1. Introduction

The in-memory solution for the Oasis Kernel is called the kernel tools or “ktools”. ktools is an independent “specification” of a set of processes which means that it defines the processing architecture and data structures. The framework is implemented as a set of components called the “reference model” which can then be adapted for particular model or business needs.

The code can be compiled in Linux, POSIX-compliant Windows and native Windows.

Background

The Kernel performs the core Oasis calculations of computing effective damageability distributions, Monte-Carlo sampling of ground up loss, the financial module calculations, which apply insurance policy terms and conditions to the sampled losses, and finally some common catastrophe model outputs.

The early releases of Oasis used a SQL-compliant database to perform all calculations. Release 1.3 included the first “in-memory” version of the Oasis Kernel written in C++ and C to provide streamed calculation at high computational performance, as an alternative to the database calculation. The scope of the in-memory calculation was for the most intensive Kernel calculations of ground up loss sampling and the financial module. This in-memory variant was first delivered as a stand-alone toolkit "ktools" with R1.4.

With R1.5, a Linux-based in-memory calculation back-end was released, using the reference model components of ktools. The range of functionality of ktools was still limited to ground up loss sampling, the financial module and single output workflows, with effective damage distributions and output calculations still being performed in a SQL-compliant database.

In 2016 the functionality of ktools was extended to include the full range of Kernel calculations, including effective damageability distribution calculations and a wider range of financial module and output calculations. The data stream structures and input data formats were also substantially revised to handle multi-peril models, user-definable summary levels for outputs, and multiple output workflows.

Architecture

The Kernel is provided as a toolkit of components (“ktools”) which can be invoked at the command line. Each component is a separately compiled executable with a binary data stream of inputs and/or outputs.

The principle is to stream data through the calculations in memory, starting with generating the damage distributions and ending with calculating the user's required result, before writing the output to disk. This is done on an event-by-event basis, which means at any one time the compute server only has to hold the model data for a single event in its memory, per process. The user can run the calculation across multiple processes in parallel, specifiying the analysis workfkow and number of processes in a script file appropriate to the operating system.

Language

The components can be written in any language as long as the data structures of the binary streams are adhered to. The current set of components have been written in POSIX-compliant C++. This means that they can be compiled in Linux and Windows using the latest GNU compiler toolchain.

Components

The components in the Reference Model can be summarized as follows;

- **Core components** perform the core kernel calculations and are used in the main analysis workflow.
- **Output components** perform specific output calculations at the end of the analysis workflow. Essentially, they produce various statistical summaries of the sampled losses from the core calculation.
- **Data conversion components** are provided to enable users to convert model and exposure data into the required binary formats.
- **Stream conversion components** are provided to inspect the data streams flowing between the core components, for more detailed analysis or debugging purposes.

Usage

Standard piping syntax can be used to invoke the components at the command line. It is the same syntax in Windows DOS, Linux terminal or Cygwin (a Linux emulator for Windows). For example the following command invokes eve, getmodel, gulcalc, fmcalc, summarycalc and eltcalc, and exports an event loss table output to a csv file.

```
$ eve 1 1 | getmodel | gulcalc -r –S100 –i - | fmcalc | summarycalc -f -1 - | eltcalc > elt.csv
```

Example python scripts are provided along with a binary data package in the /examples folder to demonstrate usage of the toolkit. For more guidance on how to use the toolkit, see Workflows.

Go to 2. Data streaming architecture overview

Back to Contents

Table of Contents

1. Introduction

2. Data streaming architecture overview

3. Specification

4. Reference model

4.1 Core components

4.2 Output components

4.3 Data conversion components

4.4 Stream conversion components

5. Financial Module

6. Workflows

Appendix A Random numbers

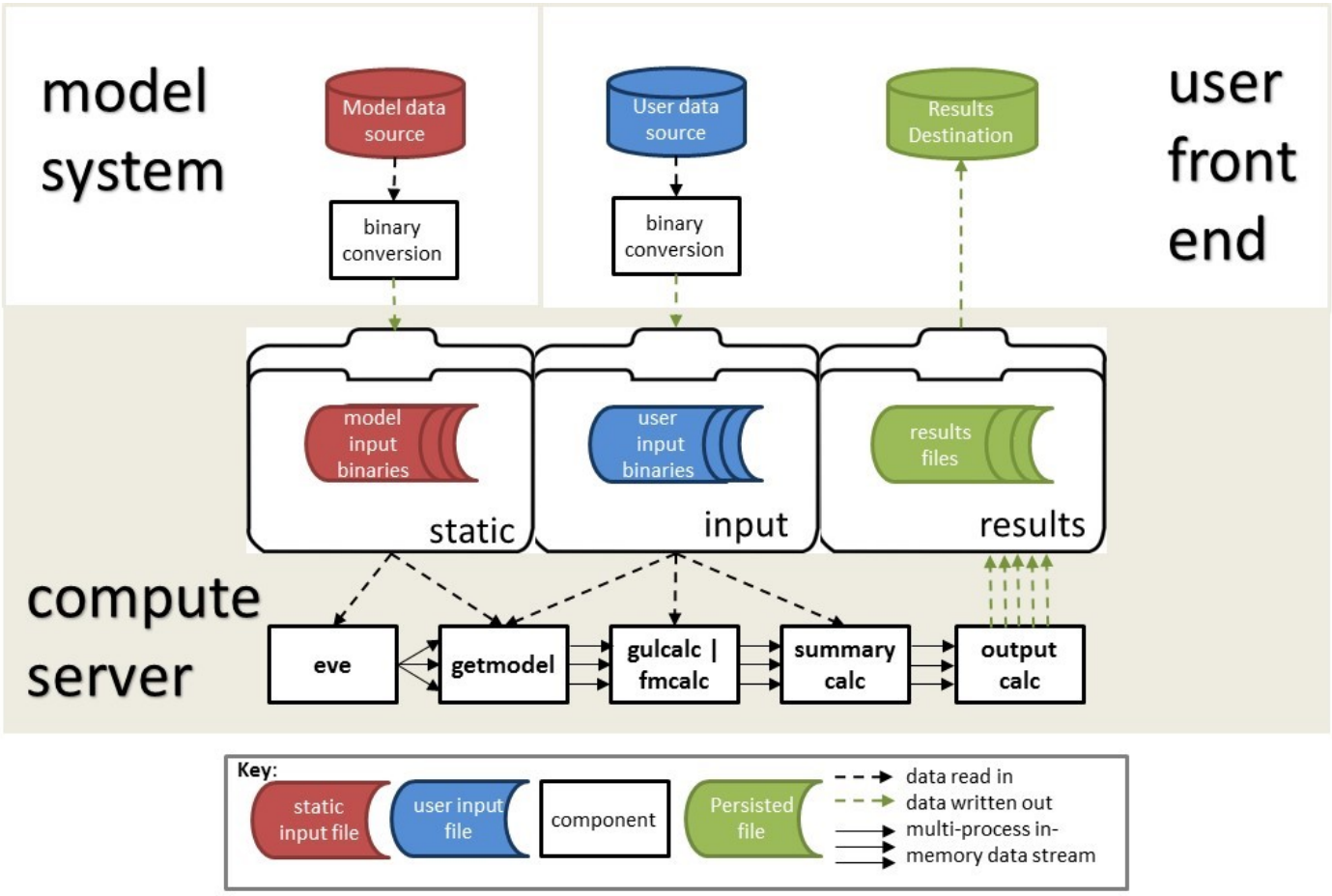
Appendix B FM profiles

Appendix C Multi-peril support

2. Data Streaming Framework Overview

This is the general data streaming framework showing the core components of the toolkit.

Figure 1. Data streaming framework



The architecture consists of;

- A **model system** which provides the model data in the correct binary format. This is a one time set-up task.
- A **user front end**, which provides the users data for an analysis against the model.
- A **compute server**, where the computations are performed. The required model data is held in a 'static' folder, and the required user data is held in an 'input' folder.

The conversion of input data to binary format is shown in the diagram as occurring outside of the compute server, but this could be performed within the compute server. ktools provides a full set of binary conversion tools from csv input files which can be deployed elsewhere.

The in-memory data streams are initiated by the process 'eve' (meaning 'event emitter') and shown by solid arrows. The read/write data flows are shown as dashed arrows.

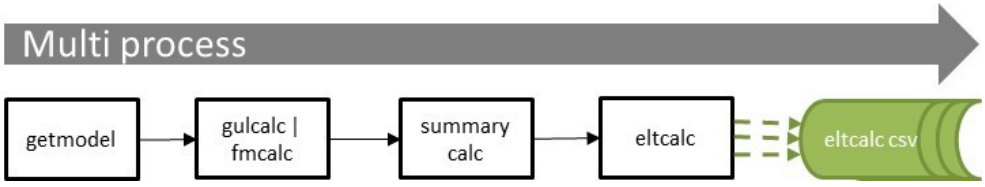
The calculation components are *getmodel*, *gulcalc*, *fmcalc*, *summarycalc* and *outputcalc*. The streamed data passes through the components in memory one event at a time and are written out to a results file on the compute server. The user can then retrieve the results (csvs) and consume them in their BI system.

The reference model demonstrates an implementation of the core calculation components, along with the data conversion components which convert binary files to csv files.

The analysis workflows are controlled by the user, not the toolkit, and they can be as simple or as complex as required.

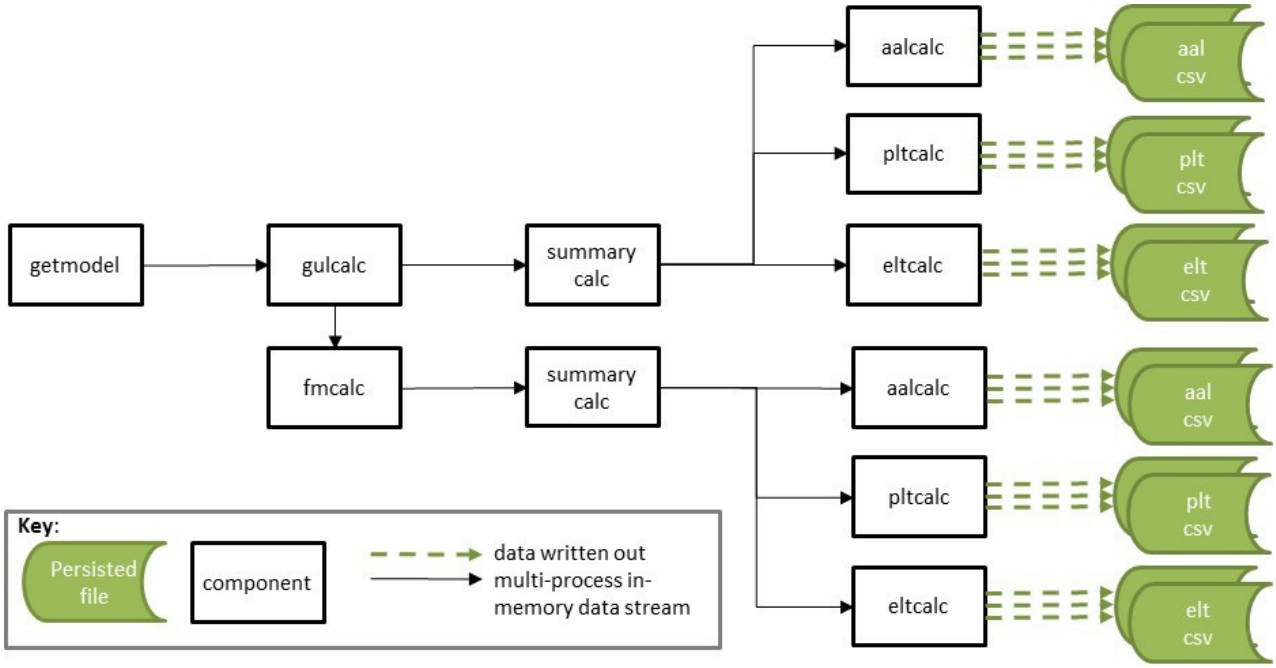
The simplest workflow is single or parallel processing to produce a single result. This minimises the amount of disk I/O at each stage in the calculation, which performs better than saving intermediate results to disk. This workflow is shown in Figure 2.

Figure 2. Single output processing



However it is possible to stream data from one process into to several processes, allowing the calculation of multiple outputs simultaneously, as shown in Figure 3.

Figure 3. Multiple output processing



For multi-output, multi-process workflows, Linux operating systems provide 'named pipes' which in-memory data streams can be diverted to and manipulated as if they were files, and 'tee' which sends a stream from one process into multiple processes. This means the core calculation is not repeated for each output, as it would be if several single-output workflows were run.

Go to 3. Specification

Back to Contents

3. Specification

Introduction

This section specifies the core components and data stream structures in the in-memory kernel.
These components are;

- **eve**
- **getmodel**
- **gulcalc**
- **fmcalc**
- **summarycalc**
- **outputcalc**

The components have a standard input (stdin) and/or output (stdout) data stream structure. eve is the stream-initiating component which only has a standard output stream, whereas outputcalc is a stream-terminating component with only a standard input stream.

The stream data structures have been designed to minimise the volume flowing through the pipeline, using data packet 'headers' to remove redundant data. For example, indexes which are common to a block of data are defined as a header record and then only the variable data records that are relevant to the header key are part of the data stream. The names of the data fields given below are unimportant, only their position in the data stream is important in order to perform the calculations defined in the program.

An implementation of each of the above components is provided in the Reference Model, where usage instructions and command line parameters are provided.

Stream types

The architecture supports multiple stream types. Therefore a developer can define a new type of data stream within the framework by specifying a unique stream_id of the stdout of one or more of the components, or even write a new component which performs an intermediate calculation between the existing components.

The stream_id is the first 4 byte header of the stdout streams. The higher byte is reserved to identify the type of stream (which component), and the 2nd to 4th bytes hold the identifier of the stream. This is used for validation of pipeline commands to report errors if the components are not being used in the correct order.

The current reserved values are as follows;

Higher byte;

Byte 1	Stream type
0	getmodel
1	gulcalc
2	fmcalc
3	summarycalc

Reserved stream_ids;

Byte 1	Bytes 2-4	Description
0	1	getmodel - Reference example for Oasis format effective damageability CDF output
1	1	gulcalc - Reference example for Oasis format item level ground up loss sample output
1	2	gulcalc - Reference example for Oasis format coverage level ground up loss sample output
2	1	fmcalc - Reference example for Oasis format insured loss sample output
3	1	summarycalc - Reference example for Oasis format summary level loss sample output

There are rules about which stream types can be accepted as inputs to the components. These are;

- gulcalc can only take stream 0/1 (getmodel output) as input
- fmcalc can only take stream 1/1 (gulcalc item level output) as input
- summarycalc can take either stream 1/2 (gulcalc coverage level output) or 2/1 (fmcalc standard output) as input
- outputcalc can take only stream 3/1 (summarycalc output) as input

eve

eve is an 'event emitter' and its job is to read a list of events from file and send out a subset of events as a binary data stream. It has no standard input.
eve is used to partition lists of events such that a workflow can be distributed across multiple processes.

Output

Data packet structure

Name	Type	Bytes	Description	Example
event_id	int	4	Oasis event_id	4545

Note that eve has no stream_id header.

Return to top

getmodel

getmodel is the component which generates a stream of effective damageability cdfs for a given set of event_ids.

Input

Same as eve output or a binary file of the same input format can be piped into getmodel.

Output

Stream header packet structure

Name	Type	Bytes	Description	Example
stream_id	int	1/3	Identifier of the data stream type.	0/1

getmodel header packet structure

Name	Type	Bytes	Description	Example
event_id	int	4	Oasis event_id	4545
areaperil_id	int	4	Oasis areaperil_id	345456
vulnerability_id	int	4	Oasis vulnerability_id	345
no_of_bins	int	4	Number of records (bins) in the data package	20

getmodel data packet structure (record repeated no_of_bin times)

Name	Type	Bytes	Description	Example
prob_to	float	4	The cumulative probability at the upper damage bin threshold	0.765
bin_mean	float	4	The conditional mean of the damage bin	0.45

Return to top

gulcalc

gulcalc is the component which calculates ground up loss. It takes the getmodel output as standard input and based on the sampling parameters specified, performs Monte Carlo sampling and numerical integration. The output is a stream of ground up loss samples in Oasis kernel format, with numerical integration mean (sidx=-1) and standard deviation (sidx=-2).

Input

Same as getmodel output or a binary file of the same data structure can be piped into gulcalc.

Output

stream_id 1

Stream header packet structure

Name	Type	Bytes	Description	Example
stream_id	int	1/3	Identifier of the data stream type.	1/1
no_of_samples	int	4	Number of samples	100

gulcalc header packet structure

Name	Type	Bytes	Description	Example

Name	Type	Bytes	Description	Example
event_id	int	4	Oasis event_id	4545
item_id	int	4	Oasis item_id	300

gulcalc data packet structure

Name	Type	Bytes	Description	Example
sidx	int	4	Sample index	10
loss	float	4	The ground up loss for the sample	5675.675

The data packet may be a variable length and so an sidx of 0 identifies the end of the data packet.

There are two values of sidx with special meaning as follows;

sidx	Meaning
-1	numerical integration mean
-2	numerical integration standard deviation

stream_id 2

The main difference here is that the field in the gulcalc header packet structure is coverage_id, representing a grouping of item_id, rather than item_id. The distinction and rationale for having this as a alternative stream is explained in the Reference Model section.

Stream header packet structure

Name	Type	Bytes	Description	Example
stream_id	int	1/3	Identifier of the data stream type.	1/2
no_of_samples	int	4	Number of samples	100

gulcalc header packet structure

Name	Type	Bytes	Description	Example
event_id	int	4	Oasis event_id	4545
coverage_id	int	4	Oasis coverage_id	150

gulcalc data packet structure

Name	Type	Bytes	Description	Example
sidx	int	4	Sample index	10
loss	float	4	The ground up loss for the sample	5675.675

Only the numerical integration mean is output for stream_id 2.

sidx	Meaning
-1	numerical integration mean

Return to top

fmcalc

fmcalc is the component which takes the gulcalc output stream as standard input and applies the policy terms and conditions to produce insured loss samples. The output is a table of insured loss samples in Oasis kernel format, including the insured loss for the mean ground up loss (sidx=-1) and the maximum exposed value (sidx=-3).

Input

Same as gulcalc output or a binary file of the same data structure can be piped into fmcalc.

Output

Stream header packet structure

Name	Type	Bytes	Description	Example
stream_id	int	1/3	Identifier of the data stream type.	2/1
no_of_samples	int	4	Number of samples	100

fmcalc header packet structure

Name	Type	Bytes	Description	Example
event_id	int	4	Oasis event_id	4545
output_id	int	4	Oasis output_id	300

fmcalc data packet structure

Name	Type	Bytes	Description	Example
sidx	int	4	Sample index	10
loss	float	4	The insured loss for the sample	5625.675

The data packet may be a variable length and so a sidx of 0 identifies the end of the data packet.

There are two values of sidx with special meaning as follows;

sidx	Meaning
-1	numerical integration mean
-3	maximum exposed value

In the case of sidx = -1, this is the numerically integrated mean ground up loss passed through the financial terms from gulcalc.

The maximum exposed value, sidx=-3, is generated by fmcalc and this is the TIV of the items passed through the financial terms. This represents a 100% loss scenario to all exposed items after applying the insurance terms and conditions and is used in outputs that require an exposed value.

Return to top

summarycalc

summarycalc is a component which sums the sampled losses from either gulcalc or fmcalc to the users required level(s) for reporting results. This is a simple sum of the loss value by event_id, sidx and summary_id, where summary_id is a grouping of coverage_id for gulcalc or output_id for fmcalc defined in the user's input files.

Input

Same as gulcalc coverage output (stream_id =2) or fmcalc output. A binary file of either data structure can be piped into summarycalc.

Output

Stream header packet structure

Name	Type	Bytes	Description	Example
stream_id	int	1/3	Identifier of the data stream type.	3/1
no_of_samples	int	4	Number of samples	100

summarycalc header packet structure

Name	Type	Bytes	Description	Example
event_id	int	4	Oasis event_id	4545
summary_id	int	4	Oasis summary_id	300
exposure_value	float	4	The TIV (from gulcalc) or maximum exposure value (fmcalc)	600000

summarycalc data packet structure

Name	Type	Bytes	Description	Example
sidx	int	4	Sample index	10
loss	float	4	The insured loss for the sample	5625.675

The data packet may be a variable length and so a sidx of 0 identifies the end of the data packet.

outputcalc

Outputcalc is a general term for an end-of-pipeline component which represents one of a potentially unlimited set of output components. Four examples are provided in the Reference Model. These are;

- eltcalc
- leccalc
- aalcalc/aalsummary
- pltcalc.

outputcalc performs results analysis such as an event loss table or loss exceedance curve on the sampled output from summarycalc. The output is a results table in csv format.

Input

summarycalc stdout. Binary files of the same data structures can be piped into the outputcalc component.

Output

No standard output stream. The results table is exported to a csv file.

Return to top

Go to 4. Reference model

Back to Contents

4. Reference Model Overview

This section provides an overview of the reference model, which is an implementation of each of the components in the framework.

There are four sub-sections which cover the usage and internal processes of each of the reference components;

- **4.1 Core components**
- **4.2 Output components**
- **4.3 Data conversion components**
- **4.4 Stream conversion components**

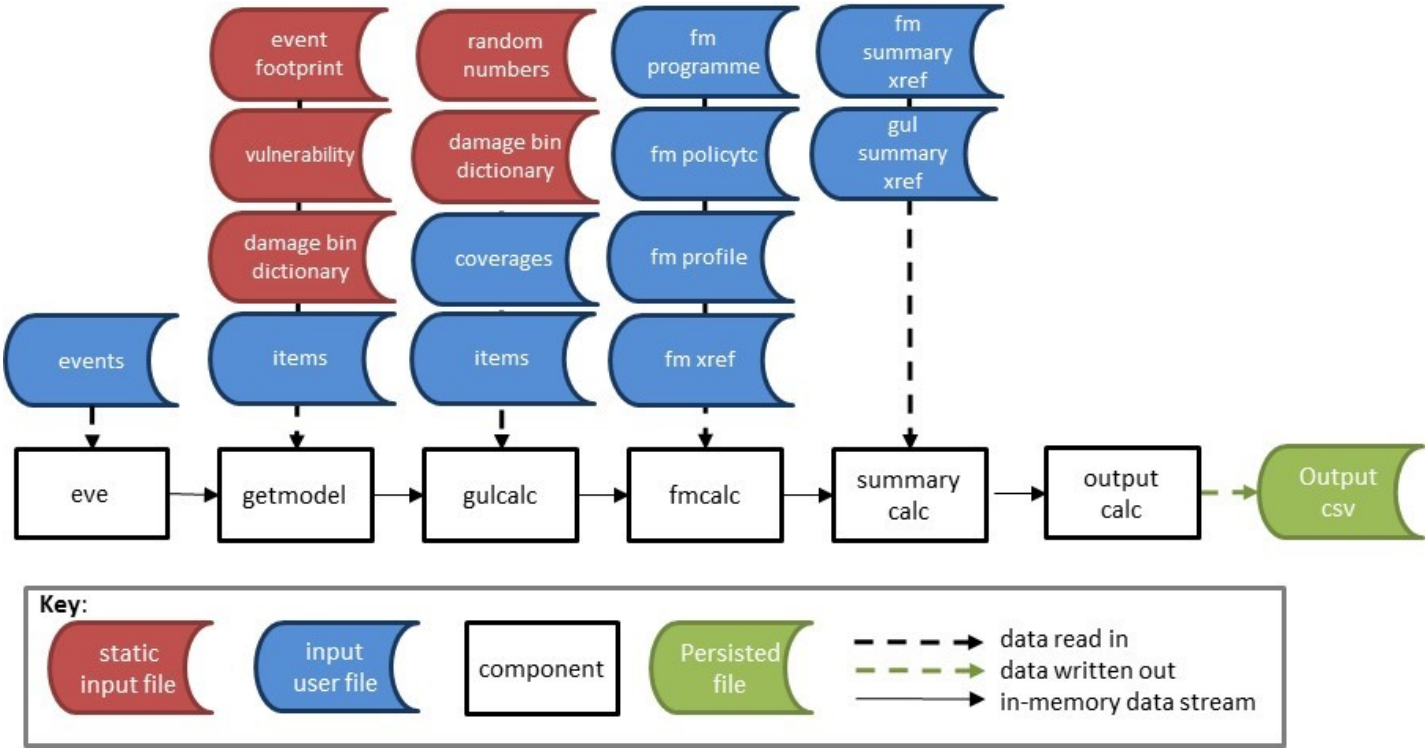
The set of **core components** provided in this release is as follows;

- **eve** is the event distributing utility. Based on the number of events in the input and the number of processes specified as a parameter, eve outputs subsets of the events as a stream. The output streams into getmodel.
- **getmodel** generates a stream of effective damageability cdfs for the input stream of events. It generates cdfs from the model files eventfootprint.bin and vulnerability.bin, and the user's exposures file which is called items.bin. getmodel streams into gulcalc or can be output to a binary file.
- **gulcalc** performs the ground up loss sampling calculations and numerical integration. The output is a stream of sampled ground up losses. This can be output to a binary file or streamed into fmcalc or summarycalc.
- **fmcalc** performs the insured loss calculations on the ground up loss samples, mean, and total insured value. The output is a stream of insured loss samples. The result can be output to a binary file or streamed into summarycalc.
- **summarycalc** performs a summing of sampled losses according to the user's reporting requirements. For example this might involve summing coverage losses to regional level, or policy losses to portfolio level. The output is sampled loss by event_id and summary_id, which represents a meaningful group of losses to the user.

The standard input and standard output data streams for the core components are covered in the Specification.

Figure 1 shows the core components workflow and the required data input files.

Figure 1. Core components workflow and required data



The model **static** data for the core workflow, shown as red source files, are the event footprint, vulnerability, damage bin dictionary and random number file. These are stored in the '**static**' sub-directory of the working folder.

The user / analysis **input** data for the core workflow, shown as blue source files, are the events, items,

coverages, fm programme, fm policytc, fm profile, fm xref, fm summary xref and gul summary xref files. These are stored in the '**input**' sub-directory of the working folder.

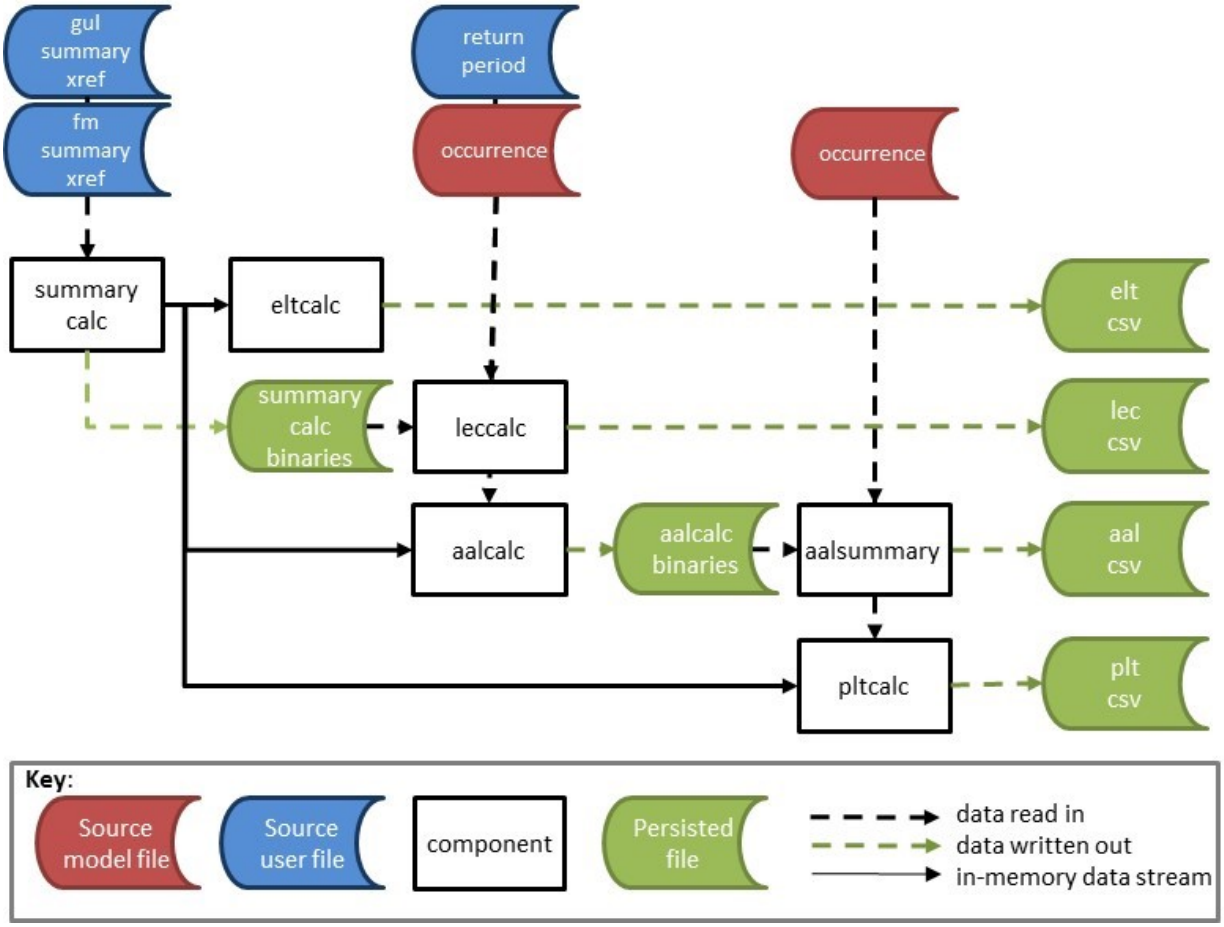
These are all Oasis kernel format data objects with prescribed formats. Note that the events are a user input rather than a static input because the user could choose to run a subset of the full list of events, or even just one event. Usually, though, the whole event set will be run.

The **output components** are various implementations of outputcalc, as described in general terms in the Specification. The results are written directly into csv file as there is no downstream processing.

- **eltcalc** generates an event loss table from the sampled losses from summarycalc. It contains sample mean and standard deviation, and total exposed value for each event at the given summary level.
- **leccalc** generates loss exceedance curve from the sampled losses from summarycalc. There are 8 variants of curves with small differences in the output format but the common output fields are summary_id, return period, and loss exceedance threshold. This output is only available for models which provide an occurrence file.
- **pltcalc** generates a period loss table from the sampled losses from summarycalc. It contains sample mean and standard deviation, and total exposed value for each event and each period (for example a year) at the given summary level. It also contains a date field, corresponding to the date of the event occurrence. This output is only available for models which provide an occurrence file.
- **aalcalc** and **aalsummary** generate the average annual loss and standard deviation of loss from the sampled losses from summarycalc, for each summary_id. The output also contains total exposed value for each summary level, which is the maximum of the total exposed value across all simulated periods. This output is only available for models which provide an occurrence file.

The files required for the output components are shown in Figure 2.

Figure 2. Output workflows and required data



The **data conversion components** section covers the formats of all of the required data files and explains how to convert data in csv format into binary format, and vice versa.

Finally, the **stream conversion components** section explains how to convert the binary data stream output to csv, plus how to convert gulcalc data in csv format into binary format. These components are useful when working with individual components at a more detailed level.

Return to top

Go to 4.1 Core Components section

Back to Contents

4.1 Core Components

eve

---

eve takes as input a list of event ids in binary format and generates a partition of event ids as a binary data stream, according to the parameters supplied.

Stream_id

None. The output stream is a simple list of event_ids (4 byte integers).

Parameters

- process number - determines which partition of events to output to a single process
- number of processes - determines the total number of partitions of events to distribute to processes

Usage

```
$ eve [parameters] > [output].bin
$ eve [parameters] | getmodel | gulcalc [parameters] > [stdout].bin
```

Example

```
$ eve 1 2 > events1_2.bin
$ eve 1 2 | getmodel | gulcalc -r -S100 -i - > gulcalc1_2.bin
```

In this example, the events from the file events.bin will be read into memory and the first half (partition 1 of 2) would be streamed out to binary file, or downstream to a single process calculation workflow.

Internal data

The program requires an event binary. The file is picked up from the input sub-directory relative to where the program is invoked and has the following filename;

- input/events.bin

The data structure of events.bin is a simple list of event ids (4 byte integers).

Return to top

getmodel

---

getmodel generates a stream of effective damageability distributions (cdfs) from an input list of events. Specifically, it combines the probability distributions from the model files, eventfootprint.bin and vulnerability.bin, to generate effective damageability cdfs for the subset of exposures contained in the items.bin file and converts them into a binary stream.

This is reference example of the class of programs which generates the damage distributions for an event set and streams them into memory. It is envisaged that model developers who wish to use the toolkit as a back-end calculator of their existing platforms can write their own version of getmodel, reading in their own source data and converting it into the standard output stream. As long as the standard input and output structures are adhered to, each program can be written in any language and read any input data.

Stream_id

Byte 1	Bytes 2-4	Description
0	1	getmodel reference example

Parameters

None

Usage

```
$ [stdin component] | getmodel | [stout component]
$ [stdin component] | getmodel > [stdout].bin
$ getmodel < [stdin].bin > [stdout].bin
```

Example

```
$ eve 1 1 | getmodel | gulcalc -r -S100 -i gulcalci.bin
$ eve 1 1 | getmodel > getmodel.bin
$ getmodel < events.bin > getmodel.bin
```

Internal data

The program requires the footprint binary and index file for the model, the vulnerability binary model file, and the items file representing the user's exposures. The files are picked up from sub-directories relative to where the program is invoked, as follows;

- static/footprint.bin
- static/footprint.idx
- static/vulnerability.bin
- static/damage_bin_dict.bin
- input/items.bin

The getmodel output stream is ordered by event and streamed out in blocks for each event.

Calculation

The program filters the footprint binary file for all 'areaperil_id's which appear in the items file. This selects the event footprints that affect the exposures on the basis on their location. Similarly the program filters the vulnerability file for 'vulnerability_id's that appear in the items file. This selects damage distributions which are relevant for the exposures. Then the intensity distributions from the footprint file and conditional damage distributions from the vulnerability file are convolved for every combination of areaperil_id and vulnerability_id in the items file. The effective damage

probabilities are calculated by a sum product of conditional damage probabilities with intensity probabilities for each event, areaperil, vulnerability combination and each damage bin. The resulting discrete probability distributions are converted into discrete cumulative distribution functions 'cdfs'. Finally, the damage bin mid-point from the damage bin dictionary ('interpolation' field) is read in as a new field in the cdf stream as 'bin_mean'. This field is the conditional mean damage for the bin and it is used to facilitate the calculation of mean and standard deviation in the gulcalc component.

Return to top

gulcalc

The gulcalc program performs Monte Carlo sampling of ground up loss using interpolation of uniform random numbers against the effective damage cdf, and calculates mean and standard deviation by numerical integration of the cdfs. The sampling methodology has been extended beyond linear interpolation to include point value sampling and quadratic interpolation. This supports damage bin intervals which represent a single discrete damage value, and damage distributions with cdfs that are described by a piecewise quadratic function.

Stream_id

Byte 1	Bytes 2-4	Description
1	1	gulcalc reference example

Parameters

Required parameters are;

- -R{number} or -r. Number of random numbers to generate or read random numbers from file
- -S{number}. Number of samples
- -c or -i {destination}. There are two alternative streams, 'coverage' or 'item'. The destination is either a filename or named pipe, or use - for standard output.

Optional parameters are;

- -L{number} loss threshold (optional) excludes losses below the threshold from the output stream
- -d debug mode - output random numbers rather than losses (optional)

Usage

```
$ [stdin component] | gulcalc [parameters] | [stout component]
$ [stdin component] | gulcalc [parameters]
$ gulcalc [parameters] < [stdin].bin
```

Example

```
$ eve 1 1 | getmodel | gulcalc -R1000000 -S100 -i - | fmcalc > fmcalc.bin
$ eve 1 1 | getmodel | gulcalc -R1000000 -S100 -c - | summarycalc -g -1 summarycalc1.bin
$ eve 1 1 | getmodel | gulcalc -r -S100 -i gulcalci.bin
```

```
$ eve 1 1 | getmodel | gulcalc -r -S100 -c gulcalcc.bin
$ gulcalc -r -S100 -c gulcalcc.bin < getmodel.bin
```

Internal data

The program requires the damage bin dictionary binary for the static folder and the item and coverage binaries from the input folder. The files are found in the following locations relative to the working directory with the filenames;

- static/damage_bin_dictionary.bin
- input/items.bin
- input/coverages.bin

If the user specifies -r as a parameter, then the program also picks up a random number file from the static directory. The filename is;

- static/random.bin

Calculation

The stdin stream is a block of cdfs which are ordered by event_id, areaperil_id, vulnerability_id and bin_index asccending, from getmodel. The gulcalc program constructs a cdf for each item, based on matching the areaperil_id and vulnerability_id from the stdin and the item file.

For each item cdf and for the number of samples specified, the program draws a random number and uses it to sample ground up loss from the cdf using one of three methods, as follows;

For a given damage interval corresponding to a cumulative probability interval that each random number falls within;

- If the conditional mean damage (of the cdf) is the mid-point of the damage bin interval (of the damage bin dictionary) then the gulcalc program performs linear interpolation.
- If the conditional mean damage is equal to the lower and upper damage threshold of the damage bin interval (i.e the bin represents a damage value, not a range) then that value is sampled.
- Else, the gulcalc program performs quadrative interpolation using the bin_mean to calculate the quadratic equation in the damage interval.

An example of the three cases and methods is given below;

bin_from	bin_to	bin_mean	Method used
0.1	0.2	0.15	Linear interpolation
0.1	0.1	0.1	Sample bin value
0.1	0.2	0.14	Quadratic interpolation

If the -R parameter is used along with a specified number of random numbers then random numbers used for sampling are generated on the fly for each event and group of items which have a common group_id using the Mersenne twister psuedo random number generator (the default RNG of the C++ v11 compiler). if the -r parameter is used, gulcalc reads a random number from the provided random number file. See Random Numbers for more details.

Each sampled damage is multiplied by the item TIV, looked up from the coverage file.

The mean and standard deviation of ground up loss is also calculated. For each cdf, the mean and standard deviation of damage is calculated by numerical integration of the effective damageability probability distribution and the result is multiplied by the TIV. The results are included in the output to the stdout stream as sidx=-1 (mean) and sidx=-2 (standard deviation), for each event and item (or coverage).

At this stage, there are two possible output streams, as follows;

- stream_id=1: If the -i parameter is specified, then the ground up losses for the items are output to the stream (stream_id 1).
- stream_id=2: If the -c parameter is specified, then the ground up losses for the items are grouped by coverage_id and capped to the coverage TIV, and the ground up losses for the coverages are output to the stream (stream_id 2).

The purpose of the coverage stream is to cap losses to the coverage TIV where items represent damage to the same coverage from different perils, where overall damage cannot exceed 100% of the TIV.

Return to top

fmcalc

---

fmcalc is the reference implementation of the Oasis Financial Module. It applies policy terms and conditions to the ground up losses and produces insured loss sample output. It reads in losses from gulcalc at item level only (stream_id = 1).

Stream_id

Byte 1	Bytes 2-4	Description
2	1	fmcalc reference example

Parameters

There are no parameters as all of the information is taken from the gulcalc stdout stream and fm input data.

Usage

```
$ [stdin component] | fmcalc | [stout component]
$ [stdin component] | fmcalc > [stdout].bin
$ fmcalc < [stdin].bin > [stdout].bin
```

Example

```
$ eve 1 1 | getmodel | gulcalc -r -S100 -i - | fmcalc | summarycalc -f -2 - | eltcalc > elt.csv
$ eve 1 1 | getmodel | gulcalc -r -S100 -i - | fmcalc > fmcalc.bin
$ fmcalc < gulcalc.bin > fmcalc.bin
```

Internal data

The program requires the item, coverage and fm input data files, which are Oasis abstract data objects that describe an insurance programme. This data is picked up from the following files relative to the working directory;

- input/items.bin
- input/coverages.bin
- input/fm_programme.bin
- input/fm_policytc.bin
- input/fm_profile.bin
- input/fm_xref.bin

Calculation

See Financial Module

Return to top

summarycalc

The purpose of summarycalc is firstly to aggregate the samples of loss to a level of interest for reporting, thereby reducing the volume of data in the stream. This is a generic first step which precedes all of the downstream output calculations. Secondly, it unifies the formats of the gulcalc and fmcalc streams, so that they are transformed into an identical stream type for downstream outputs. Finally, it can generate up to 10 summary level outputs in one go, creating multiple output streams or files.

The output is similar to the gulcalc or fmcalc input which are losses are by sample index and by event, but the ground up or insured loss input losses are grouped to an abstract level represented by a summary_id. The relationship between the input identifier and the summary_id are defined in cross reference files called **gulsummaryxref** and **fmsummaryxref**.

Stream_id

Byte 1	Bytes 2-4	Description
3	1	summarycalc reference example

Parameters

The input stream should be identified explicitly as -g for gulcalc stream or -f for fmcalc stream.

summarycalc supports up to 10 concurrent outputs. This is achieved by explictly directing each output to a named pipe, file, or to standard output.

For each output stream, the following tuple of parameters must be specified for at least one summary set;

- summaryset_id number. valid values are -0 to -9
- destination pipe or file. Use either - for standard output, or {name} for a named pipe or file.

For example the following parameter choices are valid;

```
$ summarycalc -g -1 -
'outputs results for summaryset 1 to standard output
$ summarycalc -g -1 summarycalc1.bin
'outputs results for summaryset 1 to a file (or named pipe)
$ summarycalc -g -1 summarycalc1.bin -2 summarycalc2.bin
'outputs results for summaryset 1 and 2 to a file (or named pipe)
```

Note that the summaryset_id relates to a summaryset_id in the required input data file **gulsummaryxref.bin** or **fmsummaryxref.bin** for a gulcalc input stream or a fmcalc input stream, respectively, and represents a user specified summary reporting level. For example summaryset_id = 1 represents portfolio level, summaryset_id = 2 represents zipcode level and summaryset_id 3 represents site level.

Usage

```
$ [stdin component] | summarycalc [parameters] | [stdout component]
$ [stdin component] | summarycalc [parameters]
$ summarycalc [parameters] < [stdin].bin
```

Example

```
$ eve 1 1 | getmodel | gulcalc -r -S100 -c - | summarycalc -g -1 - | eltcalc > eltcalc.csv
$ eve 1 1 | getmodel | gulcalc -r -S100 -c - | summarycalc -g -1 gulsummarycalc.bin
$ summarycalc -f -1 fmsummarycalc.bin < fmcalc.bin
```

Internal data

The program requires the gulsummaryxref file for gulcalc input (-g option), or the fmsummaryxref file for fmcalc input (-f option). This data is picked up from the following files relative to the working directory;

- input/gulsummaryxref.bin
- input/fmsummaryxref.bin

Calculation

summarycalc takes either ground up loss (by coverage) or insured loss samples (by policy) as input and aggregates them to a user-defined summary reporting level. The output is similar to the input which are losses are by sample index and by event, but the ground up or insured losses are grouped to an abstract level represented by a summary_id. The relationship between the input identifier, coverage_id for gulcalc or output_id for fmcalc, and the summary_id are defined in the internal data files.

summarycalc also calculates the maximum exposure value by summary_id and event_id. This is carried through in the stream header and output by eltcalc, aalcalc and pltcalc. For ground up losses this is the sum of TIV for all coverages which have at least one non-zero sampled ground up loss for a given event. For insured losses, this is the sum of the losses for sample index -3 from the fmcalc stdout stream. Sample index -3 is the TIV of the items in the programme which have been passed through the financial module calculations to take account of the deductibles and limits. Essentially, it is a 100% damage scenario for all items.

Return to top

Go to 4.2 Output Components section

Back to Contents

4.2 Output Components

eltcalc

The program calculates sample mean and standard deviation of loss by summary_id and by event_id.

Parameters

None

Usage

```
$ [stdin component] | eltcalc > elt.csv
$ eltcalc < [stdin].bin > elt.csv
```

Example

```
$ eve 1 1 | getmodel | gulcalc -r -S100 -c - | summarycalc -g -1 - | eltcalc > elt.csv
$ eltcalc < summarycalc.bin > elt.csv
```

Internal data

No additional data is required, all the information is contained within the input stream.

Calculation

For each summary_id and event_id, the sample mean and standard deviation is calculated from the sampled losses from the summarycalc stream and output to file. The exposure_value, which is carried in the event_id, summary_id header of the stream is also output. The type field, with value = 2 means that the loss statistics come from the samples.

A type 1 record is also included in the output, which gives the analytical (numerically integrated) mean loss. The analytical standard deviation is not calculated and is set to zero.

When zero samples are run, only type 1 losses are output, and both type 1 and 2 are output when more than one sample is run.

Output

csv file with the following fields;

Name	Type	Bytes	Description	Example
summary_id	int	4	summary_id representing a grouping of losses	10
type	int	4	1 for analytical mean, 2 for sample mean	2
event_id	int	4	Oasis event_id	45567
mean	float	4	mean	1345.678
standard_deviation	float	4	sample standard deviation, or 0 for type 1	945.89

Name	Type	Bytes	Description	Example
exposure_value	float	4	exposure value for summary_id affected by the event	70000

Return to top

leccalc

Loss exceedance curves, also known as exceedance probability curves, are computed by a rank ordering a set of losses by period and computing the probability of exceedance for each level of loss in any given period based on relative frequency. Losses are first assigned to periods by reference to the **occurrence** file which contains the event occurrences in each period. Event losses are summed within each period for an aggregate loss exceedance curve, or the maximum of the event losses in each period is taken for an occurrence loss exceedance curve. From this point, there are a few variants available as follows;

- Full uncertainty
- Wheatsheaf
- Sample mean
- Wheatsheaf mean

Parameters

- -K{sub-directory}. The subdirectory of /work containing the input summarycalc binary files.
- -r. Use return period file - use this parameter if you are providing a file with a specific list of return periods. If this file is not present then all calculated return periods will be returned, for losses greater than zero.
  Then the following tuple of parameters must be specified for at least one analysis type;
- Analysis type. Use -F for Full Uncertainty Aggregate, -f for Full Uncertainty Occurrence, -W for Wheatsheaf Aggregate, -w for Wheatsheaf Occurrence, -S for Sample Mean Aggregate, -s for Sample Mean Occurrence, -M for Mean of Wheatsheaf Aggregate, -m for Mean of Wheatsheaf Occurrence
- Output filename

Usage

```
$ leccalc [parameters] > lec.csv
```

Example

```
'First generate summarycalc binaries by running the core workflow, for the required summary set
$ eve 1 2 | getmodel | gulcalc -r -S100 -c - | summarycalc -g -1 - > work/summary1/summarycalc1.bin
$ eve 2 2 | getmodel | gulcalc -r -S100 -c - | summarycalc -g -1 - > work/summary1/summarycalc2.bin
'Then run leccalc, pointing to the specified sub-directory of work containing summarycalc binaries.
$ leccalc -r -Ksummary1 -F lec_full_uncertainty_agg.csv -f lec_full_uncertainty_occ.csv
```

Internal data

leccalc requires the occurrence.bin file;

- static/occurrence.bin

leccalc does not have a standard input that can be streamed in. Instead, it reads in summarycalc binary data from a file in a fixed location. The format of the binaries

must match summarycalc standard output. The location is in the 'work' subdirectory of the present working directory. For example;

- work/summarycalc1.bin
- work/summarycalc2.bin
- work/summarycalc3.bin

The user must ensure the work subdirectory exists. The user may also specify a subdirectory of /work to store these files. e.g.

- work/summaryset1/summarycalc1.bin
- work/summaryset1/summarycalc2.bin
- work/summaryset1/summarycalc3.bin

The reason for leccalc not having an input stream is that the calculation is not valid on a subset of events, i.e. within a single process when the calculation has been distributed across multiple processes. It must bring together all event losses before assigning event losses to periods and ranking losses by period. The summarycalc losses for all events (all processes) must be written to the /work folder before running leccalc.

Finally, and optionally, if the user would like only certain return period losses in the output (-r parameter), then a returnperiods file may be provided. If provided then the following file must exist;

- input/returnperiods.bin

Losses for return periods in the returnperiods file that are not directly calculated by leccalc based on the model's total number of periods are calculated by linear interpolation of the two bounding return period losses. If the requested return period is below the range of the calculated set of return periods, then the loss is set to zero.

If the -r option is not used, then all calculated return period losses will be returned.

Calculation

All files with extension .bin from the specified subdirectory are read into memory, as well as the occurrence.bin. The summarycalc losses are grouped together and sampled losses are assigned to period according to which period the events occur in.

If multiple events occur within a period;

- For **aggregate** loss exceedance curves, the sum of losses is assigned to the period.
- For **occurrence** loss exceedance curves, the maximum loss is assigned to the period.

Then the calculation differs by lec type, as follows;

- Full uncertainty - all sampled losses by period are rank ordered to produce a single loss exceedance curve. This treats each sample as if it were another period of losses in an extended timeline.
- Wheatsheaf - losses by period are rank ordered for each sample, which produces many loss exceedance curves - one for each sample across the same timeline.
- Sample mean - the losses by period are first averaged across the samples, and then a single loss exceedance curve is created from the period sample mean

losses.

- Wheatsheaf mean - the loss exceedance curves from the Wheatsheaf are averaged across each return period, which produces a single loss exceedance curve.

- For the Sample mean and Wheatsheaf mean curves, the analytical mean loss (sidx = -1) is output as a separate exceedance probability curve. If the calculation is run with 0 samples, then leccalc will still return the analytical mean loss exceedance curve. The 'type' field in the output identifies the type of loss exceedance curve, which is 1 for analytical mean, and 2 for the mean calculated from the samples.

The total number of periods is carried in the header of the occurrence file. The ranked losses represent the first, second, third, etc.. largest loss periods within the total number of periods of say 10,000 years. The relative frequency of these periods of loss is interpreted as the probability of loss exceedance, that is to say that the top ranked loss has an exceedance probability of 1 in 10000, or 0.01%, the second largest loss has an exceedance probability of 0.02%, and so on. In the output file, the exceedance probability is expressed as a return period, which is the reciprocal of the exceedance probability multiplied by the total number of periods. Only non-zero loss periods are returned.

Output

csv file with the following fields;

Full uncertainty loss exceedance curve

Name	Type	Bytes	Description	Example
summary_id	int	4	summary_id representing a grouping of losses	10
return_period	float	4	return period interval	250
loss	float	4	loss exceedance threshold for return period	546577.8

Wheatsheaf loss exceedance curve

Name	Type	Bytes	Description	Example
summary_id	int	4	summary_id representing a grouping of losses	10
sidx	int	4	Oasis sample index	50
return_period	float	4	return period interval	250
loss	float	4	loss exceedance threshold for return period	546577.8

Sample mean and Wheatsheaf mean loss exceedance curve

Name	Type	Bytes	Description	Example
summary_id	int	4	summary_id representing a grouping of losses	10
type	int	4	1 for analytical mean, 2 for sample mean	2
return_period	float	4	return period interval	250
loss	float	4	loss exceedance threshold for return period	546577.8

Users are able to enter the return periods they wish to be returned by specifying a return period file.

Return to top

pltcalc

The program outputs sample mean and standard deviation by summary_id, event_id and period_no. It also outputs an event occurrence date.

Parameters

None

Usage

```
$ [stdin component] | pltcalc > plt.csv
$ pltcalc < [stdin].bin > plt.csv
```

Example

```
$ eve 1 1 1 | getmodel | gulcalc -r -S100 -C1 | summarycalc -1 - | pltcalc > plt.csv
$ pltcalc < summarycalc.bin > plt.csv
```

Calculation

The occurrence.bin file is read into memory. For each summary_id, event_id and period_no, the sample mean and standard deviation is calculated from the sampled losses in the summarycalc stream and output to file. The exposure_value, which is carried in the event_id, summary_id header of the stream is also output, as well as the date field(s) from the occurrence file.

Output

There are two output formats, depending on whether an event occurrence date is an integer offset to some base date that most external programs can interpret as a real date, or a calendar day in a numbered scenario year. The output format will depend on the format of the date fields in the occurrence.bin file.

In the former case, the output format is;

Name	Type	Bytes	Description	Example
event_id	int	4	Oasis event_id	45567
period_no	int	4	identifying an abstract period of time, such as a year	56876
mean	float	4	sample mean	1345.678
standard_deviation	float	4	sample standard deviation	945.89
exposure_value	float	4	exposure value for summary_id affected by the event	70000
date_id	int	4	the date_id of the event occurrence	28616

Using a base date of 1/1/1900 the integer 28616 is interpreted as 16/5/1978.

In the latter case, the output format is;

Name	Type	Bytes	Description	Example
event_id	int	4	Oasis event_id	45567
period_no	int	4	identifying an abstract period of time, such as a year	56876
mean	float	4	sample mean	1345.678
standard_deviation	float	4	sample standard deviation	945.89
exposure_value	float	4	exposure value for summary_id affected by the event	70000
occ_year	int	4	the year number of the event occurrence	56876
occ_month	int	4	the month of the event occurrence	5
occ_year	int	4	the day of the event occurrence	16

Return to top

aalcalc

---

aalcalc produces binary files which contain the sum of means, and sum of squared means across all periods for each summary_id. This is the first stage in the calculation of average annual loss and standard deviation of loss, which requires the aggregation of the sum of means and sum of squared means across all periods. Like the leccalc component, the final calculation cannot be performed within a single process containing a subset of the data, in cases where the events have been distributed across multiple processes. Instead, the aalcalc binaries returned from all processes are read back into memory by the aalsummary component which completes the calculation of average annual loss and standard deviation.

Two types of mean are calculated in aalcalc; analytical mean (type 1) and sample mean (type 2). If the analysis is run with zero samples, then only type 1 means are returned by aalcalc.

Parameters

None

Usage

```
$ [stdin component] | aalcalc > aal.bin
$ aalcalc < [stdin].bin > aal.bin
```

Example

```
$ eve 1 1 | getmodel | gulcalc -r -S100 -c - | summarycalc -g -1 - | aalcalc > aal1.bin
$ aalcalc < summarycalc.bin > aal.bin
```

Output

binary file containing the following fields;

Name	Type	Bytes	Description	Example
summary_id	int	4	summary_id representing a grouping of losses	10

Name	Type	Bytes	Description	Example
type	int	4	1 for analytical mean, 2 for mean calculated from samples	1
mean	float	8	sum of period mean losses	67856.9
squared_mean	float	8	sum of squared period mean losses	546577.8
max_exposure_value	float	8	maximum exposure value across all periods	10098730

Internal data

The program requires the occurrence file;

- static/occurrence.bin

Calculation

The occurrence file is read into memory. From the summarycalc stream data, the event sample means and sample means squared are computed for each summary_id. The sample means and squared sampled means are summed by period and summary_id, according to which events are assigned to periods in the occurrence data. Then the means and squared means are summed across the periods, for each summary_id. The exposure_value, which is held at an event_id, summary_id in the summarycalc header is also accumulated by summary_id and period, and then across periods. In this case however, it is the maximum exposure value which is carried through the accumulations, not the sum.

Return to top

aalsummary

---

aalsummary is the final stage calculation for average annual loss and standard deviation of loss. It reads in all aalcalc binary files which contain the sum of means, and sum of squared means across different sets of periods for each summary_id, and then computes overall average annual loss, standard deviation of loss and maximum exposure value across all periods.

Two types of aal and standard deviation of loss are calculated in aalsummary; analytical (type 1) and sample (type 2). If the analysis is run with zero samples, then only type 1 statistics are returned by aalsummary.

Parameters

- -K{sub-directory}. The sub-directory of /work containing the input aalcalc binary files.

Usage

```
$ aalsummary [parameters] > aal.csv
```

Example

```
'First generate aalcalc binaries by running the core workflow, for the required summary set
$ eve 1 2 | getmodel | gulcalc -r -S100 -c - | summarycalc -g -1 - | aalcalc > work/summary1/aal/aalcalc1.bin
$ eve 2 2 | getmodel | gulcalc -r -S100 -c - | summarycalc -g -1 - | aalcalc > work/summary1/aal/aalcalc2.bin
'Then run aalsummary, pointing to the specified sub-directory of work containing the aalcalc binaries.
```

```
$ aalsummary -Ksummary1/aal > aal.csv
```

Output

csv file containing the following fields;

Name	Type	Bytes	Description	Example
summary_id	int	4	summary_id representing a grouping of losses	10
type	int	4	1 for analytical statistics, 2 for sample statistics	1
mean	float	8	average annual loss	6785.9
standard_deviation	float	8	standard deviation of loss	54657.8
exposure_value	float	8	maximum exposure value across all periods	10098730

Internal data

The program requires the occurrence file;

- static/occurrence.bin

aalsummary does not have a standard input that can be streamed in. Instead, it reads in aalcalc binary data from a file in a fixed location. The location is in the 'work' subdirectory of the present working directory. For example;

- work/aalcalc1.bin
- work/aalcalc2.bin
- work/aalcalc3.bin

The user must ensure the work subdirectory exists. The user may also specify a subdirectory of /work to store these files. e.g.

- work/summaryset1/aalcalc1.bin
- work/summaryset1/aalcalc2.bin
- work/summaryset1/aalcalc3.bin

The aalcalc losses for all events (all processes) must be written to the /work folder before running aalsummary.

Calculation

The aalcalc binaries and the occurrence file are read into memory. The sum of means and sum of squared means are summed by type and summary_id. Then using the total number of periods from the header of the occurrence data, the overall mean and standard deviation is calculated. The exposure_value is also accumulated by summary_id and type by taking the maximum value.

[Return to top]

[Go to 4.3 Data conversion components section]

[Back to Contents]

4.3 Data conversion components

The following components convert input data in csv format to the binary format required by the calculation components in the reference model;

Static data

- **damagebintobin** converts the damage bin dictionary.
- **footprinttobin** converts the event footprint.
- **occurrencetobin** converts the event occurrence data.
- **randtobin** converts a list of random numbers.
- **vulnerabilitytobin** converts the vulnerability data.

Input data

- **coveragetobin** converts the coverages data.
- **evetobin** converts a list of event_ids.
- **itemtobin** converts the items data.
- **gulsummaryxreftobin** converts the gul summary xref data.
- **fmpolicytctobin** converts the fm policytc data.
- **fmprogrammetobin** converts the fm programme data.
- **fmprofiletobin** converts the fm profile data.
- **fmsummaryxreftobin** converts the fm summary xref data.
- **fmxreftobin** converts the fm xref data.
- **returnperiodtobin** converts a list of return periods.

These components are intended to allow users to generate the required input binaries from csv independently of the original data store and technical environment. All that needs to be done is first generate the csv files from the data store (SQL Server database, etc).

The following components convert the binary input data required by the calculation components in the reference model into csv format;

Static data

- **damagebintocsv** converts the damage bin dictionary.
- **footprinttocsv** converts the event footprint.
- **occurrencetocsv** converts the event occurrence data.
- **randtocsv** converts a list of random numbers.
- **vulnerabilitytocsv** converts the vulnerability data.

Input data

- **coveragetocsv** converts the coverages data.
- **evetocsv** converts a list of event_ids.
- **itemtocsv** converts the items data.
- **gulsummaryxreftocsv** converts the gul summary xref data.
- **fmpolicytctocsv** converts the fm policytc data.
- **fmprogrammetocsv** converts the fm programme data.
- **fmprofiletocsv** converts the fm profile data.
- **fmsummaryxreftocsv** converts the fm summary xref data.
- **fmxreftocsv** converts the fm xref data.
- **returnperiodtocsv** converts a list of return periods.

These components are provided for the convenience of viewing the data and debugging.

Static data

damage bin dictionary

The damage bin dictionary is a reference table in Oasis which defines how the effective damageability cdfs are discretized on a relative damage scale (normally between 0 and 1). It is required by getmodel and gulcalc and must have the following location and filename;

- static/damage_bin_dict.bin

File format

The csv file should contain the following fields and include a header row.

Name	Type	Bytes	Description	Example
bin_index	int	4	Identifier of the damage bin	1
bin_from	float	4	Lower damage threshold for the bin	0.01
bin_to	float	4	Upper damage threshold for the bin	0.02
interpolation	float	4	Interpolation damage value for the bin (usually the mid-point)	0.015
interval_type	int	4	Identifier of the interval type, e.g. closed, open	1201

The data should be ordered by bin_index ascending and not contain nulls. The bin_index should be a contiguous sequence of integers starting from 1.

damagebintobin

```
$ damagebintobin < damage_bin_dict.csv > damage_bin_dict.bin
```

damagebintocsv

```
$ damagebintocsv < damage_bin_dict.bin > damage_bin_dict.csv
```

Return to top

footprint

The event footprint is required for the getmodel component, as well as an index file containing the starting positions of each event block. These must have the following location and filenames;

- static/footprint.bin
- static/footprint.idx

File format

The csv file should contain the following fields and include a header row.

Name	Type	Bytes	Description	Example
event_id	int	4	Oasis event_id	1
areaperil_id	int	4	Oasis areaperil_id	4545
intensity_bin_index	int	4	Identifier of the intensity damage bin	10
prob	float	4	The probability mass for the intensity bin between 0 and 1	0.765

The data should be ordered by event_id, areaperil_id and not contain nulls.

footprinttobin

```
$ footprinttobin -i {number of intensity bins} < footprint.csv
```

This command will create a binary file footprint.bin and an index file footprint.idx in the working directory. The number of intensity bins is the maximum value of intensity_bin_index.

There is an additional parameter -n, which should be used when there is only one record per event_id and areaperil_id, with a single intensity_bin_index value and prob = 1. This is the special case 'no hazard intensity uncertainty'. In this case, the usage is as follows.

```
$ footprinttobin -i {number of intensity bins} -n < footprint.csv
```

Both parameters -i and -n are held in the header of the footprint.bin and used in getmodel.

footprinttocsv

```
$ footprinttocsv > footprint.csv
```

footprinttocsv requires a binary file footprint.bin and an index file footprint.idx to be present in the working directory.

Return to top

occurrence

The occurrence file is required for certain output components which, in the reference model, are leccalc, pltcalc and aalcalc. In general, some form of event occurence file is required for any output which involves the calculation of loss metrics over a period of time. The occurrence file assigns occurrences of the event_ids to numbered periods. A period can represent any length of time, such as a year, or 2 years for instance. The output metrics such as mean, standard deviation or loss exceedance probabilities are with respect to the chosen period length. Most commonly in catastrophe modelling, the period of interest is a year.

The occurrence file also includes date fields. There are two formats for providing the date;

- occ_year, occ_month, occ_day. These are all integers representing occurrence

year, month and day.
- occ_date_id: An integer representing an offset number of days relative to some arbitrary base date.

File format

The csv file should contain the following fields and include a header row.

Option 1. occ_year, occ_month, occ_day

Name	Type	Bytes	Description	Example
event_id	int	4	The occurrence event_id	45567
period_no	int	4	A numbered period in which the event occurs	56876
occ_year	int	4	the year number of the event occurrence	56876
occ_month	int	4	the month of the event occurrence	5
occ_day	int	4	the day of the event occurrence	16

The occurrence year in this example is a scenario numbered year, which cannot be expressed as a real date in a standard calendar.

Option 2. occ_date_id

Name	Type	Bytes	Description	Example
event_id	int	4	The occurrence event_id	1
period_no	int	4	A numbered period in which the event occurs	4545
occ_date_id	int	4	An integer representing an offset number of days to a base date	28626

For example, 28626 days relative to a base date of 0/1/1900 is 16/5/1978.

occurrencetobin

A required parameter is -P, the total number of periods of event occurrences. The total number of periods is held in the header of the binary file and used in output calculations.

Option 2 format should be indicated by using the parameter -D

```
'Option 1 (occurrence files with an occ_year, occ_month, occ_day)
$ occurrencetobin -P10000 < occurrence.csv > occurrence.bin

'Option 2 (occurrence files with an occ_date_id)
$ occurrencetobin -P10000 -D < occurrence.csv > occurrence.bin
```

occurrencetocsv

```
$ occurrencetocsv < occurrence.bin > occurrence.csv
```

Return to top

Random numbers

A random number file may be provided for the gulcalc component as an option (using gulcalc -r parameter) The random number binary contains a list of random numbers used for ground up loss sampling in the kernel calculation. It must have the following location and filename;

- static/random.bin

If the gulcalc -r parameter is not used, the random number binary is not required and random numbers are instead generated dynamically during the calculation, using the -R parameter to specify how many should be generated.

File format

The csv file should contain a simple list of random numbers. It should not contain any headers.

Name	Type	Bytes	Description	Example
rand	float	4	Number between 0 and 1	0.75875

randtobin

```
$ randtobin < random.csv > random.bin
```

randtocsv

```
$ randtocsv < random.bin > random.csv
```

Return to top

vulnerability

The vulnerability file is required for the getmodel component. It contains the conditional distributions of damage for each intensity bin and for each vulnerability_id. This file must have the following location and filename;

- static/vulnerability.bin

File format

The csv file should contain the following fields and include a header row.

Name	Type	Bytes	Description	Example
vulnerability_id	int	4	Oasis vulnerability_id	45
intensity_bin_index	int	4	Identifier of the hazard intensity bin	10
damage_bin_index	int	4	Identifier of the damage bin	20
prob	float	4	The probability mass for the damage bin	0.186

The data should be ordered by vulnerability_id, intensity_bin_index and not contain nulls.

vulnerabilitytobin

```
$ vulnerabilitytobin -d {number of damage bins} < vulnerability.csv > vulnerability.bin
```

The parameter -d number of damage bins is the maximum value of damage_bin_index. This is held in the header of vulnerability.bin and used by getmodel.

vulnerabilitytocsv

```
$ vulnerabilitytocsv < vulnerability.bin > vulnerability.csv
```

Return to top

Input data

Coverages

The coverages binary contains the list of coverages and the coverage TIVs. The data format is as follows. It is required by gulcalc and fmcalc and must have the following location and filename;

- input/coverages.bin

File format

The csv file should contain the following fields and include a header row.

Name	Type	Bytes	Description	Example
coverage_id	int	4	Identifier of the coverage	1
tiv	float	4	The total insured value of the coverage	200000

Coverage_id must be an ordered contiguous sequence of numbers starting at 1.

coveragetobin

```
$ coveragetobin < coverages.csv > coverages.bin
```

coveragetocsv

```
$ coveragetocsv < coverages.bin > coverages.csv
```

Return to top

events

One or more event binaries are required by eve. It must have the following location and filename;

- input/events.bin

File format

The csv file should contain a list of event_ids (integers) and no header.

Name	Type	Bytes	Description	Example
event_id	int	4	Oasis event_id	4545

evetobin

$ evetobin < events.csv > events.bin

evetocsv

$ evetocsv < events.bin > events.csv

Return to top

items

The items binary contains the list of exposure items for which ground up loss will be sampled in the kernel calculations. The data format is as follows. It is required by gulcalc and outputcalc and must have the following location and filename;

- input/items.bin

File format

The csv file should contain the following fields and include a header row.

Name	Type	Bytes	Description	Example
item_id	int	4	Identifier of the exposure item	1
coverage _id	int	4	Identifier of the coverage	3
areaperil_id	int	4	Identifier of the locator and peril of the item	4545
vulnerability_id	int	4	Identifier of the vulnerability distribution of the item	345456
group_id	int	4	Identifier of the correlation group of the	

item | 1 |

Name	Type	Bytes	Description	Example

The data should be ordered by areaperil_id, vulnerability_id ascending and not contain nulls. It is best practice for item_id to be a contiguous sequence of numbers starting from 1.

itemtobin

```
$ itemtobin < items.csv > items.bin
```

itemtocsv

```
$ itemtocsv < items.bin > items.csv
```

Return to top

gul summary xref

The gulsummaryxref binary is a cross reference file which determines how coverage losses from gulcalc output are summed together into at various summary levels in summarycalc. It is required by summarycalc and must have the following location and filename;

- input/gulsummaryxref.bin

File format

The csv file should contain the following fields and include a header row.

Name	Type	Bytes	Description	Example
coverage _id	int	4	Identifier of the coverage	3
summary_id	int	4	Identifier of the summary level group for one or more coverages	1
summaryset_id	int	4	Identifier of the summary set (0 to 9 inclusive)	1

- The data should not contain nulls and there should be at least one summary set in the file.
- Valid entries for summaryset_id is all integers between 0 and 9 inclusive.

- Within each summary set, all coverages_id's from the coverages file must be present.

One summary set consists of a common summaryset_id and each coverage_id being assigned a summary_id. An example is as follows.

coverage_id	summary_id	summaryset_id
1	1	1
2	1	1
3	1	1
4	2	1
5	2	1
6	2	1

This shows, for summaryset_id=1, coverages 1-3 being grouped into summary_id = 1 and coverages 4-6 being grouped into summary_id = 2. This could be an example of a 'site' level grouping, for example. The summary_ids should be held in a dictionary which contains the description of the ids to make meaning of the output results. For instance;

summary_id	summaryset_id	summary_desc
1	1	site_435
2	1	site_958

This cross reference information is not required in ktools.

Up to 10 summary sets may be provided in gulsummaryxref, depending on the required summary reporting levels for the analysis. Here is an example of the 'site' summary level with summaryset_id=1, plus an 'account' summary level with summaryset_id = 2. In summary set 2, the account summary level includes both sites because all coverages are assigned a summary_id of 1.

coverage_id	summary_id	summaryset_id
1	1	1
2	1	1
3	1	1
4	2	1
5	2	1
6	2	1
1	1	2
2	1	2
3	1	2
4	1	2
5	1	2
6	1	2

gulsummaryxreftobin

```
$ gulsummaryxreftobin < gulsummaryxref.csv > gulsummaryxref.bin
```

gulsummaryxreftocsv

```
$ gulsummaryxreftocsv < gulsummaryxref.bin > gulsummaryxref.csv
```

Return to top

fm programme

The fm programme binary file contains the level heirarchy and defines aggregations of losses required to perform a loss calculation, and is required for fmcalc only.

This must have the following location and filename;

- input/fm_programme.bin

File format

The csv file should contain the following fields and include a header row.

Name	Type	Bytes	Description	Example
from_agg_id	int	4	Oasis Financial Module from_agg_id	1
level_id	int	4	Oasis Financial Module level_id	1
to_agg_id	int	4	Oasis Financial Module to_agg_id	1

- All fields must have integer values and no nulls
- Must have at least one level, starting from level_id = 1, 2 3 ...
- For level_id = 1, the set of values in from_agg_id must be equal to the set of item_ids in the input ground up loss stream (which has fields event_id, item_id, idx, gul). Therefore level 1 always defines a group of items.
- For subsequent levels, the from_agg_id must be the distinct values from the previous level to_agg_id field.
- The from_agg_id and to_agg_id values, for each level, should be a contiguous block of integers (a sequence with no gaps). This is not a strict rule in this version and it will work with non-contiguous integers, but it is recommended as best practice.

fmprogrammetobin

```
$ fmprogrammetobin < fm_programme.csv > fm_programme.bin
```

fmprogrammetocsv

```
$ fmprogrammetocsv < fm_programme.bin > fm_programme.csv
```

Return to top

fm profile

The fmprofile binary file contains the list of calculation rules with profile values (policytc_ids) that appear in the policytc file. This is required for fmcalc only.

This must be in the following location with filename format;

- input/fm_profile.bin

File format

The csv file should contain the following fields and include a header row.

Name	Type	Bytes	Description	Example
policytc_id	int	4	Oasis Financial Module policytc_id	34
calcrule_id	int	4	Oasis Financial Module calcrule_id	1
allocrule_id	int	4	Oasis Financial Module allocrule_id	0
ccy_id	int	4	Oasis Financial Module ccy_id	0
deductible	float	4	Deductible	50
limit	float	4	Limit	100000
share_prop_of_lim	float	4	Share/participation as a proportion of limit	0
deductible_prop_of_loss	float	4	Deductible as a proportion of loss	0
limit_prop_of_loss	float	4	Limit as a proportion of loss	0
deductible_prop_of_tiv	float	4	Deductible as a proportion of TIV	0
limit_prop_of_tiv	float	4	Limit as a proportion of TIV	0
deductible_prop_of_limit	float	4	Deductible as a proportion of limit	0

- All distinct policytc_id values that appear in the policytc table must appear once in the policytc_id field of the profile table. We suggest that policytc_id=1 is included by default using calcrule_id = 12 and DED = 0 as a default 'null' calculation rule whenever no terms and conditions apply to a particular level_id / agg_id in the policytc table.
- All data fields that are required by the relevant profile must be provided, with the correct calcrule_id (see FM Profiles)
- Any fields that are not required for the profile should be set to zero.
- allocrule_id may be set to 0 or 1 for each policytc_id. Generally, it is recommended to use 0 everywhere except for the final level calculations when back-allocated losses are required, else 0 everywhere.
- The ccy_id field is currently not used.

fmprofiletobin

```
$ fmprofiletobin < fm_profile.csv > fm_profile.bin
```

fmprofiletocsv

```
$ fmprofiletocsv < fm_profile.bin > fm_profile.csv
```

Return to top

fm policytc

The fm policytc binary file contains the cross reference between the aggregations of losses defined in the fm programme file at a particular level and the calculation rule that should be applied as defined in the fm profile file. This file is required for fmcalc only.

This must have the following location and filename;

- input/fm_policytc.bin

File format

The csv file should contain the following fields and include a header row.

Name	Type	Bytes	Description	Example
layer_id	int	4	Oasis Financial Module layer_id	1
level_id	int	4	Oasis Financial Module level_id	1
agg_id	int	4	Oasis Financial Module agg_id	1
policytc_id	int	4	Oasis Financial Module policytc_id	1

- All fields must have integer values and no nulls
- Must contain the same levels as the fm programme where level_id = 1, 2, 3 ...
- For every distinct combination of to_agg_id and level_id in the programme table, there must be a corresponding record matching level_id and agg_id values in the policytc table with a valid value in the policytc_id field.
- layer_id = 1 at all levels except the last where there may be multiple layers, with layer_id = 1, 2, 3 ... This allows for the specification of several policy contracts applied to the same aggregation of losses defined in the programme table.

fmpolicytctobin

```
$ fmpolicytctobin < fm_policytc.csv > fm_policytc.bin
```

fmpolicytctocsv

```
$ fmpolicytctocsv < fm_policytc.bin > fm_policytc.csv
```

Return to top

fm summary xref

The fm summary xref binary is a cross reference file which determines how losses from fmcalc output are summed together at various summary levels by summarycalc. It is required by summarycalc and must have the following location and filename;

- input/fmsummaryxref.bin

File format

The csv file should contain the following fields and include a header row.

Name	Type	Bytes	Description	Example
output_id	int	4	Identifier of the coverage	3
summary_id	int	4	Identifier of the summary level group for one or more output losses	1

Name	Type	Bytes	Description	Example
summaryset_id	int	4	Identifier of the summary set (0 to 9 inclusive)	1

- The data should not contain nulls and there should be at least one summary set in the file.
- Valid entries for summaryset_id is all integers between 0 and 9 inclusive.
- Within each summary set, all output_id's from the fm xref file must be present.

One summary set consists of a common summaryset_id and each output_id being assigned a summary_id. An example is as follows.

output_id	summary_id	summaryset_id
1	1	1
2	2	1

This shows, for summaryset_id=1, output_id=1 being assigned summary_id = 1 and output_id=2 being assigned summary_id = 2.

If the output_id represents a policy level loss output from fmcalc (the meaning of output_id is defined in the fm xref file) then no further grouping is performed by summarycalc and this is an example of a 'policy' summary level grouping.

Up to 10 summary sets may be provided in this file, depending on the required summary reporting levels for the analysis. Here is an example of the 'policy' summary level with summaryset_id=1, plus an 'account' summary level with summaryset_id = 2. In summary set 2, the 'account' summary level includes both policy's because both output_id's are assigned a summary_id of 1.

output_id	summary_id	summaryset_id
1	1	1
2	2	1
1	1	2
2	1	2

If a more detailed summary level than policy is required for insured losses, then the user should specify in the fm profile file to back-allocate fmcalc losses to items. Then the output_id represents back-allocated policy losses to item, and in the fmsummaryxref file these can be grouped into any summary level, such as site, zipcode, line of business or region, for example. The user needs to define output_id in the fm xref file, and group them together into meaningful summary levels in the fm summary xref file, hence these two files must be consistent with respect to the meaning of output_id.

fmsummaryxreftobin

```
$ fmsummaryxreftobin < fmsummaryxref.csv > fmsummaryxref.bin
```

fmsummaryxreftocsv

```
$ fmsummaryxreftocsv < fmsummaryxref.bin > fmsummaryxref.csv
```

Return to top

fm xref

The fmxref binary file contains cross reference data specifying the output_id in the fmcalc as a combination of agg_id and layer_id, and is required by fmcalc.

This must be in the following location with filename format;

- input/fm_xref.bin

File format

The csv file should contain the following fields and include a header row.

Name	Type	Bytes	Description	Example
output_id	int	4	Identifier of the output group of losses	1
agg_id	int	4	Identifier of the agg_id to output	1
layer_id	int	4	Identifier of the layer_id to output	1

The data should not contain any nulls.

The output_id represents the summary level at which losses are output from fmcalc, as specified by the user.

There are two cases;

- losses are output at the final level of aggregation (represented by the final level to_agg_id's from the fm programme file) for each contract or layer (represented by the final level layer_id's in the fm policytc file)
- losses are back-allocated to the item level and output by item (represented by the from_agg_id of level 1 from the fm programme file) for each policy contract / layer (represented by the final level layer_id's in the fm policytc file)

For example, say there are two policy layers (with layer_ids=1 and 2) which applies to the sum of losses from 4 items (the summary level represented by agg_id=1). Without back-allocation, the policy summary level of losses can be represented as two output_id's as follows;

output_id	agg_id	layer_id
1	1	1
2	1	2

If the user wants to back-allocate policy losses to the items and output the losses by item and policy, then the item-policy summary level of losses would be represented by 8 output_id's, as follows;

output_id	agg_id	layer_id
1	1	1
2	2	1
3	3	1
4	4	1
5	1	2

output_id	agg_id	layer_id
6	2	2
7	3	2
8	4	2

The fm summary xref file must be consistent with respect to the meaning of output_id in the fmxref file.

fmxreftobin

$ fmxreftobin < fm_xref.csv > fm_xref.bin

fmxreftocsv

$ fmxreftocsv < fm_xref.bin > fm_xref.csv

Return to top

return period

The returnperiods binary file is a list of return periods that the user requires to be included in loss exceedance curve (leccalc) results.

This must be in the following location with filename format;

- input/returnperiods.bin

File format

The csv file should contain the following fields with no header.

Name	Type	Bytes	Description	Example
return_period	int	4	Return period	250

returnperiodtobin

$ returnperiodtobin < returnperiods.csv > returnperiods.bin

returnperiodtocsv

$ returnperiodtocsv < returnperiods.bin > returnperiods.csv

Return to top

Go to 4.4 Stream conversion components section

Back to Contents

4.4 Stream conversion components

The following components convert the binary output of each calculation component to csv format;

- **cdftocsv** is a utility to convert binary format CDFs to a csv. getmodel standard output can be streamed directly into cdftocsv, or a binary file of the same format can be piped in.
- **gultocsv** is a utility to convert binary format GULs to a csv. gulcalc standard output can be streamed directly into gultocsv, or a binary file of the same format can be piped in.
- **fmtocsv** is a utility to convert binary format losses to a csv. fmcalc standard output can be streamed directly into fmtocsv, or a binary file of the same format can be piped in.
- **summarycalctocsv** is a utility to convert binary format summarycalc losses to a csv. summarycalc standard output can be streamed directly into summarycalctocsv, or a binary file of the same format can be piped in.
- **aalcalctocsv** is a utility to convert binary format aalcalc losses to a csv. aalcalc standard output can be streamed directly into aalcalctocsv, or a binary file of the same format can be piped in.

Additionally, the following component is provided to convert csv data into binary format;

- **gultobin** is a utility to convert gulcalc data in csv format into binary format such that it can be piped into fmcalc.

Figure 1 shows the workflows for the binary stream to csv conversions.

Figure 1. Stream conversion workflows

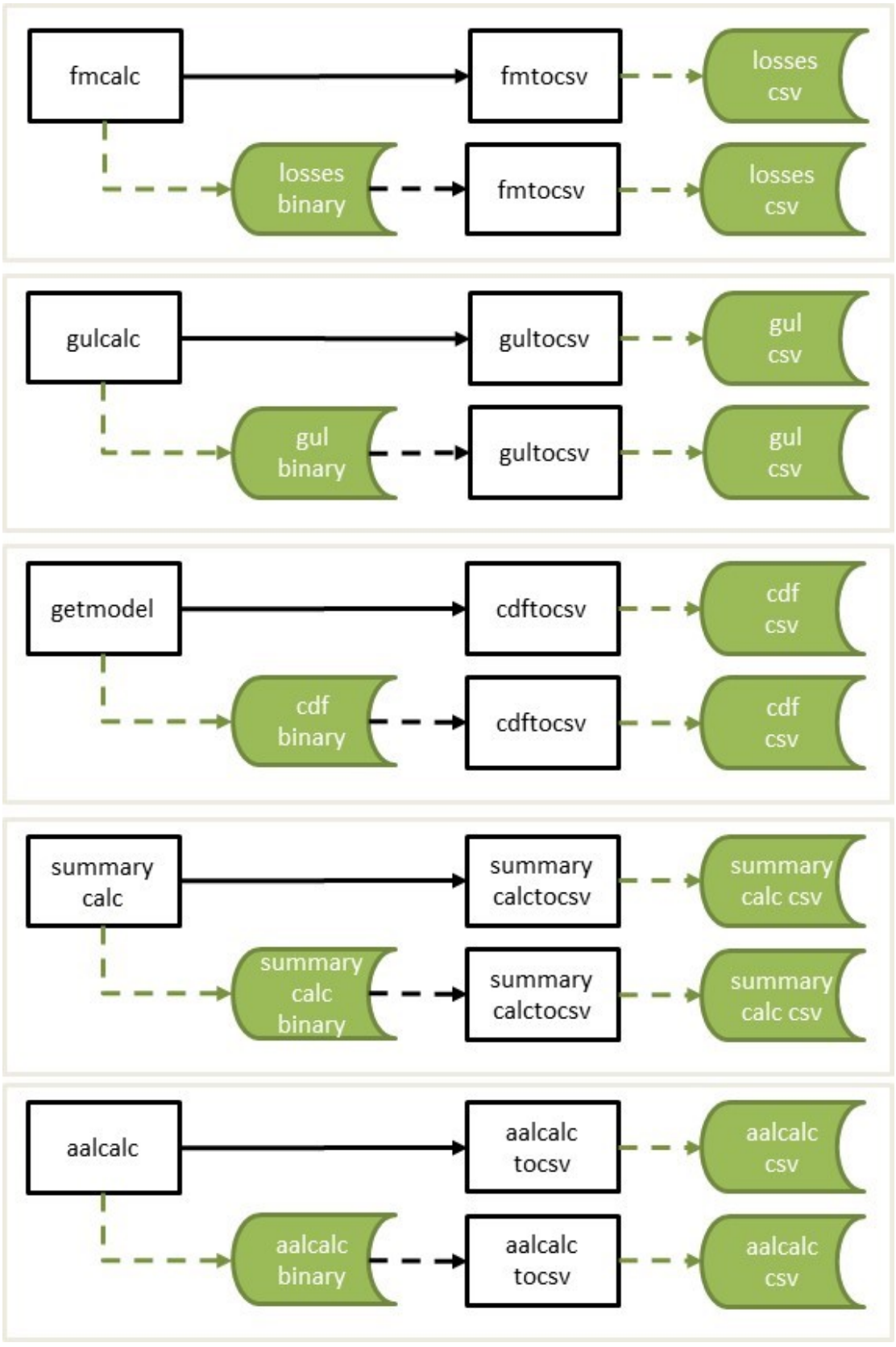
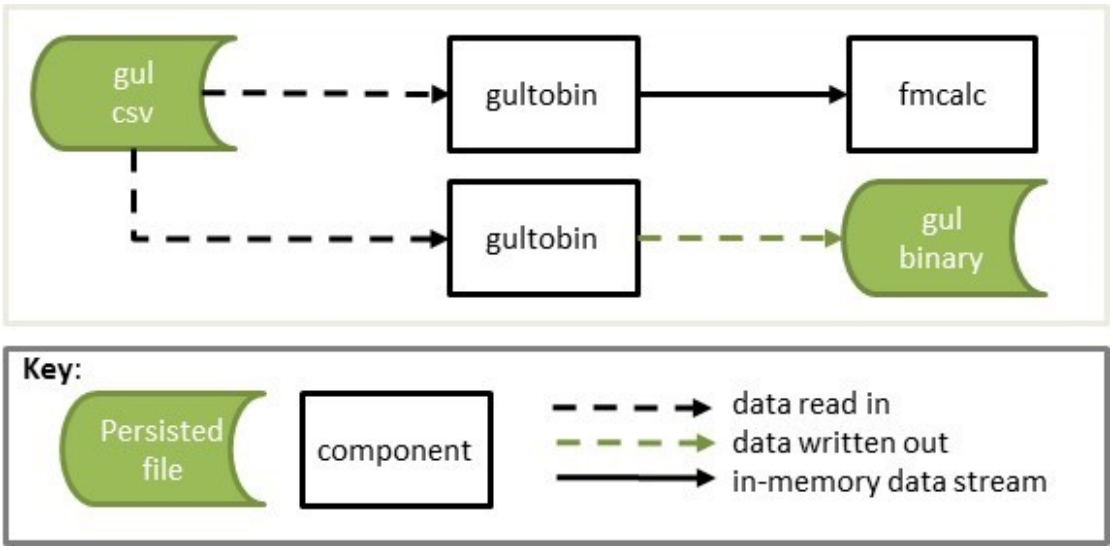


Figure 2 shows the workflows for the gultobin component.

Figure 2. gultobin workflows



cdftocsv

A component which converts the getmodel output stream, or binary file with the same structure, to a csv file.

Stdin stream_id

Byte 1	Bytes 2-4	Description
0	1	getmodel stdout

A binary file of the same format can be piped into cdftocsv.

Usage

```
$ [stdin component] | cdftocsv > [output].csv
$ cdftocsv < [stdin].bin > [output].csv
```

Example

```
$ eve 1 1 | getmodel | cdftocsv > cdf.csv
$ cdftocsv < getmodel.bin > cdf.csv
```

Output

Csv file with the following fields;

Name	Type	Bytes	Description	Example
event_id	int	4	Oasis event_id	4545
areaperil_id	int	4	Oasis areaperil_id	345456
vulnerability_id	int	4	Oasis vulnerability_id	345
bin_index	int	4	Damage bin index	20
prob_to	float	4	The cumulative probability at the upper damage bin threshold	0.765
bin_mean	float	4	The conditional mean of the damage bin	0.45

Return to top

gultocsv

A component which converts the gulcalc output stream, or binary file with the same structure, to a csv file.

Stdin stream_id

Byte 1	Bytes 2-4	Description
1	1	gulcalc item stdout
1	2	gulcalc coverage stdout

A binary file of the same format can be piped into gultocsv.

Usage

```
$ [stdin component] | gultocsv > [output].csv
$ gultocsv < [stdin].bin > [output].csv
```

Example

```
$ eve 1 1 | getmodel | gulcalc -r -S100 -c - | gultocsv > gulcalcc.csv
$ gultocsv < gulcalci.bin > gulcalci.csv
```

Output

Csv file with the following fields;

gulcalc stream_id=1

Name	Type	Bytes	Description	Example
event_id	int	4	Oasis event_id	4545
item_id	int	4	Oasis item_id	300
sidx	int	4	Sample index	10
loss	float	4	The ground up loss value	5675.675

gulcalc stream_id=2

Name	Type	Bytes	Description	Example
event_id	int	4	Oasis event_id	4545
coverage_id	int	4	Oasis coverage_id	150
sidx	int	4	Sample index	10
loss	float	4	The ground up loss value	5675.675

Return to top

fmtocsv

A component which converts the fmcalc output stream, or binary file with the same structure, to a csv file.

Stdin stream_id

Byte 1	Bytes 2-4	Description
2	1	fmcalc stdout

A binary file of the same format can be piped into fmtocsv.

Usage

```
$ [stdin component] | fmtocsv > [output].csv
$ fmtocsv < [stdin].bin > [output].csv
```

Example

```
$ eve 1 1 | getmodel | gulcalc -r -S100 -i - | fmcalc | fmtocsv > fmcalc.csv
$ fmtocsv < fmcalc.bin > fmcalc.csv
```

Output

Csv file with the following fields;

Name	Type	Bytes	Description	Example
event_id	int	4	Oasis event_id	4545
output_id	int	4	Oasis output_id	5
sidx	int	4	Sample index	10
loss	float	4	The insured loss value	5375.675

Return to top

summarycalctocsv

A component which converts the summarycalc output stream, or binary file with the same structure, to a csv file.

Stdin stream_id

Byte 1	Bytes 2-4	Description
3	1	summarycalc stdout

A binary file of the same format can be piped into summarycalctocsv.

Usage

```
$ [stdin component] | summarycalctocsv > [output].csv
$ summarycalctocsv < [stdin].bin > [output].csv
```

Example

```
$ eve 1 1 | getmodel | gulcalc -r -S100 -i - | fmcalc | summarycalc -f -1 - | summarycalctocsv > summarycalc.csv
$ summarycalctocsv < summarycalc.bin > summarycalc.csv
```

Output

Csv file with the following fields;

Name	Type	Bytes	Description	Example
event_id	int	4	Oasis event_id	4545
summary_id	int	4	Oasis summary_id	3
sidx	int	4	Sample index	10

Name	Type	Bytes	Description	Example
loss	float	4	The insured loss value	5375.675

Return to top

aalcalctocsv

A component which converts the aalcalc output stream, or binary file with the same structure, to a csv file.

Stdin stream_id

Byte 1	Bytes 2-4	Description
4	1	aalcalc stdout

A binary file of the same format can be piped into aalcalctocsv.

Usage

```
$ [stdin component] | aalcalctocsv > [output].csv
$ aalcalctocsv < [stdin].bin > [output].csv
```

Example

```
$ eve 1 1 | getmodel | gulcalc -r -S100 -i - | fmcalc | summarycalc -f -1 - | aalcalc | aalcalctocsv > aalcalc.csv
$ aalcalctocsv < aalcalc.bin > aalcalc.csv
```

Output

Csv file with the following fields;

Name	Type	Bytes	Description	Example
summary_id	int	4	summary_id representing a grouping of losses	10
type	int	4	1 for analytical mean, 2 for mean calculated from samples	1
mean	float	8	sum of period mean losses	67856.9
mean_squared	float	8	sum of squared period mean losses	546577.8
max_exposure_value	float	8	maximum exposure value across all periods	10098730

Return to top

gultobin

A component which converts gulcalc data in csv format into gulcalc binary standard output, for stream_id=1 (item stream).

Input file format

Name	Type	Bytes	Description	Example
event_id	int	4	Oasis event_id	4545
item_id	int	4	Oasis item_id	300
sidx	int	4	Sample index	10

Name	Type	Bytes	Description	Example
loss	float	4	The ground up loss value	5675.675

- For each event_id and item_id combination, a record for sidx = -1 (mean) and sidx = -2 (standard deviation) must be included, even if the loss values are zero.
- For each event_id and item_id combination, one or more sampled losses for sidx > 0 may be provided, but records for samples with zero loss may be omitted.

Parameters

-S, the number of samples must be provided. This can be equal to or greater than maximum sample index value that appears in the csv data.

Usage

```
$ gultobin [parameters] < [input].csv | [stdin component]
$ gultobin [parameters] < [input].csv > [output].bin
```

Example

```
$ gultobin -S100 < gulcalci.csv | fmcalc > fmcalc.bin
$ gultobin -S100 < gulcalci.csv > gulcalci.bin
```

Stdout stream_id

Byte 1	Bytes 2-4	Description
1	1	gulcalc item stdout

[Return to top]

[Go to 5. Financial Module]

[Back to Contents]

5. Financial Module

The Oasis Financial Module is a data-driven process design for calculating the losses on (re)insurance contracts. It has an abstract design in order to cater for the many variations in contract structures and terms. The way Oasis works is to be fed data in order to execute calculations, so for the insurance calculations it needs to know the structure, parameters and calculation rules to be used. This data must be provided in the files used by the Oasis Financial Module:

- **fm_programme**: defines how coverages are grouped into accounts and programmes
- **fm_profile**: defines the layers and terms
- **fm_policytc**: defines the relationship of the contract layers
- **fm_xref**: specifies the summary level of the output losses

This section explains the design of the Financial Module which has been implemented in the **fmcalc** component.

- Runtime parameters and usage instructions for fmcalc are covered in 4.1 Core Components.
- The formats of the input files are covered in 4.3 Data Conversion Components.

In addition, there separate github repository ktest which is an extended test suite for ktools and contains a library of financial module worked examples provided by Oasis Members with a full set of input and output files (access on request).

Note that other reference tables are referred to below that do not appear explicitly in the kernel as they are not directly required for calculation. It is expected that a front end system will hold all of the exposure and policy data and generate the above three input files required for the kernel calculation.

Scope

The Financial Module outputs sample by sample losses by (re)insurance contract, or by item, which represents the individual coverage subject to economic loss. In the latter case, it is necessary to 'back-allocate' losses when they are calculated at a higher policy level. The Financial Module does not, at present, output retained loss or ultimate net loss (UNL) perspectives. It does, though, allow the user to output losses at any stage of the calculation.

The output contains anonymous keys representing the (re)insurance programme (prog_id) and policy (layer_id) at the chosen output level (output_id) and a loss value. Losses by sample number (idx) and event (event_id) are produced. To make sense of the output, this output must be cross-referenced with Oasis dictionaries which contain the meaningful business information.

The Financial Module does not support multi-currency calculations, although currency identifiers are included in the definition of profiles (see below) as placeholders for this functionality in future.

Profiles

Profiles are used throughout the Oasis framework and are meta-data definitions with their associated data types and rules. Profiles are used in the Financial Module to perform the elements of financial calculations used to calculate losses to (re)insurance policies. For anything other than the most simple policy which has a blanket deductible and limit, say, a profile do not represent a policy structure on its own, but rather is to be used as a building block which can be combined with other building blocks to model a particular financial contract. In this way it is possible to model an unlimited range of structures with a limited number of profiles.

The FM Profiles form an extensible library of calculations defined within the fmcalc code that can be invoked by specifying a particular **calcrule_id** and providing the required data values such as deductible and limit, as described below.

The Profiles currently supported are as follows;

Supported Profiles

Profile description	profile_id	calcrule_id
Deductible and limit	1	1
Franchise deductible and limit	2	3
Deductible only	3	12
Deductible as a cap on the retention of input losses	4	10
Deductible as a floor on the retention of input losses	5	11
Deductible, limit and share	6	2
Deductible and limit as a proportion of loss	10	5
Limit with deductible as a proportion of limit	11	9
Limit only	12	14
Limit as a proportion of loss	13	15
Deductible as a proportion of loss	14	16

See Appendix B FM Profiles for more details.

Design

The Oasis Financial Module is a data-driven process design for calculating the losses on insurance policies. It is an abstract design in order to cater for the many variations and has four basic concepts:

1. A **programme** which defines which **items** are grouped together at which levels for the purpose of providing loss amounts to policy terms and conditions. The programme has a user-definable profile and dictionary called **prog** which holds optional reference details such as a description and account identifier. The prog table is not required for the calculation and therefore does not appear in the kernel input files.
2. A policytc **profile** which provides the parameters of the policy's terms and conditions such as limit and deductible and calculation rules.
3. A **policytc** cross-reference file which associates a policy terms and conditions profile to each programme level aggregation.
4. A **xref** file which specifies how the output losses are summarized.

The profile not only provides the fields to be used in calculating losses (such as limit and deductible) but also which mathematical calculation (calcrule_id) and which allocation rule (allocrule_id) to apply.

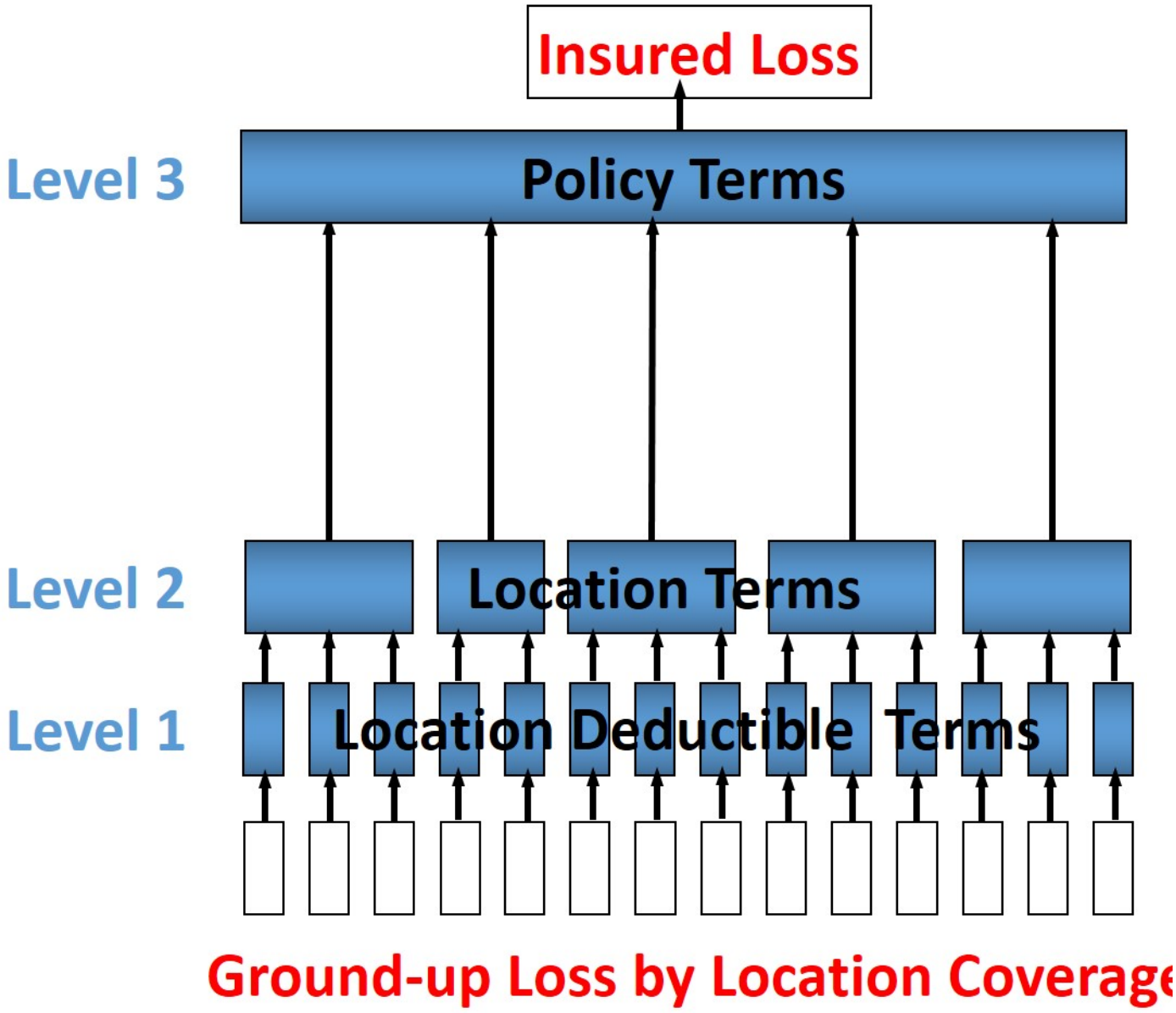
Data requirements

The Financial Module brings together three elements in order to undertake a calculation:

- Structural information, notably which items are covered by a set of policies.
- Loss values of items.
- Policy profiles and profile values.

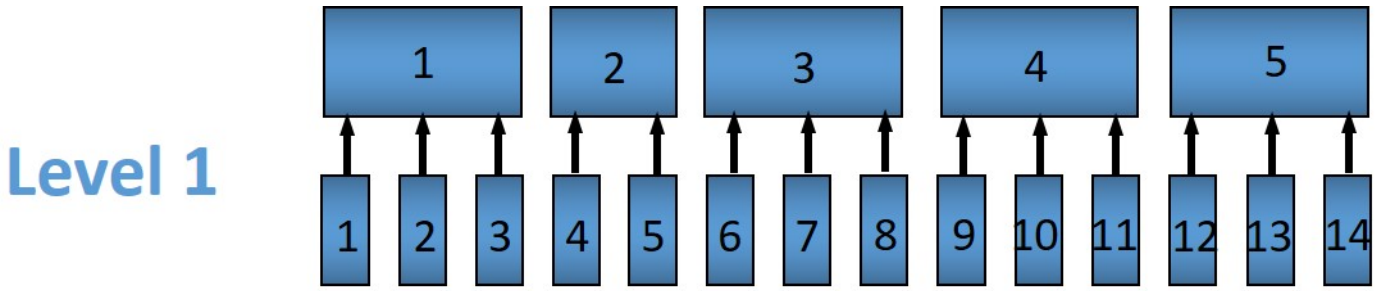
There are many ways an insurance loss can be calculated with many different terms and conditions. For instance, there may be deductibles applied to each element of coverage (e.g. a buildings damage deductible), some site-specific deductibles or limits, and some overall policy deductibles and limits and share. To undertake the calculation in the correct order and using the correct items (and their values) the structure and sequence of calculations must be defined. This is done in the **programme** file which defines a heirarchy of groups across a number of **levels**. Levels drive the sequence of calculation. A financial calculation is performed at successive levels, depending on the structure of policy terms and conditions. For example there might be 3 levels representing coverage, site and policy terms and conditions.

Figure 1. Example 3-level programme hierarchy



Groups are defined within levels and they represent aggregations of losses on which to perform the financial calculations. The grouping fields are called from_agg_id and to_agg_id which represent a grouping of losses at the previous level and the present level of the hierarchy, respectively.

Figure 2. Example level 1 grouping



Loss values

The initial input is the ground-up loss (GUL) table, generally coming from the main Oasis calculation of ground-up losses. Here is an example, for a two events and 1 sample (idx=1):

event_id	item_id	sidx	loss
1	1	1	100,000
1	2	1	10,000
1	3	1	2,500
1	4	1	400
2	1	1	90,000
2	2	1	15,000
2	3	1	3,000
2	4	1	500

The values represent a single ground-up loss sample for items belonging to an account. We use “programme” rather than "account" as it is more general characteristic of a client’s exposure protection needs and allows a client to have multiple programmes active for a given period.
The linkage between account and programme can be provided by a user defined **prog** dictionary, for example;

prog_id	account_id	prog_name
1	1	ABC Insurance Co. 2016 renewal

Items 1-4 represent Structure, Other Structure, Contents and Time Element coverage ground up losses for a single property, respectively, and this example is a simple residential policy with combined property coverage terms. For this policy type, the Structure, Other Structure and Contents losses are aggregated, and a deductible and limit is applied to the total. A separate set of terms, again a simple deductible and limit, is applied to the “time element” coverage which, for residential policies, generally means costs for temporary accommodation. The total insured loss is the sum of the output from the combined property terms and the time element terms.

Programme

The actual items falling into the programme are specified in the **programme** table together with the aggregation groupings that go into a given level calculation:

from_agg_id	level_id	to_agg_id
1	1	1
2	1	1
3	1	1
4	1	2
1	2	1
2	2	1

Note that from_agg_id for level_id=1 is equal to the item_id in the input loss table (but in theory from_agg_id could represent a higher level of grouping, if required).

In level 1, items 1, 2 and 3 all have to_agg_id =1 so losses will be summed together before applying the combined deductible and limit, but item 4 (time) will be treated separately (not aggregated) as it has to_agg_id = 2. For level 2 we have all 4 items losses (now represented by two groups from_agg_id =1 and 2 from the previous level) aggregated together as they have the same to_agg_id = 1.

Profile

Next we have the profile description table, which list the profiles representing general policy types. Our example is represented by two general profiles which specify the input fields and mathematical operations to perform. In this example, the profile for the combined coverages and time is the same (albeit with different values) and requires a limit, a deductible, and an associated calculation rule, whereas the profile for the policy requires a limit, deductible, and share, and an associated calculation rule.

Profile description	profile_id	calcrule_id
Deductible and limit	1	1
Deductible, limit and share	6	2

There is a “profile value” table for each profile containing the applicable policy terms, each identified by a policytc_id. The table below shows the list of policy terms for profile_id 1.

policytc_id	ccy_id	limit	deductible
1	1	1,000,000	1,000
2	1	18,000	2,000

And next, for profile 6, the values for the overall policy deductible, limit and share

policytc_id	ccy_id	limit	deductible	share_prop_of_lim
3	1	1,000,000	1,000	0.1

In practice, all profile values are stored in a single flattened format which contains all supported profile fields, but conceptually they belong in separate profile value tables (see fm profile in 4.3 Data Conversion Components).

For any given profile we have two standard rules:

- **calcrule_id**, being the Function used to calculate the losses from the given Profile's fields. More information about the functions can be found in FM Profiles.
- **allocrule_id**, being the rule for allocating back to ITEM level. There are really only two meaningful values here – don't allocate (0) used typically for the final level to avoid maintaining lots of detailed losses, or allocate back to ITEMs (1) used in all other cases which is in proportion to the input ground up losses.
  (Allocation does not need to be on this basis, by the way, there could be other rules such as allocate back always on TIV or in proportion to the input losses from the previous level, but we have implemented a ground up loss back-allocation rule.

Policytc

The **policytc** table specifies the insurance policies (a policy in Oasis FM is a combination of prog_id and layer_id) and the separate terms and conditions which will be applied to each layer_id/agg_id for a given level. In our example, we have a limit and deductible with the same value applicable to the combination of the first three items, a limit and deductible for the fourth item (time) in level 1, and then a limit, deductible, and line applicable at level 2 covering all items. We'd represent this in terms of the distinct agg_ids as follows:

layer_id	level_id	agg_id	policytc_id
1	1	1	1
1	1	2	2
1	2	1	3

In words, the data in the table mean;

At Level 1;

Apply policytc_id (calculation rule) 1 to (the sum of losses represented by) agg_id 1

Apply policy 2 to agg_id 2

Then at level 2;

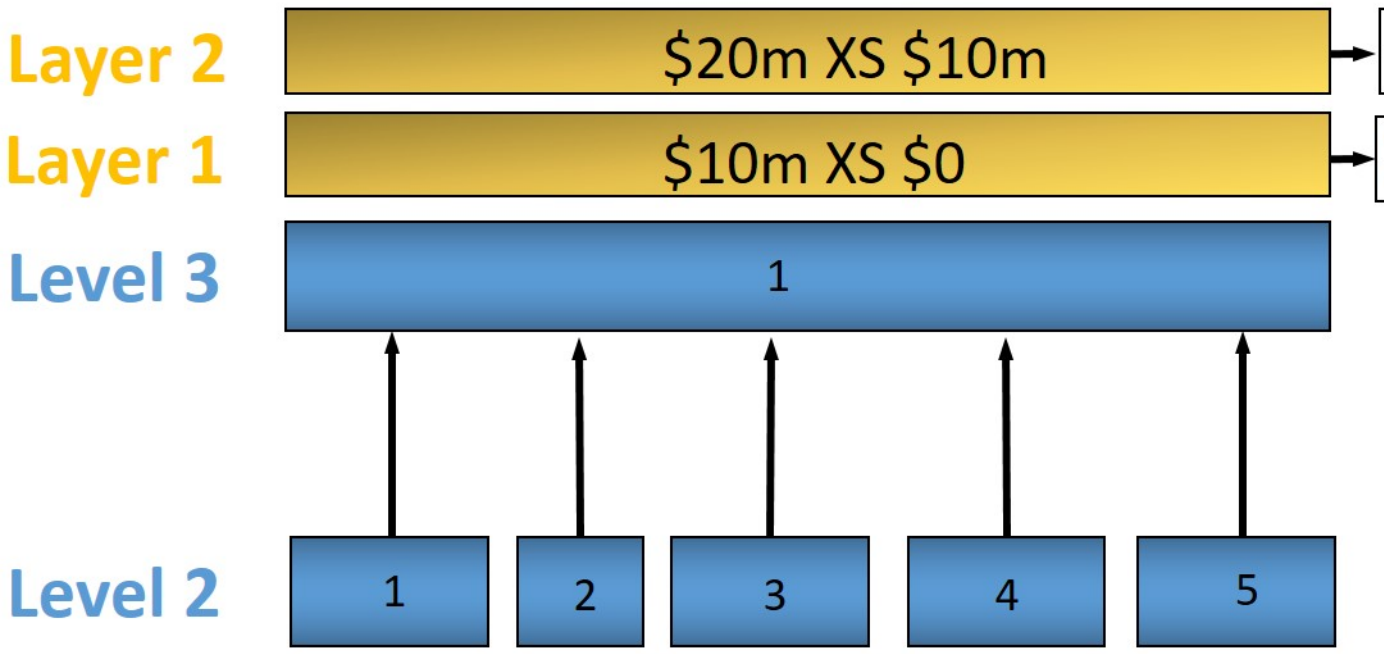
Apply policytc_id 3 to agg_id 1

Levels are processed in ascending order and the calculated losses from a previous level are summed according to the groupings defined in the programme table which become the input losses to the next level.

Layers

Layers can be used to model multiple sets of terms and conditions applied to the same losses, such as excess policies. For the lower level calculations and in the general case where there is a single contract, layer_id should be set to 1. For a given level_id and agg_id, multiple layers can be defined by setting layer_id =1,2,3 etc, and assigning a different calculation policytc_id to each.

Figure 3. Example of multiple layers



For this example at level 3, the policytc data might look as follows;

layer_id	level_id	agg_id	policytc_id
1	3	1	22
2	3	1	23

Output and back-allocation

Losses are output by event, output level and sample. The table looks like this;

event_id	output_id	sidx	loss
1	1	1	455.24
2	1	1	345.6

The output_id is specified by the user in the **xref** table, and is a unique combination of agg_id and layer_id. For instance;

output_id	agg_id	layer_id
1	1	1
2	1	2

There are two options, depending on whether the losses are allocated back to the items at the final level or not.

- If allocrule_id = 0 for all policytc_ids at the final level then agg_id = to_agg_id of the final level in the**programme** table
- If allocrule_id = 1 for all policytc_ids at the final level then output_id = from_agg_id of the first level in the**programme** table

In other words, losses are either output at the contract level or back-allocated to the lowest level, which is item_id. To avoid unnecessary computation, it is recommended not to back-allocate unless losses are required to be reported at a more detailed level than the contract level (site or zip code, for example). In this case, losses are re-aggregated up from item level (represented by output_id in fmcalc output) in summarycalc, using the fmsummaryxref table.

In R1.1 of Oasis we took the view that it was simpler throughout to refer back to the base items rather than use a hierarchy of aggregated loss calculations. So, for example, we could have calculated a loss at the site level and then used this calculated loss directly at the policy conditions level but instead we allocated back to item level and then re-aggregated to the next level. The reason for this was that intermediate conditions may only apply to certain items so if we didn't go back to the base item "ground-up" level then any higher level could have a complicated grouping of a level.

In the implementation this required back-allocating losses to item at every level in a multi-level calculation even the next level calculaton did not require it, which was inefficient. The aggregations are now defined in terms of the previous level groupings (from_agg_id in the programme table, rather than item_id) and the execution path now only supports simple hierarchies.

Return to top

Go to 6. Workflows

Back to Contents

6. Workflows

ktools is capable of multiple output workflows. This brings much greater flexibility, but also more complexity for users of the toolkit.

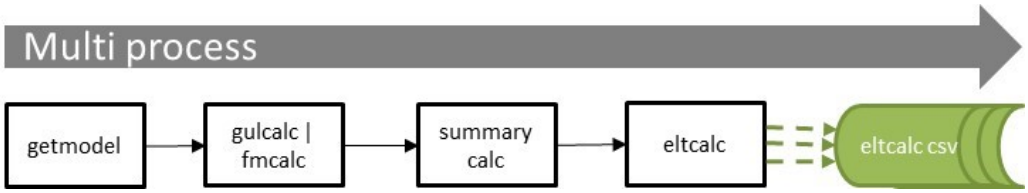
This section presents some example workflows, starting with single output workflows and then moving onto more complex multi-output workflows. There are some python scripts provided which execute some of the illustrative workflows using the example data in the repository. It is assumed that workflows will generally be run across multiple processes, with the number of processes being specified by the user.

1. Portfolio summary level insured loss event loss table

In this example, the core workflow is run through fmcalc into summarycalc and then the losses are summarized by summary set 2, which is "portfolio" summary level.
This produces multiple output files when run with multiple processes, each containing a subset of the event set. The output files can be concatinated together at the end.

```
eve 1 2 | getmodel | gulcalc -r -S100 -i - | fmcalc | summarycalc -f -2 - | eltcalc > elt_p1.csv
eve 2 2 | getmodel | gulcalc -r -S100 -i - | fmcalc | summarycalc -f -2 - | eltcalc > elt_p2.csv
```

Figure 1. eltcalc workflow



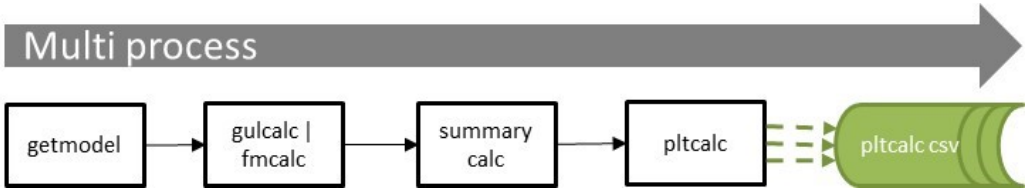
See example script eltcalc_example.py
***

2. Portfolio summary level insured loss period loss table

This is very similar to the first example, except the summary samples are run through pltcalc instead. The output files can be concatinated together at the end.

```
eve 1 2 | getmodel | gulcalc -r -S100 -i - | fmcalc | summarycalc -f -2 - | pltcalc > plt_p1.csv
eve 2 2 | getmodel | gulcalc -r -S100 -i - | fmcalc | summarycalc -f -2 - | pltcalc > plt_p2.csv
```

Figure 2. pltcalc workflow



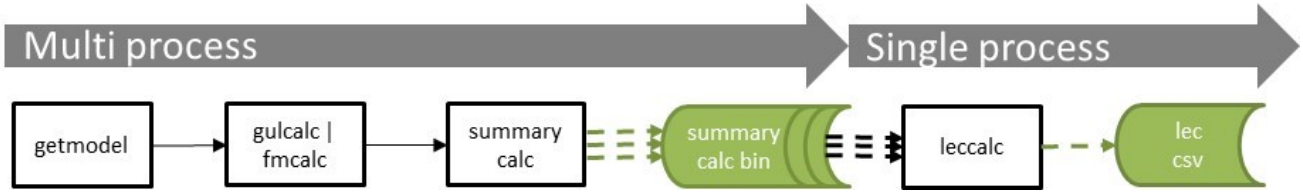
See example script pltcalc_example.py
***

3. Portfolio summary level full uncertainty aggregate and occurrence loss exceedance curves

In this example, the summary samples are calculated as in the first two examples, but the results are output to the work folder. Until this stage the calculation is run over multiple processes. Then, in a single process, leccalc reads the summarycalc binaries from the work folder and computes two loss exceedance curves in a single process. Note that you can output all eight loss exceedance curve variants in a single leccalc command.

```
eve 1 2 | getmodel | gulcalc -r -S100 -i - | fmcalc | summarycalc -f -2 - > work/summary2/p1.bin
eve 2 2 | getmodel | gulcalc -r -S100 -i - | fmcalc | summarycalc -f -2 - > work/summary2/p1.bin
leccalc -Ksummary2 -F lec_full_uncertainty_agg.csv -f lec_full_uncertainty_occ.csv
```

Figure 3. leccalc workflow



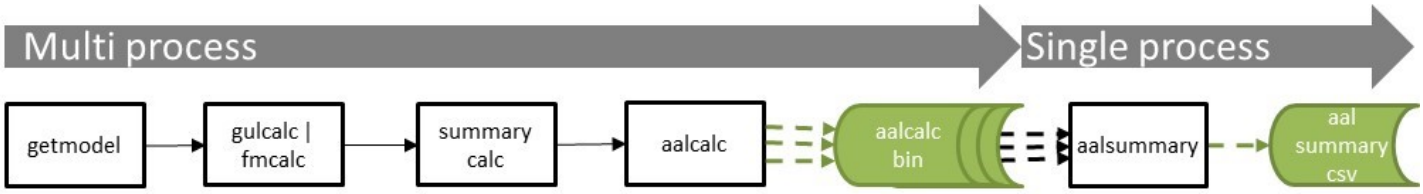
See example script leccalc_example.py
***

4. Portfolio summary level average annual loss

This time, the samples are run through to aalcalc, and the aalcalc binaries are output to the work folder. Until this stage the calculation is run over multiple processes. Then, in a single process, aalsummary reads the aalcalc binaries from the work folder and computes the aal output.

```
eve 1 2 | getmodel | gulcalc -r -S100 -i - | fmcalc | summarycalc -f -2 - | aalcalc > work/summary2/p1.bin
eve 2 2 | getmodel | gulcalc -r -S100 -i - | fmcalc | summarycalc -f -2 - | aalcalc > work/summary2/p2.bin
aalsummary -Ksummary2 > aal.csv
```

Figure 4. aalcalc workflow



See example script aalcalc_example.py
***

Multiple output workflows

5. Ground up and insured loss workflows

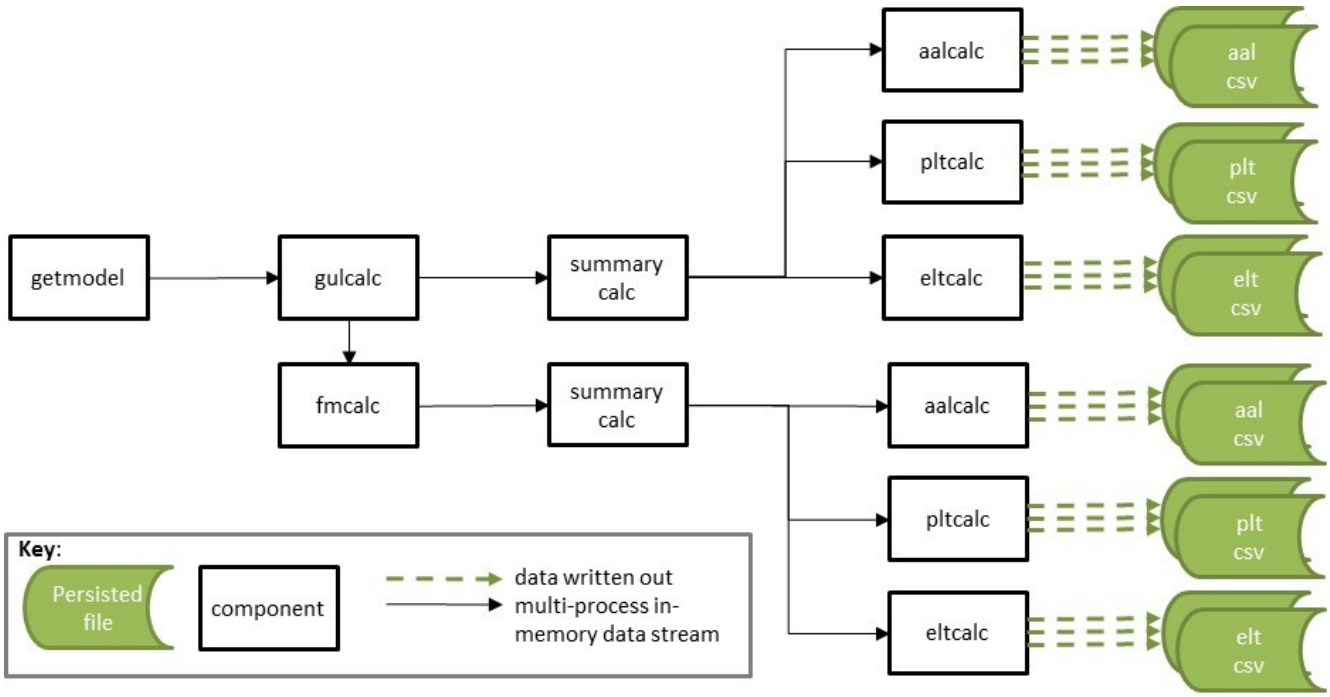
gulcalc can generate two output streams at once: item level samples to pipe into fmcalc, and coverage level samples to pipe into summarycalc. This means that outputs for both ground up loss and insured loss can be generated in one workflow.
This is done by writing one stream to a file or named pipe, while streaming the other to standard output down the pipeline.

```
eve 1 2 | getmodel | gulcalc -r -S100 -i gulcalci1.bin -c - | summarycalc -g -2 - | eltcalc > gul_elt_p1.csv
eve 2 2 | getmodel | gulcalc -r -S100 -i gulcalci2.bin -c - | summarycalc -g -2 - | eltcalc > gul_elt_p2.csv
fmcalc < gulcalci1.bin | summarycalc -f -2 - | eltcalc > fm_elt_p1.csv
fmcalc < gulcalci2.bin | summarycalc -f -2 - | eltcalc > fm_elt_p2.csv
```

Note that the gulcalc item stream does not need to be written off to disk, as it can be sent to a 'named pipe', which keeps the data in-memory and kicks off a new process. This is easy to do in Linux (but harder in Windows).

Figure 5 illustrates an example workflow.

Figure 5. Ground up and insured loss example workflow



See example script gulandfm_example.py
***

6. Multiple summary level workflows

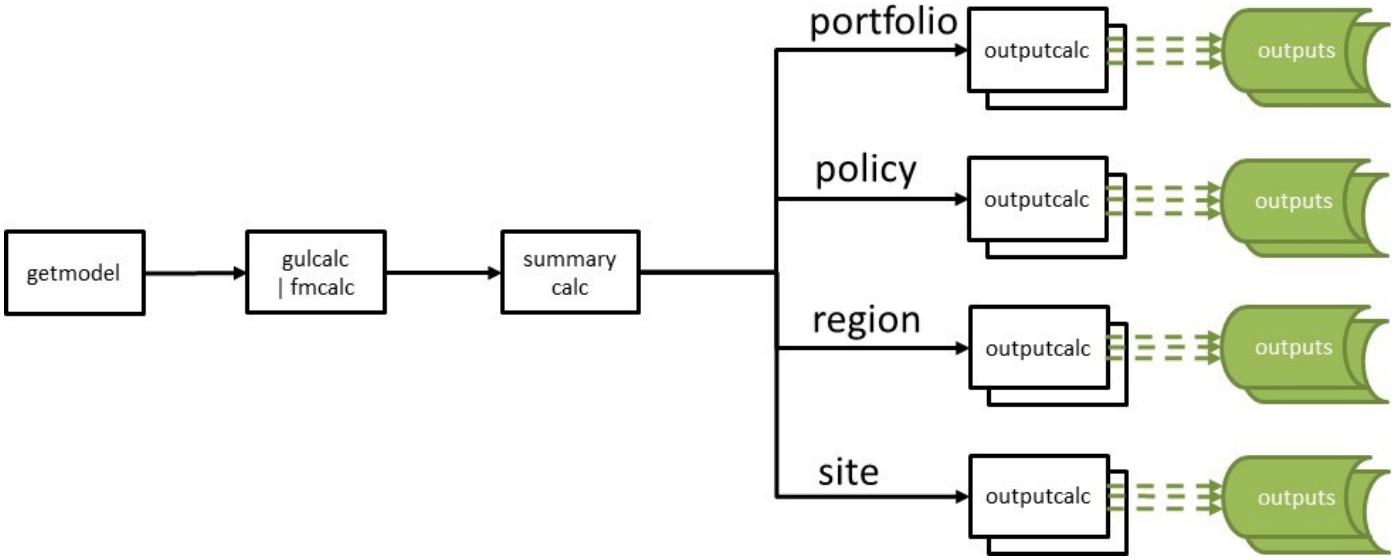
Summarycalc is capable of summarizing samples to up to 10 different user-defined levels for ground up loss and insured loss. This means that different outputs can be run on different summary levels. In this example, event loss tables for two different summary levels are generated.

```
eve 1 2 | getmodel | gulcalc -r -S100 -i - | fmcalc | summarycalc -f -1 s1/p1.bin -2 s2/p1.bin
eve 2 2 | getmodel | gulcalc -r -S100 -i - | fmcalc | summarycalc -f -1 s1/p2.bin -2 s2/p2.bin
eltcalc < s1/p1.bin > elt_s1_p1.csv
eltcalc < s1/p2.bin > elt_s1_p2.csv
eltcalc < s2/p1.bin > elt_s2_p1.csv
eltcalc < s2/p2.bin > elt_s2_p2.csv
```

Again, the summarycalc streams can be sent to named pipes rather than written off to disk.

Figure 6 illustrates multiple summary level streams, each of which can go to different output calculations.

Figure 6. Multiple summary level workflows



[Return to top](#)

[Go to Appendix A Random numbers](#)

[Back to Contents](#)

Appendix A: Random numbers

Simple uniform random numbers are assigned to each event, group and sample number to sample ground up loss in the gulcalc process. A group is a collection of items which share the same group_id, and is the method of supporting spatial correlation in ground up loss sampling in Oasis and ktools.

Correlation

Items (typically representing, in insurance terms, the underlying risk coverages) that are assigned the same group_id will use the same random number to sample damage for a given event and sample number. Items with different group_ids will be assigned independent random numbers. Therefore sampled damage is fully correlated within groups and fully independent between groups, where group is an abstract collection of items defined by the user.

The item_id, group_id data is provided by the user in the items input file (items.bin).

Methodology

The method of assigning random numbers in gulcalc uses an random number index (ridx) which is used as a position reference into a list of random numbers. S random numbers corresponding to the runtime number of samples are drawn from the list starting at the first ridx position.

There are two options in ktools for choosing random numbers to apply in the sampling process.

1. Generate dynamically during the calculation

Usage

Use -R{number of random numbers} as a parameter.

Example

```
$ gulcalc -S00 -R1000000 -i -
```

This will run 100 samples drawing from 1 million dynamically generated random numbers.

Method

Random numbers are sampled dynamically using the Mersenne twister psuedo random number generator (the default RNG of the C++ v11 compiler).
A sparse array capable of holding R million random numbers is allocated to each event. The ridx is generated from the group_id and number of samples S using the following modulus function;

ridx= mod(group_id x P1, R)

- P1 is the first prime number which is bigger than the number of samples, S.

This formula pseudo-randomly assigns ridx indexes to each group_id between 0 and 999,999.

As a ridx is sampled, the section in the array starting at the ridx position of length S is populated with random numbers unless they have already been populated, in which case the existing random numbers are re-used.

The array is cleared for the next event and a new set of random numbers is generated.

2. Use numbers from random number file

Usage

Use -r as a parameter

Example

```
$ gulcalc -S100 -r -i -
```

This will run 100 samples using random numbers from file random.bin in the static sub-directory.

Method

The random number file(s) is read into memory at the start of the gulcalc process.

The ridx is generated from the sample index (sidx), event_id and group_id using the following modulus function;

ridx= sidx + mod(group_id x P1 x P3 + event_id x P2, R)

- R is the divisor of the modulus, equal to the total number of random numbers in the list.
- P1 and P2 are the first two prime numbers which are greater than half of R.
- P3 is the first prime number which is bigger than the number of samples, S.

This formula pseudo-randomly assigns a starting position index to each event_id and group_id combo between 0 and R-1, and then S random numbers are drawn by incrementing the starting position by the sidx.

Return to top

Go to Appendix B FM Profiles

Back to Contents

Appendix B: FM Profiles

This section specifies the attributes and rules for the following list of profiles.

Profile description	profile_id	calcrule_id
Deductible and limit	1	1
Franchise deductible and limit	2	3
Deductible only	3	12
Deductible as a cap on the retention of input losses	4	10
Deductible as a floor on the retention of input losses	5	11
Deductible, limit and share	6	2
Deductible and limit as a proportion of loss	10	5
Limit with deductible as a proportion of limit	11	9
Limit only	12	14
Limit as a proportion of loss	13	15
Deductible as a proportion of loss	14	16

In the following notation;

- x.loss is the input loss to the calculation
- x.retained_loss is the input retained loss to the calculation (where required)
- loss is the calculated loss
- ded, lim, and share are alias variables for the profile fields as required

1. Deductible and limit

Attributes	Example
policytc_id	1
ccy_id	1
deductible	50000
limit	900000

Rules	Value
calcrule_id	1

Calculation logic

```
loss = x.loss - ded;
if (loss < 0) loss = 0;
if (loss > lim) loss = lim;
```

2. Franchise deductible and limit

Attributes	Example
policytc_id	1
ccy_id	1
deductible	100000
limit	1000000

Rules	Value
calcrule_id	3

Calculation logic

```
if (x.loss < ded) loss = 0;
    else loss = x.loss;
if (loss > lim) loss = lim;
```

3. Deductible only

Attributes	Example
policytc_id	1
ccy_id	1
deductible	100000

Rules	Value
calcrule_id	12

Calculation logic

```
loss = x.loss - ded;
if (loss < 0) loss = 0;
```

4. Deductible as a cap on the retention of input losses

Attributes	Example
policytc_id	1
ccy_id	1
deductible	40000

Rules	Value
calcrule_id	10

Calculation logic

```
if (x.retained_loss > ded) {
    loss = x.loss + x.retained_loss - ded;
    if (loss < 0) loss = 0;
    }
else {
    loss = x.loss;
     }
```

5. Deductible as a floor on the retention of input losses

Attributes	Example
policytc_id	1
ccy_id	1
deductible	70000

Rules	Value

Rules	Value
calcrule_id	11

Calculation logic

```
if (x.retained_loss < ded) {
   loss = x.loss + x.retained_loss - ded;
   if (loss < 0) loss = 0;
   }
else {
   loss = x.loss;
    }
```

6. Deductible, limit and share

Attributes	Example
policytc_id	1
ccy_id	1
deductible	70000
limit	1000000
share_prop_of_lim	0.1

Rules	Value
calcrule_id	2

Calculation logic

```
if (x.loss > (ded + lim)) loss = lim;
   else loss = x.loss - ded;
if (loss < 0) loss = 0;
loss = loss * share;
```

10. Deductible and limit as a proportion of loss

Attributes	Example
policytc_id	1
deductible_prop_of_loss	0.05
limit_prop_of_loss	0.3

Rules	Value
calcrule_id	5

Calculation logic

```
loss = x.loss * (lim - ded);
```

11. Limit with deductible as a proportion of limit

Attributes	Example
policytc_id	1
ccy_id	1
deductible_prop_of_loss	0.05
limit_prop_of_loss	100000

Rules	Value
calcrule_id	9

Calculation logic

```
loss = x.loss - (ded * lim);
if (loss < 0) loss = 0;
if (loss > lim) loss = lim;
```

12. Limit only

Attributes	Example
policytc_id	1
ccy_id	1
limit	100000

Rules	Value
calcrule_id	14

Calculation logic

```
loss = x.loss;
if (loss > lim) loss = lim;
```

13. Limit as a proportion of loss

Attributes	Example
policytc_id	1
limit_prop_of_loss	0.3

Rules	Value
calcrule_id	15

Calculation logic

```
loss = x.loss;
loss = loss * lim;
```

14. Deductible as a proportion of loss

Attributes	Example
policytc_id	1
deductible_prop_of_loss	0.05

Rules	Value
calcrule_id	16

Calculation logic

```
loss = x.loss;
loss = loss - (loss * ded);
if (loss < 0) loss = 0;
```

Return to top

Go to Appendix C Multi-peril model support

Back to Contents

Appendix C: Multi-peril model support

ktools now supports multi-peril models through the introduction of the coverage_id in the data structures.

Ground up losses apply at the “Item” level in the Kernel which corresponds to “interest coverage” in business terms, which is the element of financial loss that can be associated with a particular asset. In ktools 2.0, item_id represents the element of financial loss and coverage_id represents the asset with its associated total insured value. If there is more than one item per coverage (as defined in the items data) then each item represents an element of financial loss from a particular peril contributing to the total loss for the asset. For each item, the identification of the peril is held in the areaperil_id, which is a unique key representing a combination of the location (area) and peril.

Multi-peril damage calculation

Ground up losses are calculated by multiplying the damage ratio for an item by the total insured value of its associated coverage (defined in the coverages data). The questions are then; how are these losses combined across items, and how are they correlated?

There are a few ways in which losses can be combined and the reference example in ktools uses a simple rule, which is to sum the losses for each coverage and cap the overall loss to the total insured value. This is what you get when you use the -c parameter in gulcalc to output losses by 'coverage'. It is possible to make the method of combining losses function-driven using another command line parameter, say, if a few standard approaches emerge.

Correlation of item damage is generic in ktools, as damage can either be 100% correlated or independent (see Appendix A Random Numbers). This is no different in the multi-peril case when items represent different elements of financial loss to the same asset, rather than different assets. More sophisticated methods of multi-peril correlation have been implemented for particular models, but as yet no standard approach has been implemented in ktools.

Note that ground up losses by item are passed into the financial module (using gulcalc -i parameter), not losses by coverage. The reason for this is that some policy conditions can have exclusions for particular perils, which must be applied at the item level. In this case, the item ground up losses are passed though as calculated by gulcalc, uncapped, so the limit of total insured value (or the policy coverage limit if different) must be applied as part of the financial module calculations.

Return to top

Back to Contents