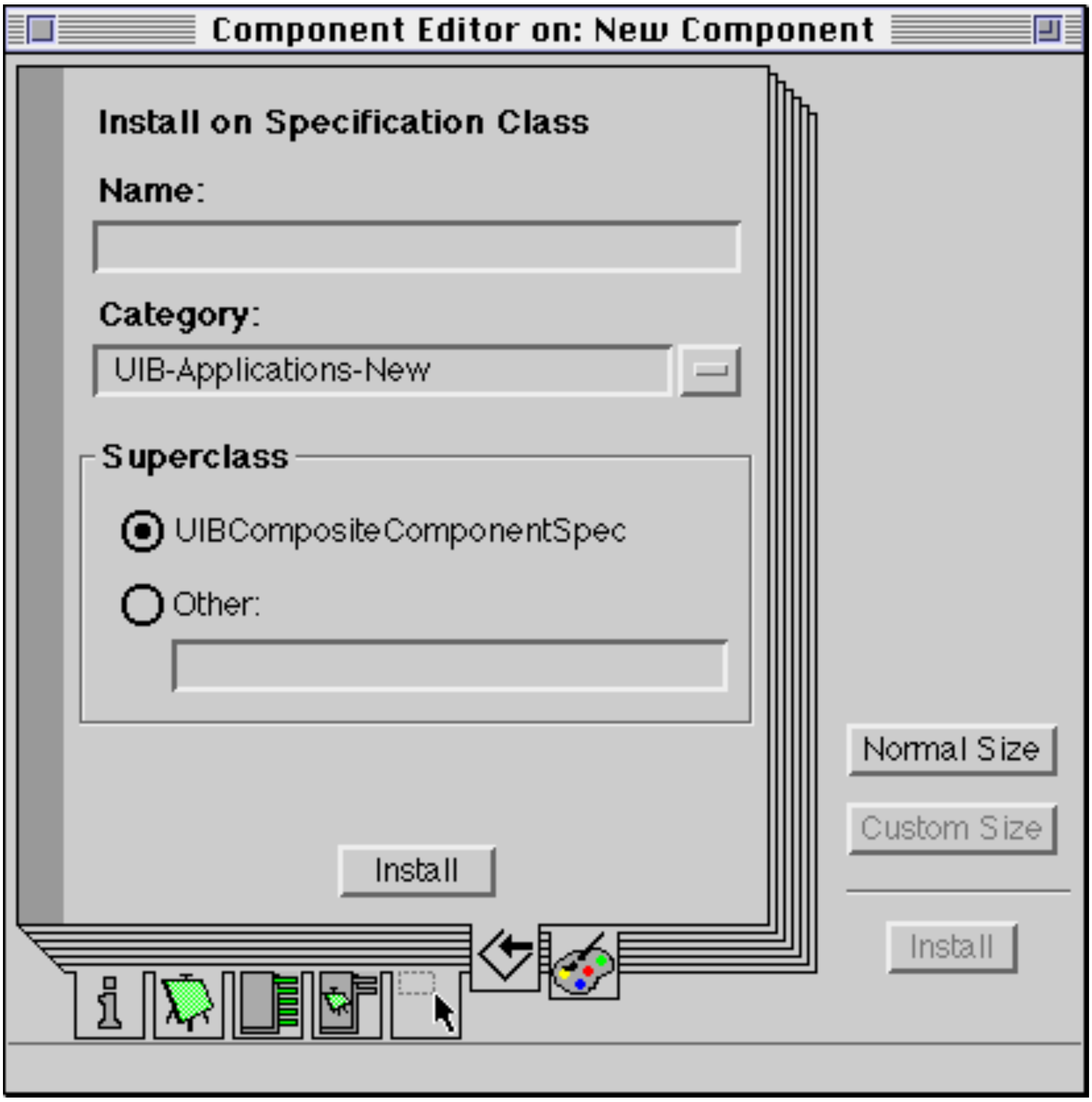
Add a label `Grid` and put a Property <u>Input</u> Field (from the *Properties* palette) next to it. On the *basics* slice in the properties tool, enter the aspect `grid` and select type `Number`. On the *details* slice, enter right alignment. On the *specials* slice, enter `13` as the *default value*, check the *Nil allowed* flag, and select *after building*.

Step 6 Turn to the next page of the Component Editor's notebook: the painting page.



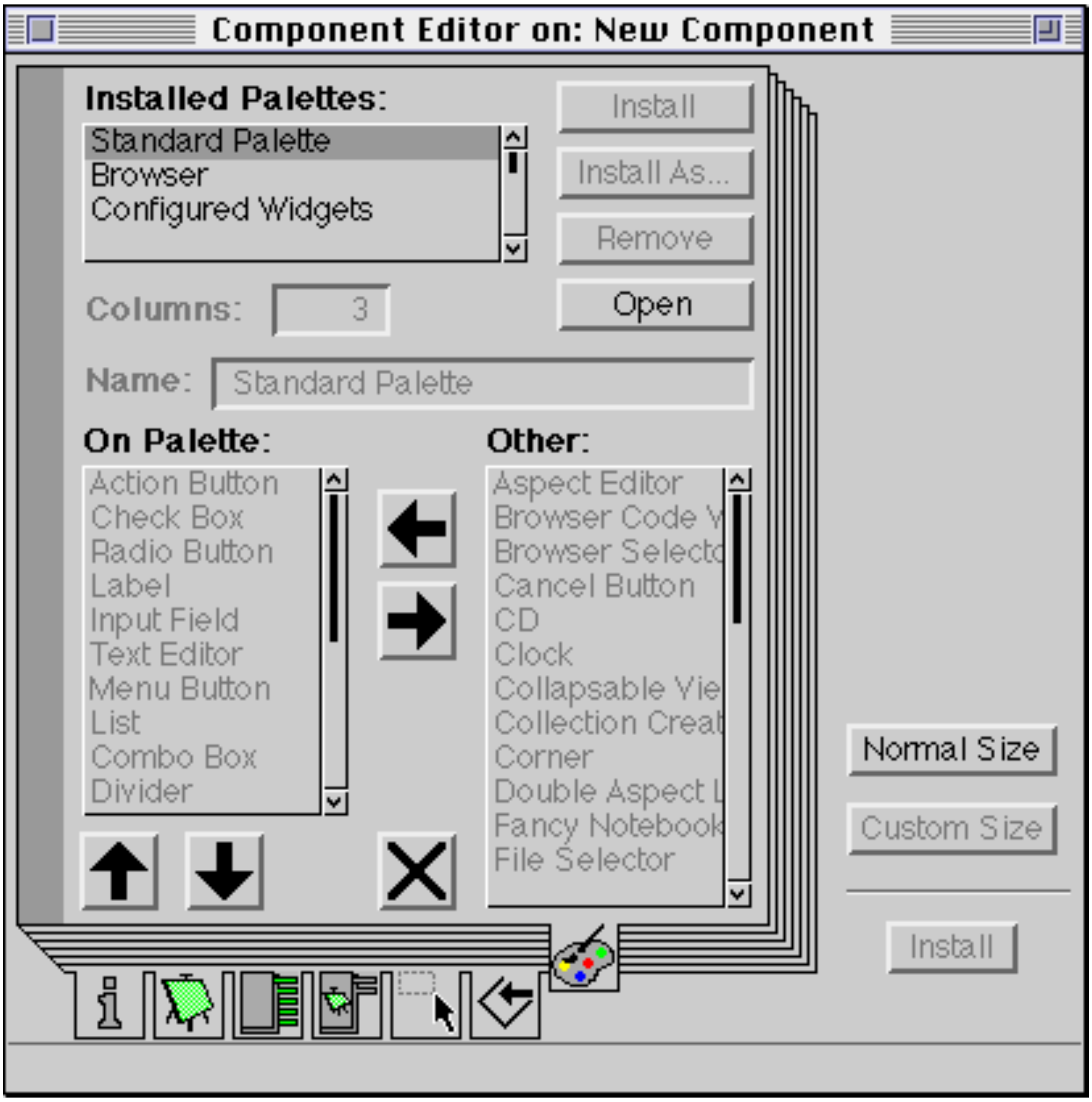
Enter 230 both for the placement width and height.
Draw a nice color icon for this component.

Step 7 Turn to the next page of the Component Editor's notebook: the installation page.



Enter `TutorialIconEditorSpec` in the *Name* field, `Tutorial` in the *Category* field, leave the other settings as they are, and press the *Install* button.

Step 8 Turn to the next page of the Component Editor's notebook: the palette editor page.



Deselect the selection in the palette list. Now you can enter the name and the number of columns of a new palette.
Enter `5` in the *Columns* field.
Enter `Tutorial` in the *Name* field.
Find and select the *Icon Editor* in the component list on the right and press the ← button.
Press the Install As... button. In the dialog window that appears, enter `TutorialPalette` in the *Name* field, `Tutorial` in the *Category* field, and press the *OK* button.

Step 9 Open a Browser on class TutorialIconEditor.
Select protocol *initialize-release*.
Enter the following methods. These messages are sent by the specification object to the application model when it is built by the Builder. They correspond to all aspect property names you entered for the property fields painted on the component's properties slices.

```
icon: anImageHolder
   icon retractInterestsFor: self.
   icon := anImageHolder.
```

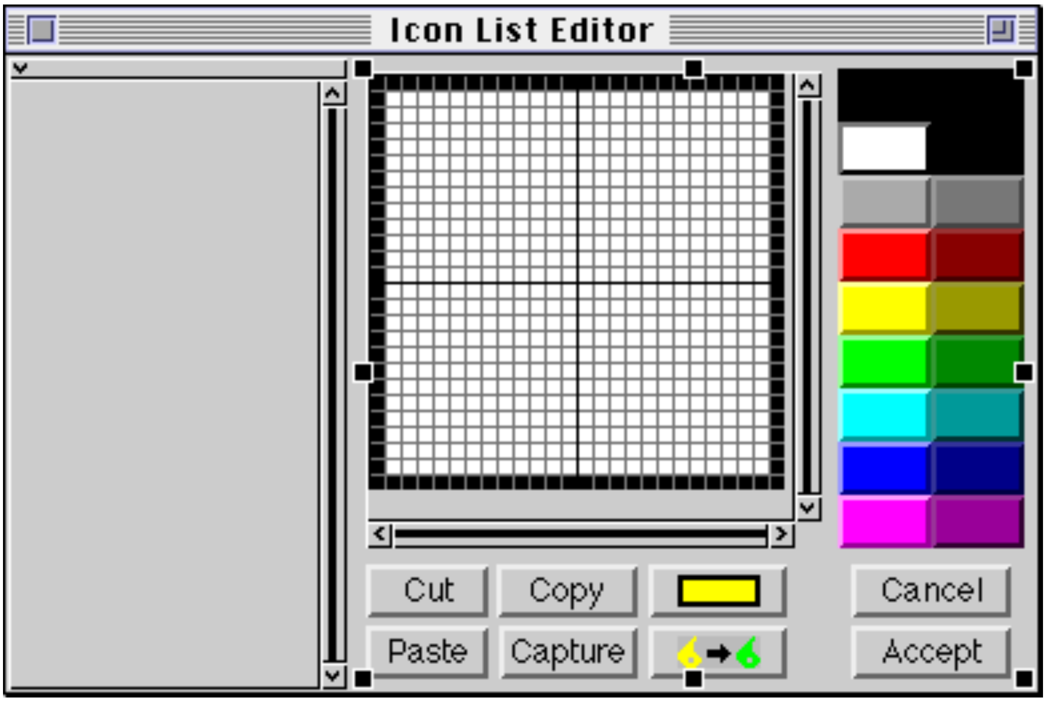
```
   self setupBitView.
   icon onChangeSend: #iconHolderChanged to: self

color:  aBooleanHolder
   colorHolder := aBooleanHolder.
   colorHolder onChangeSend: #colorHolderChanged to: self

scale:  anInteger
   self magnifiedBitView scale:
      (anInteger < 1
         ifTrue: [1@1]
         ifFalse: [anInteger @ anInteger])

grid:  anInteger
   self magnifiedBitView gridding:
      (anInteger < 1
         ifTrue: [1@1]
         ifFalse: [anInteger @ anInteger])
```

EXERCISE 2 (RE)USING THE NEW COMPONENT

The goal of this exercise is to use the self-made component of exercise 1 in an application.

Step 1 Open a new canvas.
Open the *Standard* palette.
Open the *Tutorial* palette.
Open the Properties Tool.

Step 2 Paint a list component and an icon editor component on the canvas and position them appropriately with the Position Tool, as shown in the figure below. The list takes 1/3 of the width of the canvas.



Enter `iconList` in the *Aspect* field of the list's *Basics* slice in the Properties Tool. Do the same for the *ID* field.
Enter `iconEditor` in the *ID* field of the icon editor's *Basics* slice. Enter `selectedIcon` in the *Icon* field and `inColor` in the *Color* field.
The *Pixel size* and *Grid* fields hold the default values that you specified in the Component Editor during the previous exercise. Experiment with other values. Your changes are reflected in the canvas when you apply them.

Step 3 Install the canvas on class `TutorialIconListEditor` with spec `windowSpec`. This class already exists. The *initialize* method was provided to fill the list with references to icons. Check it out with a Browser.

Step 4 Define all aspects for the canvas. Note that there are 3 aspects: *iconList*, *selectedIcon*, and *inColor*.
Open a Browser on class TutorialIconListEditor to find out what was generated by the Definer. Compare with the Code Generation settings for the property fields on the Basics slice of the icon editor component.

Step 5 Open and test the application.

EXERCISE 3 EXTENDING THE COMPONENT

The goal of this exercise is to extend the component of exercise 1 with an extra property. This exercise shows how different user interfaces for a component can be selected with the Properties Tool.

Step 1	Open class *TutorialIconEditorSpec* via the Resource Finder. Open the *Standard* palette.
Step 2	On the first page of the Component Editor's notebook, press the *Add New* button to add a new entry to the *Look and Feels* list.
Step 3	Turn to the second page of the Component Editor's notebook. Select and copy all the components on the canvas. Select the notebook's right tab that corresponds with the added look and feel from step 2 (called *New/Uninstalled*). Paste the previously copied components. Rearrange the buttons such that the color buttons are positioned at the bottom side and the other are positioned at the right side of the canvas.
Step 4	Install this canvas using the *Install...* button of the Canvas Tool or the *install...* command in the canvas's operate menu. Install on class `TutorialIconEditor` and selector `colorsAtBottomSpec`. The label of the notebook's right tab changes accordingly. (When you turn back to the Component Editor's first page, you will see that the installation is registered.)
Step 5	Turn to the third page of the Component Editor's notebook. Select *Details* in the property slices list. Press the *Add* button. A new entry is added to the list. Enter `Looks` in the *Label* field. Enter `looksEditSpec` in the *Spec Selector* field.
Step 6	Turn to the fourth page of the Component Editor's notebook. Select the notebook's right tab labeled *Looks*. Drag a radio button from the palette and put it on the canvas. Enter `Color buttons on the right side` in the *Label* (String) field. Enter `minorKey` in the *Aspect* field. Enter `windowSpec` in the *Select* field. Drag another radio button from the palette and put it on the canvas. Enter `Color buttons at the bottom` in the *Label* (String) field. Enter `minorKey` in the *Aspect* field. Enter `colorsAtBottomSpec` in the *Select* field.

Press the *Install* button in the bottom right corner of the Component Editor. Pressing this button is a shortcut for the installation page of the Component Editor.

EXERCISE 4 USING THE EXTENDED PROPERTIES OF THE COMPONENT

The goal of this exercise is to use the extra property that was added to the icon editor's properties slices. Exercise 3 and 4 show that a component can be build incrementally. Note however that, while adding properties to a component has no impact on components that are present in existing canvasses, removing (and often changing) properties makes components in existing interfaces invalid. Opening such canvasses will not be possible anymore (this would result in an exception).

Step 1 Open canvas *windowSpec* of class *TutorialIconListEditor* via the Resource Finder.

Step 2 Select the icon editor component.
Select the *Looks* slice in the Properties Tool. Click the radio button labeled *Color buttons at the bottom* and press the *Apply* button. The change is reflected immediately in the canvas.

Step 3 (Re)install the canvas.

Step 4 Open and test the application.

EXERCISE 5 BUILD A COMPONENT FROM SCRATCH

Many applications employ a component which is an aggregation of a list of objects, some buttons to manipulate the list, and an editor to edit those objects. Typical is a list combined with an input field to edit some aspect of the object selected in the list, but in general the editor could have a very complex user interface.

The goal of this exercise is to build a simple "list & input field" component.

For this exercise our domain model is persons, so we use the class Person, which is already available in the image.

Step 1 Explore class Person with a Browser.

Step 2 Open a Component Editor and build a component called "Person List Editor".
The component has 4 subcomponents: a list of persons, an input field to edit the person's first name, a button to add a new Person to the List, and a button to remove a Person from the List.
Install the component's application model on class TutorialPersonListEditor
The component has one model: a List of Person objects. This model is provided by the component user in an aspect property field on the Basics slice in the Properties Tool.
The component has a visual block: a block that returns a display string for a Person in the List. This block is proided by the component user in an action property field on the Basics slice in the Properties Tool.
Install the component on class TutorialPersonListEditorSpec.
Do not forget to put the component on the *Tutorial* palette.

Step 3 Build a test application that contains a person list editor component.
Open and test the application (and your component).

Step 4 Extend the component with the capability to change its user interface such that the component user can choose between a person list editor that only edits the first name and one that edits the first as well as the last name of a Person.

Step 5 Open your test application canvas that contains the person list editor component.
Change the property you added in the previous step.
(Re)install the canvas.
Open and test the application (and your component).