Subcanvasses

The Subcanvas widget provides one means of reusing visual entities, such as commonly used layouts, etc. Visual Reuse is just as important as code reuse. In this module we demonstrate how to reuse visual components through use of class DateModel.

1. Visual Reuse

There are three techniques for achieving visual reuse using the Subcanvas widget

1. Inheritance
2. Reuse of a Canvas specification
3. Embedding one application inside another

We'll look at each in turn. However, before going any further, let's look at the interface we wish to reuse. Class DateModel provides a simple user interface for the user to enter a Date. In this module we will explore how to reuse that interface in the BondEntry application.

Ex 1. Browse class DateModel. Experiment with an instance of it by sending the class the message open.

2. Inheritance

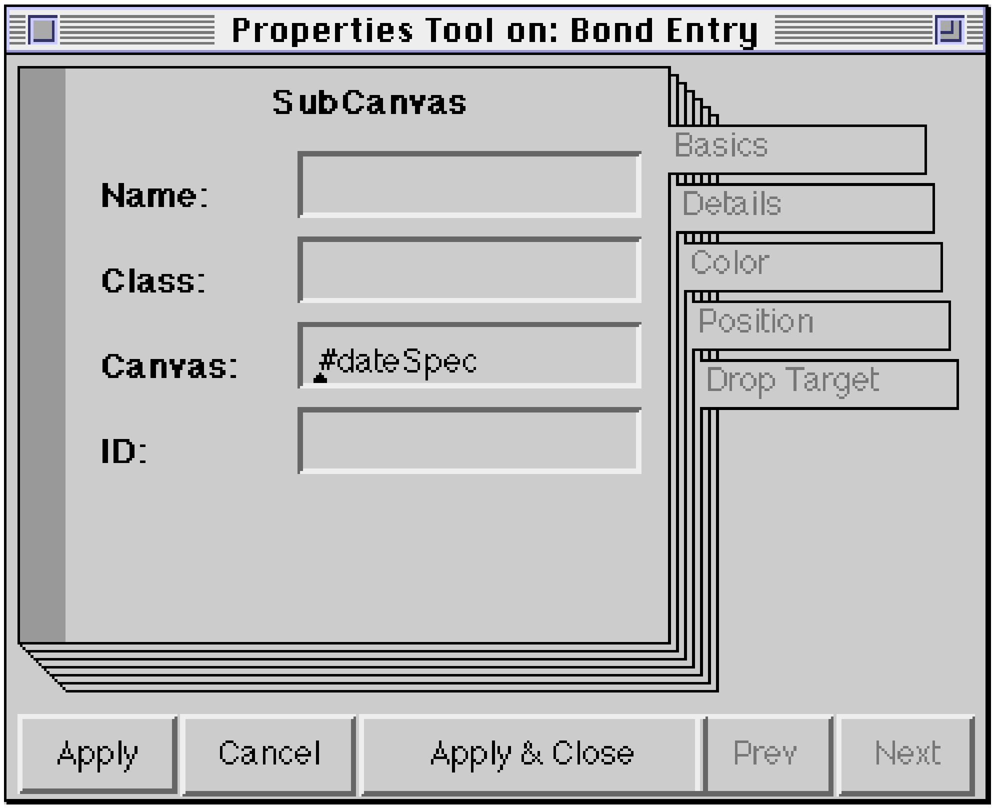
The first technique for achieving visual reuse is inheritance. In the case of the BondEntry application, we could change its superclass to be class DateModel, and thus gain access to DateModel's behavior and instance variables. A Subcanvas widget may be used to access the Canvas specification of DateModel. In Fig.1the specification that is being reused is named #dateSpec.

Ex 2. Change the definition of class BondEntry so that its superclass is DateModel.[1] Add a Subcanvas widget to reuse the dateSpec Canvas in DateModel.

At first sight, this approach appears to be very successful. However, it does have two drawbacks:

1. As the example shows, there is little other justification for creating BondEntry as a subclass of DateModel. It's a very bad example of inheritance.
2. An inherited Canvas specification cannot be used more than once in the same Canvas. For example, BondEntry could not have two Subcanvas widgets that both used the dateSpec Canvas of DateModel, because both would reference the same instance variables.

1. It's a good idea to file–out class BondEntry before changing its definition — we'll be needing it later.



Name:
Class:
Canvas: #dateSpec
ID:

Figure 1: Visual Reuse via inheritance

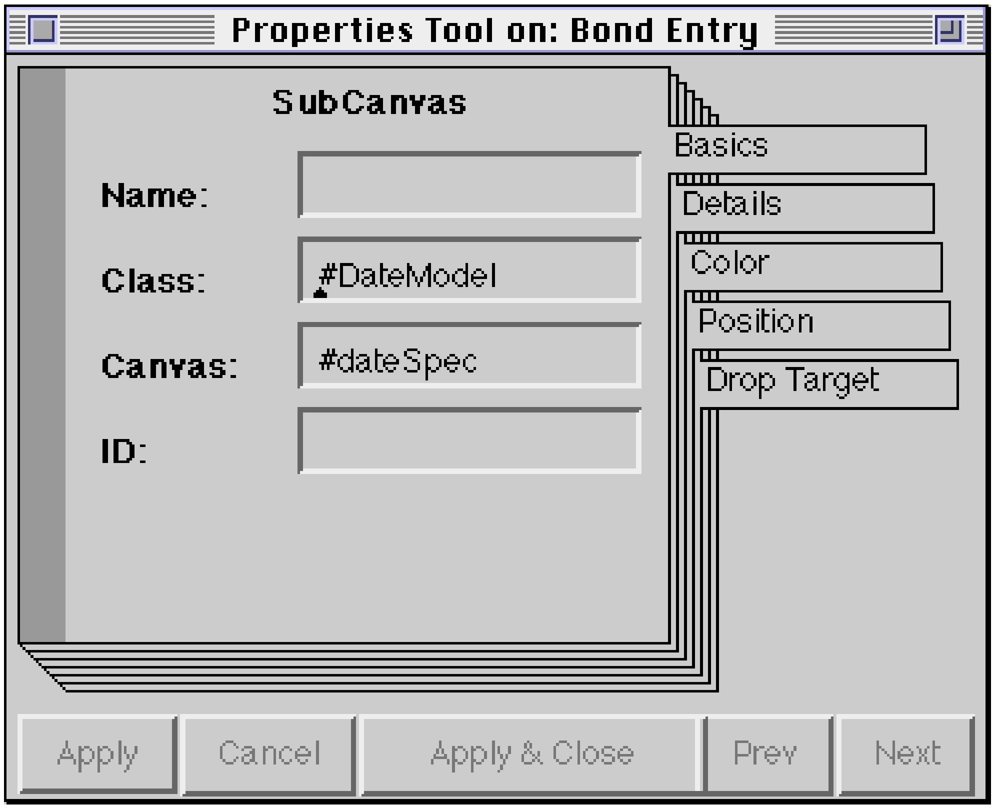
3. Reuse of a Canvas Specification

Using this technique, the Subcanvas widget is used to embed one Canvas inside another — i.e. size, location and other properties for each widget. The name of the class containing the Canvas must also be specified in the Properties Tool (see Fig.2). The model for each widget has to be provided by the class that is reusing the Canvas.

For example, if we revert to our original definition of class BondEntry, we could reuse the dateSpec Canvas of DateModel by creating a Subcanvas widget and giving it the properties in Fig.2. However, we also have to modify class BondEntry so that it provides models and behavior expected by the properties of the widgets in the embedded Canvas.

Ex 3. Revert to your original version of BondEntry and create a Subcanvas widget with the properties specified above.[1] Use the Definer to add any necessary instance variables. Modify the class (adding methods as necessary) so that the widget works correctly.

1. You may have to file–in the BondEntry class that you filed–out before Ex.2.



SubCanvas

Name:

Class: #DateModel

Canvas: #dateSpec

ID:

Figure 2: Visual Reuse via an Embedded Canvas

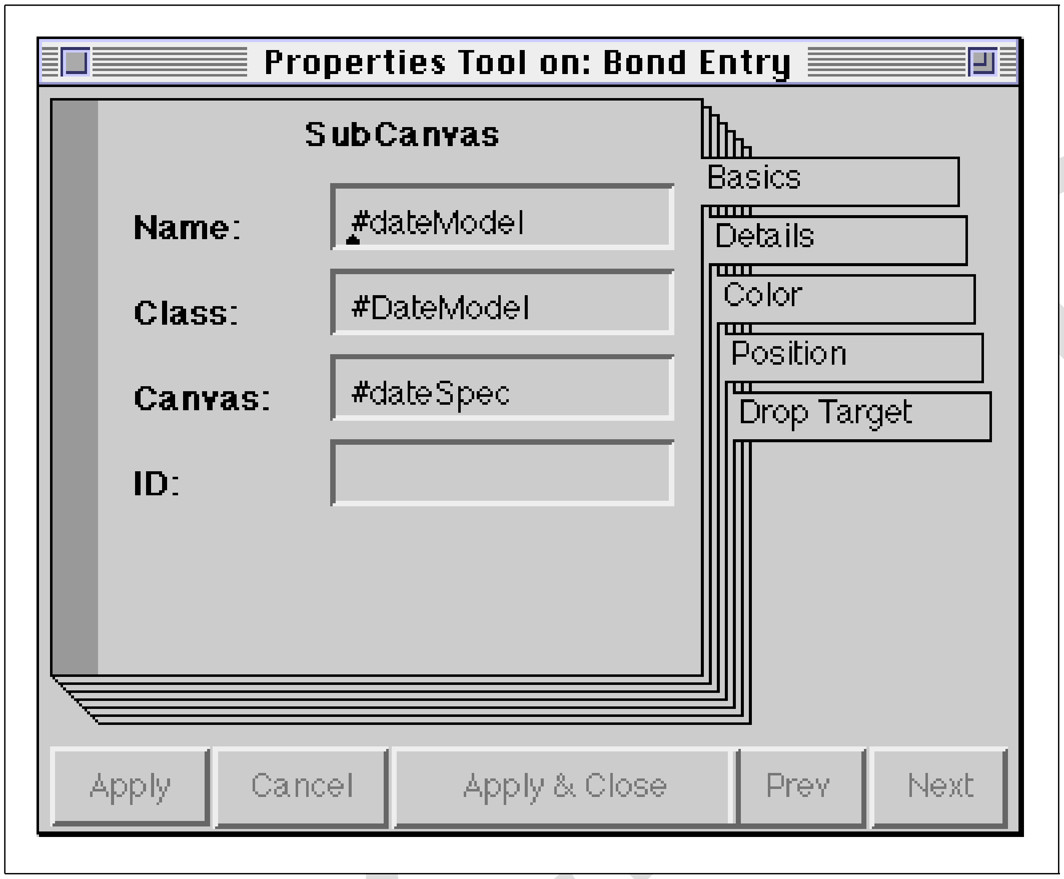
4. Embedding one Application inside another

This technique implies that a class *and* its Canvas specification is embedded in some other Canvas. It requires an additional property *Name*, which specifies the selector which will return an instance of the class which contains the Canvas specification (see Fig.3). This instance is then the receiver of the aspect messages defined by the widgets in the Canvas specification.

Ex 4. Revert back to your original version of BondEntry and create a Subcanvas widget with the properties described above. Use the Definer to create the appropriate instance variable. You should find that the widget now works as expected.

5. Visual Reuse — Summary

If a Symbol is specified for the *Name* property of a Subcanvas widget, the Definer will create an instance variable with that name and initialize it to be an instance of the class providing the Subcanvas. When the UIBuilder looks for the models of the widgets within the Subcanvas, it sends their aspect messages to this instance rather than the instance of



Name: #dateModel
Class: #DateModel
Canvas: #dateSpec
ID:

Figure 3: Visual Reuse via an Embedded Application

the application model class. Therefore, each model in the class containing the Subcanvas must be of the appropriate class for the Canvas (e.g., an instance of ValueHolder).

If the *Name* property is not specified, the UIBuilder will send the *aspect* messages defined in the widgets to an instance of the Canvas class — which in practice often means replicating all of the instance variables from the Subcanvas class to the Canvas class.

Overall, it is much easier and much more preferable to achieve visual reuse by embedding applications, one inside another. This approach avoids the cumbersome inheritance described in section 2 and the unnecessary duplication of effort identified in section 3.

6. Special Menu Options

When painting, the Subcanvas widget provides the <operate> menu with an extra item named **special** with the following options: **create sub application**, **paint sub application**, **browse sub application** and **extract contents**. The "sub application" refers to the class that provides the Subcanvas. It is possible to paint a Subcanvas when the class and its Canvas has not yet been defined. The **create sub application** menu option then creates a new class on which to install the Canvas. The remaining menu options behave as described in Table 1.

paint sub application	opens a Subcanvas for editing. If the sub application class has just been created, the Canvas is blank; if it already existed, the Canvas is opened displaying its widgets.
browse sub application	opens a Browser on the Subcanvas class.
extract contents	removes the Subcanvas wrapper, leaving the its components as separate parts of the new Canvas. This removes all trace of the other sub application class.

Table 1: Special Menu options for the Subcanvas Widget