User's Guide for ContactCenters Simulation Library

Generic simulation applications API Specification

Version: March 17, 2014

Eric Buist

This is the API specification for the generic simulators using the ContactCenters library. This API provides a common interface simulators can implement to be manipulated uniformly by application programs such as results analyzers and optimizers.

Contents

Package `umontreal.iro.lecuyer.contactcenters.app` **2**

ContactCenterInfo . . . 3
ContactCenterEval . . . 11
ContactCenterSim . . . 16
ContactCenterSimWithObservations . . . 21
ContactCenterSimWithObservationSets . . . 22
ObservableContactCenterSim . . . 24
ContactCenterSimListener . . . 25
ContactCenterProgressBar . . . 26
SimStoppingCondition . . . 27
SearchStoppingCondition . . . 28
PerformanceMeasureType . . . 30
EstimationType . . . 60
RowType . . . 62
ColumnType . . . 66
PerformanceMeasureFormat . . . 68
PerformanceMeasureFormatText . . . 74
PerformanceMeasureFormatExcel . . . 82
EvalOptionType . . . 94
ContactCenterEvalResults . . . 96
ContactCenterSimResults . . . 99
ContactCenterEvalResultsConverter . . . 100
AbstractContactCenterInfo . . . 107
AbstractContactCenterEval . . . 108
AbstractContactCenterSim . . . 109
RouterPolicyType . . . 110
ArrivalProcessType . . . 112
DialerPolicyType . . . 116

Package `umontreal.iro.lecuyer.contactcenters.app.trace` **118**

ContactTrace . . . 119
FileContactTrace . . . 120
ExcelContactTrace . . . 121
DBContactTrace . . . 122

Package umontreal.iro.lecuyer.contactcenters.app

Provides an interface for application programs interacting with a contact center simulator or evaluation system. The ContactCenters library can be used to create simulators modeling complex contact centers. These simulators may be accessed from other programs, e.g., for optimization or statistical analysis. Programs may also use analytical formulas to get approximations of some performance measures, for more efficiency or for comparing with simulation. This package defines an abstract interface permitting such programs to get their data in a standardized way. Package `umontreal.iro.lecuyer.contactcenters.msk` provides a generic simulator implementing this interface.

The defined interface, `ContactCenterEval`, represents a system capable of performing evaluations for a contact center. We define an *evaluation* as a process using user-specified parameters to estimate some performance measures with analytical formulas or simulation. The interface provides a standardized mechanism to run evaluations, and obtain estimates of performance measures. It also supports evaluation options, i.e., parameters that can be changed independently of the specific implementation.

The `ContactCenterSim` extends the `ContactCenterEval` interface with methods for obtaining the variance, the minimum, the maximum, and computing confidence intervals. These statistics are available only when performing several replications of a simulation, or when dividing a simulation into batches.

This package also provides an enumerated type, `PerformanceMeasureType`, that represents the type of a performance measure. An element of this enum can be used as a key to several methods in this package, or queried for information such as a name, the type of estimation, etc. Facilities are also provided to format matrices containing estimates of performance measures of a given type.

This package also defines some commonly-used parameter objects for storing parameters for simulation experiments, and the values of thresholds for computing service levels.

ContactCenterInfo

Represents an object capable of returning general information about a contact center.

```
package umontreal.iro.lecuyer.contactcenters.app;

public interface ContactCenterInfo
```

Methods

`public int getNumContactTypes()`

Returns the total number of contact types supported by this contact center. This should be the same as `getNumInContactTypes() + getNumOutContactTypes()`.

Returns the total number of contact types.

`public int getNumInContactTypes()`

Returns the total number of inbound contact types for this contact center.

Returns the total number of inbound contact types.

`public int getNumOutContactTypes()`

Returns the total number of outbound contact types for this contact center.

Returns the total number of outbound contact types.

`public int getNumAgentGroups()`

Returns the total number of agent groups supported by this contact center.

Returns the total number of agent groups.

`public int getNumWaitingQueues()`

Returns the total number of waiting queues capable of storing contacts.

Returns the number of waiting queues.

`public int getNumMainPeriods()`

Returns the number of main periods used for evaluation, as defined in `PeriodChangeEvent`. For a steady-state evaluation on a single period, this always returns 1, even if the model defines several period.

Returns the total number of main periods.

`public int getNumContactTypeSegments()`

Returns the number of user-defined segments regrouping contact types.

Returns the number of segments regrouping contact types.

`public int getNumInContactTypeSegments()`

Returns the number of user-defined segments regrouping inbound contact types.

Returns the number of segments regrouping inbound contact types.

`public int getNumOutContactTypeSegments()`

Returns the number of user-defined segments regrouping outbound contact types.

Returns the number of segments regrouping outbound contact types.

`public int getNumAgentGroupSegments()`

Returns the number of user-defined segments regrouping agent groups.

Returns the number of segments regrouping agent groups.

`public int getNumMainPeriodSegments()`

Returns the number of user-defined segments regrouping main periods.

Returns the number of segments regrouping main periods.

`public int getNumWaitingQueueSegments()`

Returns the number of user-defined segments regrouping waiting queues.

The result of this method depends on the role of the waiting queues, which depends on the router's policy. For example, if waiting queues correspond to contact types, this returns the result of `getNumContactTypeSegments()`.

Returns the number of segments regrouping waiting queues.

`public int getNumContactTypesWithSegments()`

Returns the number of contact types including segments regrouping several contact types. If $K \leq 1$, this returns the result of `getNumContactTypes()`. Otherwise, this returns the sum of `getNumContactTypes()`, `getNumContactTypeSegments()`, and 1.

Returns the number of contact types including segments.

`public int getNumInContactTypesWithSegments()`

Returns the number of inbound contact types including segments regrouping several inbound contact types. If $K_{\mathrm{I}} \leq 1$, this returns the result of `getNumInContactTypes()`. Otherwise, this returns the sum of `getNumInContactTypes()`, `getNumInContactTypeSegments()`, and 1.

Returns the number of inbound contact types including segments.

`public int getNumOutContactTypesWithSegments()`

Returns the number of outbound contact types including segments regrouping several outbound contact types. If $K_{\mathrm{O}} \leq 1$, this returns the result of `getNumOutContactTypes()`. Otherwise, this returns the sum of `getNumOutContactTypes()`, `getNumOutContactTypeSegments()`, and 1.

Returns the number of outbound contact types including segments.

`public int getNumAgentGroupsWithSegments()`

Returns the number of agent groups including segments regrouping several agent groups. If $I \leq 1$, this returns the result of `getNumAgentGroups()`. Otherwise, this returns the sum of `getNumAgentGroups()`, `getNumAgentGroupSegments()`, and 1.

Returns the number of agent groups including segments.

`public int getNumMainPeriodsWithSegments()`

Returns the number of main periods including segments regrouping several main periods. If $P \leq 1$, this returns the result of `getNumMainPeriods()`. Otherwise, this returns the sum of `getNumMainPeriods()`, `getNumMainPeriodSegments()`, and 1.

Returns the number of main periods including segments.

`public int getNumWaitingQueuesWithSegments()`

Returns the number of waiting queues including segments regrouping several waiting queues. If the number of waiting queues is smaller than two, this returns the result of `getNumWaitingQueues()`. Otherwise, this returns the sum of `getNumWaitingQueues()`, `getNumWaitingQueueSegments()`, and 1.

Returns the number of waiting queues including segments.

`public String getContactTypeName (int k)`

Returns the name associated with the contact type `k`, where `k` is a number greater than or equal to 0 and smaller than `getNumContactTypes()`. The first `getNumInContactTypes()` indices are inbound contact types whereas the remaining indices are outbound contact types. If no contact type name is available, this returns `null`.

Parameter

`k` the contact type identifier.

Returns the contact type name or `null` if no name is defined.

Throws

`IndexOutOfBoundsException` if the contact type identifier is negative or greater than or equal to `getNumContactTypes()`.

`public String getAgentGroupName (int i)`

Returns the name associated with the agent group identifier `i`. If no name is associated with a given agent group, this returns `null`.

Parameter

`i` the identifier of the agent group.

Returns the agent group name, or `null`.

Throws

`IndexOutOfBoundsException` if the agent group identifier is negative or greater than or equal to `getNumAgentGroups()`.

`public String getMainPeriodName (int mp)`

Returns the name corresponding to the main period `mp`. This can return `null` or an empty string for unnamed periods.

Parameter

`mp` the index of the main period.

Returns the name of the main period.

`public String getWaitingQueueName (int q)`

Returns the name of the waiting queue with index `q` used by the evaluation. If the waiting queue has no name, returns `null`.

Parameter

`q` the index of the waiting queue.

Returns the name of the waiting queue.

Throws

`IndexOutOfBoundsException` if the waiting queue identifier is negative or greater than or equal to `getNumWaitingQueues()`.

`public String getContactTypeSegmentName (int k)`

Returns the name associated with the contact type segment `k`, where `k` is a number greater than or equal to 0 and smaller than `getNumContactTypeSegments()`. If no segment name is available, this returns `null`.

Parameter

`k` the contact type segment identifier.

Returns the segment name or `null` if no name is defined.

Throws

`IndexOutOfBoundsException` if the contact type segment identifier is negative or greater than or equal to `getNumContactTypeSegments()`.

`public String getInContactTypeSegmentName (int k)`

Returns the name associated with the inbound contact type segment `k`, where `k` is a number greater than or equal to 0 and smaller than `getNumInContactTypeSegments()`. If no segment name is available, this returns `null`.

Parameter

`k` the inbound contact type segment identifier.

Returns the segment name or `null` if no name is defined.

Throws

`IndexOutOfBoundsException` if the identifier of the segment regrouping inbound contact types is negative or greater than or equal to `getNumInContactTypeSegments()`.

`public String getOutContactTypeSegmentName (int k)`

Returns the name associated with the outbound contact type segment `k`, where `k` is a number greater than or equal to 0 and smaller than `getNumOutContactTypeSegments()`. If no segment name is available, this returns `null`.

Parameter

`k` the outbound contact type segment identifier.

Returns the segment name or `null` if no name is defined.

Throws

`IndexOutOfBoundsException` if the identifier of the segment regrouping outbound contact types is negative or greater than or equal to `getNumOutContactTypeSegments()`.

`public String getAgentGroupSegmentName (int i)`

Returns the name associated with the agent group segment `i`, where `i` is a number greater than or equal to 0 and smaller than `getNumAgentGroupSegments()`. If no segment name is available, this returns `null`.

Parameter

`i` the agent group segment identifier.

Returns the segment name or `null` if no name is defined.

Throws

`IndexOutOfBoundsException` if the agent group segment identifier is negative or greater than or equal to `getNumAgentGroupSegments()`.

`public String getMainPeriodSegmentName (int mp)`

Returns the name associated with the main period segment `mp`, where `mp` is a number greater than or equal to 0 and smaller than `getNumMainPeriodSegments()`. If no segment name is available, this returns `null`.

Parameter

`mp` the main period segment identifier.

Returns the segment name or `null` if no name is defined.

Throws

`IndexOutOfBoundsException` if the main period segment identifier is negative or greater than or equal to `getNumMainPeriodSegments()`.

`public String getWaitingQueueSegmentName (int q)`

Returns the name associated with the waiting queue segment `q`, where `q` is a number greater than or equal to 0 and smaller than `getNumMainPeriodSegments()`. If no segment name is available, this returns `null`.

Parameter

`q` the waiting queue segment identifier.

Returns the segment name or `null` if no name is defined.

Throws

`IndexOutOfBoundsException` if the main period segment identifier is negative or greater than or equal to `getNumMainPeriodSegments()`.

`public String getMatrixOfAWTName (int m)`

Returns the name associated with the matrix of AWTs with index `m`. This method returns `null` if no name is associated with the matrix. This name can be used, e.g., to give the AWT if the same AWT is used for all contact types and periods.

Parameter

`m` the index of the matrix of AWTs.

Returns the name associated with the matrix.

Throws

`IllegalArgumentException` if `m` is negative or greater than or equal to the value returned by `getNumMatricesOfAWT()`.

`public Map<String, String> getContactTypeProperties (int k)`

Returns the properties associated with contact type `k`. Properties are additional strings describing a contact type. This can include the language of the customers, the originating region, etc. If no property is defined for the given contact type, this method returns an empty map.

Parameter

`k` the contact type identifier.

Returns the map of properties.

Throws

`IndexOutOfBoundsException` if the contact type identifier is negative or greater than or equal to `getNumContactTypes()`.

`public Map<String, String> getAgentGroupProperties (int i)`

This method is similar to `getContactTypeProperties (int)`, for agent groups instead of contact types.

Parameter

`i` the agent group identifier.

Returns the map of properties.

`public Map<String, String> getWaitingQueueProperties (int q)`

This method is similar to `getContactTypeProperties (int)`, for waiting queues instead of contact types.

Parameter

`q` the waiting queue identifier.

Returns the map of properties.

```
public Map<String, String> getContactTypeSegmentProperties
(int k)
```

This method is similar to `getContactTypeProperties (int)`, for contact type segments instead of contact types.

Parameter

`k` the segment identifier.

Returns the map of properties.

```
public Map<String, String> getInContactTypeSegmentProperties
(int k)
```

This method is similar to `getContactTypeProperties (int)`, for inbound contact type segments instead of contact types.

Parameter

`k` the segment identifier.

Returns the map of properties.

```
public Map<String, String> getOutContactTypeSegmentProperties
(int k)
```

This method is similar to `getContactTypeProperties (int)`, for outbound contact type segments instead of contact types.

Parameter

`k` the segment identifier.

Returns the map of properties.

```
public Map<String, String> getAgentGroupSegmentProperties
(int i)
```

This method is similar to `getContactTypeProperties (int)`, for agent group segments instead of contact types.

Parameter

`i` the segment identifier.

Returns the map of properties.

```
public Map<String, String> getWaitingQueueSegmentProperties
(int q)
```

This method is similar to `getContactTypeProperties (int)`, for waiting queue segments instead of contact types.

Parameter

`q` the segment identifier.

Returns the map of properties.

`public int getNumMatricesOfAWT()`

Returns the number of matrices containing acceptable waiting times, for estimating service levels. Usually, this returns 1.

Returns the number of matrices of acceptable waiting times.

`public TimeUnit getDefaultUnit()`

Returns the time unit in which output performance measures representing times are expressed. If this method returns `null`, performance measures representing time are output as any other performance measures; no time conversion can be performed or time unit displayed.

Returns the default time unit.

ContactCenterEval

Represents a system evaluating some performance measures of a contact center. An *evaluation* is a process estimating some performance measures according to user-defined parameters. It can be performed by approximation formulas or by simulations while parameters can be obtained from any data source, e.g., files, programs, etc.

This interface represents an evaluation system adapted for contact centers. Each system uses internal, implementation-specific parameters usually read from files and stored into dedicated objects, e.g., `CallCenterParams`. It also defines some external parameters called *evaluation options* which can be modified through this interface, by using the `setEvalOption (EvalOptionType, Object)` method. Evaluations are triggered by using the `eval()` method, and matrices of performance measures are accessible through the `getPerformanceMeasure (PerformanceMeasureType)` method.

The methods `setEvalOption (EvalOptionType, Object)` and `getPerformanceMeasure (PerformanceMeasureType)` are generic methods accepting an argument indicating the evaluation option or group of performance measures of interest, respectively. This design allows the interface to be extended in the future, by adding new options or groups of measures, without affecting current implementations.

The construction of the contact center and the supported evaluation options and performance measures are specific to the implementation of this interface, but the access to the evaluation mechanism is unified. A contact center evaluation system is not forced to support all of the defined performance measures and evaluation options, because the interface specifies facilities to enumerate supported groups of performance measures and evaluation options, and to test the support of a specific measure or option.

```
package umontreal.iro.lecuyer.contactcenters.app;

public interface ContactCenterEval extends ContactCenterInfo
```

Methods

`public EvalOptionType[] getEvalOptions()`

Returns the array of the evaluation option types supported by the implementing object. The evaluation options are the variable parameters of the contact center which can be changed between calls to `eval()`. This should never return a `null` pointer; if no evaluation options are supported for some reasons, this should return an array with length 0.

Returns the array of supported evaluation options.

`public boolean hasEvalOption (EvalOptionType option)`

Determines if the evaluation option `option` is supported by the implemented system. It should return `true` if and only if the `option` is in the array returned by `getEvalOptions()`. Otherwise, it returns `false`.

Parameter

`option` the queried evaluation option.

Returns the support status of the option.

Throws

`NullPointerException` if `option` is `null`.

`public Object getEvalOption (EvalOptionType option)`

Returns the current value of the evaluation option `option`. The class of the returned object must be assignable to the class given by `EvalOptionType.getType()`. If the option is not supported, this should throw a `NoSuchElementException`. This exception can be thrown if and only if `hasEvalOption (EvalOptionType)` returns `false` for the given `option`.

Parameter

`option` the queried evaluation option.

Returns the current value of the option.

Throws

`NoSuchElementException` if the option is not available.

`NullPointerException` if `option` is `null`.

`public void setEvalOption (EvalOptionType option, Object value)`

Sets the evaluation option `option` to `value`. If the given option is not supported, this throws a `NoSuchElementException`. If the class of the given value is incompatible, this throws a `ClassCastException`. If the evaluation option cannot be changed, this throws an `UnsupportedOperationException`.

Parameters

`option` the option to be set.

`value` the new value of the option.

Throws

`NoSuchElementException` if the given option is not supported.

`ClassCastException` if the given value is from an incompatible class.

`IllegalArgumentException` if the given value is invalid.

`UnsupportedOperationException` if the evaluation option is read-only.

`NullPointerException` if the option is `null` or the value is unexpectedly `null`.

`public void eval()`

Evaluates the contact center's supported performance measures using the current values of implementation-specific parameters and evaluation options. If parameters or seeds of random streams are not changed, multiple calls to this method should return the same results. In consequence of this, simulation should reset the random number streams before returning. This method can throw an `IllegalStateException` if there is an inconsistency in the system's parameters. It can throw an `ArithmeticException` if, for the given parameters, the performance measures cannot be estimated.

Throws

`IllegalStateException` if there are invalid parameter values.

`ArithmeticException` if no solution can be computed.

`public boolean seemsUnstable()`

Determines if the system seems to be unstable. When a system is unstable, the returned steady-state performance measures are not reliable. This method mainly applies for stationary simulators which return `true` when the system appears to be unstable, i.e., some waiting queues grow up with simulation time. The method must throw an `IllegalStateException` if it is called before `eval()` and always return `false` if no stability check applies.

Returns the result of the stability check.

Throws

`IllegalStateException` if this method is called before the evaluation is performed.

`public Map<String, Object> getEvalInfo()`

Represents information about this evaluation system that should be included in any report produced by `formatStatistics()`. The information is organized as (key, value) pairs in a map. This information may include steps of an approximation, number of iterations, etc. One can modify the returned map to add custom information. The content of this map should not affect the process of the evaluation; it is used only for building the statistical report. One can use evaluation options for system parameters.

Returns the evaluation information.

`public String formatStatistics()`

Formats information about every performance measure after `eval()` is called. It can be simulation statistics, information about the steps of an approximation algorithm, or simply the values of all performance measures. This method should call `getEvalInfo()` to obtain general information about the evaluation and incorporate the information into the returned string. For each entry in the map, the method should add a `key: value` line in the string. Then, the method appends the performance measures to the returned string. The `PerformanceMeasureFormatText` class can be used to convert matrices of performance measures into strings. If the evaluation was not triggered by calling `eval()` before this method is called, an `IllegalStateException` is thrown. If no statistical information is available even after the evaluation, this method should return an empty string instead of throwing an exception.

Returns a string containing a statistical report.

Throws

`IllegalStateException` if no statistical information is available because the evaluation was not triggered.

`public String formatStatisticsLaTeX()`

Formats and returns a statistical report that can be included into a LaTeX document. This is similar to `formatStatistics()`, except the generated report is in LaTeX rather than plain text.

Returns the formatted report.

`public boolean formatStatisticsExcel (WritableWorkbook wb)`

Constructs and returns an JExcel API workbook containing the results of the evaluation, and appends the contents of the generated report to the workbook `wb`. This method may add multiple sheets, e.g., for general and detailed information. This method should add the information in the map returned by `getEvalInfo()` to a sheet in the workbook. This method returns `true` if and only if the given workbook was modified.

One can then customize the returned workbook as needed. The method `WritableWorkbook.write()` can be used to export the workbook to an output stream. This can be used to create files that can be opened directly by Microsoft Excel for results analysis and reporting. Excel documents can also be opened by open source software such as OpenOffice.org, KOffice, etc.

Parameter

`wb` the workbook to append report to.

Returns `true` if and only if the given workbook was modified.

`public ReportParams getReportParams()`

Returns the parameters for reports formatted by `formatStatistics()`, or `formatStatistics-Excel (WritableWorkbook)`. If no object containing report parameters is available, this method should create a new one using the default constructor of `ReportParams`.

Returns the printed statistics.

`public void setReportParams (ReportParams reportParams)`

Sets the report parameters to `reportParams`.

Parameter

`reportParams` the report parameters..

See also `getReportParams()`

`public PerformanceMeasureType[] getPerformanceMeasures()`

Returns an array containing all the groups of performance measures this object can estimate. If no performance measure is supported by a given implementation, this method should return an array with length 0 instead of `null`.

Returns the array of groups of performance measures.

`public boolean hasPerformanceMeasure (PerformanceMeasureType m)`

Determines if the evaluation system can estimate performance measures in group `m`. This should return `true` if and only if the group of performance measures is listed in the array returned by `getPerformanceMeasures()`.

Parameter

`m` the group of performance measures being tested.

Returns a `true` value if the measure is supported, `false` otherwise.

Throws

`NullPointerException` if `m` is `null`.

`public DoubleMatrix2D getPerformanceMeasure (PerformanceMeasureType m)`

Returns the matrix of values corresponding to the group of performance measures `m` estimated by the last call to `eval()`. The dimensions of the matrix and the role of its elements depend on the queried group of performance measures, and the capabilities of the implementing evaluation system. See the `PerformanceMeasureType` class for more information about the defined performance measures. If the queried measure is not supported by this evaluation object, this throws a `NoSuchElementException`. If the values of the measures are not available, e.g., the `eval()` method was never called after the last call to `reset()`, this throws an `IllegalStateException`.

Parameter

`m` the queried group of performance measures.

Returns the matrix of values computed by the evaluation system.

Throws

`NoSuchElementException` if the given performance measure is not supported by this object.

`IllegalStateException` if the values are not available.

`NullPointerException` if `m` is `null`.

`public void reset()`

Resets this contact center evaluation system for new parameters. Every cached or processed parameter should be reread from the parameter objects. In the case of the simulation, one should try to preserve random number seeds whenever possible, even if the contact center needs to be reconstructed. Some implementatios of this interface provide specialized reset methods allowing to change the associated parameter objects.

`public boolean isVerbose()`

Determines if the implementation should print information during the evaluation of the performance measures. The information depends on the type of evaluation method being involved. It can include steps or iterations of an approximation algorithm, or information about important elements of a simulation. Summary information should be printed on the standard output while debugging data, e.g., traces of every event in a simulation, should be sent to log files if they are generated. By default, the verbose mode is disabled.

Returns `true` if the implementation is in verbose mode, `false` otherwise (the default).

`public void setVerbose (boolean v)`

Sets the verbose status to `v`. If `v` is `true`, verbose mode is enabled. Otherwise, it is disabled.

Parameter

`v` `true` to activate verbose mode, `false` to disable it.

See also `isVerbose()`

ContactCenterSim

Represents a simulation-based evaluation system adapted for contact centers. Simulation uses some `RandomStream` instances to generate random values, schedules events, and generates samples of observations which are used to estimate performance measures. When simulation is used, the `ContactCenterEval.getPerformanceMeasure (PerformanceMeasureType)` method, defined in `ContactCenterEval`, returns matrices of averages. This interface extends the `ContactCenterEval` interface to provide methods for obtaining matrices of sample variances, minima, maxima, and confidence intervals. If, for a group of measures `m`, `ContactCenterEval.getPerformanceMeasure (PerformanceMeasureType)` returns an $a \times b$ matrix, each of the methods of this interface must return a matrix with the same dimensions if called with `m`.

```
package umontreal.iro.lecuyer.contactcenters.app;

public interface ContactCenterSim extends ContactCenterEval
```

Methods

`public void eval()`

Performs a simulation to evaluate the performance measures. Unless `getAutoResetStartStream()` returns `false`, if `eval()` is called multiple times without changing system parameters, `ContactCenterEval.getPerformanceMeasure (PerformanceMeasureType)` should return the same matrices of estimates after each call. This requires that random streams used for simulation be reset after each evaluation. Thus, before returning, this method should use `RandomStream.resetStartSubstream()` on all random streams in order to reset the seeds. It is also recommended to always use `RandomStream.resetNextSubstream()` for all random streams after any replication to improve synchronization of random streams.

`public void newSeeds()`

Changes the seeds of the random number generators used during the simulation. When calling `eval()` multiple times to perform a simulation, the results should be identical for the same values of parameters. If one requires the simulation to be performed with new random seeds, the random streams need to be reset. This can be done by calling `RandomStream.resetNextSubstream()` method on each `RandomStream` object associated with the simulator, or by creating new random streams.

`public DoubleMatrix2D getVariance (PerformanceMeasureType m)`

Returns a matrix of sample variances for the group of performance measures `m`. If the group of performance measures is not supported, or the sample variance cannot be computed, this method throws a `NoSuchElementException`.

Parameter

`m` the queried group of performance measures.

Returns the matrix of sample variances.

Throws

`NoSuchElementException` if the given group of performance measures is not supported, or the sample variance cannot be computed.

`IllegalStateException` if the values are not available.

`NullPointerException` if `m` is `null`.

`public DoubleMatrix2D getMin (PerformanceMeasureType m)`

Returns a matrix of minimum values for the group of performance measures `m`. If the group of measures defines no minimum (e.g., a ratio of expectations), or it is not supported, this method throws `NoSuchElementException`.

Parameter

`m` the queried group of performance measures.

Returns the matrix of minima.

Throws

`NoSuchElementException` if the given group of performance measures is not supported, or the minima cannot be computed.

`IllegalStateException` if the values are not available.

`NullPointerException` if `m` is `null`.

`public DoubleMatrix2D getMax (PerformanceMeasureType m)`

Returns a matrix of maximum values for the performance measure `m`. If the group of measures defines no maximum (e.g., a ratio of expectations), or if it is not supported, this method throws `NoSuchElementException`.

Parameter

`m` the queried group of performance measures.

Returns the matrix of maxima.

Throws

`NoSuchElementException` if the given group of performance measures is not supported, or the maxima cannot be computed.

`IllegalStateException` if the values are not available.

`NullPointerException` if `m` is `null`.

```
public DoubleMatrix2D[] getConfidenceInterval (PerformanceMeasureType m,
                                               double level)
```

Returns confidence intervals on the means or ratios of means, for the group of performance measures `m`, with confidence level `level`. This must return an array of two matrices, the first containing the lower bound values and the second, the upper bound values. For an unbounded confidence interval, one of the two matrices can be `null`. For each element of the performance measure matrix, a confidence interval whose desired coverage probability is `level` must be computed, independently of the other elements in the matrix. As a result, the coverage probability of all computed intervals will be smaller than `level`. The way each interval is computed is implementation-specific.

Parameters

`m` the queried group of performance measures.

`level` desired probability that, for a given performance measure, the (random) confidence interval covers the true mean (a constant).

Returns an array of two matrices containing lower and upper bounds of the confidence intervals.

Throws

`NoSuchElementException` if the given group of performance measures is not supported or the confidence interval cannot be computed.

`IllegalStateException` if the values are not available.

`NullPointerException` if `m` is `null`.

```
public double getConfidenceLevel()
```

Returns the confidence level of the intervals output by `ContactCenterEval.formatStatistics()`. The initial confidence level is implementation-specific, and usually set by a constructor.

Returns the level of confidence for the intervals.

```
public void setConfidenceLevel (double level)
```

Sets the level of confidence for the intervals output by `ContactCenterEval.formatStatistics()` to `level`.

Parameter

`level` the level of confidence of the intervals.

Throws

`IllegalArgumentException` if `level` is smaller than or equal to 0, or greater than or equal to 1.

```
public MatrixOfStatProbes<?> getMatrixOfStatProbes (PerformanceMeasureType
                                                    m)
```

Returns the matrix of statistical probes used to manage observations for estimating the performance measures in group `m`. The particular subclass of the statistical probe matrix depends on the performance measure type only. For averages, this method must return a `MatrixOfTallies` object. For functions of multiple averages, e.g., ratios of averages, this must return a `MatrixOfFunctionOfMultipleMeansTallies`.

Parameter

`m` the group of performance measures of interest.

Returns the matrix of statistical probes.

```
public MatrixOfTallies<?> getMatrixOfTallies (PerformanceMeasureType m)
```

Returns the matrix of tallies used to manage observations for estimating the performance measures in group `m`. This matrix is available only for performance measures corresponding to expectations, not functions of expectations.

Parameter

`m` the group of performance measures of interest.

Returns the matrix of tallies.

```
public MatrixOfFunctionOfMultipleMeansTallies<?>
getMatrixOfFunctionOfMultipleMeansTallies (PerformanceMeasureType m)
```

Returns the matrix of function of multiple means tallies used to manage observations for estimating the performance measures in group `m`. This matrix is available only for performance measures corresponding functions of expectations.

Parameter

`m` the group of performance measures of interest.

Returns the matrix of function of multiple means tallies.

```
public int getCompletedSteps()
```

Returns the number of completed steps for the simulation. When using independent replications, a step corresponds to a replication. When using batch means for stationary simulation, this corresponds to the number of terminated batches.

Returns the number of completed steps.

`public boolean getAutoResetStartStream()`

Determines if the random streams are automatically reset at the end of each evaluation. By default, a simulator calls `RandomStream.resetStartStream()` on each `RandomStream` object he has created for `eval()` to use the same seeds if called multiple times. If this option is set to `false`, the streams are not reset automatically, and `eval()` always returns different results when called multiple times. However, `RandomStream.resetNextSubstream()` should still be called for all random streams after each replication.

Returns `true` if streams are reset automatically, `false` otherwise.

`public void setAutoResetStartStream (boolean r)`

Sets the automatic reset start stream indicator to `r`.

Parameter

`r` the new value of the indicator.

See also `getAutoResetStartStream()`

`public boolean getSeqSampEachEval()`

Determines if sequential sampling is done upon each call on `eval()`. If the implemented simulator uses sequential sampling, the number of steps (replications or batches) simulated is random. By default, the first call to `eval()` determines the number of simulated steps while each subsequent call to `eval()` simulates the exact same number of steps, without reapplying sequential sampling. Turning this flag on changes this behavior, forcing the simulator to perform sequential sampling upon every call to `eval()`.

Returns the value of the indicator.

`public void setSeqSampEachEval (boolean seqSamp)`

Sets the indicator for sequential sampling on each eval to `seqSamp`.

Parameter

`seqSamp` the new value of the indicator.

`public void resetStartStream()`

Calls `RandomStream.resetStartStream()` for all random streams used by the simulator.

`public void resetStartSubstream()`

Calls `RandomStream.resetStartSubstream()` for all random streams used by the simulator.

`public void resetNextSubstream()`

Calls `RandomStream.resetNextSubstream()` for all random streams used by the simulator.

ContactCenterSimWithObservations

Represents a contact simulator capable of returning individual observations for performance measures.

```
package umontreal.iro.lecuyer.contactcenters.app;

public interface ContactCenterSimWithObservations extends ContactCenterSim
```

Methods

`public int numberObs (PerformanceMeasureType pm, int row, int column)`

Returns the number of observations available for a performance measure of type `pm`, identified by row `row` and column `column`. If the number of observations is not available for the given performance measure, this method throws a `NoSuchElementException`.

Parameters

`pm` the type of performance measure.

`row` the row of the performance measure.

`column` the column of the performance measure.

Returns the number of observations.

Throws

`NoSuchElementException` if the observations are not available for the given performance measure.

`IndexOutOfBoundsException` if `row` or `column` are out of bounds.

`public double[] getObs (PerformanceMeasureType pm, int row, int column)`

Returns an array containing the observations for a performance measure of type `pm`, identified by row `row` and column `column`. If the observations are not available for the given performance measure, this method throws a `NoSuchElementException`.

Parameters

`pm` the type of performance measure.

`row` the row of the performance measure.

`column` the column of the performance measure.

Returns the array of observations.

Throws

`NoSuchElementException` if the observations are not available for the given performance measure.

`IndexOutOfBoundsException` if `row` or `column` are out of bounds.

ContactCenterSimWithObservationSets

Represents a contact center simulator producing sets of observations for performance measures. The definition of a set of observations depends on the specific simulator; it can correspond to a macro-replication, a stratum, etc.

```
package umontreal.iro.lecuyer.contactcenters.app;

public interface ContactCenterSimWithObservationSets extends ContactCenterSim
```

Methods

`public int getNumObservationSets (PerformanceMeasureType pm, int row, int column)`

Returns the number of sets of observations available for a performance measure of type `pm`, identified by row `row` and column `column`. If the number of sets of observations is not available for the given performance measure, this method throws a `NoSuchElementException`.

Parameters

`pm` the type of performance measure.

`row` the row of the performance measure.

`column` the column of the performance measure.

Returns the number of sets of observations.

Throws

`NoSuchElementException` if the observations are not available for the given performance measure.

`IndexOutOfBoundsException` if `row` or `column` are out of bounds.

`public int numberObs (PerformanceMeasureType pm, int row, int column, int set)`

Returns the number of observations available in the set `set` for a performance measure of type `pm`, identified by row `row` and column `column`. If the number of observations is not available for the given performance measure, this method throws a `NoSuchElementException`.

Parameters

`pm` the type of performance measure.

`row` the row of the performance measure.

`column` the column of the performance measure.

`set` the index of the set of observations.

Returns the number of observations.

Throws

`NoSuchElementException` if the observations are not available for the given performance measure.

`IndexOutOfBoundsException` if `row`, `column`, or `set` are out of bounds.

```
public double[] getObs (PerformanceMeasureType pm, int row, int column,
                        int set)
```

Returns the number of observations available in the set `set` for a performance measure of type `pm`, identified by row `row` and column `column`. If the number of observations is not available for the given performance measure, this method throws a `NoSuchElementException`.

Parameters

`pm` the type of performance measure.

`row` the row of the performance measure.

`column` the column of the performance measure.

`set` the index of the set of observations.

Returns the number of observations.

Throws

`NoSuchElementException` if the observations are not available for the given performance measure.

`IndexOutOfBoundsException` if `row`, `column`, or `set` are out of bounds.

ObservableContactCenterSim

Represents a contact center simulation whose simulation can be observed or stopped. An observer can be registered by using the `addContactCenterSimListener (ContactCenterSimListener)` method to be notified each time a step (replication or batch) of the simulation is done. Moreover, the `abort()` method can be used to stop the simulation before its end. This can be used to implement a user interface allowing the progress of the simulation to be displayed.

```
package umontreal.iro.lecuyer.contactcenters.app;

public interface ObservableContactCenterSim extends ContactCenterSim
```

Methods

`public void addContactCenterSimListener (ContactCenterSimListener l)`

Registers the listener `l` to be notified about the progress of the simulator.

Parameter

`l` the listener to be notified.

Throws

`NullPointerException` if `l` is `null`.

`public void removeContactCenterSimListener (ContactCenterSimListener l)`

Removes the listener `l` from the list of listeners registered with this simulator.

Parameter

`l` the listener being removed.

`public void clearContactCenterSimListeners()`

Removes all the listeners registered with this simulator.

`public List<ContactCenterSimListener> getContactCenterSimListeners ()`

Returns the listeners registered with this simulator.

Returns the list of registered listeners.

`public boolean isAborted()`

Determines if the simulation has been aborted by using the `abort()` method.

Returns `true` if the simulation was aborted, `false` otherwise.

`public void abort()`

Aborts the current simulation.

ContactCenterSimListener

Represents an observer of the progress of a simulation.

```
package umontreal.iro.lecuyer.contactcenters.app;

public interface ContactCenterSimListener
```

Methods

`public void simulationStarted (ObservableContactCenterSim sim, int numTargetSteps)`

Indicates that a new simulation was started by the simulator `sim`, and that it will consist of `numTargetSteps` steps.

Parameters

`sim` the contact center simulator.

`numTargetSteps` the predicted number of steps.

`public void simulationExtended (ObservableContactCenterSim sim, int newNumTargetSteps)`

Indicates that an in-progress simulation performed by the simulator `sim` is extended to consist of `newNumTargetSteps` steps. This occurs when sequential sampling is used to reach a certain percision.

Parameters

`sim` the contact center simulator.

`newNumTargetSteps` the new target number of steps.

`public void simulationStopped (ObservableContactCenterSim sim, boolean aborted)`

Indicates that a simulation performed by `sim` is terminated. If `aborted` is `true`, the simulation was stopped using `ObservableContactCenterSim.abort()`. Otherwise, it has terminated after the target number of steps is reached.

Parameters

`sim` the contact center simulator.

`aborted` `true` if and only if the simulation was aborted.

`public void stepDone (ObservableContactCenterSim sim)`

Indicates that a step was done by the simulator `sim`. One can use `ContactCenterSim.getCompletedSteps()` to obtain the number of completed steps.

Parameter

`sim` the contact center simulator.

ContactCenterProgressBar

Contact center simulation listener displaying a progress bar for the simulation. This listener shows the number of completed steps over the total number of steps to complete before the simulation ends, with a visual progress indicator.

```
package umontreal.iro.lecuyer.contactcenters.app;

public class ContactCenterProgressBar implements ContactCenterSimListener
```

SimStoppingCondition

Represents a simulation stopping condition which is checked before the simulation ends. By default, a simulator performs a minimal number of replications or a single replication with a minimal length to get some statistics and performs some tests to determine the additional simulation time. If an additional stopping condition is added through `ContactCenterEval.setEvalOption (EvalOptionType, Object)`, this condition is checked and the returned result is used instead of the default result.

```
package umontreal.iro.lecuyer.contactcenters.app;

public interface SimStoppingCondition
```

Method

`public int check (ContactCenterSim sim, int newReps)`

Checks the implemented stopping condition and returns the required number of additional batches or replications to simulate. This method must be given the contact center simulator and the number of additional replications or batches to simulate according to the simulator's default stopping condition. This number can be used or ignored and the returned value will be used instead by the simulator.

Parameters

`sim` the contact center simulator.

`newReps` the number of required additional batches or replications, according to the default stopping condition.

Returns the number of new replications or batches, according to the implemented stopping condition.

SearchStoppingCondition

Early stopping condition allowing to perform a first cut when using neighborhood search. When checked, this condition computes a confidence interval on the aggregate value of a given performance measure and the simulation exits when a threshold value δ falls outside the confidence interval with confidence level β. The simulation also stops when a certain number of batches or replications is reached or when the default stopping condition of the simulator is satisfied. If the stopping condition fails, only one additional batch or replication is performed before the condition is checked again.

```
package umontreal.iro.lecuyer.contactcenters.app;

public class SearchStoppingCondition implements SimStoppingCondition
```

Constructor

```
public SearchStoppingCondition (double beta, double delta,
                                PerformanceMeasureType pm, int maxReps)
```

Constructs a new search stopping condition with confidence level β, threshold value δ, on performance measure type `pm` and with a maximal number of replications or batches `maxReps`.

Parameters

`beta` the confidence level of the confidence intervals.

`delta` the threshold value.

`pm` the target performance measure.

`maxReps` the maximal number of replications or batches.

Throws

`IllegalArgumentException` if β is not in $(0, 1)$ or `maxReps` is negative.

`NullPointerException` if `pm` is `null`.

Methods

```
public double getBeta()
```

Returns the β confidence level.

Returns the confidence level.

```
public void setBeta (double beta)
```

Sets the β confidence level to `beta`.

Parameter

`beta` the new confidence level.

Throws

`IllegalArgumentException` if `beta` is not in $(0, 1)$.

`public double getDelta()`

Returns the treshold value δ.

Returns the threshold value.

`public void setDelta (double delta)`

Sets the treshold value δ to `delta`.

Parameter

`delta` the new threshold value.

`public int getMaxReplications()`

Returns the maximal number of replications or batches to simulate if the stopping condition does not apply.

Returns the maximal number of replications or batches.

`public void setMaxReplications (int maxReps)`

Sets the maximal number of replications or batches to `maxReps`.

Parameter

`maxReps` the new maximal number of replications or batches.

Throws

`IllegalArgumentException` if `maxReps` is negative.

`public PerformanceMeasureType getPerformanceMeasureType ()`

Returns the checked performance measure.

Returns the checked performance measure.

`public void setPerformanceMeasureType (PerformanceMeasureType pm)`

Sets the checked performance measure to `pm`.

Parameter

`pm` the new checked performance measure.

Throws

`NullPointerException` if `pm` is `null`.

PerformanceMeasureType

Represents types of performance measures for contact centers. A performance measure estimated by approximation formulas or simulation can be described by a type, an index, and a time interval. The type might be, for example, `SERVICELEVEL`, while the index might represent a group of contact types called a segment. All statistics concerning a given type of performance measure are regrouped into a matrix with rows corresponding to the index, and columns generally matching the time intervals. See `RowType` and `ColumnType` for the possible types of rows and columns in matrices of statistics. Statistics can be point estimators, minima, maxima, variances, or confidence intervals. Point estimators can be computed, depending on the type of performance measure, using averages, functions of averages, averages of functions, or raw statistics. See `EstimationType` for the possible types of point estimators.

Constants of this enum are used to select a group of measures when obtaining a matrix of results from an evaluation system. This enum defines groups of performance measures, and provides facilities to format results. It does not calculate any matrix of statistics.

Table 1 presents a typical matrix of performance measures whose rows correspond to segments of contact types, and columns to segments of main periods. The upper left part of the table regroups the performance measures concerning specific contact types, and specific main periods. The lower part of the table regroups performance measures concerning segments of several contact types. This lower part appears in matrices of performance measures if $K > 1$, and contains several rows only if segments of contact types are defined by the user. However, an implicit segment regrouping all contact types always appears provided that $K > 1$.

In a similar way, the right part of the table regroups performance measures concerning segments regrouping several main periods. These segments, which are time intervals too, can be used, e.g., to get statistics for the morning, the afternoon, the evening, a day of a week, etc. In a similar way to the lower part, the right part of the table shows up only if $P > 1$, and an implicit segment regrouping all main periods is always displayed. Note that the bottom right element of the matrix corresponds to the performance measure concerning all contact types and main periods.

Segments can also be defined to regroup inbound and outbound contact types, and agent groups. A segment of inbound contact types affects only matrices of performance measures concerning inbound contact types, e.g., `SERVICELEVEL`. Similarly, a segment of outbound contact types affects only matrices of performance measures concerning outbound types, e.g., `RATEOFTRIEDOUTBOUND`.

Many types of performance measures we now describe correspond to the expected number of calls counted in a time interval $[t_1, t_2]$ meeting a certain condition, e.g., served calls. By default, a call is counted in a time interval if it arrives during that interval. But using the `perPeriodCollectingMode` attribute of simulation parameters, this can be changed, e.g., to count a call if it ends its service or abandons during the interval.

Table 1: Example of a matrix of performance measures

	Main periods					Segments of main periods		
	$X_{0,0}$	$\cdots$	$X_{0,p}$	$\cdots$	$X_{0,P-1}$	$X_{0,P}$	$\cdots$	$X_{0,\cdot}$
	$\vdots$	$\ddots$	$\vdots$	$\ddots$		$\vdots$	$\ddots$	
Contact types	$X_{k,0}$	$\cdots$	$X_{k,p}$	$\cdots$	$X_{k,P-1}$	$X_{k,P}$	$\cdots$	$X_{k,\cdot}$
	$\vdots$	$\ddots$	$\vdots$	$\ddots$		$\vdots$	$\ddots$	
	$X_{K-1,0}$	$\cdots$	$X_{K-1,p}$	$\cdots$	$X_{K-1,P-1}$	$X_{K-1,P}$	$\cdots$	$X_{K-1,\cdot}$
Segments of	$X_{K,0}$	$\cdots$	$X_{K,p}$	$\cdots$	$X_{K,P-1}$	$X_{K,P}$	$\cdots$	$X_{K,\cdot}$
contact types	$\vdots$	$\ddots$	$\vdots$	$\ddots$		$\vdots$	$\ddots$	
	$X_{\cdot,0}$	$\cdots$	$X_{\cdot,p}$	$\cdots$	$X_{\cdot,P-1}$	$X_{\cdot,P}$	$\cdots$	X

```
package umontreal.iro.lecuyer.contactcenters.app;

public enum PerformanceMeasureType
```

Constants

`ABANDONMENTRATIO`

Probability of abandonment, i.e., the fraction of the expected number of contacts having left the system without service over the total expected number of arrivals.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`ABANDONMENTRATIOAFTERAWT`

Probability of abandonment after the acceptable waiting time. This corresponds to the fraction of the expected number of contacts having left the system without service and after a waiting time greater than or equal to the acceptable waiting time, over the total expected number of arrivals.

Row type INBOUNDTYPEAWT

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`ABANDONMENTRATIOBEFOREAWT`

Probability of abandonment before the acceptable waiting time. This corresponds to the fraction of the expected number of contacts having left the system without service and waiting at most for the acceptable waiting time, over the total expected number of arrivals.

Row type INBOUNDTYPEAWT

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`ABANDONMENTRATIOREP`

Corresponds to the expectation of ratio version of `ABANDONMENTRATIO`.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATIONOFFUNCTION

`AVGBUSYAGENTS`

Expected time-average number of busy agents over the simulation time, for each agent group and period. More specifically, if $N_{\mathrm{B}}(t)$ is the number of busy agents at time t, for a time interval $[t_1, t_2]$, the performance measure is given by

$$\frac{1}{t_2 - t_1}\mathbb{E}\left[\int_{t_1}^{t_2} N_{\mathrm{B}}(t)dt\right].$$

Row type AGENTGROUP

Column type MAINPERIOD

Estimation type EXPECTATION

`AVGQUEUESIZE`

Represents the expected time-average queue size for each waiting queue. This measure corresponds to the integral of the queue size over simulation time whereas `MAXQUEUESIZE` gives the maximal observed queue size. More specifically, if $Q(t)$ is the queue size at time t, for any time interval $[t_1, t_2]$, the performance measure is given by

$$\frac{1}{t_2 - t_1}\mathbb{E}\left[\int_{t_1}^{t_2} Q(t)dt\right].$$

Row type WAITINGQUEUE

Column type MAINPERIOD

Estimation type EXPECTATION

`AVGSCHEDULEDAGENTS`

Represents the expected time-average number of scheduled agents over the simulation time, for each agent group and period. This includes the busy and idle agents (available or not), as well as the ghost agents, i.e., agents finishing the service of contacts before leaving. More specifically, if $N(t)$ is the number of agents scheduled at time t, and $N_{\mathrm{G}}(t)$ is the number of extra ghost agents, for a time interval $[t_1, t_2]$, the performance measure is given by

$$\frac{1}{t_2 - t_1}\mathbb{E}\left[\int_{t_1}^{t_2}(N(t) + N_{\mathrm{G}}(t))dt\right].$$

As $N(t)$ is set according to the staffing given by the user, it is constant during main periods, and the above quantity is random only because of $N_{\mathrm{G}}(t)$. Moreover, because of the ghost agents, if this performance measure is estimated for a specific main period, the obtained estimate will often be higher than the input staffing for the same period.

Also note that this performance measure on the whole horizon does not correspond to the mean number of full-time equivalents (FTE). To get the FTE, one should multiply the time-average number of agents by $(t_P - t_0)/h$ where t_0 and t_P are the starting and ending times of the main periods, and h is the duration of an average working day for agents. Of course, every one of these quantities must be expressed in the same time unit to get a valid ratio.

Row type AGENTGROUP

Column type MAINPERIOD

Estimation type EXPECTATION

`AVGWORKINGAGENTS`

Represents the expected time-average number of working agents over the simulation time, for each agent group and period. This is similar to `AVGSCHEDULEDAGENTS` but excludes the non-available idle agents. More specifically, if $N_{\mathrm{B}}(t)$ is the number of busy agents at time t, and $N_{\mathrm{F}}(t)$ is the number of idle but available agents, for a time interval $[t_1, t_2]$, the performance measure is given by

$$\frac{1}{t_2 - t_1}\mathbb{E}\left[\int_{t_1}^{t_2}(N_{\mathrm{B}}(t) + N_{\mathrm{F}}(t))dt\right].$$

If agents cannot become unavailable, e.g., by disconnecting temporarily after service terminations, the two performance measures are identical.

Row type AGENTGROUP

Column type MAINPERIOD

Estimation type EXPECTATION

`BLOCKRATIO`

Probability of blocking, i.e., the fraction of the expected number of blocked contacts over the total expected number of arrivals.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`BLOCKRATIOREP`

Corresponds to the expectation of ratio version of `BLOCKRATIO`.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATIONOFFUNCTION

`BUSYAGENTSENDSIM`

Number of busy agents at the end of the simulation. When the simulation horizon is finite, this should always be 0.

Row type AGENTGROUP

Column type SINGLECOLUMN

Estimation type RAWSTATISTIC

`DELAYRATIO`

Probability of delay, i.e., the fraction of the expected number of contacts not served immediately over the total expected number of arrivals.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`DELAYRATIOREP`

Corresponds to the expectation of ratio version of `DELAYRATIO`.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATIONOFFUNCTION

`EXCESSTIME`

Average excess time performance measure. This corresponds to the expected sum of excess times for all contacts over the expected number of arrivals. Let $A(t_2, t_2)$ be the total number of calls counted during interval $[t_1, t_2]$ and W_i the waiting time of the ith contact counted during the interval, and s the acceptable waiting time. The average excess time is

$$\frac{\mathbb{E}\left[\sum_{i=0}^{A(t_1,t_2)-1}(W_i - s)^+\right]}{\mathbb{E}[A(t_1, t_2)]}.$$

The numerator of the ratio corresponds to `SUMEXCESSTIMES`, while the denominator corresponds to `RATEOFARRIVALS`.

Row type INBOUNDTYPEAWT

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`EXCESSTIMEABANDONED`

Average excess time performance measure for contacts having abandoned. This corresponds to the expected total excess time for contacts having abandoned over the expected number of abandoned contacts. Let $L(t_1, t_2)$ be the number of contacts counted during time interval $[t_1, t_2]$ and having abandoned, and W_i the waiting time of the ith contact counted during $[t_1, t_2]$, and s the acceptable waiting time. The average excess time is

$$\frac{\mathbb{E}\left[\sum_{i=0}^{L(t_1,t_2)-1}(W_i - s)^+\mathbb{I}[\text{Call } i \text{ abandoned}]\right]}{\mathbb{E}[L(t_1, t_2)]}.$$

The numerator of the ratio corresponds to `SUMEXCESSTIMESABANDONED`, while the denominator corresponds to `RATEOFABANDONMENT`.

Row type INBOUNDTYPEAWT

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`EXCESSTIMEABANDONEDREP`

Expectation of ratio version of `EXCESSTIMEABANDONED`.

Row type INBOUNDTYPEAWT

Column type MAINPERIOD

Estimation type EXPECTATIONOFFUNCTION

`EXCESSTIMEREP`

Expectation of ratio version of `EXCESSTIME`.

Row type INBOUNDTYPEAWT

Column type MAINPERIOD

Estimation type EXPECTATIONOFFUNCTION

`EXCESSTIMESERVED`

Average excess time performance measure for served contacts. This corresponds to the expected total excess time for contacts having been served over the expected number of served contacts. Let $S(t_1, t_2)$ be the number of served contacts counted during interval $[t_1, t_2]$ and W_i the waiting time of the ith contact counted during $[t_1, t_2]$, and s the acceptable waiting time. The average excess time is

$$\frac{\mathbb{E}\left[\sum_{i=0}^{S(t_1,t_2)-1}(W_i - s)^+ \mathbb{I}[\text{Call } i \text{ served}]\right]}{\mathbb{E}[S(t_1, t_2)]}.$$

The numerator of the ratio corresponds to `SUMEXCESSTIMESSERVED`, while the denominator corresponds to `RATEOFSERVICES`.

Row type INBOUNDTYPEAWT

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`EXCESSTIMESERVEDREP`

Expectation of ratio version of `EXCESSTIMESERVED`.

Row type INBOUNDTYPEAWT

Column type MAINPERIOD

Estimation type EXPECTATIONOFFUNCTION

`MAXBUSYAGENTS`

Represents the expected maximal number of busy agents observed for a set of agent groups. This expectation often corresponds to the number of scheduled agents, because for most models, all agents are busy at some times. However, the maximal number of busy agents may be smaller than the number of agents if too many agents were planned. If the expectation is estimated by an average of observations, taking the maximum of these observations gives the maximal number of busy agents over all the simulation.

Row type AGENTGROUP

Column type MAINPERIOD

Estimation type EXPECTATION

`MAXQUEUESIZE`

Represents the expected maximal size observed for a waiting queue. If the expectation is estimated by an average of observations, taking the maximum of these observations gives the maximal queue size observed during all the simulation.

Row type WAITINGQUEUE

Column type MAINPERIOD

Estimation type EXPECTATION

`MAXWAITINGTIME`

Represents the expected maximal waiting time observed for a set of contact types. This performance measure can be defined as follows for a specific contact type. Let W_k be the (random) waiting time for a contact of type k. The maximal waiting time for contacts of type k during the simulated horizon is $\max(W_k)$ while the performance measure is $\mathbb{E}[\max(W_k)]$. In a similar way, we can define the measure for all contacts. For this, let W be the waiting time for a contact of any type. The performance measure is then $\mathbb{E}[\max(W)]$. Note that although $\max(W) = \max(W_1, \ldots, W_K)$, in general,

$$\mathbb{E}[\max(W)] \neq \max(\mathbb{E}[W_1], \ldots, \mathbb{E}[W_K]).$$

The performance can be defined similarly for specific time intervals.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATION

`MAXWAITINGTIMEG`

Same as `MAXWAITINGTIME`, for (contact type, agent group) pairs.

Row type CONTACTTYPEAGENTGROUP

Column type MAINPERIOD

Estimation type EXPECTATION

`MAXWAITINGTIMEABANDONED`

Represents the expected maximal waiting time of contacts having abandoned, for each contact type and period.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATION

`MAXWAITINGTIMESERVED`

Represents the maximal expected waiting time of served contacts, for each contact type and period.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATION

`MAXWAITINGTIMESERVEDG`

Represents the maximal expected waiting time of served contacts, for each (contact type, agent group) pair and period.

Row type CONTACTTYPEAGENTGROUP

Column type MAINPERIOD

Estimation type EXPECTATION

`OCCUPANCY`

Agents' occupancy ratio. Defined as the expected number of busy agents over the expected total number of scheduled agents, over the simulation time. The expectation at the numerator corresponds to the `AVGBUSYAGENTS` type of performance measure while the expectation at the denominator corresponds to `AVGSCHEDULEDAGENTS`.

Row type AGENTGROUP

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`OCCUPANCY2`

Alternate agents' occupancy ratio. Defined as the expected number of busy agents over the expected total number of working agents, over the simulation time. This differs from `OCCUPANCY` only when agents are allowed to disconnect after services. The expectation at the numerator corresponds to the `AVGBUSYAGENTS` type of performance measure while the expectation at the denominator corresponds to `AVGWORKINGAGENTS`.

Row type AGENTGROUP

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`OCCUPANCY2REP`

Corresponds to the expectation of ratio version of `OCCUPANCY2`.

Row type AGENTGROUP

Column type MAINPERIOD

Estimation type EXPECTATIONOFFUNCTION

`OCCUPANCYREP`

Corresponds to the expectation of ratio version of `OCCUPANCY`.

Row type AGENTGROUP

Column type MAINPERIOD

Estimation type EXPECTATIONOFFUNCTION

`QUEUESIZEENDSIM`

Gives the queue size at the end of the simulation. This quantity should be 0 for simulations over a finite horizon, since the waiting queues are emptied at the end of each replication.

Row type WAITINGQUEUE

Column type SINGLECOLUMN

Estimation type RAWSTATISTIC

`RATEOFABANDONMENT`

Corresponds to the rate of contacts of each type having abandoned, excluding contacts blocked because of insufficient queue capacity.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATION

`RATEOFABANDONMENTAFTERAWT`

Corresponds to the rate of contacts of each inbound type having waited more than the acceptable waiting time, before they abandon.

Row type INBOUNDTYPEAWT

Column type MAINPERIOD

Estimation type EXPECTATION

`RATEOFABANDONMENTBEFOREAWT`

Corresponds to the rate of contacts of each inbound type having waited less than the acceptable waiting time, before they abandon.

Row type INBOUNDTYPEAWT

Column type MAINPERIOD

Estimation type EXPECTATION

`RATEOFARRIVALS`

Defined as the rate of contacts arriving into the router for being assigned an agent. This includes blocked and served contacts, as well as contacts having abandoned. For inbound contacts, the arrival rate can be computed easily from the input data, except for call types corresponding to transfer targets. For outbound contacts, this corresponds to the rate of right party connects during the simulation.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATION

`RATEOFARRIVALSIN`

Same as `RATEOFARRIVALS`, for inbound contacts only.

`RATEOFBLOCKING`

Corresponds to the rate of contacts blocked because the queue capacity was exceeded at the time of their arrivals.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATION

`RATEOFDELAY`

Corresponds to the rate of delayed contacts, i.e., the rate of contacts not served immediately upon arrival. Since blocked contacts would have to wait if they were not blocked, they are counted as positive waits too. For outbound contacts, this corresponds to mismatches.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATION

`RATEOFINTARGETSL`

Corresponds to the rate of served or abandoned inbound contacts of each type having waited less than the acceptable waiting time. This corresponds to the sum of performance measures `RATEOFABANDONMENTBEFOREAWT` and `RATEOFSERVICESBEFOREAWT`.

Row type INBOUNDTYPEAWT

Column type MAINPERIOD

Estimation type EXPECTATION

`RATEOFOFFERED`

Defined as the rate of contacts offered. This includes served contacts as well as contacts still in queue after the end of experiment or having abandoned, but this excludes blocked contacts. For outbound contacts, this corresponds to the rate of right party connects during the simulation. When the total queue capacity is infinite, this corresponds to the number of arrivals `RATEOFARRIVALS`.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATION

`RATEOFSERVICES`

Represents the rate of served contacts for each contact type and period.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATION

`RATEOFSERVICESAFTERAWT`

Corresponds to the rate of served inbound contacts of each type having waited more than the acceptable waiting time.

Row type INBOUNDTYPEAWT

Column type MAINPERIOD

Estimation type EXPECTATION

`RATEOFSERVICESBEFOREAWT`

Corresponds to the rate of served inbound contacts of each type having waited less than the acceptable waiting time.

Row type INBOUNDTYPEAWT

Column type MAINPERIOD

Estimation type EXPECTATION

`RATEOFSERVICESG`

Represents the rate of served contacts for each contact type, agent group, and period. This is similar to `RATEOFSERVICES`, but this gives the rate at which each agent group serves contacts of each type.

Row type CONTACTTYPEAGENTGROUP

Column type MAINPERIOD

Estimation type EXPECTATION

`RATEOFTRIEDOUTBOUND`

Defined as the rate of contacts of each outbound type the dialer or agents have tried to make. This includes the number of reached (arrived) contacts as well as the number of failed contacts.

Row type OUTBOUNDTYPE

Column type MAINPERIOD

Estimation type EXPECTATION

`RATEOFWRONGPARTYCONNECT`

Defined as the rate of contacts of each outbound type the dialer or agents have tried to make, and for which the wrong party was reached.

Row type OUTBOUNDTYPE

Column type MAINPERIOD

Estimation type EXPECTATION

`SERVEDRATES`

Represents the rate of contacts of a given type served by agents in a specific group, per simulation time unit. The element (k, i) of a served rates matrix corresponds to the rate of served contacts of type k by agents in the group i during one simulation time unit. Column i of the last row corresponds to the total number of served contacts by agents in the group i, per simulation time unit. Row k of the last column represents the total number of contacts with type k served by any agent, per simulation time unit.

This performance measure is similar to `RATEOFSERVICESG`, except that it is estimated only globally, not for each main period, with less memory than `RATEOFSERVICESG`.

Row type CONTACTTYPE

Column type AGENTGROUP

Estimation type EXPECTATION

`SERVICELEVEL`

Service level performance measure. Let $S_{\mathrm{G}}(s, t_1, t_2)$ be the number of contacts counted during interval $[t_1, t_2]$, and served after a waiting time less than or equal to the acceptable waiting time s, and $S(t_1, t_2)$ be the total number of served contacts counted during $[t_1, t_2]$. Let $L_{\mathrm{G}}(s, t_1, t_2)$ be the number of contacts counted during interval $[t_1, t_2]$ having abandoned after a waiting time smaller than or equal to the acceptable waiting time, and $A(t_1, t_2)$ be the total number of contacts counted in the $[t_1, t_2]$ interval. The service level is defined by

$$g_1(s, t_1, t_2) = \mathbb{E}[S_{\mathrm{G}}(s, t_1, t_2)]/\mathbb{E}[A(t_1, t_2) - L_{\mathrm{G}}(s, t_1, t_2)].$$

NOTE: since this performance measure is of type `FUNCTIONOFEXPECTATIONS`, the complete list of observations generated by the simulator are not available directly; instead, one must use the performance measure `SERVICELEVELREP`.

Row type INBOUNDTYPEAWT

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`SERVICELEVELREP`

Represents the expectation of ratio version of `SERVICELEVEL`.

Row type INBOUNDTYPEAWT

Column type MAINPERIOD

Estimation type EXPECTATIONOFFUNCTION

`SERVICELEVEL2`

Alternate service level performance measure. This service level is defined as

$$g_2(s, t_1, t_2) = \mathbb{E}[S_{\mathrm{G}}(s, t_1, t_2) + L_{\mathrm{G}}(s, t_1, t_2)]/\mathbb{E}[A(t_1, t_2)],$$

with the same notation as in `SERVICELEVEL`. The performance measure matrix has the same format as `SERVICELEVEL`, and this type of measure is equivalent to `SERVICELEVEL` if there is no abandonment, and all contacts exit the waiting queues before the end of the simulation.

NOTE: since this performance measure is of type `FUNCTIONOFEXPECTATIONS`, the complete list of observations generated by the simulator are not available directly; instead, one must use the performance measure `SERVICELEVEL2REP`.

Row type INBOUNDTYPEAWT

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`SERVICELEVEL2REP`

Represents the expectation of ratio version of `SERVICELEVEL2`.

Row type INBOUNDTYPEAWT

Column type MAINPERIOD

Estimation type EXPECTATIONOFFUNCTION

`SERVICELEVELG`

Service level performance measure for contact types and agent groups. Let $S_{\mathrm{G},k,i}(s, t_1, t_2)$ be the number of contacts of type k counted during time interval $[t_1, t_2]$ and served by agents in group i after a waiting time less than or equal to the acceptable waiting time s. Let $S_{k,i}(t_1, t_2)$ be the number of type-k contacts counted during the interval, and served by agents in groupe i. Let $B_k(t_1, t_2)$ and $L_{\mathrm{B},k}(s, t_1, t_2)$ be the number of contacts of type k counted during $[t_1, t_2]$, blocked and having abandoned after a waiting time greater than the acceptable waiting time, respectively. The service level is defined by

$$g_3(s, t_1, t_2) = \mathbb{E}[S_{\mathrm{G},k,i}(s, t_1, t_2)]/\mathbb{E}[S_{k,i}(t_1, t_2) + L_{\mathrm{B},k}(s, t_1, t_2) + B_k(t_1, t_2)].$$

Row type INBOUNDTYPEAWTAGENTGROUP

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`SERVICERATIO`

Probability of service, i.e., the fraction of the expected number of contacts served over the total expected number of arrivals.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`SERVICERATIOREP`

Corresponds to the expectation of ratio version of `SERVICERATIO`.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATIONOFFUNCTION

`SERVICETIME`

Expected total service time over the expected number of services, for each contact type, whether inbound or outbound. Usually, this can be computed easily from the input service time, and can therefore be used for checking parameter files. However, when call transfers or virtual queueing occur, service times can be altered by multipliers or additional random variables.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`SERVICETIMEG`

Expected total service time over the expected number of services, for each (contact type, agent group).

Row type CONTACTTYPEAGENTGROUP

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`SERVICETIMEREP`

Corresponds to the expectation of ratio version of `SERVICETIME`.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATIONOFFUNCTION

`SPEEDOFANSWER`

Average speed of answer, i.e., the expected total waiting time of served contacts over the expected number of served contacts, for each contact type, whether inbound or outbound. The numerator of the ratio corresponds to `SUMWAITINGTIMESSERVED`, while the denominator corresponds to `RATEOFSERVICES`.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`SPEEDOFANSWERG`

Average speed of answer for (contact type, agent group), i.e., the expected total waiting time of served contacts over the expected number of served contacts, for each (contact type, agent group) pair.

Row type CONTACTTYPEAGENTGROUP

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`SPEEDOFANSWERREP`

Corresponds to the expectation of ratio version of `SPEEDOFANSWER`.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATIONOFFUNCTION

`SUMEXCESSTIMES`

Represents the expected sum of excess times of contacts. For a contact with waiting time W and acceptable waiting time s used for computing the service level, the excess time is $(W - s)^+$.

Row type INBOUNDTYPEAWT

Column type MAINPERIOD

Estimation type EXPECTATION

`SUMEXCESSTIMESABANDONED`

Represents the expected sum of excess times of contacts having abandoned.

Row type INBOUNDTYPEAWT

Column type MAINPERIOD

Estimation type EXPECTATION

`SUMEXCESSTIMESSERVED`

Represents the expected sum of excess times of served contacts.

Row type INBOUNDTYPEAWT

Column type MAINPERIOD

Estimation type EXPECTATION

`SUMSERVICETIMES`

Represents the sum of service times of contacts.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATION

`SUMWAITINGTIMES`

Represents the sum of waiting times, for each contact type.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATION

`SUMSQUAREDIFFESTREALWAITINGTIMES`

Represents the sum of square of difference Estimate and Real waiting times, for each contact type.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATION

`SUMWAITINGTIMESABANDONED`

Represents the sum of waiting times of contacts having abandoned, for each contact type.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATION

`SUMSQUAREDIFFESTREALWAITINGTIMESABANDONED`

Represents the sum of square of difference Estimate and Real waiting times of contacts having abandoned, for each contact type.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATION

`SUMWAITINGTIMESSERVED`

Represents the sum of waiting times of served contacts, for each contact type.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATION

`SUMSQUAREDIFFESTREALWAITINGTIMESSERVED`

Represents the sum of square of difference Estimate and Real waiting times of served contacts, for each contact type.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATION

`SUMWAITINGTIMESVQ`

Represents the sum of waiting times in virtual queue, for each contact type.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATION

`SUMSQUAREDIFFESTREALWAITINGTIMESVQ`

Represents the sum of square of difference Estimate and Real waiting times in virtual queue, for each contact type.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATION

`SUMWAITINGTIMESVQABANDONED`

Represents the sum of waiting times in virtual queue of contacts having abandoned, for each contact type.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATION

`SUMSQUAREDIFFESTREALWAITINGTIMESVQABANDONED`

Represents the sum of square of difference Estimate and Real waiting times in virtual queue of contacts having abandoned, for each contact type.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATION

`SUMWAITINGTIMESVQSERVED`

Represents the sum of waiting times in virtual queue of served contacts, for each contact type.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATION

`SUMSQUAREDIFFESTREALWAITINGTIMESVQSERVED`

Represents the sum of square of difference Estimate and Real waiting times in virtual queue of served contacts, for each contact type.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATION

`TIMETOABANDON`

Time to abandon of contacts, i.e., the expected total waiting time of contacts having abandoned over the expected number of contacts having abandoned, for each contact type, whether inbound or outbound. The numerator of the ratio corresponds to `SUMWAITINGTIMESABANDONED`, while the denominator corresponds to `RATEOFABANDONMENT`.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`TIMETOABANDONREP`

Corresponds to the expectation of ratio version of `TIMETOABANDON`.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATIONOFFUNCTION

`WAITINGTIME`

Expected total waiting time over the expected number of arrivals, for each contact type, whether inbound or outbound, whether served or having abandoned. For outbound contacts, the expected waiting times are non-zero only when mismatches are not dropped. The numerator of the ratio corresponds to `SUMWAITINGTIMES`, while the denominator corresponds to `RATEOFARRIVALS`.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`MSEWAITINGTIME`

Expected total waiting time over the expected number of arrivals, for each contact type, whether inbound or outbound, whether served or having abandoned. For outbound contacts, the expected waiting times are non-zero only when mismatches are not dropped. The numerator of the ratio corresponds to `SUMWAITINGTIMES`, while the denominator corresponds to `RATEOFARRIVALS`.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`MSEWAITINGTIMEABANDONED`

Average ie MSE time spent in virtual queue before contact back followed by abandonment. The numerator of the ratio corresponds to `SUMWAITINGTIMESABANDONED`, while the denominator corresponds to `RATEOFABANDONMENT`.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`MSEWAITINGTIMESERVED`

Average time ie MSE spent in virtual queue for contacts served after they are contacted back. The numerator of the ratio corresponds to `SUMWAITINGTIMESVQSERVED`, while the denominator corresponds to `RATEOFSERVICES`.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`WAITINGTIMEG`

`WAITINGTIMEREP`

Corresponds to the expectation of ratio version of `WAITINGTIME`.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATIONOFFUNCTION

`WAITINGTIMEVQ`

Expected total waiting time in virtual queue over the expected number of arrivals, for each contact type, whether inbound or outbound, whether served or having abandoned. The numerator of the ratio corresponds to `SUMWAITINGTIMESVQ`, while the denominator corresponds to `RATEOFARRIVALS`. Note that this waiting time is not counted in the regular waiting time corresponding to `WAITINGTIME` type of performance measure.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`MSEWAITINGTIMEVQ`

Expected total waiting time, ie MSE in virtual queue over the expected number of arrivals, for each contact type, whether inbound or outbound, whether served or having abandoned. The numerator of the ratio corresponds to `SUMWAITINGTIMESVQ`, while the denominator corresponds to `RATEOFARRIVALS`. Note that this waiting time is not counted in the regular waiting time corresponding to `WAITINGTIME` type of performance measure.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`WAITINGTIMEVQABANDONED`

Average time spent in virtual queue before contact back followed by abandonment. The numerator of the ratio corresponds to `SUMWAITINGTIMESVQABANDONED`, while the denominator corresponds to `RATEOFABANDONMENT`.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`MSEWAITINGTIMEVQABANDONED`

Average ie MSE time spent in virtual queue before contact back followed by abandonment. The numerator of the ratio corresponds to `SUMWAITINGTIMESVQABANDONED`, while the denominator corresponds to `RATEOFABANDONMENT`.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`WAITINGTIMEVQABANDONEDREP`

Corresponds to the expectation of ratio version of `WAITINGTIMEVQABANDONED`.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATIONOFFUNCTION

`WAITINGTIMEVQREP`

Corresponds to the expectation of ratio version of `WAITINGTIMEVQ`.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATIONOFFUNCTION

`WAITINGTIMEVQSERVED`

Average time spent in virtual queue for contacts served after they are contacted back. The numerator of the ratio corresponds to `SUMWAITINGTIMESVQSERVED`, while the denominator corresponds to `RATEOFSERVICES`.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`MSEWAITINGTIMEVQSERVED`

Average time ie MSE spent in virtual queue for contacts served after they are contacted back. The numerator of the ratio corresponds to `SUMWAITINGTIMESVQSERVED`, while the denominator corresponds to `RATEOFSERVICES`.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`WAITINGTIMEVQSERVEDREP`

Corresponds to the expectation of ratio version of `WAITINGTIMEVQSERVED`.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATIONOFFUNCTION

`WAITINGTIMEWAIT`

Expected total waiting time over the expected number of contacts having to wait in queue. The numerator of the ratio corresponds to `SUMWAITINGTIMES`, while the denominator corresponds to `RATEOFDELAY`.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type FUNCTIONOFEXPECTATIONS

`WAITINGTIMEWAITREP`

Corresponds to the expectation of ratio version of `WAITINGTIMEWAIT`.

Row type CONTACTTYPE

Column type MAINPERIOD

Estimation type EXPECTATIONOFFUNCTION

Fields

```
@Deprecated public static final PerformanceMeasureType ABANDONMENTRATE
```

Deprecated Use `ABANDONMENTRATIO` instead.

```
@Deprecated public static final PerformanceMeasureType
ABANDONMENTRATEAFTERAWT
```

Deprecated Use `ABANDONMENTRATIOAFTERAWT` instead.

```
@Deprecated public static final PerformanceMeasureType
ABANDONMENTRATEBEFOREAWT
```

Deprecated Use `ABANDONMENTRATIOBEFOREAWT` instead.

```
@Deprecated public static final PerformanceMeasureType ABANDONMENTRATEREP
```

Deprecated Use `ABANDONMENTRATIOREP` instead.

```
@Deprecated public static final PerformanceMeasureType ABANDONRATE
```

Deprecated Use `ABANDONMENTRATIO` instead.

```
@Deprecated public static final PerformanceMeasureType ABANDONRATEAFTERAWT
```

Deprecated Use `ABANDONMENTRATIOAFTERAWT` instead.

```
@Deprecated public static final PerformanceMeasureType ABANDONRATEBEFOREAWT
```

Deprecated Use `ABANDONMENTRATIOBEFOREAWT` instead.

```
@Deprecated public static final PerformanceMeasureType ABANDONRATEREP
```

Deprecated Use `ABANDONMENTRATIOREP` instead.

```
@Deprecated public static final PerformanceMeasureType BLOCKRATE
```

Deprecated Use `BLOCKRATIO` instead.

```
@Deprecated public static final PerformanceMeasureType BLOCKRATEREP
```

Deprecated Use `BLOCKRATIOREP` instead.

```
@Deprecated public static final PerformanceMeasureType PATIENCETIME
```

Deprecated Use `TIMETOABANDON` instead.

```
@Deprecated public static final PerformanceMeasureType PATIENCETIMEREP
```

Deprecated Use `TIMETOABANDONREP` instead.

```
@Deprecated public static final PerformanceMeasureType POSWAITRATIO
```

Deprecated Use `DELAYRATIO` instead.

```
@Deprecated public static final PerformanceMeasureType POSWAITRATIOREP
```

Deprecated Use `DELAYRATIOREP` instead.

```
@Deprecated public static final PerformanceMeasureType QOS
```

Deprecated Use `SERVICELEVEL` instead.

```
@Deprecated public static final PerformanceMeasureType QOS2
```

Deprecated Use `SERVICELEVEL2` instead.

```
@Deprecated public static final PerformanceMeasureType QOS2REP
```

Deprecated Use `SERVICELEVEL2REP` instead.

```
@Deprecated public static final PerformanceMeasureType QOSREP
```

Deprecated Use `SERVICELEVELREP` instead.

```
@Deprecated public static final PerformanceMeasureType RATEOFPOSWAIT
```

Deprecated Use `RATEOFDELAY` instead.

Methods

`public static DoubleMatrix2D getAgentToContactTrafficMatrix (ContactCenterEval eval)`

Constructs and returns the agent-to-contact traffic matrix for the contact center evaluation system `eval`. This traffic matrix has dimensions $I' \times K$, where $I' = I + 1$ if $I > 1$, and I otherwise. Element (i, k) of the matrix gives the fraction of contacts of type k served by agents in group i over the total number of contacts served by agents in group i. This fraction is 0 if the corresponding routing is not allowed. Element (I, k) gives the total fraction of contacts of type k served by any agent. Each column of a given row always sums to 1. This matrix is computed from the served rates (see `SERVEDRATES`).

Parameter

`eval` the evaluation system.

Returns the agent-to-contact traffic matrix.

Throws

`NullPointerException` if `eval` is `null`.

`IllegalStateException` if `ContactCenterEval.eval()` was never called on `eval`.

`NoSuchElementException` if the `SERVEDRATES` performance measure type is not supported by `eval`.

`public static DoubleMatrix2D getContactToAgentTrafficMatrix (ContactCenterEval eval)`

Constructs and returns the contact-to-agent traffic matrix for the contact center evaluation system `eval`. This traffic matrix has dimensions $K' \times I$, where $K' = K + 1$ if $K > 1$, and K otherwise. Element (k, i) of the matrix gives the fraction of contacts of type k sent to agents in group i, over the total number of served contacts of type k. This fraction is 0 if the corresponding routing is not allowed. Element (K, i) gives the total fraction of contacts served by agents in group i. Each column of a given row always sums to 1. This matrix is computed from the served rates (see `SERVEDRATES`).

Parameter

`eval` the evaluation system.

Returns the contact-to-agent traffic matrix.

Throws

`NullPointerException` if `eval` is `null`.

`IllegalStateException` if `ContactCenterEval.eval()` was never called on `eval`.

`NoSuchElementException` if the `SERVEDRATES` performance measure type is not supported by `eval`.

`public String columnName (ContactCenterInfo eval, int col)`

Returns the name associated with the column `col` in the matrix of results for this type of performance measure estimated by `eval`. For example, this may return `period 0` if called with index 0 for most performance measures.

Parameters

`eval` the contact center evaluation object.

`col` the column index.

Returns the column name.

`public Map<String, String> columnProperties (ContactCenterInfo eval, int column)`

Returns the properties associated with the column `column` in a matrix of results for this type of performance measure estimated by `eval`.

Parameters

`eval` the contact center evaluation object.

`column` the column index.

Returns the column properties.

`public int columns (ContactCenterInfo eval)`

Returns the number of columns in a matrix of performance measures of this type estimated by the evaluation system `eval`.

Parameter

`eval` the queried evaluation system.

Returns the number of columns.

`public String columnTitle()`

Returns the title that should identify the columns of the matrix of results for this type of performance measure. This returns `Periods` for most performance measures.

Returns the column title.

`public ColumnType getColumnType()`

Returns the type of the columns in any matrix of this type of performance measure. Usually, columns represent main periods.

Returns the column type.

`public String getDescription()`

Returns the descriptive name of this group of performance measures. The returned name is intended to be used in reports, while the name returned by the method `Enum.name()` corresponds to the internal name of this type of performance measure, used in programs.

Returns the name of the group of performance measures.

`public EstimationType getEstimationType()`

Returns the type of estimation specified for this type of performance measure. This can be an expectation, a ratio of expectations, an expectation of ratios, or a raw statistic.

Returns the type of estimation for this performance measure type.

`public RowType getRowType()`

Returns the type of the rows in any matrix of this type of performance measure. For example, rows can represent contact types, or agent groups.

Returns the row type.

`public double getZeroOverZeroValue()`

Determines the value associated with the undefined 0/0 ratio, for performance measures of this type.

Returns the value assocaited with 0/0.

`public boolean isPercentage()`

Returns `true` if and only if performance measures of this type can be expressed in percentage. Such measures are ratios defined on $[0, 1]$, e.g., the service level, and may be formatted in percentage by reporting facilities.

Returns `true` if and only if performance measures of this type can be expressed as percentages.

`public boolean isTime()`

Determines if performance measures of this type represent time durations. This includes, e.g., waiting times and service times. Times produced by evaluation systems are expressed in the default unit returned by `ContactCenterInfo.getDefaultUnit()`. Reporting facilities can convert this time to the appropriate visual representation.

Returns `true` if and only if performance measures of this type represent times.

`public String rowName (ContactCenterInfo eval, int row)`

Returns the name associated with the row `row` in a matrix of results for this type of performance measure estimated by `eval`. For example, if the method is called for the service level, and row 0, it may return `inbound type 0`.

Parameters

`eval` the contact center evaluation object.

`row` the row index.

Returns the row name.

`public Map<String, String> rowProperties (ContactCenterInfo eval, int row)`

Returns the properties associated with the row `row` in a matrix of results for this type of performance measure estimated by `eval`.

Parameters

`eval` the contact center evaluation object.

`row` the row index.

Returns the row properties.

`public int rows (ContactCenterInfo eval)`

Returns the number of rows in a matrix of performance measures of this type estimated by the evaluation system `eval`.

Parameter

`eval` the queried evaluation system.

Returns the number of rows.

`public String rowTitle()`

Returns the title that should identify the rows of matrices of results for this type of performance measure. For example, this may return `Groups` for agents' occupancy ratio.

Returns the row title.

EstimationType

Represents the type of estimation specified for a group of performance measures. The estimation type gives clues on how performance measures are estimated.

```
package umontreal.iro.lecuyer.contactcenters.app;

public enum EstimationType
```

Constants

`RAWSTATISTIC`

Raw statistics which do not estimate expectations. For example, this can be the maximal queue size during a simulation, which has no average or sample variance. When simulating multiple replications, one observation of each raw statistic is available for each replication. On the other hand, if a single replication is simulated, which occurs when using batch means, only a single observation of the raw statistics is generated.

`EXPECTATION`

Estimation of an expectation, by an average in the case of simulation. Most expectations correspond to rates, which are part of groups of performance measures whose names begin with `RATEOF`, and which are expected counts of certain event types occurring during a time interval, For example, `PerformanceMeasureType.RATEOFABANDONMENT` is defined as the expected rate of contacts having abandoned without receiving service during some time interval. Types of performance measures whose names begin with `SUM` are also normalized the same way as rates. By default, rates are considered relative to one main period, so `PerformanceMeasureType.RATEOFABANDONMENT` corresponds to the expected number of contacts having abandoned during a main period. However, if the `normalizeToDefaultUnit` attribute in simulation parameters is set to `true`, rates are treated as relative to one simulation time unit. Expected time-averages, which are not normalized as rates, are part of groups with names beginning with `AVG`, e.g., `PerformanceMeasureType.AVGQUEUESIZE` for the time-average queue size.

`FUNCTIONOFEXPECTATIONS`

Estimation of a function of multiple expectations, e.g., a ratio of expectations. Functions of expectations, estimated by functions of averages in the case of simulation, are part of groups whose names do not have the `RATEOF` or `AVG` prefixes, e.g., `PerformanceMeasureType.SERVICELEVEL`, and `PerformanceMeasureType.ABANDONMENTRATIO`. For now, these functions are ratios estimated as follows. Let $(X_0, Y_0), \ldots, (X_{n-1}, Y_{n-1})$ be random vectors generated during an experiment. Pairs of observations can come from independent replications or from batches, depending on the method of experiment. Assuming that

$$\bar{X}_n = \frac{1}{n}\sum_{r=0}^{n-1} X_r \to \mathbb{E}[X]$$

and

$$\bar{Y}_n = \frac{1}{n}\sum_{r=0}^{n-1} Y_r \to \mathbb{E}[Y]$$

as $n \to \infty$, a simulator estimates the ratio by computing

$$\bar{\nu}_n = \frac{\bar{X}_n}{\bar{Y}_n}$$

which is an estimator of

$$\frac{\mathbb{E}[X]}{\mathbb{E}[Y]} = \nu.$$

At the end of an experiment, a single copy of the estimator is available, and only sample variance and confidence interval are available for $\bar{\nu}_n$, not observations.

`EXPECTATIONOFFUNCTION`

Estimation of the expectation of a function of several random variables whose expectations are themselves represented by other types of performance measures. For example, this can be the expectation of a ratio. Expectations of functions are part of groups with names having the `REP` suffix, and have corresponding functions of expectations. They are not recommended for analysis, because their estimators, averages of functions, are more noisy than functions of averages. They correspond to

$$\frac{1}{n}\sum_{r=0}^{n-1} \frac{X_r}{Y_r},$$

an estimator of

$$\mathbb{E}\left[\frac{X}{Y}\right].$$

When $n \to \infty$, this also estimates $\mathbb{E}[X]/\mathbb{E}[Y]$. An average of ratios can be used to estimate a short-term expectation. It is needed when several observations are necessary to compute statistics different from average, sample variance, and confidence intervals, e.g., quantiles.

RowType

Represents the row type for a matrix regrouping performance measures. Each type of performance measure has a row type that affects the number and role of rows in any matrix of performance measures of that type. Of course, the number of rows is also affected by the parameters of the contact center.

Each row of a matrix of performance measures corresponds to one type of event. Usually, there is one row per contact type or agent group, and an extra row for the aggregate measures. If estimates of some performance measures are missing in a matrix of results, e.g., an approximation cannot compute them, they can be replaced by `Double.NaN`. The aggregate value is often defined as the sum of the values for each event type. In this case, if there is a single event type, the matrix has a single row since the per-type and aggregate values are the same.

```
package umontreal.iro.lecuyer.contactcenters.app;

public enum RowType
```

Constants

`INBOUNDTYPE`

Rows representing segments of inbound contact types. More specifically, let $K'_{\mathrm{I}} \geq K_{\mathrm{I}}$ be the number of rows of this type for a specific model of contact center. If a matrix has rows of this type and if there are K_{I} inbound contact types in the model, row $k = 0, \ldots, K_{\mathrm{I}} - 1$ represents contact type k while row $K'_{\mathrm{I}} - 1$ is used to represent all contact types. Rows $K_{\mathrm{I}}, \ldots, K'_{\mathrm{I}} - 2$ represent user-defined segments regrouping inbound contact types. If $K_{\mathrm{I}} = 1$, a single row represents the single inbound contact type, and $K'_{\mathrm{I}} = K_{\mathrm{I}}$.

`INBOUNDTYPEAWT`

Rows representing segments of inbound contact types, for performance measures using acceptable waiting times. This is similar to `INBOUNDTYPE`, except that there is one group of rows for each matrix of acceptable waiting times. More specifically, if there are K'_{I} segments of inbound contact types and M user-specified matrices of acceptable waiting times (often, $M = 1$), row $mK'_{\mathrm{I}} + k$ represents segment of inbound contact types k with the mth matrix of AWTs. The total number of rows is MK'_{I}.

`OUTBOUNDTYPE`

Rows representing segments of outbound contact types. More specifically, let $K'_{\mathrm{O}} \geq K_{\mathrm{O}}$ be the number of rows of this type for a specific model of contact center. If a matrix has rows of this type and if there are K_{O} outbound contact types in the model, row $k = 0, \ldots, K_{\mathrm{O}} - 1$ represents outbound contact type k while row $K'_{\mathrm{O}} - 1$ is used to represent all contact types. Rows $K_{\mathrm{O}}, \ldots, K'_{\mathrm{O}} - 2$ represent user-defined segments regrouping outbound contact types. If $K_{\mathrm{O}} = 1$, a single row represents the single outbound contact type, and $K'_{\mathrm{O}} = K_{\mathrm{O}}$.

`CONTACTTYPE`

Rows representing segments of contact types. More specifically, let $K' \geq K$ be the number of rows of this type for a specific model of contact center. If a matrix has rows of this type and if there are K contact types in the model, row $k = 0, \ldots, K-1$ represents contact type k while row $K' - 1$ is used for representing all contact types. Rows $K, \ldots, K' - 2$ represent user-defined segments regrouping contact types. If $K = 1$, a single row represents the single contact type, and $K' = K$.

`INBOUNDTYPEAGENTGROUP`

Rows representing inbound contact types/agent group pairs. More specifically, let K'_{I} be the number of segments of inbound contact types, and I' be the number of segments of agent groups. If a matrix has this type of row, row $kI'+i$, for $k = 0, \ldots, K'_{\mathrm{I}}-1$ and $i = 0, \ldots, I'-1$, represents inbound contact types in segment k served by agents in segment of groups i. The total number of rows is $K'_{\mathrm{I}}I'$.

`INBOUNDTYPEAWTAGENTGROUP`

Rows representing inbound contact types/agent group pairs, for performance measures using acceptable waiting times. This is similar to `INBOUNDTYPEAGENTGROUP`, except that there is one group of rows for each matrix of acceptable waiting times. More specifically, if there are K_{I} segments of inbound contact types, I' segments of agent groups, and M matrices of acceptable waiting times (often, $M = 1$), row $mK'_{\mathrm{I}}I' + kI' + i$ represents segment of inbound contact types k and agent group i with the mth matrix of AWTs. The total number of rows is $MK'_{\mathrm{I}}I'$.

`OUTBOUNDTYPEAGENTGROUP`

Rows representing outbound contact types/agent group pairs. More specifically, let K'_{O} be the number of segments of outbound contact types, and I' be the number of segments of agent groups. If a matrix has this type of row, row $kI' + i$, for $k = 0, \ldots, K'_{\mathrm{O}} - 1$ and $i = 0, \ldots, I'-1$, represents outbound contact types in segment k served by agents in segment of groups i. The total number of rows is $K'_{\mathrm{O}}I'$.

`CONTACTTYPEAGENTGROUP`

Rows representing contact types/agent group pairs. More specifically, let K' be the number of segments of contact types, and I' be the number of segments of agent groups. If a matrix has this type of row, row $kI' + i$, for $k = 0, \ldots, K' - 1$ and $i = 0, \ldots, I' - 1$, represents contact types in segment k served by agents in segment of groups i. The total number of rows is $K'I'$.

`WAITINGQUEUE`

Rows representing waiting queues. More specifically, let $Q' \geq Q$ be the number of rows of this type for a specific model of contact center. If a matrix has rows of this type and if there are Q waiting queues in the model, row $q = 0, \ldots, Q - 1$ represents waiting queue q while row $Q' - 1$ is used for representing all waiting queues. Rows $Q, \ldots, Q' - 2$ represent user-defined segments regrouping waiting queues. If $Q = 1$, a single row represents the single waiting queue, and $Q' = Q$.

`AGENTGROUP`

Rows representing agent groups. More specifically, let $I' \geq I$ be the number of rows of this type for a specific model of contact center. If a matrix has rows of this type and if there are I agent groups in the model, row $i = 0, \ldots, I-1$ represents agent group i while row $I'-1$ is used for representing all agent groups. Rows $I, \ldots, I'-2$ represent user-defined segments regrouping agent groups. If $I = 1$, a single row represents the single agent group, and $I' = I$.

Methods

`public boolean isContactType()`

Determines if this row type corresponds to contact types. Returns `true` if and only if this row type corresponds to `INBOUNDTYPE`, `INBOUNDTYPEAWT`, `OUTBOUNDTYPE`, or `CONTACTTYPE`.

Returns `true` if and only if this row type is related to contact types.

`public boolean isContactTypeAgentGroup()`

Determines if this row type corresponds to (contact type, agent group) pairs. Returns `true` if and only if this row type corresponds to `INBOUNDTYPEAGENTGROUP`, `INBOUNDTYPEAWTAGENTGROUP`, `OUTBOUNDTYPEAGENTGROUP`, or `CONTACTTYPEAGENTGROUP`.

Returns `true` if and only if this row type is related to (contact type, agent group) pairs.

`public RowType toInboundType()`

Converts this row type to a row type representing inbound contact types. Returns `INBOUNDTYPE` if this row type corresponds to `CONTACTTYPE`, and `INBOUNDTYPEAGENTGROUP` if this row type corresponds to `CONTACTTYPEAGENTGROUP`. Otherwise, throws an illegal-argument exception.

Returns the equivalent of this row type for inbound contact types.

`public RowType toInboundTypeAWT()`

Similar to `toInboundType()`, but converts to inbound contact type with acceptable waiting times. Returns `INBOUNDTYPEAWT` if this row type corresponds to `CONTACTTYPE`, and `INBOUNDTYPEAWTAGENTGROUP` if this row type corresponds to `CONTACTTYPEAGENTGROUP`. Otherwise, throws an illegal-argument exception.

Returns the equivalent of this row type for inbound contact types.

`public RowType toOutboundType()`

Converts this row type to a row type representing outbound contact types. Returns `OUTBOUNDTYPE` if this row type corresponds to `CONTACTTYPE`, and `OUTBOUNDTYPEAGENTGROUP` if this row type corresponds to `CONTACTTYPEAGENTGROUP`. Otherwise, throws an illegal-argument exception.

Returns the equivalent of this row type for outbound contact types.

`public RowType toContactTypeAgentGroup()`

Returns the equivalent of this row type for pairs with agent groups. This method returns `INBOUNDTYPEAGENTGROUP` if this row type is `INBOUNDTYPE`, `CONTACTTYPEAGENTGROUP` if this row type is `CONTACTTYPE`, etc.

Returns the equivalent of this row type for pairs with agent groups.

`public RowType toContactType()`

Reverse of `toContactTypeAgentGroup()`. For example, this returns `INBOUNDTYPE` if this row type is `INBOUNDTYPEAGENTGROUP`, `CONTACTTYPE` if this row type is `CONTACTTYPEAGENTGROUP`, etc.

`public abstract String getTitle()`

Returns the title that should identify the rows of matrices of results for this type of row. For example, this may return `Groups` for `AGENTGROUP`.

Returns the row title.

`public abstract String getName (ContactCenterInfo eval, int row)`

Returns the name associated with the row `row` in a matrix of results for this type of row estimated by `eval`. For example, if the method is called for `INBOUNDTYPE` and row 0, it may return `inbound type 0`.

Parameters

`eval` the contact center evaluation object.

`row` the row index.

Returns the row name.

`public abstract Map<String, String> getProperties (ContactCenterInfo eval, int row)`

Returns the properties associated with row `row`. Properties are additional strings describing a row. This can include, e.g., the language of the customers, the originating region, etc. If no property is defined for the given row, this method returns an empty map.

Parameters

`eval` the evaluation system.

`row` the row index.

Returns the properties.

`public abstract int count (ContactCenterInfo eval)`

Returns the usual number of rows in a matrix of performance measures with rows of this type estimated by the evaluation system `eval`.

Parameter

`eval` the queried evaluation system.

Returns the number of rows.

ColumnType

Represents the column type for a matrix regrouping performance measures. Each type of performance measure has a column type that affects the number and role of columns in any matrix of performance measures of that type. Of course, the number of columns is also affected by the parameters of the contact center.

With the exception of `PerformanceMeasureType.SERVEDRATES` and `PerformanceMeasureType.MAXQUEUESIZE`, each column corresponds to a main period in the model, and the last column corresponds to the time-aggregate values. If there is a single period, e.g., for steady-state approximations or simulations, the matrix can have a single column. Note that when using batch means, matrices of results do not contain a column for each batch. To get values for each batch in a stationary simulation, one must use a contact center simulator with observations and call `ContactCenterSimWithObservations.getObs (PerformanceMeasureType, int, int)`. One must also make sure to set up the simulator to keep track of the observations, which is implementation-specific. For implementations using `BatchSimParams` for experiment parameters, the method `SimParams.setKeepObs (boolean)` can be used for this.

```
package umontreal.iro.lecuyer.contactcenters.app;

public enum ColumnType
```

Constants

`MAINPERIOD`

Columns representing main periods. More specifically, let $P' \geq P$ be the number of columns of this type for a specific model of contact center. If a matrix has columns of this type and if there are P main periods in the model, column $p = 0, \ldots, P-1$ represents main period p while column $P'-1$ is used for representing all main periods. Columns $P, \ldots, P'-2$ represent user-defined segments regrouping main periods. If $P = 1$, a single column represents the single main period, and $P' = P$.

`AGENTGROUP`

Columns representing agent groups. This is similar to `RowType.AGENTGROUP`, with rows replaced with columns.

`SINGLECOLUMN`

Single column with no particular meaning. For example, the maximal queue size has one row for each waiting queue but a single column.

Methods

`public abstract String getTitle()`

Returns the title that should identify the rows of matrices of results for this type of column. For example, this may return `Periods` for `MAINPERIOD`.

Returns the column title.

`public abstract String getName (ContactCenterInfo eval, int column)`

Returns the name associated with the column `column` in a matrix of results for this type of column estimated by `eval`. For example, if the method is called for `MAINPERIOD`, and column 0, it may return `Period 0`.

Parameters

`eval` the contact center evaluation object.

`column` the column index.

Returns the column name.

`public abstract Map<String, String> getProperties (ContactCenterInfo eval, int column)`

Returns the properties associated with column `column`. Properties are additional strings describing a column. This can include, e.g., the language of the customers, the originating region, etc. If no property is defined for the given column, this method returns an empty map.

Parameters

`eval` the evaluation system.

`column` the column index.

Returns the properties.

`public abstract int count (ContactCenterInfo eval)`

Returns the usual number of columns in a matrix of performance measures with columns of this type estimated by the evaluation system `eval`.

Parameter

`eval` the queried evaluation system.

Returns the number of rows.

PerformanceMeasureFormat

Provides basic methods for formatting matrices of performance measures.

```
package umontreal.iro.lecuyer.contactcenters.app;

public abstract class PerformanceMeasureFormat
```

Methods

```
protected double getStandardDeviation (ContactCenterSimWithObservations
                                       sim, PerformanceMeasureType pm, int
                                       row, int column)
```

Return the standard deviation of this performance measure. Returns -1 on error.

Parameters

`sim` contact center

`pm` performance measure

`row`

`column`

Returns the standard deviation

```
protected void createHistogram (double[] obs, double sigma,
                                PerformanceMeasureParams pmp, String name)
```

Create a new histogram and add it to the list of histograms. The histogram is built from the observations `obs` for the performance measure `pmp`, whose description is given in `name`. The standard deviation `sigma` is used to fix the width of the bins, if positive; otherwise it is unused.

Parameters

`obs` the observations

`sigma` empirical standard deviation of the observations

`pmp` performance measure parameters

`name` performance measure name

```
public List<HistogramChart> getHistogramList()
```

Returns the list of histograms created by the call to `formatObservations` in derived classes.

Returns the list of all histograms

`public void writeHistograms()`

Show all histograms for the chosen measures on standard output.

`public void writeHistogramsLaTeX()`

Writes all histograms for the chosen measures in a LaTex file. Each histogram is written in a separate file.

```
public static String[] getShownProperties (Collection<PropertyNameParam>
                                            properties)
```

Converts a list of property names to an array of strings.

Parameter

`properties` the list of property names.

Returns the array of strings.

```
public int countRowsSummary (ContactCenterEval eval,
                             PerformanceMeasureType... pms)
```

Returns the number of rows in the summary report. This corresponds to the number of elements in `pms` for which `isIncludedInSummary (ContactCenterEval, PerformanceMeasureType)` returns `true`.

Parameter

`pms` the array of performance measure types.

Returns the number of rows.

```
public boolean isIncludedInReport (ContactCenterEval eval,
                                   PerformanceMeasureType pm)
```

Determines if the performance measure type `pm` is included in reports. By default, this returns `false` only if `pm` is `null`, if it is equal to `PerformanceMeasureType.SUMWAITINGTIMES`, or if `pm.getEstimationType()` returns `EstimationType.EXPECTATIONOFFUNCTION`.

Parameter

`pm` the tested type of performance measure.

Returns `true` if and only if the performance measure type must be included in reports.

`public boolean isIncludedInDefaultReport (PerformanceMeasureType pm)`

Determines if the performance measure type `pm` is included in reports when printed statistics are not specified by the user. By default, this returns `false` only if `pm` is `null`, if it is equal to `PerformanceMeasureType.SUMWAITINGTIMES`, or if `pm.getEstimationType()` returns `EstimationType.EXPECTATIONOFFUNCTION`.

Parameter

`pm` the tested type of performance measure.

Returns `true` if and only if the performance measure type must be included in reports.

```
public boolean isIncludedInSummary (ContactCenterEval eval,
                                    PerformanceMeasureType pm)
```

Determines if the performance measure type `pm` is included in the summary of reports.

By default, this method returns `true` if `isIncludedInReport (eval, pm)` returns `true`, and if `pm` does not correspond to `PerformanceMeasureType.SERVEDRATES`.

Parameter

`pm` the tested type of performance measure.

Returns `true` if and only if `pm` is included in summary for reports.

```
public String getName (ContactCenterInfo eval, PerformanceMeasureType pm,
                       int row, int col)
```

Returns the name associated with the performance measure of type `pm`, at row `row`, and column `col`. This name is constructed by using `PerformanceMeasureType.rowName (ContactCenterInfo, int)`, and `PerformanceMeasureType.columnName (ContactCenterInfo, int)`.

Parameters

`eval` the evaluation system.

`pm` the performance measure type.

`row` the row index.

`col` the column index.

Returns the name of the measure.

```
public String capitalizeFirstLetter (String s)
```

Returns the string `s` with the first letter in uppercase. If `s` is empty or `null`, this returns `s` unchanged.

Parameter

`s` the string to capitalized.

Returns the new string with the first letter in upper case.

`public String[] getValColumnNames()`

Name of the columns for tables containing values of performance measures. This array contains a single string representing the "Values" column of tables of results.

`public String[] getStatColumnNames()`

Name of the columns for tables containing statistics concerning performance measures. This array contains five elements representing columns for the minimum, the maximum, the average, the standard deviation, and the confidence interval.

`public PrintedStatParams[] getDefaultPrintedStatParams (ContactCenterEval eval, ReportParams reportParams)`

Returns a default array of parameters for printed statistics, for the evaluation system `eval`. This method uses `ContactCenterEval.getPerformanceMeasures()` to obtain an array of performance measures. For each element of this array, it creates a `PrintedStatParams` instance, and adds it into the returned list. Parameters for printed statistics are set to default, i.e., detailed statistics for all periods are printed.

Parameter

`eval` the evaluation system.

Returns the array of parameters for printed statistics.

`public PerformanceMeasureType[] getPerformanceMeasures (PrintedStatParams[] pstats)`

Constructs an array of performance measure types from the given array of printed statistics.

Parameter

`pstats` the array of printed statistics.

Returns the array of performance measure types.

`public PerformanceMeasureType[] getPerformanceMeasures (PrintedStatParams[] pstats, RowType... rowTypes)`

Constructs an array of performance measure types from the given array of printed statistics, and a row type. This method is similar to `getPerformanceMeasures (PrintedStatParams[])` except it returns measure types with a row type corresponding to `rowType`.

Parameters

`pstats` the array of printed statistics.

`rowTypes` the row types.

Returns the array of performance measure types.

```
public PerformanceMeasureType[] getPerformanceMeasures (PrintedStatParams[]
                                                        pstats, boolean
                                                        onlyAverages,
                                                        RowType...
                                                        rowTypes)
```

Constructs an array of performance measure types from the given array of printed statistics, and a row type. This method is similar to `getPerformanceMeasures (PrintedStatParams[])` except it returns measure types with a row type corresponding to `rowType`, and for which `p.getOnlyAverages` corresponds to `onlyAverages`.

Parameters

`pstats` the array of printed statistics.

`onlyAverages` determines the required status of the `onlyAverages` flag.

`rowTypes` the row types.

Returns the array of performance measure types.

```
public static void addExperimentInfo (Map<String, Object> evalInfo, String
                                      ccParamsFn, String simParamsFn)
```

Returns the header for simulation results. This string contains the name of the parameter file for the model, i.e., `ccParamsFn`, the name of the parameter file for the experiment, i.e., `simParamsFn`, and the current date.

Parameters

`ccParamsFn` the name of the parameter file for the model.

`simParamsFn` the name of the parameter file for the experiment.

```
public static void formatResults (ContactCenterEval eval, String
                                  outputFileName) throws IOException,
                                  JAXBException
```

Equivalent to `formatResults (ContactCenterEval, File)`, with a string given the file name instead of a file object.

```
public static void formatResults (ContactCenterEval eval, File outputFile)
                                  throws IOException, JAXBException
```

Formats the results of the last evaluation performed by `eval` into the file with name `outputFile`. The format of the file is determined automatically based on its extension.

Parameters

`eval` the evaluation system.

`outputFile` the output file

Throws

`IOException` if an I/O error occurs.

`ParserConfigurationException` if the output format is XML, and an error occurred while constructing the intermediate DOM document.

`TransformerException` if the output format is XML, and an error occurs during the transformation of the DOM document into text.

```
public static void formatResults (ContactCenterEval eval, OutputStream
                                  stream, CCResultsFormat fmt) throws
                                  IOException, JAXBException
```

Formats the results of the evaluation system `eval`. If `fmt` is `null` or empty, this method simply prints the contents returned by `eval.formatStatistics`. Otherwise, it saves the results in `stream`. Depending on the value of `fmt`, i.e., `TEXT`, `BINARY`, `XML`, or `EXCEL`, the format of the output file is plain text, binary, XML, or MS Excel, respectively.

Parameters

`eval` the evaluation system being processed.

`stream` the output stream.

`fmt` the format of the output.

Throws

`IOException` if an I/O error occurs.

`ParserConfigurationException` if the output format is XML, and an error occurred while constructing the intermediate DOM document.

`TransformerException` if the output format is XML, and an error occurs during the transformation of the DOM document into text.

PerformanceMeasureFormatText

Defines some facilities to format performance measures as strings. For each estimated performance measure, an evaluation system can produce matrices of results which may need to be formatted to be displayed on-screen or included in printable documents. This class defines methods to create summary reports for several performance measures or a detailed report for a particular measure.

Each formatting method constructs an `ObjectMatrix2D` instance containing `String` elements, each numerical value being processed using a double formatter implementing interfaces `DoubleFormatter` or `DoubleFormatterWithError`. After the intermediate matrix is constructed, the method uses an instance of `Formatter` to turn it into a `String`. By default, this class is adapted for on-screen reports, but methods may be overridden for other types of formatting.

```
package umontreal.iro.lecuyer.contactcenters.app;

public class PerformanceMeasureFormatText extends PerformanceMeasureFormat
```

Constructors

`public PerformanceMeasureFormatText()`

Creates a performance measure formatter using the default `Formatter` implementation adapted for on-screen printing.

`public PerformanceMeasureFormatText (Formatter fmt)`

Constructs a performance measure formatter with the formatter `fmt`.

Parameter

`fmt` the user-defined formatter object.

Throws

`NullPointerException` if `fmt` is `null`.

```
public PerformanceMeasureFormatText (Formatter fmt, DoubleFormatter
                                     dfmtVal, DoubleFormatter dfmtStat)
```

Constructs a performance measure formatter with the matrix formatter `fmt`, the double-precision formatter `fmtVal` for values (with unknown error), and `fmtStat` for statistics (with an estimated error).

Parameters

`fmt` the user-defined formatter object.

`dfmtVal` the double-precision formatter for values.

`dfmtStat` the doubl-precision formatter for statistics.

Throws

`NullPointerException` if any argument is `null`.

Methods

`public Formatter getMatrixFormatter()`

Returns the matrix formatter used by this object.

Returns the matrix formatter.

`public void setMatrixFormatter (Formatter fmt)`

Sets the matrix formatter used by this object to `fmt`.

Parameter

`fmt` the matrix formatter used.

`public DoubleFormatter getDoubleFormatterValues()`

Returns the double-precision formatter used for values.

Returns the double-precision formatter used for values.

`public void setDoubleFormatterValues (DoubleFormatter dfmtVal)`

Sets the double-precision formatter used for values to `dfmtVal`.

Parameter

`dfmtVal` the new double-precision formatter used for values.

`public DoubleFormatter getDoubleFormatterStatistics()`

Returns the double-precision formatter used for statistics.

Returns the double-precision formatter used for statistics.

`public void setDoubleFormatterStatistics (DoubleFormatter dfmtStat)`

Sets the double-precision formatter used for statistics to `dfmtStat`.

Parameter

`dfmtStat` the double-precision formatter used for statistics.

`public String getPercentString()`

Returns the string representing the percentage sign in reports, the default being `%`.

Returns the percentage string.

`public void setPercentString (String percentString)`

Sets the percentage string to `percentString`.

Parameter

`percentString` the new percentage string.

```
public String formatValuesSummary (ContactCenterEval eval,
                                   PerformanceMeasureType... pms)
```

Formats a report for all the performance measures `pms` supported by the evaluation system `eval`. It uses the `ContactCenterEval.getPerformanceMeasure (PerformanceMeasureType)` method to obtain a matrix of values for each performance measure in `pms` supported by `eval`. Considering the element at the bottom right of this matrix as the aggregate value, the method then formats this value for each performance measure, using `getDoubleFormatterValues()` to convert double-precision values to strings.

The intermediate formatting matrix contains one row for each performance measure, and a single column containing its aggregate value.

Parameters

`eval` the contact center evaluation system.

`pms` the array of performance measures.

Returns the string containing the values of performance measures.

```
public String formatValuesDetailed (ContactCenterEval eval,
                                    PerformanceMeasureType pm)
```

Returns a string containing the current values of the performance measures of type `pm` estimated by the evaluation system `eval`. This method uses `formatValuesSingleRow (ContactCenterInfo, PerformanceMeasureType, DoubleMatrix2D, int, int, int, int, String)` with a matrix of values obtained via `eval.getPerformanceMeasure`, and a description obtained via `pm.getDescription()`.

Parameters

`eval` the contact center evaluation system.

`pm` the performance measure of interest.

Returns the string containing the values of performance measure.

```
public String formatValuesDetailedMatrix (ContactCenterEval eval,
                                          PerformanceMeasureType pm)
```

Returns a string containing the current values of the performance measures of type `pm` estimated by the evaluation system `eval`. This method uses `formatValuesMatrix (ContactCenterInfo, PerformanceMeasureType, DoubleMatrix2D, int, int, int, int, boolean, String)` with a matrix of values obtained via `eval.getPerformanceMeasure`, and a description obtained via `pm.getDescription()`.

Parameters

`eval` the contact center evaluation system.

`pm` the performance measure of interest.

Returns the string containing the values of performance measure.

```
public String formatValuesDetailedHidePeriods (ContactCenterEval eval,
                                                PerformanceMeasureType pm)
```

Similar to `formatValuesDetailed (ContactCenterEval, PerformanceMeasureType)` except per-period values are not displayed.

Parameters

`eval` the evaluation system.

`pm` the type of performance measure.

Returns the formatted string.

```
public String formatValuesSingleRow (ContactCenterInfo eval,
                                     PerformanceMeasureType pm,
                                     DoubleMatrix2D valm, int row, int
                                     column, int height, int width, String
                                     description)
```

Formats the values in a matrix `valm.viewPart (row, column, height, width)` concerning performance measures of type `pm` obtained with the evaluation system `eval`. Numbers are formatted using `getDoubleFormatterValues()`, and `description` provides a description for the matrix.

Suppose that the given matrix has dimensions $a \times b$. For example, the matrix can contain averages or sample variances for different contact types and periods. This method creates a $ab \times 1$ intermediate matrix of strings with one row for each element of `valm`. The names of rows are constructed using `getName (eval, pm, i, j)`.

Parameters

`eval` the evaluation system.

`pm` the type of performance measures concerned.

`valm` the matrix of values.

`row` the starting row of the matrix to be formatted.

`column` the starting column of the matrix to be formatted.

`height` the height of the formatted matrix.

`width` the width of the formatted matrix.

`description` the description for the formatted matrix.

Returns the formatted matrix.

```
public String formatValuesMatrix (ContactCenterInfo eval,
                                  PerformanceMeasureType pm,
                                  DoubleMatrix2D valm, int row, int column,
                                  int height, int width, boolean
                                  transposedValm, String description)
```

This is similar to `formatValuesSingleRow (ContactCenterInfo, PerformanceMeasureType, DoubleMatrix2D, int, int, int, int, String)`, except that the intermediate matrix of strings has dimensions $a \times b$. The formatted strings obtained via this method are often more readable than with the preceding method, but they can contain excessively long lines if the `width` is large.

If `transposedValm` is `true`, the given matrix is considered to be transposed, i.e., the meaning of its rows and columns is inverted with respect to a typical matrix of performance measures of type `pm`. For example, let `pm` correspond to `PerformanceMeasureType.SERVEDRATES`. Usually, each row of `valm` corresponds to a contact type. With `transposedValm` enabled, each row of `valm` corresponds to an agent group. This flag only affects how rows and columns are named; it does not change the values being formatted.

Parameters

`eval` the evaluation system.

`pm` the type of performance measures concerned.

`valm` the matrix of values.

`row` the starting row of the matrix to be formatted.

`column` the starting column of the matrix to be formatted.

`height` the height of the formatted matrix.

`width` the width of the formatted matrix.

`transposedValm` determines if `valm` is transposed with respect to a typical matrix of performance measures of type `pm`.

`description` the description for the formatted matrix.

Returns the formatted matrix.

```
public String formatStatisticsSummary (ContactCenterSim sim, double level,
                                       PerformanceMeasureType... pms)
```

Formats a statistical report for all the performance measures in `pms` supported by the contact center simulator `sim`. This is similar to `formatValuesSummary (ContactCenterEval, PerformanceMeasureType...)`, with additional statistical information such as miminum, maximum, standard deviation, and confidence intervals with confidence level `level`, if available. Values are formatted using the formatter `getDoubleFormatterStatistics()`.

The format of the intermediate matrix is the same as `formatValuesSummary (ContactCenterEval, PerformanceMeasureType...)`, except that columns are defined for the minimum, maximum, average, standard deviation, and confidence interval.

Parameters

`sim` the contact center simulator.

`level` the level of confidence of the intervals

`pms` the array of performance measures.

Returns the statistics formatted as a string.

```
public String formatStatisticsDetailed (ContactCenterSim sim, double level,
                                        PerformanceMeasureType pm)
```

Returns a statistical report for all the values of the performance measure `pm` estimated by the simulator `sim`, with confidence intervals with level `level`. Values are formatted using `getDoubleFormatterStatistics()`.

The intermediate matrix has a format similar to the one constructed by `formatValues-Detailed (ContactCenterEval, PerformanceMeasureType)`, except that a column is defined for the minimum, maximum, average, standard deviation, and confidence interval.

Parameters

`sim` the contact center simulator.

`level` the level of confidence intervals.

`pm` the performance measure of interest.

Returns the statistical report formatted as a string.

```
public String formatStatisticsDetailedHidePeriods (ContactCenterSim sim,
                                                   double level,
                                                   PerformanceMeasureType
                                                   pm)
```

Similar to `formatStatisticsDetailed (ContactCenterSim, double, PerformanceMeasure-Type)` but does not format per-period statistics.

Parameters

`sim` the contact center simulator.

`level` the confidence level of the intervals.

`pm` the type of performance measures.

Returns the formatted string.

```
public String formatInfo (Map<String, Object> info)
```

Constructs and returns a string containing the evaluation information `info`. For each entry in the given map, this method formats a line of the form `key: value`. Values are formatted as follows. Any `null` reference becomes the string `null`, instances of `Number` are formatted using `NumberFormat`, and instances of `Date` are formatted using `DateFormat`. Any other non-`null` value is formatted using the `Object.toString()` method.

Parameter

`info` the evaluation information.

Returns the string with the formatted information.

```
public String formatObservations (ContactCenterSimWithObservations sim,
                                  ReportParams reportParams)
```

For each element `PerformanceMeasureParams` in the list returned by `ReportParams.getPrintedObs()`, formats a report containing the complete list of observations generated by the simulator `sim` for the referred performance measure. Each measure-specific report is concatenated, and the resulting string is returned.

Parameters

`sim` the queried simulator.

`reportParams` the report parameters.

Returns the string containing observations.

```
public String formatValues (ContactCenterEval eval, ReportParams
                            reportParams)
```

Formats and returns a string containing the report of the last evaluation performed by the system `eval`. This method can be called by the implementation of `ContactCenterEval.formatStatistics()`.

This method first calls `formatInfo (Map)` with the evaluation information of `eval`. It then formats a summary report using `formatValuesSummary (ContactCenterEval, PerformanceMeasureType...)`. Then, for each performance measure a detailed report is requested for, the method appends the contents of `formatStatisticsDetailed (ContactCenterSim, double, PerformanceMeasureType)` or `formatValuesDetailedHidePeriods (ContactCenterEval, PerformanceMeasureType)` to the report. The types of performance measures to include in the report are selected using `reportParams.getPrintedStats()` which can be `null` or empty; in these two latter cases, the report includes all performance measures supported by `eval`. Each element of `printedStats` specifies a type of performance measure to include in the report (if supported by `eval`), whether a detailed report must be included, and if this detailed report includes information about each individual period.

Parameters

`eval` the evaluation system.

`reportParams` the report parameters.

Returns the string containing the formatted report.

```
public String formatStatistics (ContactCenterSim sim, ReportParams
                                reportParams)
```

Similar to `formatValues (ContactCenterEval, ReportParams)`, except this formats a full statistical report using `formatStatisticsSummary (ContactCenterSim, double, PerformanceMeasureType...)`, and `formatStatisticsDetailed (ContactCenterSim, double, PerformanceMeasureType)`. If the given report parameters contains information about observations to print, the method also calls `formatObservations (ContactCenterSimWithObservations, ReportParams)`, and appends the result to the returned string.

Parameters

`sim` the contact center simulator.

`reportParams` the report parameters.

Returns the formatted report.

PerformanceMeasureFormatExcel

Provides methods used to format matrices of performance measures into Microsoft Excel spreadsheets. This class uses the JExcel API library to construct a workbook in memory, i.e., an instance of `WritableWorkbook`. Some methods are provided to add sheets to the current workbook, and tables of results to the current sheet. Methods are finally available to transfer the in-memory workbook into a disk file that can be read by Microsoft Excel, OpenOffice.org Calc, etc.

For example, the following code creates an Excel file containing three sheets containing the statistics for each aggregate performance measures (e.g., service level for all contact types, over all periods), the statistics for time-aggregate measures (e.g., service level for contacts of type k over all periods, for all k), and statistics for all performance measures (e.g., service level for contacts of type k during period p, for all k and p). The variable `sim` corresponds to any instance of `ContactCenterSim`.

```
PerformanceMeasureFormatExcel2fmt =
   new PerformanceMeasureFormatExcel2();
fmt.newSheet ("Summary");
fmt.formatStatisticsSummary (sim, 0.95, sim.getPerformanceMeasures());
fmt.newSheet ("Detailed, without individual period");
for (PerformanceMeasureType pm : sim.getPerformanceMeasures())
   fmt.formatStatisticsDetailedHidePeriods (sim, 0.95, pm);
fmt.newSheet ("Detailed, with individual periods");
for (PerformanceMeasureType pm : sim.getPerformanceMeasures())
   fmt.formatStatisticsDetailed (sim, 0.95, pm);
fmt.writeWorkbook (new File ("output.xls"));
```

```
package umontreal.iro.lecuyer.contactcenters.app;

public class PerformanceMeasureFormatExcel extends PerformanceMeasureFormat
```

Constructor

```
public PerformanceMeasureFormatExcel (WritableWorkbook wb)
```

Constructs a new performance measure formatter with the workbook `wb`.

Parameter

`wb` the workbook used for formatting.

Methods

`public void newSheet (String sheetName)`

Creates a new `WritableSheet` with name `sheetName`, and sets this new sheet as the current one. This method resets the current row index to 0, and the starting column to 0.

If a sheet with the given name already exists, this method appends a number to the given name. The number is incremented until the resulting resulting sheet name is unused.

Parameter

`sheetName` the name of the new sheet.

`public int getCurrentRow()`

Returns the index of the current row into the current spreadsheet. This corresponds to the row at which subsequent tables of results will be inserted. After each insertion, this index is incremented automatically. Consequently, the default index corresponds to the last row in the sheet.

Returns the index of the current row.

`public void setCurrentRow (int currentRow)`

Sets the index of the current row to `currentRow`

Parameter

`currentRow` the new index of the current row.

`public int getStartingColumn()`

Returns the index of the starting column of subsequent tables of results. This defaults to 0, but can be changed to format, e.g., side-by-side tables of results. This index is not modified by methods writing data to the spreadsheet.

Returns the index of the starting column.

`public void setStartingColumn (int startingColumn)`

Sets the index of the starting column to `startingColumn`.

Parameter

`startingColumn` the new starting column.

`public WritableWorkbook getCurrentWorkbook()`

Returns the high-level object representing the current workbook.

Returns the current workbook.

`public void setCurrentWorkbook (WritableWorkbook workbook)`

Sets the current workbook to `workbook`. This also resets the current sheet to `null`.

Parameter

`workbook` the new current workbook.

`public WritableSheet getCurrentSheet()`

Returns the high-level object representing the current spreadsheet.

Returns the current spreadsheet.

`public void setCurrentSheet (WritableSheet sheet)`

Sets the current sheet to `sheet`, and resets the current row and starting column to 0.

Parameter

`sheet` the new current sheet.

`public boolean getRowOverwrite()`

Determines the status of row overwriting which affects how rows are managed when the current row index is smaller than the number of rows. When overwriting is `true`, the formating methods reuse the already created rows. When overwriting is `false` (the default), formatting methods always inert new rows.

Returns the status of row overwriting.

`public void setRowOverwrite (boolean rowOverwrite)`

Sets the status of row overwriting to `rowOverwrite`.

Parameter

`rowOverwrite` the new status of row overwriting.

`public boolean getColumnOutlines()`

Determines if column outlines are created by formatting methods. The default value of this boolean is `false`.

Returns the status of the outline creation flag.

`public void setColumnOutlines (boolean columnOutlines)`

Sets the column outlines flag to `columnOutlines`.

Parameter

`columnOutlines` the new value of the flag.

`public boolean getRowOutlines()`

Determines if row outlines are created by formatting methods. The default value of this boolean is `false`.

Returns the status of the outline creation flag.

`public void setRowOutlines (boolean rowOutlines)`

Sets the row outlines flag to `rowOutlines`.

Parameter

`rowOutlines` the new value of the flag.

`public int getMaxColumns()`

Returns the maximal number of columns a spreadsheet may contain. The default value of this variable is 256. This affects how `formatValuesMatrix (ContactCenterInfo, PerformanceMeasureType, DoubleMatrix2D, int, int, int, int, boolean, String)`, and `formatStatisticsDetailedMatrix (ContactCenterSim, double, PerformanceMeasureType)` work.

Returns the maximal number of columns in a spreadsheet.

`public void setMaxColumns (int maxColumns)`

Sets the maximal number of columns in a spreadsheet to `maxColumns`.

Parameter

`maxColumns` the maximal number of columns in a spreadsheet.

`public void writeWorkbook() throws IOException`

Writes the current workbook..

Throws

`IOException` if an I/O error occurs during writing.

`public void groupRow (int fromRow, int toRow)`

This method does nothing as JExcel API does not support outlining yet. Creates an outline for rows `fromRow` to `toRow`. Calling the `groupRow` method repeatedly with the same values creates multiple identical outlines in the spreadsheet. This method calls `groupRow` if and only if the outline was not created previously.

Parameters

`fromRow` the starting row of the outline.

`toRow` the ending row of the outline.

`public void groupColumn (int fromColumn, int toColumn)`

Similar to `groupRow (int, int)`, for creating column outlines.

Parameters

`fromColumn` the starting column.

`toColumn` the ending column.

`public WritableCellFormat createTitleCellStyle() throws WriteException`

Creates a cell style for cells containing titles for tables of results.

By default, this method uses the `CellFormat` no-argument constructor to create the cell style, and sets its alignment to "center". One can override this method to apply user-defined cell styles (colors, fill patterns, borders, etc.).

Returns the created cell style.

```
public void formatValueRow (String name, String val, boolean borderTop,
                            boolean borderBottom, boolean borderBefore,
                            boolean borderBetween, boolean borderAfter,
                            boolean wrapText) throws WriteException
```

Adds a new row containing the value `val` of a string with name `name`. The first column of the row contains the string `name` while the second column contains `val`.

Parameters

`name` the name of the quantity.

`val` the value of the quantity.

`borderTop` determines if a top border must be set for the cells.

`borderBottom` determines if a bottom border must be set for the cells.

`borderBefore` determines if a left border must be set for the first cell.

`borderBetween` determines if a border must separate the two cells.

`borderAfter` determines if a right border must be set for the second cell.

`wrapText` determines if text can be wrapped.

```
public void formatValueRow (String name, double val, boolean percent,
                            TimeUnit timeUnit, boolean borderTop, boolean
                            borderBottom, boolean borderBefore, boolean
                            borderBetween, boolean borderAfter) throws
                            WriteException
```

Similar to `formatValueRow (String, String, boolean, boolean, boolean, boolean, boolean, boolean)`, with `val` being a numeric value. The value is formatted with the general (default) style if `percent` is `false`, or in percentage notation if `percent` is `true`.

Parameters

`name` the name of the quantity.

`val` the value of the quantity.

`percent` determines if the percentage notation must be used.

`timeUnit` the time unit of the formatted value, or `null` if the value does not correspond to a time.

`borderTop` determines if the cell containing the value has a top border.

`borderBottom` determines if the cell containing the value has a bottom border.

`borderBefore` determines if a left border must be set for the first cell.

`borderBetween` determines if a border must separate the two cells.

`borderAfter` determines if a right border must be set for the second cell.

```
public void formatValueRow (String name, int value, boolean borderTop,
                            boolean borderBottom, boolean borderBefore,
                            boolean borderBetween, boolean borderAfter)
                             throws WriteException
```

Similar to `formatValueRow (String, String, boolean, boolean, boolean, boolean, boolean, boolean)`, with `val` being an integer.

Parameters

`name` the name of the quantity.

`value` the value of the quantity.

`borderTop` determines if a top border must be set for the first cell.

`borderBottom` determines if a bottom border must be set for the first cell.

`borderBefore` determines if a left border must be set for the first cell.

`borderBetween` determines if a border must separate the two cells.

`borderAfter` determines if a right border must be set for the second cell.

```
public void formatValueRow (String name, Date val, boolean borderTop,
                            boolean borderBottom, boolean borderBefore,
                            boolean borderBetween, boolean borderAfter)
                             throws WriteException
```

Similar to `formatValueRow (String, String, boolean, boolean, boolean, boolean, boolean, boolean)`, with `val` being a date.

Parameters

`name` the name of the quantity.

`val` the value of the quantity.

`borderTop` determines if a top border must be set for the first cell.

`borderBottom` determines if a bottom border must be set for the first cell.

`borderBefore` determines if a left border must be set for the first cell.

`borderBetween` determines if a border must separate the two cells.

`borderAfter` determines if a right border must be set for the second cell.

```
public void skipRow()
```

Increments the current row index to leave a blank row in the current spreadsheet.

```
public boolean formatValuesSummary (ContactCenterEval eval, String
                                    description, PerformanceMeasureType...
                                    pms) throws WriteException
```

Adds a report for all the performance measures `pms` supported by the evaluation system `eval` into the current spreadsheet. This uses the `ContactCenterEval.getPerformanceMeasure (PerformanceMeasureType)` method to obtain a matrix of values for each performance measure in `pms` supported by `eval`. Considering the element at the bottom right of this matrix as the aggregate value, the method then, for each performance measure, adds a row containing the aggregate value.

The resulting table of results contains one row for each type of performance measure, and two columns (name of measure, and value).

Parameters

`eval` the contact center evaluation system.

`pms` the array of performance measures.

Returns `true` if and only if the sheet was modified.

```
public boolean formatValuesDetailed (ContactCenterEval eval,
                                     PerformanceMeasureType pm) throws
                                     WriteException
```

Adds a table containing the current values of the performance measures of type `pm` estimated by the evaluation system `eval` to the current spreadsheet. This method uses `formatValues-SingleColumn (ContactCenterInfo, PerformanceMeasureType, DoubleMatrix2D, int, int, int, int, String)` with a matrix of values obtained via `eval.getPerformanceMeasure`, and a description obtained via `pm.getDescription()`.

Parameters

`eval` the contact center evaluation system.

`pm` the performance measure of interest.

Returns `true` if and only if the sheet was modified.

```
public boolean formatValuesDetailedMatrix (ContactCenterEval eval,
                                           PerformanceMeasureType pm)
                                           throws WriteException
```

Adds a table containing the current values of the performance measures of type `pm` estimated by the evaluation system `eval` to the current spreadsheet. This method uses `formatValues-Matrix (ContactCenterInfo, PerformanceMeasureType, DoubleMatrix2D, int, int, int, int, boolean, String)` with a matrix of values obtained via `eval.getPerformanceMeasure`, and a description obtained via `pm.getDescription()`.

Parameters

`eval` the contact center evaluation system.

`pm` the performance measure of interest.

Returns `true` if and only if the sheet was modified.

```
public boolean formatValuesDetailedHidePeriods (ContactCenterEval eval,
                                                PerformanceMeasureType pm)
                                                 throws WriteException
```

Similar to `formatValuesDetailedMatrix (ContactCenterEval, PerformanceMeasureType)` except per-period values are not displayed.

Parameters

`eval` the evaluation system.

`pm` the type of performance measure.

Returns `true` if and only if the sheet was modified.

```
public boolean formatValuesSingleColumn (ContactCenterInfo eval,
                                         PerformanceMeasureType pm,
                                         DoubleMatrix2D valm, int row, int
                                         column, int height, int width,
                                         String description) throws
                                         WriteException
```

Adds a table to the current spreadsheet containing the values in a matrix `valm.viewPart (row, column, height, width)` concerning performance measures of type `pm` obtained with the evaluation system `eval`. The string `description` provides a description for the matrix which is displayed in a row preceding the table.

Suppose that the given matrix has dimensions $a \times b$. For example, the matrix can contain averages or sample variances for different contact types and periods. This method formats the results as a $ab \times 1$ matrix with one row for each element of `valm`. The names of rows are constructed using `getName (eval, pm, i, j)`.

Parameters

`eval` the evaluation system.

`pm` the type of performance measures concerned.

`valm` the matrix of values.

`row` the starting row of the matrix to be formatted.

`column` the starting column of the matrix to be formatted.

`height` the height of the formatted matrix.

`width` the width of the formatted matrix.

`description` the description for the formatted matrix.

Returns `true` if and only if the sheet was modified.

```
public boolean formatValuesMatrix (ContactCenterInfo eval,
                                   PerformanceMeasureType pm,
                                   DoubleMatrix2D valm, int row, int
                                   column, int height, int width, boolean
                                   transposedValm, String description)
                                    throws WriteException
```

This is similar to `formatValuesSingleColumn (ContactCenterInfo, PerformanceMeasureType, DoubleMatrix2D, int, int, int, int, String)`, except that the formatted table has dimensions $a \times b$. Tables obtained via this method are often more readable than with the preceding one, but they can contain excessively long lines if `width` is large. If `width` is greater than `getMaxColumns()`, this method calls the result of `formatValuesSingleColumn (ContactCenterInfo, PerformanceMeasureType, DoubleMatrix2D, int, int, int, int, String)`.

If `transposedValm` is `true`, the given matrix is considered to be transposed, i.e., the meaning of its rows and columns is inverted with respect to a typical matrix of performance measures of type `pm`. For example, let `pm` correspond to `PerformanceMeasureType.SERVEDRATES`. Usually, each row of `valm` corresponds to a contact type. With `transposedValm` enabled, each row of `valm` corresponds to an agent group. This flag only affects how rows and columns are named; it does not change the values being formatted.

Parameters

`eval` the evaluation system.

`pm` the type of performance measures concerned.

`valm` the matrix of values.

`row` the starting row of the matrix to be formatted.

`column` the starting column of the matrix to be formatted.

`height` the height of the formatted matrix.

`width` the width of the formatted matrix.

`transposedValm` determines if `valm` is transposed with respect to a typical matrix of performance measures of type `pm`.

`description` the description for the formatted matrix.

Returns `true` if and only if the sheet was modified.

```
public boolean formatStatisticsSummary (ContactCenterSim sim, double level,
                                        String description,
                                        PerformanceMeasureType... pms)
                                         throws WriteException
```

Adds a statistical report for all the performance measures in `pms` supported by the contact center simulator `sim` to the current spreadsheet. This is similar to `formatValuesSummary (ContactCenterEval, String, PerformanceMeasureType...)`, with additional statistical information such as miminum, maximum, standard deviation, and confidence intervals with confidence level `level`, if available.

Parameters

`sim` the contact center simulator.

`level` the confidence level of the confidence intervals.

`pms` the array of performance measures.

Returns `true` if and only if the sheet was modified.

```
public boolean formatStatisticsDetailed (ContactCenterSim sim, double
                                         level, PerformanceMeasureType pm)
                                          throws WriteException
```

Adds a statistical report for all the values of the performance measure `pm` estimated by the simulator `sim`, with confidence intervals with level `level`, to the current spreadsheet.

Parameters

`sim` the contact center simulator.

`level` the level of confidence intervals.

`pm` the performance measure of interest.

Returns `true` if and only if the sheet was modified.

```
public boolean formatStatisticsDetailedHidePeriods (ContactCenterSim sim,
                                                    double level,
                                                    PerformanceMeasureType
                                                    pm) throws
                                                    WriteException
```

Similar to `formatStatisticsDetailedMatrix (ContactCenterSim, double, PerformanceMeasureType)` but does not format per-period statistics.

Parameters

`sim` the contact center simulator.

`level` the confidence level of the intervals.

`pm` the type of performance measures.

Returns `true` if and only if the sheet was modified.

`public boolean formatInfo (Map<String, Object> info) throws WriteException`

Appends rows containing the evaluation information `info` to the current spreadsheet. For each entry in the given map, this method creates a row containing one cell for the key, and a second cell for the value. Values are formatted as follows. Any `null` reference becomes the string `null`, instances of `Number` are turned into numeric cells, and instances of `Date` are converted to numeric cells formatted as dates. Any other non-`null` value is formatted using the `Object.toString()` method to become a string cell.

Parameter

`info` the evaluation information.

Returns `true` if and only if the sheet was modified.

`public boolean formatObservations (ContactCenterSimWithObservations sim, ReportParams reportParams) throws WriteException`

For each element `PerformanceMeasureParams` returned by `ReportParams.getPrintedObs()`, formats the complete list of observations generated by the simulator `sim` for the referred performance measure. Each performance measure is formatted as a separate column in the current sheet.

Parameters

`sim` the simulator to get observations from.

`reportParams` the report parameters.

Returns `true` if the current sheet is modified by this operation.

Throws

`WriteException` if an error happens when writing to the current sheet.

`public boolean formatValues (ContactCenterEval eval, ReportParams reportParams) throws WriteException`

Formats a workbook containing the report of the last evaluation performed by the system `eval`. This method can be called by the implementation of `ContactCenterEval.formatStatisticsExcel (WritableWorkbook)`.

Assuming that the current workbook is empty, this method creates a new sheet with name given by `summarySheetName`, and calls `formatInfo (Map)` with the evaluation information of `eval` to add the information to the new sheet. It then formats a summary report using `formatValuesSummary (ContactCenterEval, String, PerformanceMeasureType. ..)` into the same sheet. The method then creates a second sheet with name given by `detailedSheetNameWithoutPeriods` for the detailed report. For each performance measure a detailed report is reuqested for, the method calls `formatValuesDetailedHidePeriods (ContactCenterEval, PerformanceMeasureType)`. Finally, the method creates a third sheet with name given by `detailedSheetNameWithPeriods` and containing detailed information with individual periods, using `formatStatisticsDetailed (ContactCenterSim,`

`double, PerformanceMeasureType)` to create the new cells. The two last sheets are not created if they would be empty. The creation of any of the third sheet can be disabled by giving a `null` or empty name for that sheet. Information that would be presented on an omitted sheet is written on the next requested sheet.

The types of performance measures to include in the report are selected using `printedStats` which can be `null` or empty; in these two latter cases, the report includes all performance measures supported by `eval`. Each element of `printedStats` specifies a type of performance measure to include in the report (if supported by `eval`), whether a detailed report must be included, and if this detailed report includes information about each individual period.

Parameters

`eval` the evaluation system.

`reportParams` the report parameters.

```
public boolean formatStatistics (ContactCenterSim sim, ReportParams
                                 reportParams) throws WriteException
```

Similar to `formatValues (ContactCenterEval, ReportParams)`, except this formats a full statistical report using `formatStatisticsSummary (ContactCenterSim, double, String, PerformanceMeasureType...)`, and `formatStatisticsDetailed (ContactCenterSim, double, PerformanceMeasureType)`. If the given report parameters contains information about observations to print, the method also calls `formatObservations (ContactCenterSimWithObservations, ReportParams)`, and creates a sheet with the results.

Parameters

`sim` the contact center simulator.

`reportParams` the report parameters.

Returns the current workbook.

EvalOptionType

Represents an evaluation option type for a contact center evaluation system. An evaluation option is an external parameter that can be changed from evaluations to evaluations. In particular, it can be a decision variable or a simulation stopping condition. An object of this class must be passed as a key to `ContactCenterEval.setEvalOption (EvalOptionType, Object)` to select which evaluation option to modify.

```
package umontreal.iro.lecuyer.contactcenters.app;

public enum EvalOptionType
```

Constants

`STAFFINGVECTOR`

Corresponds to an array of integers giving the number of agents in each group and main period. For a single-period or stationary simulation, the length of the array corresponds to I and element i gives the number of agents in group i. For a multi-periods simulation, the length must be IP, and for a period p and the agent group i, the number of agents is given by the element with index $Pi + p$. For example, in a simulation with two agent groups and two main periods, the vector $\{1, 2, 3, 4\}$ would set the number of agents in group 0 to 1 for the first main period and 2 for the second one.

`STAFFINGMATRIX`

Corresponds to a 2D array of integers giving the number of agents in each group and main period. Element (i, p) of this array gives the number of agents in group i during main period p. If a single period is simulated as if it was infinite in the model, the matrix has a single column. Otherwise, it has P columns.

`SCHEDULEDAGENTS`

Corresponds to a 2D array of integers giving the number of agents in each shift for each agent group. Element (i, j) of the 2D array gives the number of agents in shift j for agent group i. If agent group i does not use a schedule, the corresponding array in the 2D array is `null`.

`QUEUECAPACITY`

Corresponds to an integer giving the maximal capacity of the waiting queue. An infinite capacity is represented by the value `Integer.MAX_VALUE`.

`SIMSTOPPINGCONDITION`

Can be used to define an additionnal stopping condition for a simulation. By default, a simulator stops the simulation when some conditions apply, e.g., a fixed simulation length or a target relative error on some predetermined performance measures. This option can

be used to add a user-defined stopping condition which will be checked in addition to the default conditions. The value of this option can be `null` (the default) or a reference to a `SimStoppingCondition` object.

`CURRENTPERIOD`

This integer can be used to set the current period for a multi-period model evaluated in a single period, as if this period length was infinite.

Methods

`public String getName()`

Returns the name of this evaluation option.

Returns the name of the option.

`public Class<?> getType()`

Returns the class of the objects representing values for this option.

Returns the type of the option.

ContactCenterEvalResults

Contains results obtained by another contact center evaluation system. This class can be used to store the results produced by a simulator, to retrieve them efficiently, or to serialize them into an XML file. Results can be retrieved by using the methods of `ContactCenterEval`. The implementation of the methods of the interface that perform evaluations throws an `UnsupportedOperationException`.

An object from this class can be constructed using any implementation of `ContactCenterEval`, or from an instance of `ContactCenterEvalResultsParams`. The `createParams()` method can be used to turn any instance of this class into a parameter object of class `ContactCenterEvalResultsParams`.

The class `ContactCenterEvalResultsConverter` provides convenience method to create an object containing results from an XML file, or to export an object of this class into XML. One can also use `PerformanceMeasureFormat.formatResults (ContactCenterEval, File)` to export simulation results to a file.

```
package umontreal.iro.lecuyer.contactcenters.app;

public class ContactCenterEvalResults implements ContactCenterEval
```

Constructors

`public ContactCenterEvalResults (ContactCenterEvalResultsParams ccp)`

Constructs a new object containing results read from the parameter object `ccp`. It is not recommended to use this constructor directly; one should use the `createFromParams (ContactCenterEvalResultsParams)` method to create instances of this class.

Parameter

`ccp` the contact centers results.

`public ContactCenterEvalResults (ContactCenterEval eval)`

Constructs a new contact center results container by getting evaluation results from the system `eval`. It is not recommended to use this constructor directly; one should call `createFromEval (ContactCenterEval)` instead.

Parameter

`eval` the contact center evaluator.

Methods

`public void writeParams (ContactCenterEvalResultsParams ccp)`

Fills `ccp` with parameters stored in this object. This method is rarely used directly; one should call `createParams()` instead.

Parameter

`ccp` the output parameter object.

`public static ContactCenterEvalResults createFromEval (ContactCenterEval eval)`

Constructs a new object storing the last results produced by the evaluation system `eval`. The resulting object is completely independent of `eval`. As a result, this method can be used to take snapshots of evaluation results, and store them, e.g., to compare different scenarios.

If `eval` is an instance of `ContactCenterSim`, this method returns an instance of `ContactCenterSimResults`. Otherwise, this returns an instance of `ContactCenterEvalResults`.

Parameter

`eval` the source evaluation system.

Returns an object containing a copy of the evaluation results.

`public static ContactCenterEvalResults createFromParams (ContactCenterEvalResultsParams ccp)`

Parses the given parameter object `ccp`, and constructs an object containing evaluation results. The class of `ccp`, `ContactCenterEvalResultsParams`, was generated using JAXB, so it does not implement the `ContactCenterEval` interface. Usually, `ccp` is constructed by using JAXB. This method reads the parameters of `ccp`, and stores them in such a way that they can be accessed efficiently using methods of `ContactCenterEval`. The resulting object is an instance of `ContactCenterSimResults` if `ccp` is an instance of `ContactCenterSimResultsParams`.

Parameter

`ccp` the parameter object containing results.

Returns the object storing results.

`public static ContactCenterEvalResults createFromParams (ContactCenterEvalResultsParams ccp, boolean reportPropertiesToEvalInfo)`

Similar to the method `createFromParams (ContactCenterEvalResultsParams)`, but if the flag `reportPropertiesToEvalInfo` is set to `true`, copies the report properties into the evaluation information. In the early version of the output format, evaluation information, e.g., properties specific to a given experiment, were stored into the properties of the `report` child element. An `evalInfo` element has recently been added to store the evaluation information, reserving report properties for user-defined options specific to reporting. This method is provided to read old-style output files, and should not be used for newer output files.

Parameters

`ccp` the contact center parameters.

`reportPropertiesToEvalInfo` determines if report properties are copied to evaluation information.

Returns the contact center evaluation results.

`public ContactCenterEvalResultsParams createParams()`

Creates a parameter object that can be marshalled using JAXB from this object, and copies its evaluation results..

Returns the parameter object containing results.

ContactCenterSimResults

Extends `ContactCenterEvalResults` to store additional information related to the simulation of a call center.

```
package umontreal.iro.lecuyer.contactcenters.app;

public class ContactCenterSimResults extends ContactCenterEvalResults
             implements ContactCenterSim
```

Constructors

`public ContactCenterSimResults (ContactCenterSimResultsParams ccp)`

Constructs a new object containing results read from the parameter object `ccp`. It is not recommended to use this constructor directly; one should use the `ContactCenterEvalResults.createFromParams (ContactCenterEvalResultsParams)` method to create instances of this class.

Parameter

`ccp` the contact centers results.

`public ContactCenterSimResults (ContactCenterSim sim)`

Constructs a new contact center results container by getting simulation results from the simulator `sim`, and compoting confidence intervals with level `level`.

Parameter

`sim` the contact center simulator.

ContactCenterEvalResultsConverter

Converter for marshalling and unmarshalling objects containing evaluation results.

```
package umontreal.iro.lecuyer.contactcenters.app;

public class ContactCenterEvalResultsConverter extends JAXBParamsConverter<
            ContactCenterEvalResultsParams>
```

Methods

```
public ContactCenterEvalResults unmarshalToEval (File file) throws
                                                    JAXBException
```

Reads evaluation results from the file `file`. This method uses `JAXBParamsConverter.unmarshal (File)` to unmarshal the given file to a parameter object, and `ContactCenterEvalResults.createFromParams (ContactCenterEvalResultsParams)` to create the final object containing results.

Parameter

`file` the input file.

Returns the evaluation results.

Throws

`JAXBException` if an exception occurs during unmarshalling.

```
public ContactCenterEvalResults unmarshalToEval (File file, boolean
                                                  reportPropertiesToEvalInfo)
                                                   throws JAXBException
```

Similar to `unmarshalToEval (File)`, but uses `ContactCenterEvalResults.createFromParams (ContactCenterEvalResultsParams, boolean)` instead of `ContactCenterEvalResults.createFromParams (ContactCenterEvalResultsParams)`.

Parameters

`file` the input file

`reportPropertiesToEvalInfo` determines if report properties are copied to evaluation information.

Returns the evaluation results.

Throws

`JAXBException` if an exception occurs during unmarshalling.

```
public ContactCenterEvalResults unmarshalGZippedToEval (File file) throws
                                                          JAXBException
```

Similar to `unmarshalToEval (File)`, but calls `JAXBParamsConverter.unmarshalGZipped (File)` for unmarshalling.

Parameter

`file` the input file.

Returns the evaluation results.

Throws

`JAXBException` if an error occurs during the unmarshalling.

```
public ContactCenterEvalResults unmarshalGZippedToEval (File file, boolean
                                                        reportPropertiesToEvalInfo)
                                                         throws
                                                        JAXBException
```

Similar to `unmarshalToEval (File, boolean)`, but calls `JAXBParamsConverter.unmarshalGZipped (File)` for unmarshalling.

Parameters

`file` the input file

`reportPropertiesToEvalInfo` determines if report properties are copied to evaluation information.

Returns the evaluation results.

Throws

`JAXBException` if an error occurs during the unmarshalling.

```
public ContactCenterEvalResults unmarshalGZippedToEval (URL url) throws
                                                        JAXBException
```

Similar to `unmarshalToEval (URL)`, but calls `JAXBParamsConverter.unmarshalGZipped (URL)` for unmarshalling.

Parameter

`url` the input URL.

Returns the evaluation results.

Throws

`JAXBException` if an error occurs during the unmarshalling.

```
public ContactCenterEvalResults unmarshalGZippedToEval (URL url, boolean
                                                        reportPropertiesToEvalInfo)
                                                         throws
                                                        JAXBException
```

Similar to `unmarshalToEval (URL, boolean)` but calls `JAXBParamsConverter.unmarshalGZipped (URL)` for unmarshalling.

Parameters

`url` the input URL.

`reportPropertiesToEvalInfo` determines if report properties are copied to evaluation information.

Returns the evaluation results.

Throws

`JAXBException` if an error occurs during the unmarshalling.

```
public ContactCenterEvalResults unmarshalToEval (URL url) throws
                                                    JAXBException
```

Similar to `unmarshalToEval (File)`, with an URL instead of a file.

Parameter

`url` the URL of the XML data.

Returns the constructed object containing results.

Throws

`JAXBException` if an exception occurs during unmarshalling.

```
public ContactCenterEvalResults unmarshalToEval (URL url, boolean
                                                   reportPropertiesToEvalInfo)
                                                    throws JAXBException
```

Similar to `unmarshalToEval (File, boolean)`, with an URL instead of a file.

Parameters

`url` the input URL of the XML data.

`reportPropertiesToEvalInfo` determines if report properties are copied to evaluation information.

Returns the evaluation results.

Throws

`JAXBException` if an error occurs during unmarshalling.

```
public ContactCenterEvalResults unmarshalToEval (Node node) throws
                                                    JAXBException
```

Similar to `unmarshalToEval (File)`, with a node instead of a file.

Parameter

`node` the input node.

Returns the constructed object containing results.

Throws

`JAXBException` if an exception occurs during unmarshalling.

```
public ContactCenterEvalResults unmarshalToEval (Node node, boolean
                                                  reportPropertiesToEvalInfo)
                                                   throws JAXBException
```

Similar to `unmarshalToEval (File, boolean)` with a node instead of a file.

Parameters

`node` the input node.

`reportPropertiesToEvalInfo` determines if report properties are copied to evaluation information.

Returns the constructed object containing results.

Throws

`JAXBException` if an exception occurs during unmarshalling.

```
public ContactCenterEvalResults unmarshalToEval (Source source) throws
                                                  JAXBException
```

Similar to `unmarshalToEval (File)`, with a source instead of a file.

Parameter

`source` the input source to read XML from.

Returns the constructed object containing results.

Throws

`JAXBException` if an exception occurs during unmarshalling.

```
public ContactCenterEvalResults unmarshalToEval (Source source, boolean
                                                  reportPropertiesToEvalInfo)
                                                   throws JAXBException
```

Similar to `unmarshalToEval (File, boolean)`, with a source instead of a file.

Parameters

`source` the input source to read XML from.

`reportPropertiesToEvalInfo` determines if report properties are copied to evaluation information.

Returns the constructed object containing results.

Throws

`JAXBException` if an exception occurs during unmarshalling.

`public ContactCenterEvalResults unmarshalToEvalOrExit (File file)`

Similar to `unmarshalToEval (File)`, but calls `JAXBParamsConverter.unmarshalOrExit (File)` to perform unmarshalling. This method should be used in simple programs with no graphical user interface, because it can print information on the standard output, and exit the program if an error occurs.

Parameter

`file` the input file.

Returns the constructed object containing results.

`public ContactCenterEvalResults unmarshalToEvalOrExit (File file, boolean reportPropertiesToEvalInfo)`

Similar to `unmarshalToEval (File, boolean)`, but calls `JAXBParamsConverter.unmarshalOrExit (File)` to perform unmarshalling.

Parameters

`file` the input file.

`reportPropertiesToEvalInfo` determines if report properties are copied to evaluation information.

Returns the constructed object containing results.

`public void marshalEval (ContactCenterEvalResults res, ContentHandler handler) throws JAXBException`

Marshals an object containing evaluation results into an XML document, and writes the resulting output to the content handler `handler`. This method uses `ContactCenterEvalResults.createParams()` to create a parameter object from the evaluation results, and uses `JAXBParamsConverter.marshal (T, ContentHandler)` to marshal the parameters to XML.

Parameters

`res` the object containing evaluation results.

`handler` the target content handler receiving XML events.

Throws

`JAXBException` if an exception occurs during marshalling.

`public void marshalEval (ContactCenterEvalResults res, Result result) throws JAXBException`

Similar to `marshalEval (ContactCenterEvalResults, ContentHandler)`, but uses `JAXBParamsConverter.marshal (T, Result)` for marshalling instead.

Parameters

`res` the object containing evaluation results.

`result` the output result for the XML contents.

Throws

`JAXBException` if an error occurs during marshalling.

```
public void marshalEval (ContactCenterEvalResults res, File file) throws
                         JAXBException
```

Similar to `marshalEval (ContactCenterEvalResults, ContentHandler)`, but uses `JAXBParamsConverter.marshal (T, File)` for marshalling into a file instead.

Parameters

`res` the object containing evaluation results.

`file` the output file.

Throws

`JAXBException` if an error occurs during marshalling.

```
public void marshalEvalAndGZip (ContactCenterEvalResults res, File file)
                                throws JAXBException
```

Similar to `marshalEval (ContactCenterEvalResults, File)`, but calls `JAXBParamsConverter.marshalAndGZip (T, File)`.

Parameters

`res` the object to be marshalled.

`file` the output file.

Throws

`JAXBException` if an error occurs during the process.

```
public void marshalEval (ContactCenterEvalResults res, Node node) throws
                         JAXBException
```

Similar to `marshalEval (ContactCenterEvalResults, ContentHandler)`, but uses `JAXBParamsConverter.marshal (T, Node)` for marshalling instead.

Parameters

`res` the object containing evaluation results.

`node` the output node.

Throws

`JAXBException` if an error occurs during marshalling.

`public void marshalEvalOrExit (ContactCenterEvalResults res, File file)`

Similar to `marshalEval (ContactCenterEvalResults, File)`, but uses `JAXBParamsConverter.marshalOrExit (T, File)` for marshalling. This method should be used in simple programs with no graphical user interface, because it can print information on the standard output, and exit the program if an error occurs.

Parameters

`res`

`file`

`public void marshalEvalAndGZipOrExit (ContactCenterEvalResults res, File file)`

Similar to `marshalEvalOrExit (ContactCenterEvalResults, File)`, but calls `JAXBParamsConverter.marshalAndGZipOrExit (T, File)`.

Parameters

`res` the object to be marshalled.

`file` the output file.

AbstractContactCenterInfo

Provides default implementations for some methods in `ContactCenterInfo`. Implemented methods giving strings return `null`, methods giving integers return 0, and methods giving properties return empty maps. Method `getNumContactTypesWithSegments()` and other similar methods return the number of contact types plus the number of segments regrouping contact types.

```
package umontreal.iro.lecuyer.contactcenters.app;

public abstract class AbstractContactCenterInfo implements ContactCenterInfo
```

AbstractContactCenterEval

Defines basic methods to implement a contact center evaluation system.

```
package umontreal.iro.lecuyer.contactcenters.app;

public abstract class AbstractContactCenterEval extends
                      AbstractContactCenterInfo
                      implements ContactCenterEval
```

AbstractContactCenterSim

Helper class to implement a contact center simulator.

```
package umontreal.iro.lecuyer.contactcenters.app;

public abstract class AbstractContactCenterSim extends
                      AbstractContactCenterEval
                      implements ContactCenterSim
```

Field

`protected boolean autoResetStartStream`

Determines if random streams are automatically reset.

Method

`public boolean hasPerformanceMeasure (PerformanceMeasureType m)`

Calls `ContactCenterEval.getPerformanceMeasures()` and searches for `m` in the returned array.

Parameter

`m` the performance measure being tested.

Returns a `true` value if the measure is supported, `false` otherwise.

Throws

`NullPointerException` if `m` is `null`.

RouterPolicyType

Represents the type of router's policies supported by blend/multi-skill call center simulations. This policy determines how the router assigns an agent to incoming calls and how free agents look for queued calls.

```
package umontreal.iro.lecuyer.contactcenters.app;

public enum RouterPolicyType
```

Constants

`QUEUEPRIORITY`

Queue priority routing policy. See `QueuePriorityRouter` for more information. This routing policy requires a type-to-group and a group-to-type maps.

`QUEUEATLASTGROUP`

Queue at last group routing policy. See `QueueAtLastGroupRouter` for more information. This routing policy requires a type-to-group map only.

`LONGESTQUEUEFIRST`

Longest queue first routing policy. See `LongestQueueFirstRouter` for more information. This routing policy requires a type-to-group and a group-to-type maps. Since the group-to-type map is used as a tie breaker only, it is not as important as with the queue priority routing policy, but it must be specified as well.

`SINGLEFIFOQUEUE`

Single FIFO queue router policy. See `SingleFIFOQueueRouter` for more information. This routing policy requires a type-to-group and a group-to-type maps. Since the group-to-type map is used as a tie breaker only, it is not as important as with the queue priority routing policy, but it must be specified as well.

`LONGESTWEIGHTEDWAITINGTIME`

Longest weighted waiting time router policy. See `LongestWeightedWaitingTimeRouter` for more information. This routing policy requires a type-to-group, a group-to-type maps, and an array of weights. Since the group-to-type map is used as a tie breaker only, it is not as important as with the queue priority routing policy, but it must be specified as well.

`AGENTSPREF`

Agents' preference-based routing policy. See `AgentsPrefRouter` for more information.

`AGENTSPREFWITHDELAYS`

Agents' preference-based routing policy with delays. See `AgentsPrefRouterWithDelays` for more information.

`LOCALSPEC`

Local-specialist routing policy. See `LocalSpecRouter` for more information. To use this router, one must specify contact type and agent group names containing a region name. If available, this router uses the matrices of ranks to select agents and waiting queues.

`QUEUERATIOOVERFLOW`

Overflow routing policy using queue ratio. See `QueueRatioOverflowRouter` for more information. This routing policy requires a contacts-to-agents matrix of ranks.

`EXPDELAY`

Routing policy using expected delay to select agent groups. See `ExpDelayRouter` for more information. This routing policy requires a contacts-to-agents weights matrix.

`OVERFLOWANDPRIORITY`

Routing policy based on overflow and priority. See `OverflowAndPriorityRouter` for more information.

The $f_{k,j}$ and $g_{k,j}$ functions are defined using sequences of triplets $(C_{k,j,l}, A_{k,j,l}, Q_{k,j,l})$, where $C_{k,j,l}$ represents a condition, and $A_{k,j,l}$ and $Q_{k,j,l}$ are vectors. First, the condition $C_{k,j,0}$ is checked. If it is true, $A_{k,j,0}$ is used as the vector of ranks for agent groups, and $Q_{k,j,0}$ is used to set up priorities for queues. Otherwise, the condition $C_{k,j,1}$ is checked, and the corresponding vectors $A_{k,j,1}$, and $Q_{k,j,1}$ are used if the condition is true. This check continues for other conditions $C_{k,j,2}, C_{k,j,3}, \ldots$, until a true condition is found, or the list of cases is exhausted. If a condition $C_{k,j,l}$ applies, and no vectors of ranks are associated with this condition, the last vectors of ranks are preserved, i.e., the jth routing stage has no effect. If the list of cases is exhausted without finding an applicable condition, a default set of vectors of ranks is used. If no such default vectors are given, the routing stage j has no effect.

A vector of ranks can also be set relatively to the preceding vector. When a relative vector is used, the ranks are summed with the previous ranks, which allows to update ranks rather than overriding them. This can be useful to accumulate the effect of several conditions at different stages of routing, e.g., increase priority at queue 1 depending on its size, decrease priority at queue 2 depending on the number of free agents, etc.

This policy requires a script for each call type in the parameter file for the call center. Such a script is set up by a `callTypeRouting` element. Such an element contains one ore more `stage` children describing the stages of routing. A `stage` element has an attribute `waitingTime` giving the waiting time $w_{k,j}$ as well as a sequence of `case` elements for the cases, and a `default` element for the default vectors of ranks. See the complex type `CallTypeRoutingParams`, in the HTML documentation of the XML Schema for the complete syntax of routing parameters, including how to encode conditions.

ArrivalProcessType

Represents the type of arrival process for a blend/multi-skill call center. This process defines at which times a new call occurs during the simulation. The used arrival process determines how the parameters, usually given using an array of double-precision values, are used. All the arrival processes defined in [1] are supported.

Note that when simulating on an infinite horizon, only arrival processes capable of generating arrivals following parameters not evolving with time are allowed. The recommended arrival processes for such simulations are `POISSON` and `PIECEWISECONSTANTPOISSON`.

If the busyness generator for B is undefined, then B is replaced by a deterministic constant = 1.

```
package umontreal.iro.lecuyer.contactcenters.app;

public enum ArrivalProcessType
```

Constants

`POISSON`

Poisson arrival process. See `PoissonArrivalProcess` for more information. Inter-arrival times are generated independently from the exponential distribution with fixed rate $B\lambda$, where λ is the base arrival rate given by the first value of the `arrivals` element, and B is a global busyness factor given by `busynessGen` element in call center parameters. The λ parameter can be estimated from data.

`PIECEWISECONSTANTPOISSON`

Non-homogeneous Poisson arrival process with piecewise-constant arrival rates. See `PiecewiseConstantPoissonArrivalProcess` for more information. Inter-arrival times are independent exponential variates with rate $\lambda(t) = B\lambda_{p(t)}$, where $p(t)$ is the period corresponding to simulation time t, and λ_p is the base arrival rate for period p. The values in `arrivals` element give the base arrival rate for each main period, while the rates for preliminary and wrap-up periods are always 0. The λ_p parameters can be estimated from data by assuming that the periods are independent, and the per-period numbers of arrivals follow the Poisson distribution. The arrival rates can also be estimated together with a busyness factor following the gamma(α_0, α_0) distribution. In this case, the number of arrivals is assumed to follow the negative multinomial distribution with parameters $\alpha_0, \rho_1, \ldots, \rho_P$, where

$$\rho_p = \frac{\lambda_p}{\alpha_0 + \sum_{k=1}^{P} \lambda_k}$$

for $p = 1, \ldots, P$. The above method for the λ_p is equivalent to

$$\lambda_p = \frac{1}{n} \sum_{k=1}^{P} X_{k,p},$$

where n is the number of days.

`PIECEWISECONSTANTPOISSONINT`

Non-homogeneous Poisson arrival process with piecewise-constant arrival rates that can change at arbitrary times. See `PoissonArrivalProcessWithTimeIntervals` for more information. This process is similar to `PIECEWISECONSTANTPOISSON`, except arrival rates can change at any time, not only at period boundaries. More specifically, let $t_0 < \cdots < t_L$ be an increasing sequence of simulation times, and let $B\lambda_j$, for $j = 0, \ldots, L-1$, be the arrival rate during time interval $[t_j, t_{j+1})$. The arrival rate is 0 for $t < t_0$ and $t \geq t_L$. The sequence of times is given using the `times` element while the sequence of rates is given using `lambdas`. Of course, the length of the sequence of rates must be one less than the length of the the sequence of times.

`UNIFORM`

Uniform arrival process with piecewise-constant arrival rates. See `PoissonUniformArrivalProcess` for more information. For each main period $p = 1, \ldots, P$, round$(B\lambda_p)$ arrivals are generated uniformly in the period, and arrival times are sorted in increasing order. The values in the `arrivals` element give the base arrival rate λ_p for each main period, while the rates for preliminary and wrap-up periods are always 0. The λ_p parameters can be estimated from data by assuming that the periods are independent, and the per-period numbers of arrivals follow the Poisson distribution.

`FIXEDCOUNTS`

Uniform arrival process with pre-determined arrival counts C_p in each period. Represents a counts-driven arrival process. See `FixedCountsArrivalProcess` for more information. For each main period $p = 1, \ldots, P$, the C_p arrivals are generated uniformly in the period, and arrival times are sorted in increasing order. The values in the `counts` element give the number of arrivals for each main period, while the counts for preliminary and wrap-up periods are always 0.

`POISSONGAMMA`

Poisson process with piecewise-constant randomized arrival rates [2]. See `PoissonGammaArrivalProcess` for more information. As with `PIECEWISECONSTANTPOISSON`, the arrival rate is given by $\lambda(t) = B\lambda_{p(t)}$. The base arrival rates λ_p are constant during each main period, but they are not deterministic: for main period p, the base rate of the Poisson process is defined as a gamma random variable with shape parameter $\alpha_{\mathrm{G},p}$, and scale parameter $\lambda_{\mathrm{G},p}$ (mean $\alpha_{\mathrm{G},p}/\lambda_{\mathrm{G},p}$). Shape parameters are stored in element `poissonGammaShape` while scale parameters are stored in `poissonGammaScale`. As with the Poisson process with deterministic arrival rates, the generated base arrival rates are multiplied by a busyness factor B to get the arrival rates, and the arrival rate is 0 during preliminary and wrap-up periods. The $\alpha_{\mathrm{G},p}$ and $\lambda_{\mathrm{G},p}$ parameters can be estimated from data by considering that the number of arrivals during period p follow the negative binomial distribution with parameters $(\alpha_{\mathrm{G},p}, \lambda_{\mathrm{G},p}/(\alpha_{\mathrm{G},p} + \lambda_{\mathrm{G},p}))$, independently of the other periods. However, the parameters of the distribution of the busyness factor cannot be estimated at the same time as the gamma parameters.

`POISSONGAMMANORTARATES`

Doubly-stochastic Gamma-Poisson process with piecewise-constant randomized correlated Gamma arrival rates. See `PoissonGammaNortaRatesArrivalProcess` for more information.

The base arrival rates λ_p are constant during each period, but they are not deterministic: for period p, the base rate of the Poisson process is defined as a correlated gamma random variable. The marginal distribution of the rate is gamma with shape parameter $\alpha_{G,p}$, and scale parameter $\lambda_{G,p}$ (and mean $\alpha_{G,p}/\lambda_{G,p}$). The correlation structure is modelled using the normal copula model with positive definite correlation matrix $\mathbf{\Sigma}$ having elements in $[-1, 1]$. If $\alpha_{G,p}$ or $\lambda_{G,p}$ are 0, the resulting arrival rate during period p is always 0. The Gamma shape parameters are stored in element `poissonGammaShape` while the Gamma scale parameters are stored in `poissonGammaScale`. A correlation matrix must be given by `copulaSigma`. As with the Poisson process with deterministic arrival rates, the generated base arrival rates are multiplied by a busyness factor B to get the arrival rates, and the arrival rate is 0 during preliminary and wrap-up periods. The parameters of the distribution of the busyness factor cannot be estimated at the same time as the gamma parameters.

`DIRICHLETCOMPOUND`

Dirichlet compound arrival process. See `DirichletCompoundArrivalProcess` for more information. The values in `arrivals` element are used to store the α_p parameters. These parameters, along with the γ parameter of the busyness factor, can be estimated from the data.

`DIRICHLET`

Dirichlet arrival process. See `DirichletArrivalProcess` for more information.

The values in `arrivals` element are used for the α_p parameters. If a busyness factor B is generated for the day, the generated A_p's are multiplied by B and rounded to the nearest integer to get the modified number of arrivals. The parameters α_p can be estimated from data, but one needs to specify a distribution for A to estimate parameters from. This arrival process cannot estimate parameters of the busyness factor.

`NORTADRIVEN`

Represents a NORTA-driven arrival process with negative binomial marginals. See `NORTADriven-ArrivalProcess` for more information.

To use this process, a correlation matrix must be given by `nortaSigma`, and parameters for the negative binomials must be supplied by `nortaGamma` and `nortaP`. If a busyness factor B is generated for the day, the generated A_p's are multiplied by B and rounded to the nearest integer to get the modified number of arrivals. This arrival process does not support parameter estimation from data.

`CUBICSPLINE`

Non-homogenous Poisson arrival process using a cubic spline to model the time-varying arrival rate. The $\lambda(t)$ function which represents the arrival rate at any time t is a cubic spline created from a sequence of n points $(t_i, \lambda(t_i))$ also called *nodes*. A *cubic spline* is a set of cubic polynomials linked by some continuity constraints. The ith polynomial of such a spline, for $i = 0, \ldots, n-2$, is defined as

$$s_i(t) = a_i(t - t_i)^3 + b_i(t - t_i)^2 + c_i(t - t_i) + d_i,$$

while the complete spline is defined as

$$s(t) = s_i(t) \text{ for } t \in [t_i, t_{i+1}].$$

For $t < t_0$ and $t > t_{n-1}$, the spline is undefined, but the implementation performs linear extrapolation.

Interpolating splines are forced to pass through every point, i.e., $s_i(t_i) = \lambda(t_i)$ for $i = 0, \ldots, n-2$, and $s_{n-2}(t_{n-1}) = \lambda(t_{n-1})$. On the other hand, *smoothing* splines tolerate some error, i.e., the spline minimizes

$$L = \rho \sum_{i=0}^{n-1} \left(\lambda(t_i) - s(t_i)\right)^2 + (1-\rho) \int_{t_0}^{t_{n-1}} \left(s''(x)\right)^2 dx.$$

The value ρ in previous equation is the *smoothing factor* of the spline.

Both kinds of splines enforce the three following continuity constraints:

$$\begin{aligned} s_i(t_{i+1}) &= s_{i+1}(t_{i+1}), \\ \frac{d}{dt} s_i(t_{i+1}) &= \frac{d}{dt} s_{i+1}(t_{i+1}), \\ \text{and} \quad \frac{d^2}{dt^2} s_i(t_{i+1}) &= \frac{d^2}{dt^2} s_{i+1}(t_{i+1}), \qquad \text{for } i = 0, \ldots, n-2. \end{aligned}$$

The cubic splines used are named *natural splines*, because $\frac{d^2}{dt^2} s(t_0) = \frac{d^2}{dt^2} s(t_{n-1}) = 0$.

To use this arrival process, one must specify the t_i's with the `times` element, the $\lambda(t_i)'s$ with `lambdas` element, and the smoothing factor with `smoothingFactor` attribute. For this arrival process, `times` and `lambdas` must share the same length.

DialerPolicyType

Represents the dialer policy specifying when a dialer must try to make calls and how many calls to try at a time. Some of the policies need parameters which are specified as part of the dialer parameters.

```
package umontreal.iro.lecuyer.contactcenters.app;

public enum DialerPolicyType
```

Constants

`DIALXFREE`

Dials only when the total number of free agents $N_{\mathrm{F}}^{\mathrm{T}}(t)$ in all agent groups is greater than or equal to the minimum $s_{\mathrm{t},k}(t)$, and the number of free agents $N_{\mathrm{F},k}^{\mathrm{D}}(t)$ capable of serving the dialed call type is greater than or equal to $s_{\mathrm{d},k}(t)$. The thresholds do not change during main periods, but they can change from period to period. If dialing is performed, $\mathrm{round}(\kappa N_{\mathrm{F},k}^{\mathrm{D}}(t)) + c$ outbound calls are produced. κ and c corresponds to predefined constants, and $\mathrm{round}(\cdot)$ corresponds to $\cdot$ rounded to the nearest integer.

`DIALONE`

Equivalent to `DIALXFREE` with $\kappa = 0$ and $c = 1$.

`DIAL1XFREE`

Equivalent to `DIALXFREE` with $\kappa = 1$ and $c = 1$.

`DIAL2XFREE`

Equivalent to `DIALXFREE` with $\kappa = 2$ and $c = 0$.

`DIALFREE_BADCALLMISMATCHRATES`

When the dialing conditions defined for `DIALXFREE` apply, i.e., $N_{\mathrm{F}}^{\mathrm{T}}(t) \geq s_{\mathrm{t},k}(t)$ and $N_{\mathrm{F},k}^{\mathrm{D}}(t) \geq s_{\mathrm{d},k}(t)$, and the rate of inbound calls of any type waiting more than the acceptable waiting time is smaller than a threshold, dials some calls. Let $d = \mathrm{round}(\kappa N_{\mathrm{F},k}^{\mathrm{D}}(t)) + c$. If the mismatch rate for outbound calls of type k being dialed is smaller than a threshold, dials $2d$ calls. Otherwise, dials d.

The number of calls waiting more than the acceptable waiting time and arrivals for all inbound call types, the number of mismatches for call type k, and the total number of tried outbound calls of type k are computed for periods with fixed duration d_{D}. When the dialer is required to take a decision, it computes the bad call and mismatch rates by taking these values during the P_{D} last checked periods.

`AGENTSMOVE`

Dialing policy with smart agent management. See `AgentsMoveDialerPolicy` for more information.

The parameters of the agent groups managed by the dialer are specified using `agentGroupInfo` children elements, in the dialer parameters.

The flags of the dialer are controlled as follows. The dialer keeps track of the global service level (over all inbound call types) for the last P_{D} periods of duration d_{D}. These parameters are set by the attributes `numCheckedPeriods` and `checkedPeriodDuration` of the dialer parameters. If the service level falls below the lower threshold s_1 given by the attribute `slInboundThresh`, the flag outbound-to-inbound is turned on, and inbound-to-outbound is turned off. On the other hand, if the service level goes above the higher threshold s_2 set by the attribute `slOutboundThresh`, the flag inbound-to-outbound is turned on while the flag outbound-to-inbound is off. When the service level is in $[s_1, s_2]$, both flags are turned off.

Package `umontreal.iro.lecuyer.contactcenters.app.trace`

Provides facilities for creating contact-by-contact traces in simulators. For debugging or computing custom statistics such as quantiles, obtaining a trace of every contact generated by a simulator can be useful. This package provides some classes to collect the relevant information about contacts, and store it into a file or database.

The tracing facility is represented by an object implementing the interface `ContactTrace`. Implementations of this interface are available for traces in text files, Excel spreadsheets, or any database for which a JDBC driver is available. The `FileContactTrace` class provides a static convenience method for creating the call trace object from parameters read from an XML file.

To use this facility, a simulator simply needs to create an instance of a class implementing `ContactTrace`, and call the `writeLine` method for every contact. Usually, this is done in an exited-contact listener.

ContactTrace

Represents an object capable of creating a contact-by-contact trace. The format and location of the produced trace depends on the implementation.

The tracing facility is used as follows. An implementation of this interface is initialized at the beginning of the simulation using the `init()` method. Each time a contact is processed, the `writeLine (int, int, int, double, double, String, int, double)` method is called by some listener. At the end of the simulation, the `close()` method is called to close the file or database connection the trace is sent to.

```
package umontreal.iro.lecuyer.contactcenters.app.trace;

public interface ContactTrace
```

Methods

`public void init()`
Initializes the tracing mechanism. This method opens the trace file or database connection, and writes headers, etc. If an error occurs during the initialization, this method should log the error, and disable tracing.

`public void close()`
Closes the trace facility after a simulation. This method closes files, database connections, etc.

```
public void writeLine (int step, int type, int period, double arvTime,
                       double queueTime, String outcome, int group, double
                       srvTime)
```
Writes a new line in the trace representing a simulated contact. The line includes the step of the simulation at which the contact occurred, the type of the contact, the period of its arrival, its time spent in queue, its outcome, the group of its serving agent, and its service time. Some of these fields might be `Double.NaN` if the information does not exist. For example, a blocked or abandoned call does not have a serving agent group, or a service time.

Parameters

`step` the step, in the experiment, during which the call occurred.

`type` the type of the call.

`period` the period of arrival of the call.

`arvTime` the arrival time.

`queueTime` the time spent by the call in the queue.

`outcome` the outcome of the call.

`group` the group of the serving agent.

`srvTime` the service time of the call.

FileContactTrace

Defines an exited-contact listener used to output a trace of every call processed by a simulator into a text file. Each time a new contact is notified to this listener, a line is appended to a writer linked to a file. This results in a call-by-call trace of the simulation. If an I/O exception is thrown at any given time by the writer, the exception's stack trace is logged, and this call tracer is disabled to avoid getting any further exception message.

```
package umontreal.iro.lecuyer.contactcenters.app.trace;

public class FileContactTrace implements ContactTrace
```

Constructor

`public FileContactTrace (File traceFile, int timePrecision)`

Constructs a new call trace sending the information to the text file `traceFile`.

Parameters

`traceFile` the output trace file.

`timePrecision` the number of digits for arrival, waiting, and service times.

Methods

`public static ContactTrace create (CallTraceParams traceParams)`

Creates a new contact trace facility from the given parameters. The class of the returned object depends on the given parameters, which are usually read from an XML file.

More specifically, if these parameters include information on a database connection, a `DBContactTrace` instance is returned. Otherwise, if the name of the output file of the trace ends with `.xls`, an `ExcelContactTrace` is returned. Otherwise, a `FileContactTrace` is returned.

Parameter

`traceParams` the parameters of the trace.

Returns the contact trace facility.

`public String getHeader()`

Returns a string containing the field names for this trace. This string can be written as an header at the beginning of trace files.

Returns the header string.

`public void writeHeader()`

Writes the output of `getHeader()` in the trace file, followed by an end-line character.

ExcelContactTrace

Outputs trace to an Excel spreadsheet using JExcel API.

```
package umontreal.iro.lecuyer.contactcenters.app.trace;

public class ExcelContactTrace implements ContactTrace
```

Constructor

`public ExcelContactTrace (File traceFile, String sheetName)`

Creates a new call trace to a spreadsheet `sheetName` in an Excel file named `traceFile`.

Parameters

`traceFile` the output trace file.

`sheetName` the name of the sheet name containing the trace.

DBContactTrace

Defines an exited-contact listener used to output a trace of every call processed by a simulator into a database. Each time a new contact is notified to this listener, a SQL request is used to update a table through JDBC. This results in a call-by-call trace of the simulation. If an SQL exception is thrown at any given time by the writer, the exception's stack trace is printed, and this call logger is disabled to avoid getting any further exception message.

```
package umontreal.iro.lecuyer.contactcenters.app.trace;

public class DBContactTrace implements ContactTrace
```

Constructor

`public DBContactTrace (DBConnectionParams dbProperties, String dbTable)`

Constructs a new call trace to a database, using the given parameters to establish the connection, and and sending the data to the table with the given name.

Parameters

`dbProperties` the database properties, for `JDBCManager`.

`dbTable` the output table for the trace.

References

[1] A. N. Avramidis, A. Deslauriers, and P. L'Ecuyer. Modeling daily arrivals to a telephone call center. *Management Science*, 50(7):896–908, 2004.

[2] G. Jongbloed and G. Koole. Managing uncertainty in call centers using Poisson mixtures. Manuscript, Vrije University, Amsterdam.