Result File Formats for OMNeT++ 4.0

Andras Varga, Tamas Borbely

The file format described here applies to *both output vector and output scalar files*. Their formats are consistent, only the types of entries occurring into them are different. This unified format also means that they can be read with a common routine.

Result files are *line oriented*. A line consists of one or more tokens, separated by whitespace. Tokens either don't contain whitespace, or or whitespace is escaped using a backslash, or are quoted using double quotes. Escaping within quotes using backslashes is also permitted.

The first token of a line usually identifies the type of the entry. A notable exception is an output vector data line, which begins with a numeric identifier of the given output vector.

A line starting with # as the first non-whitespace character denotes a comment, and is to be ignored during processing.

Result files are written from simulation runs. A simulation run generates physically contiguous sets of lines into one or more result files. (That is, lines from different runs do not arbitrarily mix in the files.)

A run is identified by a unique textual *runId*, which appears in all result files written during that run. The runId may appear on the user interface, so it should be somewhat meaningful to the user. Nothing should be assumed about the particular format of runId, but it will be some string concatenated from the simulated network's name, the time/date, the hostname, and other pieces of data to make it unique.

A simulation run will typically write into two result files (.vec and .sca). However, when using parallel distributed simulation, the user will end up with several .vec and .sca files, because different partitions (a separate process each) will write into different files. However, all these files will contain the same runId, so it is possible to relate data that belong together.

Entry types:

"Run"

Marks the beginning of a new run in the file. Entries after this line belong to this run.

Format:

run *runId*

The following forms are obsolete, but should be understood nevertheless:

run *runNumber*
run *runNumber networkName*
run *runNumber networkName dateTime*

These forms can be distinguished from the first one during processing, because runId is not numeric while runNumber is. During processing, the program can construct an artificial runId from the file name, run number, and the line number at which this line appears in the file. (The runNumber comes from omnetpp.ini, and it is not guaranteed to be unique).
In the new format, runNumber, networkName and dateTime will appear on separate lines as *run attributes*.

Old output vector files don't contain a "run" line. During processing, if any line type is encountered before a "run" line, then an implicit "run" has to be created and subsequenet entries be assumed to be part of that.

Performance note: if not the whole file is kept in memory during analysis, then the runs in the file (i.e. offsets of the "run" lines) may be indexed for more efficient random access.

Example:

```
run "largeNet-20050710-14:34:11-localhost-12374"
```

"Run Attributes"

Contains an attribute for the current run. These attributes include the network name, the time/date of execution, the experiment/measurement/replication labels, the random number seeds, configuration options that took effect such as the scheduler class, etc.

Format:

attr *name value*

TODO define the list of recognized attribute names

Example

```
attr run 1
attr network "largeNet"
attr date "2005-07-10 14:34:11"
attr host "localhost"
attr inifile "xxx.ini"
attr experiment "blabla"
attr measurement "rtete"
attr replication "12th"
```

```
attr numseeds 2
attr seed-0-mt 573367
attr seed-1-mt 124643
```

"Param"

Contains a module parameter value for the given run. This is needed so that module parameters may be included in the analysis (e.g. to identify the load for a "thruput vs load" plot).

It is not feasible to simply store all parameters of all modules in the result file (it's just too much). We assume that NED files are invariant and don't store parameters defined in them. However, we store parameter assignments that come from omnetpp.ini, in their original wildcard form (i.e. not expanded) to conserve space. Parameter values entered interactively by the user are also stored.

When the original NED files are present, it should thus be possible to reconstruct all parameters for the given simulation.

Format:

param *parameterNamePattern value*

Example:

```
param **.gen.sendIaTime   exponential(0.01)
param **.gen.msgLength    10
param **.fifo.bitsPerSec  1000
```

"Scalar"

Contains an output scalar value. This is the same as in older output scalar files.

Format:

scalar *moduleName scalarName value*

TODO room to include the unit (seconds, bits, megabit/second, etc), and possible extra data (for tagging, commenting, etc?)

Examples:

```
scalar "net.switchA.relay" "processed frames" 100
```

"Vector"

Defines an output vector. This is the same as in older output vector files.

Format:

vector *vectorId moduleName vectorName*
vector *vectorId moduleName vectorName* 1

vector *vectorId moduleName vectorName columns*

Where columns is a string encoding the meaning and ordering the columns of data lines. Characters of the string means:

E event number
T simulation time
V vector value

The default value of columns is 'TV' for compatibility with old vector files.

"Vector Attributes"

Vector attributes may follow the definition of vectors. These attibutes include vector unit, enum definitions, interpolation mode, etc.

Format:

attr name value

"Vector Data"

Adds a value to an output vector. This is the same as in older output vector files.

Format:

vectorId column1 column2 ...

Simulation times and event numbers *within an output vector* are required to be in increasing order.

Performance note: Data lines belonging to the same output vector may be written out in clusters (of sizes roughly multiple of the disk's physical block size). Then, since an output vector file is typically not kept in memory during analysis, indexing the start offsets of these clusters allows one to read the file and seek in it more efficiently. This does not require any change or extension to the file format.

"Histogram"

Contains histogram data.

Format:

histogram *moduleName histogramName*
bin –INF *value0*
bin *binLowerBound1 value1*
bin *binLowerBound2 value2*
…

Histogram name and module is defined on the **histogram** line, which is followed by several **bin** lines to contain data. Any non-**bin** line marks the end of the histogram data.

The *binLowerBound* column of **bin** lines represent the lower bound of the given histogram cell. **Bin** lines are in increasing *binLowerBound* order.

The *value* column of **bin** lines represent observation count in the given cell: *value k* is the number of observations greater or equal than *binLowerBound k*, but smaller than *binLowerBound k+1*. *Value* is not necessarily an integer, because the cKSplit and cPSquare algorithms produce non-integer estimates. The first **bin** line is the underflow cell, and the last **bin** line is the overflow cell.