pillar 1

Risk Management meets Open Source

RiskAnalytics 1.0

26 May 2011

Version: 1.0.207

The `RiskAnalytics` source code and the example code presented in this book are available at
`https://svn.intuitive-collaboration.com/RiskAnalytics/trunk/RiskAnalyticsPC`

This document has been written by various `PillarOne` core team members:

- Jon Bardola, FS-Consulta GmbH
- Jörg Dittrich, Munich Re
- Benjamin Ginsberg, Intuitive Collaboration AG
- Fouad Jaada, Intuitive Collaboration AG
- Stephan Hartmann, Munich Re
- Stefan Kunz, Intuitive Collaboration AG
- Markus Meier, Intuitive Collaboration AG
- Martin Melchior, Fachhochschule Nordwestschweiz
- Michael Spahn, Intuitive Collaboration AG
- Markus Stricker, Intuitive Collaboration AG
- Jessika Walter, Intuitive Collaboration AG

The `PillarOne` project was initiated and sponsered by Munich Re.

For further details please get in touch: contact@pillarone.org

Contents

Part I

User Guide

Chapter 1

Introduction

Solvency II can be seen as a driver for `RiskAnalytics`, but it is certainly not the only one. In general, it is the trend towards embedding the quantitative output of actuarial and risk management models in operative processes. This requires more than just correct calculations. Issues that are becoming more important are: ‘where did the input data come from?’, ‘who owns the data and who has the right to edit the data – or to sign it off?’, ‘how do we get the input data in an operationally safe way into the modeling tools?’, ‘how do we get the output data out of the modeling tool for reporting?’, ’is the used version documented?’, ‘can an auditor or a regulator review the complete solution efficiently?’, etc. In short, we forsee that actuarial and risk management applications will have to reach the same level of reliability, integration and security as financial applications.[1]

Most actuarial modeling tools cover only the quantitative aspects of actuarial models. `RiskAnalytics` strives to provide a sound base for a more complete risk management or Solvency II solution. The quantitative aspects of the Solvency II framework – ‘Pillar One’ – gave the software suite its name. But `PillarOne.RiskAnalytics`, or in short `RiskAnalytics`, is more than just an actuarial calculation engine. Auditability, security and process support, which are necessary for ‘Pillar Two’ in the Solvency II framework are also part of `RiskAnalytics`.

[1]Not long ago, the financial statements of a group of companies was consolidated using spreadsheets and copy-pasting information from back and forth. While many risk management applications still rely on this approach, nobody could imagine not using professional consolidation software these days.

'Pillar Three' of the Solvency II framework involves disclosure and transparency, which is related to reporting standards. The calculation engine of `RiskAnalytics` can provide the data for internal as well as external reporting, using industry-standard interfaces for professional reporting.

`PillarOne` was initiated at the end of 2007 and sponsored by Munich Re. Apart from Munich Re, Intuitive Collaboration and Canoo provided major resources for the developement of the software.

In a nutshell, `PillarOne` can be characterized by

- **Transparency** is a major value in risk management. `PillarOne` provides the ultimate transparency: all methods and implementations are licensed under an open source licence (GPL v3) which guarantees that anybody can get unrestricted access to the descriptions of the used methods and their implementations. Anybody is allowed to change or extend the implementation. The only thing which the GPL license forbids is to sell the whole or parts of the source code or to wrap it in a product with a commercial license.

- **IT Standards** are virtually nonexistent in most actuarial tools. `PillarOne` is a welcome exception since it is built on top of broadly accepted Java enterprise software components for database handling, client-server communication, security, etc. In short, `RiskAnalytics` is a true enterprise application.

- **Flexibility** is required for a platform to cover a broad spectrum of applications. `RiskAnalytics` makes no assumptions about the time resolution of models, the level of detail to be modeled, or which output data will be collected. As a result, `RiskAnalytics` can be used for a broad spectrum of quantitative insurance models, including risk capital or Solvency II models, reinsurance optimization, portfolio or deal valuations, profit testing . . . to mention a few.

Beyond the non-functional, or architectural, requirements mentioned above, `RiskAnalytics` offers a number of cool usability (and other) features. We mention a few below, with links to further elaborations for those with whetted appetites.

- **Compare**: The user can simultaneously textually compare two or more simulation results[2] and/or their parametrizations[3].

  Compare Graphics: The user can compare results graphically within a given result set[4]. Smoothing algorithms are provided.

- **Seamless clipboard integration**: Clicking on the top left cell of a result section selects all of its data, which can then be copied to the clipboard and pasted to a spreadsheet application with standard menu commands or keyboard shortcuts[5].

- **Dockable tabs**: Within the application window, multiple tabs can be open simultaneously, but only one tab is active at a time. Dockable tabs allow each tab to be undocked into its own window, or subsequently returned into the main application window, thus enabling the user to view and interact with multiple aspects of the modeling domain in parallel – for example, to compare or copy data side-by-side; or to open the comments-window separately on the screen wherever you like.

- **Validators**: Custom 'callback' functions implementing buisiness-specific 'rules' can be written and easily pushed/deployed to adhere to enterprise policies or to enforce data integrity at the time of entry.

- **Comments**

- **Multi Company Model (MCM)**

- **Batch runs**

- **Views**

-

-

-

[2]compare simulation results by first left-clicking them while pressing the Ctrl key to select them, next right-clicking anywhere in the shaded selection to invoke the context menu, and, finally, clicking on the compare option

[3]by clicking on the compare parametrizations option from within the result comparison

[4]compare (claims) distributions is invoked in a similar fasion

[5]e. g., right-clicking for the context menu, using the Edit menu, or using the keyboard shortcuts Ctrl-c and Ctrl-v directly to copy and paste, respectively

Chapter 2

Getting Started

Following the instructions in this chapter gets you from a to z within a short time. But only from little a to little z – the remaining chapters cover the A to Z for `RiskAnalytics`. To stick with the illustration, `RiskAnalytics` knows many different font types, even more than openly available. In this manual, however, we are able to cover only a few. Please get in touch for more details about more fonts, i.e. how to use `RiskAnalytics` in a variety of environments such as solvency and pricing, life and non-life, direct and reinsurance, one- or multi-period models: contact@pillarone.org

`RiskAnalytics` comes in two flavors: A stand-alone version which can be installed on a laptop or PC and a server version which is required for an enterprise solution. Since `PillarOne` is based on Java technology, it runs on Windows, Unix/linux and MacOS X. In the client-server version, the client can run a different OS than the server.

Laptops have become fairly powerful, but their disks are significantly slower than those on servers. Hence, we recommend the standalone version for evaluation, for development and for anybody who works more or less alone on a modelling project or has no access to a server infrastructure, e. g. consultants on the road.

If you just want to have a first glimps at `RiskAnalytics`, then you can also play with the client-server version on our test environment on pillarone.org/pillarone/try-it-online.[1]

[1] Be aware that this is a test environment; we may restart the server, clean the database and restrict the number of users or the number of iterations per user.

2.1 Installing the standalone version

The standalone version is suitable to be installed on a laptop or desktop computer with at least 1 GB of RAM and several GB of harddisk space – needed for handling the results of your simulations properly including all details as per your specification. It comes in an absolutely self-contained installer and, apart from the typical client-server components, it is identical to the client-server version.
The following steps describe how a standalone version can be installed.

1. Download the latest version from pillarone.org.
   Note: *If you have an earlier version of* `RiskAnalytics` *installed, you will need to update (uninstall and reinstall) some plugins... If, however, you will no longer need the parameters and results from the earlier installation, you might want to simply and completely uninstall the earlier installation and rightafter install the new version.*

2. Install

2.1.1 Database environments

In the standard setup, `PillarOneRiskAnalytics`comes with an already up and running database. No customization needed. Just use it.
However, if you want to use another database, the following gives a first introduction: By default there is support for two different embedded databases: **mysql** (recommended) and **derby**. Which one will be used depends on the script used to start risk analytics. These databases are started and stopped together with RiskAnalytics, which means that it is not possible to access the database externally when the application is closed.
If required it is also possible to run the application with an installed MySQL database (version 5.1 or newer is required). The database must be accessible at `localhost:3306` (3306 being the standard port). A database called `p1rat` must be created as well as user with name and password `p1rat` with the necessary privileges for the before mentioned table.
The database and the user can be setup with the following commands:

```
create database p1rat;
create user 'p1rat'@'localhost' identified by 'p1rat';
```

grant all on table p1rat.* to 'p1rat'@'localhost';
grant file on *.* to 'p1rat'@'localhost';
This would be sufficient to use `RiskAnalytics` with a standalone mysql, however for acceptable performance a script which will setup partitioning and indices should be executed. It can be downloaded at https://svn.intuitive-collaboration.com/RiskAnalytics/trunk/RiskAnalytics/src/java/mysql.sql
The database is now ready for Risk Analytics. To use this database with Risk Analytics it is necessary to edit the start script RunRiskAnalyticsMySql.cmd and replace `-Dgrails.env=mysqlembedded` with `-Dgrails.env=mysql`

2.1.2 Reset the database

To discard all changed data including results just remove the database directory in the pillarone temporary directory, which is `~/.pillarone` by default, but can be changed during the installation process. If not using the embedded mysql, but an installed one, just re-run the script used to initialize the database.

2.1.3 Accessing results directly

If you want to access the results directly they are saved in the table `single_value_result`. Only mysql can be accessed from outside the application (because derby runs in the same JVM). Keep in mind that when you want to access the results from the embedded mysql database (which runs on port 3307), `RiskAnalytics` must be running.

2.2 Installing the server version

This will give you a much more powerful set-up, but we recommend that you do this only if you have ample experience in dealing with server based installations or if you have a test server at your disposition.
Prerequisits:

- A servlet container, e. g. Tomcat. No need for a fully fledged middleware.
- A database, e. g. mysql. Note that you may have to edit the datasource information in the file DataSource.groovy[2].

[2]Groovy is an open-source scripting language based on Java. See their homepage

2.3 Your first simulation

Start `RiskAnalytics` by either opening the sandbox model from our server or the already installed version (as explained above). The releases include some demo models and we will use one of them now to verify that the installation is properly working. In the left pane, expand the Podra model and you should see the three subitems: Parameters, Result templates and Results.
Screenshot[3]
Open the parameters section, right-click on a parameterization that you want to base your simulation on, and then select run simulation. This opens the simulation tab and sets the parametrization from which you launched this view. You now want either to keep the suggested result template or make your own choice. For the first test run, entering a value between 100 and 1000 in the number of iterations will be an appropriate choice before clicking on Run.
Screenshot[4]
Now lets have a look at the result section. Open the result of the most recent simulation. There are many ways to look at the data: tables, graphics, comparison of results. You will find out more on your personal excursion through your first simulation.
Screenshot[5]

2.4 A mini result analysis: MRA

*If you want to have a look at the tiniest – still meaningful – model possible, we got something for you: **MRA**. Consisting of hard-wired elements such as: one underwriting segment, one claim generator for attritional claims and one proportional reinsurance contract.*[6]
bls bls blup

or [10] for more information.

[3]TODO: of the application as it opens

[4]TODO: of the simulation page

[5]TODO: result analysis context menu, one or two examples

[6]TODO: We need such a model and description

Chapter 3

The command line

3.1 Overview

There is a way to run simulations without using the user interface. The command line application offers the same simulation settings as the UI and the results are saved to the same database, which means that the results can later be viewed in the application (if the command line is run with the same environment).

3.2 Starting the Command Line Interface (CLI)

The command line application is located in the RiskAnalytics subfolder of the installation directory. The general syntax is: java [JAVA-OPTIONS] -jar org.pillarone.riskanalytics.core.cli.RunSimulation.jar [CLI-OPTIONS]

3.2.1 JAVA-OPTIONS

The following Java options should be set:

- `-Xmx1024m` Sets the maximum heap space to 1024MB. Depending on the simulation options, it should also run with less memory, but 1024M should be enough for everything.

- `-XX:MaxPermSize=256M` Sets the PermGen memory size to 256m.

- `-Dgrails.env=environment` Sets the grails environment within which the simulation is run. This mainly defines where the results are saved to. Possible options are: `mysqlembedded`: Uses a mysql database which is started before, and stopped after the simulation. `mysql`: Uses a already running mysql instance (must run on localhost:3306, db: p1rat, user: p1rat, pw: p1rat). `standalone`: Uses an embedded derby database.

3.2.2 CLI-OPTIONS

The CLI-OPTIONS define the simulation input parameters (the equivalent of the simulation configuration page)

- `-parameterization <path>`
  `<path>` is the parameterization file which should be used for the simulation

- `-resultConfiguration <path>`
  `<path>` is the result configuration file which should be used for the simulation. The model class must be the same as the parameterization file.

- `-force` (optional)
  By default the parameterization and the result template is not imported if one with the same name already exists. Use this option to force the import of the files.

- `-iterations <number>`
  The number of iterations to run.

- `-name <name>`
  The simulation name.

- `-comment <comment>` (optional)
  A comment for the simulation run.

- `-seed <number>` (optional)
  The random number generator seed to use for this simulation.

- `[-dboutput | -fileoutput <file> | -nooutput]`
  One of these options must be given. `-dboutput` saves the results to the DB defined by the given grails environment. `-fileoutput` saves the results to a file and `-nooutput` does not save any results at all.

Chapter 4

The user interface

When opening the `RiskAnalytics` application the user interface appears as depicted in Figure 4.1. The user interface is aranged in different functional areas that are described in Sections 4.1–4.5.

4.1 Menu

4.1.1 File menu

The file menu (cf. Figure 4.2) is used for all regards concerning parametrization files.
The following commands are available in this menu:

- Run simulation ... As soon as a parameter file of any model is opened this command becomes available and starts the dialogue to run a simulation.
- Refresh reads the current state of the underlaying database and displays all stored model information in the navigation pane (cf. Section 4.4).
- Save When an open parametrization or result template is changed this command becomes enabled in order to save the current modifications. The standard shortcut STRG + S can also be used instead.
- Save All works as the proir command except that it saves all modified parametrizations or result templates.

- Export all Parametrizations (newest version) to folder Since `RiskAnalytics` can handle different versions of a parametrization this commant can be used for only exporting the latest version to a specified folder.

- Export all Parametrizations to folder In contrast to the previous function all versions of all parametrizations are exported here.

- Import all Parametrizations from folder This function can be used in order to import all parametrizations from a specified folder.

4.1.2 Window menu

Without any data opened the Window menu (cf. Figure 4.3) only contains the menu entry Settings where the language of the user interface can be changed. If parametrizations, result templates or results are opened the according model is also listed in this menu. Since the data pane only shows data that belongs to one model this helps to keep track of the data that is currrently worked on if several models are involved.

4.1.3 Help menu

The menu Help (cf. Figure 4.4) contains the entry About RiskAnalytics where you can get information about the version, license, credits, used libraries and system properties.

4.2 Shortcuts

The shortcuts bar contains the most important functions of the File menu namely

- Refresh

- Save

- Run Simulation ...

The functions are explained in detail in Section 4.1.1.

4.3 Frame selection pane

This part of the user interface helps hiding panes that are currently not needed in order to maximize the space on the screen for the relevant panes. In its initial state only the navigation pane (cf. Section 4.4) can be hidden. When using data validation and comments on the data more options become available in this pane.

4.4 Navigation pane

In the navigation pane all the data processed in `RiskAnalytics` is structured in a tree. The tree's top layer contains all available models (e. g. Podra) as well as a node called Batches that will be explained later in this section. The logical structure within all different models is identical thus, when expanding a model by pressing the plus sign next to its name, the following folders become visible:

- Parametrization
- Result templates
- Results

Since `RiskAnalytics` is based on the data-driven-modelling paradigm a parametrization does not only contain pure data but also parts of the model structure itself. The model itself provides the components that generally can be proccessed during a simulation run such as **claims generators** or **underwriting segments**. The parametrization determines the number of the respective components as well as the actual numerical and type values of the components.
In contrast to the parametrization the result templates are structure-wise fully determined by the model itself.

4.4.1 Parametrization

When expanding the Parametrization folder all parametrizations that are available for this model are listed at this place. In some cases a plus sign is visible next to the parametrization name. This indicates that different versions of a model are available. Right-clicking on the folder Parametrization opens a context menu
In following fuctions are available:

- Export All cf. file menu.
- Export All (newest versions) cf. file menu.
- Import Select a parametrization file and import it into the database. If a parametrization is identical to a parametetrization that already exists in the database no new version is created.
- Import (force) Identical to the previous command except that the parametrization is imported at any case. If an identical parmanetrization already exists a new version with identical values is generated.
- Import all from folder This command imports all parametrization files from a specified directory into the respective model.
- Run simulation cf. file menu.
- Create default parametrization creates the empty default parametrization of the current model.
- Delete all deletes all parametrizations of the current model.

Some operations are only available for specific parametrizations. Thus, the options in the context menu slightly change when right-clicking a specific paramentrization (cf. Figure 4.6).
In following fuctions are available:

- Open opens the paramerization in the data pane.
- Delete removes the selected parametrization from the database. Note that a parametrization can only be deleted if no result has already been generated with it. Other wise you will be asked if all results based on this parametrization should also be deleted.
- Export cf. file menu.
- Rename opens a dialogue in order to change the name of the parametrization and all depending versions.
- Run simulation cf. file menu.
- Save as saves the current parametrizations under a different parametrization name.

- Create new version If a specific version of the parametrization should be kept, a new version can be generated keeping the old on in its current state. If the user wants to open a used parametrization either it can be opened in read only mode or opened by generating a new version that is editable again.

- Compare If two or more parametrizations should be compared they can be selected using the CTRL key. Then the compare option becomes enabled. The lines in the parametrizations that differ in structure or in numerical values are highlighted.

4.4.2 Result templates

The resulte template section is very similarly structured as the parametrizations section. Result templates are used in order to determine in dependently from a parametrization what results should be collected during the simulation and to what granularity. Result templates provide an easy way to get comparable results for different parametrizations. The use of templates guarantees that the same type of single result values are collected without having to specifying it again and again for different parametrizations.
All available options for dealing with result templates can be seen in Figure 4.7 that shows the according context menu. The functions are analogous to the ones of parametrization that are described in Section 4.4.1.
The functions that are available in the context menu for a specific result template are identical as the one for a parametrization thus we refer to Section 4.4.1 for explanations.

4.4.3 Results

4.5 Data pane

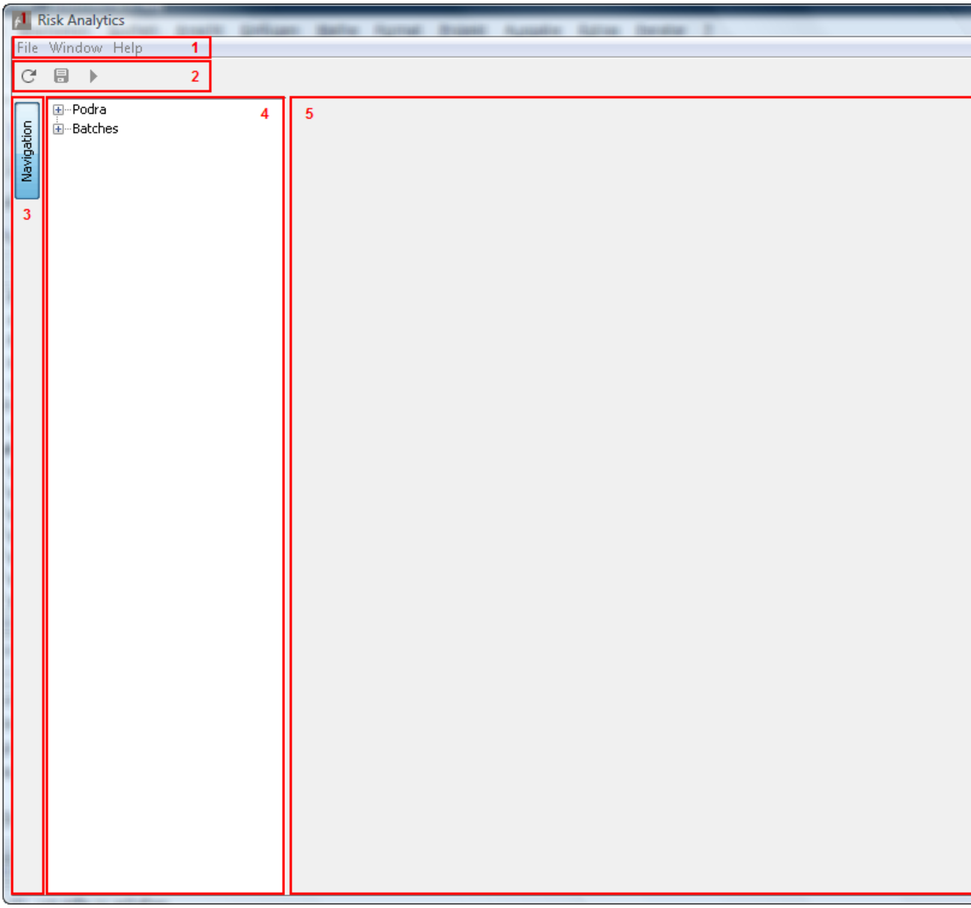


Figure 4.1: Areas of the user interface

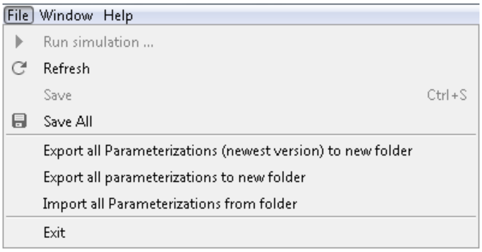

File Window Help
Run simulation ...
Refresh
Save Ctrl+S
Save All
Export all Parameterizations (newest version) to new folder
Export all parameterizations to new folder
Import all Parameterizations from folder
Exit

Figure 4.2: Expanded file menu.

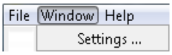


Figure 4.3: Expanded window menu.

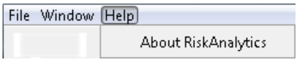


Figure 4.4: Expanded help menu.

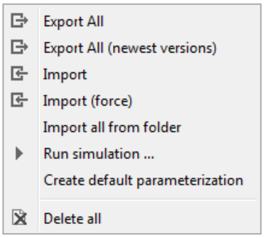


Figure 4.5: Context menu attached to the Parametrizations folder.

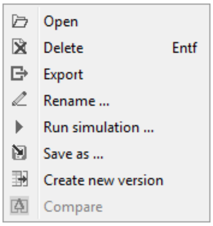


Figure 4.6: Context menu attached to a specific single parametrization.

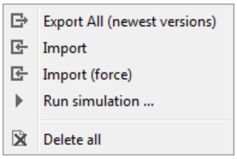


Figure 4.7: Context menu attached to the Result templates folder.

Chapter 5

A partial internal model for a non-life insurance company: Podra

The Podra Model[1] serves two purposes. First, it is a fairly powerful non-life, primary insurance risk model which can be used as a partial internal Solvency II model. Second, it can be used as a starting point for developing custom models – a topic which is taken up in more advanced sections. The Podra model is a typical representative of a dynamically extendable model (see the classification of models in *REF TO DO*[2]). We will use it here for two reasons. Firstly, because we can build a custom model with a reasonable, pre-defined structure laid out by an expert, without any programming. And secondly, because it is a fully published model – i. e., one can reproduce each of the steps below after downloading the release – making the first section below an effective tutorial.

Its components are the gross portfolio (underwriting information and claims generators), the company structure and the reinsurance covers. Its output provides a good basis to analyize capital at risk, premium, provisions and claims on different levels. Possible uses are risk analyses, portfolio optimization and reinsurance structuring.

Please refer to a workshop held in September 2010 in St.Gallen for

[1]`PillarOne` Dynamic Reinsurance Analysis[3]

[2]TODO:

illustrative examples on pillarone.org

Note: *The Podra model is a data-driven model which can be easily extended by simply adding input data for new claims generators, underwriting segments, etc. The price for this flexibility is performance. If this flexibility is not needed, then there are ways to build models with the same business logic which have about half the runtime of an equivalent dynamic model.*

5.1 Step by step

The first step is to open the application and select the Podra model. Within the Podra model, we wish to start from scratch. Hence we use the context menu within Podra on ‘Parametrisations’ (right-mouse-click) to create a new default parametrisation. You will find some more parametrisations within Podra/Parametrisations as examples. *Screenshot*[3] Afterwards, name the newly created parametrisation and open it via the context menu of the parametrisation itself.

In the right pane the (empty) model is shown.

The available segments that can be entered into the model are Underwriting, Claims Generators, Correlations, Lines of Business and Reinsurance Program.

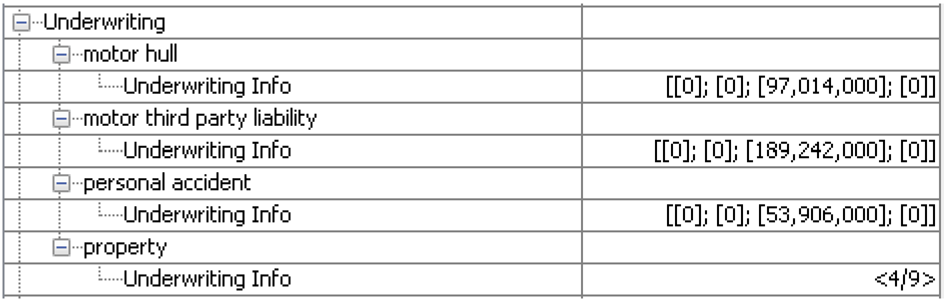
Underwriting Information As a first step the underwriting information should be feeded into the model. Within the Underwriting context menu there is a button add. Name the new Underwriting segment. Within the segment, the data can be edited by doubleclick.

The underwriting information contains: maximum sum insured, average sum insured, premium and number of policies. It is possible to edit serveral risk bands (they will be used for surplus share treaty reinsurance). Add as many Underwriting segments as you like.

Underwriting information is optional to some extent and can be either attached to a claims generator (see below) where it can be used for skaling of distributions or as exposure information for surplus reinsurance contracts; or it can be attached to a Line of Business *where it is used to*[4].

[3]TODO: example parameterizations

[4]TODO: please explain



Underwriting	
motor hull	
Underwriting Info	[[0]; [0]; [97,014,000]; [0]]
motor third party liability	
Underwriting Info	[[0]; [0]; [189,242,000]; [0]]
personal accident	
Underwriting Info	[[0]; [0]; [53,906,000]; [0]]
property	
Underwriting Info	<4/9>

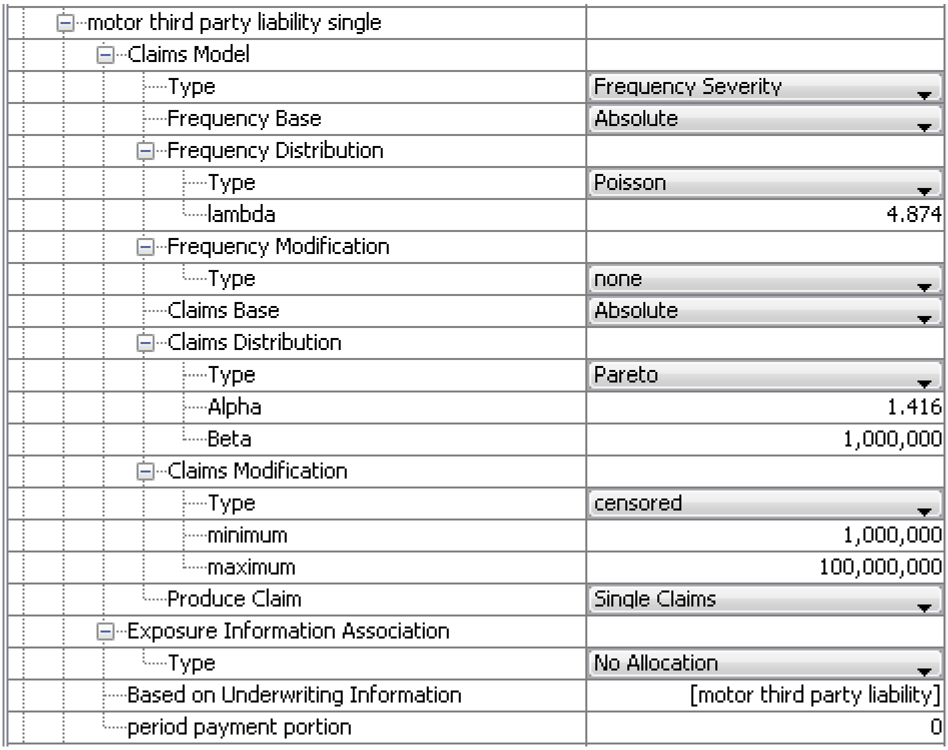
Figure 5.1: Podra underwriting information example

Claims Generators The next step is the editing of Claims Generators. Context menu add creates a new claims generator. Compared to the underwriting information a claims generator is more complex. It contains the claims model, the exposure information association, and the underlying underwiting information.
For the claims generator the type of the claims generator can be selected: Attritional, Frequency Severity and many more. Depending on the selection there are different ways to define the claims generator appropriately. The model starts with drawing a claim from the selected claims generator. If a surplus share treaty will be contained in the reinsurance program, the second step is the allocation of an exposure (sum insured, pml, eml) to the drawn loss. Therefore the option to selecting the exposure information allocation can be used.
The combo box *'claims generator is based on'*[5] is used to scale the random distributions by underwriting information from the underwriting segments. If no underwriting is provided or applicable, a given fixed value **absolute** van be selected.

Correlations Claims generators may interact, or, in other words, correlate. Correlations can be introduced by adding a Correlations Matrix using the Context Menu Add of **dependencies** on the right pane.

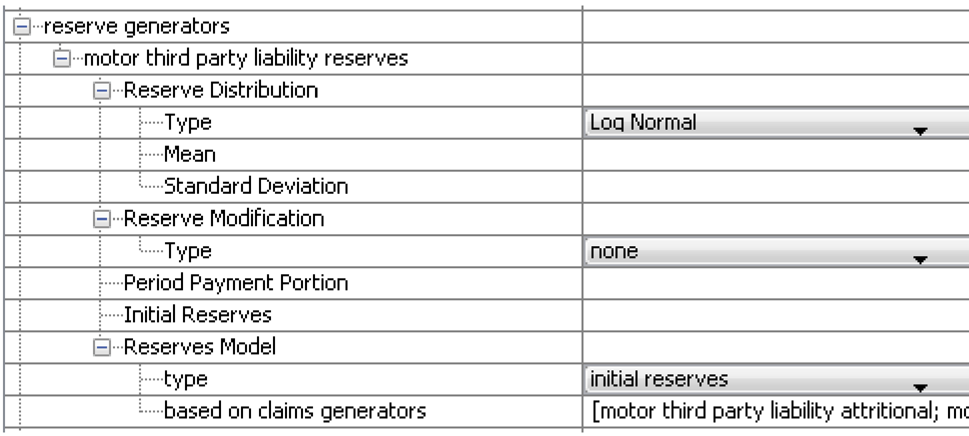
Reserve generators The reserve generators can be introduced like claims generators. The reserve generators component allows for adding reserve generators for a certain segmentation. It contains the reserve

[5]TODO: check name



motor third party liability single	
Claims Model	
Type	Frequency Severity
Frequency Base	Absolute
Frequency Distribution	
Type	Poisson
lambda	4.874
Frequency Modification	
Type	none
Claims Base	Absolute
Claims Distribution	
Type	Pareto
Alpha	1.416
Beta	1,000,000
Claims Modification	
Type	censored
minimum	1,000,000
maximum	100,000,000
Produce Claim	Single Claims
Exposure Information Association	
Type	No Allocation
Based on Underwriting Information	[motor third party liability]
period payment portion	0

Figure 5.2: Podra claims generator example



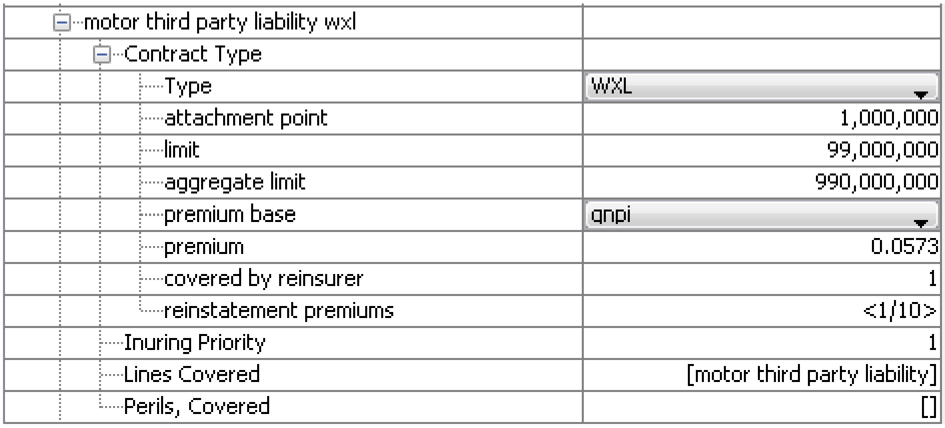
reserve generators	
motor third party liability reserves	
Reserve Distribution	
Type	Log Normal
Mean	
Standard Deviation	
Reserve Modification	
Type	none
Period Payment Portion	
Initial Reserves	
Reserves Model	
type	initial reserves
based on claims generators	[motor third party liability attritional; mo

Figure 5.3: Podra reserve generator example

development (paid plus reserved) distribution possibly with modifiers. Additionally the recent year payment portion can be specified to allow for very simple reserve development modelling. Furthermore the initial reserves can be specified directly in this component. Within a multi period model the outgoing reserves of claims generators can be linked to the reserve component, resulting in increasing the reserve volume of future periods by the amount of the outgoing reserve of the preceding period.

Lines of Business Now the risks are defined and may be combined or allocated on business segments. These are defined in the Line of Business area. Here again, it is possible to add another Line of Business. The next step is to add different Claims generators (or shares of them) to the dedicated Line of Business. Additionally, to each Line of Business the Underwriting Information is attached, so we can select this additionally.

Reinsurance Program Individual Reinsurance covers can be added to the Reinsurance Program section. The reinsurance cover consists of a contract (including the type), the inuring priority (see further below), the covered lines of business, as well as the covered claims generators. The covered lines of business and the convered claims generators are sent to the reinsurance contract to be covered.



motor third party liability wxl	
Contract Type	
Type	WXL
attachment point	1,000,000
limit	99,000,000
aggregate limit	990,000,000
premium base	gnpi
premium	0.0573
covered by reinsurer	1
reinstatement premiums	<1/10>
Inuring Priority	1
Lines Covered	[motor third party liability]
Perils, Covered	[]

Figure 5.4: Podra reinsurance contract example

The inuring priority provides the information when a specific contract is used and from which level e. g. the GNPI is taken. For the reinsurance program itself the type has to be specified. Depending on the type there are several options for the necessary parameters.

Result configuration Usually, not all possible result variables are generated and stored during a simulation run. This is for performance and disk space reasons. Within the Podra model several result configurations can be defined. By opening a result descriptor in the right pane a tree view with the available output variables is shown. On each output variable a collection type can be selected. To make a result variable available in the result set, it should be collected as aggregated or drill down*ref to new section*[6]. This output variables will be found in the Results after a simulation has been done.
Screenshot[7]

Evaluation / Simulation On Parameters and Result Descriptors within the left tree view it is possible to start simulations using the context menu. The Parameter set and the Result Descriptor to be used have to be selected in the Simulation window on the right pane. The simulation run can be started after adding the number of simulatons

[6]TODO: please include ref
[7]TODO: Result Configuration

and optionally the random generator seed. The Results of the Podra model will contain your result set thereafter.
Open the result set to see which variables have been generated. On the result variables different risk measures (exactly different functionals) can be displayed by pressing the appropriate buttons. The diverse options for evaluations include expected value, standard deviation, value at risk and tail value at risk for a given level of *security*[8]. The full set – or any branch of the tree – can be copied to the clipboard by selecting the top element of the branch or tree respectively and pressing ctrl-c.
Screenshot[9]
Given the model has been evaluated twice with different parameters the results can be compared using the compare feature selectable in the treeview. Deviations in all result variables can be shown in absolute and relative form. Sensitivity tests of paramerters can be done easily using this feature.
Screenshot[10]

[8]TODO: confirm wording
[9]TODO: Evaluation - simulation
[10]TODO: Compare

Chapter 6

Non-life risk modelling for Insurance Groups

The "Multi Company Model" is an extension of the Podra model: It consists of the same components and hence may serve the same purposes for risk modelling of non-life business of direct insurers; its main qualification however is to picture the activities of a group of insurance companies, culminating in an additional component that allows for the assignment of the modelled technical structure to the predefined business structure. The motivation for providing the extended group model with `RiskAnalytics` goes back to statutory requirements pertaining to group solvency in addition to the regulatory provisions to which the individual insurance company must comply, and also from a different point of view namely neither a solo approach or nor a consolidated approach. For a more detailed discussion of group modeling and related references we point to Chapter 16.

Parametrization

In a first step, we open the application and select the **MultiCompany** model[1][2]. The steps for creating a new parametrization correspond to the outline in Chapter 5, for the sake of convenience however we re-

[1] The model has to be enabled in the Config.groovy file by adding it to the list of configuration variable *models* Add **multicompany** and restart `RiskAnalytics`.

[2] TODO: Where is the description for editing the Config.groovy file?

capitulate the procedure. Within the parametrization menu there are two tested examples called **Three Companies** and **LE-CRTI** (**l**egal **e**ntities and their **c**apital and **r**isk **t**ransfer **i**nstruments). that may be used for analysis purposes with or without changing/modifying/extending the default input parameters. As outlined for Podra, you may either create a new parametrization or import a .groovy parameter file by right-clicking on **Parameterization** and selecting the operation from the context menu. The newly created (or any other) parametrization is opened by left double-clicking on it or, as a second possibility, right-clicking and selecting the **Open** function in the context menu.
At this stage the model pops up in the right panel listing all the components coming with the model, compare Figure 6.1. The model is similar to Podra with slight extensions in the components **Lines of Business**, **Reinsurance Market** (corresponding to **Reinsurance** in Podra), and **ALM Generators** that are related to the additional component **Companies**; altogether turning Podra into a model suited for the analysis of group risks. Next, we shortly introduce the new component and outline the resulting modifications for the components already known from Podra.

New component Companies. This component allows to feed into the model a group of contractual partners of the business lines under consideration. A new company segment is added by right click on **Companies**, selecting the button **Add** and entering the name of the (new) insurance company into the pop-up window. Alternatively, an existing company may be duplicated under a new name by right click and selecting the button **Duplicate**.
As illustrated in Figure 6.1 within the company segment, an insurance rating value[3] may be selected for each company from a list with range between **AAA** and **D** and default value **No Default**.

Modified component Lines of Business. As in Podra we here define the business classes and relate them to the previously specified components **Underwriting**, **Claims Generators** and **Reserve Generators**. In contrast to Podra the single segments show an additional parameter **Company** allowing to attach to the dedicated line of business the associated insurance company by selection from the list

[3]At present the selected value has no impact on the simulation results.

of predefined companies.[4]. Note that the whole purpose of **Lines of Business** is to manage information that is obtained from other components, in other words, the information from other components is sent to the segments of **Lines of Business**, filtered according to appropriately defined rules and re-dispatched to the **Reinsurance Market** or/and result descriptors.

Modified component ALM Generators. The component **ALM Generators** is extended in the same way as **Lines of Business**: The associated company must be attached to the dedicated ALM segment by selection from the list of predefined companies.

Modified component Reinsurance/Reinsurance Market. Expectedly, segments of **Reinsurance Market** distinguish from segments of **Reinsurance** in Podra by an additional parameter **Reinsurers**. Here, the selection of one of the predefined companies is optional in contrast to the two aforementioned components. Moreover, as reinsurance treaties may be shared between multiple reinsurers, the user may enter as many companies as he wants (sensibly up to the number of insurances in component **Companies**) together with the signed share of the treaty. As usual, this kind of parameter may be filled by double-clicking in the cell attached to **Reinsurers**.

Result configuration

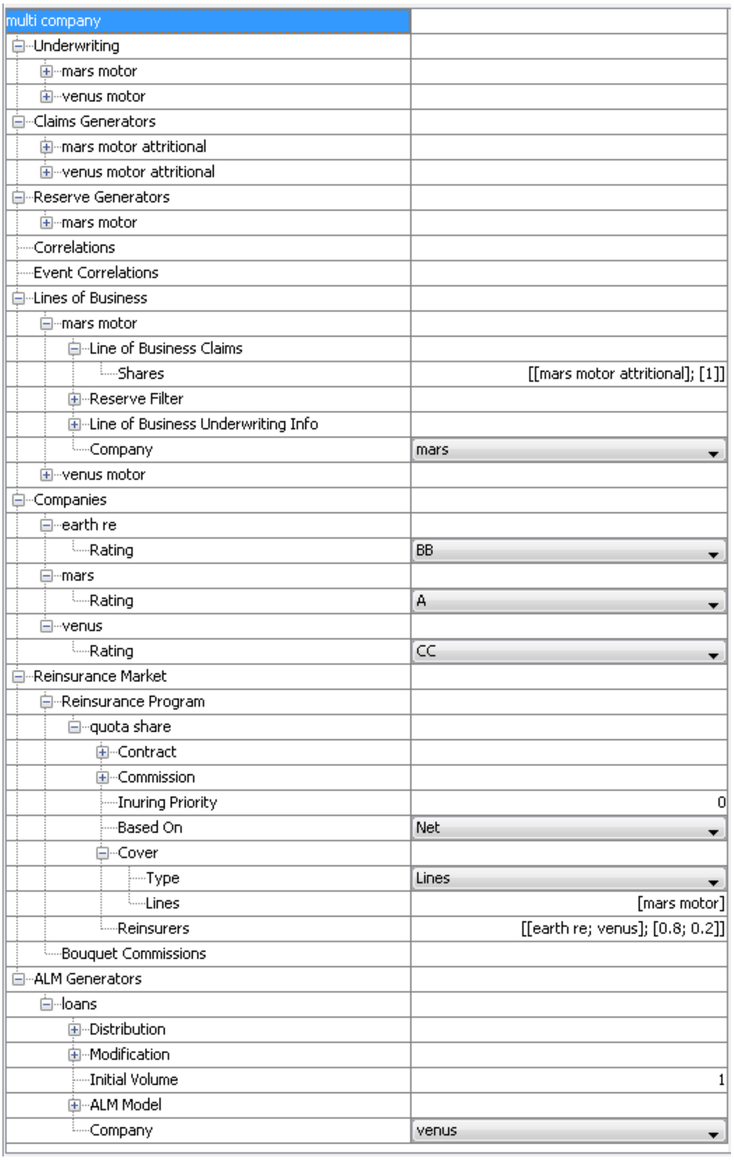
Due to performance and disk space reasons not all possible result variables are generated and stored during a simulation run. Analogous to Podra, within the "Multi Company Model" several result configurations can be defined. By opening a result descriptor (there is one predefined template named **Aggregate Overview 3**, right click on it and select open) a tree view with the available output variables is shown in the right pane. On each output variable a collection type can be selected. For a more detailed overview of various result descriptors we refer the reader to Chapter 8. Results that are specifically shown for the various company segments are described in full detail in Chapter 16.

[4]Note that the lines must be related to a predefined company unless the component **Companies** is kept empty, in which case **MultiCompany** is identical to Podra

Simulation/Evaluation

Simulations are started by right clicking on **Parameterization** or **Result templates**. Selecting **start simulation** in the appearing context menu the simulation window opens in the right panel. After selecting the dedicated parametrization, result template and number of iterations the simulation run can be started.
Open the result set to see which variables have been generated. On the result variables different risk measures can be added by pressing the appropriate buttons. The options for evaluations include expected value, standard deviation, value at risk and tail value at risk for a given level of ***Note:*** *check, please*security. Any branch of the result tree can be copied to the clipboard by selecting the top left cell of the branch and pressing Ctrl-c.
If the model has been evaluated more than once, with different parameters, the corresponding result sets can be viewed side-by-side using the **compare** feature selectable in the tree view. Deviations in all result variables can be shown in absolute and relative form. Sensitivity tests of parameters can be done easily using this feature; see Section 8.4.[5]

[5]TODO: Adjustment of bold font for terms in Parametrization.



multi company	
Underwriting	
mars motor	
venus motor	
Claims Generators	
mars motor attritional	
venus motor attritional	
Reserve Generators	
mars motor	
Correlations	
Event Correlations	
Lines of Business	
mars motor	
Line of Business Claims	
Shares	[[mars motor attritional]; [1]]
Reserve Filter	
Line of Business Underwriting Info	
Company	mars
venus motor	
Companies	
earth re	
Rating	BB
mars	
Rating	A
venus	
Rating	CC
Reinsurance Market	
Reinsurance Program	
quota share	
Contract	
Commission	
Inuring Priority	0
Based On	Net
Cover	
Type	Lines
Lines	[mars motor]
Reinsurers	[[earth re; venus]; [0.8; 0.2]]
Bouquet Commissions	
ALM Generators	
loans	
Distribution	
Modification	
Initial Volume	1
ALM Model	
Company	venus

Figure 6.1: Multi Company Model

Chapter 7

Life insurance cash flow model

The "Life insurance PillarOne cash flow model" is yet another of the public available models for `RiskAnalytics`. It allows modeling the needs of a life insurance office with respect to a portfolio of unit-linked life insurance contracts (policies) and a traditional financing life reinsurance treaty.
The major objective is to model the future cash flows for a life insurance company for pricing (profit testing), valuation and planning purposes (incl. embedded value calculations). The cash flows for the various stakeholders (policyholders, distribution channels, expenses, funds management, reinsurance, taxes, etc.) are calculated.
This chapter describes the functionalities of the PillarOne life insurance cash flow model from an end-user perspective. Throughout the chapter a concrete example is used to describe and illustrate the parameters, functionalities and results derived with `RiskAnalytics`.
With this open life insurance model, the following parameters can be described:

- Actual expenses (both company and product-based),
- reinsurance conditions,
- actual model points (existing portfolio) and future business,
- product parameters/characteristics,

- demographic assumptions (mortality, disability, lapses),
- economic and solvency assumptions,
- distribution channel commissions, and
- banking expenses.

Looking forward, `RiskAnalytics`is also enable to implement useful stochastic calculations for life insurance in an open available model.

7.1 Introduction

Based on the existing PillarOne software an extension (so-called "Life insurance PillarOne cash flow model") has been built to model the needs of a life insurance office with a portfolio of unit-linked life insurance contracts (policies) and a traditional financing life reinsurance treaty.
To model the life insurance cash flows with the various stakeholders the following elements need to be considered, as illustrated in figure 7.1.
At the very beginning there was a concrete project with the need to perform future cash flow projections of a life insurance company for pricing (profit testing), valuation and planning purposes (incl. embedded value calculations). The life insurance PillarOne application has then been developed with the following main objectives:

- Perform cash flow projections on a shorter (e.g. 3 years for planning purposes) as well as a longer (e.g. 20 years for embedded and appraisal value calculations) term horizon
- Consider an existing portfolio as well as model future new business
- Determine the shareholders' profitability for the modeled business
- Appropriately consider and analyze the reinsurance treaty (e.g. costs of financing, pay-back period)
- Apply profit testing techniques on life insurance products and analyzing LoBs

The components of the PillarOne life insurance calculations and workflow can be characterized with chart 7.2.
The building blocks are:

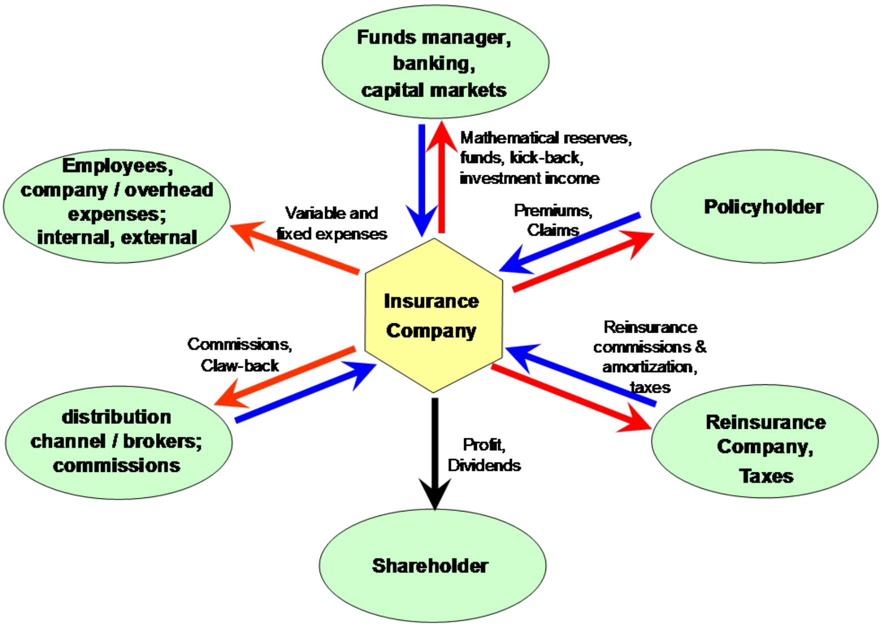

Funds manager, banking, capital markets
Mathematical reserves, funds, kick-back, investment income
Employees, company / overhead expenses; internal, external
Variable and fixed expenses
Premiums, Claims
Policyholder
Insurance Company
Commissions, Claw-back
Reinsurance commissions & amortization, taxes
distribution channel / brokers; commissions
Reinsurance Company, Taxes
Profit, Dividends
Shareholder

Figure 7.1: Stakeholders' view of Life Insurance

- Cash flow projection calculation engine for a simple life insurance policy
- Weighted by actual demographic and lapse experiences
- Aggregation on all policies on entire portfolio
- Consideration of reinsurance cash flows
- Storage of results in a database enabling to derive all kind of statistics (report-generator, key figures): EV and VIF, new business margin, IRR, break-even analysis before/after reinsurance, perform planning with future new business, sensitivities, etc.
- Consideration of an entire range of assumptions and conditions.

This chapter describes the functionalities of the PillarOne life insurance cash flow model from an end-user perspective. Throughout the chapter a concrete example is used to describe and illustrate the parameters,

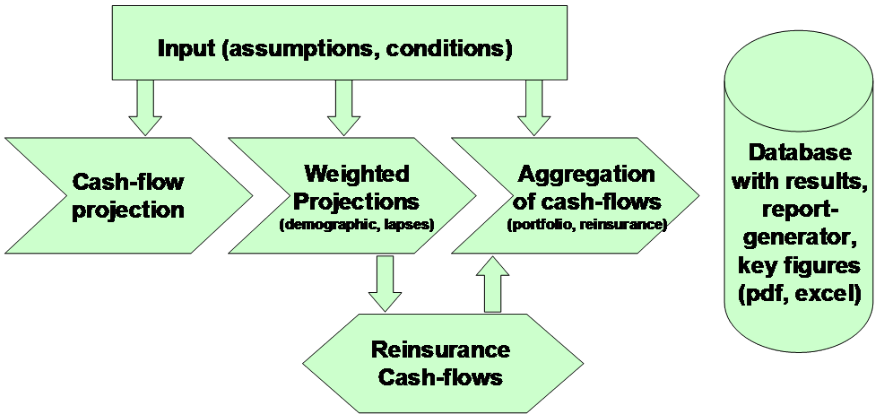

Input (assumptions, conditions)
Cash-flow projection
Weighted Projections (demographic, lapses)
Aggregation of cash-flows (portfolio, reinsurance)
Database with results, report-generator, key figures (pdf, excel)
Reinsurance Cash-flows

Figure 7.2: Life Insurance Workflow Calculations and Workflow

functionalities and results derived with the software. The GUI (graphic user interface) enables to enter a whole range of assumptions and conditions. With a well structured and modular end-user interface all the necessary input parameters can be captured.
This chapter of the manual is structured in the following way:

- In the next section 7.2 the (high-level) framework of the used example is presented.
- In subsequent sections the input, calculations and results/output are described in detail.
- At the end, in section 7.7, we will deal with potential future development of this life insurance PillarOne cash flow model.

This documentation is based on an illustrative example which includes the (direct) life insurance (a company with a portfolio of unit-linked policies) with an associated reinsurance treaty. If there is no interest in the reinsurance section, one could simply exclude the reinsurance by setting the quota share reinsurance parameters to 0. All the calculations and results are done and derived on a deterministic basis.
Please note: The example in this documentation should only be seen as an illustration. In other words: The parameters, content and result of this example cannot directly be used in the practical work without considering further input reflecting the company specific characteris-

tics, concrete products, appropriate demographic and economic environment, etc.

7.2 Example

When downloading and installing `RiskAnalytics` you directly get a preinstalled life example. In this chapter, we will discuss this example at length. We provide illustrative numbers only and you are free to change whatever you want.

7.2.1 Product characteristics

We will model three unit-linked products: Two periodic premium products which are designed for different commission payments (com5 and com4 for e.g. different distribution channels) and a single premium product. We will denominate the products with ULPPCOM5, ULPPCOM4 and ULSPCOM5. We will work only in CHF. However, the life model would also allow capturing segregated currency-pots to reflect different currencies by converting the data into CHF.
From the gross premium (premium paid by the policyholder) the expenses and charges are deducted. The remaining saving premium is then added to the actual funds value (saving or funds accumulation process). To describe the actual products (so-called pricing or "1st order" basis) we use the following parameters for the expense and cost structure:

- Acquisition expenses (alpha): 5% of the present values of the paid premiums, which are amortized over the first 10 policy years
- Collection expenses (beta): between 7% and 8% of the periodic premium
- Proportional administration expenses (gamma): 0.05% (for periodic premiums) or 0.25% (for single premium) p.a. of the funds value
- Fixed administration expenses (kappa): between CHF 50 to CHF 100 per policy p.a.
- Costs for mortality risk premium: Monthly calculated according to the sum at risk in case of death and the pricing mortality table according to age and gender of the policyholder

- Costs for waiver of premium: Monthly calculated according to the sum at risk in case of disability and the pricing waiver of premium table according to waiting period, age and gender of the policyholder
- Surrender charges: For the periodic premium products 100% of the funds value linearly falling to 0% of the funds value over the first 5 policy years.

The following risk benefits could be defined and set at policy level:

- Sum at risk in case of death could be defined as:
  - a fixed CHF-amount less the actual funds value and/or as
  - a percentage of the sum of the paid premiums.
- Waiver of premiums for the periodic premiums with various waiting periods (e.g. 6, 12 or 24 months).

The expense loadings and costs are either deducted from the (periodic) premium payments or from the funds of the policy. Similarly the mortality and disability costs ("1st order" risk premiums according to mortality and waiver of premium tables) are periodically deducted. The calculations are done on a monthly basis.

7.2.2 Distribution channel commissions

There are two commission payment types or "scales" (COM5 and COM4): Either 5% or 4% of the sum of premiums (total paid premiums over the entire policy duration, but the policy duration will be capped at a maximum of 25 years). The commissions are paid upfront. However, in case of policy surrender they have to be refunded ("claw-back") by the distribution channel (agent) linearly, in proportion within the first three policy years; i.e. only after three years the commissions are fully earned. A general shortfall of the claw-back payment of 2.5% is assumed. Finally, a portfolio commission of 0.05% p.a. of the mathematical reserves (funds value) will be paid out on a monthly basis.

7.2.3 Actual expenses

The actual expenses ("2nd order" parameters) could be defined on a company basis and/or on a product-related basis. In addition the expenses could also be linked to inflation. In our example we do not assume total company expenses (no expense over-run). We have actual initial expenses between CHF 200 and CHF 350 per policy. The actual administration expenses are varying between CHF 150 and CHF 225 p.a. The expenses are depending on the policy size (sum of total paid premiums).

7.2.4 Actual demographic assumptions

For the actual mortality ("2nd order") various, product-related probability tables could be defined and used. Similarly for the disability (waiver of premium) and the lapses actual probability tables are assumed in the example. In our example the actual mortality and disability probability rates are 80% of the pricing assumptions.

7.2.5 Reinsurance section

According to the reinsurance treaty conditions, common or general reinsurance parameters need to be defined for all the products:

- Various settlement dates and intervals
- Deposit interest rate of 0.25% and reinsurance profit participation of 90%
- Loss carried forward for all past underwriting years (amounts and interest rates) and for various currencies separately
- Reinsurance claw-back percentage of 100% and 'cost and risk annual supplement on ceded premium' of 2%, etc.

Product specific reinsurance parameters - which need to be entered - are:

- The currency (CHF) and reinsurance quota share percentage of 50%
- Reinsurance acquisition commission of either 5% or 4% of the sum of the reinsured premiums (total paid premiums to reinsurance company over the entire policy duration). However, the policy duration will be capped at a maximum of 25 years. We will have a (linear) amortization of the reinsurance acquisition commissions over the first eight years.

- Percentage (25%) of the reinsured beta and kappa expenses which the reinsurance company will pay (refund) to the insurance company.

7.2.6 Model point data

The actual portfolio and future sales can be defined:

- The actual portfolio is described policy by policy (policy #, product, annual premium, installments p.a. , gender, birth-date, policy inception date, duration, death benefits, waiver of premium, actual funds value/mathematical reserves, etc.)
- Similarly, model points (representatives) for future policy data (sales) can be defined.
- The future sales (number of policies) can be entered for these model points (representatives) on a monthly basis for the next 10 years, as from the projection start.

In our example the insurance company sold policies only during the year 2009 (i.e. we have a given portfolio as per 31.12.2009) and the company plans to sell future new business for the years 2010 to 2012. Our projection will start as per 1.1.2010 and will be done on a monthly basis for 254 months, i.e. for more than 20 years.
Finally, some global and economic parameters can be defined. We use in our example:

- Solvency I: 1% of the funds value, 0.3% of the sums at risk in case of death, and 20% of the disability risk premiums.
- Banking fees and income: An income ("kick-back") of 0.25% of the funds value (rate p.a. , but monthly calculated and accrued), custody expenses of 0.05% of the funds value (rate p.a. , monthly accrued), and transactions costs of 0.25% of the net buy-sell amount (monthly).
- Economic assumptions: Discount rate of 7.5%, risk free rate of 1.5%, investment return on funds of 5%, inflation on expenses of 1%, taxes on statutory result of 25%, initial shareholder capital (allocated to the specified portfolio and new business) of CHF 1 Mio.

Please note that the end-user could define and implement (parameterize) with this life insurance PillarOne model any number of products, expense types, demographic assumptions, etc. We simply limited this example for practical discussion purposes to three illustrative products, etc.
The above, high-level description of the example will accompany us during the subsequent sections and illustrate the various parameters and results (input and output) which we will discuss. When reading this end-user documentation it would probably be best to have the life insurance PillarOne model open on your PC with the given "standard" parameters of our example.

7.3 Input parameters

Various parameters need to be entered: Product descriptions, company and distribution channel information, actual portfolio and future business (planning) data, reinsurance descriptions, details on demographics and economics, etc. as illustrated in figure 7.3

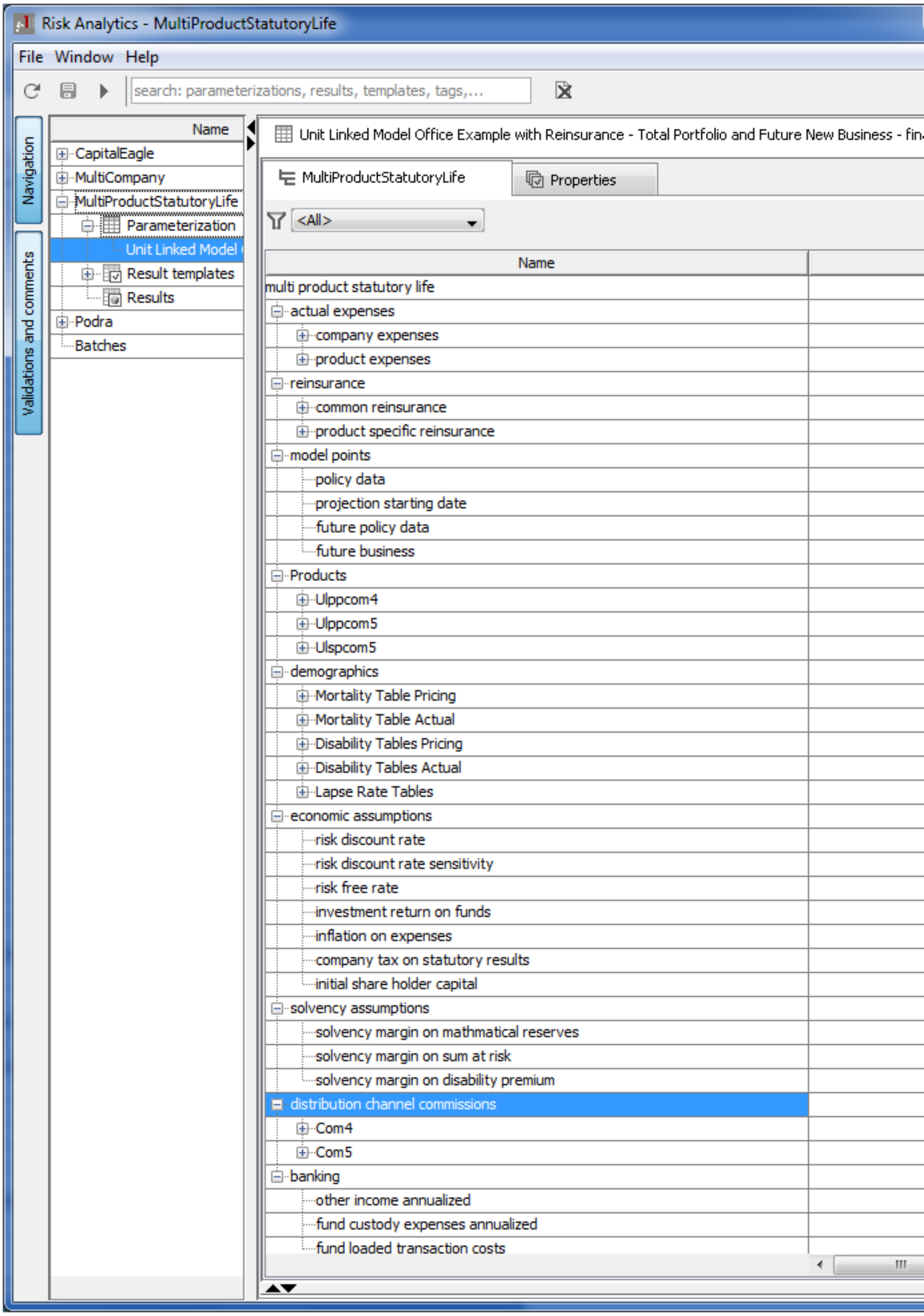
7.3.1 Actual expenses

The end-user can define two kinds of expenses: Overall company expenses to reflect the complete expense structure of the company (a.o. used in embedded value type of calculations) or product related expenses to reflect the cost-intensiveness of the different kind of products (e.g. used for profit-testing calculations). Of course one could also define both kinds of expenses in the same model (e.g. to capture the expenses-overrun).
Please note that all the expense definitions in this section are referring to actual expenses ("2nd order", outgoes) and do not necessarily need to be directly related to the "1st order" expenses (pricing, loadings) of the various products.

Company expenses

The company expenses are characterized by:

- "company overhead expenses": Have to be entered in an absolute amount (CHF) for every month of the envisaged projection period (40 years or 480 months). However, for our model example we set all the company overhead expenses equal to 0.



Risk Analytics - MultiProductStatutoryLife

File Window Help

search: parameterizations, results, templates, tags,...

Navigation | Validations and comments

Name
- CapitalEagle
- MultiCompany
- MultiProductStatutoryLife
  - Parameterization
    - Unit Linked Model
  - Result templates
  - Results
- Podra
- Batches

Unit Linked Model Office Example with Reinsurance - Total Portfolio and Future New Business - fin

MultiProductStatutoryLife | Properties

<All>

Name
- multi product statutory life
  - actual expenses
    - company expenses
    - product expenses
  - reinsurance
    - common reinsurance
    - product specific reinsurance
  - model points
    - policy data
    - projection starting date
    - future policy data
    - future business
  - Products
    - Ulppcom4
    - Ulppcom5
    - Ulspcom5
  - demographics
    - Mortality Table Pricing
    - Mortality Table Actual
    - Disability Tables Pricing
    - Disability Tables Actual
    - Lapse Rate Tables
  - economic assumptions
    - risk discount rate
    - risk discount rate sensitivity
    - risk free rate
    - investment return on funds
    - inflation on expenses
    - company tax on statutory results
    - initial share holder capital
  - solvency assumptions
    - solvency margin on mathmatical reserves
    - solvency margin on sum at risk
    - solvency margin on disability premium
  - distribution channel commissions
    - Com4
    - Com5
  - banking
    - other income annualized
    - fund custody expenses annualized
    - fund loaded transaction costs

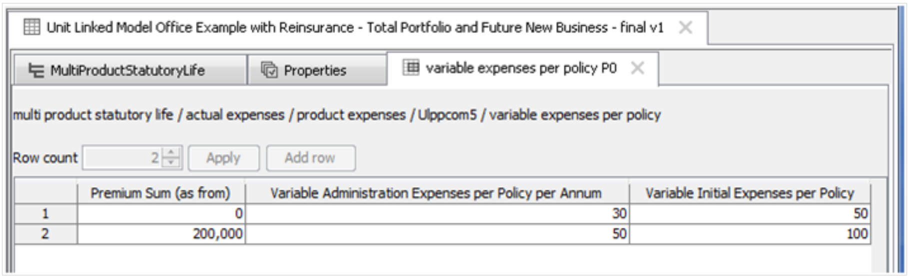
Figure 7.3: The input screen for mentioned parameters

- "apply inflation on company overhead expenses": One has to indicate whether the company expenses need to be indexed with inflation (true/false).

Product expenses

For all the defined products, expenses need to be entered separately. To add a product 'right click' on the "product expenses" tab and add a new product. Various expense types (absolute in CHF per policy, initial and administrative (i.e. annually recurring) as well as policy-size dependent expenses) need to be defined. We use the "ulppcom5" product as example. For the other two products similar parameters are entered.

- "product": "ulppcom5" needs to be entered this is required for a clear product/expense allocation required, otherwise the program is failing).
- "fixed administration expenses per policy per annum": We define them at CHF 100, which in the calculation are then broken down on a monthly basis.
- "apply inflation on fixed administration expenses per policy": To define whether the expenses need to be indexed with inflation.
- "fixed initial expenses per policy": We define them at CHF 100, to capture e.g. the initial underwriting expenses. These expenses are charged only once at the beginning.
- "apply inflation on fixed initial expenses per policy": To reflect whether future new business have unchanged initial expenses or whether these expenses are indexed with inflation.
- "variable expenses per policy": We can enter any number (by changing the "row count") of classes for policy sizes in CHF. Policy size = annual premium times premium payment duration. Depending on the policy size the "administration expenses per policy p.a. " and "initial expenses per policy" can be defined. In our example we allocate (in addition to the above mentioned fixed expenses) for policies with a sum of premiums below CHF 200'000 administration expenses per policy of CHF 30 p.a. and initial expenses per policy of CHF 50. For policies with a sum of


Unit Linked Model Office Example with Reinsurance - Total Portfolio and Future New Business - final v1
MultiProductStatutoryLife
Properties
variable expenses per policy P0
Apply
Add row

multi product statutory life / actual expenses / product expenses / Ulppcom5 / variable expenses per policy

Row count 2

	Premium Sum (as from)	Variable Administration Expenses per Policy per Annum	Variable Initial Expenses per Policy
1	0	30	50
2	200,000	50	100

Figure 7.4: Variable expenses per policy

premiums of CHF 200'000 or more the administration expenses are CHF 50 and the initial expenses CHF 100, cf. figure 7.4

In our example the product expenses for the other two products are very similar to "ulppcom5". However, in the real world the expenses can if necessary be entered with much more sophistication.

7.3.2 Reinsurance

Every underwriting year and currency will be separately treated (accounting series with different conditions; i.e. with separate loss carried forward amounts and loss carried forward interest rates).
For each settlement period a profit & loss calculation for the reinsurance is made:

- It consists of
  - the ceded reinsurance premium (ceded expenses and costs), plus
  - the surrender claw-back and
  - the extra deposit investment income.
- This is reduced by
  - the reinsurance commissions (initial and running/recurring), and
  - death and waiver of premium benefits (sums at risk).

- The initial loss (financing via the initial reinsurance commission) will be carried forward and increased by an interest rate plus an administrative charge (”Cost and Risk Annual Supplement on Ceded Premium”). In case that we have a profit, the reinsurer pays a profit participation to the insurance company.

Please note that saving premiums as well as the surrender payments need not to be part of the above calculation in case that the funds remain with the insurance company and are not passed to the reinsurer. Furthermore, the administrative charge to the reinsurer and the loss carried forward interest rates are not directly paid by the insurance company to the reinsurer, but only notionally allowed for in the profit & loss calculation by increasing the loss carried forward.
The end-user needs to define two kinds of parameter groups for modeling the reinsurance:

- Overall company reinsurance conditions to reflect the general treaty characteristics (”common reinsurance”)

- Product specific reinsurance parameters to reflect the various reinsurance conditions (quota share, commissions, amortization/-payback) on a product level.

Common reinsurance

In this section we enter the general reinsurance treaty characteristics:

- ”First Settlement Period Begin Date”: Enter the beginning of a month date as from which on the reinsurance calculation should start. This has to be in-line with the amounts entered under ”loss carried forward”. We have a US-date format; i.e. MM/DD/YY.

- ”First Settlement Period Ends Before”: This date defines the duration of the (first) settlement period. Could be e.g. 3 months after the begin date (in case of quarterly reinsurance settlements). However, in case we wish to have monthly results (as in our example), we enter one month after the begin date (format: MM/DD/YY).

- ”First Settlement Date”: Needs to be at least one day after the above end date (format: MM/DD/YY).

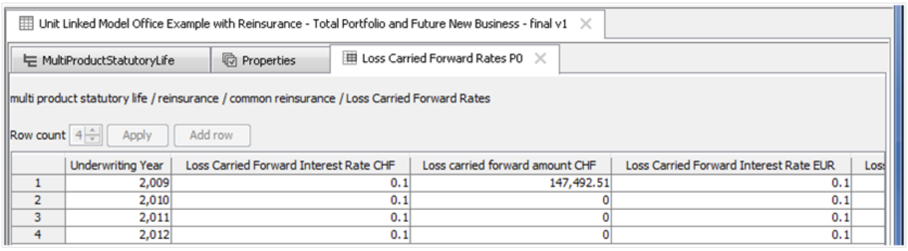
- "Settlement Interval in Months": Defines the time for interval of the settlements periods. In case of quarterly reinsurance settlements we would set 3 months.

- "Extra Deposit Interest Rate": The extra interest rate (p.a.) on the reinsured funds values (saving accounts) which the insurance company will pay to the reinsurer (e.g. to pass a part of the "kick-back" to the reinsurer). The calculation is done based on the average fund values over the specified period "Month Count for Average Fund Value" (see below). In case of quarterly settlements we might take the average fund value over the last 4 end of months (e.g. for end 31.12., 31.1., 28.2. and 31.3.). In our example we will, however, calculate for every month the average fund value (begin/end of month) as the basis for the calculation of the extra deposit interest.

- "Profit Participation": The participation (in our case we have 90%) which the reinsurer pays to the insurance company in case that the loss carried forward becomes positive.

- "Loss Carried Forward Rates": Here we define for every underwriting year and currency separately the loss carried forward amounts and loss carried forward interest rates) shown in figure 7.5. In the past the insurance company in our example sold only policies during the year 2009 with a loss carried forward amount (due to the reinsurer) of CHF 147'493. The interest rate which will be used also in future accounting years for the underwriting year 2009 is 10%. We need to specify for all the desired future years (with new business) the loss carried forward interest rate (in our example in addition to 2009 for the years 2010 to 2012; i.e. we set the row count = 4). Of course one could have (e.g. due to different economic situations) in the past different loss carried forward interest rates or going forward one would like to plan the new business with different loss carried forward interest rates.

- "Claw Back Percentage": In case of a lapse for a given policy the insurance company will in our example immediately amortize for this policy all the non-amortized initial reinsurance commissions (i.e. we set the claw-back percentage = 100%). The non-amortized reinsurance commissions are determined in the following way:


Unit Linked Model Office Example with Reinsurance - Total Portfolio and Future New Business - final v1

MultiProductStatutoryLife | Properties | Loss Carried Forward Rates P0

multi product statutory life / reinsurance / common reinsurance / Loss Carried Forward Rates

Row count 4

	Underwriting Year	Loss Carried Forward Interest Rate CHF	Loss carried forward amount CHF	Loss Carried Forward Interest Rate EUR	Los
1	2,009	0.1	147,492.51	0.1	
2	2,010	0.1	0	0.1	
3	2,011	0.1	0	0.1	
4	2,012	0.1	0	0.1	

Figure 7.5: Loss Carried Forward Rates

- Initial reinsurance commissions paid by the reinsurer,
- minus part of the initial reinsurance commissions which have been paid back by the insurance company before the lapse of the policy,
- minus administration and unit expenses as well as the risk premiums ceded by the insurance company to the reinsurer,
- minus extra deposit investment income ceded for this policy,
- plus/minus (the balance of) the acquisition commissions which have been reallocated over time
- plus the administration and unit expense running commissions paid by the reinsurer.

For a more detailed understanding of the parameters, see also next section.

- "Month Count for Average Fund Value": The calculation for the average fund value should be in-line with the settlement interval (see also above). In our example we simply assume a monthly calculation and we take the average fund value (begin/end of month; i.e. we set the month count equal to 2).

- "Cost and Risk Annual Supplement on Ceded Premium": The loss carried forward will be increased by an additional administrative charge ("Cost and Risk Annual Supplement on Ceded Premium": In our example it equals 2% of the ceded premiums).

Product specific reinsurance

For all the products we need to define a separate section (use the same name as under the "products" section) and enter the reinsurance parameters, in particular: The reinsured quota share of the premium, the payment of the reinsurance acquisition (initial) commission, the reinsurance treatment of the recurring expenses as well as the amortization/payback. In case that a product is not reinsured, we simply set the quota share equal to 0. To add a product simply 'right click' on the "product specific reinsurance" tab and add a new product.
We use the "ulppcom5" product as example:

- "product": "ulppcom5" needs to be entered (this - somehow redundant information - is required for a clear product allocation).
- "currency": The currency of the product has to be selected.
- "quota share": A number in the range of 0 to 1 has to be entered. In our example we have a 50% quota share reinsurance; i.e. 50% of the original premium is paid to the reinsurer.
- "truncated policy duration": To determine the truncated sum of premiums (SOP_{trunc}) the policy duration (in years) is "truncated" on our example at 25 years, in case the premium payment duration is larger. If no truncated duration is required on could e.g. enter 99 years ("lifelong"), or more.
- "reinsurance acquisition commission": We can define for various classes (= row count) of premium payment durations (in years) the four acquisition commission parameters A, B, C, D. In our example we only have one class (row count = 1), i.e. for all the premium payment durations we apply the same parameters. (In our example we define the class relating to all the premium payment durations > 0; i.e. we capture all the policies because they have a premium payment duration which is greater than zero.) However one could define a row count > 1 and input for various premium payment durations a separate set for the four acquisition commission parameters. (To determine which row or class of parameter set has to be applied: Premium payment duration is $\geq$ the indicated values. Of course, in this input table the listed premium payment durations need to be in strict increasing order.)

  Lets use the following abbreviations:

  - QS: quota share
  - SOP: sum of premiums (over the entire premium payment duration)
  - SOP_{trunc}: sum of premiums over the truncated premium payment duration
  - A, B, C, D: acquisition commission parameters for a given premium payment duration

  We further define:

  "reinsurance acquisition commission" =
  $\max(A; \min(B; C + D/(QS \times SOP))) \times QS \times SOP_{trunc}$
  Given the values in our example (A=0.05 and B=C=D=0) we therefore have for all the premium payment durations a reinsurance acquisition commission of 5% of 50% of the truncated (at 25 years) sum of premiums. This is an illustrative and simple assumption; however, the parameters allow a more sophisticated parameterization if required.

- "fixed and beta expenses": To compensate for the larger administration work done by the insurance company, the reinsurer pays back to the insurance company a percentage of the fixed (kappa) and beta expenses he receives[1]. The paid-back percentage is equal to: $1 - \min(A; B + (QS \times SOP)/C)$, where A, B and C are the parameters for the reinsurance expenses. QS and SOP defined as above. The Parameter C is a kind of calibration for the SOP and needs always to be set > 0 (otherwise we would have an error due to a division by 0), also in the case where QS is equal to 0%. Please note that for the "fixed and beta expenses" parameters the row count has always to be equal to 1. Given the values in our example (A=B=0.75 and C=1) the reinsurer therefore pays back 25% of the received fixed (kappa) and beta expenses to the insurance company.

- "amortization of acquisition commission in years": This parameters defines the "acquisition commission reallocation over time". The policyholder has to amortize the charged acquisition commission over a given period (in our example: 10 years; see later in the section 7.4 about "products"). However, the amortization of

[1] The reinsurer receives the quota share of the original fixed (kappa) and beta expense loadings.

the reinsurance acquisition commission does not necessarily need to be done over the same period. In our example we have defined 8 years, i.e. we have to reallocate in a different cash flow pattern the amortization of the acquisition commission and the insurance company will pay 1.25 (=10/8) of the original acquisition commission loading times the quota share to reinsurer. However, one could also imagine another example where the amortization of the acquisition commission for the reinsurance has a longer duration than the amortization in the original product. In such a case the reinsurer would pay-back in a first phase a part of the received acquisition commission to the insurance company. Please note that the calculation of the amortization patterns assumes that the sum of the original acquisition commission loadings (paid by the policyholder) times the quota share is equal to the sum of the loadings of the reinsurance acquisition commission.

For the other two products similar parameters are entered. Note please that there is no reinsurance for the single premium product ”ulspcom5” (quota share = 0).

7.3.3 Model points and projection starting date

Policy data

In this section we describe the existing portfolio: In our example the portfolio consists of 100 policies which have been sold in 2009. The so-called model point (one line in the policy data table) represents one sold policy. We need to enter the following policyholders' information, which in general can (automatically) be extracted from the administration system:

- ”Policy #”: This could be a number to identify the policy.
- ”Product”: The identifier used in the products section to denominate the product.
- ”Annual Premium”: The amount of the gross annual premium or of the single premium paid by the policyholder. (If 0 is entered, the policy is not considered.)
- ”Alpha”: represents the acquisition commission loading paid by the policyholder. In our example for all the policies α is equal to

5% of the present value of premiums or of the sum of premiums. However, because in the life insurance model this parameter does not depend from the product description, one could also imagine that the policies could have variable acquisition commission loadings.

- "Installments pa": The annual installments, e.g. monthly (12), quarterly (4), half-yearly (2) or annual (1).
- "Gender": 'male' or 'female'.
- "Birthdate": In the format YYYY-MM-DD.
- "Policy Inception Date": In the format YYYY-MM-DD.
- "Premium Payment Duration [Years]": The original payment duration in the policy contract. e.g. for single premium the premium payment duration is equal to 1.
- "Policy Duration [Years]": The original policy duration in the contract.
- "Risk Sum (relative)": In case of death of the policyholder the funds value (mathematical reserves) plus the "risk sum (relative)" will be paid-out. The "risk sum (relative)" is expressed as a factor of the sum of premiums.
- "Minimal Death Benefit": An absolute amount (in CHF), which will at least be paid-out in case of death of the policyholder. In case that the funds value (mathematical reserves) is larger than the minimal death benefit, only the funds value will be paid out. The sum at risk is equal to the difference (≥ 0) between the 'minimal death benefit' and the funds value (mathematical reserves).
- "Waiver of Premium Covered": 'false' or 'true'.
- "Waiver of Premium Waiting Delay": We have in our disability tables values for 6, 12 and 24 months of waiting periods. In case that no waiver of premium is covered, set this value equal to 0. The sum at risk is calculated as the risk free discounted value of premiums after the waiting period.

- "Mathematical Reserves": The actual reserves are usually extracted from the administration system and calculated for date as per the projection start date (in our example as per 31.12.2009).

Projection starting date

In our example we define the begin of the first period with 1.1.2010 (Input: DD/MM/YYYY). We have an existing portfolio as per 31.12.2009 and we assume for three years going forward future new business (i.e. for 2010 – 2012).

Future policy data

With the row count we define the number of future model points. In our example we have three model points; i.e. one for each of the three products. This describes the kind of policies sold in future. (For this open available life model, the input (= row count) is limited to 20 model points.) Of course, one could also define several and different policies (model points) for one product to capture and reflect the characteristics of the product for some particular parameters:

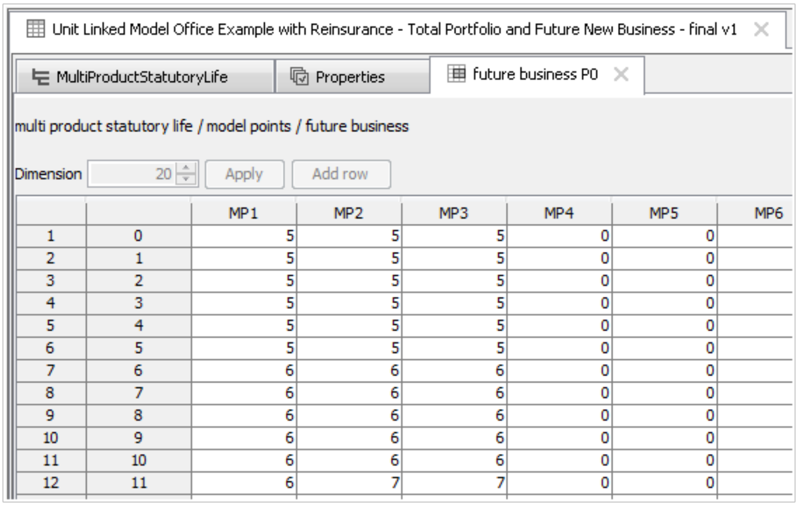
- "Model Points": The model points are denoted with MP1, MP2 and MP3, etc.
- "Product": The identifier used in the products section to denominate the product.
- "Annual Premium": The amount of the gross annual premium paid by the policyholder.
- "Alpha": α represents the acquisition commission loading paid by the policyholder. In our example for all the policies α it is equal to 5% of the present value of premiums or of the sum of premiums. However, because in the life insurance model this parameter does not depend from the product description, one could also imagine that the policies could have variable acquisition commission loadings.
- "Installments pa": The annual installments, e.g. monthly (12), quarterly (4), half-yearly (2) or annual (1).
- "Gender": 'male' or 'female'.

- "Age": Age of the policyholder in years when the policy will be sold (in future).
- "Premium Payment Duration [Years]": The original payment duration in the policy contract. e.g. for single premium the premium payment duration is set equal to 1.
- "Policy Duration [Years]": The original policy duration in the contract.
- "Risk Sum (relative)": In case of death of the policyholder the funds value (mathematical reserves) plus the "risk sum (relative)" will be paid-out. The "risk sum (relative)" is expressed as a factor of the sum of premiums.
- "Minimal Death Benefit": An absolute amount (in CHF), which will at least be paid-out in case of death of the policyholder. In case that the funds value (mathematical reserves) is larger than the minimal death benefit, only the funds value will be paid out. The sum at risk is equal to the difference (≥ 0) between the 'minimal death benefit' and the funds value (mathematical reserves).
- "Waiver of Premium Covered": 'false' or 'true'.
- "Waiver of Premium Waiting Delay": We have in our disability tables values for 6, 12 and 24 months of waiting periods. In case that no waiver of premium is covered, set this value equal to 0. The sum at risk is calculated as the risk free discounted value of premiums after the waiting period.

Of course, policies which are sold going forward do not have mathematical reserves to be considered at the beginning of the projection (as we have in the existing portfolio – see above). The model points for the future policy data should represent in a reasonable way the characteristics (types) of the policies which are sold in future. The number of sold policies (model points) is entered in the next session.

Products

The number of sold policies need to be defined by starting in period 0 (this corresponds to the starting of the projection; i.e. to January 2010 in our example): Month by month, for up to 120 periods or 10 years the


Unit Linked Model Office Example with Reinsurance - Total Portfolio and Future New Business - final v1
MultiProductStatutoryLife
Properties
future business P0
multi product statutory life / model points / future business
Dimension 20
Apply
Add row

		MP1	MP2	MP3	MP4	MP5	MP6
1	0	5	5	5	0	0	
2	1	5	5	5	0	0	
3	2	5	5	5	0	0	
4	3	5	5	5	0	0	
5	4	5	5	5	0	0	
6	5	5	5	5	0	0	
7	6	6	6	6	0	0	
8	7	6	6	6	0	0	
9	8	6	6	6	0	0	
10	9	6	6	6	0	0	
11	10	6	6	6	0	0	
12	11	6	7	7	0	0	

Figure 7.6: Future business - entering number of sold policies (by model points and months)

number of sold policies (as defined above by the model points - future policy data) can be entered. (Please note that in the open available life model, the row and column counts can not be changed.)
For the next 3 years we expect in our example total (future) sales of 900 policies for the three products (model points): 200 policies in 2010, 300 policies in 2011 and 400 policies in 2012. With the existing portfolio we therefore capture and model 1'000 policies in total. This is illustrated in figure 7.6.

7.4 Input parameters

The following parameters are the core input to describe the products, through these characteristics of the unit-linked policies. Combined to these unit-linked products one could also have death benefits and waiver of premium insurance. These coverages are not further described here; they are captured on a per policy basis in the policy data description (see above). However, the costs for these coverages are either charged to the premium or to the actual funds values (saving account or component of the product).

One can define as many products as desired, also by using meaningful, internal names: We describe subsequently the parameters of the (first) product "ulppcom5" (periodic premium payment and linked to commission payment rule "com5"):

- "Products": 'Right click' on "products" and add a new product by entering the desired name; in our example "ulppcom5".
- "currency": Choose the desired currency

Acquisition expenses alpha (α):

- "alpha calculation base": The calculation base for the acquisition commission loading α (which is defined in the model points - policy data section) could either be the present value of premiums PV or of the sum of premiums SOP.
- "technical interest rate for present value of premium": The technical interest rate i which is used to calculate the present value of premiums is in our example 1.75%. The total acquisition commissions AC charged to the policyholder are therefore:

  $AC = \alpha \times PV$ or $AC = \alpha \times SOP$

  The AC are amortized over a duration m_1 or m_2 depending on various conditions (for the policy duration n and for the annual premium AP):
- "alpha threshold policy duration in years": This is denoted with $T(n)$ and is in our example 10 years.
- "alpha threshold annual premium": This is named with $T(AP)$ and is in our example CHF 1'000.
- "alpha amortization time in years min": This is denoted with m_1 and is in our example 10 years.
- "alpha amortization time in years max": This is denoted with m_2 and is in our example 10 years. With t we denote the actual policy year (from 1 to n) and with μ the number of the annual installments [e.g. monthly (12), quarterly (4), half-yearly (2) or annual (1)].

The acquisition commissions AC_t charged in year t (charged against the actual premium payment) are then defined by:

$$AC_t = \begin{cases} \dfrac{AC}{u \cdot \ddot{a}^{(u)}_{\overline{m_1}|}} & \text{if } [n < T(n) \text{ or } AP > T(AP)] \text{ and } t \leq m_1 \\[2ex] \dfrac{AC}{u \cdot \ddot{a}^{(u)}_{\overline{m_2}|}} & \text{if } [n \geq T(n) \text{ and } AP \leq T(AP)] \text{ and } t \leq m_2 \\[2ex] 0 & \text{otherwise} \end{cases}$$

and where

$$\ddot{a}^{(u)}_{\overline{t}|} = \frac{1 - v^t}{d^{(u)}}, \quad v = \frac{1}{1+i}, \quad \text{and} \quad d^{(u)} = u \cdot [1 - (1+i)^{-1/u}] .$$

With i we denote the technical interest rate. In our example we have simplified assumptions where the AC are always amortized over a fixed period of 10 years ($m_1 = m_2 = 10$). The threshold for the annual premium $T(AP)$ does basically not play a role in the above formulae. Finally, the product design makes only sense if the policy duration is at least 10 years (i.e. no policies with a policy duration $n < 10$ should be sold).

Collection expenses beta (β):

During the premium payment period the beta expenses β in proportion to the periodic premium are charged to the policyholder. With the 5 beta parameters (A, B, C, D and E) various cost structures can be modeled. We define: $\beta = \min(A + \max(B; (n - C) \times D); E)$, where n is the policy duration. In our example we have beta expenses varying between 7% and 8% of the premium:

- "beta a": Is set equal to 0.07
- "beta b": Is set equal to 0
- "beta c": Is set equal to 15
- "beta d": Is set equal to 0.001
- "beta e": Is set equal to 0.08

Proportional administration expenses gamma (γ):

During the entire policy duration the gamma expenses γ in proportion to the funds value are charged (by reducing the funds value respectively the mathematical reserves). In our example we charge 0.05% p.a. of the funds value.

- "gamma1 of mathematical reserves": Is set equal to 0.0005 p.a.
- "gamma2 of mathematical reserves": Is set equal to 0 p.a.

We have two gamma parameters. With gamma2 one could e.g. model a bonus parameter or a kick-back refund to the policyholder by setting a negative parameter (negative charge). Please note that the gamma expenses are calculated at the beginning of every month and charged on a monthly basis; i.e. the above gamma parameters are divided by 12.

Fixed administration expenses kappa (κ):

During the entire policy duration the (annual) kappa expenses κ in CHF are charged to the policyholder at the beginning of every month (i.e. 1/12). With the 5 (periodic premium products) respectively 6 (single premium products) kappa parameters (A, B, C, D, E and F) various cost structures can be modeled.
We define for the periodic premium products: $\kappa = \min(A + \max(B; (n - C) \times D); E)$, where n is the policy duration. And for the single premium products (i.e. premium payment duration = 1 year): $\kappa = \min(A + \max(B; (n \times SP/F - C) \times D); E)$, where SP is the single premium. In our example we have kappa expenses varying between CHF 50 and CHF 100 p.a. :

- "kappa a": Is set equal to 50
- "kappa b": Is set equal to 0
- "kappa c": Is set equal to 15
- "kappa d": Is set equal to 2.50
- "kappa e": Is set equal to 100
- "kappa f": Is set equal to 0 (as it is n/a for the "ulppcom5" product)

Other parameters:

- "surrender charge of funds": In case of a surrender a penalty (charge) in proportion to the funds value is charged to the policyholder depending on the elapsed time. A variable scale can be entered. In our example we have a monthly linearly falling charge (from 100% to 0%) over the first 5 policy years. However, one could also define a flat (constant) surrender percentage over the entire policy duration.
- "mortality table pricing": The desired mortality table has to be selected. The risk premium (1/12 of the q(x)) is charged at the beginning of every month (reduction of policyholders' funds).
- "mortality table actual": The desired mortality table has to be selected.
- "waiver of premium pricing": The desired waiver of premium table has to be selected. The risk premium (1/12 of the i(x)) is charged at the beginning of every month (reduction of policyholders' funds).
- "waiver of premium actual": The desired waiver of premium table has to be selected.
- "lapses": The desired lapse table has to be selected.
- "distribution channel selection": The desired commission payment type or "scale" has to be selected (according to the required distribution channel remuneration, etc.)

For the other two products similar parameters are entered.

7.4.1 Demographics

Mortality

Two mortality tables have been defined, for pricing purposes and for actual outcomes:

- "bfs mort19982003": Derived from BFS-data (Bundesamt fuer Statistik) a pricing mortality table with ultimate q(x) for male and q(y) for female has been entered (to import the data into PillarOne: simply use copy/paste from Excel).

- "bfs actual": The mortality table with the actual values has been defined as 80% of the pricing mortality table.

For pricing purposes the risk premium charges are determined according to the "age last birthday". For actual valuation purposes the mortality is calculated by interpolating (i.e. "exact").
The end-user can define the size of the table and at which age the table should start. There are no limits with respect to the number of mortality tables (and also other tables, e.g. for disability, lapses, commissions, etc.) one would like to define. To create a new table simply 'right click' on "mortality table" tab and add a new table. For example one could define a different mortality table for every product.

Disability

Two disability tables have been defined, for pricing purposes and for actual outcomes (as they are derived from the "Invaliditaetsstatistik 1996/2000 in der schweizerischen Kollektivlebensversicherung", SAV Bulletin 2/2004):

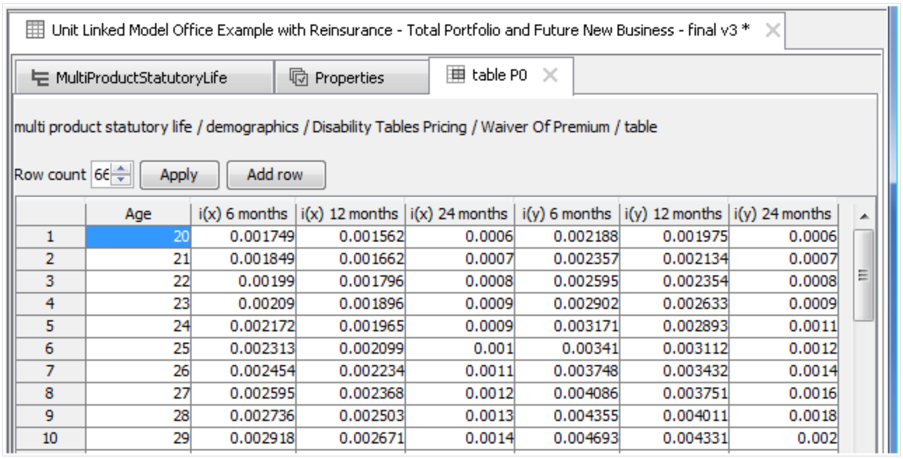
- "waiver of premium": Disability table for pricing purposes for both gender ($i(x)$ and $i(y)$) and 3 waiting periods of 6, 12 and 24 months, cf. figure 7.7

- "waiver of premium actual": This disability table has been calculated as 80% of the pricing disability table.

For pricing purposes the risk premium charges are determined according to the "age last birthday". For actual valuation purposes the disability rate is calculated by interpolating (i.e. "exact").
The end-user can define the size of the table and at which age the table should start. There are no limits with respect to the number of disability tables.
Disability is modeled as an event terminating the policy, like death or surrender. In case of disability, the policyholder receives his savings (funds value) at end of month plus the risk free discounted value of premiums after the waiting period. No reactivations are modeled.

Lapses

Various lapse rate tables have been defined, e.g.:


Unit Linked Model Office Example with Reinsurance - Total Portfolio and Future New Business - final v3 *
MultiProductStatutoryLife
Properties
table P0
Apply
Add row

multi product statutory life / demographics / Disability Tables Pricing / Waiver Of Premium / table

Row count 66

	Age	i(x) 6 months	i(x) 12 months	i(x) 24 months	i(y) 6 months	i(y) 12 months	i(y) 24 months
1	20	0.001749	0.001562	0.0006	0.002188	0.001975	0.0006
2	21	0.001849	0.001662	0.0007	0.002357	0.002134	0.0007
3	22	0.00199	0.001796	0.0008	0.002595	0.002354	0.0008
4	23	0.00209	0.001896	0.0009	0.002902	0.002633	0.0009
5	24	0.002172	0.001965	0.0009	0.003171	0.002893	0.0011
6	25	0.002313	0.002099	0.001	0.00341	0.003112	0.0012
7	26	0.002454	0.002234	0.0011	0.003748	0.003432	0.0014
8	27	0.002595	0.002368	0.0012	0.004086	0.003751	0.0016
9	28	0.002736	0.002503	0.0013	0.004355	0.004011	0.0018
10	29	0.002918	0.002671	0.0014	0.004693	0.004331	0.002

Figure 7.7: Waiver of premium table for pricing

- "lapse rate1" (the 'main' table in our example for the periodic premium products): To reflect that in the 6th year the lapse rate is increasing again for the periodic premiums because of the discontinuation of the surrender charge (after 5 years).
- "lapse rate2": With continuing falling lapse rates for the single premium product.

Various other tables have been created for testing purposes (for example for calculation with one policy): e.g. to simulate "no lapse", "surrender in year2" or "surrender in year5".

7.4.2 Economic assumptions

Further, simple parameters (not e.g. interest curves, remain constant over the entire projection period) can be entered in the economic assumptions section:

- "risk discount rate": 7.5%
- "risk discount rate sensitivity": not used
- "risk free rate": 1.5%
- "investment return on funds": 5.0%

- "inflation on expenses": 1.0%
- "company tax on statutory results": 25.0%
- "initial share holder capital": CHF 1'000'000

7.4.3 Solvency assumptions

Only the capital requirements under the Solvency I regime are modeled in this life insurance PillarOne model to compute the minimal solvency capital which is locked-in. In our example we set 100% of the standard solvency parameters:

- "solvency margin on mathematical reserves": 1.0%
- "solvency margin on sum at risk": 0.3%
- "solvency margin on disability premium": 20% of the annualized disability (waiver of premiums) risk premiums

The values are calculated at the end of every month.

7.4.4 Distribution channel commissions

Various distribution channels can be defined: 'Right click' on "distribution channel commissions" and add a new commission section by entering the desired name; in our example "com5". For "com5" we define the following parameters:

- "acquisition commission": In our example the insurance company pays an acquisition commission of 5.0% of the calculation base. The calculation base could either be the present value of premiums or of the sum of premiums. The acquisition commission is fully paid out in the month when the policy is sold (i.e. upfront at the policy inception date).
- "acquisition commission base": In our example we select "sum of premiums" as calculation base.
- "discount rate for present value of premiums": The interest rate which is used to calculate the above mentioned present value of premiums. In our example we do not use it.

- "apply max duration": We have to select whether or not the policy duration is capped (or 'truncated').
- "max duration in year": In our example the policy duration will be capped at a maximum of 25 years.
- "portfolio commission annualized": In our example the insurance company also pays out a portfolio commission of 0.05% p.a. of the 'mathematical reserves' ("portfolio commission base"). The payment is done on a monthly basis (i.e. 1/12).
- "portfolio commission base": Is the 'mathematical reserves' (no selection possibility).
- "claw back share": In case of a surrender the acquisition commission has to be refunded ("claw-back") by the distribution channel (agent) to the insurance company ("non-amortized" part of the initial or acquisition commissions due by the agent). A variable (row count) claw-back scale can be entered. In our example we have a linearly falling claw-back share within the first 3 policy years (or 36 months). Only after the first 3 years the acquisition commissions are fully earned.
- "shortfall of claw back": Additionally we can define a shortfall of the claw-back payment; i.e. when the distribution channel fails to pay back the due part of the acquisition commissions. A probability table for the shortfall of claw-back can be entered. (Variable size of the table is being defined with the row count.) In our example we assume a general (flat: row count = 1) probability of shortfall of 2.5% of the claw-back share for all the months.

For the distribution channel commission "com4" we have the same input except for the "acquisition commission": 4.0%. This could be justified if e.g. the administrative work done by the insurance company is varying between the different distribution channels or for various products.

7.4.5 Banking

The banking transactions are creating additional expenses or other income. We have three types of parameters which can be used to simulate these effects:

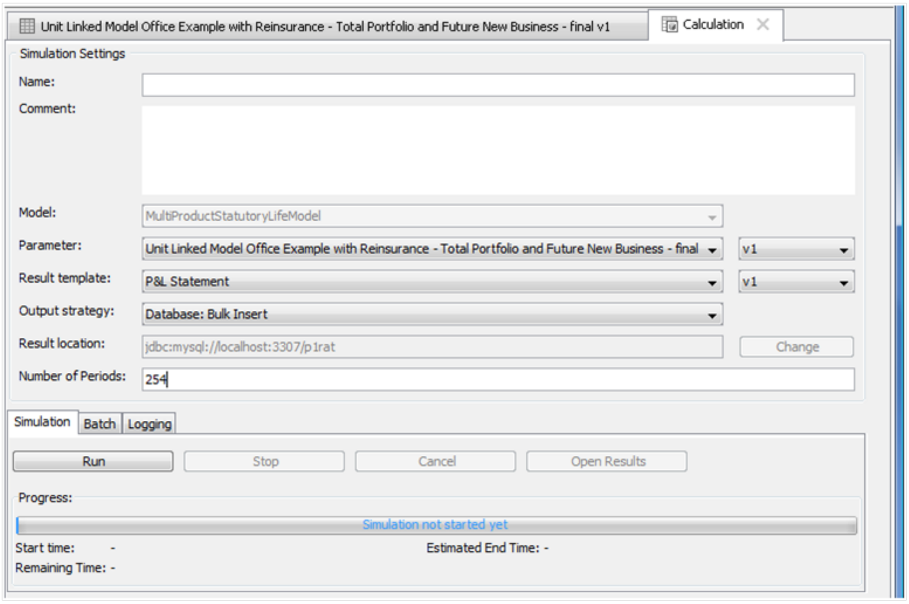
- "other income annualized" ("kick-back"): In some circumstances the banks are paying a fee to the insurance company ("kick-back") to compensate for the various investment processes. In our example we assume that the insurance company will get 0.25% p.a. (or 25 bp) of the funds value. The model, however, calculates and accrues the 'other income' on a monthly basis (i.e. 1/12).
- "fund custody expenses annualized": In our example we assume for the insurance company fund custody expenses of 0.05% p.a. of the funds value. These expenses are also calculated and accrued on a monthly basis (i.e. 1/12).
- "fund loaded transaction costs": Finally, in our example, we also have modeled transaction costs of 0.25% of the net buy-sell amount (monthly) which the insurance company will have to pay (0.25% of the monthly change in reserves).

7.5 Running calculations

For doing the calculations there are a few parameters which need to be entered. Usually the simulation settings are defined by the newest version of the parameters as illustrated in figure **??**:
Important is – in addition to the version of the parameters – the definition of the duration of the projection (simulation) to start the calculation of the projections:

"Number of Periods" (in months):

We do ("only") one run respectively a projection with 254 periods or months (because of exporting the results with "copy/paste" into the old Excel which has only 256 columns). This corresponds to a projection period of over 21 years. However, one could do without problems also projections over 40 years or 480 periods (the Excel 2007 version has enough columns to import this directly).
One could also prepare several runs and start them on a batch-basis (e.g. over night) to do various calculations and analysis with no further input and manual interventions.
The run times need to be analyzed for models with larger portfolios. However, our small portfolio and calculation/projection over 21 years needed approximately 15 seconds of computation time on an older laptop PC.


Unit Linked Model Office Example with Reinsurance - Total Portfolio and Future New Business - final v1
Calculation
Simulation Settings

Name:

Comment:

Model: MultiProductStatutoryLifeModel

Parameter: Unit Linked Model Office Example with Reinsurance - Total Portfolio and Future New Business - final v1

Result template: P&L Statement v1

Output strategy: Database: Bulk Insert

Result location: jdbc:mysql://localhost:3307/p1rat

Number of Periods: 254

Change
Simulation
Batch
Logging
Run
Stop
Cancel
Open Results
Progress:

Simulation not started yet

Start time: -

Estimated End Time: -

Remaining Time: -

Figure 7.8: Calculation settings

7.6 Results and output presentation

In this section we comment the results which are presented with a standard Profit & Loss grid for the statutory net income and some portfolio information (balance sheet, key figures). The results can easily be imported into an Excel spreadsheet (”copy/paste”). The results are illustrated in figure **??**

The various positions are presented on a monthly basis; i.e. we calculate and see monthly results. During the year, the data is accumulated (i.e. the model shows ”year-to-date” information) and at the beginning of a new year the data start again from zero. One can therefore easily read from the output the quarterly or half-yearly information and of course the year-end (December columns). One can 'click' (drill-down) on the various profit & loss items and in general one receives a detailed, technical split (”internal view”) of the elements. Positive figures denote an income for the insurance company, negative figures an outgo. Therefore a positive net income is a profit and a negative net income is a loss.

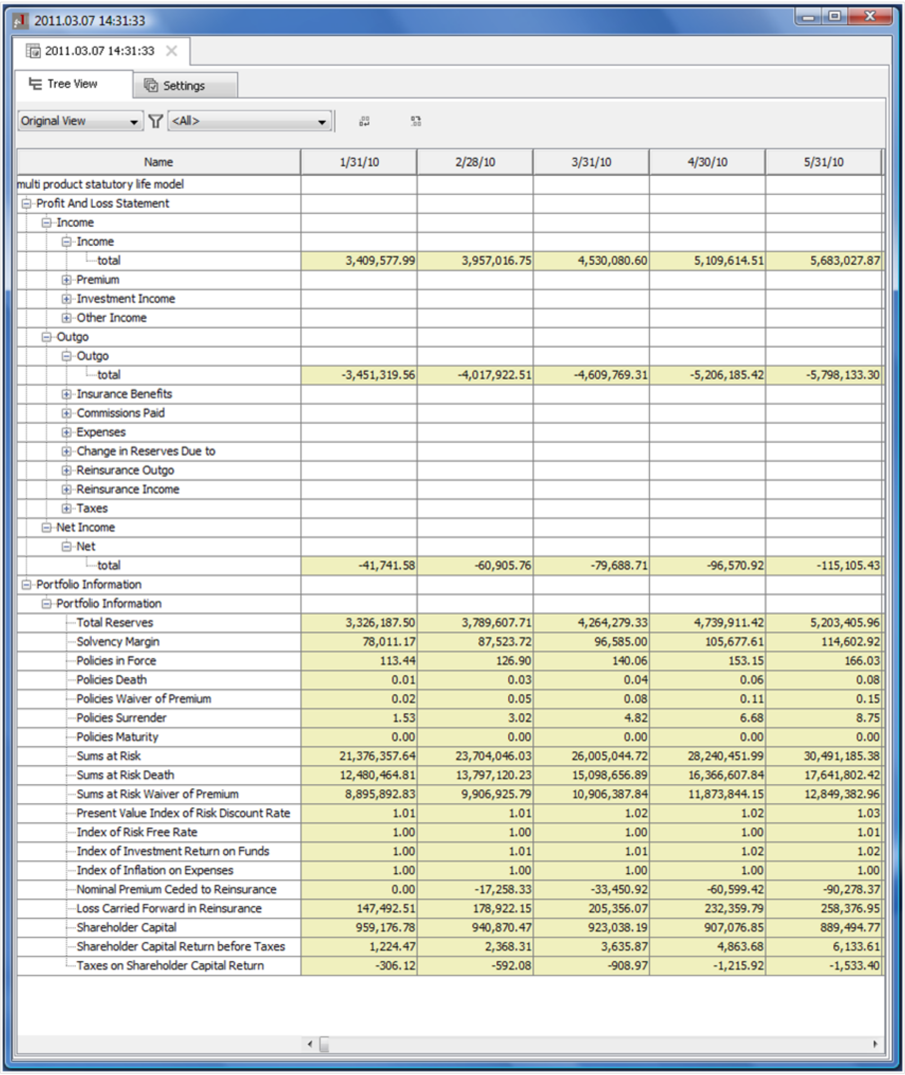
7.6.1 Income

The income consists of premiums, investment and other revenues.

Premium

The total premium is directly segregated (presented) into the following (technical) elements:

- saving premium: Paid premium less the alpha, beta and kappa expenses.
- alpha: The acquisition commissions which are charged to the policyholders.
- beta: The beta expenses (collection expenses) which are charged to the policyholders.
- kappa: The fixed administration expenses (kappa) which are charged to the policyholders.



Name	1/31/10	2/28/10	3/31/10	4/30/10	5/31/10
multi product statutory life model					
Profit And Loss Statement					
Income					
Income					
total	3,409,577.99	3,957,016.75	4,530,080.60	5,109,614.51	5,683,027.87
Premium					
Investment Income					
Other Income					
Outgo					
Outgo					
total	-3,451,319.56	-4,017,922.51	-4,609,769.31	-5,206,185.42	-5,798,133.30
Insurance Benefits					
Commissions Paid					
Expenses					
Change in Reserves Due to					
Reinsurance Outgo					
Reinsurance Income					
Taxes					
Net Income					
Net					
total	-41,741.58	-60,905.76	-79,688.71	-96,570.92	-115,105.43
Portfolio Information					
Portfolio Information					
Total Reserves	3,326,187.50	3,789,607.71	4,264,279.33	4,739,911.42	5,203,405.96
Solvency Margin	78,011.17	87,523.72	96,585.00	105,677.61	114,602.92
Policies in Force	113.44	126.90	140.06	153.15	166.03
Policies Death	0.01	0.03	0.04	0.06	0.08
Policies Waiver of Premium	0.02	0.05	0.08	0.11	0.15
Policies Surrender	1.53	3.02	4.82	6.68	8.75
Policies Maturity	0.00	0.00	0.00	0.00	0.00
Sums at Risk	21,376,357.64	23,704,046.03	26,005,044.72	28,240,451.99	30,491,185.38
Sums at Risk Death	12,480,464.81	13,797,120.23	15,098,656.89	16,366,607.84	17,641,802.42
Sums at Risk Waiver of Premium	8,895,892.83	9,906,925.79	10,906,387.84	11,873,844.15	12,849,382.96
Present Value Index of Risk Discount Rate	1.01	1.01	1.02	1.02	1.03
Index of Risk Free Rate	1.00	1.00	1.00	1.00	1.01
Index of Investment Return on Funds	1.00	1.01	1.01	1.02	1.02
Index of Inflation on Expenses	1.00	1.00	1.00	1.00	1.00
Nominal Premium Ceded to Reinsurance	0.00	-17,258.33	-33,450.92	-60,599.42	-90,278.37
Loss Carried Forward in Reinsurance	147,492.51	178,922.15	205,356.07	232,359.79	258,376.95
Shareholder Capital	959,176.78	940,870.47	923,038.19	907,076.85	889,494.77
Shareholder Capital Return before Taxes	1,224.47	2,368.31	3,635.87	4,863.68	6,133.61
Taxes on Shareholder Capital Return	-306.12	-592.08	-908.97	-1,215.92	-1,533.40

Figure 7.9: Results of the calculations (projections)

Investment income

The total investment income is equal to the investment income on the policyholders' funds:

- ph funds: Investment income on the policyholders' funds.
- sh assets and claims reserves: These amounts are not calculated (Investment income on shareholders' funds is not presented in the profit & loss statement; see also comment under net income).

Other income

The total other income includes the portfolio transfer:

- other: Other income respectively "kick-back" paid by banks.
- portfolio transfer: In the first month of the projection the mathematical reserves at the start of the projection.

7.6.2 Outgo

The outgo consists of insurance benefits, commissions paid, expenses, change in reserves, reinsurance outgo/income and taxes. These elements are explained in the following sections.

Insurance benefits

The total insurance benefits are the amounts paid to the policyholders. The payments are financed by reducing the reserves (policyholders' funds) or by the risk premium payments (which are also financed at the beginning of every month by reducing the policyholders' funds).

- death: Payment in case of death according to the actual mortality table.
- waiver of premium: This (disability) is modeled as an event terminating the policy, similar to a surrender. In case of disability, the policyholder receives his savings at end of month plus the risk free discounted value of premiums after the waiting period.
- surrender: Surrender benefit which is paid to policyholders (please note that the surrender charge is reduced from the policyholders' funds and flows as an income to the insurance company).

- terminal benefit or maturity payment: The funds value which will be paid to policyholders when the policies expire.

Commissions paid

The commissions paid relate to the distribution channel and include all the commission types plus the claw-back less the shortfall of the claw-back:

- acquisition: Initial commission payment to the distribution channel in the month of the policy inception.
- portfolio: The ongoing portfolio commission payments.
- nominal clawback: The "non-amortized" initial or acquisition commissions ("claw-back") which are due from the distribution channel in case of a lapse. This is an income for the insurance company and has a positive sign.
- clawback shortfall: The shortfall of the above.

Expenses

The total expenses are the actual expenses ("2nd order") and include company and product expenses as well as the banking costs:

- company overhead
- fixed initial per policy
- fixed admin per policy
- variable initial per policy
- variable admin per policy
- banking transaction costs
- banking custody expenses

Change in reserves

The total change in reserves consists in negative (outgo) and positive (income) items for the insurance company:

- saving premium: Same amount (but with negative sign) as the saving premium under income.
- kappa: Fixed administration expenses ("1st order"), positive amount (part of the kappa expenses which cannot be deducted from the premiums, e.g. for single premium products).
- gamma1: Proportional administration expenses ("1st order"), positive amount.
- gamma2: Proportional administration expenses ("1st order"), positive amount.
- risk premium for death: Risk premium charged to policyholders ("1st order").
- risk premium for waiver of premium: Risk premium charged to policyholders ("1st order").
- investment income: Corresponds to the investment income on the policyholders' funds (see above, but with negative sign).
- death benefit: Reserves which become free due to deaths of policyholders (positive amount).
- surrender benefit: Reserves which become free due to surrenders of policyholders (positive amount).
- waiver of premium benefit: Reserves which become free due to disability of policyholders (positive amount).
- terminal benefit or maturity payment: Reserves which become free due to maturity of the policies (positive amount).
- portfolio transfer: Corresponds in the first month of the projection to the mathematical reserves at the start of the projection (see other income above, but with negative sign).

Reinsurance outgo

The reinsurance cash flow is generally presented under outgo with two items, a negative (outgo) and a positive (income) one. Usually, on a longer term the reinsurance is an expenditure which the life insurance company has. The reinsurance outgo and income positions are calculated at the end of each month but presented in the output at the beginning of following month. In case one would need a precise monthly closing one would need to manually include the reinsurance positions of the subsequent month to have a "clean" (correct) accrual with a corresponding tax adjustment (this can easily be done in Excel). The reinsurance outgo includes the following positions (calculated according or in proportion to the quota share of the various products):

- premium risk death ceded
- premium risk disability ceded
- premium alpha ceded
- premium unit expenses ceded [i.e. fixed administration expenses (kappa)]
- premium admin expenses ceded [i.e. beta (collection) expenses]
- surrender clawback
- extra deposit investment income

Reinsurance income

The reinsurance income includes the following positions:

- reinsurance commissions
- acquisition commission reallocation over time: The acquisition commission is reallocated over time according to the "amortization of acquisition commission in years" parameter (see also section "Product specific reinsurance"). This item could have a positive or negative sign.
- admin running commissions [i.e. beta (collection) expenses]: Reinsurance company pays back to the reinsurance company a part of the expenses it receives.

- unit running commissions [i.e. fixed administration expenses (kappa)]: Reinsurance company pays back to the reinsurance company a part of the expenses it receives
- death benefits in excess of savings
- waiver of premium benefits
- profit participation

Taxes

Taxes are calculated as a simple percentage based on all the above income less outgo positions.

7.6.3 Net income

The net income only deals with the policyholders' funds and not with the shareholders' funds (which are included under 'portfolio information'). It is the sum of the income minus the outgo.

7.6.4 Portfolio information

Following portfolio information (a.o. balance sheet elements) and/or key figures are calculated at the end of each month:

- total reserves
- solvency margin
- policies in force: Total number of policies.
- policies death: Number of policyholders who died.
- policies waiver of premium: Number of policyholders who became disabled.
- policies surrender: Number of policyholders who surrendered their policy.
- policies maturity: Number of policies which came to maturity.
- sums at risk: Total sums of death plus disability (waiver of premium)

- sums at risk death
- sums at risk waiver of premium: Risk free discounted value of premiums after the waiting period
- present value index of risk discount rate
- index of risk free rate
- index of investment return on funds
- index of inflation on expenses
- nominal premium ceded to reinsurance
- loss carried forward in reinsurance: Includes the 'cost and risk annual supplement on ceded premium'.
- shareholder capital: Is equal to the 'shareholder capital' of the previous period plus the 'shareholder capital return before taxes' minus the 'taxes on shareholder capital return'.
- shareholder capital return before taxes
- taxes on shareholder capital return

With the last three lines (above) one could also approximately build-up a financial position of the shareholder and e.g. compare this with the required solvency margin. In our example we see that between March 2012 and May 2013 the company would have enough liquidity but the required Solvency I margin would not be fulfilled.

7.6.5 Subsequent treatment

Separate (batch) calculations could be set-up and done for a single policy (profit testing purposes), a specific product, a sub-portfolio or the entire business of the life insurance office. The various results of PillarOne can easily be imported into an Excel spreadsheet ("copy/-paste")[2].
With the grouping function and an Excel-template sheet the monthly PillarOne results could then easily be presented on e.g. an annual or a quarterly basis.

[2]Please note that: Excel 2003 has only 256 columns, which correspond to number of projected months (runs) if one would simply like to "copy/paste" the PillarOne results into Excel.

Subsequent treatment could e.g. be the present value calculations of the cash flows (value of business in force) or cost of capital derivation (cost of locking-in), etc.

7.7 Future development of the life insurance PillarOne cash flow model

We hope to receive feedback and input from the users of this life insurance PillarOne model.
We could further develop this first building block on various areas. Here are some high-level ideas:

- Extend this model for including traditional life insurance products (not only unit-linked)
- Model assets and asset-dependent dynamics of life insurance business (e.g. dynamic lapse rates, bonus participation, investment strategies)
- Enhance to do stochastic simulations (to do TVOG, MCEV, etc. calculations)
- Perform dynamic ALM-analysis, economic capital and solvency II calculations, etc.

Chapter 8

The application

8.1 Input Parameters

to open them[1] *to edit them*[2]

8.1.1 Versions

A parametrization can be in different states: editable or locked; valid or invalid. If a result is based (depending) on a (valid!) parametrization, this parameterization is no longer editable but locked. This can be seen by a small icon of a lock. Unless the result is removed, the parameters which were used to produce this result stay locked. This is important to guarantee that the results are reproducable and auditable. Nevertheless, you may want to create different versions of the input parameters before running any simulation. Often users start with a base or a planing scenario and then derive for example a high interest rate scenario or a different reinsurance program.
Note: *In such a context many people use the term scenario interchangably with parametrization. This stems from the fact that for business people the input parameters capture operational options or scenarios: What happens if the interest rates increase or, how does a different reinsurance program affect the risk capital, etc.?*
Most of the parameters in such derived parametrizations are left identical to the original parametrization and it would be cumbersome to

[1]TODO: td
[2]TODO: td

enter or load them again. `RiskAnalytics` offers a convenience function that lets you explicitly create a new version of a parametrization, i. e. making a copy of a parametrization. Independent of whether the original parametrization was locked or not, the copy will be editable. *graphic*[3]

8.2 Defining outputs

A model does not define itself which output variables are collected during a simulation run and then stored for subsequent analysis or reporting. Different users or even the same user in different situations may have different requirements. With a new model or model version the users often start by collecting a lot of output variables because they want to get a better feel for the model and verify that it performs as expected. Later on, for the operational use of the model, the actuaries may get more technical key figures collected and the business people more of the financial key figures. In principle, you could collect always all available output variables. People who are building models in Excel are used to this mode. For Monte Carlo simulation models this gets very quickly unwieldy. More importantly, the output would not be tailored to the user of the output: An actuary, a business analyst, a portfolio manager, etc.
Result templates are used to define which output variables are collected during a simulation run and at which granularity. Hence, they are the means to tailor the output to the users.
The different strategies for collecting output data are:

- foo bar

to open them[4] *to edit them*[5]

8.2.1 Versions

The version behaviour of result templates is analogue to the one for parametrizations. A result template can be in two states: editable or locked. The latter happens if a result depends on this result template. Unless the result is removed, the result template, which was used to

[3]TODO: graphic
[4]TODO: td
[5]TODO: td

produce this (dependent) result, stays locked. This is important to guarantee that the results are reproducable and auditable. Nevertheless, you may want to create different versions of a result template before running any simulation. The typical use case for this is that a power user defines the templates for less experienced users, or users who are less familiar with the model.
If you want to derive a result template from another one because you want to leave large sections of the settings identical to the original result template, then it would be cumbersome to enter or load them again. `RiskAnalytics` offers a convenience function that lets you explicitly create a new version of a result template, i. e. copy it. Independent of whether the original result template was locked or not, the copy will be editable. *graphic*[6]

8.3 Running calculations / simulations

complete section[7] There are two kinds of models: Deterministic calculation models and Monte Carlo simulation models. Both are supported by `RiskAnalytics`. The common parameters are described here and the ones which are different (in the two worlds) are described in the following subsections.
common inputs here
name
comment
parametrization
graphic[8]

8.3.1 Deterministic calculations

complete section[9] no seed, no number of iterations

8.3.2 Monte Carlo simulations

seed optional
number of iterations

[6]TODO: graphic
[7]TODO: needs to be done!
[8]TODO: graphic
[9]TODO: needs to be done!

8.4 Results

The main interface to a result can be opened via the context menu in the left pane
screenshot
where do we put ‘working with tabs’?

Part II

Reference Guide

Chapter 9

Concepts

In order to understand the concepts of `PillarOne RiskAnalytics`, it is necessary to get comfortable with the words that we use to denote various aspects in their context, i. e.

- **Model** contains all the risk drivers, applying them in a specific order by using a data driven workflow
- **Components** are the building blocks of a model
- **Packets** contain the information sent from component to component and persisted as results
- **Wiring** describes the relations among the components of a model

as introduced in Section 9.1 and

- **Simulation**: executing a model with a specific parameterization collecting results as described in a result template
- **Iteration**: one possible realization in a stochastic context, also referred to as an 'iteration run'; one specific, actual realization in a deterministic context
- **Period**: projection step; i. e. a calender based time interval, such as a business year
- **Results**: Resultanalysis, graphically and numericalle by different graphics and **views**

as introduced in Section 9.2.
Before entering into more details, let us give some definitions:

- **Persistence**
- **Parameterization**
- **Result**
- **Business Logic**
- **Collection Mode**
-
-

9.1 Risk Model

There are three different kinds of models:

- deterministic models especially designed for life modelling
- stochastic models with no time concept
- stochastic models with a time concept

Examples are the models **Capital Eagle** or **Podra**.

9.1.1 Model

9.1.2 Components

9.1.3 Packets

9.1.4 Wiring

9.2 Simulation

A **simulation** runs a **risk model** a given number of times. Each of the single executions – all together makeing up a simulation – is an **iteration**. In other words, a simulation starts a number of iterations and works on the collected results. Every iteration runs the model for a given number of **periods**.

Periods can most easily be thought of as business years (but they can be any regular, calendar-based time interval). Each period may have a different set of parameters. Periods have a natural sequence, and output from one period can be transferred as input to the next period. The number and length of periods is defined in the parameterization. If all periods are of equal length this may be defined in the model. *different lengt?*[1]
Schematically, for a simulation with m iterations and a parameterization defining n periods:

simulation

	period 1	period 2	...	period n
iteration 1	...	...	...	...
$\vdots$	$\vdots$	s_{ip}	$\vdots$	$\vdots$
iteration m	...	...	...	...

Simulation

In order to execute a simulation, the user has to select a risk model, a parameterization and a result template.

Iteration

screen[2]
deterministic models[3] *stochastic models*[4]

- The number of iterations required depends upon the statistical key figures to be evaluated. Generally speaking, more iterations are required for distributions with fatter tails, and/or evalutaions of higher/lower quantiles.

- A simulation of a DeterministicModel is called *calculation.* It contains one iteration only. It's results are deterministic and not stochastic.

[1]TODO: ; if they are of different length ...
[2]TODO: shot
[3]TODO: how to use them???
[4]TODO: how to use them???

Period

For each period, every component executes the following methods exactly once, after all information from its prerequisite components is available:

- doCalculation() evaluates incoming information and parameters and prepares outgoing information

- publishResults() sends the produced information to following components or collectors

- reset() prepares the component for the next period

Results

The result is a set of simulation, iteration, period, path, field, value.

9.3 Component

Components are the *building blocks* of models. Most of the business logic is kept in components. Components have typed interfaces which receive and send *packets* from and to other components and parameters. Thinking of a model as a *directed graph*, components are the *nodes* in such a graph. An *edge* links a sender interface to a receiver interface of another component.

Advice

- Try to keep components simple; a component should do one thing, not many things. This makes it easier to test it and also increases the chances that it can be re-used. Better buiding two components running after one other than only one.

- Before starting to implement any component, draw the directed graph with pencil and paper. Specify where specific information is required and produced. Once the directed graph can be mentally traversed, it is time to start coding.

Concept

- The method doCalculation() is executed once only per iteration and period. Therefore no loops are possible within a single component.

- Execution is triggered by the framework: If all wired interfaces (properties of type PacketList starting with in) have received their information, doCalculation() is executed. Afterwards, packets to be sent to following components are available in the out properties.

- Content of in and out properties is not shared between different iteration paths and periods. Once the framework has sent outgoing information to following components, the in and out property lists are cleared.

Implementation Details

- All components are derived from the class org.pillarone.riskanalytics.core.components.Component

- Use helper methods such as isSenderWired(PacketList sender) to check if a component will receive information in a specific model or isReceiverWired(PacketList receiver) to check if a following component or a collector is wired to an out property.

9.3.1 Conventions

Conventions make a developer's life easier by reducing the amount of code to be written and increasing its readability. In `RiskAnalytics`, conventions are used to generate the user interface and database scheme.

Naming convention

Beside clean code and greater readability, naming conventions increase coding speed in modern IDEs (integrated development environments) which support code completion.
A component may have several properties. Component property naming employs four prefixes:

- parm: whenever a variable name starts with parm, it is displayed on the parameterisation user interface automatically. A parm property may have one of the following types: int, double, String, DateTime, enum.
- in: all in variables are of type PacketList. Once they are wired they receive packets from other components.
- out: all out variables are of type PacketList and can send packets to other components. It is displayed on the result template user interface and on the result page if the property was collected. The collection mode can be specified in the result template.
- sub: a component may have subcomponents. Their name has to start with sub. They are helpful to provide building blocks and introduce a hierarchy.

Note: *If a component is written in Java, variables starting with any of these prefixes need a public setter and getter method. In Groovy a variable declared with no access modifier generates a private field with public getter and setter (i.e. a property). Therefore it is not necessary to write any getter or setter methods with default behaviour in Groovy.*

Grouping of Components and Parameters in the GUI

The order of parameters, out channels and subcomponents within a component determines the order in the GUI.
`RiskAnalytics` comes with a powerful way of deriving a default user interface for all models, components and its parameters – a model or component developer does not have to write any GUI code. Yet he can simply group the parameters.

9.3.2 Composed Component

To prevent modelers from always having to start from scratch, the concept of *composed components* enables a degree of re-use by allowing to define building blocks consisting of several components. Typical building blocks may be for example lines of business or a reinsurance program. *A composed component is very similar to a model, except that it can't be executed.*[5] The following three sections describe three different scenarios for composed components.

[5]TODO: explain

- static building blocks containing a *fixed number* of *different* component types
- dynamically composed components containing an *arbitrary number* of *equal* components. They are data-driven in the sense that the exact number of subcomponents is specified in the data and not in the model itself. Think of them as containers containing an arbitrary number of predefined subcomponents, wired according to specific rules. Examples: a reinurance program with an arbitrary number of contracts, an arbitrary number of lines of business or claims generators.
- multiple calculation phase components. This concept is new and its API not yet stable.

Static Building Blocks Whenever several components are always used in the same manner or additional hierarchy levels would facilitate the understanding of a model, a ComposedComponent is the appropriate solution. Composed components may be nested. *Furthermore a ComposedComponent may contain one or many DynamicComposedComponent.*[6]

[6]TODO: please explain

Chapter 10

Modelling claims

Using `RiskAnalytics` usually starts with either selecting an existing or creating a new parameterization. If you start `RiskAnalytics` for the first time, there are already several models with illustrative parameterizations made available for an easy introduction to the software.[1] Within a parameterization, there are several subsections to be addressed before you can run your first simulation (cf. Chapter 5.1 for an overview) such as the definition of (in order of appearance in the parameterization tree):

- underwriting information by use of **Underwriting Segments**†;
- the generation of claims through **Claims Generators***;
- the development of incoming reserves at the beginning of a period through **Reserve Generators***;
- **Dependencies*** between two ore many claims generators;
- **Event Generators*** allowing the simulation of events for per-event reinsurance cover or for events affecting several claims generators; and
- **Lines of Business**† as a means of grouping the underwriting segments, claims and reserves, the

[1] If you produce any teaching material – including parameterizations, it would be warmly appreciated if you share your insights and teaching material (and not your company's secrets) with the community: contact@pillarone.org

- **Reinsurance** Program (cf. Chapter 13) reflecting the applicable reinsurance coverage on (parts of) the portfolio, and
- **ALM Generators** (cf. Chapter 15).

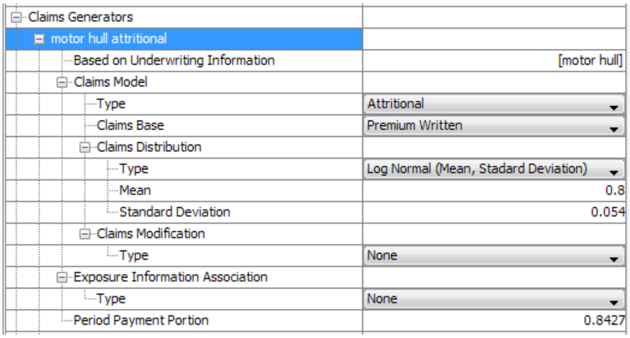
In this chapter we will focus on the claims-related parameters (marked with *), whilst in Chapter 12 we will focus on exposure-related parameters (marked with †).
Being intertwined with claims – but from previous periods – we also have a close look at the stochastical development of claims reserves. Please find more details in Chapter 14 about **Modelling Non-Life Reserves** on page 149 ff.

10.1 Claims Generators

Given an empty parameterization, or no **Claims Generator** respectively, you first have to define a **Claims Generator** by right clicking and selecting **add** in the context menu.
`RiskAnalytics`'s claim 'A claim is more than a number' can be proven easily. Whilst it is possible to generate different claim types – such as attritional, single or event – each claim consists of a value and a date. Ceded claims keep a reference to the original claim.
Furthermore event claims have a reference to the event they belong too.
The different claim types are required for reinsurance modeling as different contracts require different levels of information to calculate correctly the ceded part.
A claims generator consists of a – preferably – descriptive name (e.g. 'Motor Hull') and four elements, namely

- Based on Underwriting Information,
- Claims Model,
- Exposure Information Associaton, and
- Period Payment Portion,

as beeing explained in the following sections.



Claims Generators	
motor hull attritional	
Based on Underwriting Information	[motor hull]
Claims Model	
Type	Attritional
Claims Base	Premium Written
Claims Distribution	
Type	Log Normal (Mean, Stadard Deviation)
Mean	0.8
Standard Deviation	0.054
Claims Modification	
Type	None
Exposure Information Association	
Type	None
Period Payment Portion	0.8427

Figure 10.1: Claims Generator: Motor Hull Attritional

10.1.1 Underwriting Information

As your first selection, you have to decide on which **Underwriting Information** your claims generator will be based[2]: As a starting point, you will usually select one underwriting segment per claims generator and vice versa. Given the situation, where you have data available for two similar underwriting segments – such as 'motor tpl own business' and 'motor tpl co-insured business', both with the same characteristics but internally treated separately for regulatory reasons – you can use the same claims generator for both of them. This will reduce the maintenance of your model as there will be less parameters to update and less time needed for the simulation procedure.[3] Essentially, any Underwriting Segement as setup within the given parameterization (cf. Section 12.1) and an 'additive' combination thereof is a valid selection for basing a claims generator on it.
Having a look at a simulation from the other end, you will find the following components being usually requesters of underwriting information:

- claims generators, as their calibration may be based on sum insured, premium, or number of policies,

[2]It is also possible to run a simulation for claims generators and reinsurance only. However, if you want to calibrate the claims generator not only by **absolute** values but rather by **premium** or similar, this information must be provided here.

[3]However, this simplification will add a maybe unwanted dependency between the two underwriting segments. Better creaty a duplicate of the first claims generator by right-clicking on the existing claims generator.

- frequency generators – as an element of claims generators – in case being calibrated based on number of policies,
- allocation of sum insured to claims,
- proportional reinsurance contracts where they need gross underwriting information in order to calculate the ceded underwriting information, or
- working/cat excess of loss and stop loss reinsurance contracts where as the ceded premium may be specified as GNPI, rate on line or number of policies.

10.1.2 Claims Model

Depending on the application, a Claims Model **Type** from Table 10.1 can be selected. The Claims Model Type defines the form of the claims model; the names and structure of subordinate parameters depend on it. The default value **None** provides an empty Claims Generator, producing no claims.
The intent of the **none** type is to easily toggle a claims generator on or off, e.g. in case you want to use it in the first period but not in the second period of a multi-period model.

None

This selection is useful for quickly de-activating an already defined claims generator e.g. for setting up a multi period model where a specific claims generator is active only during a limited number of periods.

Attritional

Modelling 'attritional' claims is started by selection of the claims base as measure of calibration of the simulated claims. This calibration directly refers to the underlying underwriting information stored in **Underwriting Segments**. Let us choose to scale our claims by **sum insured** where the non-scaled claims distribution is given by a lognormal distribution LogNormal$(0.8; 0.054)$. The selection above will result in non-scaled claims with mean 0.8 and standard deviation 0.054.
The scaled claims, as defined by multiplying a given constant – such as the sum insured in this example – with the non-scaled claims, are again

Claims Model Type	(Additional) **Input Fields**	**Example**
None	*Disabled*	*No Claims produced*
Attritional	Claims Base	Premium Written
	Claims Distribution	Poisson(λ)
	Claims Modification	Truncated (min, max), Shift
Attritional with Date	*with a random incurred Date*	Uniform($0.4, 0.8$)
Frequency Average Attritional	Frequency Base	Number of Policies
	Frequency Distribution	Binomial(n, p)
	Frequency Modification	Censored (min, max)
Frequency Severity	*no additional fields*	
Frequency Severity with Date	Occurence Distribution	Normal(μ, σ)
	Produce Claim	Single Claims
Severity of Event Generator	*no additional fields*	

Table 10.1: Claims Model Types

lognormal distributed with parameters LogNormal($0.8 *$ Sum Insured; $0.054 *$ Sum Insured).

Assuming we will have to update our model some months later, as a first step, we might want to update just the sum insured amount, automatically getting a model based on both the updated sum insured amounts, to reflect a growing portfolio, as well as the parameters selected during the last exercise. Assuming again that we now want to update not only the sum insured, but also the mean of the distribution, which we found to be too low by 3%, selecting **Shift** as **Claims Modification** will allow us to shift the claims manually based on our calibration exercise.

The parameter used for shifting will modify the distribution before being applied to the scaling measure; i. e., for a given shifting parameter a, $a = 0$ will result in no shift, while $a \neq 0$ shifts the generated claims either positively or negatively. In descriptive words: claims generator$_{shift_a}$ = (claims distribution + a) $*$ scaling measure: Assuming a sum insured of EUR 1bn, the claims distributed as specified above and now shifted by a factor of $a = 0.8 * 0.03 = 0.024$ would inflate the simulated claims C: $C = \text{LogNormal}(0.8 + 0.024; 0.054) * \text{EUR 1bn}$. An overview and explanation of all possible selections is given in the table on page 103.

Attritional with Date

Whereas attritional claims are per default modelled as one aggregated (i. e. grouped) claim occuring in the middle of the period (e.g. 30 June of a standardized year), the user has the possibility to select a statistical method for allocation of a date within the time period to the simulated claim. In many situations it is important to know, when a claim occurred, namely for reinsurance contracts incepting during the period and therefore not covering the whole period as being modelled. For seasonal business such as a hail cover in Europe it might make sense to select Uniform$(0.4; 0.8)$ resulting in one attritional claim occuring – based on the selected claims distribution – sometime between May and October. If you setup a one-year reinsurance contract with inception date 1. January, all claims occuring during this period will be covered by the reinsurance contract. But if you setup a reinsurance contract with inception date on 1. October, the reinsurance will not cover the hail claims in each and every iteration.

Frequency Average Attritional

This selection is used for modelling claims with more details than 'just attritional' and less than a classical frequency-severity model as it still aggregates to claims to being one. While the former ('attritional') results in one attritional claim per time period, and the latter ('frequency severity') in several claims with different severities, the selection here is used for modelling one attritional claim as specified above but by using a different kind of parameterization. The user might select a constant distribution of 5 resulting in one attritional claim being parameterized by the number (frequency) of five claims and the severity as specified.

Frequency Severity

A very flexible claims generator is made available through this option: In addition to specific modelling of both, frequency and severity of the claims, the user has the option to group the simulated claims to one event selecting **Produce Claim** as **Aggregate Event Claims**; this has an impact on some reinsurance programs.

Frequency Severity with Date

In addition to the already explained claims model **Frequency Severity** the hereby explained model allows the modelling of assumptions around the occurence of the claims. As default, a uniform$(0; 1)$ distribution is applied, resulting in equally distributed claims over the complete time period. Alternatively, for example,
Piecewise Linear$((0; 0.25; 0.5; 1); (0; 0.33; 0.66; 1))$ would result in an occurence pattern with no occurences from January to March, 33/100 of claims occuring between April and June, another 33/100 between July and December and the remaining 34/100 on the latest day of the (standardized) year. Again, this is important to know namely for reinsurance contracts incepting during the period and therefore not covering the whole period as being modelled.

Severity of Event Generator

Severity of event generator is selected in case you want to first simulate severities by use of the **Event Generator** (cf. the section on **Event Generators** on page 107 for more details) and afterwards apply them to – usually – several underwriting segments: By using this approch, the **Event Generator** (cf. Section 10.4) will firstly draw one or several event as specified in your parameterization whereof a percentile is drawn. This percentile then gets applied to the **Claims Distribution** here.

10.1.3 Exposure Information Associaton

As in some cases the size of a claim is not sufficient for the following depending calculations, exposure information, like the sum insured (SI) and the probable maximum loss (PML) can be attached to the claim. Especially the surplus reinsurance cover requires to know pairs of exposure and claim size to calculate the coverage.[4] Except **None** there are two exposure information association mechanisms available: **Risk to Band** and **Sum Insured Generator**; all of them explained further below.
In principle, there are many different ways of allocating claims to risk bands. `RiskAnalytics` offers two different approaches, one of which is applied to aggregate losses and the other to single losses. In both approaches, the allocation uses a target allocation that specifies for each risk class the percentage of claims that should be associated with the given risk class. In the case of aggregate claims, the target allocation can always be easily met since aggregate claims can always be split up.

None

In case no exposure association to the claims is needed, this selection is appropriate. If, on the other hand, the exposure association is needed but not specified here, the portion of ceded risk is set to zero even if an instrument such as surplus reinsurance is specified further below in the parameterization.

Risk to Band

In case of single claims, the following modified procedure is applied: For an initial guess, the incoming (individual) losses are allocated to the risk classes by just referring to the claim size, i. e. the claim must

[4]Surplus reinsurance is a more advanced and more complicated form of proportional reinsurance where the portion covered by the reinsurer depends on the sum insured (or other quantities) specified for the underlying insured risks. The portion of the claim ceded to the reinsurer is given by the ratio of the risk ceded divided by the total risk.

This is a summary of names and entries for comboboxes as related to claims generators. Part I/II

Name, Combobox	**Description**
Type	**Selection of Claims Model Type**
None	No claims are generated.
Attritional	One attritional claim is generated occuring in the middle of the time period: 1 claim occuring on 30 June of a standardized year.
Attritional With Date	One attritional claim is generated occuring during the time period based on a specified distribution: 1 claim per period with randomly distributed occurrence date (attached to the claim for further use!).
Frequency Average Attritional	One attritional claim as per the specified parameters is simulated.
Frequency Severity	Distributions for both, frequency and severity, are specified.
Frequency Severity with Date	Frequency and Severity model including distribution of the occurrence. Dates are attached to the claim for further use.
Severity of Event Generator	Please see the sections on **severity of event generator** (page 101) and on **Event Generators** (page 107).
Frequency Base	**Specification of multiple to the mean of the specified claims frequency**
Absolute	The mean of the distribution is pre-determined as absolute value;
Number of Policies	or by the number of policies.
Frequency Distribution	**Please refer to the SSJ Library**

Table 10.2: Claims Model: Summary and Explanations of Comboboxes Part I/II

This is a summary of names and entries for comboboxes as related to claims generators. Part II/II

Name, Combobox	**Description**
Frequency Modification	**Applying a modification to the claims generator *before* multiplying with the Frequency Base**
None	No modification to the selected frequency distribution.
Truncated	Cutting the distribution at a specified minimum and maximum value respectively. No observations outside the given range are observed. Applying a deductible is an illustrative example for truncation at the lower end of the distribution: lower claims are not registered.
Truncated, Shift	Combination of both methods by first truncating and afterwards shifting the distribution.
Censored	Censoring the distribution at a specified minimum and maximum value respectively. Any observation outside the given range is 'rounded' to lie within the range. Applying a maximum sum insured is an illustrative example for censoring at the higher end of the distribution: higher claims are cut to the maximum sum insured.
Censored, Shift	Combination of both methods by first censoring and afterwards shifting the distribution.
Shift	Additive shifting of the mean of the distribution by the parameter.
Claims Base	**Specification of multiple to the mean of the specified claims severity**
Absolute	The mean of the distribution is pre-determined as absolute value;
Premium Written	or by written premium,
Number of Policies	by number of policies, or
Sum Insured	by sum insured.
Claims Distribution	**Please refer to the SSJ Library** Discrete Empirical (cumulative) allows to load external information such as RMS-tables.
Claims Modification	**cf. above *Frequency Modification***

Table 10.3: Claims Model: Summary and Explanations of Comboboxes Part II/II

fit in between the lower and the upper limit of the risk band. Based on the selection such as **Premium**, an algorithm re-alocates the single claims to higher risk bands where needed in order to allow a best-fit to the weight per band as set by **Premium** (in this example).

Sum Insured Generator

The sum insured generator produces the required exposure information using the claim and the maximum sum insured. The claim (X_i) is used as generated from the claims generator, the maximum sum insured or PML is taken from the appropriate underwriting information. The sum insured is calculated using the following formula $SI_i = a(PML - X_i) + X_i$. The random distribution a, usually $a \in [0, 1]$, is configurable by the parameters of the sum insured generator.
Furher details on surplus reinsurance are explained in Section 13.2.2 on page 135.

10.1.4 Period Payment Portion

The parameter period payment portion p is part of the reserve model (cf. Section 10.2). The share $p \in [0, 1]$ is the part of a generated claim which is paid during the current period. The share $1 - p$ is shown as reserved at the end of the period.

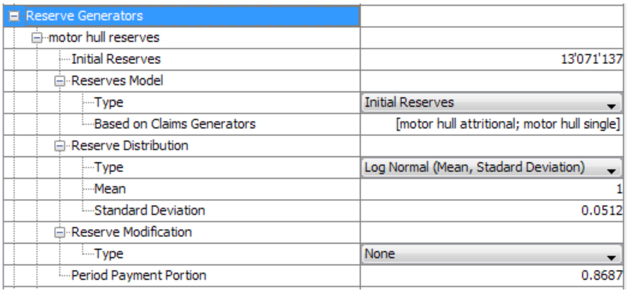
10.2 Reserve Generators

The reserve generators provide an additional option to model random effects of IBNyR/IBNeR. As a volume parameter the initial reserve volume can be put in. The reserve generator allows for the *types listed below*[5].
More details are explained in the specific chapter on page 149 ff.

10.3 Dependencies

This is a short overview only. For more details, please refer to chapter 11 on pages 109ff.
Given a portfolio with two underwriting segments, for instance **motor hull private** and **motor hull commercial**, dependencies in-between

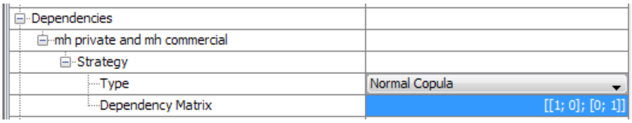
[5]TODO: tbd



Reserve Generators	
motor hull reserves	
Initial Reserves	13'071'137
Reserves Model	
Type	Initial Reserves
Based on Claims Generators	[motor hull attritional; motor hull single]
Reserve Distribution	
Type	Log Normal (Mean, Stadard Deviation)
Mean	1
Standard Deviation	0.0512
Reserve Modification	
Type	None
Period Payment Portion	0.8687

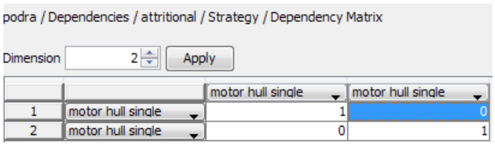
Figure 10.2: Reserve Generator

these two underwriting segments will occur: By selecting a dependency for two or more underwriting segments, all **attritional** and/or **attritional with date** claims will be modelled with the specified dependency.



Dependencies	
mh private and mh commercial	
Strategy	
Type	Normal Copula
Dependency Matrix	[[1; 0]; [0; 1]]

Figure 10.3: Dependencies



podra / Dependencies / attritional / Strategy / Dependency Matrix

Dimension 2 Apply

		motor hull single	motor hull single
1	motor hull single	1	0
2	motor hull single	0	1

Figure 10.4: Dependency Matrix

Whilst such dependecies are usually not linear, they are modelled by use of copulae belonging to these families:

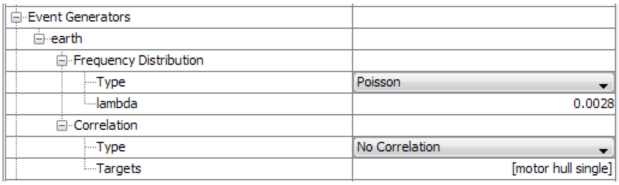
- **Normal** (Gaussian), specified by a dependency matrix
- **Frechet upper bound**, applied to a target, i. e. a claims generator
- **No Correlation**, i. e. independent targets
- **T-Copula**, specified by the degrees of freedom and a dependency matrix

- **Gumbel**, specified by parameters Lambda, the dimension and the targets

Whilst further details on how to use dependencies in `RiskAnalytics`are given in the respective chapter on page 109 ff..Details on the implementation of these copulae can be found here www.link.me\please; details on how to calibrate the dependency structure given a specific portfolio are usually provided together with spezialized calibration software. `PillarOne RiskAnalytics` releases come with parametrizations that furnish examples of companies having dependencies between selected underwriting segments.

10.4 Event Generators

Event Generator vs Event Correlation in MCM[6] An insurance contract always specifies the covered risk: Some reinsurance contracts cover the complete balance sheet of a company (such as Stop Loss), others only cover one specific building or vehicle (such as facultative reinsurance cover for the construction of a tunnel). XL treaties usually either are specified as **per risk** or **per event**. In the latter case, an often high number of – relatively – small claims lead to a large loss for the insurance company; however, none of these small claims would be covered by reinsurance separately. So we would have to select a claims generator with the option **Produce Claims: Aggregated Event**.



Event Generators	
earth	
Frequency Distribution	
Type	Poisson
lambda	0.0028
Correlation	
Type	No Correlation
Targets	[motor hull single]

Figure 10.5: Event Generator

If you select an **Event Generator** it will simulate the severity of an event by determining the percentile – as number between 0 and 100. If for example, the percentile of 84% is drawn, this percentile is fed to the claims generators where the option **severity of event generator** is selected. Depending on the loss distribution for each underwriting

[6] TODO: todo

segment the loss amount will be calculated depending on the specified percentile.[7]

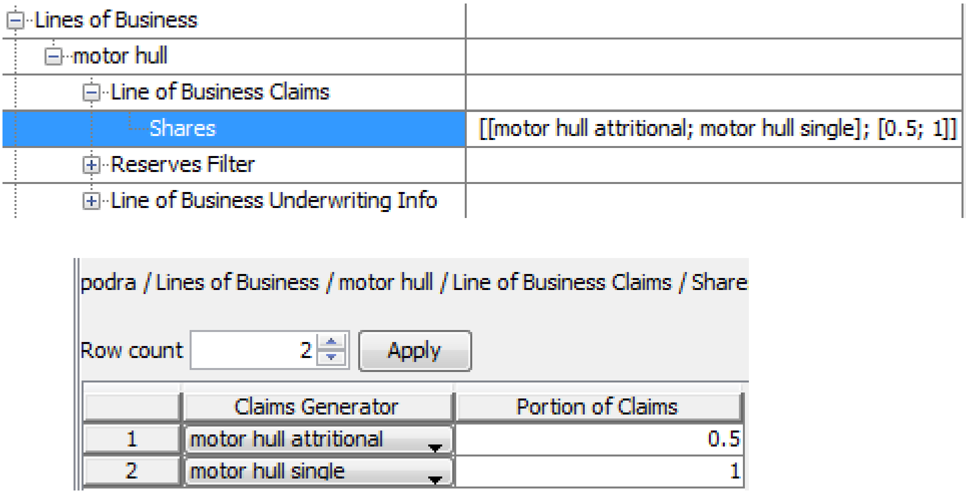
[7] TODO: This section needs rewording and more explanation. the first paragraph is really misleading!

Chapter 11

Dependency Modelling

11.1 Scaling and allocating claims

If an attritional claim is simply a fraction of another claim, then this is a dependency structure, albeit a very simple one. Of course, we could explicitly re-scale the claims distribution and capture the new distribution parameters. But for various reasons this may not always be the best solution. If for example we get market loss distributions from an external provider, then for review and audit reasons we capture these original parameters and scale the claims to the appropriate scale of the portfolio. Another reason may be that the same claims source is used with different scalings in different parts of a model. And a third application is if one quickly wants to perform a simple sensitivity analysis with respect to a claims distribution.
Figure 11.1 shows that scaling factors are entered in the multi-dimensional parameter called Share. This multi-dimensional parameter is used to associate the claims generators to the lines of business and set their scaling factors. The term Share suggests another use of this scaling: Parts of a claims can be allocated to different lines of business. This may be necessary if policies cover multiple risks, but the historic data may not be sufficient to calibrate claims distributions for each risk which are covered by the policy. In the cases in which we talk about allocating claims to multiple lines of business their shares usually add up to 1. This is not the case if for example a market loss is scaled. In fact, in some cases scaling factors which are greater than 1 may make sense. Hence, there is no validator which checks that the shares add



Lines of Business	
motor hull	
Line of Business Claims	
Shares	[[motor hull attritional; motor hull single]; [0.5; 1]]
Reserves Filter	
Line of Business Underwriting Info	

podra / Lines of Business / motor hull / Line of Business Claims / Share

Row count 2 Apply

	Claims Generator	Portion of Claims
1	motor hull attritional	0.5
2	motor hull single	1

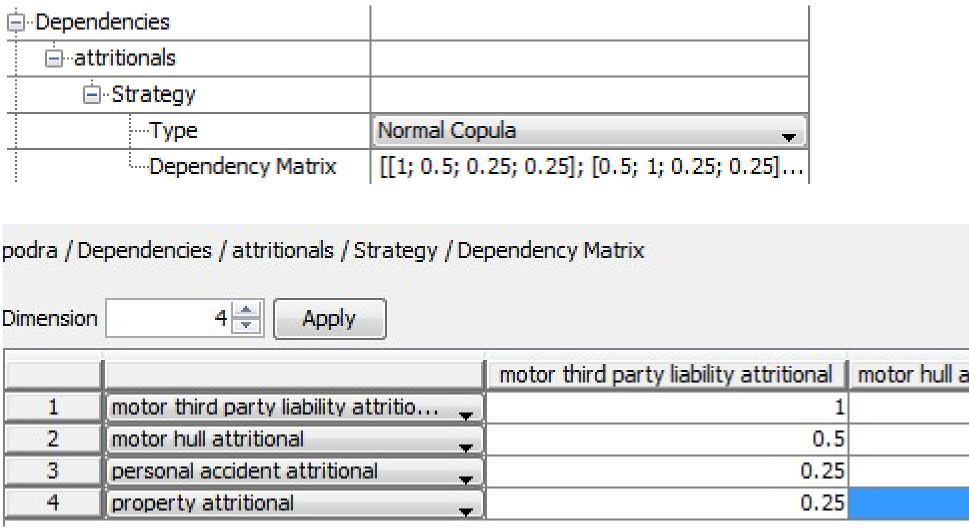
Figure 11.1: Scaling or allocating claims

up to 1 and/or each share is less than 1.

11.2 Dependency models for attritional claims

Copulas offer a way to construct joint distributions with given marginals. This is a frequently encountered problem in actuarial work. If one cause leads to several different claim types, e. g. property damage and fire following, then there exists a joint distribution of these claims. If we fit claims distributions to the claims, one per claims type, then these are the marginal distributions of an unknown joint distribution. Hence, to sample from these marginal in a correlated way, we need to construct a joint distribution.
For an introduction to mathematical foundations of copulas see [14]. For a discussion more focused on insurance risk applications we refer the reader to McNeil et al article in [13]. It is important to notice that there are also other ways to model dependencies, e. g. causal dependencies. *Impact*[1]
`RiskAnalytics` offers different copulae (compare Figure 11.3 further below):

[1]TODO: please explain



Dependencies	
attritionals	
Strategy	
Type	Normal Copula
Dependency Matrix	[[1; 0.5; 0.25; 0.25]; [0.5; 1; 0.25; 0.25]...

podra / Dependencies / attritionals / Strategy / Dependency Matrix

Dimension 4 Apply

		motor third party liability attritional	motor hull a
1	motor third party liability attritio...	1	
2	motor hull attritional	0.5	
3	personal accident attritional	0.25	
4	property attritional	0.25	

Figure 11.2: 4-dimensional normal copula

- Comonotonic or the Frechet upper bound, i. e. the maximal possible correlation. This corresponds to drawing from all claims distributions a claim which corresponds to the same percentile. This copula has no parameters.
- Normal
- Student t
- Gumbel

It is not hard to add other copulas. Let us know which ones you use and we will add them to the library. For a copula model we always have to specify its parameters and which claimstributions should be correlated, i. e. the marginal distributions. Some copulas may have no parameters.

It is beyond the scope of this manual to discuss which copula model can be applied in which situation and for what reasons. As demonstrated in Figure 11.3, the choice of copula has a major influence on the dependency structure and needs careful consideration. Figure 11.2 shows a normal copula which correlates four attritional claims generators.

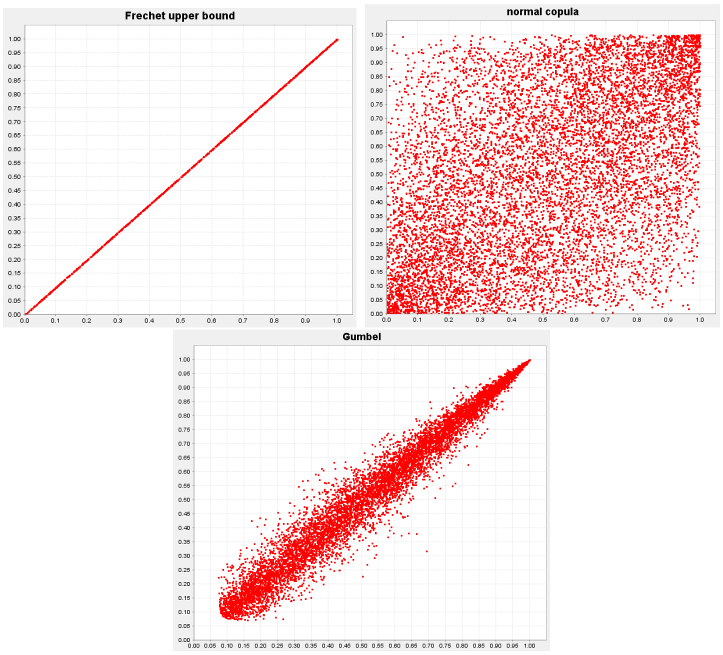


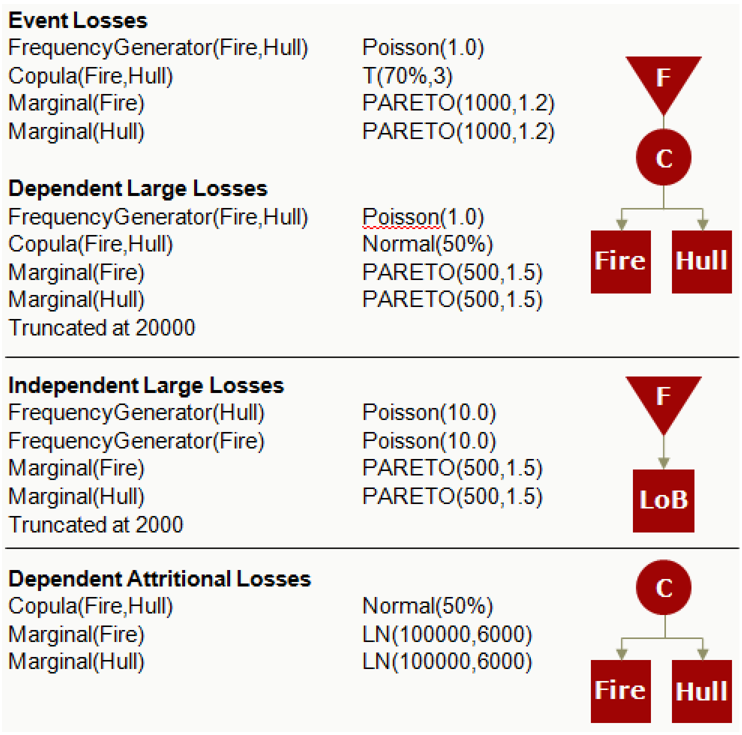
Figure 11.3: Comparison of comonotonic, normal and Gumbel copulas

11.3 Dependency models for single claims

If we are working with a frequency-severity model to produce single claims, then the most obvious way to correlate these single claims is to postulate a common cause model. In other words, we assume that the claims stem from the same underlying events. This does not imply that the claim sizes must be correlated; but usually this will be the case. Hence, for each iteration we can draw one claims frequency per simulation period for all correlated single claims. And for each of the claim events we can proceed as in the attritional claims case.
It is worthwhile noting that even if some single claims are correlated that does not force us to correlate all single claims. For example, if you work with Poisson distributed claims frequencies, then it is easy to split the frequency into a common event frequency and an idiosyncratic claims frequency. The two average frequencies, the parameters of the Poisson process for the event and the idiosyncratic process, have to add up to the total average frequency. This holds because the sum of two Poisson processes with parameters λ_1 and λ_2 is again a Poisson process with parameter $\lambda_1 + \lambda_2$.

11.4 A simple dependency example

The following example shows how the above discussed dependency models can be combined in a practical case. It is worthwhile noting that this is a simple example and yet, the resulting correlation structure is already fairly complex. We do not mean to discourage modellers from using such dependency models. But their structure and the calibration of their parameters needs to be carefully reviewed and validated. The limitation of dependency modeling is definitely not a technical one. What can be modelled in RiskAnalytics by far exceeds what most actuaries can soundly explain and validate. Hence, the slogan for dependency modeling should definitely be: Keep it as simple as possible.
Let us assume that we have the two lines of business property and motor hull. For each of these lines of business we model independent large losses using a frequency severity model with Poisson frequency and Pareto severity. The parameters can be seen in the section 'Indenpendent LargeLosses' in Figure 11.4. For the large claims which stem from catastrophic events we assume a common cause model with a Poisson frequency and Pareto severities which are correlated using a



Event Losses	
FrequencyGenerator(Fire,Hull)	Poisson(1.0)
Copula(Fire,Hull)	T(70%,3)
Marginal(Fire)	PARETO(1000,1.2)
Marginal(Hull)	PARETO(1000,1.2)
Dependent Large Losses	
FrequencyGenerator(Fire,Hull)	Poisson(1.0)
Copula(Fire,Hull)	Normal(50%)
Marginal(Fire)	PARETO(500,1.5)
Marginal(Hull)	PARETO(500,1.5)
Truncated at 20000	
Independent Large Losses	
FrequencyGenerator(Hull)	Poisson(10.0)
FrequencyGenerator(Fire)	Poisson(10.0)
Marginal(Fire)	PARETO(500,1.5)
Marginal(Hull)	PARETO(500,1.5)
Truncated at 2000	
Dependent Attritional Losses	
Copula(Fire,Hull)	Normal(50%)
Marginal(Fire)	LN(100000,6000)
Marginal(Hull)	LN(100000,6000)

Figure 11.4: Typical dependency structure for two lines of business

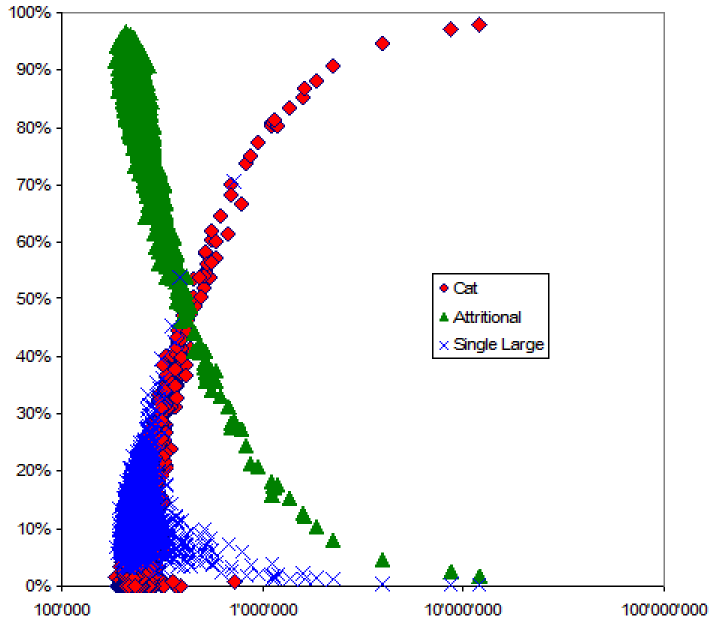


Figure 11.5: Split of the aggregate claims into the three risk types. On the horizontal axis is the claim size in monetary units and the vertical axis is the contribution of each claims type to the aggregate.

student-T copula. Finally, for the attritional claims we use lognormal distributed claims which are correlated using a normal copula.
This is an absolutely standard dependency structure and the marginal distributions as well as the copulas are not exotic. The resulting aggregate claims have a complex structure as shown in Figure 11.5. For small aggregate claims there are hardly any cat claims involved. With increasing size of the aggregate claims the contribution of the single, non-cat claims and the attritional claims decrease.

Chapter 12

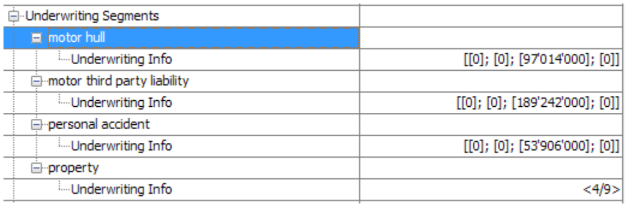
Modelling exposure

Whilst we started our focus on modelling exposure and claims by firstly looking at the claims generators in chapter 10 (this makes sense, as in the real world, claims occur even without being exposed to insurance contracts), we will now learn how to model the exposure of an insurance company by use of a simple two-level hierarchy. The first level of the hierarchy consists of **Underwriting Segments** and the second of **Lines of Business**. The former consists of the lowest available level of underwriting information, such as *motor hull* or *motor tpl*; the latter either consists of one line of business only or of a group of lines of business: *motor* = *motor hull* ∪ *motor tpl*. Depending on the granularity of the available data, underwriting segments might be granular to any level you can dream of: type of business, currency, region, legal entity, etc. It is your professional judgement on the data available on which level you want to run simulations. Modelling every single contract will most often be too granular, modelling the whole company only will most often be too general.[1]

12.1 Underwriting Segments

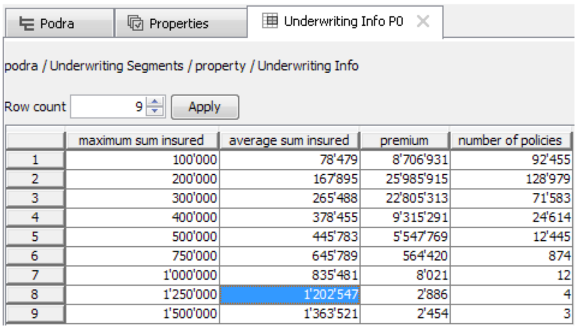
For the sake of convenience, a new underwriting segment can either be built by right-clicking on **Underwriting Segment** and selecting **add** – resulting in an empty underwriting segment – or by right-clicking

[1]Both extremes are existing in the real life: A reinsurer covering a niche might model contract by contract; a relatively small legal entitiy within a relatively large group will be modelled as one underwriting segment.



Underwriting Segments	
motor hull	
Underwriting Info	[[0]; [0]; [97'014'000]; [0]]
motor third party liability	
Underwriting Info	[[0]; [0]; [189'242'000]; [0]]
personal accident	
Underwriting Info	[[0]; [0]; [53'906'000]; [0]]
property	
Underwriting Info	<4/9>

Figure 12.1: Underwriting Segments



podra / Underwriting Segments / property / Underwriting Info

Row count 9

	maximum sum insured	average sum insured	premium	number of policies
1	100'000	78'479	8'706'931	92'455
2	200'000	167'895	25'985'915	128'979
3	300'000	265'488	22'805'313	71'583
4	400'000	378'455	9'315'291	24'614
5	500'000	445'783	5'547'769	12'445
6	750'000	645'789	564'420	874
7	1'000'000	835'481	8'021	12
8	1'250'000	1'202'547	2'886	4
9	1'500'000	1'363'521	2'454	3

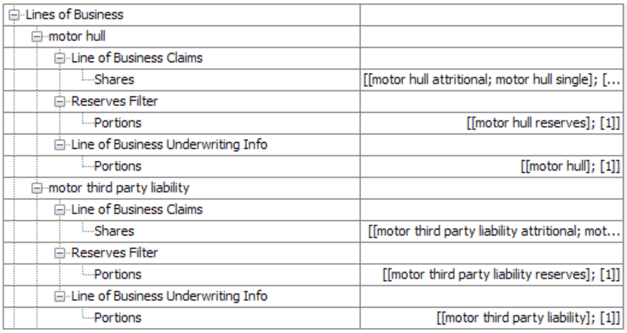
Figure 12.2: Underwriting Segments: Details for Property shown in a 4/9 matrix above

an existing segment and selecting **duplicate**: The latter will copy all information from the 'parent' underwriting segment to the 'child' being saved under a new name.

The underwriting information contains per segment: *maximum sum insured*, *average sum insured*, *premium* and *number of policies*. It is possible and often appropriate to edit serveral risk bands: These risk bands will be used for surplus share treaty reinsurance (cf. Section 13.2). It is important to remember these risk bands are directly interlinked to each other as shown in the following table:

Note: *Therefore, it is not possible to 'group' a heterogenious portfolio after the merger of two companies, even if one of them with higher insured amounts, the other with lower insured amounts etc. This would rather be achieved by building a Line of Business as explained in the next section.*

Double-clicking on the cell next to **Underwriting Info** opens a table where you select the number of bands needed and type in the criteria



Lines of Business	
motor hull	
Line of Business Claims	
Shares	[[motor hull attritional; motor hull single]; [...
Reserves Filter	
Portions	[[motor hull reserves]; [1]]
Line of Business Underwriting Info	
Portions	[[motor hull]; [1]]
motor third party liability	
Line of Business Claims	
Shares	[[motor third party liability attritional; mot...
Reserves Filter	
Portions	[[motor third party liability reserves]; [1]]
Line of Business Underwriting Info	
Portions	[[motor third party liability]; [1]]

Figure 12.3: Lines of Business

Risk band r_i	Minimum Sum Insured	Maximum Sum Insured
r_1	0	$\max_1$ = User Input
r_2	$\max_1$	$\max_2$ = User Input
r_3	$\max_2$	$\max_3$ = User Input
...	...	...

Table 12.1: Risk Bands

for each band:

- **maximum sum insured**: The upper bound of the range of sums insured. The lower bound is given by the upper bound of the previous class (in the previous row) or 0, cf. Table 12.1.
- **average sum insured**: The average sum insured of the policies included in the given risk class. By definition, the average should be smaller than or equal to the maximum sum insured of the given row, but larger than the maximum sum insured of the previous row.
- **premium**: The total premium of all the policies included in the risk band.
- **number of policies**: The number of policies included in the risk band.

Note that the risk bands have to be in ascending order.

12.2 Lines of Business

Lines of business are built by the union of one or several underwriting segments; exposing them to claims generators and reserve filters.
As now the risks are defined they may be combined or allocated to business segments. Adding a new line of business (right-click, as usual) opens three sets of parameters: Adding different claims generators (or shares[2] of them) to the dedicated Line of Business, as well as Reserve Filters (cf. Section 10.2). Finally, to each Line of Business the Underwriting Information is attached, completing the information needed in this area.
*If you want to further cascade the grouping, i. e. looking at **Non Life** business consisting of motor and fire, sticking with the above example, you have to setup a line of business on the level of the existing granular underwriting segments: Non Life = motor hull $\cup$ motor tpl $\cup$ fire. This allows separate modelling of company-wide effects such as inflation for underwriting segments in specific countries.*[3]

[2]The share might be greater than 1 in case you want to scale the distribution.
[3]TODO: Is this true?

Chapter 13

Reinsurance

The reinsurance components model the reinsurance contracts. The following discussion explains how the different claims model types (cf. Section 10.1.2) are processed by the reinsurance components. It is not meant to explain all the different features and options of reinsurance contracts. Here and there we make a comment about the effects of reinsurance, because these effects can easily be visualized in `RiskAnalytics`. But overall, we assume that the reader has a basic understanding of reinsurance. For details about reinsurance we refer the reader to [8, 11].
Each reinsurance contract has a strategy defining the contract type. Parameters common to all strategies are:

- Share covered by reinsurer, defining the share signed
- Inuring Priority, which is used if the user can add and remove contracts (see Section 13.4.2). It defines the order in which contracts are processed within a reinsurance programm. Claims and underwriting information are first processed by the contract with the lowest inuring priority. The resulting net is then processed by the contract with the next higher inuring priority. If several contracts have the same inuring priority, they are applied in parallel. Parallel contracts may be used in combination with "covered by reinsurer" if a contract is split among several reinsurers. Another use case consists of several XL layers.

The remainder of this chapter is structured as follows:

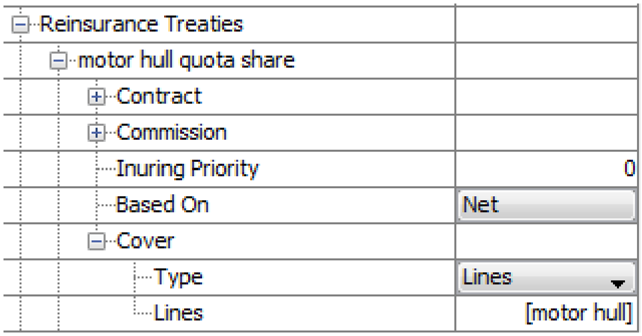
- General reinsurance paremeters being used for (almost) every reinsurance contract are explained in section 13.1, including an extensive part on commission
- Modelling Proportional Reinsurance follows in section 13.2, while
- Modelling Non-Proportional Reinsurance follows in section 13.3
- 13.4 are covered in the closing section of this chapter

13.1 General reinsurance parameters

Some parameters are common to all reinsurance contracts.

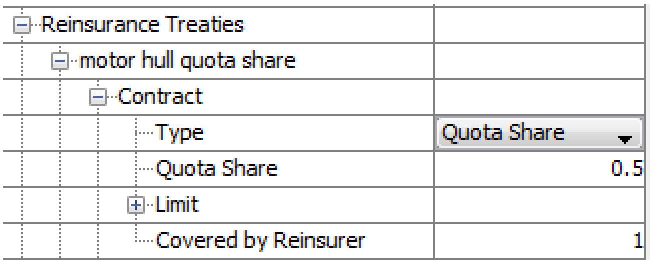
13.1.1 Associating claims, lines of business and reserves to reinsurance contracts

In some form or another we have to define which claims are covered and hence which claim generators should send their claims to the reinsurance component. The most obvious selection is to associate claim generators directly with a reinsurance component. For reinsurance components which cover reserves the analogue is to directly associate the reserve generators. But claim generators or reserve generators can also be associated indirectly with a reinsurance component by specifying which lines of business are covered by the contract. In this case reinsurance contract covers the claims and/or reserve generators which are associated to the covered lines of business. The lines of business and peril criteria can be combined with a logical **and**. This allows to set-up a reinsurance contract which only covers some perils from a line of business.
At first glance this looks as if chosing a line of business and a peril is identical to just chosing the peril. Indeed it is identical, but only if there are no preceeding reinsurance contracts which cover the same peril, i. e. the gross claims from this peril are covered by the reinsurance contract, and the perils are 100% associated to a line of business.
To specify what is covered by a reinsurance contract, we have to complete the section **Cover** in the reinsurance component (see Figure 13.1). Since a reinsurance contract can cover multiple perils or lines of business, these parameters are set in a multi-dimensional parameter. A typical situation is that the claims are modeled with an attritional, a



Reinsurance Treaties	
motor hull quota share	
Contract	
Commission	
Inuring Priority	0
Based On	Net
Cover	
Type	Lines
Lines	[motor hull]

Figure 13.1



Reinsurance Treaties	
motor hull quota share	
Contract	
Type	Quota Share
Quota Share	0.5
Limit	
Covered by Reinsurer	1

Figure 13.2

frequency-severity and a cat distribution and all of them are covered by the reinsurance contract.

13.1.2 Partially underwritten reinsurance contracts

A reinsurance contract needs not be placed 100% in the market. The fraction which is underwritten can be specified in the contract section of a reinsurance contract (see Figure 13.2). In the interface it is called **Covered by Reinsurer**. First the ceded claims are calculated as if this fraction were 1, i. e. 100% of the contract is covered by a reinsurer. After these contract specific-calculations are done, then the fraction is applied to obtain the ceded claims.
To reduce credit risk exposure towards reinsurers, reinsurance contracts are often placed with several reinsurers. If this credit risk is not modelled, then the sum of the parts covered by all reinsurers participating at a contract can be added up and used as the fraction covered by

reinsurer. Hence, if for example two reinsurers participate with 50%, then we can still model one reinsurance contract with a fraction of 1. If the credit risk of reinsurance is modelled, then this contract has to be modelled as two identical contracts – one in detail, the second by duplicating the first – with the same injuring priority, but each with the appropriate fraction from the corresponding reinsurer.

13.1.3 Contract basis

If the claims covered by a reinsurance contract are directly coming from a claims generator, then this parameter has no effect. But in the context of reinsurance programs a reinsurance contract has to define whether it covers the net or the ceded claims of the preceeding contract (see Figure 13.2*please check the figure*[1]).
Taking the net claims of the preceeding contracts as the gross claims for the next contract is usually done in primary insurance models where the insurer is interested in its net claims. Taking the ceded claims of the preceeding contracts as the gross claims for the next contract is typically done in retrocession models.

13.1.4 Injuring priority

Like the contract basis, the injuring priority is only used in the context of several reinsurance contracts, i.e. a reinsurance program. In a reinsurance program, the injuring priorities of the involved contracts defines the order in which the contracts are applied. It is a positive integer. Several contract may have the same injuring priority if they are to be applied in parallel; for example different layers of a WXL cover. In the following we first explain the contracts before we continue to structure reinsurance programs (see section 13.4) by means of the injuring priority.

13.1.5 Reinsurance Commission

In reinsurance, the primary insurance company usually pays the reinsurer its proportion of the gross premium it receives on a risk. The reinsurance commission is another factor that plays an important role in determining the price of reinsurance: The reinsurer allows the company a commission on gross premium received, large enough to reimburse the

[1]TODO:

company for the commission paid to its agents, plus internal administration expenditures, loss adjustment costs, taxes and its overhead[2]. The commission may be calculated by the reinsurer as the difference between premium and expected claim burden after deduction of his profit expectations.
Interpreting the commission as a reimbursement of the direct insurers expenses different strategies for designing reinsurance commission have evolved. In what follows we consider three different forms to calculate commission:

1. The *fixed commission* is the most common form of reinsurance commission obtained by a fixed percentage of the ceded written premiums,
2. the *profit commission* providing one way of taking the actual reinsured business into account, and
3. the *sliding scale commission* that is a way of encouraging the cedant to write profitable business just as the profit commission.

We also outline the bouquet commission as an integral part of the component **Reinsurance**, respectively **Reinsurance Market** for the 'Multi Company Model'. As the name implies the essential feature of this additional (sub)component is the application of one of the three commission strategies (fixed, profit, sliding) to a bundle of reinsurance treaties.

Fixed Commission

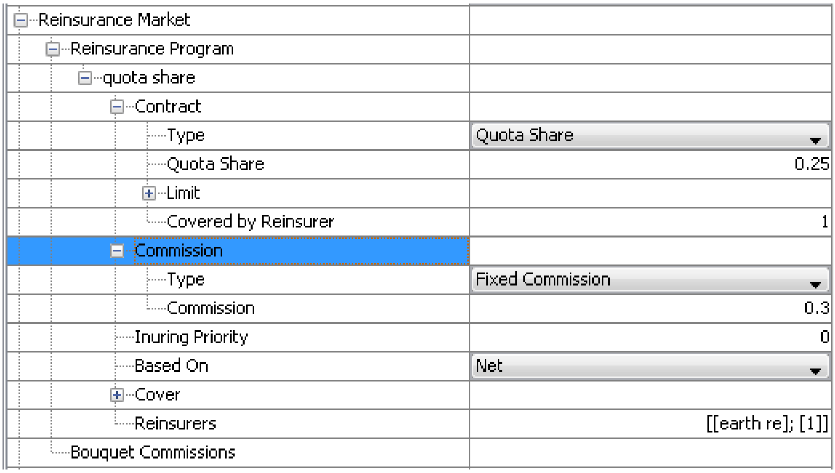
The fixed commission with its clear contractual agreements is easy to handle and allows for predictability in the primary- and reinsurer's planning process. With the predetermined (ceded or fixed) commission rate c_{fix} and the ceded premium written P_{ceded} the formula for the commission C_{fix} reads

$$C_{\text{fix}} = c_{\text{fix}} \cdot P_{\text{ceded}}. \tag{13.1}$$

The direct insurer's operating costs will be fully defrayed if the fixed commission rate matches the cost ratio of the primary insurer given by the ratio between costs and written premium.

[2]In this illustrative example, we do not consider economical strategies of both, the insurer as well as the reinsurer, by setting the commission in a way of optimizing the profit of each involved party, but rather assuming the commission shall be determined as fair means of sharing costs.

Example: A primary insurer and a reinsurer conclude a 25% quota share treaty in the accident line of business.[3] The written premium that the direct insurer obtains from the policyholders is 100 million, the operating costs are 30 million and the expected claim burden is 60 million. The primary insurer cedes 25% to the reinsurer who thus receives 25 million premium and must pay 15 million of the expected losses. The commission rate is determined to be 30%, hence 7.5 million is returned to the direct insurer as his commission. Due to consistency of commission and cost ratio, both contractual partners can expect a profit of 10% of the received premium, that is, $0.1 = 7.5/75 = 2.5/25$.



Reinsurance Market	
Reinsurance Program	
quota share	
Contract	
Type	Quota Share
Quota Share	0.25
Limit	
Covered by Reinsurer	1
Commission	
Type	Fixed Commission
Commission	0.3
Inuring Priority	0
Based On	Net
Cover	
Reinsurers	[[earth re]; [1]]
Bouquet Commissions	

Figure 13.3: Fixed commission strategy in "Multi Company Model"

Note: *In the* **`RiskAnalytics`** *parametrization form as illustrated in Figure 13.3 the user may find the parameter name* ***Commission*** *after selecting* ***Fixed Commission*** *for the Commission Type. In abuse of notation it should be clear from the context that the value that is associated to this parameter is the commission rate* c_{fix}.

Profit Commission

It is often felt to be inappropriate to have a fixed commission that is given independently of the actual profits and losses resulting from the

[3]Notice that this example strongly simplifies reality in that operating costs of the reinsurer are simply ignored.

(reinsurance) contract. If the results are good (and commission rate was below the actual direct insurer's costs), the direct insurer wants to participate in the profits by a more equitable share in the expenses leading to a partial reimbursement. On the other hand, if results are poor following from high losses the reinsurer may want some compensation. Profit share agreements and variable (sliding-scale) commissions discussed in the next section allow for a more flexible adjustment of the commission.

Contingent Commission (or *Profit Commission*) is defined as "an allowance payable to the ceding company in addition to the normal ceding commission allowance. It is a pre-determined percentage of the reinsurer's net profits after a charge for the reinsurer's overhead, derived from the subject treaty[4]." For the purpose of this terminology the formula to derive the profit commission C_{profit} is given by means of a percentage $c_{\text{profit}} \in [0, 1]$ of the net profit as follows[5]

$$C_{\text{profit}} \;=\; C_{\text{fix}} + \max\left\{0, P_{\text{ceded}} - E - C_{\text{fix}} - L_{\text{ceded}}\right\} \cdot c_{\text{profit}}\,, \quad (13.2)$$

where C_{fix} is a fixed commission given by Equation (13.1) and P_{ceded} are the ceded written premium, , E the reinsurer's expenses, and and L_{ceded} ceded losses (given by the ceded claims), respectively. Note that the formula clearly reveals that only the positive profit is taken into account, that is, if the net profit is negative, the cedent does not get any allowance additional to the fixed commission. Moreover for the parametrization it is important to mention that the expenses E of the reinsurer are given by a cost (or expense) rate e_{RI} of the premium received, explicitly,

$$E \;=\; e_{\text{RI}} \cdot P_{\text{ceded}}\,. \quad (13.3)$$

We exemplify the idea of profit sharing commissions by returning to the example in Section 13.1.5[6] with one modification: Due to heavy competition the direct insurer is forced to lower his premium income by a percentage of the original premium in order to maintain the accident line of business.

Example: We consider the quota share treaty from the example in above with a reduced written premium of 90 million rather than the original 100 million. Let us assume for the moment that commission is based on the fixed commission strategy with 30% commission rate as

[4]source: http://www.captive.com/service/signetstar/GlosRein.html

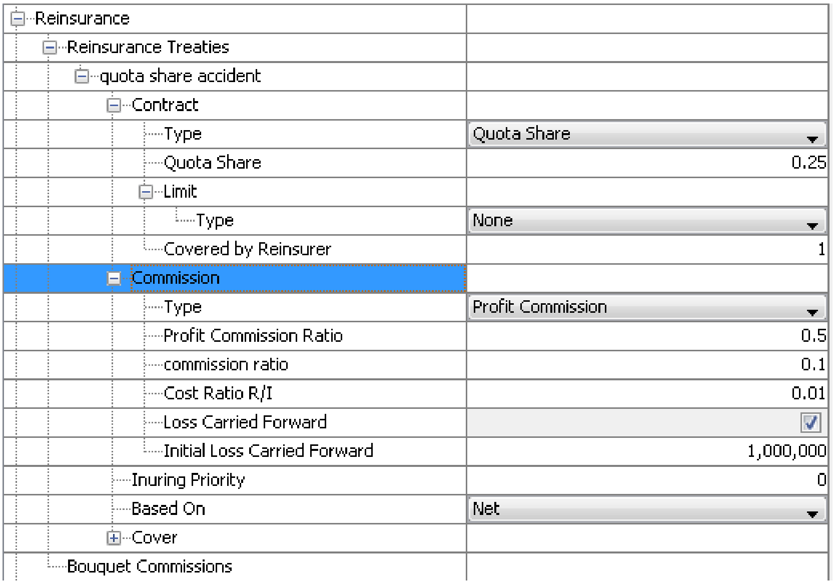
[5]TODO: Adapt variable names to make them consistent in the whole document.

[6]TODO: Define an example environment so that direct references are possible.

before. Without doing any explicit calculations, we easily observe that the reinsurer gains a marginal profit (neglecting his expenses) realized at the expense of the direct insurer's profit owing to the disparity of commission ratio and cost ratio $> 30\%$. Nevertheless, the reinsurer does not want to share in the economic inefficiency of the primary insurer and may expect a better profit rate. Hence, the reinsurer on his part suggests a profit commission strategy on the basis of a 10% fixed commission rate and the expectation to get a net profit *after profit commission* of 2.5 million. Presetting the reinsurer's expenses to 0.25 million, the profit commission rate may accordingly be contractually specified by the expected net profit of the reinsurer

$$\text{netProfit} \quad = \quad P_{\text{ceded}} - E - C_{\text{fix}} - L_{\text{ceded}} \; = \; 5.0\,\text{million}\,. \quad (13.4)$$

leading to a profit share of 50%, hence $c_{\text{profit}} = 0.5$
Figure 13.4 illustrates the parameters that have to be specified by the user in the corresponding parametrization form after selecting the Commission Type **Profit Commission**.



Reinsurance	
Reinsurance Treaties	
quota share accident	
Contract	
Type	Quota Share
Quota Share	0.25
Limit	
Type	None
Covered by Reinsurer	1
Commission	
Type	Profit Commission
Profit Commission Ratio	0.5
commission ratio	0.1
Cost Ratio R/I	0.01
Loss Carried Forward	✓
Initial Loss Carried Forward	1,000,000
Inuring Priority	0
Based On	Net
Cover	
Bouquet Commissions	

Figure 13.4: Parameters for profit commission strategy

The parameters **commission ratio** and **Profit Commission Ratio** correspond to the variables c_{fix} and c_{profit} used in Equations (13.1)&(13.2).

The expense rate of the reinsurer e_{RI} is stored in the parameter **Cost Ratio R/I** to compute the reinsurer's expenses due to formula (13.3). If **Loss Carried Forward** is enabled by clicking the checkmark, we are left with another parameter **Initial Loss Carried Forward** that is not included in equation (13.2). The motivation for including the feature **Loss Carried Forward** into the commission strategy is given by the request of the reinsurer to settle his losses from the reinsurance treaty with future profits in subsequent periods resulting in a diminish of the "commissionable" profit that is given for equation (13.2) according to

$$\mathrm{comProfit} := \max\left\{0, \mathrm{netProfit}\right\}, \qquad \mathrm{netProfit} = P_{\mathrm{ceded}} - E - C_{\mathrm{fix}} - L_{\mathrm{ceded}}\,. \tag{13.5}$$

In the modified situation, a mathematically rigorous derivation of the commissionable profit is based on a simple recursion in a multiple period model with periods $n \in \{0, 1, \ldots, N\}$ allowing for a loss carry-over of period n to period $n+1$ for all $n = 0, \ldots, N-1$. The variables in formula (13.2) thus have to be indexed by the respective period they belong to resulting in

$$P_{\mathrm{ceded}} \mapsto P_{\mathrm{ceded}}^{(n)}, \qquad E \mapsto E^{(n)}, \qquad C_{\mathrm{fix}} \mapsto C_{\mathrm{fix}}^{(n)}, \qquad L_{\mathrm{ceded}} \mapsto L_{\mathrm{ceded}}^{(n)}\,.$$

Rather than using the net profit of the reinsurer to compute the commissionable profit we now use a modified net profit denoted $\mathrm{modProfit}^{(n)}$ and characterized by an additional term for the losses (negative profit) from the preceding period $n-1$. With the above preparations the equation for the modified net profit in period $n+1$ is expressed for all $n \geq 0$ as

$$\mathrm{modProfit}^{(n+1)} = P_{\mathrm{ceded}}^{(n+1)} - E^{(n+1)} - C_{\mathrm{fix}}^{(n+1)} - L_{\mathrm{ceded}}^{(n+1)} + \min\left\{0, \mathrm{modProfit}^{(n)}\right\} \tag{13.6}$$

The above equation recursively defines the sequence of modified net profits $\{\mathrm{modProfit}^{(n)}\}_{n\geq 1}$ and can be solved as soon as the initial condition $\mathrm{modProfit}^{(0)}$ is given. Now the additional parameter **Initial Loss Carried Forward** comes into play by using the thereby predetermined value in order to define

$$\mathrm{modProfit}^{(0)} = P_{\mathrm{ceded}}^{(0)} - E^{(0)} - C_{\mathrm{fix}}^{(0)} - L_{\mathrm{ceded}}^{(0)} - \{\textbf{Initial Loss Carried Forward}\}\,.$$

Note that subtracting the last term on the right hand side of the above equation is completely consistent with equation (13.6) in that losses are in general defined by negative profits. We furthermore remark that the

approach is carried out from the reinsurer's perspective. As a natural implication the user has to be aware of the fact that the parameter **Initial Loss Carried Forward** in the user parametrization stands for the loss of the reinsurer.[7]
In a last step we define the commissionable profit $\mathrm{comProfit}^{(n)}$ for period n similar to (13.5) according to

$$\mathrm{comProfit}^{(n)} = \max\left\{0, \mathrm{modProfit}^{(n)}\right\}, \qquad n \geq 0,$$

that is used to define the profit commission $C^{(n)}_{\text{profit}}$. The overall formula for the profit commission that is payable for period n hence reads

$$C^{(n)}_{\text{profit}} = C^{(n)}_{\text{fix}} + \mathrm{comProfit}^{(n)} \cdot c_{\text{profit}}\,. \tag{13.7}$$

Note that cost ratio e_{RI}, fixed rate c_{fix} and profit rate c_{profit} are constants that are given independently of the period n, but the fixed commission and the reinsurer's expenses expressed as

$$C^{(n)}_{\text{fix}} = c_{\text{fix}} \cdot P^{(n)}_{\text{ceded}}\,, \qquad E^{(n)} = e_{\text{RI}} \cdot P^{(n)}_{\text{ceded}}\,.$$

do depend on the period by the ceded written premium.

Sliding Scale Commission

The sliding-scale commission is another instrument to effectively encourage the direct insurer to influence the results of the reinsurance contract by his underwriting policy, appropriate risk selection, rating, etc. It may be considered as a kind of "provisional" commission as the amount directly depends on the result of the ceded business indicated by the loss ratio of the ceded business

$$l_{\text{ceded}} := \frac{L_{\text{ceded}}}{P_{\text{ceded}}} \in \mathbb{R}^0_+$$

in a given calendar or cedent year. In so doing, the ceding commission C_{slide} given by

$$C_{\text{slide}} = c_{\text{slide}} \cdot P_{\text{ceded}}$$

is decreasingly linked to the loss ratio by defining the sliding commission rate $c_{\text{slide}} = c_{\text{slide}}(l_{\text{ceded}})$ as a function of the loss ratio l_{ceded} and stipulating

$$l^{(1)}_{\text{ceded}} < l^{(2)}_{\text{ceded}} \quad \Longrightarrow \quad c_{\text{slide}}(l^{(1)}_{\text{ceded}}) \geq c_{\text{slide}}(l^{(2)}_{\text{ceded}})\,.$$

[7]TODO: clarify if initial loss carried forward may be negative

In general, the postulation may be realized by using any monotonically decreasing function, practical purposes however will delimitate the choice to a clearly arranged amount. Easy and safe to handle implementations for instance are given by continuous piecewise linear functions defined by a finite number of tuples . The implementation in `PillarOne` is built on another widely-used class of functions characterized by piecewise constancy though at the expense of right-sided continuity. Figure 13.5 shows a typical left-continuous step-function (given by the sum of characteristic functions of disjoint subsets) that may be used to generate a commission rate. As loss ratios are elements of $\mathbb{R}_0^+$, we are considering step-functions on the non-negative axis that may be entirely defined by a finite number of tuples with jump discontinuities and corresponding values together with an initial provision rate at $l_{\text{ceded}} = 0$.

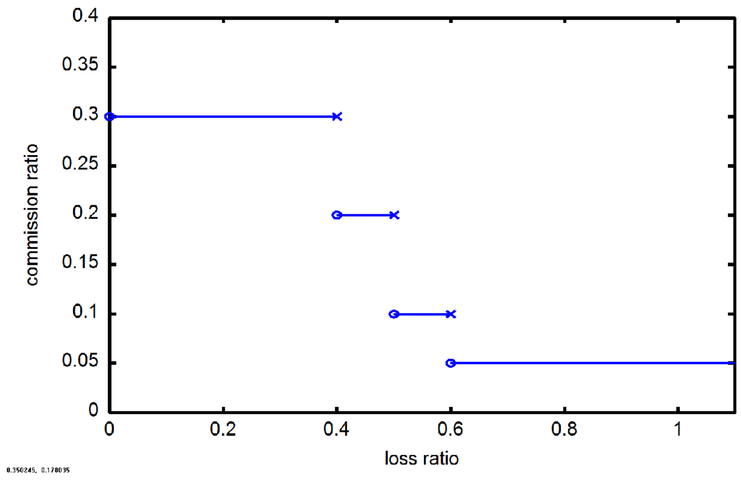
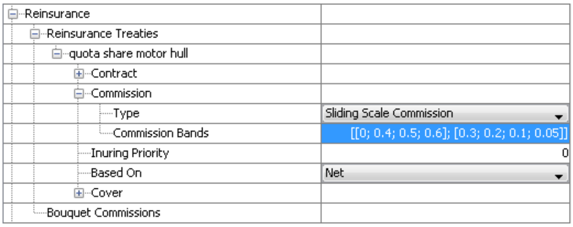


Figure 13.5: Typical step-function for **Sliding Scale Commission**.

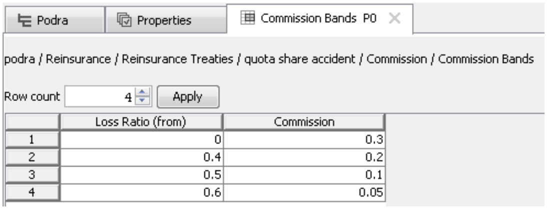
After selecting **Sliding Scale Commission** the parametrization form simply has to be filled by the so-called commission bands as illustrated in Figure 13.6a. To this end, the corresponding values are entered into the form that is shown in Figure 13.6b obtained by double-clicking on the blue marked cell next to **Commission Bands**[8].

[8]TODO: Comparison of sliding and profit commission



Reinsurance	
Reinsurance Treaties	
quota share motor hull	
Contract	
Commission	
Type	Sliding Scale Commission
Commission Bands	[[0; 0.4; 0.5; 0.6]; [0.3; 0.2; 0.1; 0.05]]
Inuring Priority	0
Based On	Net
Cover	
Bouquet Commissions	

(a) Sliding Scale Commission



podra / Reinsurance / Reinsurance Treaties / quota share accident / Commission / Commission Bands

Row count 4 Apply

	Loss Ratio (from)	Commission
1	0	0.3
2	0.4	0.2
3	0.5	0.1
4	0.6	0.05

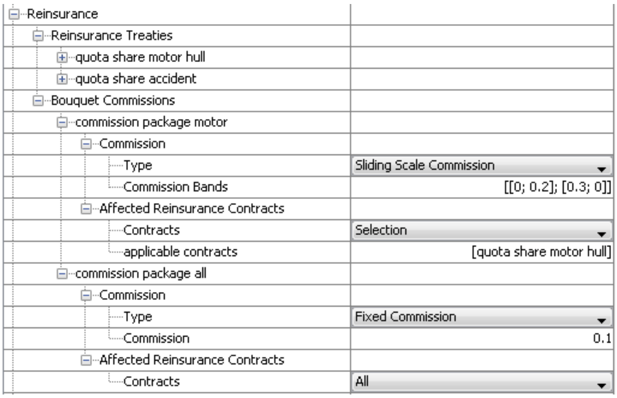
(b) Commission Bands

Figure 13.6: Parameters for sliding scale commission.

The values shown in Figure 13.6 correspond to the step-function as displayed by Figure 13.5.
Note: *The first entry in* ***Commission Bands*** *has to be given by the zero loss ratio otherwise evoking a validation error. To avoid mistakes, the loss ratios have to be entered in strictly increasing order. Furthermore, the step-function specified by the commission bands has to be monotonically decreasing which is a reasonable settlement in practice.*

Bouquet Commission

In the preceding sections we have discussed different commission strategies so far considered as integral part of each reinsurance contract. `RiskAnalytics` supplies an extended implementation thereof by additionally providing the allocation of a selected commission strategy to a bundle of reinsurance treaties.
To this end, the subcomponent **Bouquet Commissions** is considered as the second structural element of the business component **Reinsurance**, respectively **Reinsurance Market**. It allows for dynamically adding (one or several) commission strategies that are applied to a bundle of reinsurance contracts .



Reinsurance	
Reinsurance Treaties	
quota share motor hull	
quota share accident	
Bouquet Commissions	
commission package motor	
Commission	
Type	Sliding Scale Commission
Commission Bands	[[0; 0.2]; [0.3; 0]]
Affected Reinsurance Contracts	
Contracts	Selection
applicable contracts	[quota share motor hull]
commission package all	
Commission	
Type	Fixed Commission
Commission	0.1
Affected Reinsurance Contracts	
Contracts	All

Figure 13.7: Subcomponent **Bouquet Commissions**

As usual, a new segment is created by right-clicking on **Bouquet Commissions**, selecting **Add** and entering the dedicated name. Each segment consists of two parameters **Commission** and **Affected Reinsurance Contracts** permitting to select the commission type from the list of implemented commission strategies and the designated reinsurance contracts by opting for **None**, **All**, or **Selection**. In the latter situation the **applicable contracts** are chosen by double clicking on the right cell, setting the row count in the tab to the number of contracts and picking from the predefined reinsurance treaties.

The computation of the amount of commission is accomplished as described in the preceding sections implying the same set of predetermined parameters that are associated to the particular commission strategy. The important feature intrinsic to the component **Bouquet Commissions** as an entity may be described best by the hierarchic application of the single segments with respect to the selected reinsurance contract bundles: For each contract from the union of all contract bundles commission is allowed once only with top down priority. In doing so, commissions that are derived from single contracts within the component **Reinsurance Treaties** (respectively **Reinsurance Program**) are left unconsidered. This may be mathematically expressed by the sequence of contract bouquets $B_1, B_2, \ldots, B_n$ hierarchically ordered according to the entry in the parametrization form. The commission strategy predetermined in segment $i = 1, \ldots, n$ is then applied to the reduced set

$$B_i \backslash \cup_{j=1}^{i-1} (B_i \cap B_j)$$

rather than B_i excluding duplex allowance. To exemplify the idea let us consider the parametrization as given in Figure 13.7 that is based on two reinsurance contracts “quota share motor hull” and “quota share accident”, and two bouquet commissions strategies “bouquet commission motor” applicable to “quota share motor hull” and “bouquet commission all” applicable to both reinsurance treaties. In the user parametrization “bouquet commission motor” is the foremost bundle resulting in a sliding scale commission for “quota share motor hull” that is added to the commission inherent to the reinsurance treaty. The next executable segment “bouquet commission all” is applied only to “quota share accident” excluding the already considered treaty “quota share motor hull”.

13.2 Proportional Reinsurance

The following proportional reinsurance components are available in RiskAnalytics:

- Quota share with various options for setting limits
- Surplus
- Loss portfolio transfer

The order in which the claims are processed by a proportional contract does not matter, as long as there are no non-linear features added to a proportional contract. If there are limits involved, then we have to attach dates to the claims; both incurred and paid. This way the claims can be sorted and then properly be processed by the contract limits. Instead of dates, we can also work with fractions of a simulation period. ***Note:*** *Working with fractions of periods is computationally more efficient, and as long as we are treating all days in a period equally – no weekends or banking holidays – this is equivalent to working with dates.* Attritional claims distributions are calibrated using the sum of many small claims with different incurred dates, but all lying in a well-defined time period. So by definition, an attritional claim does not have an incurred date since it is an aggregate of many small claims. In order to be able to model more complicated reinsurance programs and proportional contracts with limits, we nevertheless associate an incurred date

with attritional claims. To learn how to set the date of an attritional claim, see Section 10.1.2.
Reinsurance components for proportional contracts process all claim types (see Table 13.1).

	attritional	single claims	event claims
quota share	x	x	x
surplus	x (see)	x	x (see)
loss portfolio transfer	x	x	x

Table 13.1: Claims types processed by proportional reinsurance components

13.2.1 Quota Share

The quota share is a simple form of proportional reinsurance, and has many extensions for setting limits:

- annual aggregate limit (AAL)
- annual aggregate deductible (AAD)
- event limit

Some of these limits can be combined, e. g. AAD and AAL.

13.2.2 Surplus

Surplus reinsurance is a more advanced and more complicated form of proportional reinsurance. In contrast to quota share reinsurance, the portion covered by the reinsurer depends on the sum insured[9] specified for the underlying insured risks. All the risks with sum insured up to a certain retention are fully covered by the reinsured - the exceeding part up to a certain limit is then covered by the reinsurer. The portion of the *claim ceded* to the reinsurer is given by the ratio of the *risk ceded* divided by the *total risk*.
As a result, the modeling of surplus reinsurance requires additional information - a manifest link of the claims to the underlying risks. We will give details on how this link can be provided. Note that for the

[9]or other quantities used to characterize the underlying risk such as 'probable maximum loss' (PML).

other reinsurance treaty types modeled in `RiskAnalytics` no such link is needed.
We see two different approaches to model the link between risk and claims information:

- Attach risk information while generating the claims: Typically, start with the risk information and use risk-dependent characteristics for simulating the claims. This method is called ‘risk to band’.
- Attach risk information after generating the claims: Here, we can start with standard claims generators and try to map the claims to the risks such that suitable characteristics on the risk portfolio are met. This method is called ‘Sum insured generator’.

The risk information is captured in the underwriting segments (see section 10.1.1) and in the claims generators one can refer to the exposure information by selecting either one of the above methods (see section 10.1.3.
Generally, we consider the first approach more sound and we suggest using this whenever possible. However, more detailed data (including claims per risk or per risk class statistics) are needed for a calibration of such risk-dependent claims generators. Therefore, we opt for the second approach in case not all data are available for a calibration of risk-dependent claims generators. Here, suitable ‘RiskAllocator’ components are used which are typically based on more general risk portfolio characteristics.
In addition to the general contract parameters, the surplus-specific parameters are:

- **Retention** (R) of the surplus contract. It must be specified in the same units as the sum insured.
- Number of **lines** (L) the contract covers. This implies that the contract covers up to a limit of $L \cdot R$.
- The **default ceded loss share**. This parameter is used to ced claims for which no risk information is available.

For a risk with sum insured v the proportion of the claim ceded to the reinsurer is then given by

$$\rho(v) = \frac{\min(L \cdot R, (v - R)_+)}{v} \tag{13.8}$$

For each claim, this risk information (v) is used to compute the fraction ceded given by the above formula (13.8). For losses for which no risk information is found, the default ceded loss share is used as the fraction ceded.
Note: *At the moment, different retentions specified for different claims types can only be modeled by using different surplus components and allocating the associated claims accordingly. Different claims types and different retentions should be considered when setting up the model. However, work on generalizing the component to allow for specifying different retentions for different claims types is in progress.*

13.2.3 Loss portfolio transfer

The loss portfolio transfer is an quota share on reserves. On the user interface of `RiskAnalytics` it looks exactly like a quota share without the additional limits feature. It is up to the user to map the appropriate reserves into this contract by selecting the type '**Reserves**' in the setion **Cover**.

13.3 Non-Proportional Reinsurance

The following non-proportional reinsurance components are available in RiskAnalytics:

- working excess of loss (WXL)
- catastrophe excess of loss (CatXL or CXL)
- stop loss
- adverse development cover
- Goldorak
- Finite Re

[10]

Aggregate event claims are not processed by the WXL component. In contrast, the CXL does not process attritional and single claims. A reinsurance component which does not process a particular claim type

[10]TODO: We need some explanation on per risk, per event cover, where appropriate.

simply feeds the gross claims (the input) of this type directly to the net claims (the output).

	attritional	single claims	event claims
WXL		x	
CXL			x
stop loss	x	x	x
adverse development cover	x	x	x
Goldorak[11]	(x)	(x)	x
Finite Re	x	x	x

Table 13.2: Claims types processed by non-proportional reinsurance components

Unlike proportional reinsurance components (Table 13.1), which process any claim type, non-proportional reinsurance components can only process some claim types (see Table 13.2).

13.3.1 Working Excess of Loss

13.3.2 Cat-XL

13.3.3 Stop Loss

13.3.4 Adverse Development cover

13.3.5 Goldorak

The goldorak contract is a reinsurance type mainly used in the French market. It is a mixture of an Cat-XL (cf. Section 13.3.2) and a Stop Loss (cf. Section 13.3.3) i. e. depending on a given threshold it either behaves as CXL or SL. Thus for this type of contract we need all the parameters of the two individual treaties as well as an additional threshold parameter according to which the effective sub-contract is chosen.
Goldorak parameters:

- Premium base
- Premium

[11] Depending on the effective sub-contract only event claims may be affected.

- Goldorak SL Threshold

The parameters **Premium base** and **Premium** have already been explained above. The **Goldorak SL Threshold** determines the *absolute* value of the sum of all claims above which the specified SL sub-contract is applied. If the total sum of all claims is smaller than this threshold the CXL is applied. Thus except the premium related parameters all necessary remaining parameters of the sub-contracts are listed below.
CXL parameters:

- (CXL) attachment point
- (CXL) limit
- (CXL) aggregate limit
- (CXL) reinstatement premiums

Stop loss parameters:

- Stop loss attachment point
- Stop loss limit

Typically, the Goldorak SL Threshold is equal to the stop loss limit.

13.3.6 Finite Reinsurance Contract

The finite reinsurance contract is a multi-period contract consisting of two parts: an experience account and a risk component.

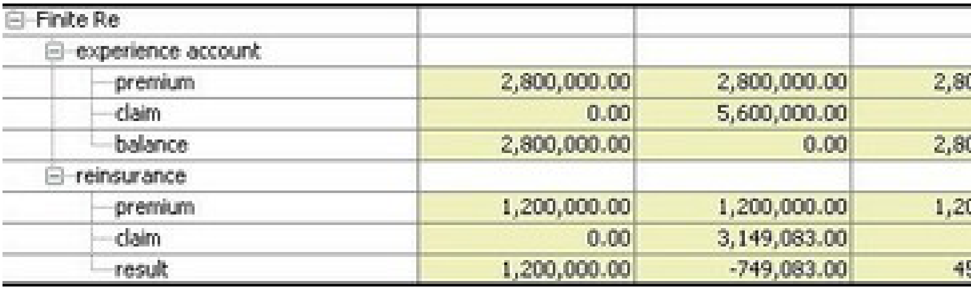
Input

Model Inputs:

For each period t, the finite reinsurance component receives a list of claims C. These can be either the ceded or the net claims produced by another reinsurance component. This choice is decided by the model developer and cannot be changed at runtime by a model user.

User Inputs:

T he **total premium** $P(t)$ of the finite reinsurance contract collected in the **time period** t (for $0 \leq t$)



Finite Re			
experience account			
premium	2,800,000.00	2,800,000.00	2,80
claim	0.00	5,600,000.00	
balance	2,800,000.00	0.00	2,80
reinsurance			
premium	1,200,000.00	1,200,000.00	1,20
claim	0.00	3,149,083.00	
result	1,200,000.00	-749,083.00	49

Figure 13.8: An example of finite reinsurance output

The **fraction of premium** $\alpha(t)$ which is allocated to the experience account.

Output

Experience Account:

premium $P_{ea}(t)$

claims $C_{ea}(t)$

balance $B(t)$ of the experience account

Risk Part:

premium $P_{risk}(t)$

claims $C_{risk}(t)$

balance $R_{risk}(t)$ from the outset until the end of period t

A sample finite reinsurance output for three time periods is shown in Figure 13.11.

Validation

none implemented, but should be:

$P(t) > 0$

$0 \leq \alpha(t) \leq 1$

Calculation

- $P_{ea}(t) = a(t) \cdot P(t)$
- $P_{risk}(t) = (1 - a(t)) \cdot P(t)$

- $C_{ea}(t) = min\{B(t-1) + P_{ea}(t), C(t)\}$
- $C_{risk}(t) = C(t) - C_{ea}(t)$
- $B(t) = B(t-1) + P_{ea}(t) - C_{ea}(t)$
- $R_{risk}(t) = R_{risk}(t-1) + P_{risk}(t) - C_{risk}(t)$

  Note:

  Both premiums (P_{ea} & P_{risk}) are deterministic since α and P are.

  We take $B(t-1)$ and $R_{risk}(t-1)$ to be 0 for time periods $t-1$ before the contract started.

  The definition of $C_{ea}(t)$ ensures that $B(t) \geq 0$.

13.4 Reinsurance Programs

For the examples in the following reinsurance sections, we will work with a property and a motor hull line of business. The claims which we associate to the property line are generated using three different types of claims generators:

- attritional claims using an annual aggregate distribution (denoted Prop$_{attr}$)
- single claims using a frequency-severity model (denoted Prop$_{single}$)
- aggregate cat event claims using a frequency-severity model (Prop$_{cat}$)

The claims which we associate to the motor hull are generated using two types of claims generators:

- attritional claims using an annual aggregate distribution (denoted M$_{attr}$)
- single claims using a frequency-severity model (denoted M$_{single}$)

Let us assume that we want to model the reinsurance structure for the property line shown in Figure 13.9. The attritional claims from Prop$_{attr}$ are not processed by the WXL and CXL components.
Note: *This has implications for calibrating the distribution of the attritional claims generator. Only claims which are completely in the retention — in this example, below* 1 *Mio or possibly even less than* 1

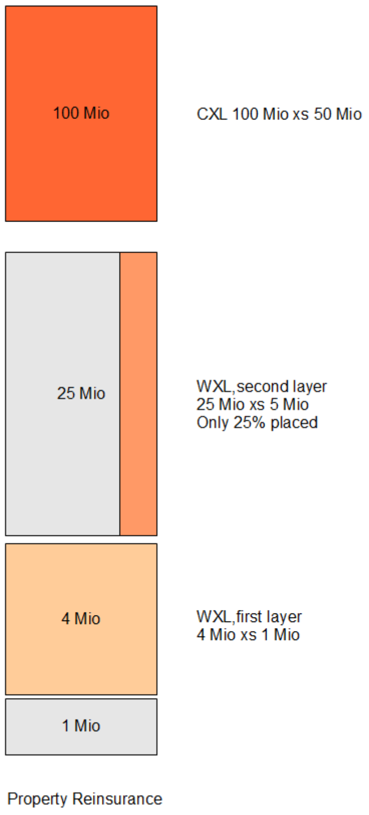


Figure 13.9: Excess of loss

Mio to have a safety margin — should be used to determine the distribution.

The **inuring priority**, an integer value > 0, is used to express the order in which the reinsurance contracts process claims. If two reinsurance contracts need to receive the same claims information, then their inuring priority needs to be set to the same value. Thus, in our example, the two WXL layers must have the same inuring priority in order to both receive the claims from Prop_{single}.

As in reality, each layer is a separate contract. This has the advantage that we can vary some of the parameters per layer, e. g. how much is placed or with which reinsurer it is placed. The latter is only necessary if we want to model reinsurance default. A disadvantage of modeling each layer as a separate contract is that `PillarOne` cannot easily validate whether the retentions and covers of the different layers have meaningful values. It is the user's responsiblity to make sure that the retention of the second layer (5 Mio in our example) is not below the sum of the retention plus the cover of the first layer. Theoretically, the retention of the second layer could be larger than 5 Mio; but in practice, it does not make sense to have a gap between consecutive layers. During the initial parametrization these conditions are usually observed. The danger is that during a reinsurance optimization process, the retention and layers are changed individually. It is thus necessary to check at least the parameters of all other contracts with the same inuring priority.

Reinsurance contracts are usually grouped within a reinsurance program. `PillarOne` can model reinsurance programs either per line of business, or globally with multiline coverage, or even using a mixture of both concepts.

`PillarOne` models reinsurance programs as either 'static', with a fixed number of serial, ordered contracts, or as 'dynamic', allowing the user to add and remove contracts within the graphical user interface and to define the order in which they are applied.

The GUI allows each contract's 'strategy'[12] to be specified (example further below). Currently implemented strategies[13] (and the abbreviations we employ for them, if any) are given in Figure 13.4.[14]

[12]In business language, the 'contract type'. As a `PillarOne` (model) developer, these are ContractStrategy classes, since they embody a methodological choice; we also refer to them as 'reinsurance components' or just 'components'.

[13]we list the strategies that are of general interest; trivial, WCXL and AggregateXL are not discussed here.

[14]TODO: introduce a ContractStrategies figure near here

13.4.1 Simple Reinsurance Programs

Consider a reinsurance program with a fixed number of three reinsurance contracts, which are processed in serial order:

- Each of the three reinsurance contracts can select its own reinsurance component for one of the available contract types as in Figure 13.4. This 'strategy' (contract type/component) selection is made in the GUI under Parameterization → Model Name → in the left pane, and under Reinsurance Program → Contract Name → Contract Type → Type in the right pane.

- Net claims of contract 1 will be transfered to contract 2, in which those formerly net claims will be interpreted as gross claims.

This type of reinsurance program is provided in the Capital Eagle Model (where the parameter **inuring priority** is not available because the processing order is predetermined).
Alternatively to this type of static reinsurance program, with a fixed order and number of contracts, there is also a dynamic version available. The dynamic version provides the user with greater flexibility in determining both the number of contracts, and the order in which they are processed. This dynamic concept is provided in the model **Podra** and explained in the following section.

13.4.2 Dynamic Reinsurance Programs

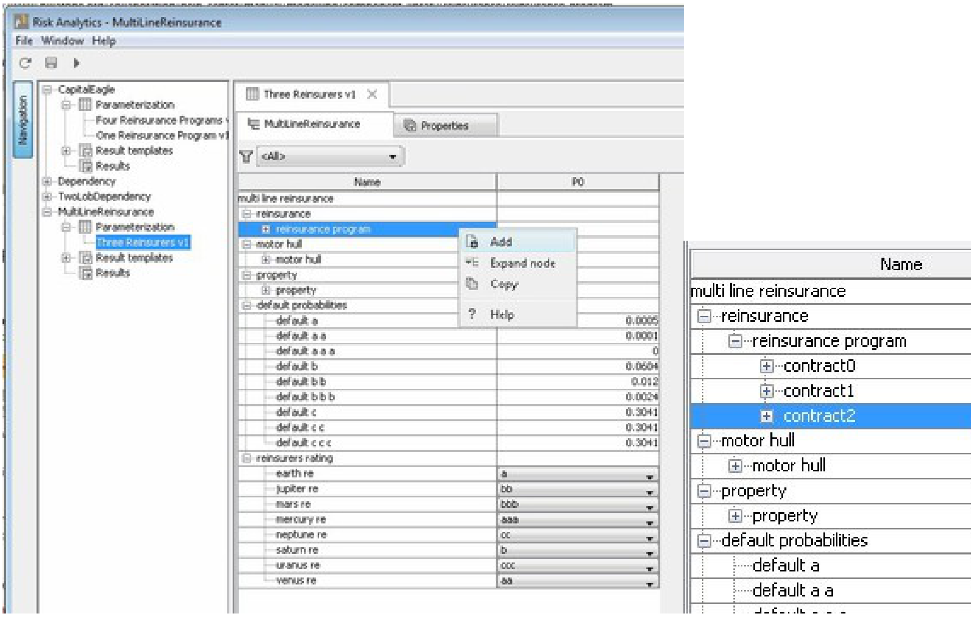
A dynamic reinsurance program enables the user to specify the number and order of contracts with the parameterization.

- In order to add an additional contract to the reinsurance program, right click on reinsurance program and select 'Add'.

- To remove a contract, right click on it and select 'Remove'.

- Each reinsurance contract can select its own reinsurance component for one of the available contract types given in Figure 13.4. The procedure is described in the previous Section (13.4.1).

- The order of the contracts is defined with the parameter inuring priority, which assumes positive integer values.



Name	P0
multi line reinsurance	
reinsurance	
reinsurance program	
motor hull	
motor hull	
property	
property	
default probabilities	
default a	0.0005
default a a	0.0001
default a a a	0
default b	0.0604
default b b	0.012
default b b b	0.0024
default c	0.3041
default c c	0.3041
default c c c	0.3041
reinsurers rating	
earth re	a
jupiter re	bb
mars re	bbb
mercury re	aaa
neptune re	cc
saturn re	b
uranus re	ccc
venus re	aa

Name
multi line reinsurance
reinsurance
reinsurance program
contract0
contract1
contract2
motor hull
motor hull
property
property
default probabilities
default a
default a a

Figure 13.10: Adding or removing a contract from a dynamic reinsurance program

– The contract with the smallest inuring priority will be the first executed. If several contracts have the same inuring priority, they will be applied on the same ‘gross’ claims and underwriting figures.

– If a program has several excess of loss (XL) layers, add as many *contracts as layers are required*[15] and set the inuring priority to an equal number.

– In order to keep a program flexible for extensions, one can use e. g. multiples of ten (10, 20, 30, . . .) as inuring priorities. The gaps leave room to place further contracts as needed within the processing sequence.

[15]TODO: clarify this passage! Does it mean: ‘add as many contracts as each layer requires, using ascending inuring priorities for subsequent layers and equal inuring priorities for each contract within the same layer’?

If a model shouldn't allow a user to edit the number and order of contracts, it should use a static program as described in the previous Section 13.4.1.

13.4.3 Dynamic Multiline Reinsurance Programs

A dynamic multiline reinsurance program is not part of any line of business. However, the dynamic multiline reinsurance program is usually attached on the same level as lines of business in the model tree.

- Dynamic multiline reinsurance programs have the same properties as dynamic reinsurance programs.
- Each contract has an additional 'covered lines' property. Double-click on *the list*[16] in order to select the covered lines using combo boxes.

13.4.4 Multiline Reinsurance Contract with Default

This contract allows to define the lines covered and the reinsurer acting as counterparty on the contract.
Once the reinsurer is defined, it is possible to model the reinsurer's probability of defaulting. The model should contain a reinsurer rating table and default probabilities per rating. If a reinsurer defaults, all contracts with this reinsurer will stop ceding claims.
If a contract is placed at different reinsurers, this can be modeled by selecting combinations of 'covered by reinsurer' and 'reinsurer'.

13.4.5 Finite Reinsurance Contract

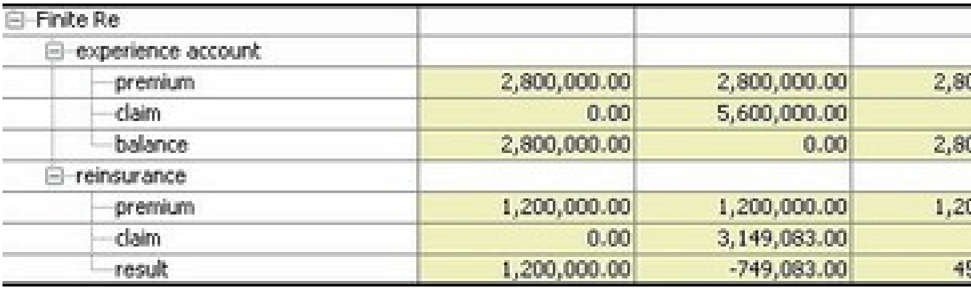
The finite reinsurance contract is a multi-period contract consiting of two parts: an experience account and a risk component.

Input

Model Inputs:

For each period t, the finite reinsurance component receives a list of claims C. These can be either the ceded or the net claims produced by another reinsurance component. This choice is decided by the model developer and cannot be changed at runtime by a model user.

[16]TODO: the list of contracts?



Finite Re			
experience account			
premium	2,800,000.00	2,800,000.00	2,80
claim	0.00	5,600,000.00	
balance	2,800,000.00	0.00	2,80
reinsurance			
premium	1,200,000.00	1,200,000.00	1,20
claim	0.00	3,149,083.00	
result	1,200,000.00	-749,083.00	45

Figure 13.11: An example of finite reinsurance output

User Inputs:

the **total premium** $P(t)$ of the finite reinsurance contract collected in the **time period** t (for $0 \leq t$)

The **fraction of premium** $\alpha(t)$ which is allocated to the experience account.

Output

Experience Account:

premium $P_{ea}(t)$

claims $C_{ea}(t)$

balance $B(t)$ of the experience account

Risk Part:

premium $P_{risk}(t)$

claims $C_{risk}(t)$

balance $R_{risk}(t)$ from the outset until the end of period t

A sample finite reinsurance output for three time periods is shown in Figure 13.11.

Validation

none implemented, but should be:

$P(t) > 0$

$0 \leq \alpha(t) \leq 1$

Calculation

- $P_{ea}(t) = a(t) \cdot P(t)$
- $P_{risk}(t) = (1 - a(t)) \cdot P(t)$
- $C_{ea}(t) = min\{B(t-1) + P_{ea}(t), C(t)\}$
- $C_{risk}(t) = C(t) - C_{ea}(t)$
- $B(t) = B(t-1) + P_{ea}(t) - C_{ea}(t)$
- $R_{risk}(t) = R_{risk}(t-1) + P_{risk}(t) - C_{risk}(t)$

  Note:

  Both premiums (P_{ea} & P_{risk}) are deterministic since α and P are.

  We take $B(t-1)$ and $R_{risk}(t-1)$ to be 0 for time periods $t-1$ before the contract started.

  The definition of $C_{ea}(t)$ ensures that $B(t) \geq 0$.

Chapter 14

Modelling non-life reserves

After successfully setting up a model for premium risks modelling the reserve risks is certainly important. Reserve generators are accessible similar to claims generators e. g. within the Podra model there exists a Dynamic Composed Component Reserve Generators. By selecting add in the context menu of Reserve Generators a reserve generator can be added. It consists of a descriptive name and

- Initial Reserves
- Reserves Model
- Reserve Distribution
- Period Payment Portion

explained in the following sections.
We distinguish between two different methods,

- Calendar Year Method
- Pay-Out Pattern Method

The Calendar Year Method is derived from standard reserving principles. A calendar year starts with knowing about a reserve portfolio consisting of real liabilities and the initial reserve representing their estimated volume. We aim in tracing the risk during one period or

calendar year. ***Note:*** *The following can be equally well applied to a model using a quarter or monthly period length.* Within one calendar year reserve loss payments are made and and the outgoing reserve is set. Deviations of the sum of reserve loss payments and outgoing reserves from the initial reserves may be called risk. So we generate the mentioned sum of payments and outgoing reserves as a random variable. The initial reserve of the next period or next calendar year can be set to the outgoing reserve of the current. Pay-out ratios are fixed and can be set for reserves and first year losses seperately.
The Pay-Out Pattern Method is derived from the observed pay-out patterns. Depending on the pay-out pattern each loss is allocated to the model periods. The reserves can be set as the sum of all future loss payments of the preceding periods.

14.1 Calendar Year Method

14.1.1 Initial Reserves

The **Initial Reserves** of the calendar year are currently handeled within the reserve generator component. They are not tracked by an individual component like the Underwriting Information in the claims generators. Therefore the parameter initial reserves is used.

14.1.2 Reserves Model

There exist three types of the reserve model, depending on the usage of the initial reserves value. They are **Absolute**, **Initial Reserves** and **Prior Period**. All of them having a modifyable reserve distribution. The type absolute defines the result of the reserve distribution as an absolute value of paid plus reserved. Using initial reserves scales (multiplies) the result of the reserve distribution by the initial reserve. And the Prior Period method scales the reserve distribution with the outgoing reserve of the prior period when available resp. the initial reserve in the first period.

14.1.3 Reserve Distribution

Please refer to Subsection 10.1.2 for details on modifyable reserve distributions.

14.1.4 Period Payment Portion

The Period Payment Portion gives the option to split the generated value for the sum of paid and reserved to these values. It is the relative share of the payment in the recent period.

14.2 Pay-out pattern method

The Pay-out pattern method generates a pay out pattern with each claim.

Chapter 15

ALM generators

Asset libility mismatch (ALM) risk is an important part of the insurer's set of risks. The models for technical insurance risks are modeled on a very granular level. To model the ALM risk, it is necessary to incorporate an economic scenario generator (ESG). The outcome, an ALM risk random distribution, can be put in `RiskAnalytics` using the ALM generators component. The component is dynamic, so that several ALM generators can be used in a model.
The parametrization of ALM generators is similar to that of attritional claims.
Usually, ALM generators are of type Absolute, so the generated value of the modifiable distribution is stored as the ALM end of period value. The initial volume is stored in the appropriate parameter. Additional types may be introduced to scale the distribution by Absolute Result, Relative Value or Relative Result.

Chapter 16

Modelling an Insurance Group

As is the case for single corporations, the consolidated insurance group must meet group solvency regulatory requirements to preserve financial stability of the group and protect policy holders and beneficiaries.[1] Within the European Union, large-scale changes are in progress based on the proposal of the Solvency II Framework Directive, entailing a shift of competence for group supervision from national solo-supervisory authorities to a genuine Group Supervisor on the European level ([1, 5, 15, 17]).
There are currently two internationally prevalent approaches to modelling relevant group risks, exhibiting varying perspectives. A very nice outline of the basic principles of these approaches is given in Circular 2008/44 of FinMa ([6, 7]), from which we quote:

> The first approach models the risks of a group on a consolidated basis as if the group were a single legal entity ("consolidated group modeling"). As a consequence, the lead supervisor can determine whether the group satisfies capital adequacy requirements as based on its external liabilities. In so doing, intra-group liabilities are not modeled as it is implicitly assumed that assets are freely transferable within the group.

[1] An **Insurance Group** consists of several insurance companies, while an **Insurance Conglomerate** must own both insurance and banking companies.

> The second approach is based on modeling the risks of the individual legal entities of a group, possibly forming a cluster, and thus models the relations between these units in addition to the external relations ("granular group modeling"). In this second method uniform rules are applied to assess the risks of the assets and liabilities of the individual legal entities, in addition to the risks resulting from the relations within the group. This second method enables FinMa, in its capacity of lead supervisor of the group, to determine whether any risk potential is posed to policyholders and the financial stability of the group by individual parts of the group itself for other parts of the group or for the group as a whole, and whether the individual parts of the group aid the others as needed or whether other risk-mitigating measures are available.
>
> In the SST it is assumed that granular group modeling is applied, with consolidated group modeling additionally being permitted upon application by the group with FINMA or being requested by FINMA. The "group SST" is based on granular group modeling, which can be supplemented by consolidated group modeling.

The "Multi Company Model" provided by `RiskAnalytics` may serve as an appropriate internal model for granular determination, quantification, and analysis of all relevant risks of insurance conglomerates. In what follows we discuss the implementation of the model, its findings in extension to the Podra model, and the internal processing of data related to specific group segments. Instructions for creating a parametrization in the group model are given in Chapter 6.

16.1 Company Segments

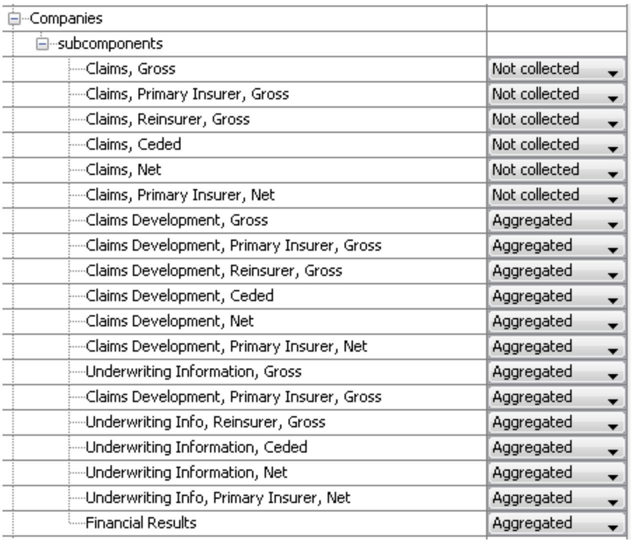
As usual, a new insurance company may be created by right-clicking on **Companies** and selecting **Add** in the appearing context menu. Another possibility is to right-click on an existing company and choose **Duplicate** in the context menu leading to a one-to-one copy of the existing segment. Each company segment consists of a rating parameter that has to be selected from a list with rating values ranging from **AAA** to **D**. Rating parameters play an important role for the quantification of contingency risks, particularly in regard to the Solvency II Framework a precautionary approach in selecting business partners should be taken. To this end, credit-worthiness and solvency information from rating agencies may serve as an important mean to guarantee long-term competitive advantages. For a comprehensive discussion about the importance of rating for reinsurances, the interested reader is referred to [4].

Each predefined company segment represents a legal entity[2] belonging to the insurance group built by the union of all the company segments. Hence, in contrast to the Podra model that provides a tool for modeling the exposure of one insurance company only, we now have to specify the respective insurance unit when editing risks, reinsurance contracts, etc. In so doing, we obtain a consistent possibility of correct valuation of the assets and all liabilities of all the legal units of the insurance group and modeling all the relevant risks of the individual legal units. In view of the fact that capital and risk transfers, such as loans or reinsurance contracts, are frequently used among the legal entities of an insurance group, the "Multi Company Model" provides an important and sound base for managing the risks of groups on a granular perspective level.

The feature of including multiple legal units into the model exhibits its utility after running the simulation and opening the preselected result descriptor. For the sake of illustration we will consider an example below, for starters we focus on the result template as shown in Figure 16.1.

In the template we nicely see that the result descriptor shows the activities of each of the single units of the group separately. As outlined in Chapter 8 the model does not define which output variables have to be collected during the simulation. Instead, the user has to determine before starting the simulation if at all and to which granular level (ranging from the coarse-grained **Aggregated** to the fine grained **Single**) he wants to collect each of the positions listed below **subcomponents** (which is a placeholder for the single entities). The items that are shown in the result tree-view may be roughly separated into claims, underwriting information and financial results. In order to apply the model correctly we have to understand how the resulting items listed in Figure 16.1 are fed into the company segments. This is explained

[2]Or a so called **Cluster**, i. e. well-defined set of legal entities. Clustering is usually allowed for non-significant legal entities or where some legal entities cover the same risk but consist of two carriers. The latter might be the case if one legal entity is funded by a SPV ("Special Purpose Vehicle"); or for fully fronted business.



Companies	
subcomponents	
Claims, Gross	Not collected
Claims, Primary Insurer, Gross	Not collected
Claims, Reinsurer, Gross	Not collected
Claims, Ceded	Not collected
Claims, Net	Not collected
Claims, Primary Insurer, Net	Not collected
Claims Development, Gross	Aggregated
Claims Development, Primary Insurer, Gross	Aggregated
Claims Development, Reinsurer, Gross	Aggregated
Claims Development, Ceded	Aggregated
Claims Development, Net	Aggregated
Claims Development, Primary Insurer, Net	Aggregated
Underwriting Information, Gross	Aggregated
Claims Development, Primary Insurer, Gross	Aggregated
Underwriting Info, Reinsurer, Gross	Aggregated
Underwriting Information, Ceded	Aggregated
Underwriting Information, Net	Aggregated
Underwriting Info, Primary Insurer, Net	Aggregated
Financial Results	Aggregated

Figure 16.1: **MultiCompany**: Results for segments of component **Companies**

in full detail in Section 16.2. Another issue worth discussing is how claims[3] and underwriting information are split up into

- gross claims and gross underwriting information,
- gross claims of primary insurer and gross underwriting information of primary insurer,
- gross claims of reinsurer and gross underwriting information of reinsurer,
- ceded claims and ceded underwriting information,
- net claims and net underwriting information, and
- net claims of primary insurer and net claims of reinsurer.

This breakup of information is rooted in the fact that an insurance company [4] may be fully licensed to transact general primary insurance *and* reinsurance business.

[3]We restrict to claims and note that the following discussion applies to the developed claims as well.

[4]We speak here of a single unit of the group and NOT of the group as an entity.

We close this section by exemplifying the aforementioned matter by means of three imaginary companies “Mars Re”, “Venus” and “Pluto Re”, each of them trading in the primary- as well as re-insurance business. At first view the set up of the example may be a disincentive to the user, however in anticipation of the following section it also gives an idea of the internal handling of information. As will be clear below, the example is constructed as a small tutorial where the user is requested to create a parametrization in `PillarOne` leading to a simulation model that renders the outlined results.

Example: The three insurance companies “Mars Re”, “Venus” and “Pluto Re” are trading in different classes of business, contingently purchasing reinsurance from one or several reinsurers. The contracts (direct business with related reinsurance contracts) are shown in Table 16.1 below, where each column belongs to one class of business.

line of business:	motor Venus	motor Mars Re	motor Pluto Re	accident Mars Re
direct insurer:	Venus	Mars Re	Pluto Re	Mars Re
reinsurance treaty:	QS motor Venus	QS motor Mars	—	QS accident Mars
reinsurer:	Mars Re	—	—	Pluto Re: 0.7, Venus: 0.3

Table 16.1: Insurance business activities and contractual relationships

Note that the last entry in the last column denotes a reinsurance contract that is shared between two reinsurers registered with associated portions. Moreover the line of business “motor Pluto Re” has no specified reinsurance contracts, whereas “motor Mars Re” reinsures its business but with an reinsurer that is unknown here[5]. Having listed all the necessary information, we may process the incoming information of claim burdens and underwriting segments and allocate it to the predefined companies. In the next two tables we give a list of all the incoming gross and ceded information allowing for calculating the profit of each of the three companies in the group.

The attentive reader may have noticed that underwriting information for the line “motor Venus” is listed twice in Table 16.2. This may result, e. g. from a partition into motor third party and motor comprehensive cover[6]. As an exercise we evaluate the results for the single insurance

[5]This corresponds exactly to the parametrization where the user is not forced to select a reinsurer from the list of companies in contrast to the lines of business component.

[6]We are aware of the fact that subsuming such segments under one line of business is a rather ‘quick and dirty’ approach.

	written premium	**line of business**
packet 1	1 mio	motor Venus
packet 2	2 mio	motor Mars Re
packet 3	0.5 mio	motor Venus
packet 4	1.8 mio	accident Mars Re
packet 5	0.05 mio	motor Pluto Re

	ceded written premium	**line of business**	**reinsurance**
packet 1	0.1 mio	motor Venus	QS motor Ve
packet 2	0.4 mio	motor Mars Re	QS motor Ma
packet 3	0.05 mio	motor Venus	QS motor Ve
packet 4	0.9 mio	accident Mars Re	QS accident

Table 16.2: **Incoming underwriting information:** gross info (left side), ceded info (right side)

	claims	**line of business**
packet 1	1.5 mio	motor Venus
packet 2	1.0 mio	motor Mars Re
packet 3	0.2 mio	motor Venus
packet 4	1.2 mio	accident Mars Re
packet 5	0.04 mio	motor Pluto Re

	ceded claims	**line of business**	**reinsurance contract**
packet 1	0.15 mio	motor Venus	QS motor Venus
packet 2	0.2 mio	motor Mars Re	QS motor Mars
packet 3	0.02 mio	motor Venus	QS motor Venus
packet 4	0.6 mio	accident Mars Re	QS accident Mars

Table 16.3: **Incoming claims burden:** gross claims (left side) and ceded claims(right side)

units of the group that will be shown in the result descriptor when the collector **Aggregated** is selected. To this end, the reader is invited to

1. compute the results by hand in order to understand the specific characteristic of **Companies** and,

2. to practice the handling of the simulation tool by filling the parametrization form of the "Multi Company Model" with parameters and values reflecting the situation given in Tables 16.1, 16.2 and 16.3.

After running the simulation the results can be taken from the result template. In Figure 16.2 the aggregated results are illustrated for the insurances "Mars Re" and "Pluto Re", while for "Venus" the problem is accomplished step-by-step.
We start our considerations with the remark that the business activity of an insurance company is quantified by merging operations in the direct insurance business with the reinsurance business implying for the example that "Venus" not only obtains premiums from the left-sided list in Table 16.2, but also from possible reinsurance business listed on the right-hand side of Table 16.2. Explicitly: Subsequently

using the monetary unit one million, the total gross premium P_{gross} of "Venus" is derived as the sum[7]

$$P_{\text{gross}} = P^{(PI)}_{\text{gross}} + P^{(RI)}_{\text{gross}}\,, \qquad P^{(PI)}_{\text{gross}} = 1.5\,, \qquad P^{(RI)}_{\text{gross}} = 0.27\,,$$

where $P^{(RI)}_{\text{gross}}$ originates from the cession of the business line "accident Mars Re" (0.9 ceded premium) weighted with the "Venus"-share of 0.3 in the reinsurance treaty. The resulting net premium $P_{\text{net}} = P_{\text{gross}} - P_{\text{ceded}}$ then results by subtracting from P_{gross} the total of ceded premiums $P_{\text{ceded}} = 0.15$ of "Venus", and reporting also the direct insurer's portion $P^{(PI)}_{\text{net}}$ of the net premium, we end up with

$$P_{\text{net}} = 1.62\,, \qquad P^{(PI)}_{\text{net}} = 1.35\,.$$

Note that the net results are not reported for exclusive reinsurance business as by definition and excluding retrocession[8] it corresponds to $P^{(RI)}_{\text{gross}}$.
In the same way we handle the designated commission where it is clear that the commission in gross portfolios of direct insurance business is zero by default. Moreover notice about the sign convention for commissions that becomes clear when compared to premiums. Following the guidance for premiums we then obtain for gross, ceded and net commissions C_{gross}, C_{ceded}, C_{net}

$$C_{\text{gross}} = -0.135\,, \quad C_{\text{ceded}} = -0.03\,, \quad C_{\text{net}} = C_{\text{gross}} - C_{\text{ceded}} = -0.105\,.$$

Splitting these results into exclusive reinsurance and primary insurance business yields

$$C^{(RI)}_{\text{gross}} = C_{\text{gross}}\,, \qquad C^{(PI)}_{\text{net}} = -C_{\text{ceded}}\,.$$

Table 16.3 finally supplies the results for the claims (or losses) denoted by L. Using the above rules for sub- and super-indexing we end up with

$$L_{\text{gross}} = L^{(PI)}_{\text{gross}} + L^{(RI)}_{\text{gross}}\,, \qquad L^{(PI)}_{\text{gross}} = 1.7\,, \qquad L^{(RI)}_{\text{gross}} = 0.18$$

for the gross losses, $L_{\text{ceded}} = 0.17$ for ceded claims and

$$L_{\text{net}} = L_{\text{gross}} - L_{\text{ceded}} = 1.71\,, \qquad L^{(PI)}_{\text{net}} = L^{(PI)}_{\text{gross}} - L_{\text{ceded}} = 1.53\,.$$

for net results.

[7] We use the abbreviations PI and RI for primary insurer and reinsurer, respectively.

[8] At this stage `PillarOne` does not allow for cover strategies that apply to predefined reinsurance contracts.

16.2 Internal Processing of Data

[9] For the understanding of the model it is of utmost importance to get an overall picture of the internal handling of the information stemming from the parametrization defined by the user. We will now discuss this issue in full detail with the help of the **Companies** component processing information about claims, underwriting segments, and financial results.

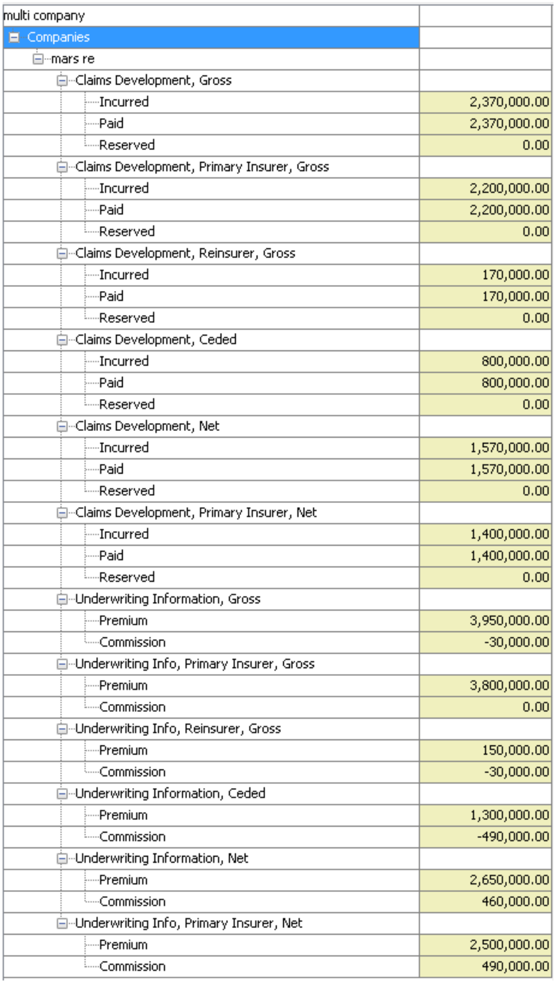
The value of **Financial Results** as listed in Figure 16.1) is evaluated in a direct way using the fact that each segment of **ALM Generators** has a parameter **Company** forcing the user to attach to one segment of **Companies**. By this means progression is straightforward in that each segment of **ALM Generators** generates output value(s) stored in a variable that is marked with the associated company. We finally obtain a list of results collecting outputs of all the ALM segments. In a next step this list is sent to all segments of the component **Companies**. To put it roughly, for each of these segments the incoming list is filtered with respect to the company-marker that is attached to the result variables. If the company under consideration is “Venus” we will pass through the list and collect all results that are marked with company “Venus”, whereas the leftover is discarded. The “Venus”-labeled results are summed up and finally prepared by the dedicated result descriptor.

Handling of gross claims is slightly more sophisticated in that segments of **Claims Generators** are not attached to the component **Companies** directly. Instead, interrelation is given indirectly via the component **Lines Of Business** acting as a kind of intermediary. As nicely illustrated in Figure 6.1, each segment of **Lines of Business** has two parameters that are of interest here: The parameter **Company** attaching the participating insurance unit to the dedicated business line, and the parameter **Line of Business Claims** – ***Shares*** providing the link between segments of **Companies** and **Claims Generators**. Figures 16.3 and 16.4 [10] give a rough sketch of internal handling of the data that is based on the component-coupling as outlined before. With these preparations it is easy to understand how gross claims are processed: The claim packets are sent to all of the objects of **Lines of Business**, where their member variable *claim.line* is instantiated with the business line under consideration if and only if it is related to

[9] TODO: Where to place this section?

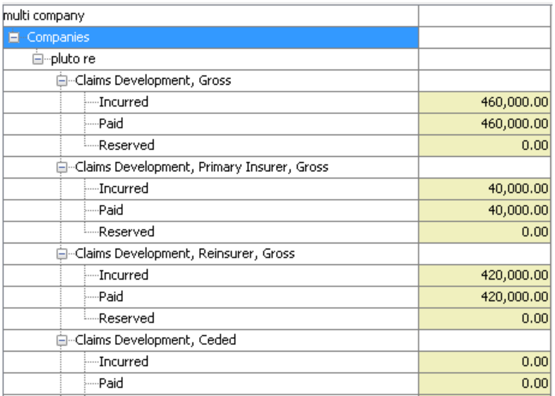
[10] TODO: This and the next figure need a revision. I am open for suggestions!

the peril (that is, the origin of the claim) by means of the parameter **Line of Business Claims**. Hence filtering of claims in the company segments is carried out by means of the member variable *claim.line* pointing to the business line with assigned parameter **Company**. For managing of ceded claims we refer to Figure 16.4 and remark that underwriting information is processed analogously to claims by using the component **Underwriting** rather than **Claims Generators** (compare Figure 6.1).



multi company	
Companies	
mars re	
Claims Development, Gross	
Incurred	2,370,000.00
Paid	2,370,000.00
Reserved	0.00
Claims Development, Primary Insurer, Gross	
Incurred	2,200,000.00
Paid	2,200,000.00
Reserved	0.00
Claims Development, Reinsurer, Gross	
Incurred	170,000.00
Paid	170,000.00
Reserved	0.00
Claims Development, Ceded	
Incurred	800,000.00
Paid	800,000.00
Reserved	0.00
Claims Development, Net	
Incurred	1,570,000.00
Paid	1,570,000.00
Reserved	0.00
Claims Development, Primary Insurer, Net	
Incurred	1,400,000.00
Paid	1,400,000.00
Reserved	0.00
Underwriting Information, Gross	
Premium	3,950,000.00
Commission	-30,000.00
Underwriting Info, Primary Insurer, Gross	
Premium	3,800,000.00
Commission	0.00
Underwriting Info, Reinsurer, Gross	
Premium	150,000.00
Commission	-30,000.00
Underwriting Information, Ceded	
Premium	1,300,000.00
Commission	-490,000.00
Underwriting Information, Net	
Premium	2,650,000.00
Commission	460,000.00
Underwriting Info, Primary Insurer, Net	
Premium	2,500,000.00
Commission	490,000.00

(a) Aggregated results for "Mars Re"



multi company	
Companies	
pluto re	
Claims Development, Gross	
Incurred	460,000.00
Paid	460,000.00
Reserved	0.00
Claims Development, Primary Insurer, Gross	
Incurred	40,000.00
Paid	40,000.00
Reserved	0.00
Claims Development, Reinsurer, Gross	
Incurred	420,000.00
Paid	420,000.00
Reserved	0.00
Claims Development, Ceded	
Incurred	0.00
Paid	0.00

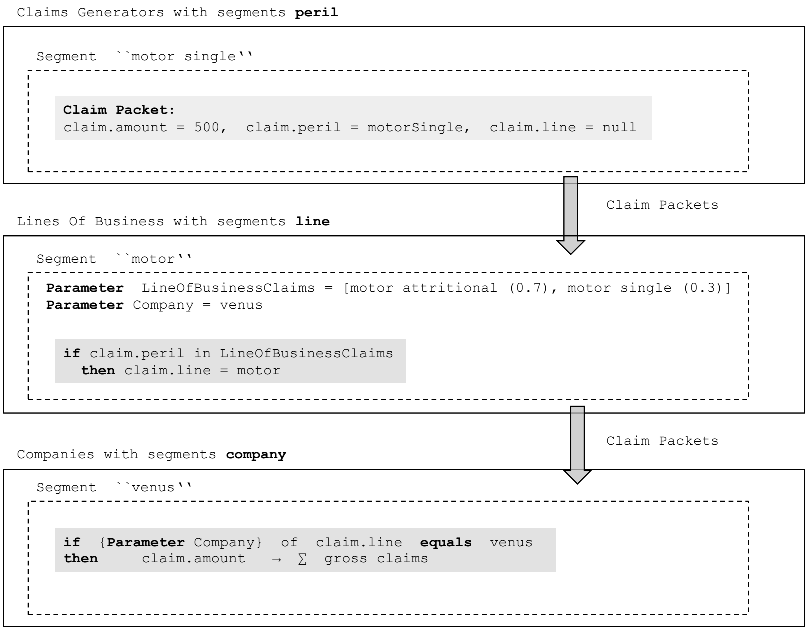

Claims Generators with segments **peril**
Segment ``motor single''
Claim Packet:
claim.amount = 500, claim.peril = motorSingle, claim.line = null
Claim Packets
Lines Of Business with segments **line**
Segment ``motor''
Parameter LineOfBusinessClaims = [motor attritional (0.7), motor single (0.3)]
Parameter Company = venus
if claim.peril in LineOfBusinessClaims **then** claim.line = motor
Claim Packets
Companies with segments **company**
Segment ``venus''
if {**Parameter** Company} of claim.line **equals** venus
then claim.amount → ∑ gross claims

Figure 16.3: Internal handling of gross claims in **Companies**

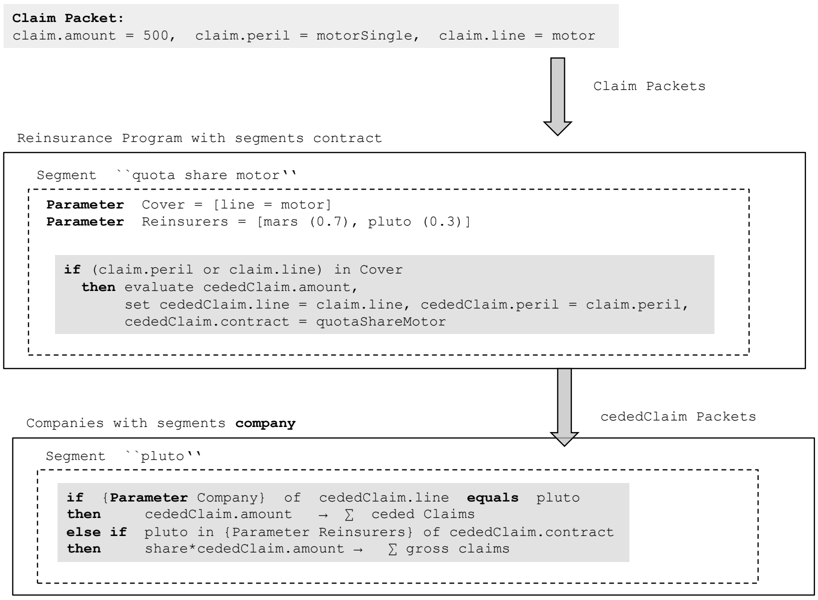

Claim Packet:
claim.amount = 500, claim.peril = motorSingle, claim.line = motor
Claim Packets
Reinsurance Program with segments contract
Segment ``quota share motor''
Parameter Cover = [line = motor]
Parameter Reinsurers = [mars (0.7), pluto (0.3)]
if (claim.peril or claim.line) in Cover then evaluate cededClaim.amount, set cededClaim.line = claim.line, cededClaim.peril = claim.peril, cededClaim.contract = quotaShareMotor
cededClaim Packets
Companies with segments company
Segment ``pluto''
if {Parameter Company} of cededClaim.line equals pluto then cededClaim.amount → ∑ ceded Claims
else if pluto in {Parameter Reinsurers} of cededClaim.contract then share*cededClaim.amount → ∑ gross claims

Figure 16.4: Internal handling of ceded claims in **Companies**

Part III

Developer Guide

Chapter 17

Introduction

This part of the `RiskAnalytics` documentation is meant for software developers who want to change or extend business logic, or simply add new features to the `RiskAnalytics` application. To develop code which is maintainable and fits well with the `PillarOne` framework, it helps to understand the basic concepts which are described in Chapter 9.
Adopters of `RiskAnalytics` are free to modify source code, e.g. to add features for their own or a client's use. `RiskAnalytics` is released under the GPL license, and hence, anything which is compliant with this license is allowed.
If you would like some of your changes to be included in an official release of `RiskAnalytics`, please get in touch with a core team member.[1]

[1]Code contribution to an open source project is by necessity discretionary, in both directions: contributors should not reveal information that should remain private to their enterprise, such as information from (or about) their clients, while the core project team must exercise some degree of control to maintain the goals of the project, such as commonality or performance requirements.

Chapter 18

Development environment

First you need the source code which can be downloaded from the `PillarOne` website, `www.pillarone.org`. Apart from the business logic which is completely written by the core team and the community members, `RiskAnalytics` is built on top of leading software frameworks. The most prominent one is Grails[1], which itself contains the Spring framework, Hibernate and ULC, to mention the most important ones. You don't need to download and install these frameworks separately. The `RiskAnalytics` source distribution contains the appropriate versions of these frameworks.
To download the source distribution...
Developing with such a sizable software system is more than just editing files. Hence, we recommend that you set up an integrated development environment (IDE). The core team uses IntelliJ IDEA and consequently the source distribution contains the project files for setting up the complete `RiskAnalytics` source code in IDEA. The project can also be set up in Netbeans or Eclipse.
For source code versioning we use Git. Anyone can obtain read access, by following the steps below. First you should configure your Git username and password for the project:

```
git config user.name "Your Name"
```

[1]Grails is the open-source web application framework based on Groovy. See their homepage for more information.

```
git config user.email "your.email@email.com"
```

Git usernames can also be configured globally, but (at least on Windows) this is not recognized when committing through IDEA.
When the project is started for the first time, IDEA might complain that path variables are not set. They have to be set once globally and are necessary to allow for stable project files. If IDEA does not complain, they can be set at **File** → **Settings** → **Path variables**.

- PILLARONE_GRAILS: Should be set to the location where the modified Grails 1.2.0 version was checked out (see above)
- USER_HOME: Should be set to your user home direcotry (where the .grails folder is)

Chapter 19

Modularization

`RiskAnalytics` is split up in different modules, currently core, application and business logic. As `RiskAnalytics` is a Grails application, a module is actually a Grails plugin.
For our own and modified public plugins we use our own Grails plugin repository, which is located at: `https://svn.intuitive-collaboration.com/GrailsPlugins/`
Read only access does not require authentication, but a SVN account is required to release plugins into our repository.

Chapter 20

Working on existing plugins

Our plugins are based on a modified version of Grails 1.2.0, which is also available from Git. gitaccess@svn.intuitive-collaboration.com:riskanalytics-grails.git

The project files of the plugins are already configured correctly to use this grails version, however you will need to set GRAILS_HOME to the correct folder if you want to use grails commands on the CLI.

The core and application plugins can be obtained from our Git repository:

```
git clone git://github.com/pillarone/risk-analytics-core.git
git clone git://github.com/pillarone/risk-analytics-application.git
```

Additional plugins are no longer checked into the repository, because we have our own plugin repository now. Plugins are defined in application.properties and will be downloaded automatically.

Changes to the plugin can be commited and pushed to **git** as usual. However it is only possible to change files inside the plugin scope. For example, the application plugin cannot change anything in `riskanalytics.core.*`. If such a change is required, a new version of the core plugin must be released and then installed in the application plugin.

20.1 Releasing a plugin

At first the distribution location (Grails plugin repository to save the plugin) must be defined. The discovery location, which is used to download plugins, is already defined in BuildConfig.groovy.
To avoid to accidently commit SVN credentials the distribution location should not be defined at the same place. Instead a `/.grails/settings.groovy` file should be created and the following line added:
grails.plugin.repos.distribution.pillarone =
"https://username:password@svn.intuitive-collaboration.com/GrailsPlugins/"

Before releasing, the plugin version number has to be corrected in the file `RiskAnalytics*GrailsPlugin.groovy`. If everything is set up, the plugin can be released using the `release-plugin` ant target.

It is probably a good idea to tag the Plugin release in git: git tag -m 'tagged plugin version x.y' 'v.n.n'

20.2 Running it all together

As the development is now split up between the different plugins, developing the UI means that there are only simple test models available and there is no UI for developing business logic. This should encourage to write test cases instead. However sometimes it is necessary to run for example a business logic model with the UI available. For this case, a 'development runner' project is available from git.
git clone gitaccess@svn.intuitive-collaboration.com:riskanalytics-devrunner.git

The external dependencies required by core and application are already included. The location of the `RiskAnalytics` plugins to be used can be configured in BuildConfig.groovy. That way it is possible to run and test a complete application, which uses the current plugin 'Snapshots' contrary to the plugin projects which use only released plugin versions and usually only depend on the core plugin.
However this project should only be used as 'application runner' and NOT to edit files of other plugins! All changes and running tests should be done in the individal plugin project.

Chapter 21

Creating your own plugin

All grails plugins are based on a Grails 1.3.4. However due to a bug in the Groovy 1.7.4 compiler, we replaced it with Groovy 1.7.5. You should therefore use that version which can be obtained from: gitaccess@svn.intuitive-collaboration.com:riskanalytics-grails.git
Make sure the environment variable `GRAILS_HOME` points to the above mentioned grails and `PATH` includes `GRAILS_HOME/bin`. Then simply run grails create-plugin PluginName to create a new plugin.

21.0.1 Define `PillarOne` repository

To be able to download from our plugin repository, add the following lines to BuildConfig.groovy:

```
grails.project.dependency.resolution = {
    inherits "global" // inherit Grails' default dependencies
    log "warn"

    repositories {
        grailsHome()
        grailsCentral()
    }

    def myResolver = new URLResolver()
    myResolver.addArtifactPattern "https://build.intuitive-collaborati

    resolver myResolver
}
```

Most plugins will need the risk-analytics-core plugin: grails install-plugin risk-analytics-core The core plugin and all its dependencies will then be downloaded and installed. Also the file YourPluginGrailsPlugin.groovy should be edited, to define name, version, dependencies and more. An example *GrailsPlugin.groovy and an example build.xml file can be obtained from the Core or Application plugin.
It is also recommended to update the Config.groovy & DataSource.groovy files, because certain options should be set. The easiest way to do this is to look at the corresponding files in the core & application projects and take the necessary options from there.

21.1 Git Hints

- Working on a branch
  - create the branch locally using git branch –track 0.5.x remotes/origin/0.5.x
  - switch the working branch using git checkout 0.5.x

21.2 Environments

As Risk Analytics is a Grails application we also use Grails' support for different environments. The environment Risk Analytics runs in is defined at startup with the parameter `-Dgrails.env=environment-name`. We use the environments for the database connection settings (in grails-app/conf/DataSource.groovy) and several other application settings in grails-app/conf/Config.groovy.
It is possible to add new environments by editing these files and copy-pasting existing environments. This is typically done when a new database product will be used.
For more information see: http://grails.org/doc/latest/guide/3. Configuration.html#3.2 Environments

21.3 User Management

The server version of RiskAnalytics includes a user management, which includes login, saving of user preferences and the user information is linked to modelling items. The user management is based on Spring Security and there are two pre-defined roles, ROLE_USER who can use the application and ROLE_ADMIN who can additionaly manage the users.
New users can be added in the BootStrap of your application. An example of how to add new users can be found in the class CoreBootStrap in the core plugin.

Chapter 22

Scalability

The principle behind stochastic simulations as applied in RiskAnalytics is to generate a large number of so called independent, identically distributed realizations of the future ('iteration') and then to statistically analyze the ensemble of all these iterations.
The accuracy (confidence) of the results can be improved by increasing the number of iterations, i.e. the ensemble the statistical estimates are based on. Typically, asking for better accuracy of the results implies longer runtimes. This is not necessarily true if

- a larger number of computing resources (CPU's) is available and
- the software used to run the simulation is capable of distributing the computations to many CPU's.

Stochastic simulations are an ideal candidate to distribute the computations since independent iterations can be generated just as well on different CPU's.

22.1 Application Structurue Revisited

The RiskAnalytics platform is split up into three modules:

- core which provides a basic infrastructure such as data access and reporting services and for setting up and running models;
- application which, basically, contains the graphical user interface;

- business logic.

The SimulationRunner is part of the core module and is the component from which the stochastic simulations are configured, started and run. The following basic steps are processed in a simulation run:

- Pre-simulation actions
  - Instantiating a model
  - Loading of input data
- Simulation
  - Computing the random scenarios (n iterations with m periods each)
  - Writing the raw simulation output to a file
- Post-simulation actions [1]
  - Loading data from file
  - Computing the statistics
  - Storing the statistics output in the database

The computational intensive and time consuming part of a simulation run clearly is the simulation itself. In order to achieve scalability it is sufficient to distribute the computation of the random scenarios. Here it is important to assure that all iterations remain stochastically independent. Furthermore it is important to obtain reproducibility independent of the computing grid configuration (e.g. number of nodes).

[1] Up to RiskAnalytics 1.1 the raw simulation output was automatically written to the database as part of the postsimulation actions. In view of eliminating the bottleneck of the data persistence layer to obtain a scalable distributed solution this approach was changed: Now the raw simulation output is written to local files and only the statistics output is written to the database. The raw simulation output can be loaded into the database on demand.

22.2 GridGain

Since 1.2 scalability is a standard feature of the platform. The application should scale on a single multi-core laptop as well as on a grid consisting of multiple computing resources. No additional deployment overhead is needed; once set-up and configured, the simulations are automatically distributed. The framework used for distributing the computations is GridGain[2]. GridGain is open source, java-based hence multi-platform and allows simple deployment. It allows to be integrated smoothly into the existing platform with minimal intrusion. By using a threaded architecture, it scales well on a multi-core single machine and on multiple computing resources combined to a grid.

22.3 Implementation

In this section a short description of the implementation of the computing and data persistence part in the gridified RiskAnalytics application is given.

22.3.1 Computing

As discussed in 22.1 the Simulation Runner, used to start a stochastic simulation, consists of 3 steps. The "Simulation" step is computing the random scenarios during multiple iterations. These iterations are independet from each other and can therefore be run in parallel. In order to achieve this using the GridGain Framework, the following changes and additions to RiskAnalytics have been made:

- The total number of iterations is split into blocks. Each block is allocated to a grid job which then processes the iterations.

- A grid job is basically an execution unit which can be run on a grid node. Multiple grid jobs run in parallel. A grid job consists of a simulation configuration, a list of simulation blocks and the execution routine. The simulation configuration describes all runtime aspects e.g. numberOfIterations, numberOfPeriods, the parameterization, etc. A simulation block specifies the iterations to

[2]For more information about GridGain check www.gridgain.com

be processed and how to handle the random number generation. The execution routine creates and starts a Simulation Runner.[3]

- There is actually one grid node embedded in the Riskanalytics application. This grid node is always active and acts as master node. The master node is responsible for creating jobs and distributing them to other grid nodes (slave nodes). These slave nodes process the iterations included in the assigned blocks and send the intermediate results back to the master node. The master node then also manages the data persistence (see 22.3.2).
- The number of created grid jobs per simulation run is chosen equal to the number of cpus (or cores) in a grid.

22.3.2 Data persistence

Following changes and additions have been made to support the local file persistence:

- The master node is continuously storing the received intermediate simulation results.
- This raw data consisting of iteration meta information and simulated values is no longer automatically stored into a database but directly written in binary form into files.
- As soon as all grid jobs have finished their simulation run, the reduce method of the master node is called. This method processes the statistical analysis of the raw data (post simulation process) and stores the statistical characteristics into a database.

22.4 Configuration

A grid consists of several computing resources on which a distributed task can be executed. To allow this distributed computing, a considerable amount of communication and coordination between the computing resources is necessary. GridGain provides the necessary communication and coordination out-of-the-box, meaning no additional configuration effort is needed to add grid nodes to an existing grid. The

[3]As of RA 1.2 the Simulation Runner executes just the pre-simulations actions and the simulation actions; the post-simulation actions are no longer part of the Simulation Runner and are called later (see 22.3.2).

communication is done via network broadcasting, so each grid node in the same subnet automatically joins the grid.
As mentioned in 22.3.1 there is already a grid node embedded in the RiskAnalytics application. When executing a simulation run, this master node is executing the application multi-threaded by considering all available cpu cores of its host system. If an additional grid node should join the grid make sure the host system is running in the same subnet as the Riskanalytics application system and follow these instructions to set up a GridGain node:

1. Unzip `gridgain-3.0.0c-win.zip`
2. Set `GRIDGAIN_HOME` environment variable to the path you unzipped (e.g. `C:\gridgain-3.0.0c-win`)
3. Copy the Riskanalytics library `RiskAnalytics.jar` to `gridgain-3.0.0c-win\libs\ext`
4. Adjust `gridgain-3.0.0c-win\config\default-spring.xml` Replace:

```
 <!--
<property name="p2PLocalClassPathExclude">
    <list>
        <value>org.springframework.*</value>
        <value>org.openspaces.*</value>
        <value>org.hibernate.*</value>
    </list>
</property>
-->
```

   with

```
 <property name="p2PLocalClassPathExclude">
    <list>
        <value>org.apache.commons.logging.*</value>
        <value>org.slf4j.*</value>
    </list>
</property>
```

5. Execute `gridgain-3.0.0c-win\bin\ggstart.bat` A successful start shows on the prompt:
`>>> GridGain started OK`

6. Start the Riskanalytics application. On the prompt of the gridgain node following message should be visible (as soon as the application has started):
   ```
   >>> Node JOINED [nodeId8=..., addr=[...], CPUs=...]
   >>> Topology snapshot [nodes=2, CPUs=...]
   ```

7. When you start a simulation run with at least 2000 iterations (so at least 2 jobs are generated and can be distributed), you should be able to follow the log output of this simulation run on the GridGain prompt.

Chapter 23

Writing business logic: Components

The following examples can all be found within in the `RiskAnalytics` source code repository:
`https://svn.intuitive-collaboration.com/RiskAnalytics/trunk/RiskAnalyticsPC/src/groovy/org/pillarone/riskanalytics/domain/examples/groovy`.

23.1 Step-by-Step Component Example

We create a component providing a premium value to other components. The premium will be the product of number of policies and the price per policy parameter.

- Create a new Groovy or Java class in the corresponding package. (Line 1, 6)
  Domain classes can be found in src/groovy/org.pillarone.riskanalytics.domain
- The class has to extend org.pillarone.riskanalytics.core.components.Component (Line 6)
- The class has two parameters parmNumberOfPolicy and parmPricePerPolicy (Line 8, 9) The order in which the parameters are defined is also the order how they will be displayed in the UI. If parameters are inherited from superclasses, the parameters of the superclass are displayed first.

- The class has an output channel outPremium (Line 11) in order to send premium information to other components.
- Premium has to be calculated (Line 14), wrapped in a packet and added to the output channel (Line 15). Once doCalculation() has finished, the framework will send this premium packet to following .nents being wired to PremiumCalculation.outPremium.

```
    package org.pillarone.riskanalytics.domain.examples.groovy

    import org.pillarone.riskanalytics.core.components.Component
 4  import org.pillarone.riskanalytics.core.packets.PacketList

    class PremiumCalculation extends Component {

      double parmNumberOfPolicy
 9    double parmPricePerPolicy

      PacketList<PremiumPacket> outPremium = new PacketList<PremiumPacket↩
            >(PremiumPacket)

      void doCalculation() {
14        double premium = parmNumberOfPolicy * parmPricePerPolicy
          outPremium << new PremiumPacket(value: premium)
        }
    }
```

Let's say we want another component providing index information which influences the price per policy. In order to cover this use case we write a component with extended functionality:

- Add an additional input channel inIndex (Line 7)
- Adjust the premium calculation: build the product of all received indices (Line 14 - 15) and adjust the price per policy accordingly (Line 14)

```
    package org.pillarone.riskanalytics.domain.examples.groovy

    import org.pillarone.riskanalytics.core.packets.PacketList
 4
    class PremiumCalculationWithIndex extends PremiumCalculation{
```

```
    PacketList<IndexPacket> inIndex = new PacketList<IndexPacket>(IndexPacket)

    void doCalculation() {
        double level = 1.0
        for (IndexPacket idx in inIndex) {
            level *= idx.value
        }
        double pricePerPolicy = parmPricePerPolicy * level
        double premium = parmNumberOfPolicy * pricePerPolicy
        outPremium << new PremiumPacket(value: premium)
    }
}
```

For the sake of completeness, the following listing contains the IndexPacket and PremiumPacket classes. Both classes are derived from SingleValuePacket having a value property.

```
package org.pillarone.riskanalytics.domain.examples.groovy

import org.pillarone.riskanalytics.core.packets.SingleValuePacket

class IndexPacket extends SingleValuePacket {
}


package org.pillarone.riskanalytics.domain.examples.groovy

import org.pillarone.riskanalytics.core.packets.SingleValuePacket

class PremiumPacket extends SingleValuePacket {
}
```

23.2 Step-by-Step Example of Composed-Component

We create a component composed of a frequency and claims size generator.

- Create a new Groovy class in the corresponding package (Line 1, 2).
  Domain classes can be found in src/groovy/org.pillarone.riskanalytics.domain
  Caveat: Composed components have to be defined in a Groovy

class, as wiring would not work with a Java class in the current implementation.

- The class has to extend (see Line 9):
  org.pillarone.riskanalytics.core.components.ComposedComponent

- The class has two sub-components subIndexProvider and subPremiumCalculation. Make sure all sub-components name start with sub! Components not starting with sub will not be displayed in any of the user interfaces. (Line 11, 12)

- The class has an output channel outPremium (Line 14) in order to send premium to other components such as claims generators or reinsurance contracts.

- As this composed component does not have any input channels, we have to define which component is executed first in doCalculation() (Line 17).
  Caveat: As composed components contain no business logic by itself, it is not necessary to implement doCalculation(). The only exception are composed components without any (wired) in channels. Similar to the starting components concept in a model, doCalculation() is required for triggering. If any input channel is wired, ComposedComponent.doCalculation() has to be used!

- Each composed component has to implement the abstract wire() method. Wiring between the sub-components is done similar to the wiring in models using WireCategory. (Line 21-23). As sub-components are nested we have to make sure that the information provided or required outside the composed component is replicated. This is done with the so called PortReplicatorCategory (Line 24-26).
  Caveat: Using the PortReplicatorCategory the properties on the left and right side have to be either both in or both out properties, whereas for WireCategory the left side property has to be an in and the right side an out property.

```
1  package org.pillarone.riskanalytics.domain.examples.groovy

   import org.pillarone.riskanalytics.core.components.ComposedComponent
   import org.pillarone.riskanalytics.core.packets.PacketList
   import org.pillarone.riskanalytics.core.wiring.WiringUtils
```

```
 6  import org.pillarone.riskanalytics.core.wiring.WireCategory
    import org.pillarone.riskanalytics.core.wiring.PortReplicatorCategory

    class IndexedPremium extends ComposedComponent {

11      IndexProvider subIndexProvider = new IndexProvider()
        PremiumCalculationWithIndex subPremiumCalculation = new ↩
              PremiumCalculationWithIndex()

        PacketList<PremiumPacket> outPremium = new PacketList<PremiumPacket↩
              >(PremiumPacket)

16      protected void doCalculation() {
            subIndexProvider.start()
        }

        void wire() {
21          WiringUtils.use(WireCategory) {
                subPremiumCalculation.inIndex = subIndexProvider.outIndex
            }
            WiringUtils.use(PortReplicatorCategory) {
                this.outPremium = subPremiumCalculation.outPremium
26          }
        }
    }
```

In order to model a 'global' and a line specific inflation, an in channel accepting an additional external index is required.

- The class has an input channel inIndex (Line 10)
- As inIndex is wired to the second sub component of the work-flow, the implementation of doCalculation() in the super class is required and therfore not overwritten.
  Note: *The method* isReceiverWired(inIndex) *(Line 13), which is implemented in the base class* Component*, returns true if any* out *property of another component is connected to* inIndex*. Therefore, it is not mandatory to wire all channels.*
- Finally, it is necessary to provide the index information to the index provider component using the closure[1] PortReplicatorCategory to wire them (Lines 25–27).

[1]A Groovy closure is an object declared within matching curly brackets containing a code block, optional argument declarations and, implicitly, a call() method. When and how many times this method is called (and, as a result, the contained code block gets executed) depends on the context within which the closure is declared. See [9], particularly the Formal and Informal Guide pages for more information.

```
package org.pillarone.riskanalytics.domain.examples.groovy

import org.pillarone.riskanalytics.core.packets.PacketList
import org.pillarone.riskanalytics.core.wiring.PortReplicatorCategory
import org.pillarone.riskanalytics.core.wiring.WiringUtils
import org.pillarone.riskanalytics.core.wiring.ITransmitter

class AdvancedIndexedPremium extends IndexedPremium {

    PacketList<IndexPacket> inIndex = new PacketList<IndexPacket>(↩
        IndexPacket)

    protected void doCalculation() {
        if (isReceiverWired(inIndex)) {
            // corresponds to super.super.doCalculation() which is not possible
            for (ITransmitter transmitter : allInputReplicationTransmitter) {
                transmitter.transmit()
            }
        }
        subIndexProvider.start()
    }

    void wire() {
        super.wire()
        if (isReceiverWired(inIndex)) {
            WiringUtils.use(PortReplicatorCategory) {
                subPremiumCalculation.inIndex = this.inIndex // code within ↩
                    closure
            }
        }
    }
}
```

23.3 Arbitrary Number of Equal Components

Composed components consist of a variable number of subcomponents, each of the same type. The number of subcomponents is defined in the parameterization. *According the available records in a database*[2] the same number of sub-components are instantiated.

[2]TODO: clarify the english!

Concept

- A dynamically composed component is similar to a composed component, in the sense that it contains components, but also different, in the sense that it contains a variable number of components of a specific type. The type of these components has to be defined in the function abstract Component getDefaultSubComponent()
- The wire() and doCalculation() methods follow exactly the same concepts as for a composed component.
- The abstract class DynamicComposedComponent handles adding, removing, naming and other checks for all derived classes. The components are administered using a private list of component called componentList.
- This concept enables a data driven modelling. The Podra model actually contains nine void containers (underwriting segments, claims generators, reserve generators, dependencies, event generators, segments, reinsurance, ALM generators and structures).
- Models using dynamically composed components typically include several filter components for allocation. As an example the reinsurance program and all its reinsurance contracts will receive all claim and underwriting packets. Not all of them will be treated by every contract. As the wiring is fixed and cannot be optimized, it is necessary to filter packets. Therefore as a drawback of the higher flexibility a lower performance has to be accepted.
- *The higher flexibility increases the potential of parameterization errors*[3].

Step-by-step example

We create a dynamically composed component containing reinsurance contracts.[4]

- Create a new Groovy class in the corresponding package (Line 1, 2). Domain classes can be found in `src/groovy/org.pillarone.riskanalyti`

[3]TODO: This is not part of the concept but a warning.
[4]TODO: (sku): add an easier example

- Composed components have to be defined in a Groovy class as wiring would not work with a Java class in the current implementation.
- The class has to extend org.pillarone.riskanalytics.core.components.DynamicComposedComponent (Line 2)
- The DynamicPremiumCalculation has an arbitrary number of PremiumCalculation components. Their default instantiation is defined in getDefaultSubComponent(). The method is called whenever an application user right clicks on the dynamic premium calculation and adds a new premium calculation component.

```
package org.pillarone.riskanalytics.domain.examples.groovy

import org.pillarone.riskanalytics.core.components.DynamicComposedComponent
import org.pillarone.riskanalytics.core.components.Component
import org.pillarone.riskanalytics.core.packets.PacketList

class DynamicPremiumCalculation extends DynamicComposedComponent {

    PacketList<IndexPacket> inIndex = new PacketList<IndexPacket>(↩
        IndexPacket)
    PacketList<PremiumPacket> outPremium = new PacketList<PremiumPacket↩
        >(PremiumPacket)

    void wire() {
        replicateInChannels this, 'inIndex'
        replicateOutChannels this, 'outPremium'
    }

    Component createDefaultSubComponent() {
        return new PremiumCalculation()
    }
}
```

23.4 Filtering and Allocation

Whenever a reallocation is needed, loops become unavoidable in the model graph.[5] There are specific composed components containing components being executed in different phases.

[5]TODO: Provide example here.

The concept and implementation is relatively new. The API is not yet stable. The current implementation allows two phases. It is used to reallocate ceded and net claims and underwriting information to lines of business in order to display resulting gross, ceded and net figures in one place.
An example implementation is ConfigurableLob. The components

- MultipleCalculationPhaseComposedComponent
- MultiPhaseDynamicComposedComponent

can be found in the package org.pillarone.riskanalytics.core.components.

23.5 Different Behaviors

In order to provide a preferably small and well structured set of components combined with a high user flexibility, component parameters may be defined in an exchangable strategy[6]. A strategy may consist of an arbitrary number of parameters or containing even a specific implementation. In order to implement flexible behavior several classes are required:

- an interface extending IParameterObject and common methods needed for all behaviors. This interface is then used in the component as type of the parameter and for accessing different functionality of a behavior.
- a class per behavior implementing the interface mentioned before
- a type class extending AbstractParameterObjectClassifier containing a list of all available behaviors
- components containing a strategy parameter have to be adjusted accordingly

As a use case for a strategy different index calibrations should be supported. As a consequence all strategies of the example will need to provide an index. This component producing the IndexPacket has to correctly handle the different use cases. In order to easily access methods available in all strategies an interface is required:

[6]TODO: provide example

```
   package org.pillarone.riskanalytics.domain.examples.groovy;

   import org.pillarone.riskanalytics.core.parameterization.IParameterObject;

5  public interface IIndex extends IParameterObject {

       double getIndex();
   }
```

The implementation for this behavior is trivial as the strategy keeps only the parameters and does not contain any index transformation logic. getType() (Line 7) and getParameters() (Line 11) are methods of the IParameterObject interface. It's up to the developer to split up implementation code between component and strategy class. Reinsurance contracts might be an interesting example. Contract strategies contain the specific contract logic and the components the common implementation.

```
   package org.pillarone.riskanalytics.domain.examples.groovy
2
   class RelativeIndexStrategy implements IIndex {

       double changeIndex

7      Object getType() {
           IndexType.RELATIVEPRIORPERIOD
       }

       Map getParameters() {
12         return ['changeIndex': changeIndex]
       }

       double getIndex() {
           return changeIndex
17     }
   }
```

The type class contains all available strategies as a static type including the default parameters for each strategy. Variable names should not contain any special characters like underscores. The user interface will provide all strategies in the combobox being part of the static list all. All the other methods are required e. g. for im-/export and the UI.

```
   package org.pillarone.riskanalytics.domain.examples.groovy
2
```

```
import org.pillarone.riskanalytics.core.parameterization.↩
      AbstractParameterObjectClassifier
import org.pillarone.riskanalytics.core.parameterization.IParameterObjectClassifier
import org.pillarone.riskanalytics.core.parameterization.IParameterObject

class IndexType extends AbstractParameterObjectClassifier {

    public static final IndexType ABSOLUTE = new IndexType("absolute", "↩
          ABSOLUTE", ["index": 1d])
    public static final IndexType RELATIVEPRIORPERIOD = new IndexType("↩
          relative prior period", "RELATIVEPRIORPERIOD", ["changeIndex": 0d])

    public static final all = [ABSOLUTE, RELATIVEPRIORPERIOD]

    protected static Map types = [:]
    static {
        IndexType.all.each {
            IndexType.types[it.toString()] = it
        }
    }

    private IndexType(String displayName, String typeName, Map parameters) {
        super(displayName, typeName, parameters)
    }

    public static IndexType valueOf(String type) {
        types[type]
    }

    public List<IParameterObjectClassifier> getClassifiers() {
        return all
    }

    public IParameterObject getParameterObject(Map parameters) {
        return IndexType.getStrategy(this, parameters)
    }

    static IIndex getStrategy(IndexType type, Map parameters) {
        IIndex index
        switch (type) {
            case IndexType.ABSOLUTE:
                index = new AbsoluteIndexStrategy(index: parameters['index'])
                break
            case IndexType.RELATIVEPRIORPERIOD:
                index = new RelativeIndexStrategy(changeIndex: parameters['↩
                      changeIndex'])
                break
        }
        return index
    }
}
```

The implementing component can query the parameter for the used strategy (Line 20, 23) and the index value. To keep the prior period index a member variable is added (Line 14). This member variable has to be reset for every iteration (Line 29). Explanations on the scope concept is available in the next section.

```
 1  package org.pillarone.riskanalytics.domain.examples.groovy

    import org.pillarone.riskanalytics.core.components.Component
    import org.pillarone.riskanalytics.core.packets.PacketList
    import org.pillarone.riskanalytics.core.simulation.engine.PeriodScope
 6
    class FlexibleIndexProvider extends Component {

        PeriodScope periodScope

11      IIndex parmIndex = IndexType.getStrategy(IndexType.ABSOLUTE, ['index': 1↩
            d])
        PacketList<IndexPacket> outIndex = new PacketList<IndexPacket>(↩
            IndexPacket)

        double priorIndex = 1d;

16      protected void doCalculation() {
            if (periodScope.getCurrentPeriod() == 0) {
                initIteration()
            }
            if (parmIndex.getType().equals(IndexType.ABSOLUTE)) {
21              outIndex << new IndexPacket(value: parmIndex.getIndex())
            }
            else if (parmIndex.getType().equals(IndexType.RELATIVEPRIORPERIOD)↩
                ) {
                outIndex << new IndexPacket(value: parmIndex.getIndex() * ↩
                    priorIndex)
            }
26          priorIndex = outIndex[0].value
        }

        private void initIteration() {
            priorIndex = 1d
31      }
    }
```

23.6 Accessing External Information

By default a component is unaware of the simulation context. If business logic is depending on specific dates context information can be injected. There exist three different, nested scopes:

- SimulationScope provides information being valid throughout the whole simulation such as the number of iterations, model, parameter dao, result configuration, output strategy (file/database) and the IterationScope.
- IterationScope provides information being valid throughout a single iteration such as the current iteration number, number of periods, period stores and the PeriodScope.
- PeriodScope provides information being valid for a single period such as the current period. If a model is based on exact dates specific methods are available in order to get the current and next period start date.

The injection of information is done by simply adding a property of the required scope to a component. Java components will need according getter and setter methods additionally.

23.7 Period Store

A PeriodStore is used whenever a Component needs a memory in order to access data of former periods or prepare data that the Component itself will need in future periods. The content may be any kind of objects e. g. packets being sent out in future periods. Period stores are cleared after an iteration. Period stores are injected and governed by the framework. *The component itself can only read and write elements to it*[7]. Index handling is done by the framework. Elements are accessed with a relative index e. g. elements of the current period will be accessed with index 0, previous with -1. If a component is written in Java getPeriodStore() and setPeriodStore(PeriodStore periodStore) have to be implemented.

[7]TODO: What is ‘only’ refering to? Does it mean, no modifying or deleting?

23.8 Packet

- Packets are type safe objects sent from a source component to a target component using a channel.
- A packet may contain any kind and number of objects.
- For convenience a packet implementation should overwrite String toString(). It makes debugging much easier.
- When is an implementation of the following methods required?
  - equals
  - hashCode
  - compareTo

  TODO: provide answer.
- Content modification (Copy/not copy transmitter)
  TODO: explain

Business Logic in Packets

Generally packets should be kept lean. *However there are cases where it is useful including business logic in them.*[8]

Storing and Displaying Packets

There are two base classes allowing to persist and display results: SingleValuePacket and MultiValuePacket.

- For SingleValuePacket, the value property is displayed in the result view. It's name can be configured, by overwriting String getValueLabel().
- For MultiValuePacket, all numeric properties are displayed in the result view according their order in the packet class. Derived classes should override the method getValuesToSave() in order to limit displayed properties.

Note: *If a packet class is implemented in Java, getters and setters have to be written. Use IDE refactoring features to do this.*

[8]TODO:

Packet Pooling for Performance Optimization

TODO: Implementation still missing.

Chapter 24

Testing Business Logic

24.1 Purposes and forms of Testing

In order to avoid changes breaking existing code different kinds of tests are required to provide an immediate feedback to the developer. According to our code contribution process all tests have to be executed before changes are committed into the source code repository. Furthermore the webapplication hudson will execute all test cases again and provide feedback to the committer if any test has failed.
Two different kinds of tests of business logic which we utilize are:

- unit tests, for testing single components, strategies, packets or wiring; and
- integration tests, to check that model simulations run consistently and completely[1].

24.2 Unit Tests

- Directory structure: Each source class should be accompanied by a corresponding unit test class with a parallel classpath e. g., test class

  ClaimsAggregatorTests in test/unit/org.pillarone.riskanalytics.domain.pc.aggregat

  tests source class

[1]this implicitly includes regression tests, so that old parametrization files will still run in newer `RiskAnalytics` releases

ClaimsAggregator in src/groovy/org.pillarone.riskanalytics.domain.pc.aggregators.

- Directory structure: A unit test has to be placed in the corresponding package to the source class, e. g. ClaimsAggregatorTests has to be placed in test/unit/org.pillarone.riskanalytics.domain.pc.aggregators as ClaimsAggregator is placed in src/groovy/org.pillarone.riskanalytics.domain.pc.

- Each unit test class is written in Groovy, has to end with *Tests in order to be found by the Grails test framework and be derived from GroovyTestCase.

- Test cases are a mean to document the usage of a component, therefore every test class should include a method testUsage() as documentation of the basic usage of a component.

- If business logic calculations is done with doubles or floats, results won't match completly. Therefore it is possible to define an ϵ.

  assertEquals 'message', 4, component.testFigure, 1E-8

- If a component contains code throwing exceptions, it's necessary to test if this failures really occure. The syntax is as follows:

  shouldFail IllegalArgumentException, { component.doCalculation() }

  The first argument of shouldFail is the expected exception, the second contains a closure with the code to be executed in order to get the exception.

  TODO: code snippet throwing the same exception type in several blocks: how to make sure exception was thrown where expected? Evaluating exception message?

- If objects are used in several test methods it is recommended to create them in static methods. Example from org.pillarone.riskanalytics.domain.p

```
static ReinsuranceContract getContract0() {
    return new ReinsuranceContract(
        parmContractStrategy: ReinsuranceContractStrategyFactory.↩
            getContractStrategy(
            ReinsuranceContractType.QUOTASHARE,
            ["quotaShare": 0.5,
             "commission": 0.0,
             "coveredByReinsurer": 1d]))
}
```

This contract can then be used in any other test class. Add parameters to the static functions to get a more flexible usability. Be aware that a new object should be created in every call of the static method to avoid side effects between tests.

- When testing several components it won't be sufficient to call doCalculation() of the first component as this won't trigger following components. Instead start() has to be called. This method includes publishing of results to following components. Unfortunately it includes the clearing of the out channel lists too. Inspecting out channels for results won't work in this case. Therefore the TestProbe concept was introduced. It is a probe that can be connected to any out channel and will collect published content, making it available after start() has been executed. Another scenario for using TestProbe is to immitate an out channel to be wired in order to trigger calculations. Whenever a TestProbe is connected to an out channel isSenderWired() will return true. Example: def inChannelWired = new TestPretendInChannelWired(claimsGenerator, "inEventSeverities") Examples:

  Use def probeGross = new TestProbe(aggregator, "outUnderwritingInfoGross") to pretend an out channel is wired and

  List quotaShareNet = new TestProbe(quotaShare, "outUncoveredClaims").result to collect outcome from quotaShare.outUncoveredClaims. Details about the differences of doCalculation(), execute() and start() can be examined in the source code of org.pillarone.riskanalytics.core.comp

- In order to pretend an in channel to be wired TestPretendInChannelWired has to be used. If an in channel of a component is wired to such a component isReceiverWired() will be true.

- Whenever a ComposedComponent is tested internalWiring() has to be called before start() is executed. Omitting internalWiring() will result in an execution of the first component within a composed component only.

24.3 Model Tests

Model tests run a simulation on a specified model and optionally check its output. The ModelTest class and associated framework code provide a simple yet extensible way to test that simulations run completely

(without runtime errors) and consistently (reproducibly) across release versions.
Like simulations, model tests take as input a model, a parametrization, and a result template. The first element, the model (class name), is required; the latter two, parametrization and result template (names) have defaults if they are not given explicitly. Additional options specify

- how many iterations to run (default: 10),
- whether the model test should compare the results collected (default: no), *and if so, where the reference data can be found (default naming convention applies)*[2]
- whether the results should be saved to a file or a database (default: file), and
- the test result's display name and *filename*[3].

When collected, the model simulation result consists of tuples of (iteration, period, path, field, value). When saved to a file, each tuple appears on one line of a so-called tab-delimited text file with extension .tsl. [4] The result template defines which values – i. e., which (path, field) combinations – to collect.
The driver for each model test is a Groovy class extending ModelTest. Each ModelTest subclass implements a method, getModelClass, to tell the framework which model (class) it is testing. Likewise, the method shouldCompareResults, wich returns a boolean value[5], tells the framework whether to collect the results defined in the result template. These are the required ModelTest elements.
This forces the model testing framework to instantiate the model at runtime, and then run a simulation on it using the assigned parameters and result template. If the option to save the results to a file was selected, the results are written to a file in the directory . If the option to compare results was selected, the results are compared with those in a file of the same format, which must be given in the directory . Any differences result in the test failing. If there are no runtime errors and no differences between the reference result set and the test's result set, the test passes.
Through this mechanism, one can adopt the following methodology to initially verify specific aspects of a model, and subsequently en-

[2]TODO: verify this statement!
[3]TODO: or database tablename?
[4]TODO: Where and when is the output directory specified or configured (e. g. at run or compile time)?
[5]true or false

force the same behavior, saving the specification as an artifact [6] in `RiskAnalytics`.

1. Generate specific parametrizations and result templates for a model, targeting specific behavior.
2. Run the corresponding model test, saving a result file with no comparison.
3. Inspect and verify the results (once; iterating to this point until correct).
4. Copy the result file into `RiskAnalyticsPC/test/data/`.
5. Change the test class option to subsequently require comparison, using the copied file from the previous step as the reference result.

The model test class AggregateDrillDownCollectingModeStrategyTests illustrates many of the points mentioned above.
This section is not yet finished![7]

[6]more precisely, a regression test. These codify and guarantee application behavior, acting as a measurestick/safeguard/constraint to protect against unintended side-effects or incompatible interpretations resulting from future code development efforts. This helps not only to fulfill the enterprise-level goals of transparency and standards compliance, but also to efficiently expend development efforts while reaching them.

[7]TODO: check veracity of and expand/expound on these statements! e. g., AggregateDrillDownCollectingModeStrategyTests

Bibliography

[1] The insurance groups and solvency ii, 2007.

[2] Stephen P. D'Arcy, Robert J. Finger, Charlse C. Hewitt, Charles L. McClenahan, Gary S. Patrik, Matthew Rodermund, Margaret Wilkinson Tiller, Gary G. Venter, and Ronald F. Wiser. *Foundations of Casualty Actuarial Science*. Casualty Actuarial Society, 4th edition, 2001.

[3] Jörg Dittrich, Ali Majidi, Norbert Kuschel, and Laurent Berthaut. PillarOne Dynamic Reinsurance Analysis (PODRA): The easy way. http://www.munichre.com/publications/302-06030_en.pdf, February 2009.

[4] Kathleen Ehrlich and Margarita von Tautphoeus. Solvency II and reinsurance: How important is a reinsurer's rating? http://www.munichre.com/publications/302-06232_en.pdf, November 2009.

[5] FINMA. Circular 2008/29: Internal business transactions – insurance groups. http://www.finma.ch/e/regulierung/Documents/finma-rs-2008-29-e.pdf, 20 November 2008.

[6] FINMA. Circular 2008/44 SST: Swiss Solvency Test (SST). http://www.finma.ch/e/regulierung/Documents/finma-rs-2008-44-e.pdf, 28 November 2008.

[7] FINMA. Rundschreiben 2008/44 SST: Schweizer Solvenztest (SST). http://www.finma.ch/d/finma/publikationen/Documents/finma-rs-2009-sammlung-d.pdf, 28 November 2008.

[8] K. Gerathewohl. *Reinsurance principles and Practice.* Verlag Versicherungswirtschaft, Karlsruhe, 1980.

[9] groovy.codehaus.org. Groovy – closures. `http://groovy.codehaus.org/Closures`. See also the Formal and Informal Guide pages.

[10] Dierk Koenig, Andrew Glover, Paul King, Guillaume Laforge, and Jon Skeet. *Groovy in Action.* Manning Publications Co., Greenwich, CT, USA, 2007.

[11] P. Liebwein. *Klassische und moderne formen der Rückversicherung.* Verlag Versicherungswirtschaft, Karlsruhe, 2009.

[12] Thomas Mack. *Schadenversicherungsmathematik.* Deutsche Gesellschaft für Versicherungsmathematik, Schriftenreihe Angewandte Versicherungsmathematik. Verlag Versicherungswirtschaft, Karlsruhe, 2nd edition, 2002.

[13] Alexander J. McNeil, Rüdiger Frey, and Paul Embrechts. *Quantitative Risk Management.* Princeton University Press, 2005.

[14] Roger B. Nelson. *An introduction to copulas.* Springer Series in Statistics. Springer, 2nd edition, 2006.

[15] David Sehrbrock. Gruppenaufsicht unter Solvency II. *Zeitschrift für die gesamte Versicherungswissenschaft*, 97, Supplement 1:27–36, 2008.

[16] Heinz Stettler, Fritz Eugster, Michael Kuhn, and numerous specialists. *Reinsurance Matters. A manual of the non-life branches.* Swiss Reinsurance Company, 2nd edition, December 2005.

[17] HM Treasury. Supervising insurance groups under Solvency II: a discussion paper. `http://www.abi.org.uk/Solvency_II/15588.pdf`, November 2006.