BACKBASE

Lab exercises

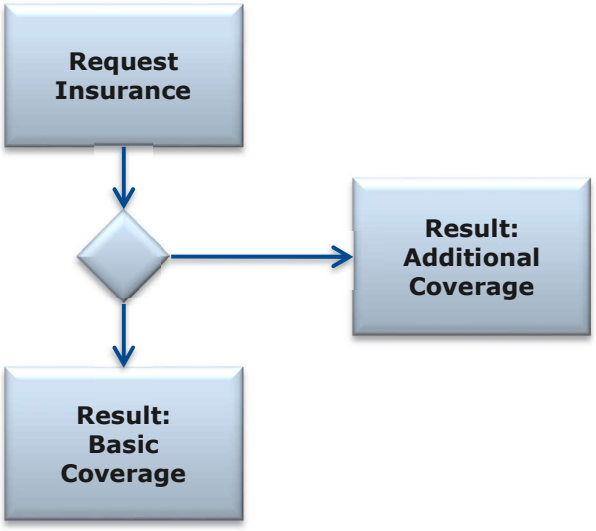
Training: BMF
Version: 1.9.2
Date: 20-2-2017

CONTENTS

INTRODUCTION

During this workshop we will work on a project to create a new application for the imaginary health insurance company 'Care for Less'. With this application a user can apply for health insurance online.

The following figure shows the steps a user can complete using this application.



On the first page the user fills out personal, lifestyle and coverage details. Then the premium is calculated and shown to the user.

Depending on whether the customer applied for just basic coverage or an additional coverage, the appropriate response page is shown.

In this exercise, we will start by creating elements in the *domain model*. Next, we will create some basic *user dialogs* and set up the *navigation*, i.e. the order in which pages are presented (as drawn above).

Then we will define *logic* to calculate premiums and determine whether the user can apply for additional health coverage or not.

EXERCISE 1 – DOMAIN MODEL

The first step to our health insurance application is to create a domain model, which includes customer information and possible premium elements.

The following high-level **requirements** are established for the application:

1	In order to get a health insurance policy with a minimum of fuss as a customer I want to be able to request a policy online.
2	In order to be able to get the best coverage as a customer I want to be able to choose additional health and/or dental coverage.
3	In order to decide whether I want the policy as a customer I want to know the premium beforehand.
4	In order to get a lower premium as a customer I want to be able to select a higher excess amount.
5	In order to avoid confusion as a service desk employee I want to prevent minors (under 18) from applying for insurance.
6	In order to feel I'm treated like an individual as a customer I want to be addressed by my name and gender in future communication.
7	In order to comply with regulations as a product manager I want to make sure the excess amount is at least € 170.
8	In order to correct mistakes as a customer I want to be able to go back to a previous page.
9	In order to keep the insurance company from going bankrupt as a risk manager I want customers with a very unhealthy lifestyle to not be able to apply for additional health coverage.
10	In order to minimize hassle as a customer I want to insure additional people on a single policy.
11	In order to get the right premium as a customer I want to enter data for all the people I want to insure.
12	In order to get the quote quickly as a customer I want to download it as a PDF.
13	The end date of the additional coverage can not exceed the end date of the basic coverage.

1A: SHARPEN YOUR PENCIL

Review the requirements.

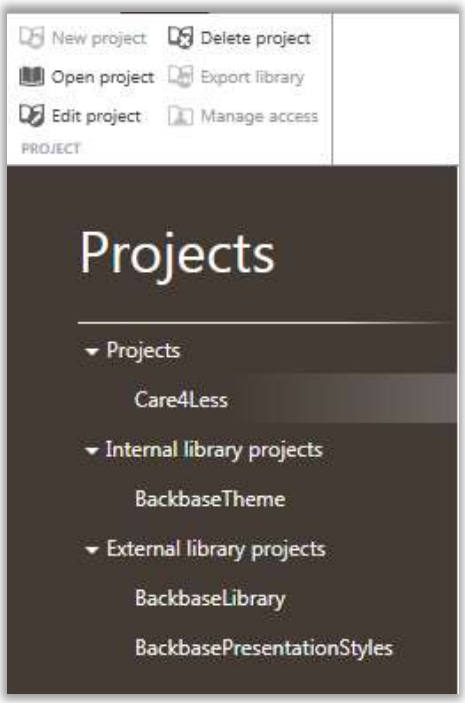
1. Which *entities* can we derive?

2. Which *attributes* can you identify and what are their types (boolean, string, number, date, date time, percentage, currency, integer)?

3. Which *validation rules* do we need?

4. Which requirements should come with additional documentation?

1B: CREATING THE MODEL



Opening the Studio project

For this assignment you log on to Studio

- Username s[x]
- Password s[x]

Choose 'Branch selection'.

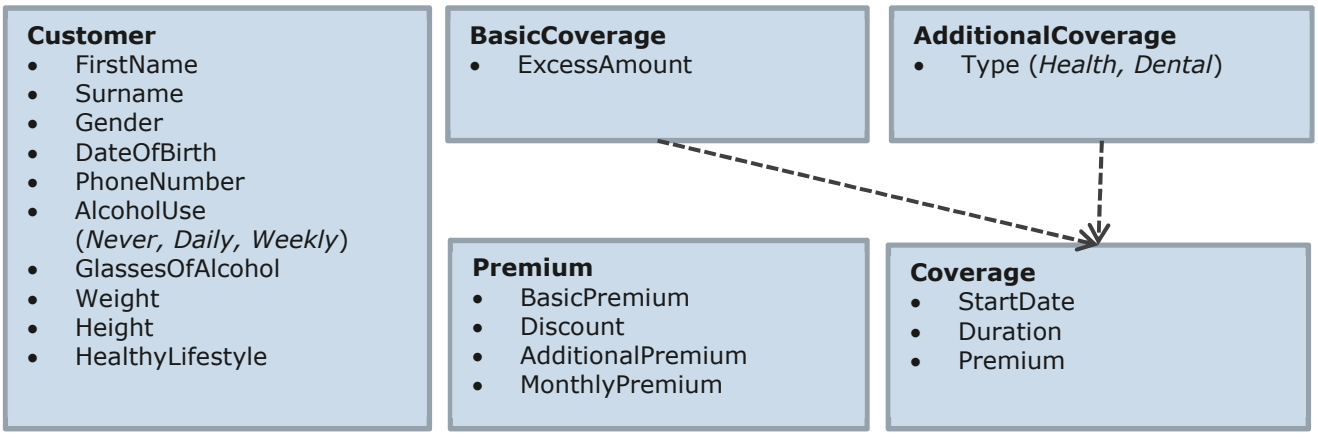
Then you choose the repository 'LaughAtLife' and select branch 'Student[x]'.

Select the project 'Care4Less' and click 'Open project'.

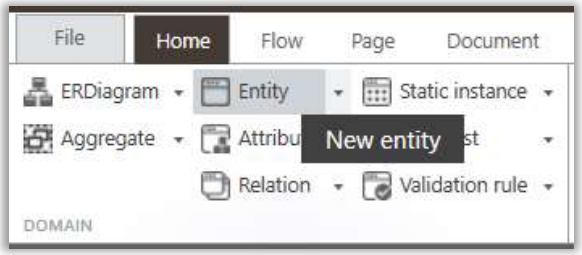
You can now start modeling!

The domain model

Now that we've established all the information that we need in our domain, we can start filling it in Studio. The figure below shows the entities and attributes needed. Use this as a guide to create the domain model.

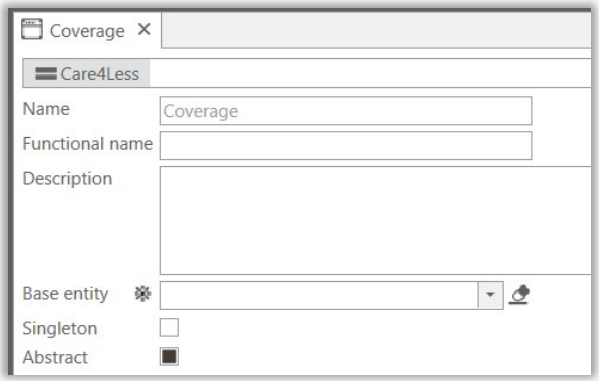


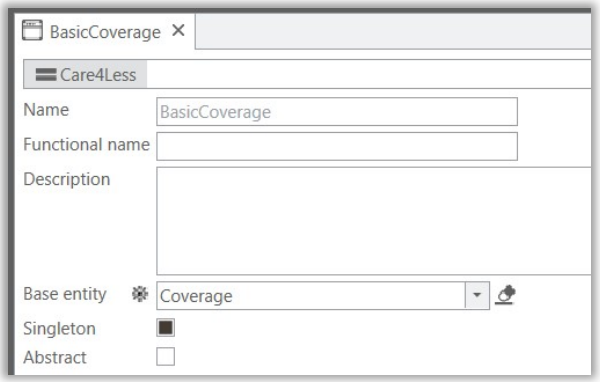
1. First create the entities.



For now, all *entities* are *singleton*, except for the entity Coverage. This means there can be more than one coverage, namely an additional and a basic coverage.

The entity Coverage should be abstract.





Make sure the entities AdditionalCoverage and BasicCoverage both use Coverage as a *base entity* so they can use the attributes of Coverage.

2. Now create the attributes. What should be the base type of each attribute?

   Don't forget to fill out the Question text, this is the text shown to the user in the Runtime.

3. For attributes that have a predefined set of possible values (like a multiple-choice answer), you can use a *value list*.

   In order to use a *value list*, you must create it and link it to the attribute. Give your attribute the right type before you create a valuelist (of the same type, e.g. *String*)

4. A multivalued attribute can represent more than one value. Which attribute(s) should be multivalued?

EXERCISE 2 – FLOWS AND PAGES

Now that we've created the *domain model* that we need for our *application*, we'll set up the user dialogs and the navigation. To do so, we will use *pages*, *flows*, *buttons* and *events*.

2A: SHARPEN YOUR PENCIL

Review the requirements.

1. Which flow(s) can you identify?

   _____

   _____

   _____

   _____

   _____

2. When should you make a flow exposed?

   _____

   _____

   _____

   _____

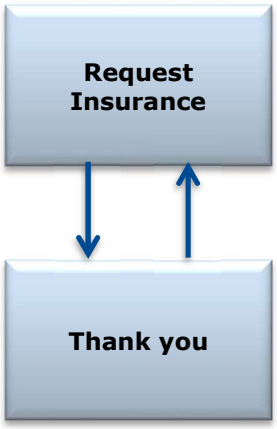
   _____

3. Which of the following statements is/are true?

   - ☐ It is possible to loop in a flow.
   - ☐ With a 'cancel' event type on a button, all changes on this page will be undone.
   - ☐ With a 'continue' event on a button, all changes will be validated.
   - ☐ A flow can have multiple exit nodes with the same event.

2B: CREATING PAGES AND FLOWS

For this assignment, we will create the following flow.



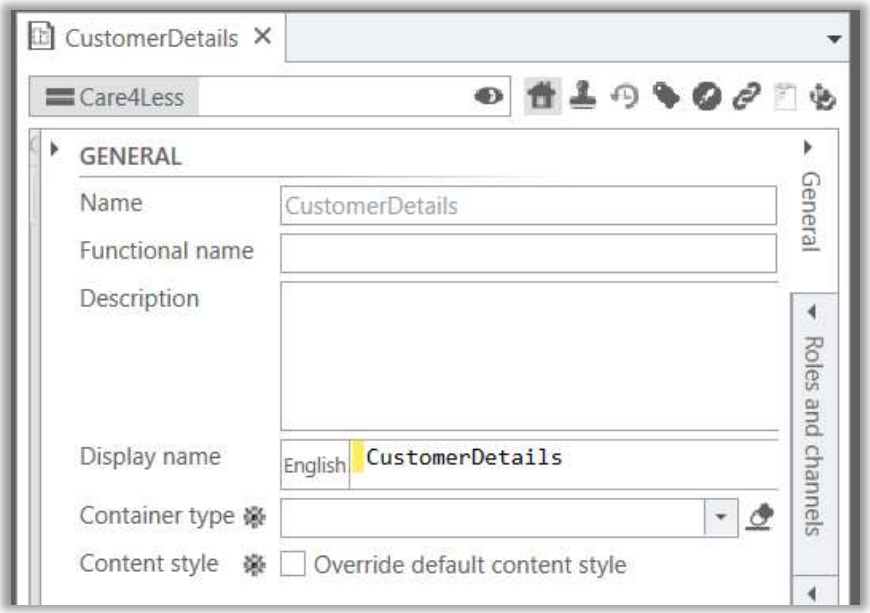
The first page allows the customer to fill in personal, lifestyle and coverage details. This page will also display the premium.

Finally, we will display a result page that contains a thank-you note. The customer should be able to navigate back and forth between pages as the arrows indicate.

Creating your first page

Let's create a page that contains all the customer and coverage details.

1. In order to create a page, first Create a Container and choose a name of your liking (e.g. *CustomerDetails*). A container is used to group attributes that belong together. We now have an empty container.



   Don't forget to fill out the Display name, this is the text shown to the user in the Runtime.

2. Fold the general menu by clicking on the arrow in the top left corner or Close button, so that we can start adding *attributes* to it.

3. This container will hold all customer detail attributes. You can add multiple attributes to one container, by dragging and dropping attributes from the *attribute list* onto the container.

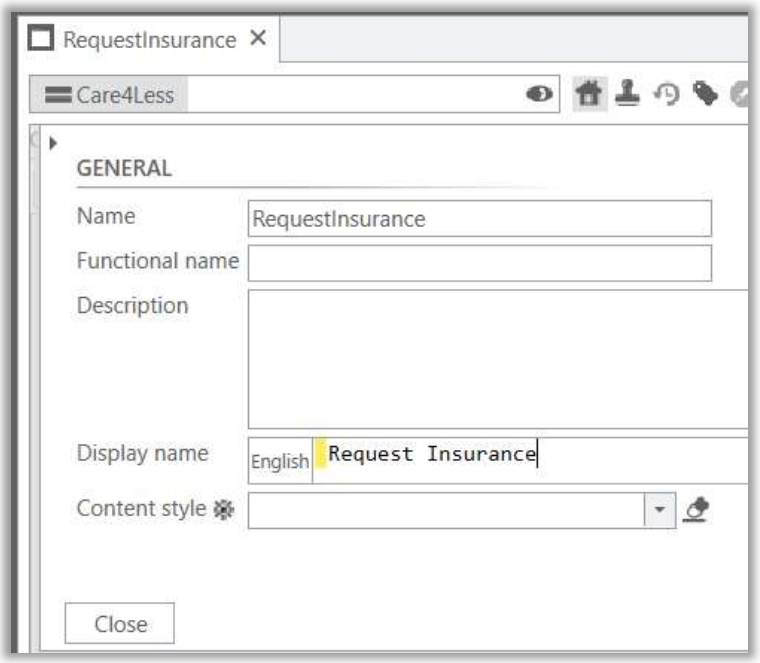
Hint: use the icon bar on the left of the Studio screen to open a list of attributes, while the page itself is open on the main part of the screen.

4. Now create a Container for the lifestyle attributes.

5. And finally create containers for the basic coverage, additional coverage and premium attributes. Make sure these are in three different containers.

   You will now have five containers, let's put them on a Page.

6. Create the Page and choose a name of your liking (e.g. *RequestInsurance*). We now have an empty page.



7. Fold the general menu so that we can start adding the *containers* to it. First we will add the personal detail to the page.

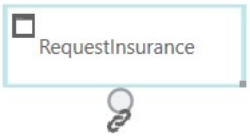
   Drag and drop the *containers* you just made from the *container list*, onto the page.

   You will now have a page with five containers on it.

Creating your first flow

Flows are used to model the order in which a user can visit *pages* (and what the application should do in between pages, as we will learn later).

1. Create a new flow, choose a name of your liking and check the box 'exposed'.



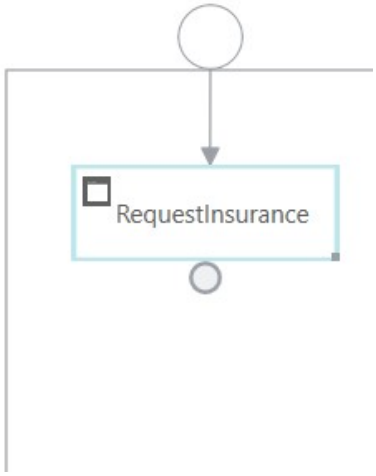
2. When the empty flow screen (*canvas*) is displayed, you can drag and drop your page from the list onto the flow.

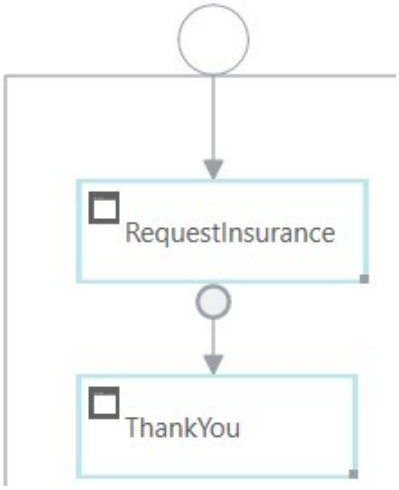
3. When your mouse pointer hovers above the *start node* (the empty circle), a chain icon appears. Drag and drop this chain on the page node. The nodes are now connected.

4. Read Appendix A to see how you can test your flow in the Runtime environment. Now test your application in the Runtime.

Navigating the flow

Our current flow only contains one page. In order to navigate to the second page we need to add a *button* and provide an *event* so that we can create a link to the second page.

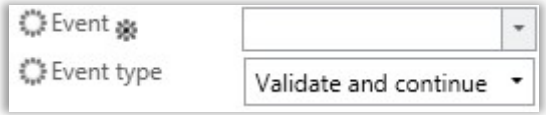




1. First create two *event*s, one to *go back* and one to *continue*. To create an event, only a name is needed; don't worry about the field 'Parameters'.

2. Create two *button*s as well, one to *go back* and one to *continue*. Don't forget the captions.

3. Open your first page and add the *continue* button using an inline container. You will see that there are several options available for a *button.* Set the correct event and event type. Which option should you select?



4. Save the page and open the *flow*. A dot should be visible on the edge of the page node. This dot corresponds to the event you just created. You can use this dot to connect the first page to a second page.

5. The second page should display a thank-you note. You can use a *text item* to display a piece of text.

Create the second page and a text item. Add the text item and a button to the page for the navigation. Set an event and event type to this button.

6. Now adjust the flow so that you can navigate between these two pages. That is, make sure you can go from page 1 to page 2 by clicking on the "continue" button; and from page 2 to page 1 by clicking on the "go back" button.

   Hint: hover over the dot to see which event *it represents.*

7. The customer's name, date of birth and phone number should be required fields. Adjust the page accordingly.

8. All premium attributes should be read-only fields. Adjust the page accordingly.

9. Test your application in the Runtime.

EXERCISE 3 – LOGIC

The application now displays pages and you can navigate between them. On the pages you can enter information, but this isn't used for anything yet. In this exercise, we will add *logic* to the application.

The logic elements necessary to calculate the premium can be derived from the following specifications. Remember requirements no. 3 and 4? They mentioned the premium, but didn't specify the calculation.

Also remember requirement no. 9; some people cannot apply for additional health coverage, but based on what criteria?

This is supporting documentation to go with those requirements.

The premium is calculated by using the following formula:

Monthly premium = (Basic premium – Discount) + Additional premium

The premium for the basic coverage for our health insurance is € 99,-.

The minimum excess amount is € 170. Customers who choose an excess amount of more than € 250 get 5% discount on the basic premium.

The premium for additional health coverage is € 10,- and for additional dental coverage € 12,-.

By law, Care for Less cannot refuse a customer for a standard health insurance. However, Care for Less can refuse people for additional health coverage, based on their lifestyle. The policy is that if a customer is obese or alcoholic, their lifestyle is seen as unhealthy and additional health coverage is not applicable.

A customer is considered obese when his or her BMI is over 30. The BMI is calculated using the following formula:

$$BMI = \frac{Weight}{Height^2}$$

A customer is alcoholic if he or she drinks more than 4 glasses of alcohol each day or more than 18 glasses a week.

3A: SHARPEN YOUR PENCIL

Use Appendix B and the presentation to answer these questions.

1. Which of the following statements is/are true?

   - ☐ Values that are set (filled out) by the user can be overwritten by business logic, such as *decision tables* or *business rules*.
   - ☐ A *decision table* takes precedence over a *default rule.*
   - ☐ The inference engine uses a forward chaining mechanism.
   - ☐ A *business rule* takes precedence over a *decision table.*

2. The premium for a health insurance is calculated with the specified formula. Which logic element (business rule, decision table, default value) would you use to …

   … calculate the discount?

   _____

   _____

   _____

   _____

   … calculate the additional premium?

   _____

   _____

   _____

   _____

   … calculate the monthly premium?

   _____

   _____

   _____

   _____

3. A customer can apply for additional health coverage if he or she is neither obese nor alcoholic. To determine if a customer is alcoholic we will create a decision table. How would you write the decision table?

Is alcoholic?					

4. A customer is considered obese when his or her BMI is over 30. The BMI is not used in any other calculation, nor does it have to be shown to the user. Which elements would you use to determine if the customer is obese?

   - ☐ A business rule
   - ☐ A default value
   - ☐ A result attribute
   - ☐ A decision table
   - ☐ A reusable expression
   - ☐ A local named variable

   Please explain why.

   _____

   _____

   _____

3B: MODELLING LOGIC

Calculating the premium

Follow these steps to calculate the premium.

1. Create a decision table to determine the premium of the additional coverage (AdditionalCoverage.Premium).
2. Set a default value to determine the premium of the basic coverage (BasicCoverage.Premium).
3. Create a business rule and default value to determine the discount (Premium.Discount).
4. Set a default value to determine the basic premium minus discount (Premium.BasicPremium).
5. Set a default value to determine the additional premium (Premium.AdditionalPremium).
6. Set a default value to calculate the monthly premium (Premium.MonthlyPremium).
7. Now that we've created the logic elements to calculate the premium, open your application in the Runtime and test your calculations.
8. The premium is not recalculated after you change the excess amount or apply for additional coverage. How can you solve this problem?

   Which attributes should trigger a refresh ?

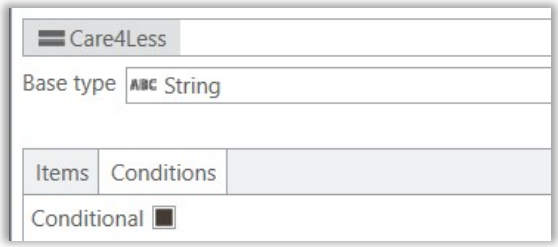
Applying for additional health coverage

Follow these steps to determine if a customer can apply for additional health coverage.

1. Create a decision table to determine if a customer is alcoholic.
2. Create a business rule and default value to determine if a customer is obese.
3. Create a business rule and default value to determine if the lifestyle of the customer is unhealthy.
4. Now we want to make sure additional health coverage cannot be selected when the lifestyle of the customer is unhealthy.

   Therefore, make the valuelist of the attribute AdditionalCoverage.Type conditional.
5. Test your application in the Runtime environment.

You may need to create "result attributes" to determine if a customer is alcoholic or obese.



3C: ADD LOGIC TO FLOWS AND PAGES

Adding logic to your flow

The company's response time depends on the type of coverage applied for. For basic coverage this is three working days, but for additional coverage the response time can be up to two weeks.

That's why we want to show a different result page to the user depending on the type of coverage applied for: basic coverage or basic and additional coverage.

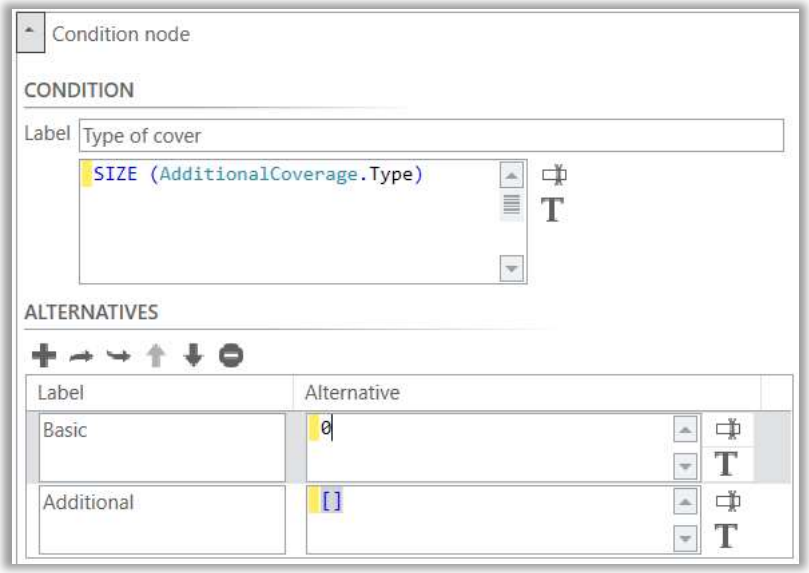
1. Change the 'Thank you' page to show the customer that the application for basic coverage is received. Write an appropriate response message.

2. Create a new page to show the customer the application for basic *and additional coverage* is received. Also write an appropriate response message.

3. Now we have to adjust the flow to display the different result pages. We have to know whether the customer applied for additional coverage or not. Which attribute holds this information?

   Add a condition node to the flow with two alternatives. What should the expression in the condition node look like?

   Connect each alternative to the correct result page.



Adding logic to a page

Depending on the screen resolution you may need to use F11 for full screen reading.

1. Create a validation rule to check if the applicant is not under 18 years old.
   Add the validation rule to the correct attribute.

2. If a customer does not consume alcohol, we do not want to bother him or her with the question: "How many glasses of alcohol do you drink?"

   Add a precondition to the "glasses of alcohol" attribute.

3. If a customer does not select an additional coverage we do not want to bother him with the other questions about the additional coverage. Add preconditions to these attributes.

4. Which attribute(s) should now trigger a refresh ?

5. Run your application to see if it is working correctly.

3D: EXPANDING OUR MODEL (OPTIONAL EXERCISE)

Validation rules

1. Create validation rules to:
   - check if the excess amount is at least € 170,-.
   - check if the start date of a coverage is at least two weeks in the future.
   - check if the end date of the additional coverage does not exceed the end date of the basic coverage.

Healthy lifestyle

The requirements to determine if a customer can apply for additional health coverage have changed. These are the new requirements:

To determine how healthy the lifestyle of a customer is, we will classify the lifestyle of the customer in one of three groups: *Unhealthy*, *Healthy* or *Very Healthy*.

Only customers with healthy and very healthy lifestyles can apply for additional health coverage. For customers with a very healthy lifestyle the premium will remain € 10,-. For customers with a healthy lifestyle the premium will be € 14,-.

The lifestyle of alcoholic customers is always classified as unhealthy. The classification of the lifestyle will also depend on the BMI:

BMI	Lifestyle
< 16	Unhealthy
16-19	Healthy
20-26	VeryHealthy
27-34	Healthy
>= 35	Unhealthy

1. Change the healthy lifestyle attribute so that it can contain the three values. Replace the business rule used to determine the value of this attribute by a decision table.

2. Change the logic used in the conditional value list for the type of additional coverage.

3. Change the logic used to determine the premium of the additional coverage so that the right premium is set, depending on the lifestyle.

4. Test your application in the Runtime.

EXERCISE 4 – MULTIPLE INSURED PERSONS

Today, we'll be making sure that a single insurance policy can cover multiple people. One policy will cover all members of a household.

In the previous exercises, we've been working with a single customer, but now we'll need to deal with multiple instances. Furthermore, we want to make sure the domain model contains more reusable elements.

4A: SHARPEN YOUR PENCIL

Read appendix C and D about instance management.

1. Which new *singleton entity* do we have to add to the domain model?

_____

_____

_____

_____

2. Which *entities* can appear more than once in the new situation?

_____

_____

_____

_____

3. Which *relationships* between entities should there be?

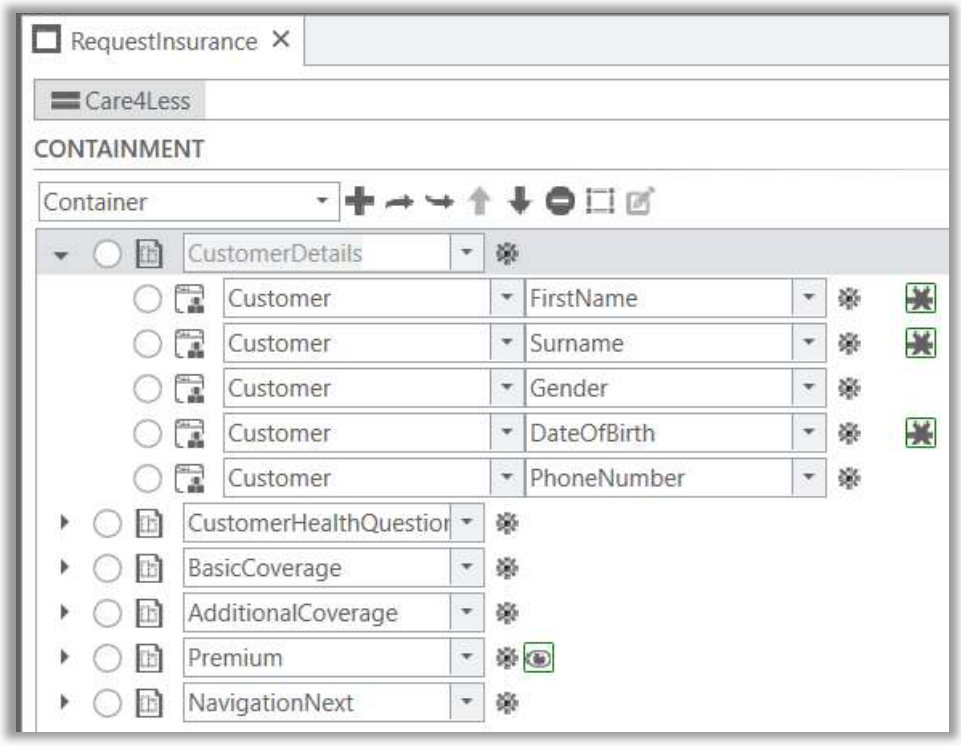
_____

_____

_____

_____

4. What is wrong with this page in the new situation?



_____

_____

_____

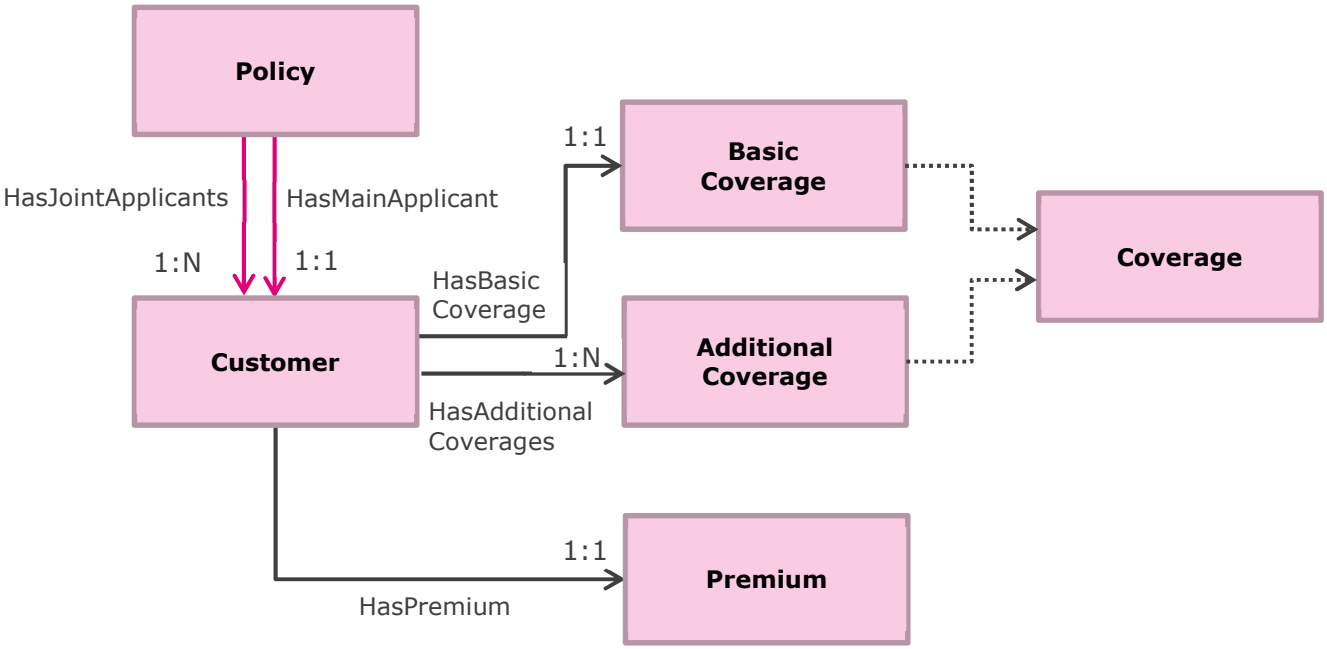
_____

5. Which of the following statements is/are true?

- ☐ All instances must be created explicitly.
- ☐ There can be only one active instance for each entity.
- ☐ If a transactional flow ends with an "OK" endpoint, all created instances will be stored.
- ☐ The inference engine can clear user set values in the Backbase Runtime.

4B: ADJUSTING THE DOMAIN MODEL

Now that we've established all the information that we need to adjust our domain, we can start making changes in Studio. The figure below shows the entities needed. Use this as a guide to adjust the domain model.

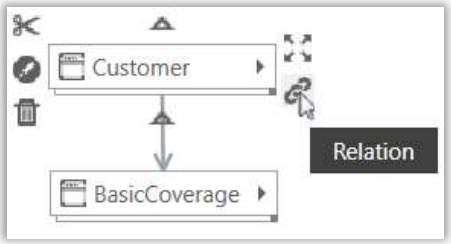


1. Create the policy entity and adjust the other ones. Are all entities singleton?

2. Create the relations. This can be done by creating new relation elements or by creating an *entity-relationship diagram* (ERD).

   Right-click in an ERD to add existing entities. Drag and drop the *chain* icon to create a relation.

   Each relation has a name and a reverse name. Both can be used in your logic to indicate which instance you are referring to. Make sure these names have a functional meaning.

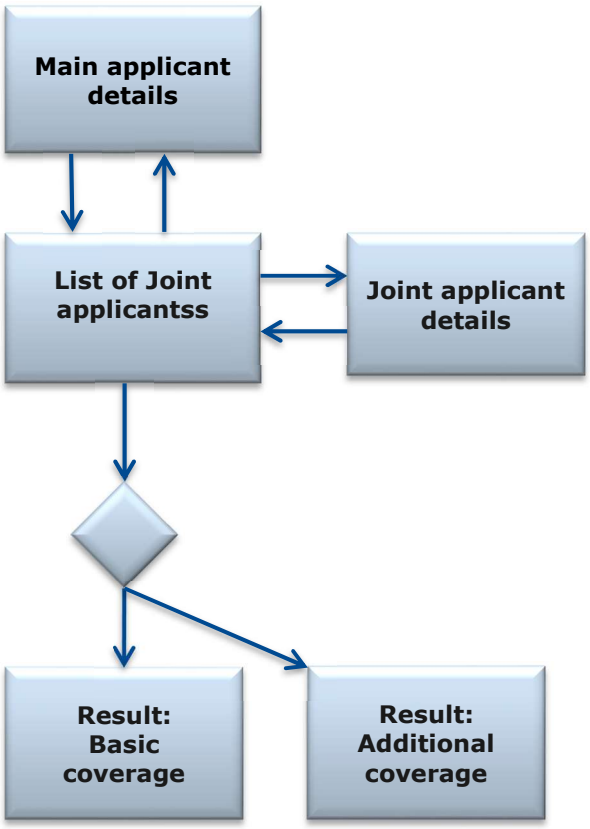


4C: ENTERING DATA FOR MULTIPLE INSTANCES

Take a look at requirement no. 11. The customer should be able to add all people that need insurance to the policy, the so called joint applicants.

The customer should also be able to add instances of AdditionalCoverage for each type of additional coverage for each customer.

To implement these requirements, we will change our main flow.



On the first page the *Main applicant* will be able to request his/her insurance.

The second page will display a *list of all Joint applicants*. On this page, the user has the ability to add, edit and delete a joint applicant.

On the third page the insurance for a *joint applicant* can be requested.

Finally, we will display the *thank you* page for insurance with only basic or insurance with additional coverage as well.

Read Appendix C and D for more information on Instance Management.

The main applicant page

First, we will adjust the *Request Insurance* page so that it contains the details for the main applicant.

Because the customer entity has become non-singleton, you will need to indicate which customer's information you would like to display, in this case those of the main applicant.

1. Create a new *container* called "Customer" to contain all customer details. Add the containers that contain the personal and lifestyle attributes to this container.

2. Use the 1:1 relation between customer and BasicCoverage to indicate which basic coverage details should be displayed. Drag and drop the relation onto the "Customer" container and select the container which holds the coverage attributes.

3. Now use the 1:1 relation between customer and premium to indicate which premium details should be displayed. Display all attributes in this block as read-only.

   Now there are four containers in the container "Customer", but this container is not yet on a page.

4. Now add the Customer container to the page via a relation. Which relations should you use?

5. Test whether you can view the first page. Don't worry about navigating or your logic components just yet.

Adding additional coverages

Now we will adjust the Customer container to make it possible to add multiple instances of the entity AdditionalCoverage. For each type of additional coverage that the customer wants to apply for, an instance should be created.

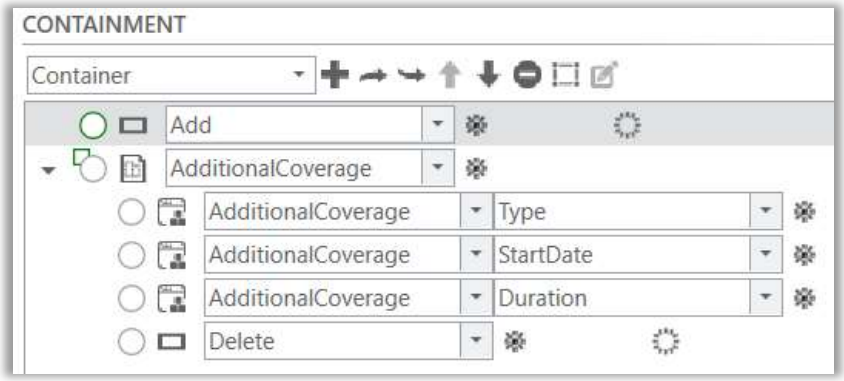
1. First we will adjust the attribute holding the type of additional coverage. The attribute should no longer be multivalued. The belonging valuelist should no longer be conditional.

2. Create buttons and events to be able to add and delete additional coverages.

3. Open the container holding the additional coverage attributes and add the delete button to this container. Don't forget to select an event and event type for this button.

4. Now create a new container to hold all instances of AdditionalCoverage and give it a name (e.g. AdditionalCoverages). Add the add button to this container.

5. Add the container with the additional coverage attributes with a repeat expression. What is the repeat expression?



6. Add the AdditionalCoverages container to the Customer container.

7. Create a service to delete an instance of AdditionalCoverage (AQ_DeleteInstance). Fill out the parameters 'type' and 'type-scope'. Add the service to the flow.

8. Create a service to create an instance of AdditionalCoverage (AQ_CreateInstance). Fill out the parameters 'type' and 'attribute-path'. Add the service to the flow.

9. Create a validation rule to check if the customer does not apply for additional health coverage when his lifestyle is unhealthy. Add the validation rule to the attribute holding the type of additional coverage.

10. Test whether you can add and delete additional coverages.

11. You can now add as many instances of AdditionalCoverage as you like. This is of course not correct. To make sure you can only add two additional coverages, add a precondition to the add button.

Joint applicants

Let's create a new page to display a list of joint applicants. We will use a special kind of container, the AQ_InstanceselectorPlus, to display a list of joint applicants.

1. Appendix D gives an example of how to use the AQ_InstanceselectorPlus container. The appendix explains how to create a page that will display a list of instances of family members and how to manage these instances. For more information, read the appendix, then continue with the exercise.

2. First create the container of type AQ_InstanceselectorPlus to display the list of joint applicants. Create buttons and events to be able to add, edit and delete joint applicants.

3. Create a page 'JointApplicants' and add the new container holding the list of joint applicants to this page. Add the page to your main flow.

4. Create a new flow (sub flow) for adding a joint applicant. This flow must contain a *service* to create a customer instance and create a *page* on which you can fill out the details of the insurance request for a joint applicant.

   Can you reuse some of the containers you've created earlier for the main applicant page? Do you need a relation to display the correct customer details? Why (not)?

5. Add two exit nodes to the sub flow. Don't forget to add events to the exit nodes.

6. Add the sub flow to the main flow. Connect the sub flow to the JointApplicants list page.

   It is now possible to add joint applicants.

7. To edit a joint applicant, add a page to the main flow. Which page should that be? Connect this page to the JointApplicants list page.

   It is now possible to edit joint applicants.

8. Create a service to delete a joint applicant. Add the service to the main flow.

   It is now possible to delete joint applicants.

9. Test whether you can add, edit and delete joint applicants. Don't worry about your logic components just yet.

EXERCISE 5 – LOGIC AND MULTIPLE INSTANCES

5A: SHARPEN YOUR PENCIL

The user should now be able to enter personal details for each applicant, and details about their lifestyle and desired coverage(s). All this information is needed to calculate the right premium, and to determine whether the customer can apply for additional health coverage. However, the logic to do this, was made with one customer in mind.

1. What should the premium be for two insured persons, who both take additional health and dental coverage, but one choosing the minimum excess, and the other choosing € 260 excess?

   _____

   _____

   _____

   _____

2. Which expression can you use to get the monthly premium of an active customer?

   _____

   _____

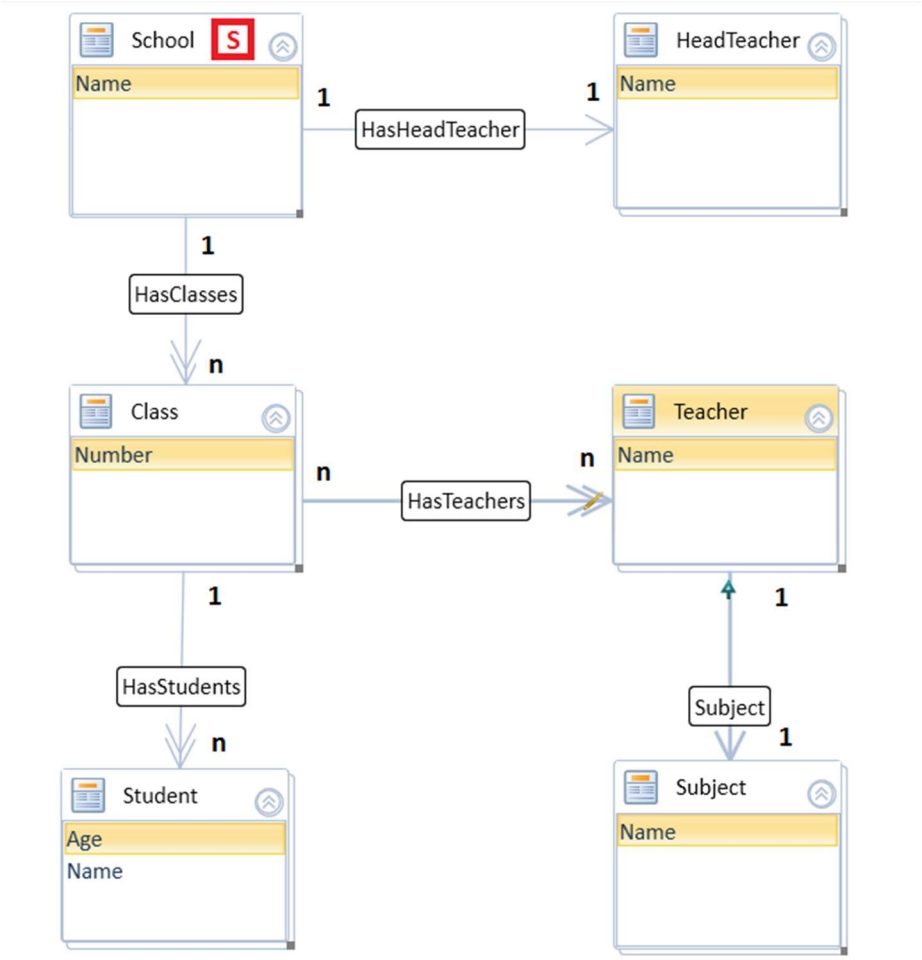
   _____

   _____

3. Which of the following functions can you use to calculate the total of all monthly premiums?

   - ☐ PRODUCT
   - ☐ SUM
   - ☐ COUNT
   - ☐ none of the above

4. Let's practice some COLLECT statements.
   Study this model:



Which expression can you use to get:

a) All students

_____

_____

_____

_____

b) All students with age 10

c) All students aged 10 in class 6

d) The age of the oldest student of class 6

5. Which expressions can you use to get a collection of all monthly premiums?

5B: IMPLEMENTING THE PREMIUM CALCULATION

Currently, the Runtime displays errors when you select your application. To resolve these errors we need to modify or logic elements.

1. Add the *relationship* to the logic used to calculate the discount, so that the premium relates to the right basic coverage.

2. Add relationships to the logic used to determine the basic premium.

3. Change the logic used to determine the additional premium. What happens if there is no additional coverage?

4. Adjust the condition node in your flow. If there is at least one customer who applied for an additional coverage, the result page for additional coverages has to be shown. Otherwise the basic coverage result page has to be shown.

5. Add an attribute 'OverallPremium' to the policy entity and add a default value of type expression.

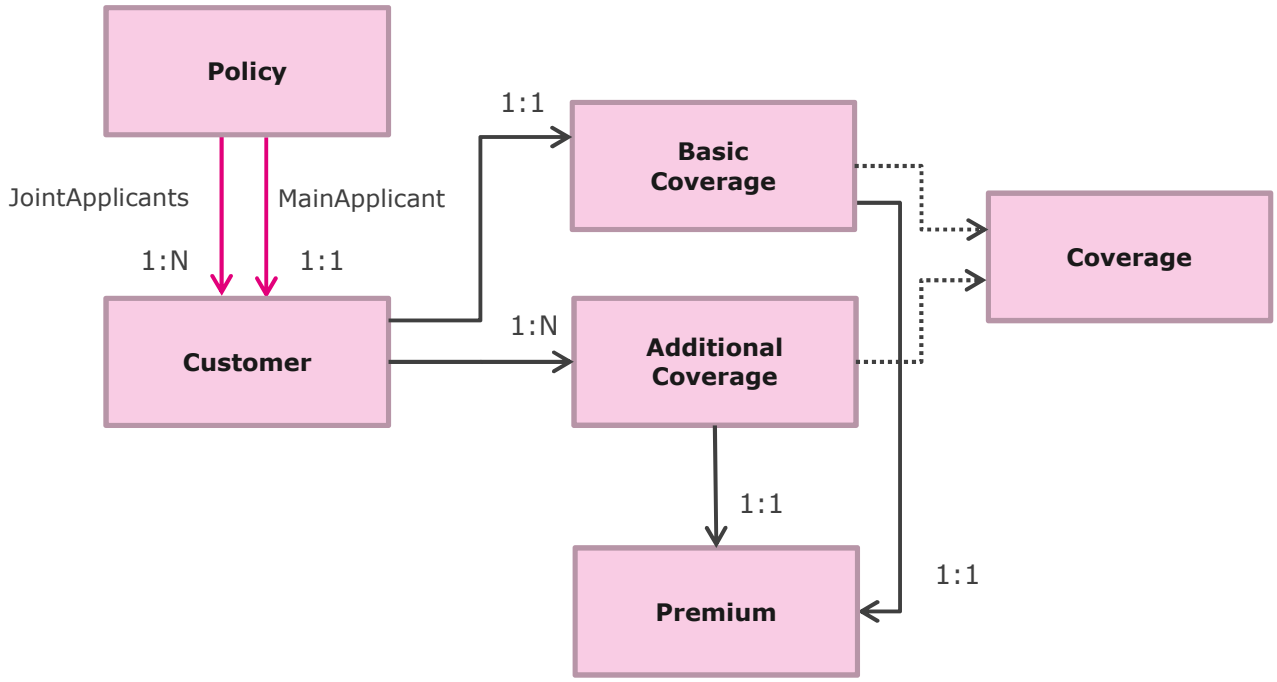
   Which expression should you use?

   Now add this attribute to your result pages.

6. Test your application in the Runtime.

5C: EXPANDING OUR DOMAIN MODEL (OPTIONAL EXERCISE)

Each instance of Coverage should have its own Premium. So the domain model will look like this.



1. Open the ERD, delete the relation from Customer to Premium and create the two new relations.
2. The Premium entity should only have 2 attributes, namely Amount and Discount. Create these attributes and delete the others. Also delete all belonging logic elements.
3. The Coverage entity should no longer contain premium information. Delete this attribute. Also delete all belonging logic elements.
4. Change the logic to determine the discount.
5. Create the logic to determine the value of the premium amount.
6. Change the logic to calculate the overall premium.
7. Now open the Customer container, delete the old attributes and relations and add the new attributes and relations.
8. Test your application in the Runtime.

EXERCISE 6 – DOCUMENTS

In this exercise, we will focus on requirement no. 13.

6A: SHARPEN YOUR PENCIL

1. Can you think of some differences between pages in a (web) application and documents?

_____

_____

_____

_____

2. What is the difference between a content style and a presentation style?

_____

_____

_____

_____

3. What is the difference between a content item and a text item?

_____

_____

_____

_____

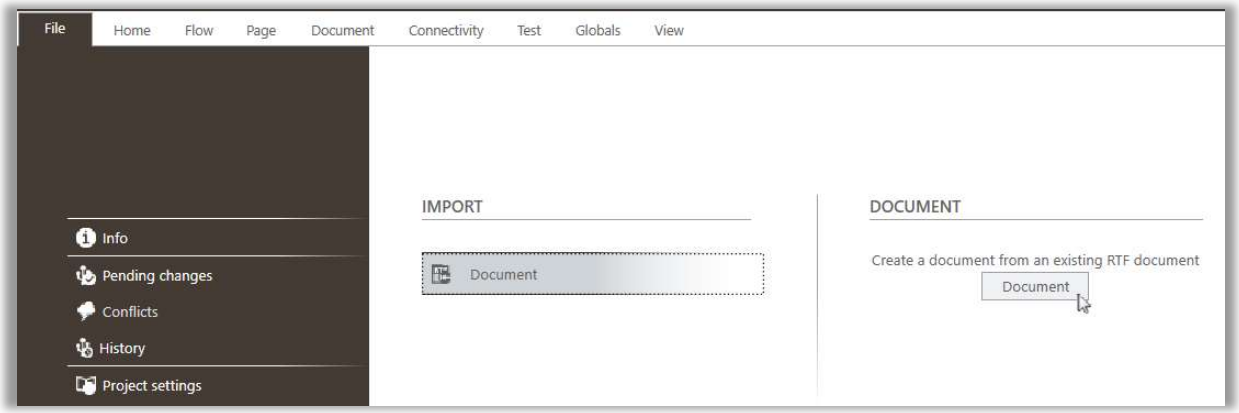
6B: MODELLING DOCUMENTS

You can create a completely new document in Backbase from scratch, but you can also import an existing RTF document that can serve as a starting point. For our case, the company 'Care for less' has prepared three RTF documents. The documents show which content is dynamic, i.e. which data from the domain model must be used. In this exercise, you will import at least one from these three RTF documents, the Quote, and optionally the Letter and Terms as well. Each RTF document will be imported as a *content item*, and together they become one *document* in Backbase.

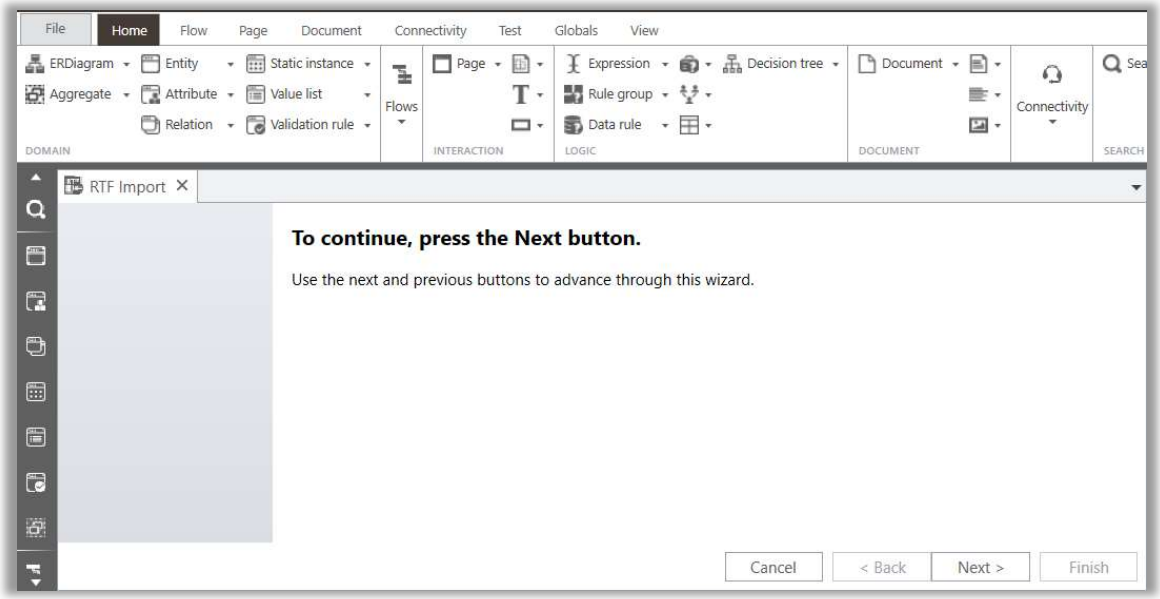
Importing documents

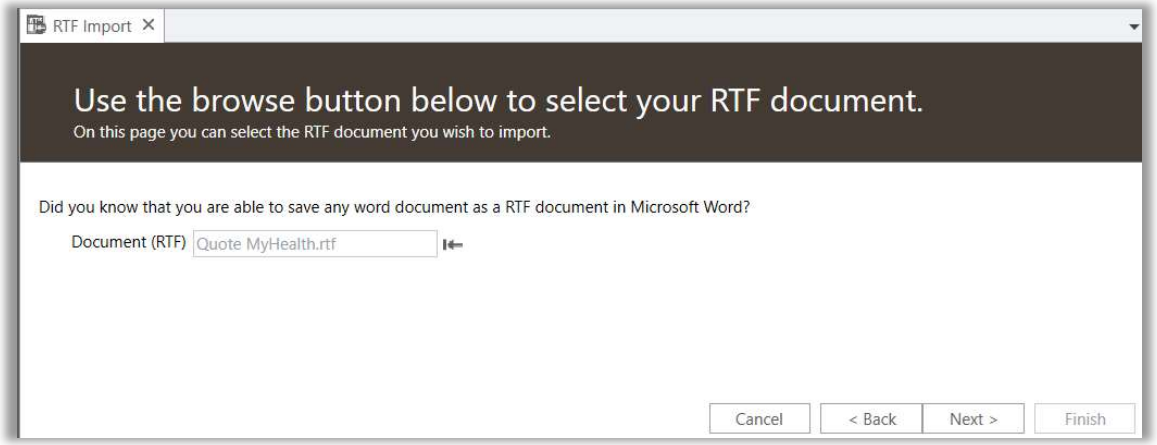
Follow these steps to import the prepared documents.

1. Import the source document "Quote_.rtf":
   Choose **File > Import**, and press the button Document.



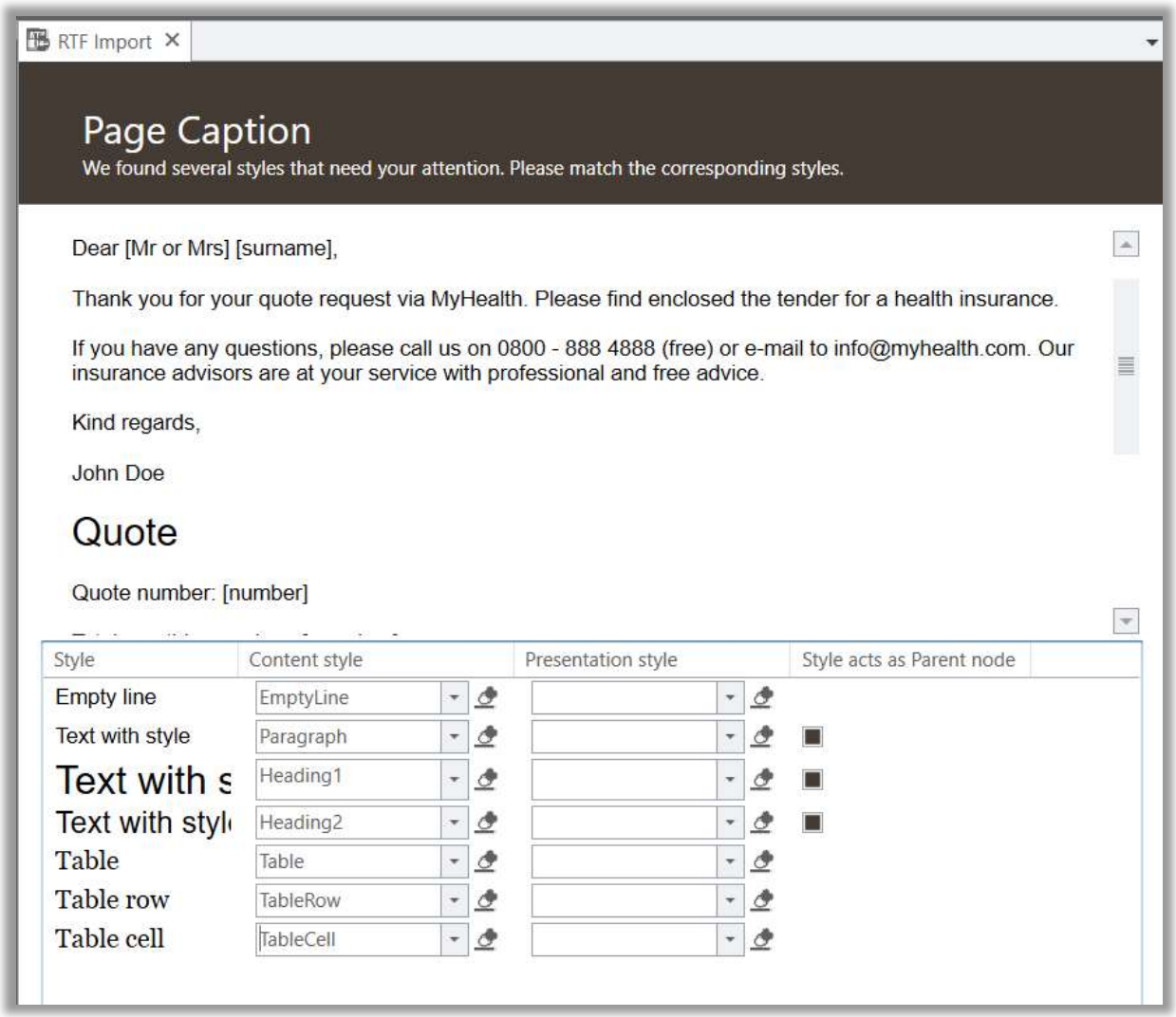
A new tab page *RTF Import* appears.
Press **Next >**, and use the **Browse** button **|<-** to select the RTF-file from the folder "BMF – Documents exercise" on your desktop.



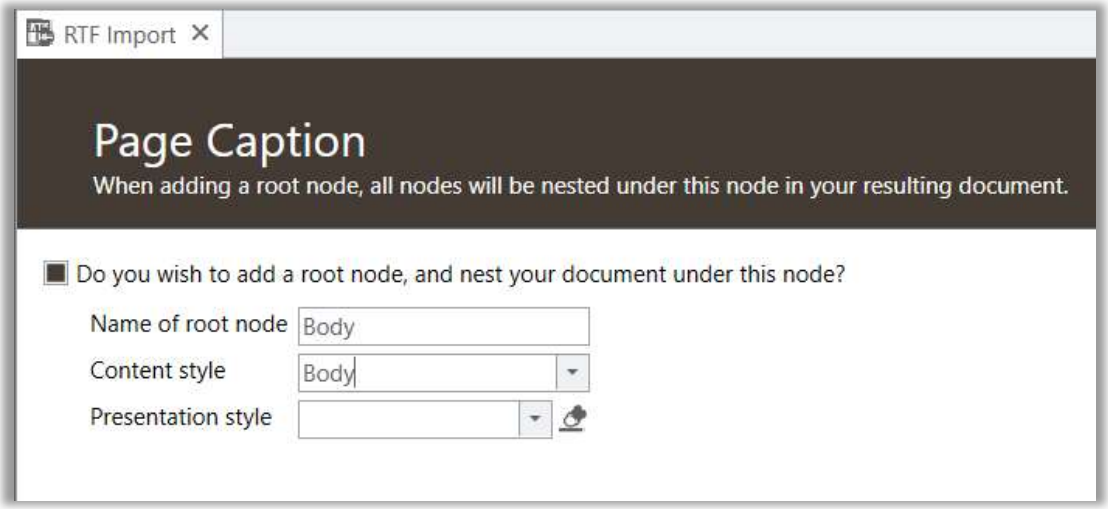


2. The import wizard shows the document in the first pane. Below that it show a pane to map the styles. On the left the styles detected in the source document. When you select a style, you can see the associated text appear in green in the first pane.

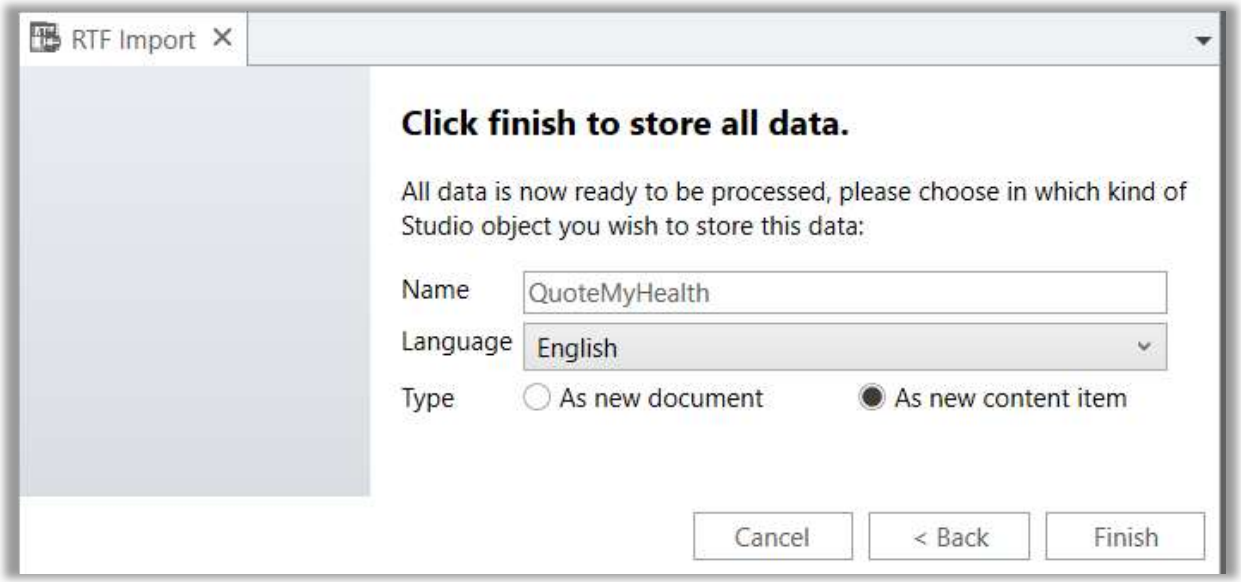
3. Now map the styles from the source document onto pre-defined content styles in Backbase. Indicate that the Heading styles will ***act as parent node.***


RTF Import
Page Caption
We found several styles that need your attention. Please match the corresponding styles.
Dear [Mr or Mrs] [surname],
Thank you for your quote request via MyHealth. Please find enclosed the tender for a health insurance.
If you have any questions, please call us on 0800 - 888 4888 (free) or e-mail to info@myhealth.com. Our insurance advisors are at your service with professional and free advice.
Kind regards,
John Doe
Quote
Quote number: [number]
Style
Content style
Presentation style
Style acts as Parent node
Empty line
EmptyLine
Text with style
Paragraph
Text with s
Heading1
Text with styl
Heading2
Table
Table
Table row
TableRow
Table cell
TableCell

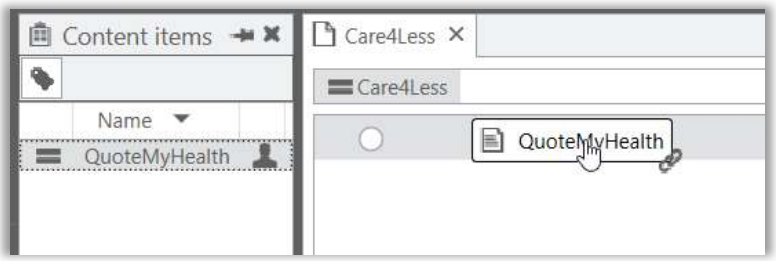
Press the **Next >** button. Check the box that you wish to add a root node, give it a name (e.g. *Body*) and select the content style *Body*.

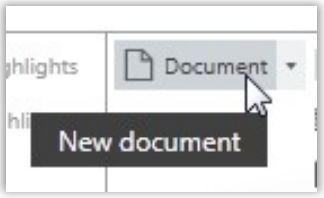


4. Press the **Next >** button. Select the radio button *As new content item* and enter a name for the content item. Then press *Finish*. The content item opens, and press *Save* before you continue!



5. You now have a new content item, but no document yet. Create a new document and drag and drop the new content item onto the document (on the top left circle **O**) and *Save* this.



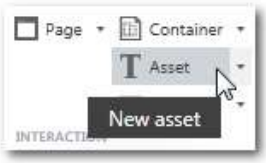


Generating a document

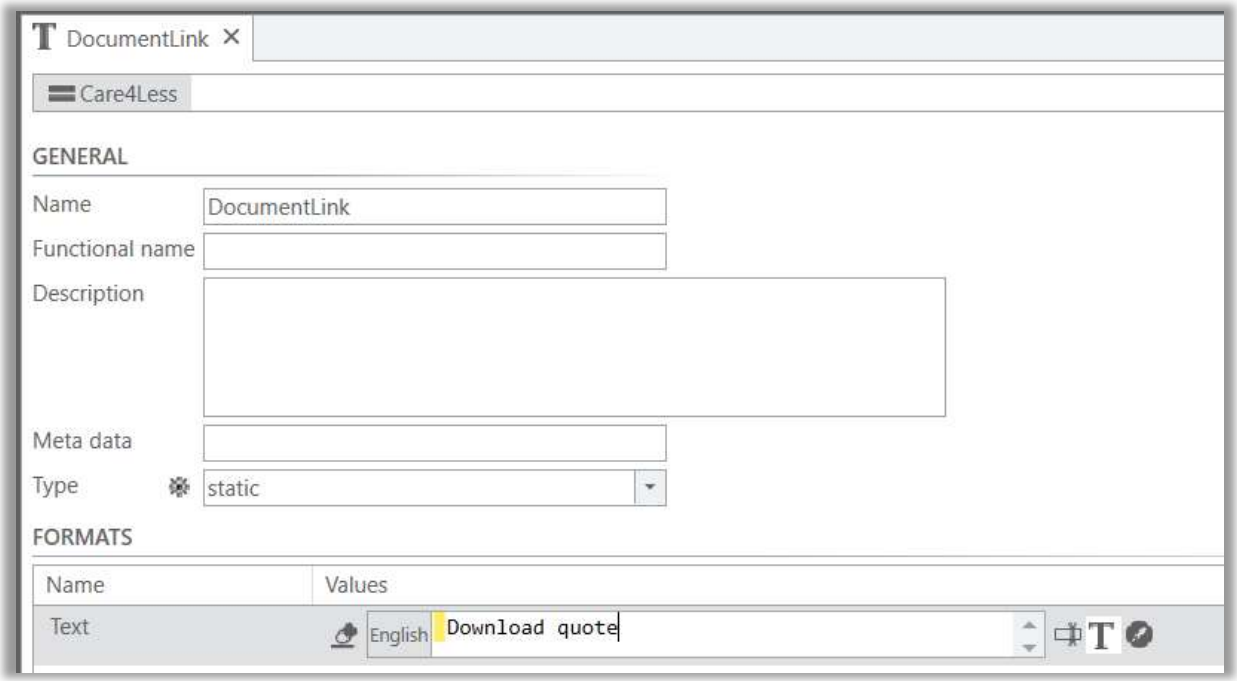
Before we are going to enhance and integrate our document with the domain model, we will first generate the document to see how it looks in its current state.

We need a special container, the **AQ_DocumentLink** container, to display a link on the page that the user can click on to download or view the document. You will create an asset and then a new container of this type.

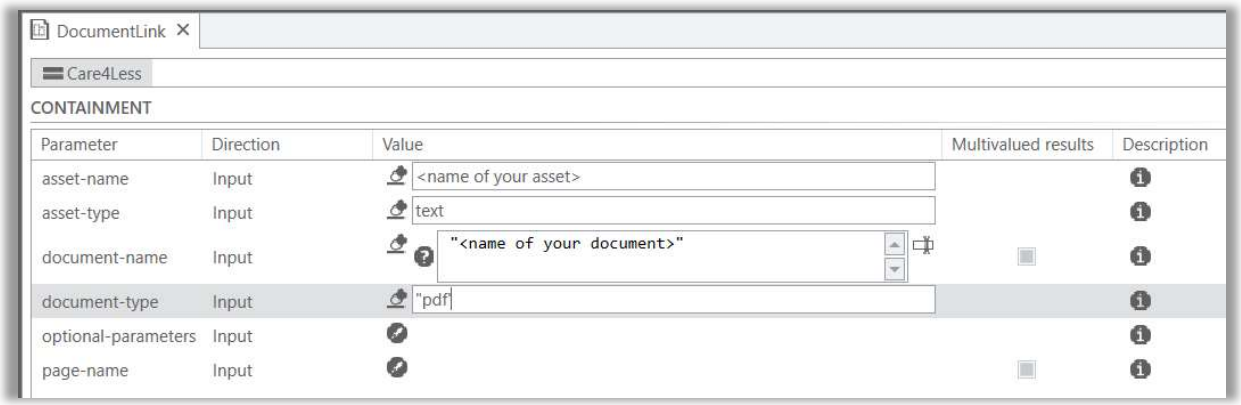
1. The container uses an *Asset* that contains the text that will be displayed as a link. Create a new asset first and name it DocumentLink.



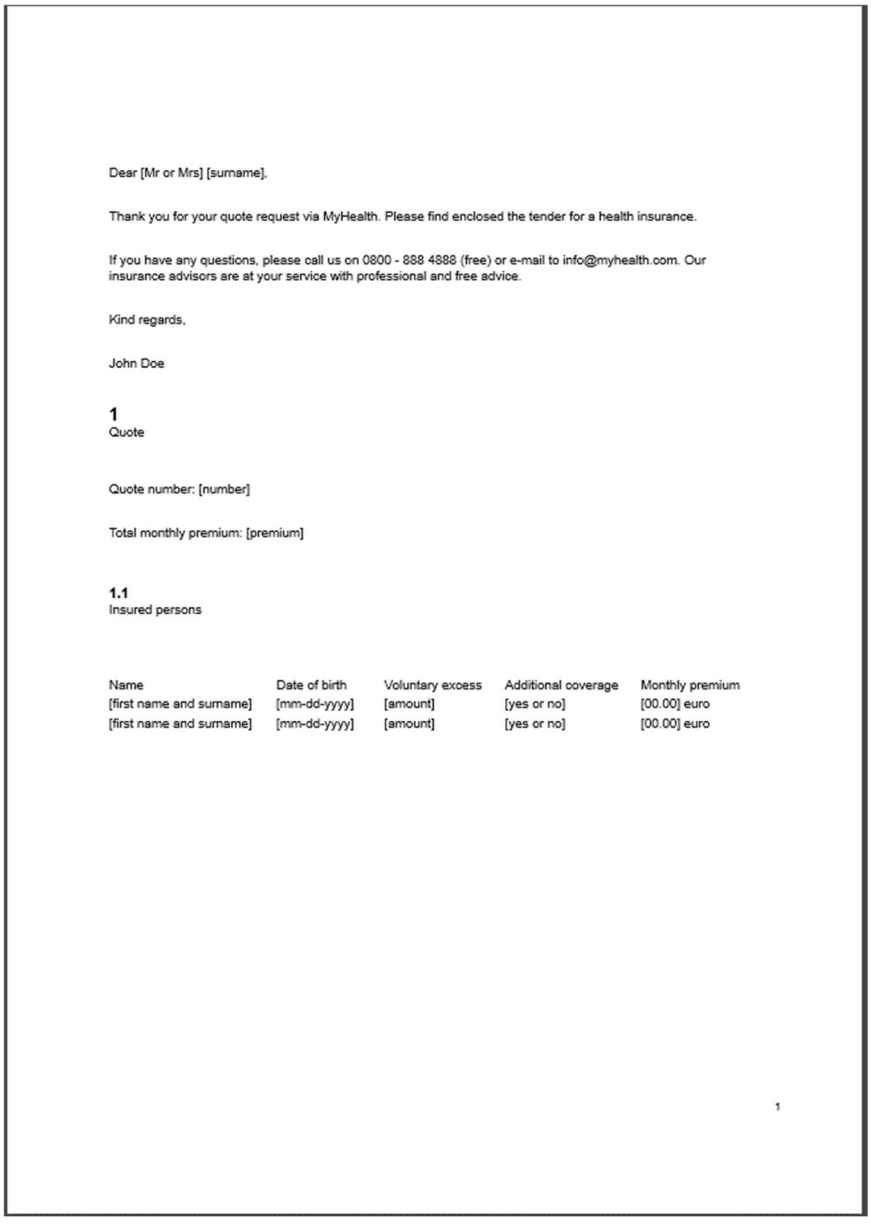
   - Choose type *static*.
   - The static asset type has only one format, which is *text*. Enter the text that will be shown as a link on your page, e.g. *Download quote*. Save it.



2. Then create a container of type **AQ_DocumentLink**. Enter the parameter values:
   a. "asset name": the name you just gave to the Asset,
   b. "asset type": the asset contains text,
   c. "document-name": enter the name of your document in Backbase.
   d. "document-type": This defines the output type of the document, which will be "pdf" in our case.



3. Add the container to one or both of the result pages (i.e. the Thank you pages), and Save the page(s).



Dear [Mr or Mrs] [surname],

Thank you for your quote request via MyHealth. Please find enclosed the tender for a health insurance.

If you have any questions, please call us on 0800 - 888 4888 (free) or e-mail to info@myhealth.com. Our insurance advisors are at your service with professional and free advice.

Kind regards,

John Doe

1
Quote

Quote number: [number]

Total monthly premium: [premium]

1.1
Insured persons

Name	Date of birth	Voluntary excess	Additional coverage	Monthly premium
[first name and surname]	[mm-dd-yyyy]	[amount]	[yes or no]	[00.00] euro
[first name and surname]	[mm-dd-yyyy]	[amount]	[yes or no]	[00.00] euro

1

4. Test the document in the Runtime. The result should look like this:

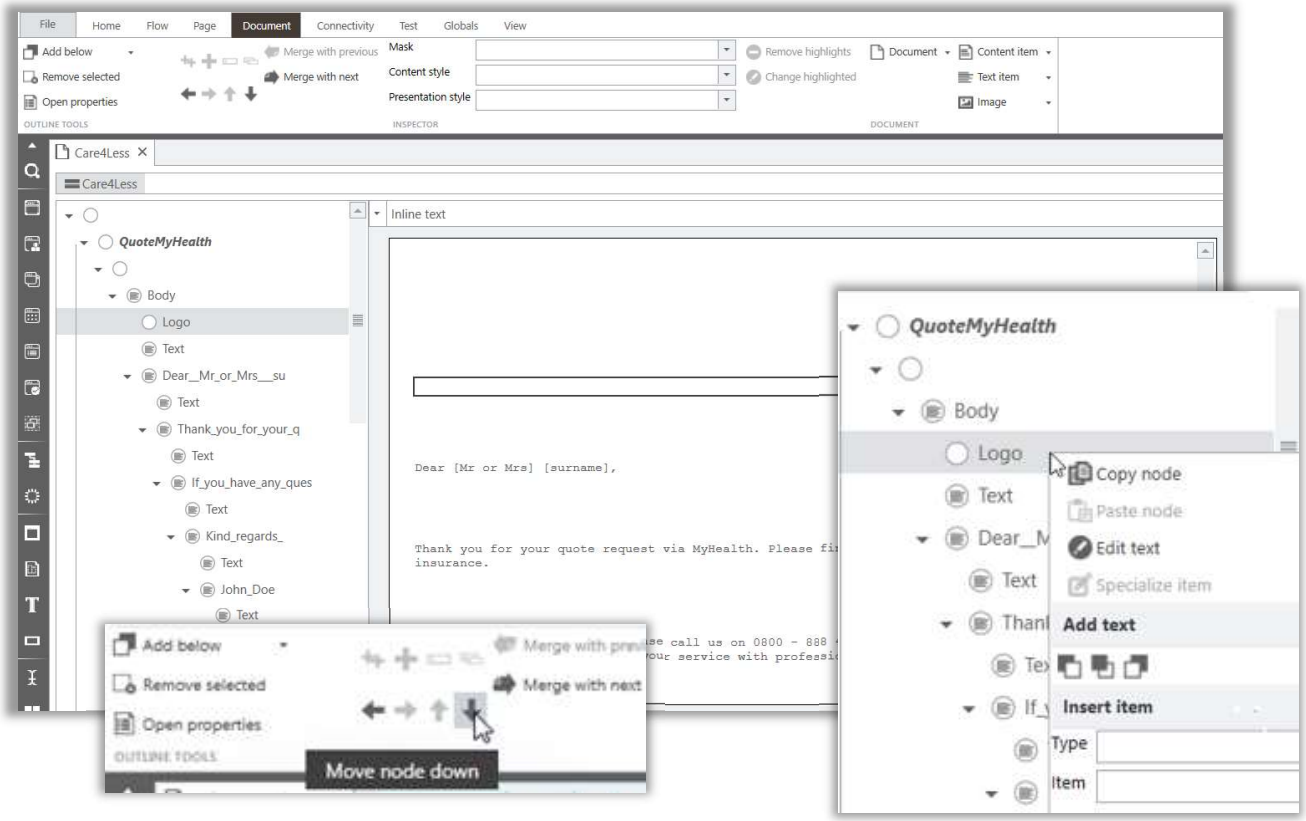
If it doesn't look like this, then you could try to change it, but it is probably faster and less work to throw away the imported content item and document, and import the rtf-document(s) again.

Make sure you follow each step carefully.

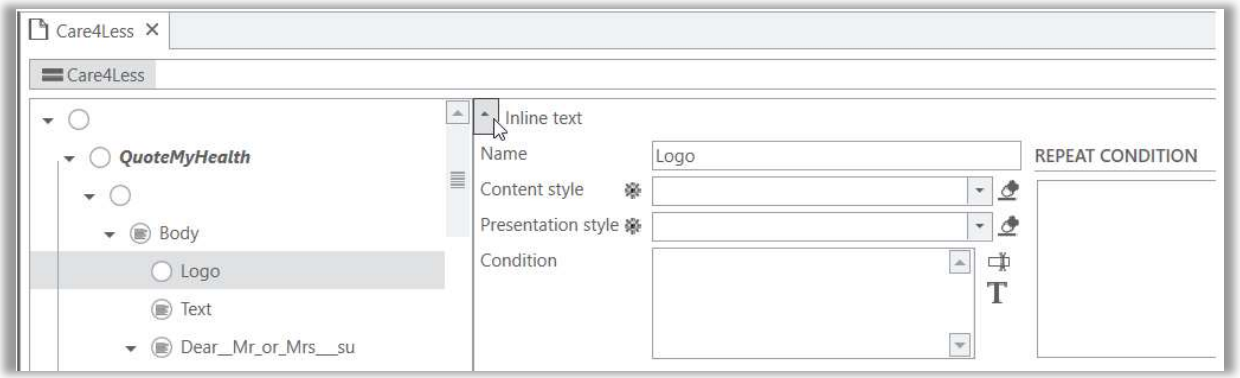
Navigating a document in Studio

Before we start making changes to the document, you need to learn the basics of navigating through it:

On the left you see a tree (a hierarchy of parent-child relations) that represents the structure of your document. When you select a **node**, i.e. an element in the tree on the left side, the related items in the document are highlighted on the right side (you might have to scroll to see it).



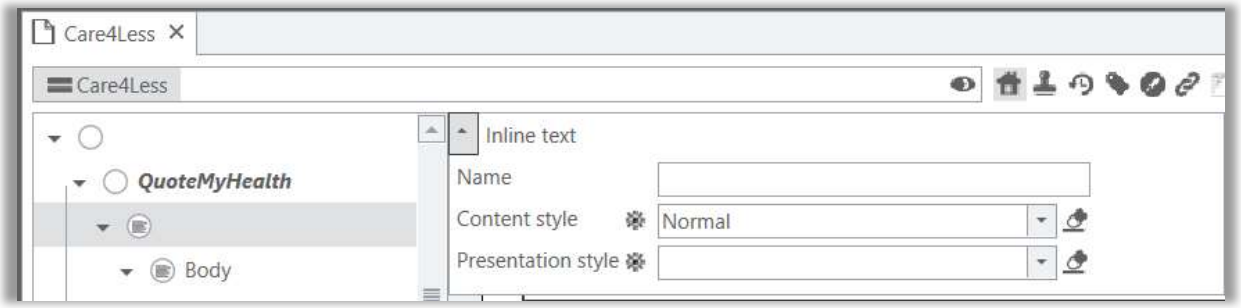
You can create, remove and manipulate nodes with the tools in the ribbon, or with a right mouse-click. You can click on the small arrow in the top left corner to see and edit the properties of the selected node, i.e. the name, the styles and maybe a repeat condition.



Quote

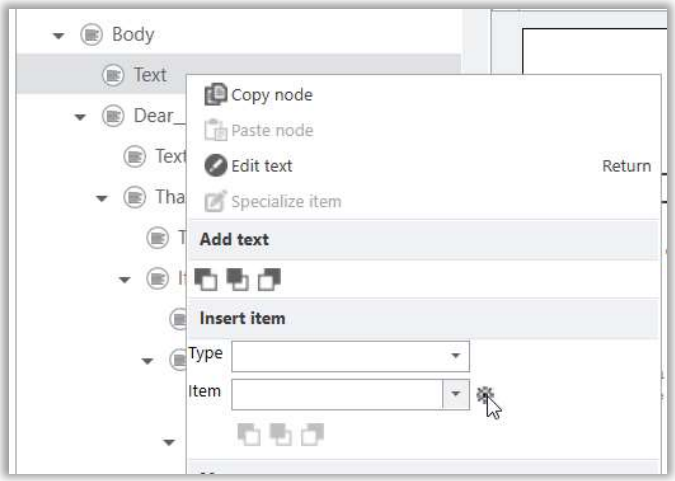
We will now go through some changes to make the document look more presentable. We will start with the Quote.

1. The first content item (just below Quote) must have content style "Normal". You can select the content style in the properties. If this content style does not yet exist, create it first and then add it to this node.

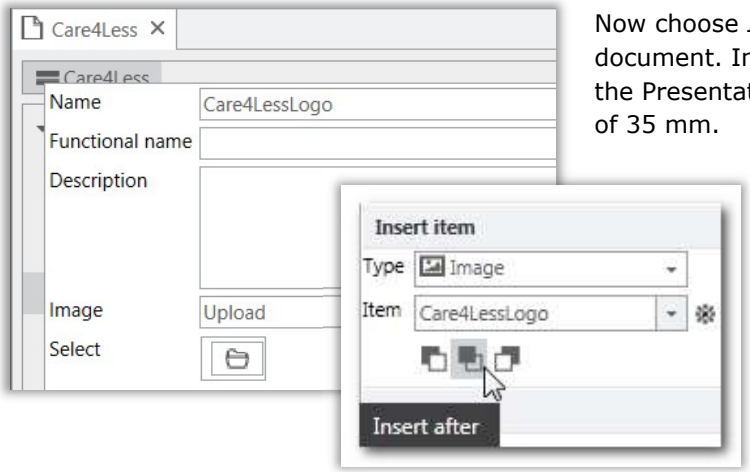


Tip: content styles and presentation styles are global objects. To add or view global objects, go to the Globals tab.

2. You are now going to add the logo to the document. Select the node Text (right under the Body node), open the pop-up menu with a right mouseclick, and under *Insert item* choose type **Image** and create a new image.



Give the new image a name, e.g. Care4LessLogo, and browse for the Care4Less image.

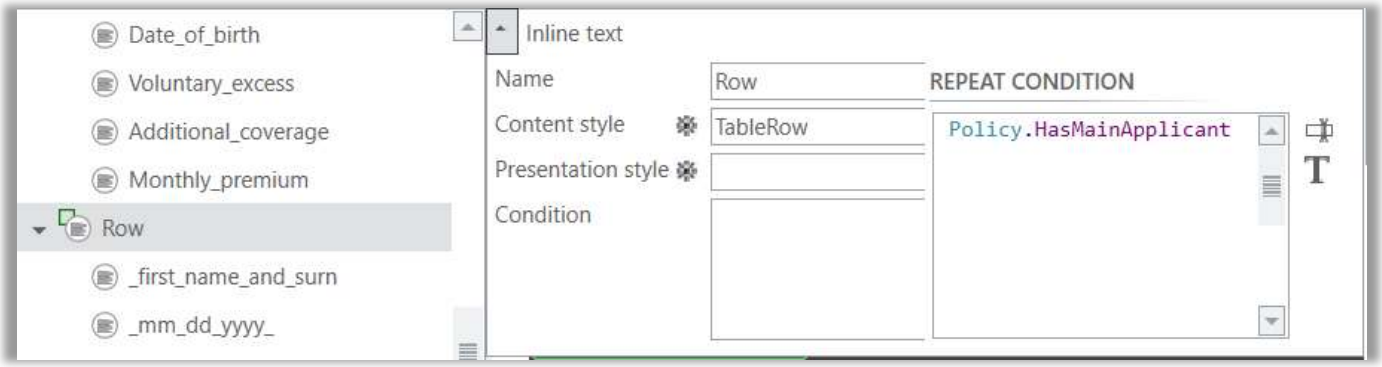


Now choose *Insert after* to add the logo to the document. In the properties of the logo, give it the Presentation Style *RightAligned* and a width of 35 mm.

3. The source document indicates that for all customers (i.e. both the main applicant and the joint applicants) we want to see the name, date of birth, premium, excess amount and a table with all coverages. So that will be our next goal.

   How can we do that?

   In the document tree, under the node Table, you can find a nodes called *Row*. The first Row is used as a header of your table. You want to repeat the second Row for Main applicants (and thereby all of its children). So select this node, open its properties and add the right *repeat condition*.



   Let's also add repeat to the 3rd Row for Join applicants. What relation will you use?

   Note that the repeat condition on this node allows you to refer to a Customer in this node *and all of its children*. Save the document.

4. Now it is time to add some fields to the document. Doubleclick on the node in the tree that contains the placeholder for the name and date of birth.

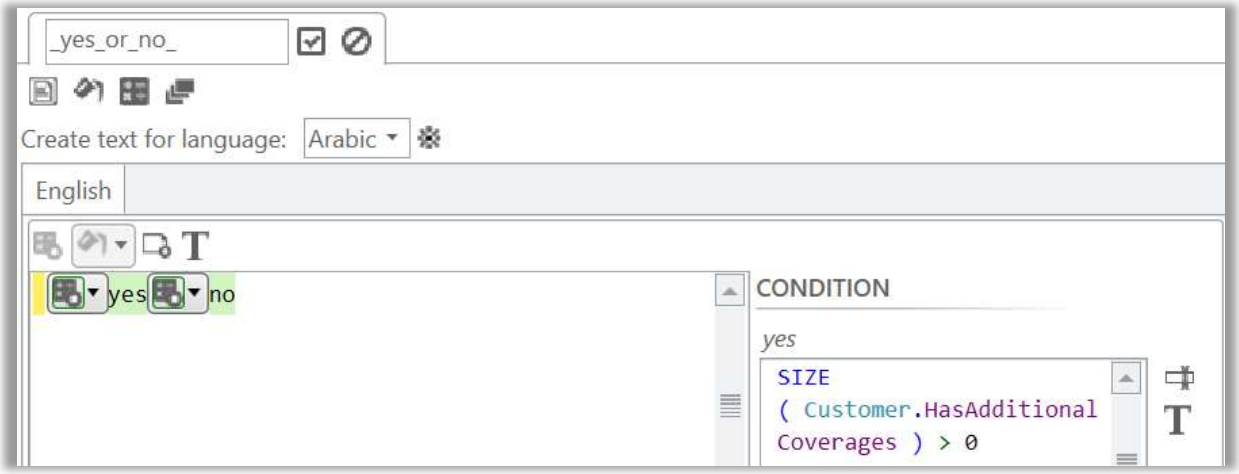
   The text item opens on the right side (you might need to close the properties and scroll to it). Select the text of the placeholder and add a field. Edit the field to enter the customers first name and surname.

Add another field to the same node to show the customers date of birth. Close a node when you are finished with it, by clicking on the small checkmark.

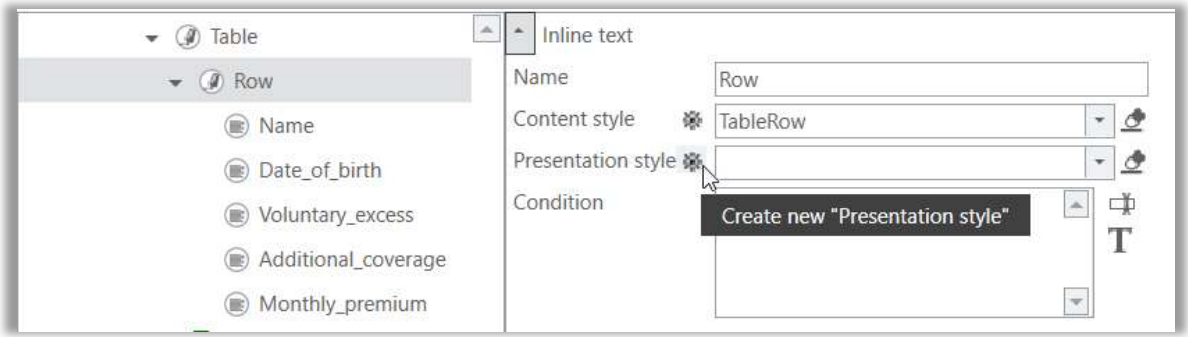
Save your document, and run a test every now and then to see if you are making progress!

Replace the placeholders in your document for Date of Birth, Monthly premium and Excess amount with the appropriate fields. The next step will be to add coverage data to your document.

5. There is a cell in the customer table to display the additional coverage. There are 2 options for customer who has added additional coverage it should state "yes", otherwise "no". Let's add condition to the text we have there.



6. The source rtf document shows that the table rows should have a blue background colour. You can realize this by making custom presentation styles with a specific name that represents the colour code.



Create two presentation styles for the different shades of blue, e.g. named *blck_bg_77c6ff_tc_000000_tw_100_* for the header row, and *blck_bg_c6e7ff_tc_000000_tw_100_* for the other rows.

Save the document and run a test. You should see a coloured table like this one:

Name	Date of birth	Voluntary excess	Additional coverage	Monthly premium
John Doe	1972-01-01	€ 170.00	yes	€ 109.00
Jane Doe	1980-01-01	€ 325.00	yes	€ 106.05
Johnny Doe	2000-02-02	€ 250.00	yes	€ 104.05
Janie Doe	2010-12-12	€ 170.00	yes	€ 109.00

7. Review the quote details and replace the other placeholders with the correct attribute values.

8. Save and test your document in the Runtime!
   Is all the required data visible?


Care 4 less
The best medical care
Dear Mr Doe,
Thank you for your quote request via MyHealth. Please find enclosed the tender for a health insurance.
If you have any questions, please call us on 0800 - 888 4888 (free) or e-mail to info@myhealth.com. Our insurance advisors are at your service with professional and free advice.
Kind regards,
John Doe
1
Quote
Quote number: QMH_01008
Total monthly premium: € 428.10
1.1
Insured persons
Name | Date of birth | Voluntary excess | Additional coverage | Monthly premium
John Doe | 1972-01-01 | € 170.00 | yes | € 109.00
Jane Doe | 1980-01-01 | € 325.00 | yes | € 108.05
Johnny Doe | 2000-02-02 | € 250.00 | yes | € 104.05
Janie Doe | 2010-12-12 | € 170.00 | yes | € 109.00
1

OPTIONAL: EXERCISE 7 – DECISION TREES

The customer should be able to get some advice about his or her weight. In this assignment we will therefore create three decision trees:

1. To diagnose the health risk of the customer
2. To advice the customer about types of treatment
3. To advice the customer about a diet

7A: SHARPEN YOUR PENCIL

Read Appendix E about how to create a decision tree.

1. What would be good symptoms for the decision trees? Write down at least three symptoms for each decision tree.

_____

_____

_____

_____

2. When would you make an attribute not askable?

_____

_____

_____

_____

3. How do you start building a decision tree?

_____

_____

_____

_____

7B: CREATING DECISION TREES

A tree to diagnose the health risk

To diagnose the health risk, we need to know the customer's BMI classification and abdominal girth classification.

The abdominal girth is classified as **'normal'** if smaller then 102cm for a male and smaller then 88cm for a female. It is classified as **'big'** if larger then 102cm for a male and larger then 88cm for a female.

This BMI classification is as follows:

BMI	Classification
< 18.5	Underweight
18.5 – 25	Normal
25 – 30	Overweight
30 – 35	Level 1 obesity
35 - 40	Level 2 obesity
>=40	Level 3 obesity

The health risk is determined as follows:

Health Risk		
	Abdominal girth classification	
BMI classification	***Normal***	***Big***
Overweight	Increased	High
Level 1 obesity	High	Very High
Level 2 obesity	Very High	Very High
Level 3 obesity	Extreme	Extreme

1. Create an attribute for the BMI classification of the customer. Make sure this attribute is not askable. Create a decision table to determine the BMI classification.

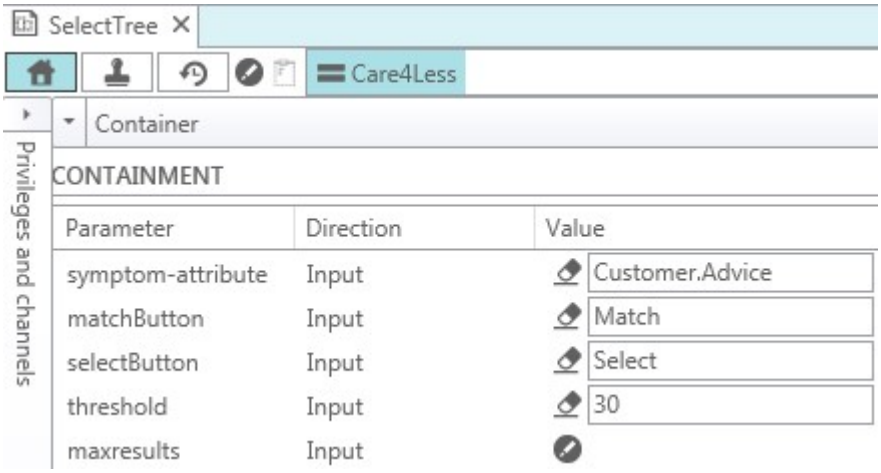
2. Create an attribute for the abdominal girth and an attribute for the abdominal girth classification of the customer. Make sure this last attribute is not askable. Create a decision table to determine the abdominal girth classification.

3. Create a new attribute for the health risk. This attribute will be set in the decision tree.

4. Read Appendix E for more information on how to create a decision tree. Now create a decision tree for the diagnose of the health risk. Do not forget to add symptoms to the decision tree.

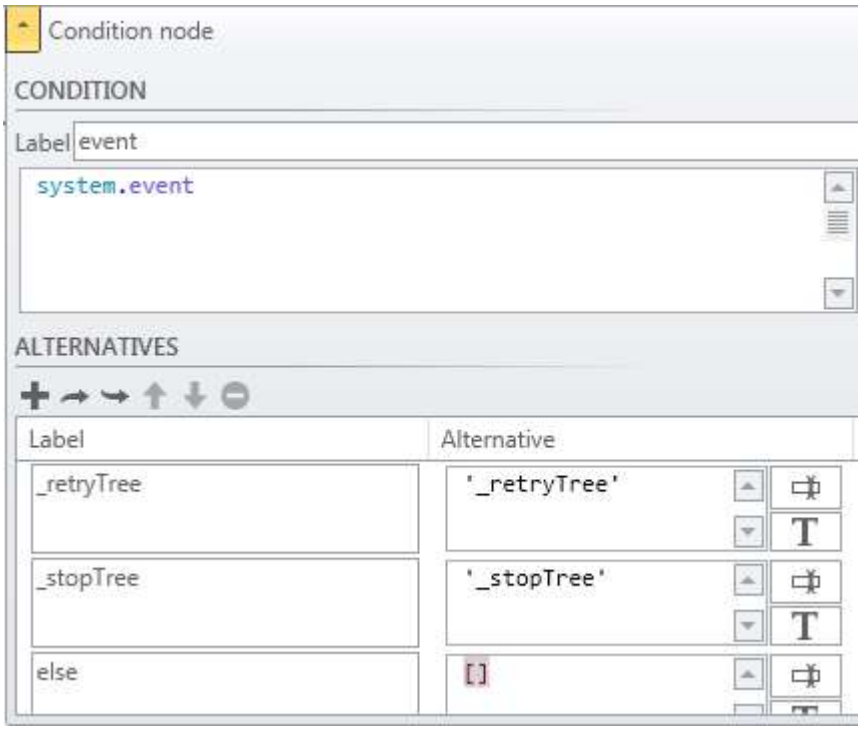
   The attributes for the BMI classification and the abdominal girth classification should both be in question nodes. The attribute for the health risk will be in the action nodes (leafs).

5. Create a container of type AQ_DecisionTreeEvaluator and a container of type AQ_DecisionTreeSelector. Also create an attribute that represents the search field and all necessary buttons.



6. Create a sub flow and an advice page. Add the containers you have just created to this page and add the page to the flow.

7. Create a button and event to start the advice flow and add this button to the container with lifestyle details. Add the sub flow to the main flow and make sure the navigation works correctly.



8. Test the advice page in the Runtime.

A tree to advice the customer about types of treatment

Which type of treatment a customer needs, depends on the BMI, abdominal girth and comorbidity.

BMI class	Abdominal girth		Comorbidity
	Normal	Big	
Overweight			
Level 1 obesity			
Level 2 obesity			
Level 3 obesity			

Advice on lifestyle and healthy food
Diet, physical activity and psychological intervention
Diet, physical activity, psychological intervention and possibly medication
Diet, physical activity, psychological intervention, possibly medication and possibly surgery

A customer has comorbidity if he or she has one or more of the following medical conditions: diabetes mellitus type 2, hypertension, cardiovascular condition, dyslipidemia, osteoarthritis or sleep apnoea.

Which type of treatment is adviced depends on which type is needed and the customer's preference. A customer could have a preference for medication or surgery.

Medication or surgery can only be considered if the customer has tried combined lifestyle interventions (diet, physical activity and psychological intervention) for at least a year and if the customer's weight loss is less than 5%.

1. Create a new entity Treatment and a new relation from Customer to Treatment. A customer has one treatment and a treatment belongs to one customer.

2. Now outline a diagram for the decision tree with questions, answers and conclusions.

   Which new attributes do you need to determine the treatment advice? Create these attributes.

3. Now create the decision tree for the treatment advice. Do not forget to add symptoms to the decision tree.

4. If the customer does not have a treatment yet, create a new treatment and add this service call to the sub flow. It the customer has a treatment, activate it.

5. Test the decision tree in the runtime.

A tree to advice the customer about a diet

Which diet is the best for a customer depends on a few factors. What is the goal of the customer? Long or short term weight loss? How much is the customer willing to spend? And is the amount of calories the diet subscribes more than the minimum amount of calories the customer needs?

Diet	Goal	Calories	Costs
Energy-restricted diet	Long term	1400	Low
Low caloric diet (LCD)	Short term	1100	High
Low fat diet	Long/short term	variable	Low
Very low caloric diet (VLCD)	Short term	600	High
High-protein, low-carbohydrate diet	Long term	1600	Low

The minimum amount of calories the customer needs is determined by the Average energy need (AEN) – 700 calories.

The AEN is calculated with the following formula:

Mainly sitting:	BMR + 20%
Little active (sports 1-3 times a week):	BMR + 30%
Average active (30 min of movement daily):	BMR + 40%
Very active (sports almost every day):	BMR + 50%
Extra active (heavy physical work daily):	BMR + 60%

The Basal metabolic rate (BMR) is calculated with this formula:
For men:
66 + (weight(kg) x 13,7) + (length(cm) x 5) - (age x 6,8)
For women:
655 + (weight(kg) x 9,6) + (length(cm) x 1,7) - (age x 4,7)

1. Now outline a diagram for the decision tree with questions, answers and conclusions.

   Which new attributes do you need to determine the diet advice? Create these attributes.

2. Now create the decision tree for the diet advice. Do not forget to add symptoms to the decision tree.

3. Now test the decision tree in the runtime.