Introduction

The purpose of this project is to provide a specification and reference model of the POSIX in-memory variant of the Oasis Kernel.

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

Background

The in-memory solution for the Oasis Monte-Carlo sampling of damage calculations (including financial module and simple outputs) is called the kernel tools or “ktools”. The origin of ktools is that a secure compilable version of the kernel calculation in Oasis was needed for the Oasis Solutions Project so this gave the team the opportunity to bring forward the always planned C-based calculation option. This is timely as the existing SQL-based Oasis provides a comparison calculation and also has full audit trails, so that it can be used to check that the compute engine of ktools is giving correct answers.
ktools is an independent “specification” of a set of processes which means that it defines the processing architecture and data structures and is then implemented as “reference model” which can then be adapted for particular model or business needs. For example, we have a related project for the implementation into Oasis called "oatools" (see 'Related projects' below).

Release 1.3 (and later releases) includes an “in-memory” version of the Oasis central calculations (termed the “Kernel”) written in C++ and C to provide streamed calculation at high computational performance. Prior releases of Oasis used SQL-based back-ends with SQL code generated by the mid-tier libraries. The new variant was delivered as a stand-alone toolkit with R1.4 and in future releases, the compiled code will be distributed by the mid-tier.

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 outputs.
The principle is to stream data through by event end-to-end, with multiple processes being used either sequentially or concurrently, at the control of the user using 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++ and C. This means that they can be compiled in Linux and Windows using the latest GNU compiler toolchain.

Components

The set of components in the Reference Model provided is as follows;

- **eve** is the process distributing utility. Based on the number of events in the input and the number of processes specified as a parameter, eve distributes the events to the processes. The output streams into getmodel.

- **getmodel** is a CDF plug-in to generate the CDFs for a specified list of events. It is the reference example of a plug-in using Oasis kernel format data in binary format. getmodel streams into gulcalc or can be output to a binary file.
- **gulcalc** is the core component which performs the GUL sampling calculations and numerical integration. The output is the Oasis format gul results table. This can be output to a binary file or streamed into another plug-in component, such as a Financial Module or output calculation.
- **fmcalc** is the core component which performs the insured loss calculations from the input ground up loss samples. It has the same functionality as the Oasis Financial Module. The output is the Oasis format FM results table. This can be output to a binary file or streamed into an output calculation.
- **outputcalc** performs results analysis on the ground up loss or insured loss samples, and exports the results table to a csv file. The example given produces an event loss table.
- **cdftocsv** is a utility to output binary format CDFs to a csv.
- **gultocsv** is a utility to output binary format GULs to a csv.
- **fmtocsv** is a utility to output binary format losses to a csv.
- **evetobin** is a utility to convert a list of event_ids into binary format.
- **damagetobin** is a utility to convert the Oasis damage bin dictionary table into binary format.
- **exposuretobin** is a utility to convert the Oasis exposure instance table into binary format.
- **randtobin** is a utility to convert a list of random numbers into binary format.
- **cdfdatatobin** is a utility to convert the Oasis cdf data into binary format.
- **.htmlatatobin** is a utility to convert the Oasis FM instance data into binary format.
- **fmxreftobin** is a utility to convert the Oasis FM xref table into binary format.
- **evetocsv** is a utility to convert the event binary into csv format.
- **damagetocsv** is a utility to convert the Oasis damage bin dictionary binary into csv format.
- **exposuretocsv** is a utility to convert the Oasis exposure instance binary into csv format.
- **randtocsv** is a utility to convert the random numbers binary into csv format.
- **cdfdatatocsv** is a utility to convert the Oasis cdf data binary into csv format.
- **.htmlatatocsv** is a utility to convert the Oasis FM instance binary into csv format.
- **fmxreftocsv** is a utility to convert the Oasis FM xref binary into csv format.

Usage

Standard piping syntax can be used to invoke the components at the command line. For example the following command invokes eve, getmodel, gulcalc, fmcalc, and outputcalc, and exports the results to a csv file.

```
$ eve 1 1 2 | getmodel 1 | gulcalc –S100 –C1 | fmcalc | outputcalc > output.csv
```

Example bash shell, python and vbs scripts are provided along with a binary data package in the /examples folder to demonstrate usage of the toolkit.

Related projects

oatools is a related (private) github project containing additional components which are specific to an implementation of the in-memory kernel in Oasis.

Specifically, it contains a component **gendata** that generates the input data required for the in-memory calculations as binary files, reading from an Oasis SQL Server back-end database. In order to generate the package of executables for the Oasis implementation it is necessary to first build ktools, and then build oatools.

gendata is an example of an implementation-specific component for extracting data to create the binary input files required by ktools, which is principally just the calculation componentware. The input data conversion tools provided in ktools are intended to allow users to generate the required input binaries from csv independently of the original data store and technical environment.

Go to Data streaming architecture overview

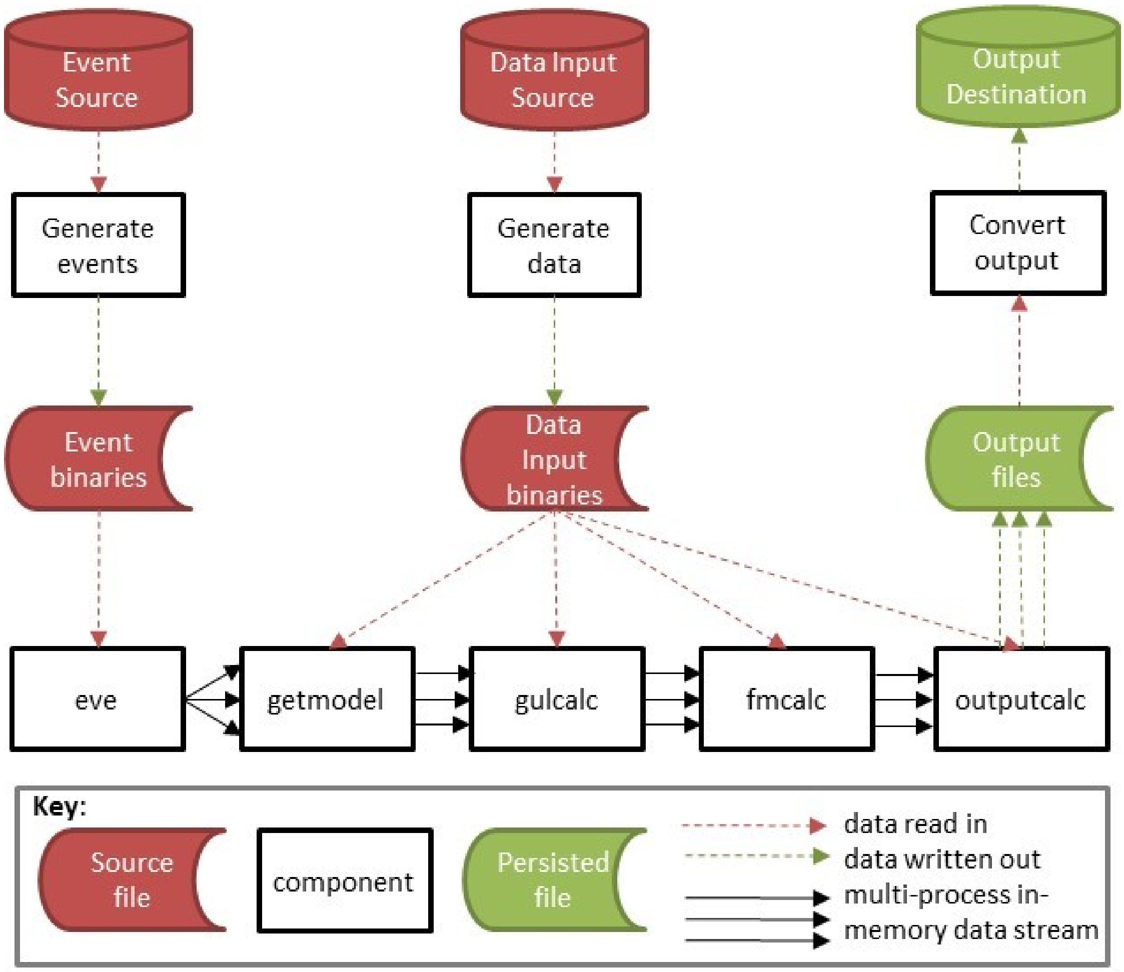
Back to Contents

Table of Contents

Data Streaming Framework Overview

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

Figure 1. Data streaming framework



The in-memory data streams are initiated by the process 'eve' (meaning 'event emitter' rather than in the biblical sense) and shown by solid arrows. The read/write data flows are shown as dashed arrows. Multiple arrows mean multiple processes.

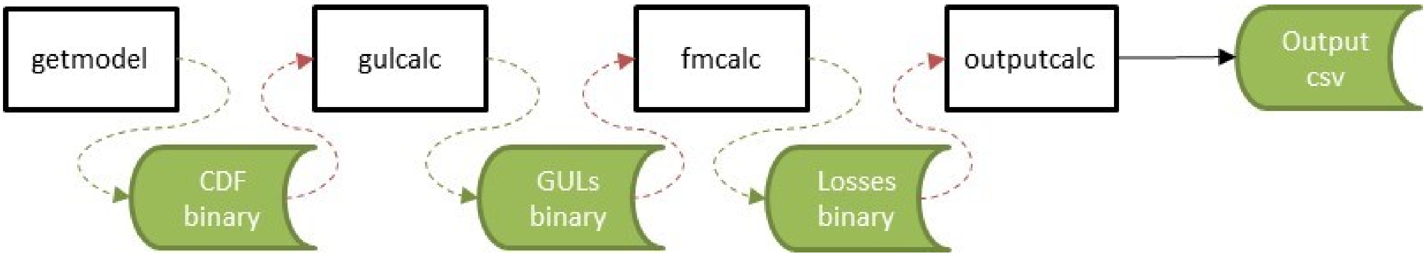
The calculation components are *getmodel*, *gulcalc*, *fmcalc* and *outputcalc*. Each has its own internal data requirements and in the reference model provided with this specification, and displayed in Figure 1, the internal data inputs come from the same source.
However all components are plug-and-play, so the internal data for each can be retrieved from independent external sources.

The standard workflow is straight through in-memory processing to produce a single output file. This minimises the amount of disk I/O at each stage and results in the best performance. This workflow is shown in Figure 2.

Figure 2. Straight-through processing

However it is possible to write the results of each calculation to a binary file, if the data is required to be persisted. This workflow is shown in Figure 3.
At present, intermediate results binaries are required to be read back into memory for downstream calculations.

Figure 3. Multiple output file processing



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

[Go to Specification]

[Back to Contents]

Specification

Introduction

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

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

Most components have a standard input (stdin) and output (stdout) data stream structure. These data structures are not defined explicitly as meta data in the code as they would be in a database language, and they have been designed to minimise the volume flowing through the pipeline. 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 in order to perform the calculations defined in the program.

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, and the 2nd to 4th bytes hold the identifier of the stream.

The current reserved values are as follows;

Higher byte;

Byte 1	Stream type
0	getmodel
1	gulcalc
2	fmcalc

Reserved stream_ids;

Byte 1	Bytes 2-4	Description
0	1	getmodel - Reference example for Oasis format CDF output
1	1	gulcalc - Reference example for Oasis format ground up loss sample output
		fmcalc - Reference example for Oasis format insured loss sample

Byte 1	Bytes 2-4	Description
2	1	output

The final calculation component, outputcalc, has no stream_id as it outputs results directly to csv.

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

- gulcalc can only take stream type 0 (getmodel standard output) as input
- fmcalc can only take stream type 1 (gulcalc standard output) as input
- outputcalc can take either stream type 1 (gulcalc standard output) or 2 (fmcalc standard 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 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

Header packet structure

Name	Type	Bytes	Description	Example
stream_id	int	1/3	Identifier of the data stream type.	0/1
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 |
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 cdfs as standard input and based on the sampling parameters specified, performs Monte Carlo sampling and numerical integration. The output is a table of ground up loss samples in Oasis kernel format, with mean (sidx=0) and standard deviation (sidx=-1).

Input

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

Output

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

Gul header packet structure

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

Gul data packet structure

Name	Type	Bytes	Description	Example
sidx	int	4	Sample index	10
gul	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

sidx	Meaning
-2	numerical integration standard deviation

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).

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
prog_id	int	4	Oasis prog_id	300
layer_id	int	4	Oasis layer_id	300
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.

The sidx field is the same as the sidx in the gul stdout stream.

Return to top

outputcalc

outputcalc is the component which performs results analysis such as an event loss table or EP curve on the sampled output from either the gulcalc or fmcalc program. The output is a results table in csv format.

Input

gulcalc stdout or fmcalc stdout. Binary files of the same data structures can be piped into outputcalc.

Output

No standard output stream. The results table is exported to a csv file. See the Reference model for example output.

Return to top

Go to Reference model

Back to Contents

Reference Model

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

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

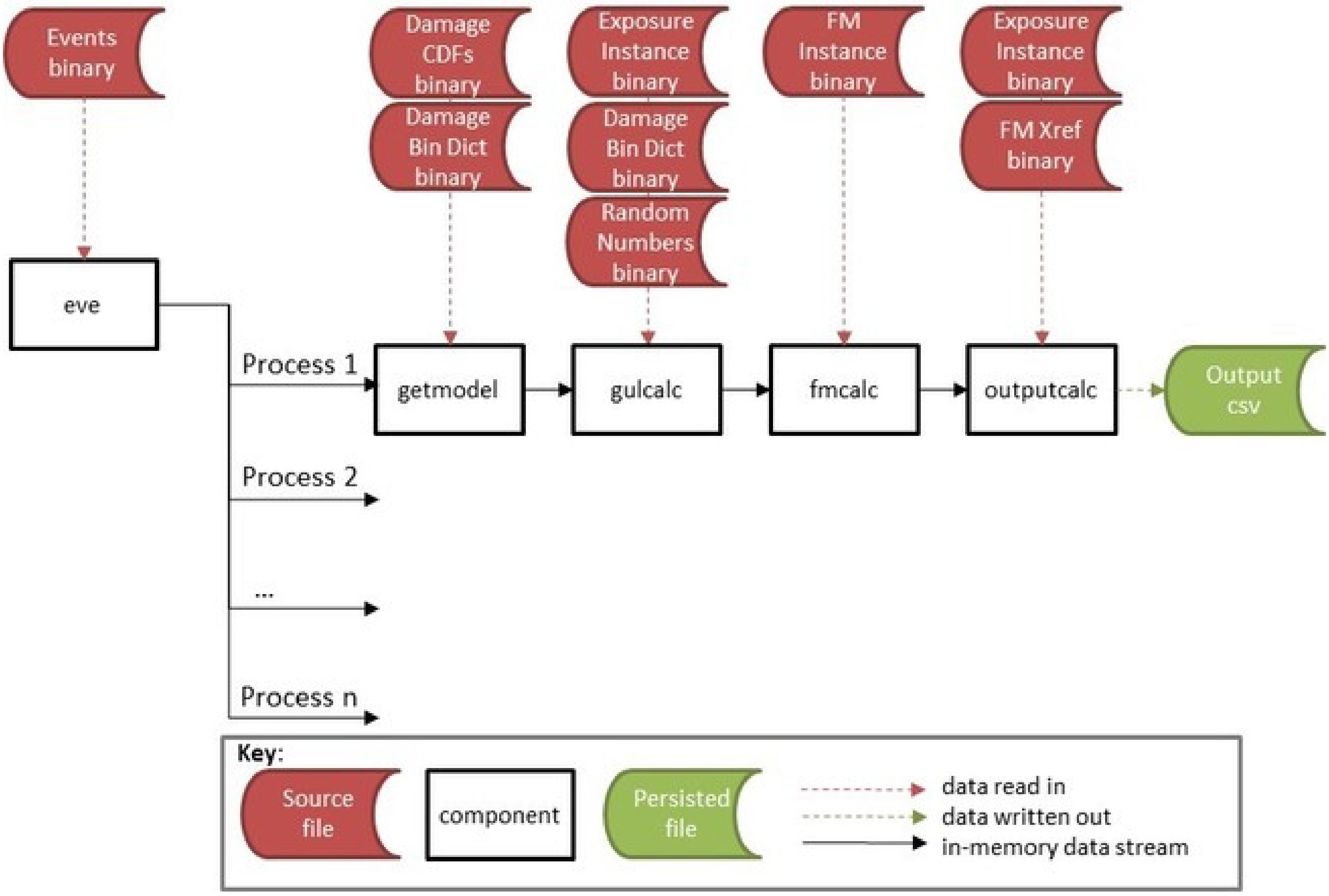
- **eve** is the event distributing utility. Based on the number of events in the input and the number of processes specified as a parameter, eve outputs subsets of the events as a stream. The output streams into getmodel.
- **getmodel** generates a stream of effective damageability cdfs for the input stream of events. The reference example reads in Oasis format damage cdf data from binary file. 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 the Oasis kernel gul sample table. This can be output to a binary file or streamed into fmcalc or outputcalc.
- **fmcalc** performs the insured loss calculations on the ground up loss samples and mean. The output is the Oasis format loss sample table. The functionality covered in fmcalc is the same as the current financial Module in Oasis R1.4 (see R1.2 Financial Module documentation for more information). The result can be output to a binary file or streamed into outputcalc.
- **outputcalc** performs an output analysis on the ground up or loss samples. The reference example is an event loss table containing TIV, sample mean and standard deviation for each event at portfolio/programme summary level. The results are written directly into csv file as there is no downstream processing.

Other components in the Reference Model include;

- **Input data components** to convert input data between csv and binary.
- **Output data components** to convert the binary data stream output to csv.

Figure 1 shows the data stream workflow of the reference model with its particular internal data files.

Figure 1. Reference Model Workflow



The input data for the reference components, shown as red source files, are the events, Damage CDFs, Exposure Instance, Damage Bin Dictionary and FM Instance. These are Oasis concepts with Oasis format data as outlined below.

The following sections explain the usage and internal processes of each of the reference components. The standard input and output data streams for the components are generic and are covered in the Specification. The input data requirements are covered in Input data 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

- chunk number - corresponds to the chunk number in the input file (see below)
- process number - the process number that determines which partition of events are streamed out
- number of processes - the total number of processes which determines the number of partitions

Usage

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

Example

```
$ eve 1 1 2 > events1_2.bin
$ eve 1 1 2 | getmodel 1 | gulcalc -C1 -S100 > gulcalc.bin
```

In this example, the events from the file e_chunk_1_data.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 directory where the program is invoked and has the following filename;

- e_chunk_{chunk}_data.bin

The data structure of e_chunk_{chunk}_data.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 reads pre-generated Oasis format cdfs and converts them into a binary stream. The source input data must have been generated as binary files by a separate program.

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, the 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

The single parameter is chunk_id (int).

Usage

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

Example

```
$ eve 1 1 1 | getmodel 1 | gulcalc -C1 -S100
$ eve 1 1 1 | getmodel 1 > cdf_chunk1.bin
$ getmodel 1 < e_chunk1_data.bin > cdf_chunk1.bin
```

Internal data

The program requires the damage bin dictionary for the model, the Oasis damage cdf in chunks, and an index file for each cdf chunk as binary files. The files are picked up from the directory where the program is invoked and have the following filenames;

- damage_bin_dictionary.bin

And in a cdf specific subdirectory;

- cdf/damage_cdf_chunk_{chunk_id}.bin
- cdf/damage_cdf_chunk_{chunk_id}.idx

The cdfs are ordered by event and streamed out in blocks for each event.

Note that the prob_from field from the existing database table structure of Oasis damage cdfs has been dropped to minimise the size of the table, as it is implied from the prior record prob_to field.

Calculation

The program reads the damage bin mid-point (interpolation field) from the damage bin dictionary and includes it 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. No calculations are performed except to construct the standard output stream.

Return to top

gulcalc

The gulcalc program performs Monte Carlo sampling of ground up loss and calculates mean and standard deviation by numerical integration of the cdfs. The sampling methodology of Oasis classic has been extended beyond linear interpolation to include bin 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

The parameters are;

- -C chunk_id
- -S number of samples
- -R reconciliation mode (optional)
- -r read random numbers from file (optional)
- -L loss threshold (optional)
- -d output random numbers (optional)

Usage

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

Example

```
$ eve 1 1 1 | getmodel 1 | gulcalc -C1 -S100 | fmcalc
$ eve 1 1 1 | getmodel 1 | gulcalc -C1 -S100 > gul_chunk1.bin
$ gulcalc -C1 -S100 < cdf_chunk1.bin > gul_chunk1.bin
```

Internal data

The program requires the damage bin dictionary for the model and the exposure instance table, both as binary files. The files are picked up from the directory where the program is invoked and have the following filenames;

- damage_bin_dictionary.bin
- exposures.bin

If the user specifies -r as a parameter, then the program also picks up a random number file from the working directory. The filename is;
random_{chunk_id}.bin

Calculation

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

For each item cdf and for the number of samples specified, the program reads a random number from the random number file if the -r parameter is used and uses it to sample ground up loss from the cdf using one of three methods. If the -r parameter is not used, a random number is 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).

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.

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

Each sampled damage is multiplied by the item TIV and output to the stdout stream.

A second calculation which occurs in the gulcalc program is of the mean and standard deviation of ground up loss. 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 item TIV. The results are output to the stdout stream as IDX=-1 (mean) and IDX=-2 (standard deviation), for each item and event.

Return to top

fmcalc

fmcalc is the in-memory implementation of the Oasis Financial Module. It applies policy terms and conditions to the ground up losses and produces insured loss sample output.

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 gul stdout stream and internal data.

Usage

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

Example

```
$ eve 1 1 1 | getmodel 1 | gulcalc -C1 -S100 | fmcalc | outputcalc > output.csv
$ eve 1 1 1 | getmodel 1 | gulcalc -C1 -S100 | fmcalc > fm_chunk1.bin
$ fmcalc < gul_chunk1.bin > fm_chunk1.bin
```

Internal data

The program requires the FM Instance data, which is the Oasis native format data tables which describe an insurance programme. This file is picked up from the fm subdirectory;

fm/fm_data.bin

Calculation

fmcalc performs the same calculations as the Oasis Financial Module in R1.5. Information about the Oasis Financial Module can be found on the public area of the Oasis Loss Modelling Framework website, and detailed information and examples are available to Oasis community members in the members area.

[Return to top]()

outputcalc

The reference example of an output produces an event loss table 'ELT' for either ground up loss or insured losses.

Stream_id

There is no output stream_id, the results table is written directly to csv.

Parameters

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

Usage

Either gulcalc or fmcalc stdout can be input streams to outputcalc

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

Example

Either gulcalc or fmcalc stdout streams can be input streams to outputcalc. For example;

```
$ eve 1 1 1 | getmodel 1 | gulcalc -C1 -S100 | outputcalc > output.csv
$ eve 1 1 1 | getmodel 1 | gulcalc -C1 -S100 | fmcalc | outputcalc > output.csv
$ outputcalc < gul.bin > output.csv
$ outputcalc < fm.bin > output.csv
```

Internal data

The program requires the exposures.bin file, and for fmcalc input it requires an additional cross reference file to relate the output_id from the fm stdout stream (which represents an abstract grouping of exposure items) back to the original item_id. This file is picked up from the fm subdirectory;

fm/fmxref.bin

Calculation

The program sums the sampled losses from either gulcalc stream or fmstream across the portfolio/programme by event and sample, and then computes sample mean and standard deviation. It reads the TIVs from the exposure instance table and sums them for the group of items affected by each event.

Return to top

Go to Input data tools

Back to Contents

Input data tools

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

- **evetobin** is a utility to convert a list of event_ids into binary format.
- **damagetobin** is a utility to convert the Oasis damage bin dictionary table into binary format.
- **exposuretobin** is a utility to convert the Oasis exposure instance table into binary format.
- **randtobin** is a utility to convert a list of random numbers into binary format.
- **cdfdatatobin** is a utility to convert the Oasis cdf data into binary format.
- **.htmlatatobin** is a utility to convert the Oasis FM instance data into binary format (to be deprecated).
- **fmprogrammetobin** is a utility to convert the Oasis FM programme data into binary format.
- **fmprofiletobin** is a utility to convert the Oasis FM profile data into binary format.
- **fmpolicytctobin** is a utility to convert the Oasis FM policytc data into binary format.
- **fmxreftobin** is a utility to convert the Oasis FM xref table into binary format.

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).

Note that oatools contains a component 'gendata' which generates all of the input binaries directly from a SQL Server Oasis back-end database. This component is specific to the implementation of the in-memory kernel as a calculation back-end to the Oasis mid-tier which is why it is kept in a separate project.

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

- **evetocsv** is a utility to convert the event binary into csv format.
- **damagetocsv** is a utility to convert the Oasis damage bin dictionary binary into csv format.
- **exposuretocsv** is a utility to convert the Oasis exposure instance binary into csv format.
- **randtocsv** is a utility to convert the random numbers binary into csv format.
- **cdfdatatocsv** is a utility to convert the Oasis cdf data binary into csv format.
- **.htmlatatocsv** is a utility to convert the Oasis FM instance binary into csv format (to be deprecated).
- **fmprogrammetocsv** is a utility to convert the Oasis FM programme data into csv format.
- **fmprofiletocsv** is a utility to convert the Oasis FM profile data into csv format.
- **fmpolicytctocsv** is a utility to convert the Oasis FM policytc data into csv format.
- **fmxreftocsv** is a utility to convert the Oasis FM xref binary into csv format.

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

Events

One or more event binaries are required by eve and getmodel. It must have the following filename format, each uniquely identified by a chunk number (integer >=0);

- e_chunk_{chunk}_data.bin

The chunks represent subsets of events.

In general more than 1 chunk of events is not necessary for the in-memory kernel as the computation can be parallelized across the processes. This feature may be removed in future releases.

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 < e_chunk_1_data.csv > e_chunk_1_data.bin
```

evetocsv

```
$ evetocsv < e_chunk_1_data.bin > e_chunk_1_data.csv
```

Return to top

Damage bins

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 filename format;

- 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 with bin_index starting from 1 and not contain nulls.

damagetobin

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

damagetocsv

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

Return to top

Exposures

The exposures binary contains the list of exposures for which ground up loss will be sampled in the kernel calculations. The data format is that of the Oasis Exposure instance. It is required by gulcalc and outputcalc and must have the following filename format;

- exposures.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
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
tiv	float	4	The total insured value of the item	200000
group_id	int	4	Identifier of the correlation group of the item	1

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

exposuretobin

```
$ exposuretobin < exposures.csv > exposures.bin
```

exposuretocsv

```
$ exposuretocsv < exposures.bin > exposures.csv
```

[Return to top]

Random numbers

One or more random number files 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 should be provided for the same number of chunks as events and must have the following filename format;

- random_{chunk}.bin

If the gulcalc -r parameter is not used, the random number binary is not required and random numbers are generated dynamically in the calculation.

File format

The csv file should contain the runtime number of samples as the first value followed by a simple list of random numbers. It should not contain any headers.

First value;

Name	Type	Bytes	Description	Example

Name	Type	Bytes	Description	Example
number of samples	integer	4	Runtime number of samples	100

Subsequent values;

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

The first value in the file is an exception and should contain the runtime number of samples (integer > 0). This is required for running gulcalc in 'Reconciliation mode' (using gulcalc -R parameter) which uses the same random numbers as Oasis classic and produces identical sampled values. If not running in reconciliation mode, it is not used and can be set to 1.

randtobin

```
$ randtobin < random_1.csv > random_1.bin
```

randtocsv

```
$ randtocsv < random_1.bin > random_1.csv
```

[Return to top](#)

CDFs

One or more cdf data files are required for the getmodel component, as well as an index file containing the starting positions of each event block. These should be located in a cdf sub-directory of the main working directory and have the following filename format;

- cdf/damage_cdf_chunk_{chunk}.bin
- cdf/damage_cdf_chunk_{chunk}.idx

If chunked, the binary file should contain the cdf data for the same events as the corresponding chunk event binary.

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
vulnerability_id	int	4	Oasis vulnerability_id	345456
bin_index	int	4	Identifier of the damage bin	10
prob_to	float	4	The cumulative probability at the upper damage bin threshold	0.765

The data should be ordered by event_id, areaperil_id, vulnerability_id, bin_index ascending with bin_index starting at 1, and not contain nulls. All bins corresponding to the bin indexes in the damage bin dictionary should be present, except records may be truncated after the last bin where the prob_to = 1.

cdfdatatobin

```
$ cdfdatatobin damage_cdf_chunk_1 102 < damage_cdf_chunk_1.bin
```

This command will create a binary file damage_cdf_chunk_1.bin and an index file damage_cdf_chunk_1.idx in the working directory.

In general the usage is;

```
$ cdfdatatobin damage_cdf_chunk_[chunk] [maxbins] < input.csv
```

chunk is an integer and maxbins is the maximum number of bins in the cdf. input.csv must conform to the csv format given above.

cdfdatatocsv

```
$ cdfdatatocsv < damage_cdf_chunk_1.bin > damage_cdf_chunk_1.csv
```

[Return to top](#)

fm data

The fm data binary file contains the policy terms and conditions required to perform a loss calculation, and is required for fmcalc only. The source format is Oasis FM Instance data, which is the Oasis native format data tables which describe an insurance programme. These four tables have been combined into one with the below structure.

This file should be located in a fm sub-directory of the main working directory and have the following filename.

- fm/fm_data.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	56745
agg_id	int	4	Oasis Financial Module agg_id	546
prog_id	int	4	Oasis Financial Module prog_id	4
level_id	int	4	Oasis Financial Module level_id	1
policytc_id	int	4	Oasis Financial Module policytc_id	34
layer_id	int	4	Oasis Financial Module layer_id	1
calcrule_id	int	4	Oasis Financial Module calcrule_id	2
allocrule_id	int	4	Oasis Financial Module allocrule_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	0.25
deductible_prop_of_loss	float	4	Deductible as a proportion of loss	0.05
limit_prop_of_loss	float	4	Limit as a proportion of loss	0.5
deductible_prop_of_tiv	float	4	Deductible as a proportion of TIV	0.05
limit_prop_of_tiv	float	4	Limit as a proportion of TIV	0.8
deductible_prop_of_limit	float	4	Deductible as a proportion of limit	0.1

The data should be ordered by item_id and not contain any nulls (replace with 0).

.htmlatatobin

```
$ .htmlatatobin < fm_data.csv > fm_data.bin
```

.htmlatatocsv

```
$ .htmlatatocsv < fm_data.bin > fm_data.csv
```

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 file should be located in a fm sub-directory of the main working directory and have the following filename.

- fm/fm_programme.bin

File format

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

Name	Type	Bytes	Description	Example
prog_id	int	4	Oasis Financial Module prog_id	1
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 where 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.
- Each programme table may only have a single integer value in prog_id. Note that this field is a cross-reference to a separate prog dictionary and business meaningful information such as account number, and is not currently used in calculations. This field may be deprecated in future versions.
- 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
```

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 file should be located in a fm sub-directory of the main working directory and have the following filename.

- fm/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
sourcerule_id	int	4	Oasis Financial Module sourcerule_id	0
levelrule_id	int	4	Oasis Financial Module levelrule_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 fields not currently used at all are ccy_id, sourcerule_id and levelrule_id

fmprofiletobin

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

fmprofiletocsv

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

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 is required for fmcalc only.

This file should be located in a fm sub-directory of the main working directory and have the following filename.

- fm/fm_policytc.bin

File format

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

Name	Type	Bytes	Description	Example
prog_id	int	4	Oasis Financial Module prog_id	1
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 xref

The fmxref binary file contains cross reference data linking the output_id in the fmcalc output back to item_id, and is required for outputcalc only. This should be located in a fm sub-directory of the main working directory and have the following filename.

- fm/fmxref.bin

File format

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

Name	Type	Bytes	Description	Example

Name	Type	Bytes	Description	Example
item_id	int	4	Identifier of the exposure item	56745
output_id	int	4	Oasis Financial Module output_id	546

The data should not contain any nulls.

fmxreftobin

```
$ fmxreftobin < fmxref.csv > fmxref.bin
```

fmxreftocsv

```
$ fmxreftocsv < fmxref.bin > fmxref.csv
```

[Return to top](#)

[Go to Output data tools](#)

[Back to Contents](#)

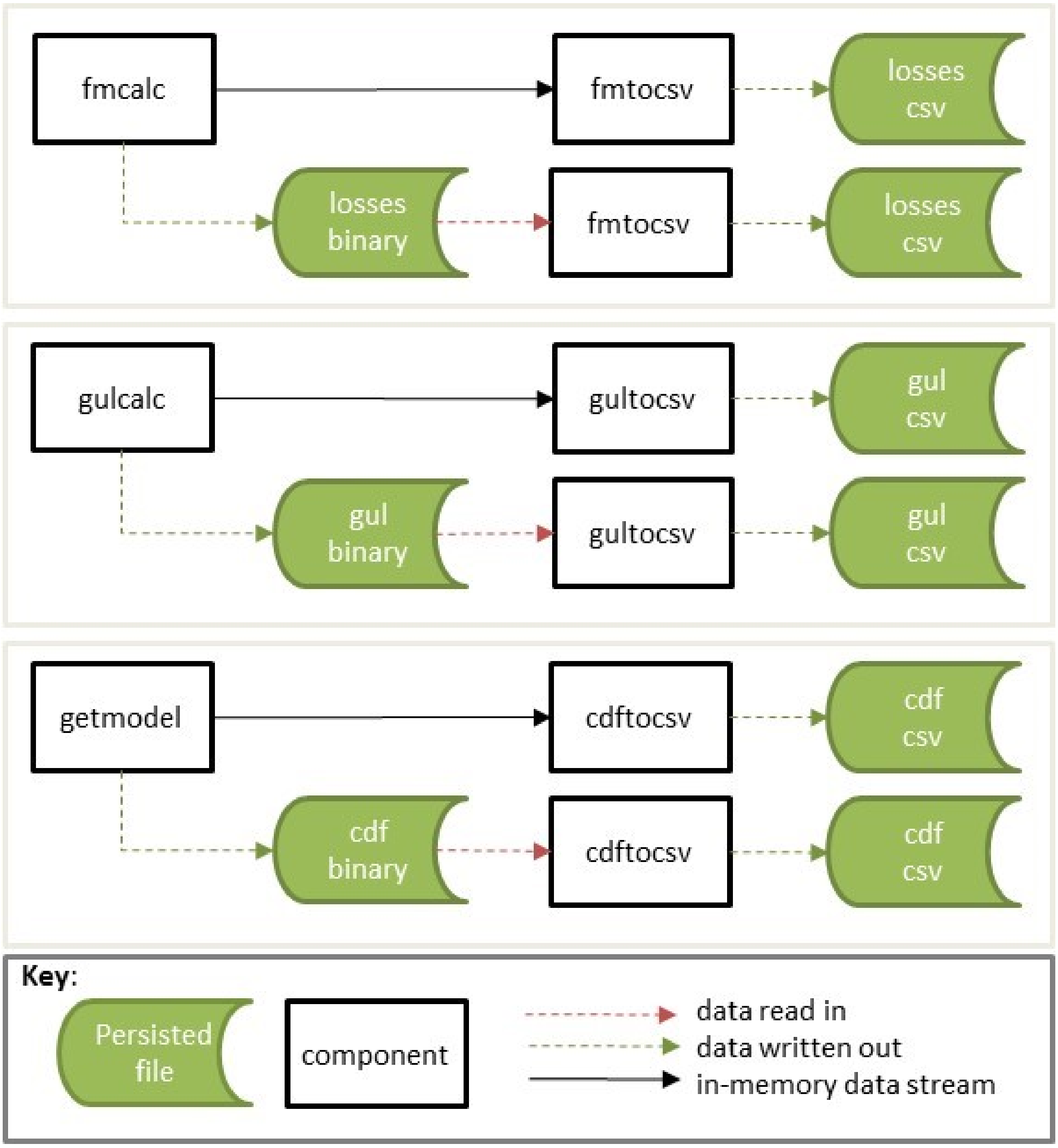
Output data tools

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 input.
- **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 input.
- **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 input.

Figure 1 shows the workflows for the data conversion components.

Figure 1. Data Conversion 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 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 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 1 | getmodel | gulcalc -S100 -C1 | gultocsv > gulcalc.csv
$ gultocsv < gulcalc.bin > gulcalc.csv

Output

Csv file with the following fields;

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
gul	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 1 | getmodel | gulcalc -S100 -C1 | fmcalc | fmtocsv > fmcalc.csv
$ fmtocsv < fmcalc.bin > fmcalc.csv

Input

Same as fmcalc output or a binary file of the same format can be piped into fmtocsv.

Output

Csv file with the following fields;

Name	Type	Bytes	Description	Example
event_id	int	4	Oasis event_id	4545
prog_id	int	4	Oasis prog_id	1
layer_id	int	4	Oasis layer_id	1
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

Go to Planned work

Back to Contents

Planned work

1. Generate cdfs in getmodel

The cdf data is assumed to be pre-calculated and read into ktools from binaries. This will be changed to calculate cdfs as part of getmodel from the model file inputs.

2. Remove chunk concept from eve and getmodel

The input data for the reference components eve and getmodel can be split across several files, where each is identified by a chunk_id under a fixed naming convention. Eve and getmodel have 'chunk_id' as an input parameter which identifies the relevant input binary file.

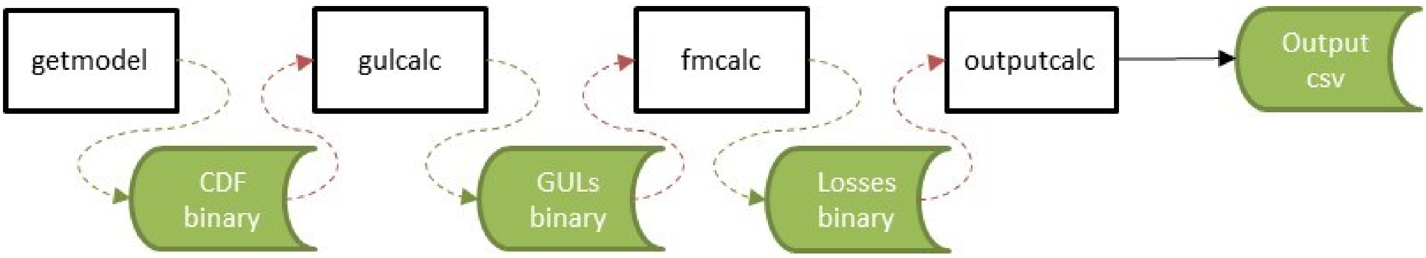
Because chunk is an Oasis mid-tier concept, the chunk parameter will be removed and each internal data input to the ktools reference model will be a single consolidated file.

Note that the events can still be chunked and each chunk distributed to a separate back-end running the in-memory kernel.

3. Multi-output workflows

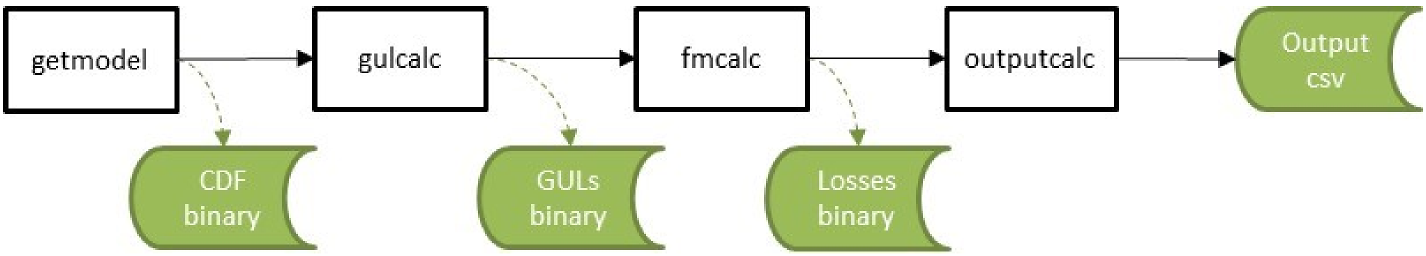
Currently if the results of the intermediate calculation steps are required to be persisted, then they must be written out to disk and read back into memory to continue downstream processing.

Figure 1. Multiple output file processing - now



The plan is to enable intermediate calculation steps to be written out to disk whilst continuing the in-memory workflow.

Figure 2. Multiple output file processing - future



4. Reduce sidx field from 4 bytes to 2 bytes

The sample index field is currently a 4 byte integer format for simplicity of design but we estimate gulcalc stdout data volumes could be reduced by approximately 25% if we reduce it to 2 bytes. This would mean a maximum number of samples of 64,000.

5. Add random number limit as a parameter to gulcalc

There is a fixed limit of 1 million random numbers per event when the dynamic random number option (no -r parameter) is used in gulcalc. This limit was imposed to improve performance. We plan to add this limit as an optional parameter to gulcalc.

6. More outputs

More in-memory output components to calculate EP curves and event/year loss tables at different summary levels will be added.

[Go to Random numbers]

[Back to Contents]

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 exposure input file (exposures.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 ridx position.

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

1. Generate dynamically during the calculation

Usage

No random number parameters are required, this is the default.

Example

```
$ gulcalc -C1 -S100
```

This will run 100 samples using 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 1 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 S, 1000000)

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 -C1 -S100 -r
```

This will run 100 samples using random numbers from file random_1.bin.

Method

The random number file(s) is read into memory at the start of the gulcalc process. The same set of numbers are used for all events in the corresponding event file e_chunk_1_data.bin.

The ridx is generated from the event_id and group_id using the following modulus function;

ridx=mod(event_id x P1 + group_id x P2, D)

D 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 D.

This formula pseudo-randomly assigns ridx indexes to each event_id and group_id combo between 0 and D-1.

3. Use numbers from random number file in 'reconciliation mode'

Usage

Use -r -R as parameters

Example

```
$ gulcalc -C1 -S100 -r -R
```

This will run 100 samples using random numbers from file random_1.bin in reconciliation mode.

Method

Reconciliation mode uses the same method as 2. but calculates the ridx in the same way as Oasis classic, thereby producing exactly the same losses for comparison.

In Oasis classic, random numbers are held in a matrix with the sample index in columns and the ridx is a reference into a row of random numbers for a sample set of size S.

In reconciliation mode, the divisor D is set to the total number of rows. This is calculated from the number of columns S which is the first 4 byte integer value in the random number file.

Go to Financial Module

Back to Contents

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

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 Reference Model.
- The formats of the input files are covered in Input Data.

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 FM Profiles for more details.

Design

The Oasis Financial Module is a data-driven process design for calculating the losses on insurance policies. Like the Exposure Module, it is an abstract design in order to cater for the many variations and has three 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.

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.

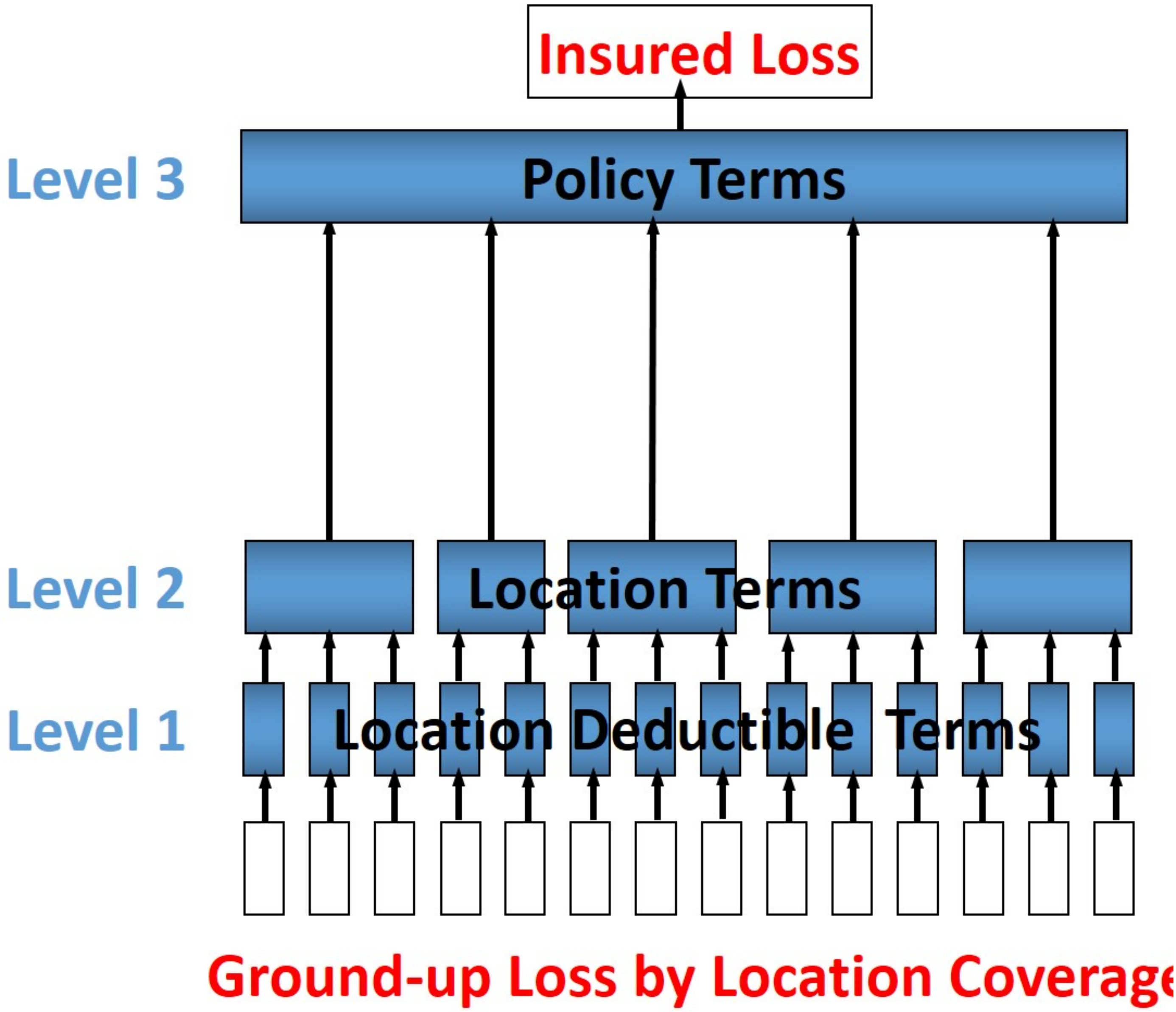
Data requirements

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.

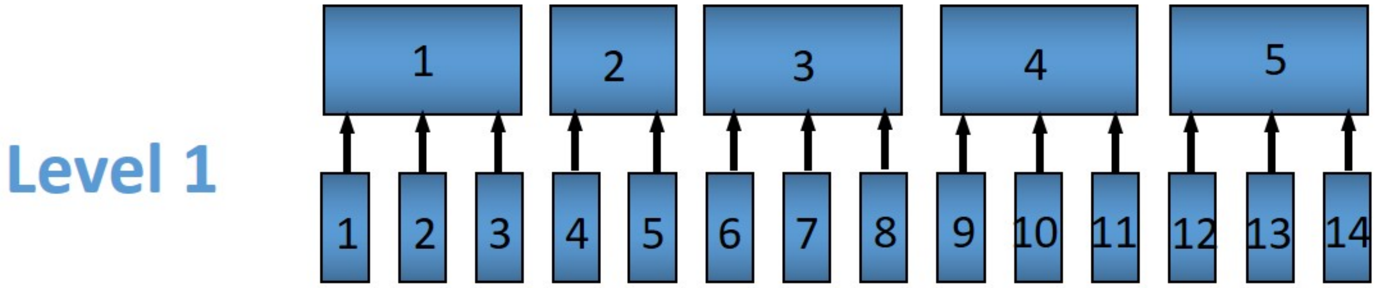
There are many ways an insurance loss can be calculated with many different terms and conditions. For instance, there may be deductibles applied to each element of coverage (e.g. a buildings damage deductible), some site-specific deductibles or limits, and some overall policy deductibles and limits and 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	idx	gul
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:

prog_id	from_agg_id	level_id	to_agg_id
1	1	1	1
1	2	1	1
1	3	1	1
1	4	1	2
1	1	2	1
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 Input tools).

For any given profile we have four 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.
- **sourcerule_id** (not currently used), which is used for conditional logic if TIV (for example) needs to be used in a calculation.
- **levelrule_id** (not currently used) used for processing level-specific rules such as "special conditions".

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:

prog_id	layer_id	level_id	agg_id	policytc_id
1	1	1	1	1
1	1	1	2	2
1	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;

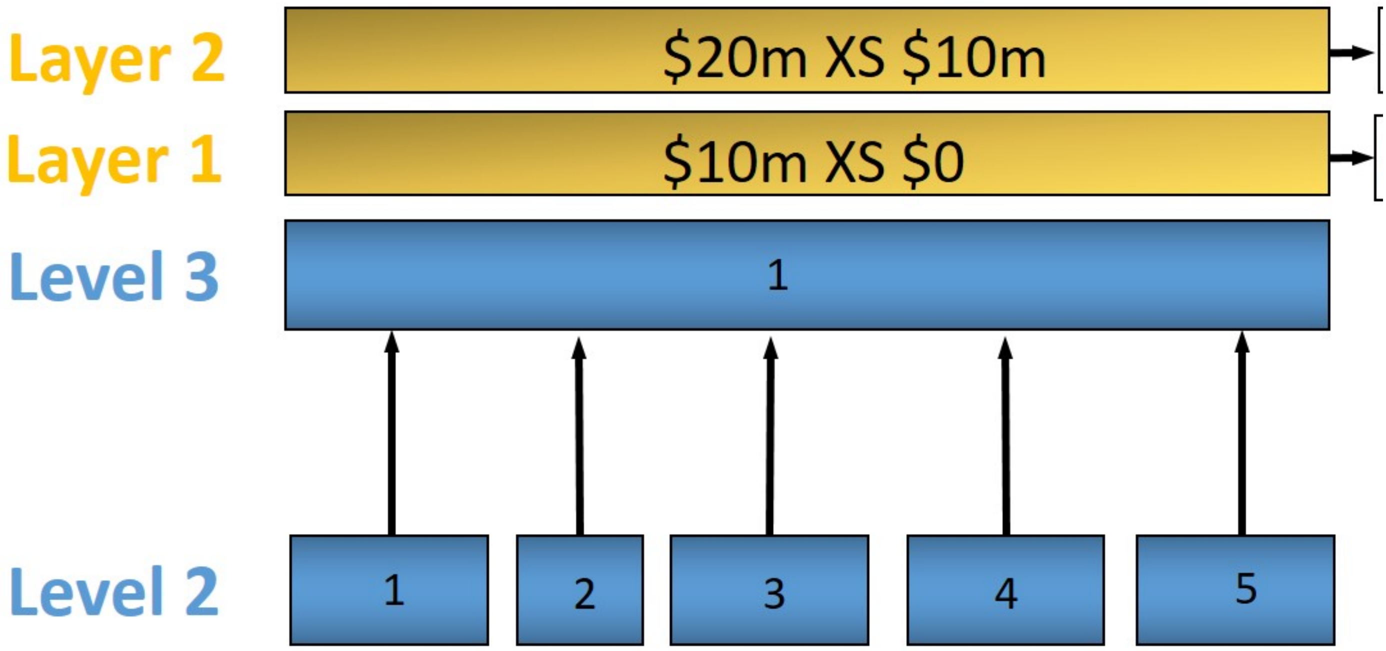
Apply policytc_id 3 to agg_id 1

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;

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

Outputs and back-allocation

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

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

The output_id represents some grouping of items, depending on what allocation rule is applied at the final level of calculation;

- If allocrule_id = 0 for all policytc_ids at the final level then output_id = agg_id of the final level
- If allocrule_id = 1 for all policytc_ids at the final level then output_id = from_agg_id of the first level.

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 in a separate output module, using an item_id to summary level cross reference 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.

Go to FM profiles

Back to Contents

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
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

Attributes	Example
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;
```

Back to Contents