**Sopra Steria**

# Hands On MDK iOS

Version 1.00 of Monday, February 22nd 2016

# Destinataire(s)

# Historique

| Version | Date | Reason | Edited by | Approved by |
|---|---|---|---|---|
| 1.00 | 17/02/2016 | Refonte HandsOn iOS | Quentin Lagarde | Sébastien Maitre |
| 1.10 | 2016/02/23 | English translation | Julien Lacroix | |
| | | | | |

# Table of contents



# 1. Introduction

During this "Hands-On MDK" training session, you will learn to use the "Mobile Development Kit" in order to generate, overload end customize an application for the *iOS* platform.

The app that will serve as example throughout this session will be named "My Expenses". As its name suggests, it will allow managing expense reports and associated expenses.

## 1.1. Overview of the application

*My Expenses* consists of four screens:

- **The main screen**, allowing navigating to the three other screens using navigation buttons.
- **The "my expenses" screen** that gathers:
  - A list displaying expense reports per customer, as shown left of Figure 1.
  - A partially editable detailed display of an expense report and its expenses, illustrated in the middle of Figure 1.
  - The completely editable detail of an expense, right of Figure 1.
- **The "about" screen** that contains a Web view presenting the app.
- **The "my travel books" screen.** This one is empty for the moment.

## 1.2. Roadmap

As said earlier, the app development will be broken down in three parts:

1. Designing and generating the application.
2. Overriding the business model and customizing the display.
3. Achieving client/server synchronization.

Only the first step is mandatory. The other two can be achieved in any order.

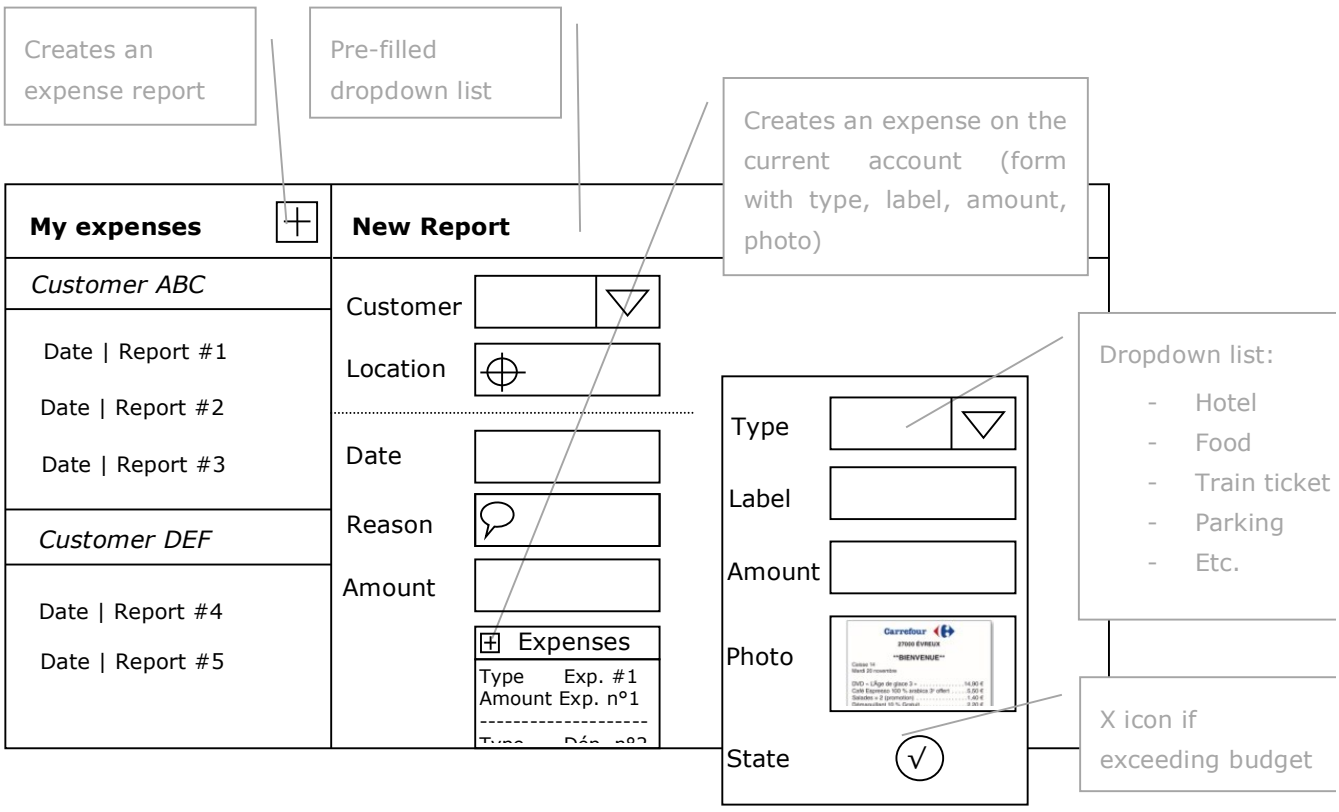


*Figure 1 : the « My expenses » screen*

## 1.3. Prerequisites

To follow this session, you must:

- Install the Command Line Interface of MDK:
  - Install Node.js
  - Install mdk-cli:

```
$ npm install -g mdk-cli
```

- Install all the tools needed to develop an iOS app with the MDK:

```
$ mdk tools-install ios
```

- Install Xcode (Last version, non-bêta)
- Install a UML modelling tool:
  - **MagicDraw** UML v17.0.5
    http://www.nomagic.com/products/magicdraw.html

# 2. Initializing the project

Every MDK project uses a common source tree. Its creation is the first step.

In you command prompt:

1. Navigate to a folder that will contain the project root
2. Initialize the project with the following command:

```
$ mdk create -t "My Expenses Tuto" com.soprasteria.mdk.handson.myexpenses
```

   a. *com.soprasteria.mdk.handson.myexpenses* is the unique id of the application
   b. The *–t* option is facultative. It allows to use a template to generate the app. In that case, it will apply a visual theme and add a folder containing the overloading files.
   c. When it is asked, enter a login and a password.

2. Add the iOS platform.

```
$ cd myexpenses

$ mdk platform-add ios
```

   a. The « ***myexpenses*** » folder has been created. Its content must be as shown in Figure 2.

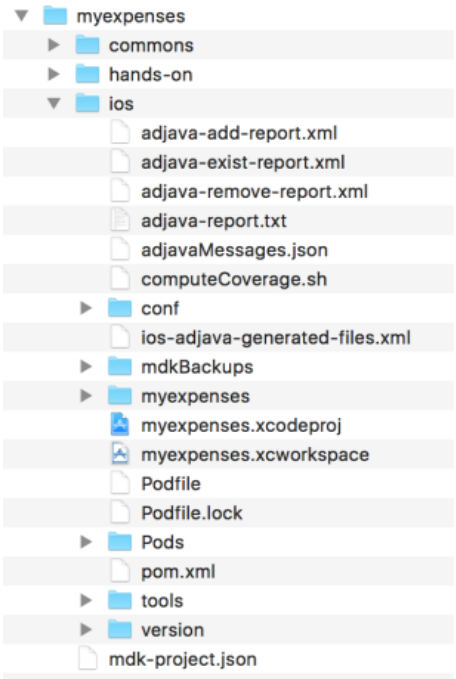


*Figure 2 : « myexpenses » source tree.*

# 3. Generating the application from a UML model

Once the project is initialized, it is necessary to design the model of the application using UML class diagrams. Then we can proceed to generating the app.

## 3.1. Generating the provided model

1. Overwrite *myexpenses/commons/modelisation.xml* with the file included in *myexpenses/hands-on/ios/31 - Generation.*
2. Generate and build the app.

```
$ mdk platform-build ios
```

   a. The generated app will be found in *Debug-iphonesimulator/myexpenses.app*
   b. It can be installed on a terminal/emulator

```
$ xcrun simctl install booted myexpenses/ios/build/Build/Products/Debug-iphonesimulator/myexpenses.app
```

3. Tap on *MY EXPENSES*. The following screen should be blank. We will see in section **Error! Reference source not found.** how to add data to the generated app.

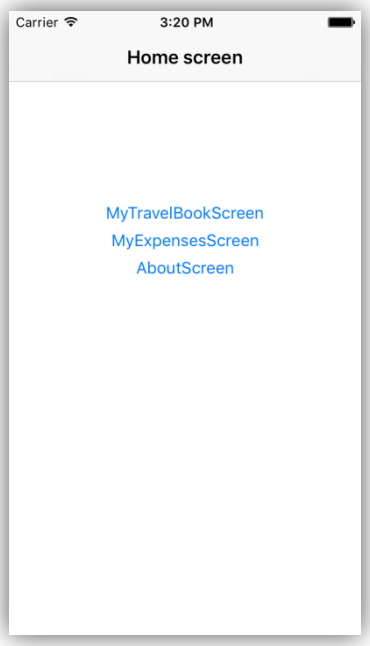


*Figure 3 : Main screen of the generated app.*

4. Tap on *MY EXPENSES*. The following screen should be blank. We will see in section 3.3 how to add data to the generated app.

## 3.2. UML model

Open the app's UML model (*myexpenses/commons/modelisation.xml*) inside *MagicDraw*

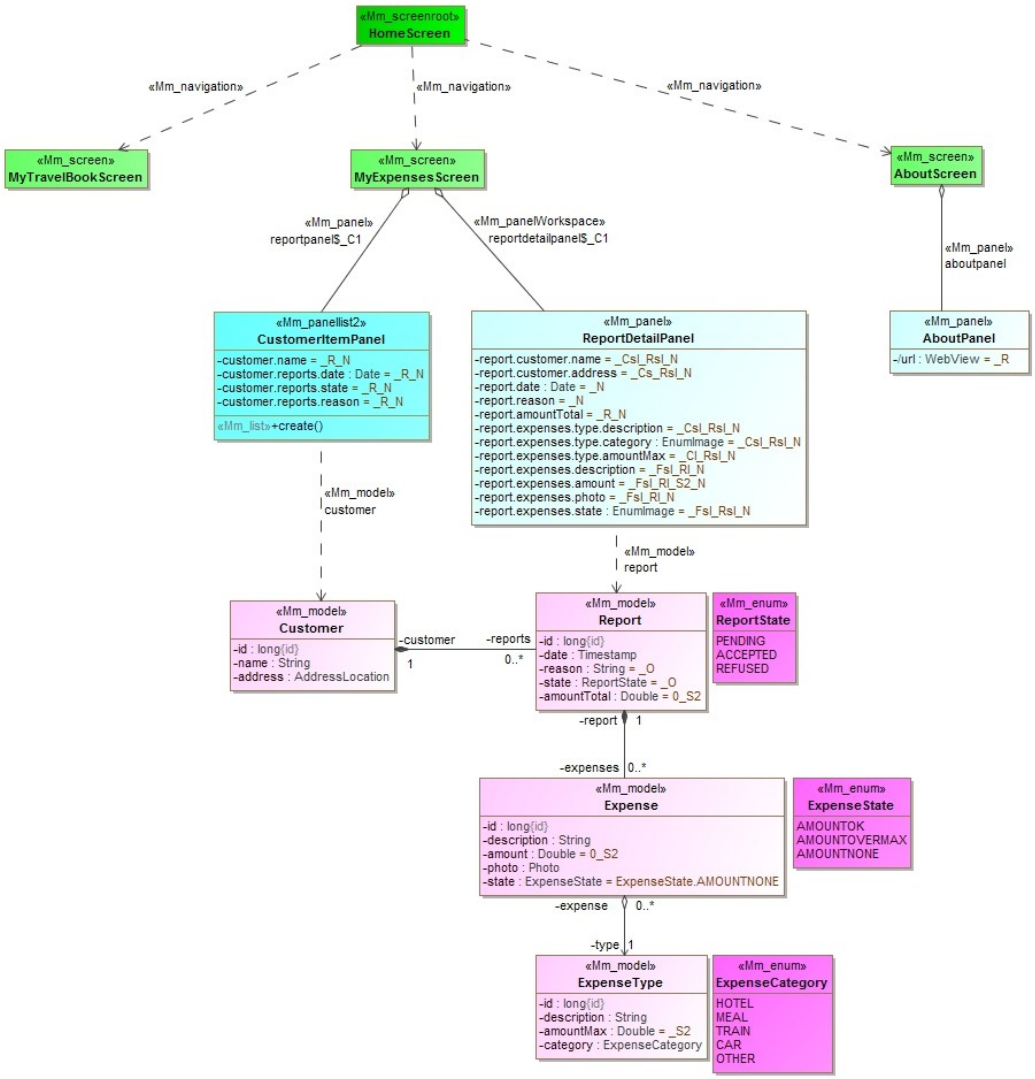


*Figure 4 : Complete UML model.*

The MDK modelling builds atop a classic UML modelling with a few additional mechanisms:

- The usage of UML stereotypes on classes and navigations.
- The usage of MDK types for attributes.
- Default values with the MDK syntax, to configure the attributes.

The application model is layered, from bottom to top:

1. The business model layer: containing the "**Mm_model**" stereotyped classes, as shown in Figure 5. It consists of:
    a. A ***Customer*** that brings ***Report***s.
    b. Each report breaks down in **Expense**s.
    c. Each expense has an ***ExpenseType***
    d. Some classes can have attributes with enumerated types:
        - ***NoteState, ExpenseState*** and ***ExpenseCategory*** are enumerations, flagged with « ***Mm_enum*** ».

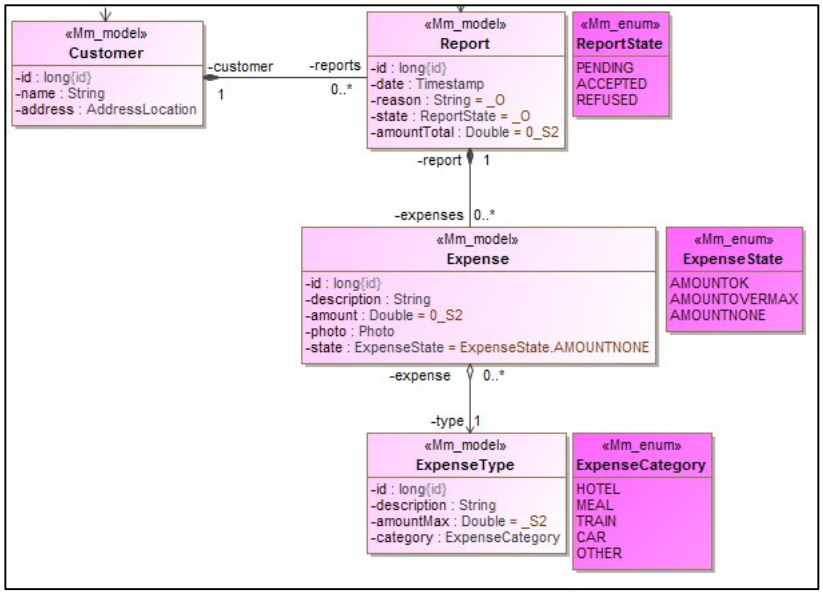


*Figure 5: « Mm_model » stereotyped classes.*

2. The panel layer, describing screen fragments: containing the "***Mm_panel***[…]" stereotyped classes, as shown in Figure 6. Three classes have been modelled here:
    a. **CustomerItemPanel** presents attributes of the *Report* and *Customer* classes.
    b. **ReportDetailPanel** presents the detail of an expense report, and allows displaying and/or updating some attributes of *Customer*, *Report*, *Expense* and *ExpenseType*.
    c. **AboutPanel** presents a Web view accessible from the app.

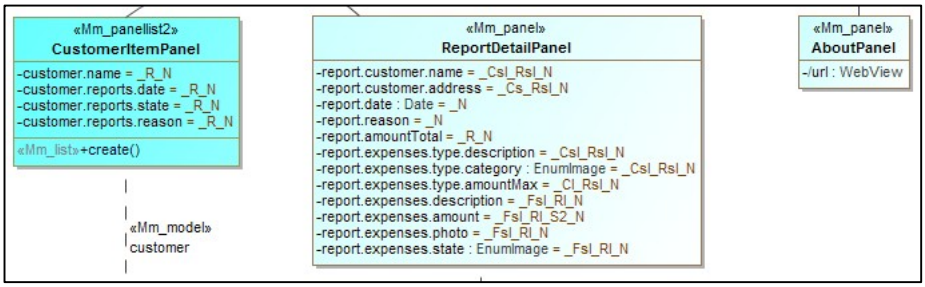


*Figure 6 : « Mm_panel[…] » stereotyped classes.*

3. The screen layer, describing screen contents and navigation: containing the "***Mm_screen***" stereotyped classes, along with one "***Mm_screenroot***", as shown in Figure 7.
   We find here all the "***Mm_screen***" stereotyped classes, each corresponding to one screen (and containing the "***Mm_panel***" above):
   a. **MyTravelBookScreen** is an empty screen, containing no panel.
   b. **MyExpensesScreen** has a list panel, *CustomerItemPanel*. Choosing a list item will bring us to the second panel, *ReportDetailPanel*
   c. **AboutPanel** is composed of a panel that displays a Web view.

   At last, we find here the "**Mm_screenroot**" classcorresponding to the main screen shown at the application startup. This screen is used to navigate to the other screens.

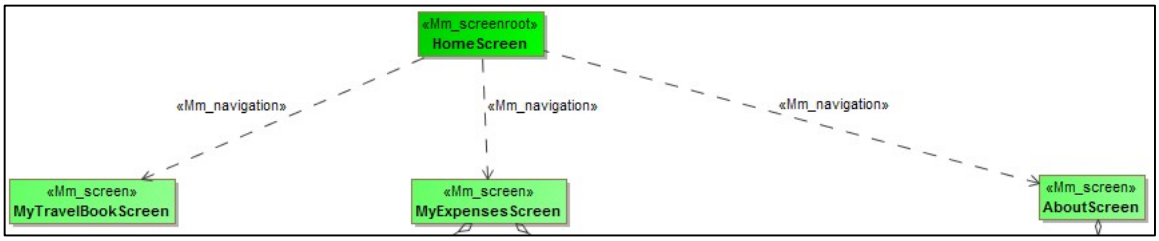


*Figure 7 : « Mm_screen » and « Mm_screenroot » stereotyped classes.*

## 3.3. Dataset

Using a dataset is a facultative step in the process of creating an application. It can be useful during the first development iterations if no web service can be consumed to populate the application data.

The MDK provides a tool that can initialize a dataset from an Excel worksheet

With a few steps, it can build a file containing, for each business entity in the UML model, data that will be injected at first launch of the application.

It can also import and export data in the CSV format.

For this session an Excel worksheet has already been filled:

1. Open ***myexpenses/hands-on/ios/****33 – Datas*
2. Copy all CSV files in this folder *myexpenses/ios/myexpenses/resources/csvdatas/*
3. Double click on *myexpenses.xcworkspace to open your Xcode project and navigate in source-tree at*: *myexpenses/myexpenses/resources/csvdatas/*
4. You must add csv files in Xcode: Right click on folder « csvdatas » the « Add files to "myexpenses" » et add new csv files.

5. Remove your app on your simulator or do this action « *Simulator > Reset Contents and Settings…* »
6. Build and start your app on simulator
7. Tap on *MyExpensesScreen button*

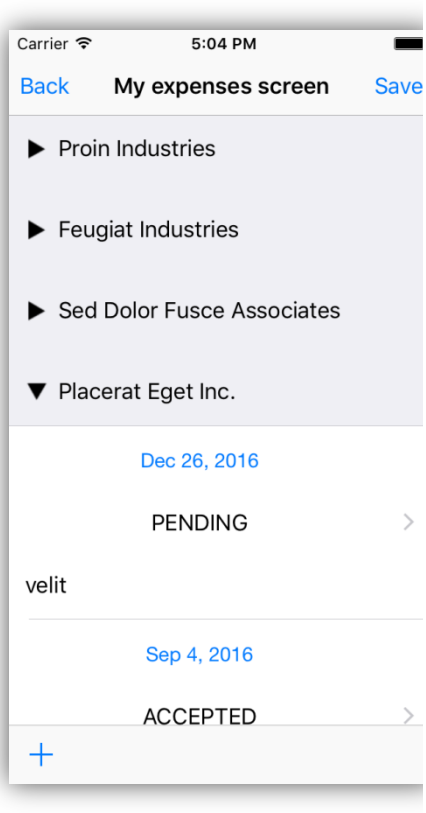


*Figure 8 : Panel presenting the list of reports.*

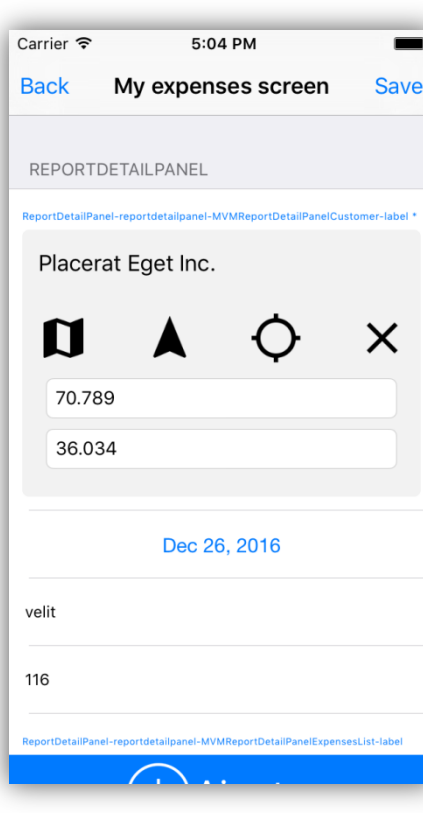


*Figure 9 : Panel presenting the detail of a report.*

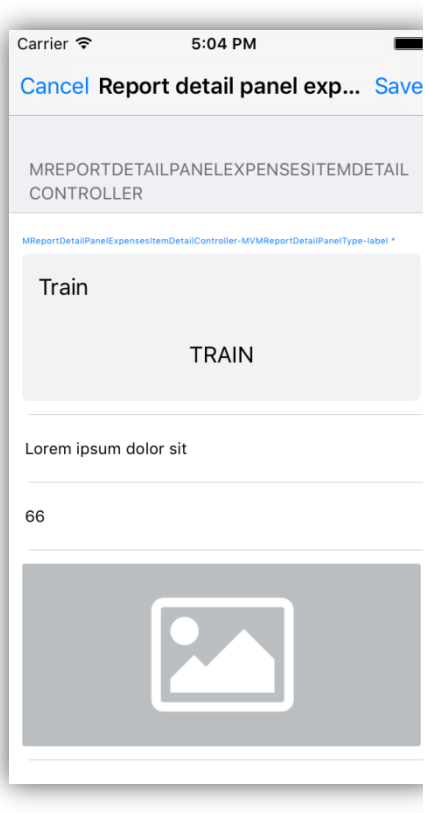


*Figure 10 : Panel allowing the modification of an expense.*

8. Tap on a customer on the list. The customer report display in below
9. Tap on one report to display its details. Figure 9.
10. Tap on expense of report in bottom of screen. The screen of modification of an expense Figure 10.

# 4. Customizing

Once the app is generated, it is possible to customize some elements to finalize the project.

## 4.1. Customizing labels

The generated project contains a file named *dev__project_labels.xml* inside the directory *myexpenses/ios/myexpenses/resources/strings/XX.lproj*. The content of this file comes from the UML model:

- One label per screen
- One label per button
- One label for each attribute that doesn't have the default value option "_N" (No label)
- One label for each element of an enumeration
- …

Furthermore, it is possible to redefine these labels in multiple languages, in order to internationalize the application.

1. Copy the content of *myexpenses/hands-on/ios/41 – Labels/* into *myexpenses/ios/*
2. Build the app

```
$ mdk platform-compile ios
```

3. Deploy and execute the app



## 4.2. Removing a label

The goal here is to demonstrate that modifying the UML model does not necessary cause a total loss of the custom labels.

Overwrite *myexpenses/commons/modelisation.xml* using the file in *myexpenses/hands-on/ios/42 – Delete label.*

1. Overwrite « *myexpenses/commons/modelisation.xml* » using the file in this directory « *myexpenses/hands-on/ios/42-Delete Label* »

This model differs from the previous one by adding the option *_N* on the *url* field of *AboutPanel*

2. Generate and build

```
$ mdk platform-gensrc ios
```

3. Build and start your application « Cmd+R »

4. The label of the unique field in the *About* screen is gone

# 5. Overriding

Once the application is generated, it is possible to overload it to add any feature. In this chapter we will see how to overload the business model of an application.

To achieve this, we will need to dive into the Objective-C. We recommend you import the project into Xcode.

## 5.1. Automatic calculation of the total amount

We want to automatically update the field *amountTotal* (shown in Figure 11) in the detail of a report, by calculating the sum of all *amount* fields of its expenses (Figure 11). This update must be done at every *expense adding, removal or modification*.

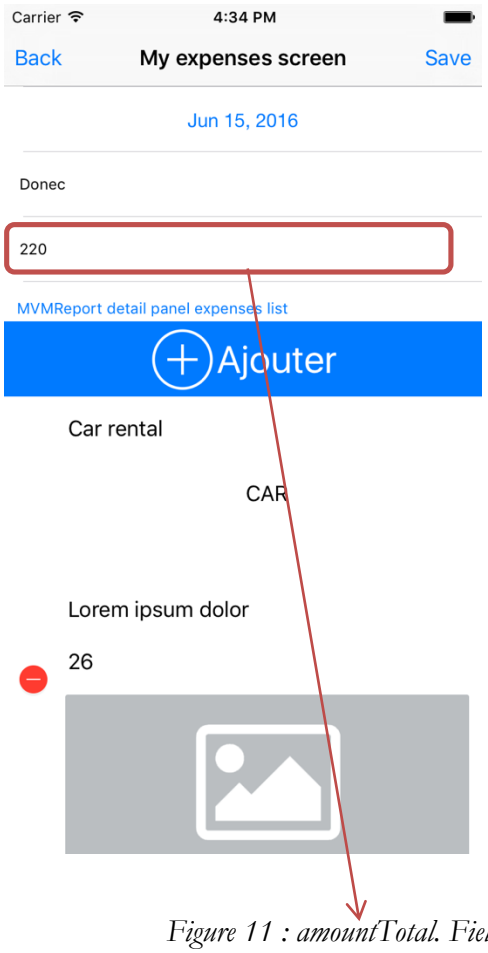


*Figure 11 : amountTotal. Field*

When you check the UML model of application, you can see the amount total about expense is displayed in the panel named « ReportDetailPanel ». The business rule must be implemented inside the *view model* of this panel: MVMReportDetailPanel.

1. Open *MVMReportDetailPanel.m* inside: « myexpenses/ios/viewmodel »
2. Look for this comment: *//@non-generated-start[other-methods][X]*
    a. Delete this tag [X]: In this way, the generator will not overwrite your code
    b. Between *//@non-generated-start* and *//@non-generated-end* you can add your code.
3. Copy this lines of code between both comment:

```
-(void) createViewModelConfiguration {
    MFViewModelConfiguration *config = [MFViewModelConfiguration
configurationForViewModel:self];

    MFListenerDescriptor *propertyChangedListener = [MFListenerDescriptor
listenerDescriptorForType:MFListenerEventTypeViewModelPropertyChanged withFormat:
                                                    @"computeTotalAmount :
mVMReportDetailPanelExpensesList",
                                                    nil];
    [config addListenerDescriptor:propertyChangedListener];
}
-(void)computeTotalAmount {
    float total = 0.0;
    for(MVMReportDetailPanelExpensesItemCell *itemVM in
self.mVMReportDetailPanelExpensesList.viewModels) {
        total += [itemVM.amount floatValue];
    }
    self.amountTotal = @(total);

}
```

    a. A view model configuration is created for this view model
    b. An observer is created to listen to each value changes of field « mVMReportDetailPanelExpensesList »
    c. The listener is added to the view model configuration
    d. This method « computeTotalAmount » retrieves a list of expenses to do the calculation of all amounts. Therefore, the total amount is updated.
    e. The bi-directional binding update in the main thread the total amount field.

4. Build your application either within Xcode or Command Line

```
$ mdk platform-compile ios
```

5. Deploy, execute and test

## 5.2. Automatic calculation of the state of an expense

We want to automatically update the state of an expense (shown in 12), for each modification of amount and type of expense, by comparing the amount of expense with maximum allowed for this type of expense.

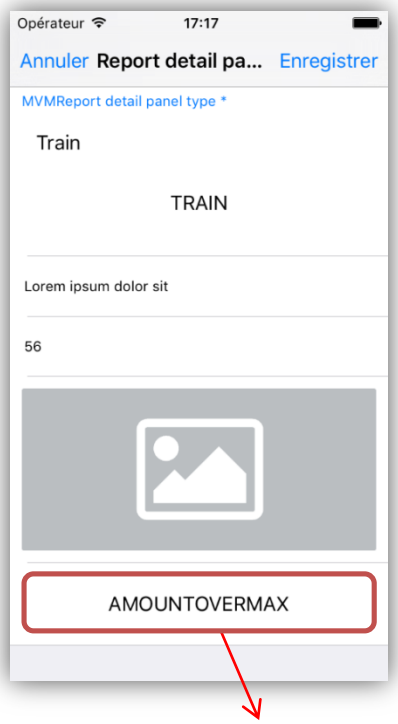


*Figure 12 : Champ state.*

When you check the UML of your application, you can see the amount of an expense are display in the panel named ReportDetailPanel.

With the previous override, we have seen ReportDetailPanel is affiliated to the *view model* MVMReportDetailPanel.

This *view model* represents data displayed in the panel. You can find a representation report in a list of sub view model (MVMReportDetailPanelExpensesList).

Each item of this list is type of MVMReportDetailPanelExpensesItemCell with Expenses is deducted of the end name of association between Report and Expense in the UML model. ItemCell show the type of view model affiliated: It's a view model of cell in the list of item.

The expense amount modification listener will be implemented through this *view model*:

1. Navigate to this directory **myexpenses/ios/viewmodel/** and open *MVMReportDetailPanelExpensesItemCell.m*
2. Look for the comment: *//@non-generated-start[other-methods][X]*
    a. Delete this tag [X]: In this way, the generator will not overwrite your code
    b. Between *//@non-generated-start* and *//@non-generated-end* you can add your code.
3. Copy this line of code:

```
-(void) createViewModelConfiguration {
 MFViewModelConfiguration *config = [MFViewModelConfiguration configurationForViewModel:self];

        MFListenerDescriptor *propertyChangedListener = [MFListenerDescriptor
listenerDescriptorForType:MFListenerEventTypeViewModelPropertyChanged withFormat:
                                    @"updateExpenseState : selectedMVMReportDetailPanelTypeItem,
amount" ,
                                    nil];
   [config addListenerDescriptor:propertyChangedListener];
}

-(void) updateExpenseState {
   if(self.selectedMVMReportDetailPanelTypeItem && self.selectedMVMReportDetailPanelTypeItem.amountMax
&& [self.selectedMVMReportDetailPanelTypeItem.amountMax intValue] >0) {
      if([self.amount intValue] > [self.selectedMVMReportDetailPanelTypeItem.amountMax intValue] ) {
         self.state = MEXPENSESTATE_AMOUNTOVERMAX;
  }
      else {
         self.state = MEXPENSESTATE_AMOUNTOK;
      }
   }
   else {
      self.state = MEXPENSESTATE_AMOUNTNONE;
   }
}
```

    a. This first method « createViewModelConfiguration » allow to create a configuration of view model: it's a MDK Movalys object allowing to configure this view model.

b. The configuration is set for this view model. Then, we add a listener (MFListernerDescriptor) to affiliated the call of this method « updateExpenseState » at all changes values for field « selectedMVMReportDetailPanelTypeItem » and « amount ».
c. You should implement this method « updateExpenseState », according to expense type and expense amount, it set its state (OK, OVERMAX or NONE).

4. Build your application either within Xcode or Command Line

```
$ mdk platform-compile ios
```

5. Deploy, execute and test

## 5.3. Hide the state of an expense based on its type

In this part, you must display the expense state only if the maximum amount is defining on the expense type.

1. Open *MVMReportDetailPanelExpensesItemCell*.m in the directory ***myexpenses/ios/viewmodel/***
2. Look for this comment *//@non-generated-start[methods]*
3. Copy this line of code following code already added:

```
-(void) updateExpenseStateVisibility {
   BOOL hideState = !self.selectedMVMReportDetailPanelTypeItem ||
!self.selectedMVMReportDetailPanelTypeItem.amountMax;
   [[NSNotificationCenter defaultCenter]
postNotificationName:@"MReportDetailPanelExpensesItemDetailController_Notification" object:self
userInfo:@{@"hideState":@(hideState)} identifier:@"expenseStateVisibility"];
}
```

   a. This method returns a boolean according to the maximum amount state.
   b. This boolean is sent to the controller by notification.

4. Add a listener on the field "selectedMVMReportDetailPanelTypeItem" to call this method for each change of expense type.

```
@"updateExpenseStateVisibility: selectedMVMReportDetailPanelTypeItem"
```

5. You must handle this notification in the *view controller* who manages details (MReportDetailPanelExpensesItemDetailController), you can find it in this directory **myexpenses/ios/controller**
   a. Look for this comment *@//non-generate-start[other-methods][X]*
   b. Delete this tag [X]: In this way, the generator will not overwrite your code

c. Between *//@non-generated-start* and *//@non-generated-end* you can add your code.

d. Add these lines of code:

```
-(void) didReceiveClassNotification:(NSNotification *)classNotification {
    //Check notification identifier
    if([classNotification.identifier isEqualToString:@"expenseStateVisibility"]) {
        //Retrieve descriptor to hide
        MFBindingAbstractDescriptor *descriptor = [self bindingDescriptorAtIndexPath:[NSIndexPath
indexPathForRow:4 inSection:0]];
        descriptor.hidden = [classNotification.userInfo[@"hideState"] boolValue];
        //Reload data
        [self reloadData];
    }
}
```

e. This method must be implemented, because notifications are sent to « MReportDetailPanelExpensesItemDetailController » with the name « *MReportDetailPanelExpensesItemDetailController*_Notification »

i. The notification id is tested.

i. The descriptor position is retrieved

ii. This property *hidden* takes the value sent by the notification.

b. Build your application either within Xcode or Command Line

```
$ mdk platform-compile ios
```

6. Deploy, execute and test

## 5.4. Amount formatting: XX,XX€

Throughout this Hands-on, the currency of all amounts in the application is €. The goal of this overriding is to display all amounts in read-only mode, while respecting the locale of the device/simulator. The formatting will be two decimal followed by €.

This formatting must be implemented for all fields of expense details panel and also in the expense entry panel, according to **Figure 13**, **Figure 14** et **Figure 15**.

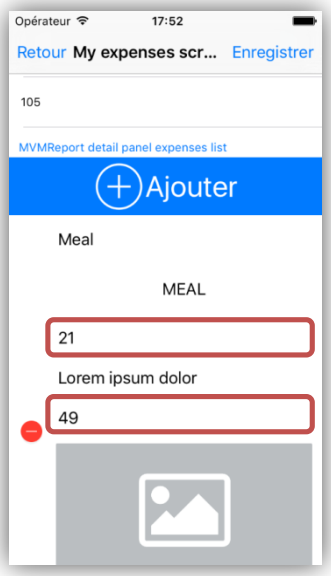


*Figure 13 : Field to format in the report detail panel*

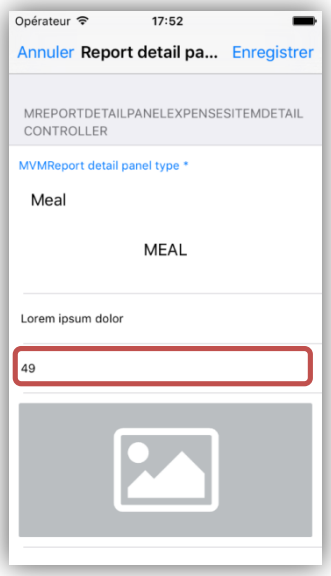


*Figure 14 : Field to format in the expense entry panel*

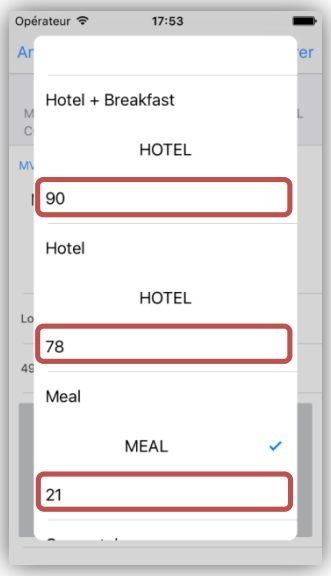


*Figure 15 : Field to format in the expense type dropdown list*

1. Copy the directory content ***myexpenses/hands-on/ios/55 – Amount/viewmodel*** **dans le répertoire** ***myexpenses/ios/myexpenses/src/viewmodel/***
2. Open implementation files (.m) from the 3 following ViewModel:
   a. *MVMReportDetailPanel* (ViewModel of expense)
   b. *MVMReportDetailPanelExpensesItemCell* (ViewModel of a report of an expense)
   c. *MVMReportDetailPanelTypeItem* (ViewModel représentant un type de dépense)
3. For each class:
   a. Retrieve this comment *//@non-generated-start[custom-imports][X]*
   b. Removing the *[X]*
   c. Delete this tag [X]: In this way, the generator will not overwrite your code
   d. Add this line of code before *//@non-generated-end :*

```
#import "MFUIBaseViewModel+AmountHelper.h"
```

   e. Look for this comment *//@non-generated-start[other-methods]*
   f. Add this line of code:

```
-(NSString *) humanReadableAmount {
    return [self humanReadableAmountFromNumber:self.amount];
}
-(void)setHumanReadableAmount:(NSString *)humanReadableAmount {
    self.amount = [self numberFromHumanReadableAmount:humanReadableAmount];
}
```

   g. According to ViewModel, « *self.amount* » will must be replace by « *self.amountMax* » or « *self.amountTotal*
4. In this way, the system accepts « € », components must be transformed in MDKTextField. To do that, replace the following files:
   a. ***myexpenses/ios/myexpenses/resources/storyboard/MyExpensesScreen.storyboard*** by ***myexpenses/hands-on/ios/55 – Amount/MyExpensesScreen.storyboard***
   b. ***myexpenses/ios/myexpenses/resources/xib/ReportDetailPanelExpensesItemCell.xib*** *by* ***myexpenses/hands-on/ios/55 – Amount/ReportDetailPanelExpensesItemCell.xib***
   c. ***myexpenses/ios/myexpenses/resources/xib/ReportDetailPanelTypeListItem.xib*** *by* ***myexpenses/hands-on/ios/55 – Amount/ ReportDetailPanelTypeListItem.xib***

5. Open the implementation of the 4 following classes:
   *a.* *MMyExpensesScreenDetailColumn1Controller* (*view controller* representing the report expense detail column)
   *b.* *ReportDetailPanelExpensesView* (*View Delegate* representing an expense in the list of report)
   *c.* *MReportDetailPanelExpensesItemDetailController* (*view controller* representing the details of an expense)
   *d.* *ExpensesTypeListItemBindingDelegate* (*view delegate* representing a type of an expense)
6. Each of these classes initializes a binding in the method "createBindingStructure". A binding is initialized when you create a configuraton on his inherited object. In this object we added descriptors; each of this descriptors contains information for graphic component property of view model associated. For this overriding, you must modify the *view model* property.

a. Retrieve this comment *//@non-generated-start[custom-imports][X]]*, remove this tag *[X] In this way, the generator will not overwrite your code*
b. For each binding structure, find: **c.data<->vm[.xxxx].amount** where [.xxxx] represents a step in the property path of the *view model*.
c. Replace the last part (amount, amountMax or amountTotal) with our new property: *humanReadableAmount*
d. You must obtain this result: **c.data<->vm[.xxxx].humanReadableAmount**

7. Build your application either within Xcode or Command Line

```
$ mdk platform-compile ios
```

8. Deploy, execute and test

## 5.5. Content of screen « About »

In this part, you must display an html page embedded in the screen panel "About" (Figure 17)

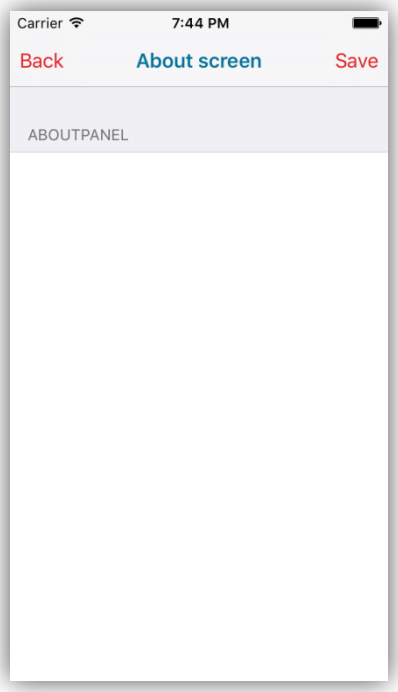


*Figure 16 : AboutPanel without Web.*


*Figure 17 : AboutPanel with Web « About ».*

1. Copy
   a. Content in the directory ***myexpenses/hands-on/ios/56 – About***
   b. In ***myexpenses/ios/html/***
   c. Add the directory « html » to your Xcode project in your new folder « Folder references » (Folder with blue color)
2. In the view model associated to about screen, you must override this attribute value:
   a. Look for this comment *//@non-generated-start[other-methods][X]*
   b. Remove this tag *[X] In this way, the generator will not overwrite your code*
   c. Before end tag *//@non-generated-end*, Add this line of code:

```
-(NSString *) url {
    return [[NSBundle mainBundle] URLForResource:@"about" withExtension:@"html" subdirectory:@"html"];
}
```

3. Build your application either within Xcode or Command Line

```
$ mdk platform-compile ios
```

4. Deploy, execute and test

# 6. Theming

Once the application is generated, it is possible to customize its visual aspect entirely.

This short part helps you to copy resources allowing to reach the expected design

1. Execute the script for design.

```
$ sh "hands-on/ios/62 - Design/applyDesign.sh"
```

2. At the end, open the new workspace and build, start your application on Xcode to check changes.

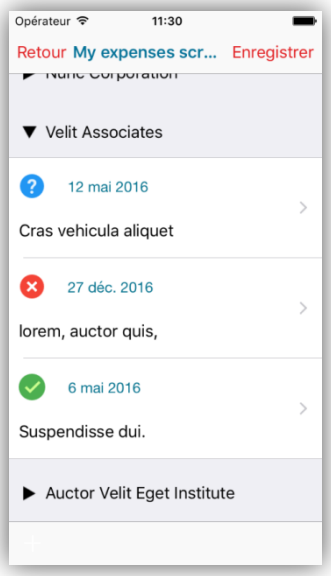


*Figure 18*


*Figure 19*

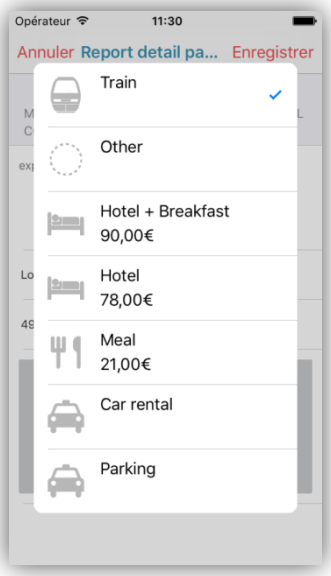


*Figure 20*