Package 'pdq'

November 25, 2020

Type Package

Title PDQ

Version 7.0-0

Date 2020-11-21

Author Neil Gunther <njgunther@perfdynamics.com>, with contributions from Paul Puglia <pjpuglia@sourceforge.net>

Maintainer Neil Gunther <njgunther@perfdynamics.com>, with contributions from Paul Puglia <pjpuglia@sourceforge.net>

Description
The PDQ performance analyzer uses queueing-theory concepts to represent computer systems and analyze their performance characteristics. It is a queueing model solver, not a simulator. The queueing-theory models discussed in \{ }emph{Analyzing Computer System Performance with Perl::PDQ} are incorporated into the solution methods used by PDQ.

License MIT

URL http://www.perfdynamics.com/Tools/PDQ.html,
https://github.com/DrQz/pdq-qnm-pkg

Depends methods, Rcpp (>= 0.10.2)

LinkingTo Rcpp

NeedsCompilation yes

R topics documented:

`pdq-package` *PDQ (Pretty Damn Quick) Queueing Network Analyzer*

Description

The PDQ performance analyzer uses queueing-theory concepts to represent computer systems and analyze their performance characteristics. It is a queueing model solver, not a simulator. PDQ models consist of workloads or PDQ streams and queueing faciities or PDQ nodes. Workloads are characterized by whether they are part of an open or closed network (i.e., whether or not new customers enter and leave the system) and by their volume (arrival rate for open networks or number and think time for closed networks). PDQ nodes are the places in the network where requests are serviced and where queues can form. Nodes are characterized by their scheduling discipline, e.g., FCFS, the number of servers available, and the average service time or inverse service rate. The results of solving a particular queueing network can be displayed n the R console either, as selected performance metrics or, as a complete summary of the entire network.

Details

Package:	pdq
Version:	7.0-0
Date:	2020-11-21
License:	MIT
URL:	`http://www.perfdynamics.com/Tools/PDQ.html`
	`https://github.com/DrQz/pdq-qnm-pkg/`
Built:	R 3.6.3; x86_64-apple-darwin15.6.0; 2020-11-26 02:25:06 UTC; unix

Author(s)

Neil Gunther with contributions from Paul Puglia and Phil Feller for R versions.

Maintainer: Neil Gunther <info@perfdynamics.com>

References

Gunther, N. J. (2000) *The Practical Performance Analyst* . Lincoln, Nebraska: iUniverse Press. `http://www.perfdynamics.com/iBook/ppa.html`

Gunther, N. J. (2011) *Analyzing computer systems performance with PERL::PDQ (2nd edn.)*. Heidelberg, Germany: Springer-Verlag. `http://www.perfdynamics.com/iBook/ppa_new.html`

Examples

```
library(pdq)

# Define and solve a PDQ model
Init("Grocery Checkout")             # Initialize PDQ and name the model
CreateOpen("Customers", 0.75)        # Defines an open queueing network
CreateNode("Cashier", CEN, FCFS)     # Single checkout queue serviced in FIFO order
SetDemand("Cashier", "Customers", 1.0) # Assign the average service time
SetWUnit("Bags")                     # Change reported work units
SetTUnit("Min")                      # Change reported timebase units
Solve(STREAMING)                     # Solve the open continuous workflow model
Report()                             # Display performance metrics in the R console
```

`CreateClosed` *Create a source of work for a closed queueing network*

Description

Define the workload for a closed queueing network. A closed circuit has a fixed population of customers: no new customers enter the system and none exit. The workload is characterized by the number of customers and the amount of time between requests (the "think time"). Separate calls are required for workload streams having different characteristics.

Usage

```
CreateClosed(name, class, pop, think)
```

Arguments

`name`	A character string used to identify the workload by name in a PDQ Report and as an argument to other PDQ functions
`class`	The type of PDQ workload: TERM or BATCH
`pop`	The customer "population" or total number of requests in the closed circuit. Since pop can be a statistical average arising from measurements, it is a numeric type. Often used to represent users in a time-share computer system or load generators in a load-test system.
`think`	The user or generator delay known as the "think time" that occurs between the completion of the previous request and the start of the next request in a TERM workload. Set to zero for a BATCH workload. Numeric type.

Details

CreateClosed should be invoked before defining any queueing nodes.

There are two types of closed-circuit workloads:

TERM A workload with non-zero think time that causes a `think` delay to occur before a new request enters the closed system

BATCH A workload without any think time. Requests immediately re-enter the closed system

Author(s)

Neil J. Gunther

References

Gunther, N. J. (2011) *Analyzing computer systems performance with PERL::PDQ*, 2nd edn., Heidelberg, Germany, Springer-Verlag. http://www.perfdynamics.com/iBook/ppa_new.html

See Also

CreateOpen, CreateMultiNode

Examples

```
library(pdq)
Init("CreateClosed Example")
CreateClosed("DB Users", TERM, 500.0, 30.5)
CreateNode("DB Server", CEN, FCFS)
SetDemand("DB Server", "DB Users", 1.0)
Solve(EXACT)
Report()
```

CreateMultiNode *Create a queueing node with multiple servers*

Description

Define a multi-server queueing node for either an open or closed queueing network. A multi-server node consists of a single waiting line feeding into more than one server, e.g., M/M/m, and is characterized by the number of servers, the device type, and the service schedule. The device type can be either MSC for a closed network or MSO for an open network. A separate function call is required to instantiate each queueing node.

Usage

```
CreateMultiNode(servers, name, device, sched)
```

Arguments

`servers`	The number of servers (usually an integral value)
`name`	The character string used to identify the node name in a PDQ report and as an argument to other PDQ functions
`device`	The type of multi-server queueing facility: MSO or MSC
`sched`	The service schedule

Details

There are two possible types of multi-server:

- MSO multi-server queueing center in an OPEN network
- MSC multi-server queueing center in an CLOSED network

The only valid type of service schedule is:

- FCFS or first-come first-served

Solution methods:

- Single OPEN work stream use the CANON or STREAMING solution method with Solve
- Single CLOSED work stream use the EXACT solution method with Solve

Author(s)

Neil J. Gunther

References

Gunther, N. J. (2011) *Analyzing computer systems performance with PERL::PDQ*, 2nd edn., Heidelberg, Germany, Springer-Verlag. `http://www.perfdynamics.com/iBook/ppa_new.html`

See Also

`CreateNode`,`Solve`

Examples

```
library(pdq)
Init("Banking Example")
CreateOpen("Customers", 0.75)
CreateMultiNode(3, "Tellers", MSO, FCFS)
SetDemand("Tellers", "Customers", 1.0)
Solve(STREAMING)
Report()
```

`CreateNode` *Create a single-server queueing node*

Description

Define a queueing service node for either a closed or open circuit model. A node is characterized by the type of server and the queue disciplne. A separate invocation is required to create each queueing node in the model.

Usage

```
CreateNode(name, device, sched)
```

Arguments

`name`	The string used to identify the node in reports and as an argument to other functions
`device`	The type of server
`sched`	The queue discipline

Details

The two types of servers are:

- CEN Generic queueing center
- DLY Generic delay center: a delay without a queue

The only valid type of queue discipline is:

- FCFS or first-come first-served

Solution methods:

- Single PDQ workstream use the CANON solution method with Solve
- Multiple PDQ workstreams use the CANON solution method with Solve

Author(s)

Neil J. Gunther

References

Gunther, N. J. (2011) *Analyzing computer systems performance with PERL::PDQ*, 2nd edn., Heidelberg, Germany, Springer-Verlag. http://www.perfdynamics.com/iBook/ppa_new.html

See Also

CreateMultiNode,Solve

Examples

```
library(pdq)
Init("CreateNode Example")
CreateOpen("httpGet", 0.75)
CreateNode("cpu",  CEN, FCFS)
CreateNode("disk", CEN, FCFS)
SetDemand("cpu", "httpGet", 0.1)
SetDemand("disk", "httpGet", 1.0)
Solve(CANON)
Report()
```

CreateOpen *Define Workload for Open Queuing Circuit*

Description

Define the workload for a open queueing circuit. A open circuit has an unbounded population of customers: new customers enter the system at a defined rate and exit. once processed The workload is characterized by the arrival rate of new customers. Separate calls are required for workload streams having different characteristics.

Usage

```
CreateOpen(name, lambda)
```

Arguments

`name`	The string used to identify the workload in reports and as an argument to other functions
`lambda`	The arrival rate per unit time into the queueing circuit.

Details

CreateOpen should be invoked before defining any queueing nodes.

Author(s)

Neil Gunther

References

Gunther, N. J. 2005 *Analyzing computer systems performance with PERL::PDQ*. Berlin, Heidelberg: Springer.

See Also

`CreateClosed`

Examples

```
library(pdq)
Init("CreateOpen Example")
CreateOpen("Customers", 0.75)
CreateMultiNode(3, "Bank Tellers", CEN, FCFS)
SetDemand("Bank Tellers", "Customers", 1.0)
Solve(STREAMING)
Report()
```

`GetLoadOpt` *Get the optimal user load*

Description

Returns the calculated optimal user load for a specified workload in a closed queueing circuit.

Usage

```
GetLoadOpt(class, wname)
```

Arguments

`class`	Type of workload: BATCH or TERM
`wname`	Character string containing the name of the workload

Details

The two types of closed-circuit workloads are:

- TERM a workload with non-zero think time: there will be `think` delay before requests re-enter the system
- BATCH a workload with no think time: requests immediately re-enter the system

GetLoadOpt should only be called after the PDQ model has been solved.

Value

Optimal user load as a numeric type

Author(s)

Neil J. Gunther

References

Gunther, N. J. (2011) *Analyzing computer systems performance with PERL::PDQ*, 2nd edn., Heidelberg, Germany, Springer-Verlag. http://www.perfdynamics.com/iBook/ppa_new.html

Examples

```
library(pdq)
Init("GetLoadOpt Example")
CreateClosed("DB Users", TERM, 500.0, 30.5)
CreateNode("DB Server", CEN, FCFS)
SetDemand("DB Server", "DB Users", 1.0)
Solve(EXACT)
nopt <- GetLoadOpt(TERM, "DB Users")
print(nopt)
```

`GetNodesCount` *Get the number of nodes*

Description

Determine the number of queueing nodes that have been assigned in the PDQ model.

Usage

```
GetNodesCount()
```

Value

Integer value of numeric type.

Author(s)

Neil J. Gunther

References

Gunther, N. J. (2011) *Analyzing computer systems performance with PERL::PDQ*, 2nd edn., Heidelberg, Germany, Springer-Verlag. http://www.perfdynamics.com/iBook/ppa_new.html

Examples

```
library(pdq)
Init("GetNodesCount Example")
CreateClosed("work", TERM, 100, 10)
for (k in 1:4) {
    nname <- paste("node", k, sep="")
    CreateNode(nname, CEN, FCFS)
    SetDemand(nname, "work", 500)
}
if(GetNodesCount() > 3) {
   stop("This model has too many nodes!")
} else {
   Solve(EXACT)
   Report()
}
```

`GetQueueLength` *Get the queue length at a particular node*

Description

Determine the queue length (number of requests waiting plus the number in service) of the designated node servicing a specified workload.

Usage

```
GetQueueLength(device, work, class)
```

Arguments

`device`	String containing the name of the queueing node.
`work`	String containing the name of the workload.
`class`	Type of workload: TRANS, TERM, or BATCH

Details

The classes of workloads are:

- TRANS a workload that is defined by an arrival rate, not a think time and is only valid for an OPEN network
- TERM a workload with non-zero think time: there will be `think` delay before requests re-enter the system; only valid for a CLOSED network
- BATCH a workload with no think time: requests immediately re-enter the system; only valid for a CLOSED network

GetQueueLength should only be called after the PDQ model has been solved.

Value

Queue length as a numeric type.

Author(s)

Neil J. Gunther

References

Gunther, N. J. (2011) *Analyzing computer systems performance with PERL::PDQ*, 2nd edn., Heidelberg, Germany, Springer-Verlag. http://www.perfdynamics.com/iBook/ppa_new.html

Examples

```
library(pdq)

Init("GetQueueLength Example")
CreateClosed("DB Users", TERM, 500.0, 30.5)
CreateNode("DB Server", CEN, FCFS)
SetDemand("DB Server", "DB Users", 1.0)
Solve(EXACT)

qsize <- GetQueueLength("DB Server", "DB Users", TERM)
print(qsize)
```

`GetResidenceTime` *Get the residence time spent at a particular node*

Description

Determine the residence time at the designated service node by the specified workload.

Usage

```
GetResidenceTime(device, work, class)
```

Arguments

`device`	String containing the name of the queueing service node.
`work`	String containing the name of the workload.
`class`	TRANS, TERM, or BATCH type.

Details

The classes of workloads are:

- TRANS a workload that is defined by arrival rate, not think time; only valid for an open circuit
- TERM a workload with non-zero think time: there will be `think` delay before requests re-enter the system; only valid for a closed circuit
- BATCH a workload with no think time: requests immediately re-enter the system; only valid for a closed circuit

GetQueueLength should only be called after the PDQ model has been solved.

Value

Residence time as a numeric type.

Author(s)

Neil J. Gunther

References

Gunther, N. J. (2011) *Analyzing computer systems performance with PERL::PDQ*, 2nd edn., Heidelberg, Germany, Springer-Verlag. http://www.perfdynamics.com/iBook/ppa_new.html

Examples

```
library(pdq)
Init("GetResidenceTime Example")
CreateClosed("DB Users", TERM, 10.0, 30.5)
CreateNode("DB Server", CEN, FCFS)
SetDemand("DB Server", "DB Users", 1.0)
Solve(EXACT)
reztime <- GetResidenceTime("DB Server", "DB Users", TERM)
reztime
```

`GetResponse` *Get the system response time*

Description

Determine the system response time for the specified workload. The response time is the sum of all the residence times spent each of the queueing nodes defined in the network.

Usage

```
GetResponse(class, wname)
```

Arguments

`class`	TRANS, TERM, or BATCH type.
`wname`	Character string containing the name of the workload.

Details

The classes of workloads are:

- TRANS a workload that is defined by arrival rate, not think time; only valid for an open circuit
- TERM a workload with non-zero think time: there will be `think` delay before requests re-enter the system; only valid for a closed circuit
- BATCH a workload with no think time: requests immediately re-enter the system; only valid for a closed circuit

GetResponse should only be called after the PDQ model has been solved.

Value

System response time as a numeric type.

Author(s)

Neil J. Gunther

References

Gunther, N. J. (2011) *Analyzing computer systems performance with PERL::PDQ*, 2nd edn., Heidelberg, Germany, Springer-Verlag. http://www.perfdynamics.com/iBook/ppa_new.html

Examples

```
library(pdq)
Init("GetResponse Example")
CreateClosed("DB Users", TERM, 500.0, 30.5)
CreateNode("DB Server", CEN, FCFS)
SetDemand("DB Server", "DB Users", 1.0)
Solve(EXACT)
rt <- GetResponse(TERM, "DB Users")
rt
```

`GetStreamsCount` *Get the number of streams*

Description

Determine the number of workload streams that have been assigned in the PDQ model.

Usage

```
GetStreamsCount()
```

Value

Integer value of numeric type.

Author(s)

Neil J. Gunther

References

Gunther, N. J. (2011) *Analyzing computer systems performance with PERL::PDQ*, 2nd edn., Heidelberg, Germany, Springer-Verlag. http://www.perfdynamics.com/iBook/ppa_new.html

Examples

```
library(pdq)
Init("GetStreamsCount Example")
CreateNode("node", CEN, FCFS)
for (strm in 1:4) {
    wname <- paste("work", strm, sep="")
    CreateClosed(wname, TERM, 10, 30)
    SetDemand("node", wname, strm)
}
if(GetStreamsCount() > 3) {
   stop("This model has too much work to do!")
} else {
   Solve(APPROX)
   Report()
}
```

`GetThruput` *Get the system throughput*

Description

Determine the system throughput for the specified workload.

Usage

```
GetThruput(class, wname)
```

Arguments

`class`	TRANS, TERM, or BATCH type.
`wname`	Character string containing the name of the workload.

Details

The classes of workloads are:

- TRANS a workload that is defined by arrival rate, not think time; only valid for an open circuit
- TERM a workload with non-zero think time: there will be `think` delay before requests re-enter the system; only valid for a closed circuit
- BATCH a workload with no think time: requests immediately re-enter the system; only valid for a closed circuit

Value

System throughput as a decimal number.

Author(s)

Neil J. Gunther

References

Gunther, N. J. (2011) *Analyzing computer systems performance with PERL::PDQ*, 2nd edn., Heidelberg, Germany, Springer-Verlag. http://www.perfdynamics.com/iBook/ppa_new.html

Examples

```
library(pdq)
Init("GetThruput Example")
CreateClosed("DB Users", TERM, 10.0, 30.5)
CreateNode("DB Server", CEN, FCFS)
SetDemand("DB Server", "DB Users", 1.0)
Solve(EXACT)
tp <- GetThruput(TRANS, "DB Users")
tp
```

GetUtilization — *Get the utilization of a service node*

Description

Determine the utilization of the designated service node by the specified workload. It should only be called after the PDQ model has been solved.

Usage

```
GetUtilization(device, work, class)
```

Arguments

`device`	String containing the name of the service node.
`work`	String containing the name of the workload.
`class`	TRANS, TERM, or BATCH type.

Details

The classes of workloads are:

TRANS a workload that is defined by arrival rate, not think time; only valid for an open circuit

TERM a workload with non-zero think time: there will be `think` delay before requests re-enter the system; only valid for a closed circuit

BATCH a workload with no think time: requests immediately re-enter the system; only valid for a closed circuit

Value

Utilization as a decimal fraction in the range 0.0 to 1.0 (i.e. 100%).

Author(s)

Neil Gunther

References

Gunther, N. J. 2005 *Analyzing computer systems performance with PERL::PDQ*. Berlin, Heidelberg: Springer.

Examples

```
library(pdq)
Init("GetUtilization Example")
CreateClosed("DB Users", TERM, 10.0, 30.5)
CreateNode("DB Server", CEN, FCFS)
SetDemand("DB Server", "DB Users", 1.0)
Solve(EXACT);
ut <- GetUtilization("DB Server", "DB Users", TERM)
ut
```

`Init` *Initialize the PDQ model*

Description

Initializes all internal PDQ variables. Must be called prior to any other PDQ function. It also resets all internal PDQ variables so that no separate cleanup function call is required.

Usage

```
Init(name)
```

Arguments

`name`	A string containing the name of the queueing model that will appear in the PDQ Report banner. To maintain cosmetic appearances, the model name should not exceed 24 characters (including spaces).

Details

Init must be called for each model in the same script.

Author(s)

Neil J. Gunther

References

Gunther, N. J. (2011) *Analyzing computer systems performance with PERL::PDQ*, 2nd edn., Heidelberg, Germany, Springer-Verlag. `http://www.perfdynamics.com/iBook/ppa_new.html`

Examples

```
# Solve successive PDQ model instances of 'users' as the independent variable xc
# to plot the dependent throughput variable yc
library(pdq)
users <- 100
think <- 8e-3   # milliseconds
stime <- 500e-3 # milliseconds
work  <- "DB query"
xc <- 0 # plot vector
yc <- 0 # plot vector
# THe following for-loop solves successive PDQ models
# Each pass requires that the pdq::Init function be called
for (i in 1:users) {
    Init("Multiple Init Example")   # reinitialize PDQ variables
    CreateClosed(work, TERM, as.double(i), think)
    # create a PDQ queueing network with 20 queueing nodes
    for (j in 1:20) {
        nname <- paste("node", sep="", j)   # add j index to node name
        CreateNode(nname, CEN, FCFS)
        SetDemand(nname, work, stime)
    }
    Solve(EXACT)
    # store PDQ results in plot vectors
    xc[i] <- as.double(i)
    yc[i] <- GetThruput(TERM, work)
}
plot(xc, yc, type="l", col="blue", xlab="Users", ylab="Throughput")
```

Report — *Generate a PDQ report*

Description

Report produces a formatted summary of all modeling results.

Usage

```
Report()
```

Details

A PDQ report is comprised of several sections.

Model INPUTS Name of the PDQ model.

- Date when the model was executed.
- Version of PDQ used.
- A comment field is included if SetComment is called.
- A summary of all workload parameters.

Model OUTPUTS

- Itemized summary of each queueing node and its workload.
- Summary of the queeuing network parameters.
- Solution method used.
- System-level performance metrics.
- Node-level performance metrics such as: Capacity, Utilization, and Waiting time.

Author(s)

Neil J. Gunther

References

Gunther, N. J. (2011) *Analyzing computer systems performance with PERL::PDQ*, 2nd edn., Heidelberg, Germany, Springer-Verlag. http://www.perfdynamics.com/iBook/ppa_new.html

See Also

SetComment

Examples

```
library(pdq)
arate <- 0.75
srate <- 1.0
Init("Report Example")
SetComment("This is just an toy example.")
CreateNode("server", CEN, FCFS)
CreateOpen("work", arate)
SetDemand("server", "work", 1/srate)
Solve(CANON)
Report()
```

`SetComment` *Include a comment in a PDQ Report*

Description

Useful for keeping track of parameter variations across the same or similar PDQ models. Appears near the top of the Report ouput.

Usage

```
SetComment(comment)
```

Arguments

`comment`	A character string.

Author(s)

Neil J. Gunther

References

Gunther, N. J. (2011) *Analyzing computer systems performance with PERL::PDQ*, 2nd edn., Heidelberg, Germany, Springer-Verlag. http://www.perfdynamics.com/iBook/ppa_new.html

Examples

```
library(pdq)
arate <- 0.75
srate <- 1.0
Init("SetComment Example")
SetComment("This is just an toy example.")
CreateNode("server", CEN, FCFS)
CreateOpen("work", arate)
SetDemand("server", "work", 1/srate)
Solve(STREAMING)
Report()
```

SetDemand *Assign a service demand to a workload*

Description

Define the service demand (or service time) for a workload serviced by a given node. The named node and workload must have been defined previously. A separate call is required for each workload stream that accesses the same node.

Usage

```
SetDemand(nodename, workname, time)
```

Arguments

nodename	Character string to name the queueing node.
workname	Character string to name the workload.
time	Service demand (in the appropriate time units) required by the workload at that node.

Author(s)

Neil J. Gunther

References

Gunther, N. J. (2011) *Analyzing computer systems performance with PERL::PDQ*, 2nd edn., Heidelberg, Germany, Springer-Verlag. http://www.perfdynamics.com/iBook/ppa_new.html

Examples

```
library(pdq)
Init("SetDemand Example")
CreateClosed("processA", TERM, 57.4, 31.6)
CreateClosed("processB", BATCH, 10, 0)
CreateNode("CPU", CEN, FCFS)
SetDemand("CPU", "processA", 0.130)
SetDemand("CPU", "processB", 3.122)
Solve(EXACT)
Report()
```

`SetTUnit` *Change time unit*

Description

Change the time unit label that appears in PDQ Report. The default time unit is seconds.

Usage

```
SetTUnit(unitName)
```

Arguments

`unitName`	Character string containing the name of the time unit as it will appear in the Report output.

Details

Cannot be invoked prior to calling CreateOpen or CreateClosed.

Author(s)

Neil J. Gunther

References

Gunther, N. J. (2011) *Analyzing computer systems performance with PERL::PDQ*, 2nd edn., Heidelberg, Germany, Springer-Verlag. `http://www.perfdynamics.com/iBook/ppa_new.html`

Examples

```
library(pdq)
Init("SetTUnit Example")
CreateOpen("Customers", 0.75)
CreateMultiNode(3, "Bank Tellers", CEN, FCFS)
SetDemand("Bank Tellers", "Customers", 1.0)
SetTUnit("Minutes")
Solve(CANON)
Report()
```

`SetVisits` *Define the service demand in terms of visits*

Description

An alternative to SetDemand where a workload is defined in terms of the explicit service time as well as a visit count. The named node and workload must exist. A separate call is required for each workload stream that accesses the same node. SetVisits is different from SetDemand in the way node-level performance metrics are formatted in the Report output. The number of visits shows up in the Report INPUTS section. The throughput in the RESOURCE Performance section shows up as counts per unit time.

Usage

```
SetVisits(nodename, workname, visits, service)
```

Arguments

`nodename`	The string name of the queueing node.
`workname`	The string name of the workload.
`visits`	Number of visits to that node.
`service`	Service time the workload demands at that node (in time units).

Author(s)

Neil J. Gunther

References

Gunther, N. J. (2011) *Analyzing computer systems performance with PERL::PDQ*, 2nd edn., Heidelberg, Germany, Springer-Verlag. http://www.perfdynamics.com/iBook/ppa_new.html

See Also

`SetDemand`

Examples

```
library(pdq)
Init("SetVisits Example")
CreateClosed("DB Users", TERM, 10.0, 30.5)
CreateNode("DB Server", CEN, FCFS)
SetVisits("DB Server", "DB Users", 10.0, 0.13)
Solve(EXACT)
Report()
```

`SetWUnit` *Change work unit*

Description

Change the work unit label that appears in PDQ Report.

Usage

```
SetWUnit(unitName)
```

Arguments

`unitName`	Character string containing the name of the work unit as it will appear in the Report output.

Author(s)

Neil J. Gunther

References

Gunther, N. J. (2011) *Analyzing computer systems performance with PERL::PDQ*, 2nd edn., Heidelberg, Germany, Springer-Verlag. `http://www.perfdynamics.com/iBook/ppa_new.html`

Examples

```
library(pdq)
library(pdq)
Init("SetWUnit Example")
CreateOpen("Customers", 0.75)
CreateMultiNode(3, "Bank Tellers", CEN, FCFS)
SetDemand("Bank Tellers", "Customers", 1.0)
SetWUnit("Cheques")
Solve(CANON)
Report()
```

`Solve` *Solve the PDQ model*

Description

Solve is called after the PDQ queueing network model has been defined. An appropriate solution method must be supplied as an argument.

Usage

```
Solve(method)
```

Arguments

	Available methods are:
	An approximation to the EXACT solution method when there is a large number of requests in the system. Only valid for solving a closed queueing circuit.
`APPROXMSQ`	Required for solving a multiple server queueing center(using CreateMultiNode) with multiple open-circuit workloads.
`CANON`	Canonical solution technique. Only valid for an open queueing circuit.
`EXACT`	Iterative mean value analysis (MVA) solution technique. Only valid for a closed queueing circuit.

Author(s)

Neil J. Gunther

References

Gunther, N. J. (2011) *Analyzing computer systems performance with PERL::PDQ*, 2nd edn., Heidelberg, Germany, Springer-Verlag. `http://www.perfdynamics.com/iBook/ppa_new.html`

See Also

`CreateClosed`, `CreateMultiNode`, `CreateNode`

Examples

```
library(pdq)
Init("Solve Example")
CreateOpen("Customers", 0.75)
CreateMultiNode(3, "Bank Tellers", CEN, FCFS)
SetDemand("Bank Tellers", "Customers", 1.0)
Solve(STREAMING)
Report()
```

Index