

# Spec – Technical Report

Benjamin Van Ryseghem  
RMod, Inria Lille – Nord Europe

June 19, 2012

## 1 Code

The code is available on squeaksource:

```
Gofer new
  url: 'http://ss3.gemstone.com/ss/Spec';
  package: 'ConfigurationOfSpec';
  load.
```

```
(Smalltalk at: #ConfigurationOfSpec) perform: #loadFull.
```

## 2 Introduction

This paper presents the basic use of Spec by showing an example of how to build a Spec UI, how to reuse one and how to specify sub-widgets. This paper also provides the full public API of the basic widgets.

## 3 Example

The most important point of Spec is the reuse and the possibility of composition at two levels, UI and models. This section shows how to build an abstract method browser (similar to the senders/implementor ) and how to reuse this browser to build an extended class browser (similar to the code browser).

Here comes a simple example to quickly introduce to the use Specs. For now do not try to understand all the details but focus on the results. We will provide detailed insights right after the example.

### 3.1 Methods Browser

Firstly, let's think about how the visual structure of a method browser GUI. It is basically composed of two areas: a list and a text zone. Moreover there is an entry point which is the list of methods to be browsed.

So let's define a MethodBrowser class

```
ComposableModel subclass: #MethodBrowser
  instanceVariableNames: 'listModel textModel'
  classVariableNames: ''
```

```
poolDictionaries: ''
category: 'Spec-Examples'
```

Class 1: *MethodBrowser class*

We omit here the getter for a matter of space. Nevertheless they will be needed by the SpecLayout.

It's still miss the entry point. By the way the entry point of the method browser could also be the list model's entry point.

```
methods: aList
```

```
"Here I reroute my entry point to the list model's entry point"
self listModel items: aList
```

So now we have the entry point and the models. So lets make the connections between them.

```
initializeWidgets
```

```
self instantiateModels:
{
    #listModel -> #ListComposableModel.
    #textModel -> #TextModel.
}

textModel aboutToStyle: true.
```

```
initializePresenter
```

```
listModel whenSelectedItemChanges: [:selection |
    selection
        ifNil: [
            textModel text: ''.
            textModel behavior: nil ]
        ifNotNil: [:m |
            textModel text: m sourceCode.
            textModel behavior: m methodClass ]].
```

Now that links are created, we have to specify the spec on the class side:

```
spec
<spec: #default>

    ↑ SpecLayout composed
        add: #listModel origin: 0@0 corner: 1@0.5;
        add: #textModel origin: 0@0.5 corner: 1@1;
        yourself.
```

We specify the layout to be a column with submorphs added from top to bottom. And then for each model we add its spec recursively.

The below code produce the window shown in the Figure 1.

```
| browser |
browser := MethodBrowser new.
browser openWithSpec.
```

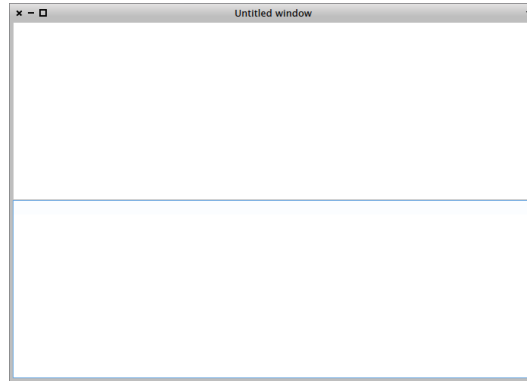


Figure 1: Our browser with an empty list

Then to populate the widget, the method `methods` can be used, as shown in the following example where the widget is populated with the methods of `ComposableModel` and `ListComposableModel`.

```
browser methods: (ComposableModel methods , ListComposableModel methods).
```

The list should automatically be updated, and looks like Figure 2.

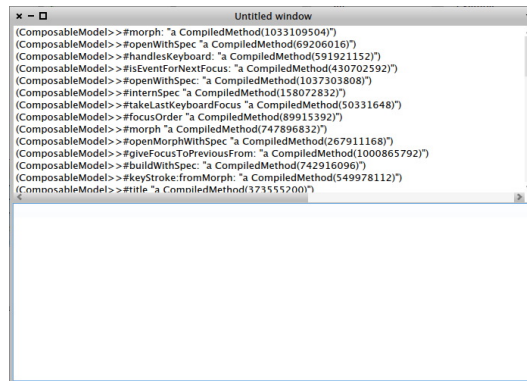


Figure 2: Our browser with list of methods

And if a method is selected its source code should appear, like Figure 3

We are almost done, only the window title is still wrong. So we will define

```
title
```

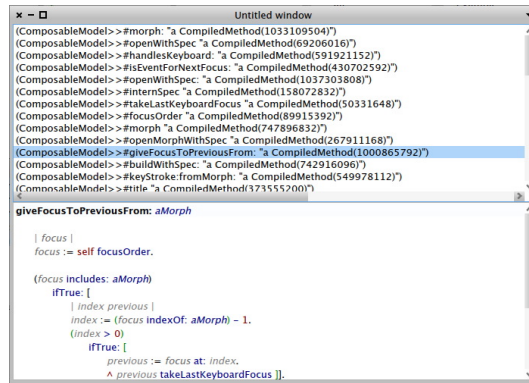


Figure 3: Our browser with a method selected

↑ 'Method Browser'

So now, if the code is re-evaluated, the result looks like on Figure 4

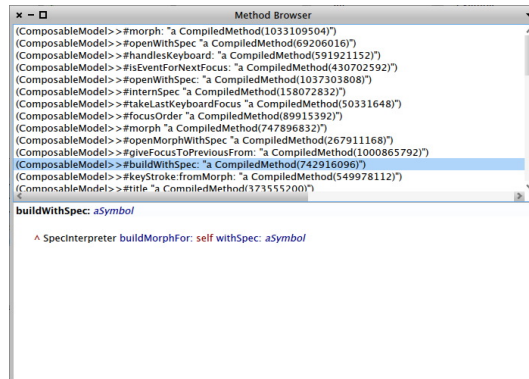


Figure 4: Now with a nice title

We can still improve the way the items are displayed. We just have to precise to the list model how to wrap items. So let's add the following line into `MethodBrowser>>#initializeWidgets`:

```
listModel displayBlock: [:item | item methodClass name, '>>#', item selector ].
```

The items display could also be seen as a new entry point, and define

```
displayBlock: aBlock
```

```
listModel displayBlock: aBlock
```

We can now use this method into `MethodBrowser>>#initializeWidgets` to finally obtain `initializeWidgets`

```
self instantiateModels:
{
    #listModel -> #ListComposableModel.
    #textModel -> #TextModel.
}

textModel aboutToStyle: true.
self displayBlock: [:item | item methodClass name,'>>#', item selector ].
```

Then the resulting browser looks like Figure 5

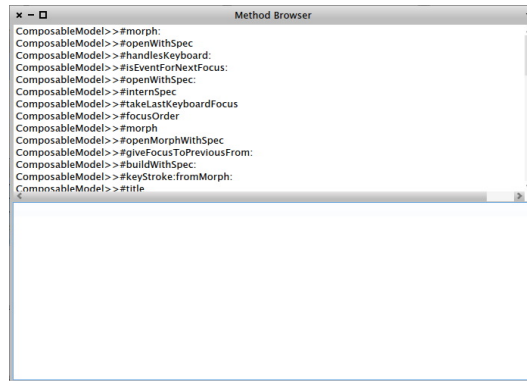


Figure 5: Voila

### 3.2 Classes and method browser

Now, we want to build a class and method browser. To do that we need a list for class and a method browser. As previously the entry point is a list but this time a list of methods.

```
ComposableModel subclass: #ClassMethodBrowser
    instanceVariableNames: 'listModel methodModel'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Spec-Examples'
```

Class 2: *ClassMethodBrowser class*

As previously, we omit here the getters.

For the entry point, it's déjà vu:

```
classes: aList
```

```
self listHolder contents: aList
```

Now we have to create the link. It's simpler here than before because there is only one link to create between the list of classes and the method browser:

```
initializeWidgetsinitializeWidgets
```

```
self instantiateModels:
{
    #listModel -> #ListComposableModel.
    #methodModel -> #MethodBrowser.
}
```

```
methodModel displayBlock: [:method | method selector ].
```

```
initializePresenter
```

```
listModel whenSelectedItemChanges: [:selection |
    selection
        ifNotNil: [:class |
            methodModel methods: (class methodDict values sort: [:a :b | a selector < b
selector])].
            methodModel listModel resetSelection ]].
```

So now, let's do the spec.

```
defaultSpec
```

```
<spec>
```

```
↑ SpecLayout composed
```

```
add: #listModel origin: 0@0 corner: 0.5@0.5;
add: #(methodModel listModel)origin: 0.5@0 corner: 1@0.5
add: #(methodModel textModel) origin:0@0.5 corner:1@1;
yourself.
```

In each spec we call the spec of our internal models.

Lastly, we can specify a title for the window:

```
ClassMethodBrowser class>>#title
```

```
↑ 'Class Method Browser'
```

Then if the following code is evaluated the result looks like in Figure 6.

```
| browser |
browser := ClassMethodBrowser new.
browser openWithSpec.
browser classes: (Smalltalk allClasses).
```

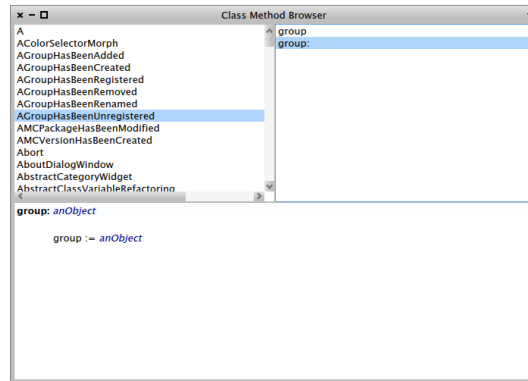


Figure 6: Our classes and method browser with a method selected

### 3.3 Conclusion

Here ends this example which present how to implement basic widget and how to compose Spec widgets together to get new powerful widgets.

Those examples (with few improvements) and few others are provided in the package `Spec-Example`, feel free to have a look.

## 4 ComposableModel API

`ComposableModel` is an abstract class which is the superclass of all widget models used with specs. The two main goals of `ComposableModel` are to link the current instance of the model with its `specLayout` via the interpreter, and to provide convenience methods used among a lot of widgets.

The Table 1 provides the full public API of `ComposableModel`.

## 5 Basic widgets API

Selector	Result
buildWithSpec	Build the widget using the default spec
buildWithSpec:	Build the widget using the spec name provided as argument
isDisplayed	Return true if the widget is currently displayed on screen
openWithSpec	Build the widget using the default spec and display it into a window
openWithSpec:	Build the widget using the spec name provided as argument and display it into a window
specSelectors	Return all the spec names
title	Return the window's title
updateTitle	Update the window title
eventStringForNextFocus	String describing the keystroke to perform to jump to the next widget
eventStringForPreviousFocus	String describing the keystroke to perform to jump to the previous widget
isEventForNextFocus:	Return true if the event provided as argument corresponds to the shortcut used to switch to the next sub-widget
isEventForPreviousFocus:	Return true if the event provided as argument corresponds to the shortcut used to switch to the previous sub-widget

Table 1: ComposableModel public API

Selector	Result
action:	set the block performed when the button is clicked
enabled:	set if the button is enabled (clickable)
label:	set the label of the button
state:	set if the button is highlighted
whenActionPerformedDo:	set a block to perform after that the button has been aclicked, and its action performed

Table 2: ButtonModel public API



Selector	Result
activationAction:	This method is used to set the action to perform when I am activated
click	Simulate a click on the checkbox Used when the checkboc is a list item
desactivationAction:	This method is used to set the action to perform when I am desactivated
enabled:	Set if the button is enabled (clickable)
label:	Set the label of the checkbox
labelClickable	Return true if the label can be clicked to select the checkbox
labelClickable:	Set if the label can be clicked to select the checkbox
state:	Set if the checkbox is activated or not
toggleState	Toogle the current state of the checkbox
whenActivatedDo:	This method is used to propagate the event that I have been activated
whenDesactivatedDo:	This method is used to propagate the event that I have been desactivated

Table 3: CheckBoxModel public API

Selector	Result
items:	Populate the drop list with a list of DropItems
listItems	Return the list of DropItems used to populate the drop list
resetSelection	Reset the current selection state
selectedIndex	Useless method but it provides a better and more consistent API
selectedItem	Return the selected item
setSelectedIndex:	Force the selection of the item at index anIndex
setSelectedItem:	Force the selection of the item anItem
whenSelectedItemChanged:	Set a block to perform when the selected item is changed
whenSelectionChanged:	Set a block to perform when the selection is changed
whenSelectionIndexChanged:	Set a block to perform when the selected index is changed

Table 4: DropListModel public API

Selector	Result
allowToSelect	Return whether the list items can be selected or not
allowToSelect:	Set if the list items can be selected or not
beMultipleSelection	Make list selection multiple
beSingleSelection	Make list selection single
displayBlock:	Set the one argument block used to wrap your domain specific items. The block should return something that can be displayed in a list - like a String or a Text
filteringBlock:	To set the filtering of the items
icons:	Set a block which takes an item as argument and returns the icon to display in the list
items:	Set the items of the list. aList is a collection of your domain specific items
listItems	Return the items of the list. They are your domain specific items
menu:	Set the block used to defined the menu
multiSelection	Return true if the list has a multiple selection. False if the list has a single selection
multiSelection:	Make the list seelction become multiple if aBoolean is true. Otherwise set the selection as single
resetFilteringBlock	Reset the filtering block with the default value which consists in showing everything
resetSelection	Unselect every items
resetSortingBlock	Reset the sorting block with the default value which consists in not sorting
selectedIndex	Return the index of the selected item In the case of a multiple selection list, it returns the last selected item
selectedIndexes	Return the indexes of selected items on the case of a multiple selection list
selectedItem	Return the selected item. In the case of a multiple selection list, it returns the last selected item
selectedItems	Return all the selected items in the case of a multiple selection list
setSelectedIndex:	Set the index of the item you want to be selected
setSelectedItem:	Set the item you want to be selected
shortcuts:	Set the block used to defined the shortcuts
sortingBlock:	To set the ordering of the items
updateList	Refresh the list
whenListChanged:	Specify a block to value after the contents of the list has changed
whenSelectedItemChanged:	Set a block to value when the select item is changed
whenSelectionChanged:	Set a block to value when the selection of the list has changed
whenSelectionIndexChanged:	Set a block to value when the selection index has changed

Table 5: IconListModel public API

Selector	Result
disable	Disable the label
enable	Enable the label
enabled:	Set the state of the label
text:	Set the text of the label
whenTextChanged:	Set a block to performed when the text is changed

Table 6: LabelModel public API

Selector	Result
allowToSelect	Return whether the list items can be selected or not
allowToSelect:	Set if the list items can be selected or not
beMultipleSelection	Make list selection multiple
beSingleSelection	Make list selection single
displayBlock:	Set the one argument block used to wrap your domain specific items. The block should return something that can be displayed in a list - like a String or a Text
filteringBlock:	To set the filtering of the items
items:	Set the items of the list. aList is a collection of your domain specific items
listItems	Return the items of the list. They are your domain specific items
menu:	Set the block used to defined the menu
multiSelection	Return true if the list has a multiple selection. False if the list has a single selection
multiSelection:	Make the list seelction become multiple if aBoolean is true. Otherwise set the selection as single
resetFilteringBlock	Reset the filtering block with the default value which consists in showing everything
resetSelection	Unselect every items
resetSortingBlock	Reset the sortering block with the default value which consists in not sorting
selectedIndex	Return the index of the selected item In the case of a multiple selection list, it returns the last selected item
selectedIndexes	Return the indexes of selected items on the case of a multiple selection list
selectedItem	Return the selected item. In the case of a multiple selection list, it returns the last selected item
selectedItems	Return all the selected items in the case of a multiple selection list
setSelectedIndex:	Set the index of the item you want to be selected
setSelectedItem:	Set the item you want to be selected
shortcuts:	Set the block used to defined the shortcuts
sortingBlock:	To set the ordering of the items
updateList	Refresh the list
whenListChanged:	Specify a block to value after the contents of the list has changed
whenSelectedItemChanged:	Set a block to value when the select item is changed
whenSelectionChanged:	Set a block to value when the selection of the list has changed
whenSelectionIndexChanged:	Set a block to value when the selection index has changed

Table 7: ListComposableModel public API

Selector	Result
allowToSelect	Return whether the list items can be selected or not
allowToSelect:	Set if the list items can be selected or not
beMultipleSelection	Make list selection multiple
beSingleSelection	Make list selection single
displayBlock:	Set the one argument block used to wrap your domain specific items. The block should return something that can be displayed in a list - like a String or a Text
filteringBlock:	To set the filtering of the items
items:	Set the items of the list. aList is a collection of your domain specific items
listItems	Return the items of the list. They are your domain specific items
menu:	Set the block used to defined the menu
multiSelection	Return true if the list has a multiple selection. False if the list has a single selection
multiSelection:	Make the list seelction become multiple if aBoolean is true. Otherwise set the selection as single
resetFilteringBlock	Reset the filtering block with the default value which consists in showing everything
resetSelection	Unselect every items
resetSortingBlock	Reset the sortering block with the default value which consists in not sorting
selectedIndex	Return the index of the selected item In the case of a multiple selection list, it returns the last selected item
selectedIndexs	Return the indexes of selected items on the case of a multiple selection list
selectedItem	Return the selected item. In the case of a multiple selection list, it returns the last selected item
selectedItems	Return all the selected items in the case of a multiple selection list
setSelectedIndex:	Set the index of the item you want to be selected
setSelectedItem:	Set the item you want to be selected
shortcuts:	Set the block used to defined the shortcuts
sortingBlock:	To set the ordering of the items
updateList	Refresh the list
whenListChanged:	Specify a block to value after the contents of the list has changed
whenSelectedItemChanged:	Set a block to value when the select item is changed
whenSelectionChanged:	Set a block to value when the selection of the list has changed
whenSelectionIndexChanged:	Set a block to value when the selection index has changed

Table 8: MultiColumnListModel public API

Selector	Result
activationAction:	This method is used to set the action to perform when I am activated
click	Simulate a click on the radioButton. Used when the checkbox is a list item
desactivationAction:	This method is used to set the action to perform when I am deactivated
enabled:	Set if the button is enabled (clickable)
label:	Set the label of the radioButton
labelClickable	Return true if the label can be clicked to select the radioButton
labelClickable:	Set if the label can be clicked to select the radioButton
state:	Set if the checkbox is activated or not
toggleState	Toogle the current state of the radioButton
whenActivatedDo:	This method is used to propagate the event that I have been activated
whenDesactivatedDo:	This method is used to propagate the event that I have been deactivated

Table 9: RadioButtonModel public API

Selector	Result
accept	Accept the current pendingtext
acceptBlock:	Set the block to perform when the text is accepted. The block must have one argument, which will be the accepted text
acceptOnCR	Return true if the text is accepted when the Enter key is stroked
acceptOnCR:	Set if the text is accepted when the Enter key is stroked or not
autoAccept	Return true if the text is accepted after each keystroke
autoAccept:	Set if the text is accepted after each keystroke or not
enabled	Return if the text zone is enabled
enabled:	Set if the text zone is enabled
entryCompletion:	Set an entry completion used to suggest text while typing
getSelection	Get the text selection
ghostText	Return the ghost text of the text zone
ghostText:	Set the ghost text of the text zone
hasEditingConflicts	Return if the text zone has editing conflicts
hasEditingConflicts:	Set if the text zone has editing conflicts
hasUnacceptedEdits	Return if the text zone has unaccepted edits (orange corner)
hasUnacceptedEdits:	Return if the text zone has unaccepted edits (orange corner)
pendingText	Return the current pending text
pendingText:	Set the pending text. Do not accept it
readSelectionBlock	Return the block used to calculate the text selection
readSelectionBlock:	Set the block used to calculate the text selection
removeEntryCompletion	Remove the entry completion
setSelection:	Set the text selection without changing the readSelectionBlock
text:	Set the text of the text zone
whenTextAccepted:	Set a block to be performed when the text is accepted
whenTextChanged:	Set a block to be performed when the text changes
aboutToStyle:	Set if the text zone must be styled
aboutToStyleBlock	Return the block used to know if the text must be styled
aboutToStyleBlock:	Set the block used to know if the text must be styled. The block must return a boolean
behavior	Return the class corresponding to the method class of the source code you are editing
behavior:	Set the class corresponding to the method class of the source code you are editing
isAboutToStyle	Return if the text zone is styled or not

Table 10: TextFieldModel public API

Selector	Result
accept	Accept the current pending text
acceptBlock:	Set the block to perform when the text is accepted. The block must have one argument, which will be the accepted text
enabled	Return if the text zone is enabled
enabled:	Set if the text zone is enabled
getSelection	Get the text selection
hasEditingConflicts	Return if the text zone has editing conflicts
hasEditingConflicts:	Set if the text zone has editing conflicts
hasUnacceptedEdits	Return if the text zone has unaccepted edits (orange corner)
hasUnacceptedEdits:	Set if the text zone has unaccepted edits (orange corner)
pendingText	Return the current pending text
pendingText:	Set the pending text. Do not accept it
readSelectionBlock	Return the block used to calculate the text selection
readSelectionBlock:	Set the block used to calculate the text selection
setSelection:	Set the text selection without changing the readSelectionBlock
text:	Set the text of the text zone
whenTextAccepted:	Set a block to perform when the text is accepted
whenTextChanged:	Set a block to perform when the text changed
aboutToStyle:	Set if the text zone must be styled
aboutToStyleBlock	Return the block used to know if the text must be styled
aboutToStyleBlock:	Set the block used to know if the text must be styled. The block must return a boolean
behavior	Return the class corresponding to the method class of the source code you are editing
behavior:	Set the class corresponding to the method class of the source code you are editing
isAboutToStyle	Return if the text zone is shouted or not

Table 11: TextModel public API

Selector	Result
childrenBlock:	Set the block used to retrieve the children of a node. The optional block arguments are: - the node - the tree
displayBlock:	Set the block used to generate the display of the items. The optional block arguments are: - the item - the tree
menu:	Set the block used to generate the tree menu
roots:	Set the tree roots
selectedItem	Return the selected item
updateTree	Force the tree to refresh
whenSelectedItemChanged:	Set a block to perform when a new item is selected

Table 12: TreeModel public API