

# Magritte Tutorial

Lukas Renggli  
renggli@iam.unibe.ch  
www.lukas-renggli.ch

Software Composition Group  
Institut für Informatik und angewandte Mathematik  
University of Bern, Switzerland

February 5, 2007

## 1 Getting Started

Fire up the prepared image and get yourself ready for the exercises. Make sure your Seaside server is running within the image by browsing the counter application at <http://localhost:8080/seaside/counter>.

The image you are using has been created from a fresh Squeak 3.9 image, by loading the following packages from SqueakMap: *Shout*, *eCompletion* and *Magritte*. Also a package called *Magritte-Tutorial* has been added, where you can put your code.

You can easily create your own development image by taking a fresh 3.7, 3.8 or 3.9 image and loading Magritte from SqueakMap yourself, note that your computer will be busy for a while because this will pull in a few other prerequisites (such as a Web server and Seaside).

The exercises start simple and with detailed instructions on how to perform the given tasks. Exercises marked with a star are a bit trickier, you might want to solve them later on. Sometimes you will probably not exactly know what class to use, what method to call or what parameters to pass, use the power of Smalltalk (senders, implementers, references, ...) to browse the source-code of Magritte, you might even discover some other features that were not presented during the lecture.

## 2 Juggle with Descriptions

Have a look at the source-code of the classes `MAPersonModel`, `MAAddressModel`, and `MAPhoneModel`, that have been presented during the lecture and that are pre-installed in your image. Browse to <http://localhost:8080/seaside/person-editor> and check if you can see all the features presented during the lecture. All the following exercises will be built upon this simple model.

**Exercise 1** Add a new field that holds a *Comment* about the person, display it as a text-area field as the last element. Test it in the Web browser by starting a new session on the same application. If you are required to add more than one new method by hand you did something wrong.

**Exercise 2** Add a new field that holds the *Nationality* of the person, display it as a sorted drop-down box with a few selectable countries. Put it right after the *Address* field. The default choice for new objects should be *Switzerland*.

**Exercise 3** Add a new field that holds the *E-Mail* of the person, display it as a required text-field. Add custom conditions to force the user to give a valid e-mail address. Don't allow addresses from the providers `hotmail.com` and `gmx.com`, `gmx.de`, `gmx.it`, etc. Test your code in the Web browser.

**Exercise 4★** Reuse the description of the *Nationality* from `MAPersonModel` in the model of the address `MAAddressModel` by calling the appropriate description of the person and changing the label to *Country*. Make sure not to modify the original description by creating a copy. Test it in the Web browser.

**Question 5** Make a rough guess on how much code was saved by using Magritte compared to a manual approach in Seaside. From when on does it make sense to use a meta-model? What are the advantages? What are the disadvantages?

### 3 Integration into Seaside

Up to now we have been working with descriptions only, we didn't write any line of Seaside code. Moreover the model was not remembered somewhere and therefor was lost between different sessions. In this section we will concentrate on Seaside and build a simple user-interface that allows us to manage multiple persons. To get an idea of how the result could look like, see Figure 1.

## Person Manager

[Add](#)

1. [Lukas Renggli view](#)
2. [Stéphane Ducasse view](#)

Figure 1: Person Manager List

**Exercise 6** Start by creating a new sub-class of `WComponent` called `MAPersonManager`. Add a class-instance-variable called `Persons` that will serve us as a simple place to keep the model objects. Initialize it with an empty `OrderedCollection`. Register the newly created class as a new Seaside entry point. Create a method `#renderContentOn:` that displays the heading *Person Manager*.

Test the setup of your new application in the Web browser.

**Exercise 7** Create an action-method `#add`, that creates a new instance of `MAPersonModel`, calls the default Magritte editor and adds it to our collection. Note that Magritte will answer `nil`, if the user hits the cancel-button in the editor. Create a link in your page that will execute the `#add` method.

**Exercise 8** Create an accessor `#persons` for the class-variable `Persons` on the instance-side. Render a list of all persons within `MAPersonManager`.

**Exercise 9** Render the list entries as links, so that existing persons can be edited.

**Exercise 10** Render an additional link called *view*, as seen in Figure 1, to display the entry in a read-only view. You can turn a component into read-only mode by sending the message `#readonly:`.

## 4 Using Magritte Reports

Rendering a list like this is nice, but it doesn't scale well if you have many entries. As well it doesn't take advantage of the nice descriptive model we have. Luckily Magritte includes built in reporting facilities that we will be extending our application with. In this section we make our little application look like Figure 2.

### Person Manager

[Add](#) [Filter](#) [Clear](#)

[First Name](#) [Last Name](#) [E-Mail](#)

[Nationality](#)

Lukas Renggli renggli@iam.unibe.ch Switzerland [view](#) [edit](#) [remove](#)

Stéphane Ducasse ducasse@iam.unibe.ch France [view](#) [edit](#) [remove](#)

Figure 2: Person Manager Report

**Exercise 11** Add a new child component to `MAPersonManager` and initialize it, lazily or within the method `#initialize`, with `MAReport rows: self persons description: MAPersonModel description`. Render the report instead of the list.

**Exercise 12** Play with the report in your browser: sort the rows, add new persons... You will probably notice that the report doesn't update itself when adding new persons. It's because the report creates a copy. Call `#refresh` to do so.

**Exercise 13★** Right now almost all the information about a person is displayed within the report, if you enter much data this gives a very wide Web page. Filter out all the descriptions except the ones for `#firstName`, `#lastName`, `#email`, and `#nationality`.

**Exercise 14** Add a new column to the report to hold commands to *view*, *edit* and *remove* entries. To get a hint on how to archive the desired behaviour, brows the references of `MACCommandColumn`.

**Exercise 15★** Implement a search functionality within your report. The user should be asked for a filter string that will be compared to all the string-fields within the persons. To get some pointers have a look at the implementors and senders of `#rowFilter:`, `#toString:` and `#readUsing:`, and `#matches:`. Add a *clear* link as well, to remove any filter.

## 5 Described Descriptions★

As experience shows, customers are often not completely satisfied with the features the application developers are giving to them. They want more and they want to be able to customize everything by themselves. Thanks to the reflective nature of Magritte – all the descriptions within Magritte are described with Magritte descriptions itself – we can easily provide the necessary functionality.

**Question 16** Point your browser to <http://localhost:8080/seaside/example-browser> and select `MADescriptionEditor` in the first drop-down box. What is this? What could it be useful for?

**Question 17** Browse the source code of `MADescriptionEditor`. As you can see for the preview and instance of `MAAdaptiveModel` is created. This is a class that has only two instance-variables: one references a description-container, the other a dictionary mapping description-elements to actual values. Have a look at the methods `#readUsing:` and `#write:using:`, and their default implementations in `Object`. Why are those methods overridden in `MAAdaptiveModel`?

**Exercise 18** Add a class-instance-variable to `MAPersonModel` called `CustomDescription`. Initialize it with an empty instance of `MAContainer`. Create an accessor method. Override `#description` on the instance-side, call `super` and compose it with the `CustomDescription`. So far, you should see no difference in the behaviour of the application when testing it.

**Exercise 19** Add another link above your report to let the user edit the custom description. Unfortunately the default implementation of `MADescriptionEditor` has no way to go back to the caller. The simplest solution is to subclass `MADescriptionEditor` and render an additional button by overriding `#renderButtonsOn:` that answers the modified description.

**Exercise 20** Now you are able to add additional fields to your `MAPersonModel` class, unfortunately the editor doesn't work anymore. If you try to add or edit

a person you get the error-message: *This message is not appropriate for this object.* Obviously the custom-descriptions that have been added through the web do not have an accessor and you probably don't want to let your customers care about those low-level things anyway. To solve the problem you need to override the methods `#readUsing:` and `#write:using:` similar to the way it is done in `MAAdaptiveModel`.

**Question 21** Test your application toughly. How does it behave if you add, remove or change fields within living instances of `MAPersonModel`?

### Edit Person

**Title:**

**First Name:**

**Last Name:**

**E-Mail:**

**Home Address:**

**Office Address:**

**Nationality:**

**Picture:**  no file selected

**Birthday:**

**Age:** 26  
[Kind](#) [Number](#)

**Phone Numbers:** The report is empty.

**Comment:**

**Custom Field 1:**

**Custom Field 2:**  Custom Field 2

Figure 3: Person Editor Customized