Chapter 1

Keymappings

Since its 2.0 release, Pharo includes the Keymappings library. Keymappings is a library for configuring shortcuts for the current UI library (Morphic). It models concepts like: shortcuts, key combinations and event bubbling. It is a very simple library which weâĂŹll introduce gradually in this post. We gradually present the key concepts.

1.1 Key combinations

Keymappings main task is its ability to associate a key combination to an action. So we have to build up those key combinations. The simplest key combination is the one that gets activated when a single key is pressed. We call these combinations single key combinations. We create them using the message asShortcut.

```
$a asShortcut. "single key combination for the key a."
Character cr asShortcut.   "single key combination for the space key."
```

Usually key combinations get a bit more complex. It is very common to combine single keys with meta keys or modifiers. These meta keys or modifiers are the well known ctrl, shift, alt and command keys. To build a modified key combination we can do as follows:

```
$a ctrl. "a modified key combination for Ctrl+A"
$a ctrl shift.   "a modified key combination for Ctrl+Shift+A"
```

important It is important to notice that all key combinations are not case sensitive. It takes a and A characters as the same, since they are the same key.

Have you ever used emacs, Eclipse or Visual Studio? Then you probably know sequences of key combinations that launch one only action. Like Alt+Shift+X, T (to run JUnit tests in eclipse)? So keymappings can do that too:

```
$a command shift, $b shift. "key sequence (Cmd+Shift+A, Shift+B)"
```

Sometimes, you want to configure an action to be activated in two different cases. Those are Keymapping options, and get activated when one of the options gets activated:

```
$a command | $b command        "key combination (Cmd+A or Cmd+B)"
```

Finally, since Pharo is a cross platform system and it is important to provide a good user experience by with the most suitable shortcut layout, keymapping implements platform specific shortcuts, which get activated only when running in the specific platform:

```
$a command win | $b command unix     "Cmd+A on windows, but Cmd+B on unix"
```

1.2 Shortcut configurations

Now you know how to build key combinations for your purposes, you probably want to go to the action. Map those combinations to actions and make them work!

Single shortcut configuration

The simplest way to attach a shortcut to a morph is by sending him the on:do: message. The first argument expected is a key combination and the second one is an action. In the example below, a workspace is created with two shortcuts:

- when Cmd+Shift+A is pressed, the workspace is deleted
- when Cmd+Shift+D is pressed, an information growl should appear yelling 'this shortcut works!'

```
w:= Workspace new.
morph := w openLabel: 'keymapping test'.
morph on: $a shift command do: [ morph delete ].
morph on: $d shift command do: [ UIManager default inform: 'this shortcut works!' ].
```

Easy, huh? So let's move on...

Shortcut categories

Sometimes you want to group and organize shortcuts in a meaningful way and apply them all together on a morph. Sometimes you want some morphs from different hierarchies to share the same group of shortcuts easily. Those groups of shortcuts are what keymapping calls Categories. A category is a group of shortcuts, so far (will change in the future) defined statically by using a keymap pragma on class side:

```
"defining a category"
SystemWindow class>>buildShortcutsOn: aBuilder
    <keymap>
```

A class side method marked as $<$keymap$>$ will be called with a builder object, which can be used to define a named set of shortcuts:

```
SystemWindow class>>buildShortcutsOn: aBuilder
   <keymap>
   (aBuilder shortcut:  #close)
       category:  WindowShortcuts
       default: $w ctrl | $w command mac
       do: [ :target | target delete ]
       description: 'Close this window'.
```

Shortcuts defined through the builder specify the name of the category they belong to, a default key combination, an action, and a description. All this metadata is there to be used as settings in the future.

Finally to get your morph handle those shortcuts you can use the `attachKeymapCategory:` message as in:

```
w:= Workspace#new.
morph:= w openLabel: 'keymapping test'.
morph attachKeymapCategory: #Growling.
```

remove a key binding ???

1.3 Bubbling

Keymappings' shortcuts bubble to their parent (sd: which one?) if not handled, up until the main world morph. That has two main consequences:

- Shortcuts for your application can be designed in a hierarchical way and;
- Every time a shortcut does not work for you, it means that a morph below you has handled it ;) (be careful with text editors that handle many of key combinations)

SD: can’t we disable them?