Revision 1.0, August 7, 1997

CardView Widget

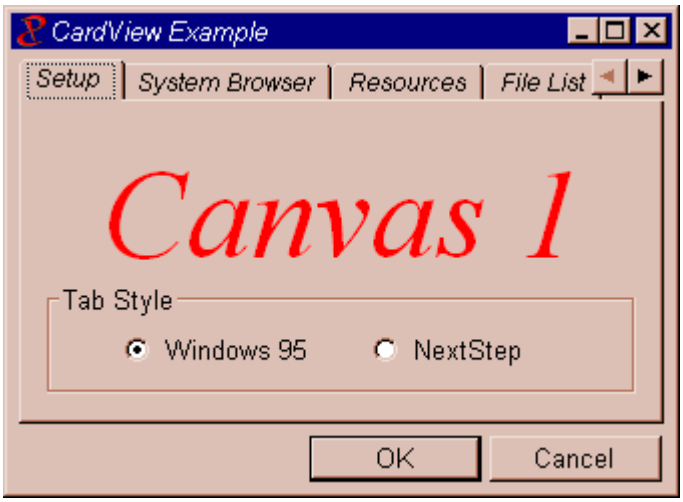
User's Guide & Cookbook

The purpose of this document is to give you an overview of the features and facilities provided by *CardView Widget* and how to access them programmatically. It starts with an introductory chapter. Subsequently step-by-step instructions show how to build applications employing CardView widgets.

Contents

Introduction

CardView widget provides functionality of tabbed controls. It is an alternative to the *Notebook* widget delivered with VisualWorks. As such it conforms to the method interface of Notebook widget, i.e. a CardView uses a *SelectionInList* as its model and a *SubCanvas* for displaying the view's contents. The differences to the Notebook widget are on the one hand a different look & feel, that rather adopts that of the Windows 95 tab controls. On the other hand, CardView doesn't support vertical tab bars nor does it support two separate tab bars at all.



User Interface

The principal behaviour of a CardView or tab control is assumed to be well known: Selecting a tab changes the view's contents to display the information associated with this tab. Changing the contents can mean either to change the contents of the widgets displayed in the card view client area or displaying a new client page (subcanvas) with completely different widgets.

The tabs are arranged in a single row at the widget's top[1]. If the tabs' width exceeds the widget's boundaries, scroller buttons are displayed at the

[1] CardView does not support stacked rows of index tabs as known from certain Windows 95 tabbed dialogs, e.g. *My Computer ® Properties*.

top right corner. Clicking on the right scroller button causes the tabs to be shifted to the left such that the obscured tabs to the right become visible. The left scroller button shifts the tabs to the right.

In addition to selecting an index tab with the mouse one can use keyboard shortcuts to navigate to a certain tab:

- **Ctrl-PgUp/PgDn** moves the selection to the next/previous tab.
- **Ctrl-Home/End** moves the selection to the first/last tab.

A CardView's tabs can have the keyboard focus. This is indicated by rectangular a border around the selected tab's label string. Having the keyboard focus results in additional keyboard shortcuts to be available. In particular:

- **PgUp/PgDn** moves the selection to the next/previous tab.
- **Left/Right** moves the selection to the next/previous tab.
- **Ctrl-Tab** moves the selection to the next tab.
- **Ctrl-Shift-Tab** moves the selection to the previous tab.
- **Home/End** moves the selection to the first/last tab.

Advanced Features

Basically, CardView provides the same interface for building sub-canvases as Notebook does: An application model sends *#client:#spec:* – or variants of this message – to the notebook widget as result to a selection changed notification from the tabs' selection index holder. This in turn results in the client app model's window spec being traced and rebuilt each time a tab is selected.

CardView provides advanced features in order to simplify the task of swapping CardView pages:

- Building sub-canvases automatically
- Caching sup-canvases.

Building sub-canvases automatically

A CardView can be configured in Canvas Painter to automatically build the corresponding page when a tab is selected, without any intervention of the application model. The necessary information to do so is provided in the widget's aspect, a SelectionInList, by using associations with the key specifying the tab's label and the value specifying the app model and spec to use for building the page. The app model can be specified either as an *ApplicationModel* subclass or as an instance of which. The spec can be omitted if it is *#windowSpec*.

Caching sub-canvases

CardView is capable to cache the contents of a page and reuse them when the same tab is selected again subsequently. Whenever a new tab is selected, the view copies the contents of the formerly displayed page to a private dictionary. When this page's associated tab is selected again, the cached components are re-installed in the subcanvas instead of rebuilding them from the client's window spec. This results in a notably faster display and can be particularly useful in certain cases, e.g. where a rebuild would require an entirely new client instance, such as with UIFinderVW2.

Summary

CardView …

- Conforms to the Notebook widget's message interface.
- Adopts the Look and Feel of Windows 95 tab controls,
- Provides advanced support for building pages automatically.
- Provides support for caching pages.
- Doesn't support vertical tab bars.

Cookbook:
How to deploy a CardView

In this chapter we will describe how to deploy CardView widgets in VisualWorks applications. This description can be divided into descriptions of the advanced features introduced with CardView and in conventional steps. Since CardView complies to the interface of Notebook, the conventional ways to deploy a CardView are quite the same as those to deploy a Notebook widget. You may compare those steps to the VisualWorks Cookbook chapter on Notebooks (Chapter 17).

Advanced: Adding a CardView

Strategy

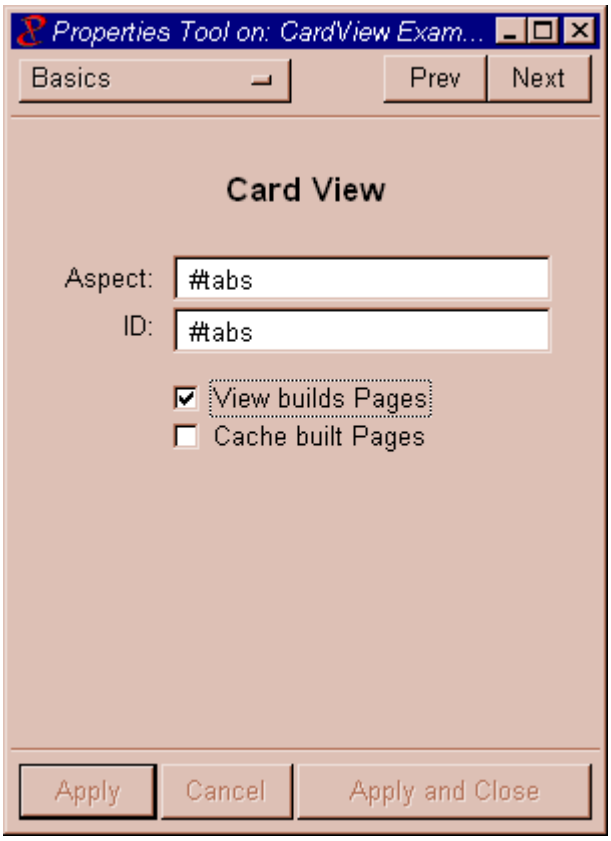
Most often a CardView is used for displaying a separate page — i.e. a different subcanvas — for each selected tab. The most convenient way to achieve this is by exploiting the advanced feature of configuring a CardView to automatically build a subcanvas's contents. The basic steps show how a CardView is added using this approach.

Basic Steps

Tutorial Example: *CardViewExample*

1. Use a Palette to add a CardView widget to your canvas. Leave the CardView selected.

2. In a Properties Tool (*Basics* page), fill in the CardView's **Aspect** property with the name of a method (*tabs*) to return a *SelectionInList* containing the labels for the index tabs.



3. Turn on the CardView's **View builds Pages** property.

4. Use a System Browser or the canvas's **define** command to create the instance variable (*tabs*) and accessing method (*tabs*) for the CardView's list of index labels.

```
tabs
    ^tabs
```

5. Initialize the aspect variable, either in the accessing method or in an *initialize* method (as in the example), with a *SelectionInList*, containing *Associations*. The key of each association specifies the tab's label string, the value specifies the parameters to build the associated subcanvas (details are described in "Analysis").

```
initialize

    tabs := SelectionInList
        with: (List new
            add: 'Setup' -> (Array with: self with: #canvas1Spec);
            add: 'System Browser' -> Browser new;
            add: 'Resources' -> UIFinderVW2;
            add: 'File List' -> FileBrowser;
            add: 'My Implementation' -> (Array
                    with: (Browser new onClass: self class)
                    with: #classBrowserSpec);
        yourself).
```

Analysis

As described above each association's value specifies the necessary parameters to build the subcanvas associated with a tab. This is basically an instance of *ApplicationModel* and a *Symbol* denoting the window spec that defines the interface of the subcanvas. These two parameters are basically provided by means of a two-elements array, as shown in the example's "Setup" page.

For the purpose of convenience, these rules can be applied:

- If the canvas's window spec is *#windowSpec*, the window spec parameter can be left out. Only the application model has to be specified as the association's value (as shown in the example's "System Browser" page).
- The application model can be specified either as an instance of *ApplicationModel* or as an application model class (as shown in the example's "Resources" page).

Specifying the application model by means of its class, results in re-instantiating the application model each time the associated tab is selected (only unless the "Cache built Pages" property is turned on). This is in many cases not desirable, thus you should use instances in these cases. However, in certain situations this very behavior is necessary. An example is the Resource Finder (class *UIFinderVW2*). Due to the implementation of this class, if the same instance would be used to repeatedly built ever new subcanvases, the subcanvas would not display spec or resource icons after a second build in the right list.

Variant A:
Setting the Starting Page

By default, a CardView shows up with a blank page (such as a Notebook does). You can set the starting page by changing the CardView aspect's selection index.

Tutorial Example: *CardViewExample*

1. In a method in the application model (such as *postOpenWith:*), send a *selectionIndex:* message to the *SelectionInList* that holds the CardView's aspect (in the example this is *tabs*). The argument is the index number of the desired page to be displayed initially.

```
postOpenWith: aBuilder
    tabs selectionIndex: 1
```

Variant B:
Using Page Caching

CardView is capable to cache the contents of a page and reuse them when the same tab is selected again subsequently. Whenever a new tab is selected, the view would copy the contents of the formerly displayed page to a private dictionary. When this page's associated tab is selected again, the cached components are re-installed in the subcanvas instead of re-building them from the client's window spec. This results in a notably faster display and can be particularly useful in certain cases, e.g. where a rebuild would require an entirely new client instance, such as with UI-FinderVW2.

Tutorial Example: *CardViewExample*

1. In a Properties Tool (*Basics* page), turn on the CardView's **Cache built Pages** property.

Conventional: Adding a CardView

Strategy

Since CardView complies to the interface of Notebook, the conventional steps to add a CardView to a canvas are quite the same as those to add a Notebook widget. In particular, a CardView also uses an instance of *SelectionInList* to hold the list of tab labels, along with a selection index holder. The major difference in this regard is that CardView does not support a secondary list of tabs. The steps described below are for the most parts copied from the VisualWorks Cookbook section on Notebooks (Chapter 17).

Basic Steps

Tutorial Example: *CardViewExample1*

1. Use a Palette to add a CardView widget to your canvas. Leave the CardView selected.

2. In a Properties Tool (*Basics* page), fill in the CardView's **Aspect** property with the name of a method (*tabs*) to return a *SelectionInList* containing the labels for the index tabs.

3. In the CardView's **ID** property enter an identifying name for the widget (*tabs*).

4. Apply the properties and install the canvas.

5. Use a System Browser or the canvas's **define** command to create the instance variable (*tabs*) and accessing method (*tabs*) for the CardView's list of index labels.

```
tabs
    ^tabs
```

6. Initialize the variable, either in the accessing method or in an *initialize* method (as in the example), with a *SelectionInList*, containing either strings or associations.

```
initialize

    tabs := SelectionInList
        with: #('Views' 'Controllers' 'Models' 'Application Models').
    tabs selectionIndexHolder onChangeSend: #pageChanged to: self.

    classes := SelectionInList new.
```

7. Create a second canvas for the interface that is to be shown inside the CardView. Install this canvas in its own resource method (*canvasSpec*).

8. Use a System Browser or the canvas's **define** command to create any variables and methods needed by the subcanvas. (In the example there are the *classes* variable, the *classes* method and the *initialize* method).

9. In the *initialize* method, use a *onChangedSend:to:* message to arrange for the CardView to send a message (*pageChanged*) to the application model whenever the user selects an index tab.

```
initialize

    tabs := SelectionInList
    with: #('Views' 'Controllers' 'Models' 'Application Models').
    tabs selectionIndexHolder onChangeSend: #pageChanged to:
self.

    classes := SelectionInList new.
```

10. Create the change notification method (*pageChanged*) in which the subcanvas contents are updated, based on the index tab that has been selected. (In the example, the *classes* list is updated with appropriate classes).

---

pageChanged

```
| chosenTab rootClass |
chosenTab := tabs selection.
chosenTab = 'Views' ifTrue: [rootClass := View].
chosenTab = 'Controllers' ifTrue: [rootClass := Controller].
chosenTab = 'Models' ifTrue: [rootClass := Model].
chosenTab = 'Application Models' ifTrue: [rootClass := Application-
Model].
classes list: rootClass withAllSubclasses
```

---

11. Create a *postOpenWith:* method. In this method, first get the CardView from the application model's builder, using the CardView's **ID** (*tabs*). Then install the subcanvas by sending a *client:spec:* message to the CardView. The first argument is the subapplication's application model (in the example it's *self*). The second argument is the name of the spec method (*listSpec*) that defines the desired canvas.

---

postOpenWith: aBuilder

```
(aBuilder componentAt: #tabs) widget
client: self
spec: #canvasSpec.
tabs selectionIndex: 1
```

---

12. In the *postOpenWith:* method send message *selectionIndex:* to the CardView's aspect variable to arrange for initial contents to be displayed (see above).

Variant A:
Using a list of associations as the CardView's tab labels

Tutorial Example: *CardViewExample2*

Instead of providing a simple list of label strings, you can initialize a CardView with a list of *Associations*. The key of each association specifies a tab's label string. The associated value can carry any information of use for application model to identify a selected tab. (Compare "Determining which tab is selected, Variant C").

```
initialize

        classes := SelectionInList new.

        tabs := SelectionInList with:
            (List new
                add: 'Views' -> View;
                add: 'Controllers' -> Controller;
                add: 'Models' -> Model;
                add: 'Application Models' -> ApplicationModel;
                add: 'My Examples' -> #myExamples;
            yourself).
        tabs selectionIndexHolder onChangeSend: #pageChanged to: self.
```

Conventional: Changing the Page Layout (Subcanvas)

Strategy

The conventional way — as opposed to automatically building pages — of displaying a different interface on a page is supported by sending a *client:spec:* message to the CardView. This is also compliant to the Notebook alternative. Thus again, the steps described below are for the most parts copied from the VisualWorks Cookbook section on Notebooks (Chapter 17).

Basic Steps

Tutorial Example: *CardViewExample3*

1. In the application model's *initialize* method, arrange for the SelectionInList that holds the CardView's aspect to notify the application model when a tab is selected by sending *onChange:Send:* to the *selectionIndexHolder*. (In the example, a *pageChanged* method is triggered.)

```
initialize

        tabs := SelectionInList with: #(First Second Third).
        tabs selectionIndexHolder onChangeSend: #pageChanged to:
    self.
```

2. In the notification method (*pageChanged*), get the CardView widget from the application model's builder via *componentAt:*. Send a *client:spec:* message to the CardView.

```
pageChanged

        | wrapper subcanvasSpec |
        subcanvasSpec := 'canvas' , tabs selectionIndex printString ,
    'Spec'.
        (wrapper := builder componentAt: #tabs) isNil ifTrue: [^self].
        wrapper widget
            client: self
            spec: subcanvasSpec asSymbol
```

Analysis

A CardView forwards the *client:spec:* messages — including variants such as *client:* and *client:spec:builder* — to a private subcanvas. The first argument (*client*) is an instance of the desired application model (in the example, this is the example class itself). The second argument (*spec*) is the name of the desired window spec (in the example this is *canvas1Spec*, *canvas2Spec* or *canvas3Spec*). The third argument is a *UIBuilder* to be used for building the subcanvas. If the spec argument is omitted (message *client:*), *#windowSpec* is assumed. If the builder argument is omitted (message *client:spec:*), a new builder is created.

Setting the Starting Page

Strategy

By default, a CardView shows up with a blank page (such as a Notebook does). You can set the starting page by changing the CardView aspect's selection index.

Basic Steps

Tutorial Example: *CardViewExample*

1. In a method in the application model (such as *postOpenWith:*), send a *selectionIndex:* message to the *SelectionInList* that holds the CardView's aspect (in the example this is *tabs*). The argument is the index number of the desired page to be displayed initially.

```
postOpenWith: aBuilder
        tabs selectionIndex: 1
```

Determining which tab is selected

Strategy

Detecting which tab of a CardView is selected is done in exactly the same way as with Notebook widgets, i.e. getting and comparing the selected tab's label string, its index number or its associated value. (Compare the following with the corresponding section in the VisualWorks Cookbook).

Variant A:
Getting the selected tab's label

Tutorial Example: *CardViewExample1*

1. In a method in the application model, get the selected tab's label string by sending a *selection* message to the CardView's aspect variable (a *SelectionInList*).

---

pageChanged

```
| chosenTab rootClass |
chosenTab := tabs selection.
chosenTab = 'Views' ifTrue: [rootClass := View].
chosenTab = 'Controllers' ifTrue: [rootClass := Controller].
chosenTab = 'Models' ifTrue: [rootClass := Model].
chosenTab = 'Application Models' ifTrue: [rootClass := Application-
Model].
classes list: rootClass withAllSubclasses
```

Variant B:
Getting the selected tab's index number

Tutorial Example: *CardViewExample1*

1. In a method in the application model, get the selected tab's index number by sending a *selectionIndex* message to the CardView's aspect variable (a *SelectionInList*).

---

pageChanged

```
| chosenTab rootClass |
chosenTab := tabs selectionIndex.
rootClass := #(View Controller Model ApplicationModel) at: chosenTab.
classes list: (Smalltalk at: rootClass) withAllSubclasses
```

Variant C:
Getting the value associated with the selected tab

Tutorial Example: *CardViewExample2*

1. Initialize the CardView's aspect variable with a list of associations as shown in section "Adding a CardView, Variant A" above.

2. In a method in the application model, get the selected tab's association by sending a *selection* message to the CardView's aspect variable (a *SelectionInList*).

3. Send a *value* message to the resulting association. (In the example the value is either the symbol *#myExamples* or a class.

---

pageChanged

```
        | newContents selection |
    selection := tabs selection value.
        newContents :=
        (selectedTab == #myExamples)
                ifTrue: [Smalltalk keys select: [:each | 'CardViewExample*' match:
    each]]
                ifFalse: [selection withAllSubclasses collect: [:each | each name]].
        classes list: newContents asSortedCollection
```

Setting the Tabs' Appearance

Strategy

The CardView class can be configured to either display the tabs in a Windows 95 conformant look or in an appearance that resembles that of CardViews in NextStep.

Basic Steps

Tutorial Example: *CardViewExample*

1. Send message *tabStyle:* to the CardView class. The argument is either the Symbol *#slanted* for Nextstep-like tabs or *#straight* for Windows 95-like tabs.

```
CardView tabStyle: #slanted.
    "or"
CardView tabStyle: #straight.
```

(You can inquire the current tab style by sending message *tabStyle* to class *CardView*.)

Index