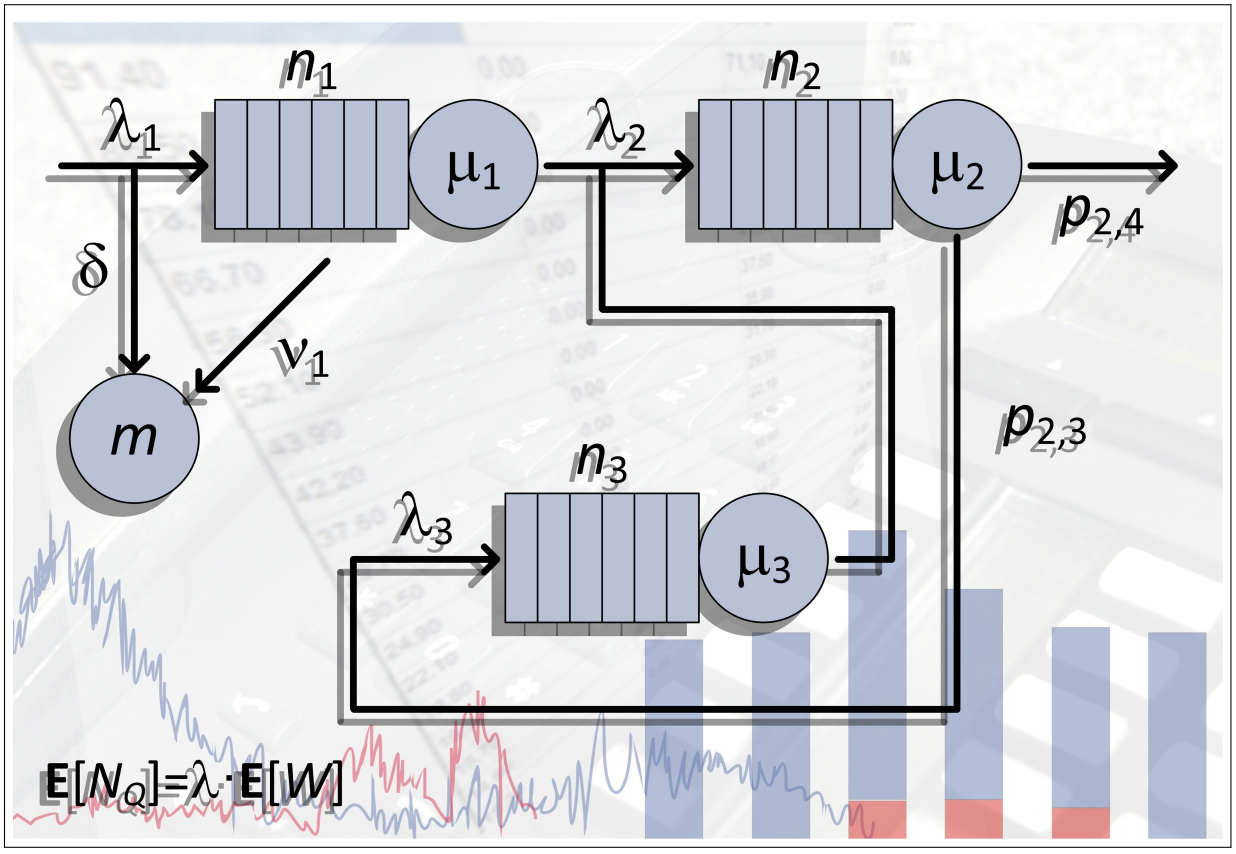
Calculation and scripting command reference for Warteschlangensimulator

Alexander Herzog (alexander.herzog@tu-clausthal.de)



This reference refers to version 5.1.0 of Warteschlangensimulator.
Download address: https://github.com/A-Herzog/Warteschlangensimulator/.

Contents

Chapter 1

Calculation commands and scripting in Warteschlangensimulator

Calculation commands can be used in the simulator e.g. to determine time periods (such as processing times) or to specify in which branching direction a client should be directed.

Scripts can be used both for the determination of branch directions and for the evaluation of simulation results and for running of parameter series. The Warteschlangensimulator uses Javascript and Java as languages.

1.1 Create expressions

To the right of all input fields into which calculations commands can be entered always the following button is displayed:  By using this button the **Edit expression** dialog (see figure 1.1) can be accessed. The dialog contains a complete list of all the commands available in the current context, and makes it easy to put together more complex commands and expressions.

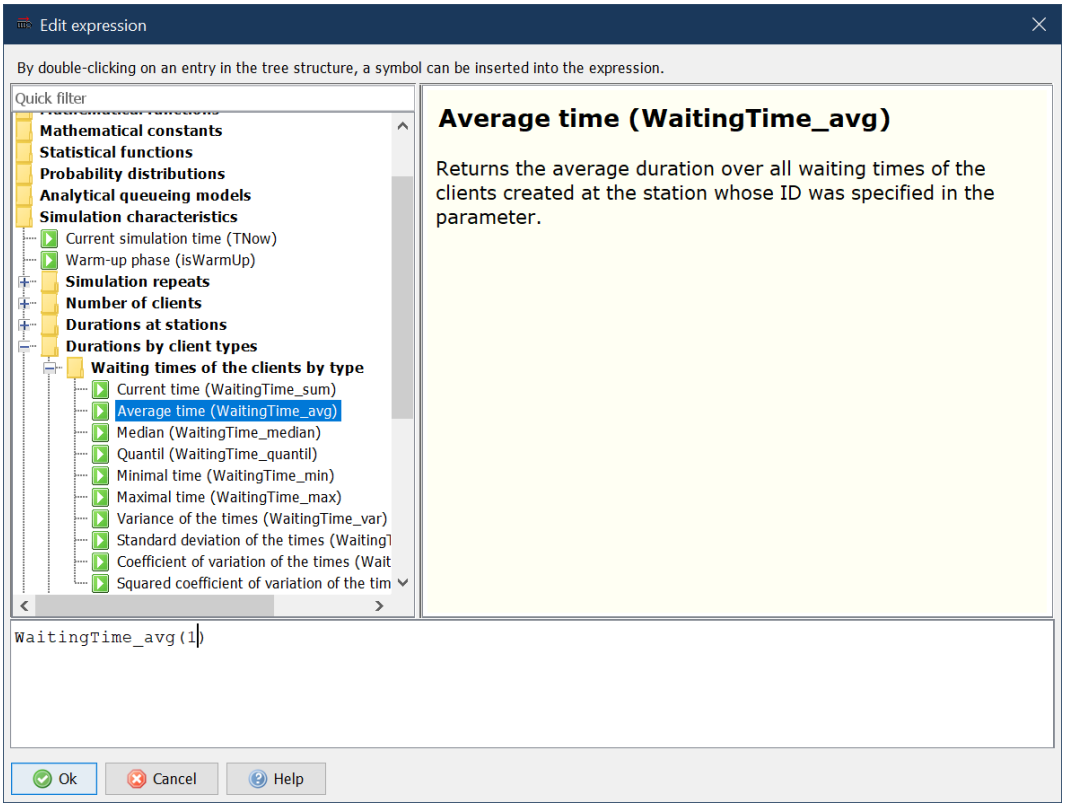


Fig. 1.1: Edit expression dialog

Part I

Calculation commands reference

When using calculation commands in Warteschlangensimulator, it is distinguished between **expressions** and **comparisons**. Expressions are used to calculate a numerical values, which are e.g. used as periods of time. Comparisons provide a yes/no decision (for example, whether a client should be directed in a particular direction). Unlike expressions, comparisons always contain at least one comparison operator.

All commands presented below are each recognized in **any case**. There is no distinction between different case types.

Chapter 2
Constants

The following constants are available in all calculation commands:

- `"e"`: Returns the basis of the exponential function e^x. It is $e \approx 2.718281828459$.
- `"pi"`: Returns the value of the circle constant π. It is $\pi \approx 3.1415926535898$.

Chapter 3
Variables

When calculating values in the context of a concrete client, the variables

- `"w"` for the **previous waiting time** of the client,
- `"t"` for the **previous transfer time** of the client ant
- `"p"` for the **previous operating time** of the client are always available.

If the calculation is done for getting a clients score, the variable `"w"` does not contain the total waiting time of the current client but the waiting time of the current client at the current station.

Furthermore, all variables that are defined by an assignment element are always available. Before the first assignment of a value to a variable, it has the value 0.

Chapter 4
Basic arithmetic operations

Supported instructions for the basic arithmetic operations:

- Addition: "+"
- Subtraction: "−"
- Multiplication: "∗"
- Division: "/"
- Potentiate: "^"

The rule **point before line calculation** is taken into account. To enforce deviating evaluations, **brackets** can be set.

Chapter 5
Trailing instructions

The following expressions can be written directly behind a number:

- `"%"`: The numerical value left to this symbol is interpreted as a percent value, for example $30\% = 0.3$.
- `"²"`: Exponentiate number by 2.
- `"³"`: Exponentiate number by 3.
- `"!"`: Calculate factorial of the number, for example $4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24$.
- `"°"`: Converts the value left to this symbol from grad to radian, for example $180° = 3.1415\ldots$.
  (See also section 7 in which the supported trigonometric functions are presented.)

Chapter 6
General functions

- "`abs(x)`": Absolute value, for example `abs(-5)=5`.
- "`ceil(x)`": Round to next bigger integer number, for example `ceil(2.1)=3`
- "`exp(x)`": Exponential function e^x.
- "`factorial(x)`": Factorial, for example $4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24$.
- "`binom(n;k)`": Binomial coefficient
- "`floor(x)`": Round to next smaller integer number, for example `floor(2.9)=2`
- "`frac(x)`": Fraction part, for example `frac(1.3)=0,3`
- "`gamma(x)`": Gamma funkcion, for example `gamma(5)=4!=24`
- "`int(x)`": Integer part, for example `int(2.9)=2`
- "`log(x)`": Logarithm to the base e.
- "`log(x;b)`": Logarithm to the base b.
- "`ld(x)`": Logarithm to the base 2, for example `ld(256)=8`.
- "`lg(x)`": Logarithm to the base 10, for example `lg(100)=2`.
- "`ln(x)`": Logarithm to the base e.
- "`modulo(a;b)`" oder "`mod(a;b)`": Division reminder when dividing a/b
- "`pow(x;y)`": Exponentiate x^y.
- "`round(x)`": Round, for example `round(4.4)=4` and `round(4.5)=5`.
- "`sign(x)`": Sign of a number, for example `sign(3)=1` and `sign(-3)=-1`.
- "`sqrt(x)`": Square root, for example `sqrt81=9`.
- "`sqr(x)`": Square the number, for example `sqr(4)=16`.

6.1 Random numbers

The following commands can be used to generate random numbers that are **equally distributed** in a certain area. Section 9 introduces additional functions for generating random numbers according to certain distribution functions.

- "`random()`": Random number between 0 (inclusive) and 1 (exclusive).

- `"random(x)"`: Random number between 0 (inclusive) and x (exclusive).

Chapter 7
Trigonometric functions

The trigonometric functions always refer to 2π as a full circle (radians). If angles in degrees (360° for the full circle) are to be specified in the elementary trigonometric functions, these have to be converted to radians using the angle functions, for example `sin(90°)=1`.

7.1 Elementary trigonometric functions

- `"sin(x)"`: Sine
- `"cos(x)"`: Cosine
- `"tan(x)"`: Tangent
- `"cot(x)"`: Cotangent

7.2 Hyperbolic trigonometric functions

- `"sinh(x)"`: Sine hyperbolicus
- `"cosh(x)"`: Cosine hyperbolicus
- `"tanh(x)"`: Tangent hyperbolicus
- `"coth(x)"`: Cotangent hyperbolicus

7.3 Inverse of the elementary trigonometric functions

- `"arcsin(x)"`: Arcus sine
- `"arccos(x)"`: Arcus cosine
- `"arctan(x)"`: Arcus tangent
- `"arccot(x)"`: Arcus cotangent

7.4 Inverse of the hyperbolic trigonometric functions

- `"arcsinh(x)"`: Arcus sine hyperbolicus

- `"arccosh(x)"`: Arcus cosine hyperbolicus
- `"arctanh(x)"`: Arcus-Tangent hyperbolicus
- `"arccoth(x)"`: Arcus-Cotangent hyperbolicus

Chapter 8
Functions with multiple parameters

The following functions can accept any number of parameters. The individual parameters have to be specified separately by semicolon ";".

- `"Min(a;b;c;...)"`: Calculates the minimum of the given numbers.
- `"Max(a;b;c;...)"`: Calculates the maximum of the given numbers.
- `"Sum(a;b;c;...)"`: Calculates the sum of the given numbers.
- `"Mean(a;b;c;...)"`: Calculates the mean value of the given numbers.
- `"Median(a;b;c;...)"`: Calculates the median of the given numbers.
- `"Var(a;b;c;...)"`: Calculates the sample variance of the given numbers.
- `"SD(a;b;c;...)"`: Calculates the sample standard deviation of the given numbers.
- `"SCV(a;b;c;...)"`: Calculates the squared coefficient of variation of the given numbers.
- `"CV(a;b;c;...)"`: Calculates the coefficient of variation of the given numbers.

Chapter 9
Probability distributions

By using the following commands both values of the density and the cumulative distribution function of the following probability distributions can be calculated as well as random numbers are based on one of these probability distributions:

9.1 Hypergeometric distribution $Hg(N, K, n)$

- `"HypergeometricDist(k;N;K;n)"`: Calculates the counting probability density at k.
- `"HypergeometricDist(N;K;n)"`: Generates a random number based on this distribution.

9.2 Binomial distribution $B(n, p)$

- `"BinomialDist(k;n;p)"`: Calculates the counting probability density at k.
- `"BinomialDist(n;p)"`: Generates a random number based on this distribution.

9.3 Poisson distribution $P(l)$

- `"PoissonDist(k;l)"`: Calculates the counting probability density at k.
- `"PoissonDist(l)"`: Generates a random number based on this distribution.

9.4 Exponential distribution with mean a

- `"ExpDist(x;a;0)"`: Calculates the probability density at x.
- `"ExpDist(x;a;1)"`: Calculates the cumulative distribution function at x.
- `"ExpDist(a)"`: Generates a random number based on this distribution.
- `"ExpDistRange(min;max;a)"`: Generates a random number based on this distribution but limits the range of the value to $min \ldots max$.

9.5 Uniform distribution in the interval $[a; b]$

- `"UniformDist(x;a;b;0)"`: Calculates the probability density at x.
- `"UniformDist(x;a;b;1)"`: Calculates the cumulative distribution function at x.
- `"UniformDist(a;b)"`: Generates a random number based on this distribution.

9.6 Normal distribution with mean a and standard deviation b

- `"NormalDist(x;a;b;0)"`: Calculates the probability density at x.
- `"NormalDist(x;a;b;1)"`: Calculates the cumulative distribution function at x.
- `"NormalDist(a;b)"`: Generates a random number based on this distribution.
- `"NormalDistRange(min;max;a;b)"`: Generates a random number based on this distribution but limits the range of the value to $min \ldots max$.

9.7 Log-normal distribution with mean a and standard deviation b

- `"LogNormalDist(x;a;b;0)"`: Calculates the probability density at x.
- `"LogNormalDist(x;a;b;1)"`: Calculates the cumulative distribution function at x.
- `"LogNormalDist(a;b)"`: Generates a random number based on this distribution.
- `"LogNormalDistRange(min;max;a;b)"`: Generates a random number based on this distribution but limits the range of the value to $min \ldots max$.

9.8 Gamma distribution with parameters $\alpha = a$ and $\beta = b$

- `"GammaDist(x;a;b;0)"`: Calculates the probability density at x.
- `"GammaDist(x;a;b;1)"`: Calculates the cumulative distribution function at x.
- `"GammaDist(a;b)"`: Generates a random number based on this distribution.
- `"GammaDistRange(min;max;a;b)"`: Generates a random number based on this distribution but limits the range of the value to $min \ldots max$.

9.9 Gamma distribution with mean a and standard deviation b

- `"GammaDistDirect(x;a;b;0)"`: Calculates the probability density at x.
- `"GammaDistDirect(x;a;b;1)"`: Calculates the cumulative distribution function at x.
- `"GammaDistDirect(a;b)"`: Generates a random number based on this distribution.
- `"GammaDistDirectRange(min;max;a;b)"`: Generates a random number based on this distribution but limits the range of the value to $min \ldots max$.

9.10 Erlang distribution with parameters n and $\lambda = l$

- `"ErlangDist(x;n;l;0)"`: Calculates the probability density at x.
- `"ErlangDist(x;n;l;1)"`: Calculates the cumulative distribution function at x.
- `"ErlangDist(n;b)"`: Generates a random number based on this distribution.
- `"ErlangDistRange(min;max;n;b)"`: Generates a random number based on this distribution but limits the range of the value to $min \ldots max$.

9.11 Beta distribution in the interval $[a; b]$ and with parameters $\alpha = c$ and $\beta = d$

- `"BetaDist(x;a;b;c;d;0)"`: Calculates the probability density at x.
- `"BetaDist(x;a;b;c;d;1)"`: Calculates the cumulative distribution function at x.
- `"BetaDist(a;b;c;d)"`: Generates a random number based on this distribution.

9.12 Beta distribution in the interval $[a; b]$ and with mean c and standard deviation d

- `"BetaDistDirect(x;a;b;c;d;0)"`: Calculates the probability density at x.
- `"BetaDistDirect(x;a;b;c;d;1)"`: Calculates the cumulative distribution function at x.
- `"BetaDistDirect(a;b;c;d)"`: Generates a random number based on this distribution.

9.13 Weibull distribution with parameters Scale=a and Form=b

- `"WeibullDist(x;a;b;0)"`: Calculates the probability density at x.
- `"WeibullDist(x;a;b;1)"`: Calculates the cumulative distribution function at x.
- `"WeibullDist(a;b)"`: Generates a random number based on this distribution.
- `"WeibullDistRange(min;max;a;b)"`: Generates a random number based on this distribution but limits the range of the value to $min \ldots max$.

9.14 Cauchy distribution with mean a and Scale=b

- `"CauchyDist(x;a;b;0)"`: Calculates the probability density at x.
- `"CauchyDist(x;a;b;1)"`: Calculates the cumulative distribution function at x.
- `"CauchyDist(a;b)"`: Generates a random number based on this distribution.
- `"CauchyDistRange(min;max;a;b)"`: Generates a random number based on this distribution but limits the range of the value to $min \ldots max$.

9.15 Chi^2 distribution with n degrees of freedom

- `"ChiSquareDist(x;n;0)"`: Calculates the probability density at x.
- `"ChiSquareDist(x;n;1)"`: Chi^2 distribution with n degrees of freedom.
- `"ChiSquareDist(n)"`: Generates a random number based on this distribution.
- `"ChiSquareDistRange(min;max;n)"`: Generates a random number based on this distribution but limits the range of the value to $min \dots max$.

9.16 Chi distribution with n degrees of freedom

- `"ChiDist(x;n;0)"`: Calculates the probability density at x.
- `"ChiDist(x;n;1)"`: Chi distribution with n degrees of freedom.
- `"ChiDist(n)"`: Generates a random number based on this distribution.
- `"ChiDistRange(min;max;n)"`: Generates a random number based on this distribution but limits the range of the value to $min \dots max$.

9.17 F distribution with a degrees of freedom for the numerator and b degrees of freedom for the denominator

- `"FDist(x;a;b;0)"`: Calculates the probability density at x.
- `"FDist(x;a;b;1)"`: Calculates the cumulative distribution function at x.
- `"FDist(a;b)"`: Generates a random number based on this distribution.
- `"FDistRange(min;max;a;b)"`: Generates a random number based on this distribution but limits the range of the value to $min \dots max$.

9.18 Johnson SU distribution with parameters $\gamma = a$, $\xi = b$, $\delta = c$ and $\lambda = d$

- `"JohnsonSUDist(x;a;b;c;d;0)"`: Calculates the probability density at x.
- `"JohnsonSUDist(x;a;b;c;d;1)"`: Calculates the cumulative distribution function at x.
- `"JohnsonSUDist(a;b;c;d)"`: Generates a random number based on this distribution.
- `"JohnsonSUDistRange(min;max;a;b;c;d)"`: Generates a random number based on this distribution but limits the range of the value to $min \dots max$.

9.19 Triangular distribution over $[a; c]$ with most likely value b

- `"TriangularDist(x;a;b;c;0)"`: Calculates the probability density at x.
- `"TriangularDist(x;a;b;c;1)"`: Calculates the cumulative distribution function at x.
- `"TriangularDist(a;b;c)"`: Generates a random number based on this distribution.

9.20 Pert distribution over $[a; c]$ with most likely value b

- `"PertDist(x;a;b;c;0)"`: Calculates the probability density at x.
- `"PertDist(x;a;b;c;1)"`: Calculates the cumulative distribution function at x.
- `"PertDist(a;b;c)"`: Generates a random number based on this distribution.

9.21 Laplace distribution with mean mu and scale factor b

- `"LaplaceDist(x;mu;b;0)"`: Calculates the probability density at x.
- `"LaplaceDist(x;mu;b;1)"`: Calculates the cumulative distribution function at x.
- `"LaplaceDist(mu;b)"`: Generates a random number based on this distribution.
- `"LaplaceDistRange(min;max;mu;b)"`: Generates a random number based on this distribution but limits the range of the value to $min \dots max$.

9.22 Pareto distribution with scale parameter $x_{\min} = xmin$ and shape parameter $\alpha = a$

- `"ParetoDist(x;xmin;a;0)"`: Calculates the probability density at x.
- `"ParetoDist(x;xmin;a;1)"`: Calculates the cumulative distribution function at x.
- `"ParetoDist(xmin;a)"`: Generates a random number based on this distribution.

9.23 Logistic distribution with mean $\mu = mu$ and scale parameter s

- `"LogisticDist(x;mu;s;0)"`: Calculates the probability density at x.
- `"LogisticDist(x;mu;s;1)"`: Calculates the cumulative distribution function at x.
- `"LogisticDist(mu;s)"`: Generates a random number based on this distribution.
- `"LogisticDistRange(min;max;mu;s)"`: Generates a random number based on this distribution but limits the range of the value to $min \dots max$.

9.24 Inverse gaussian distribution with $\lambda = l$ and mean mu

- `"InverseGaussianDist(x;l;mu;0)"`: Calculates the probability density at x.
- `"InverseGaussianDist(x;l;mu;1)"`: Calculates the cumulative distribution function at x.
- `"InverseGaussianDist(l;mu)"`: Generates a random number based on this distribution.
- `"InverseGaussianDist(min;max;l;mu)"`: Generates a random number based on this distribution but limits the range of the value to $min \dots max$.

9.25 Rayleigh distribution with mean mu

- `"RayleighDist(x;mu;0)"`: Calculates the probability density at x.

- `"RayleighDist(x;mu;1)"`: Calculates the cumulative distribution function at x.
- `"RayleighDist(mu)"`: Generates a random number based on this distribution.
- `"RayleighDistRange(min;max;mu)"`: Generates a random number based on this distribution but limits the range of the value to $min \dots max$.

9.26 Log-Logistic distribution with α and mean β

- `"LogLogisticDist(x;alpha;beta;0)"`: Calculates the probability density at x.
- `"LogLogisticDist(x;alpha;beta;1)"`: Calculates the cumulative distribution function at x.
- `"LogLogisticDist(alpha;beta)"`: Generates a random number based on this distribution.
- `"LogLogisticDistRange(min;max;alpha;beta)"`: Generates a random number based on this distribution but limits the range of the value to $min \dots max$.

9.27 Power distribution on $[a; b]$ with exponent c

- `"PowerDist(x;a;b;c;0)"`: Calculates the probability density at x.
- `"PowerDist(x;a;b;c;1)"`: Calculates the cumulative distribution function at x.
- `"PowerDist(a;b;c)"`: Generates a random number based on this distribution.

9.28 Gumbel distribution with expected value a and standard deviation b

- `"GumbelDist(x;a;b;0)"`: Calculates the probability density at x.
- `"GumbelDist(x;a;b;1)"`: Calculates the cumulative distribution function at x.
- `"GumbelDist(a;b)"`: Generates a random number based on this distribution.
- `"GumbelDistRange(min;max;a;b)"`: Generates a random number based on this distribution but limits the range of the value to $min \dots max$.

9.29 Fatigue life distribution with location parameter μ, scale parameter β and form parameter γ

- `"FatigueLifeDist(x;mu;beta;gamma;0)"`: Calculates the probability density at x.
- `"FatigueLifeDist(x;mu;beta;gamma;1)"`: Calculates the cumulative distribution function at x.
- `"FatigueLifeDist(mu;beta;gamma)"`: Generates a random number based on this distribution.
- `"FatigueLifeDistRange(min;max;mu;beta;gamma)"`: Generates a random number based on this distribution but limits the range of the value to $min \dots max$.

9.30 Frechet distribution with location parameter δ, scale parameter β and form parameter α

- `"FrechetDist(x;delta;beta;alpha;0)"`: Calculates the probability density at x.
- `"FrechetDist(x;delta;beta;alpha;1)"`: Calculates the cumulative distribution function at x.
- `"FrechetDist(delta;beta;alpha)"`: Generates a random number based on this distribution.
- `"FrechetDistRange(min;max;delta;beta;alpha)"`: Generates a random number based on this distribution but limits the range of the value to $min \ldots max$.

9.31 Hyperbolic secant distribution with mean a and standard deviation b

- `"HyperbolicSecantDist(x;a;b;0)"`: Calculates the probability density at x.
- `"HyperbolicSecantDist(x;a;b;1)"`: Calculates the cumulative distribution function at x.
- `"HyperbolicSecantDist(a;b)"`: Generates a random number based on this distribution.
- `"HyperbolicSecantDistRange(min;max;a;b)"`: Generates a random number based on this distribution but limits the range of the value to $min \ldots max$.

9.32 Left sawtooth distribution over $[a; b]$

- `"LeftSawtoothDist(x;a;b;0)"`: Calculates the probability density at x.
- `"LeftSawtoothDist(x;a;b;1)"`: Calculates the cumulative distribution function at x.
- `"LeftSawtoothDist(a;b)"`: Generates a random number based on this distribution.

9.33 Left sawtooth distribution with mean a and standard deviation b

- `"LeftSawtoothDistDirect(x;a;b;0)"`: Calculates the probability density at x.
- `"LeftSawtoothDistDirect(x;a;b;1)"`: Calculates the cumulative distribution function at x.
- `"LeftSawtoothDistDirect(a;b)"`: Generates a random number based on this distribution.

9.34 Right sawtooth distribution over $[a; b]$

- `"RightSawtoothDist(x;a;b;0)"`: Calculates the probability density at x.
- `"RightSawtoothDist(x;a;b;1)"`: Calculates the cumulative distribution function at x.
- `"RightSawtoothDist(a;b)"`: Generates a random number based on this distribution.

9.35 Right sawtooth distribution with mean a and standard deviation b

- `"RightSawtoothDistDirect(x;a;b;0)"`: Calculates the probability density at x.

- `"RightSawtoothDistDirect(x;a;b;1)"`: Calculates the cumulative distribution function at x.
- `"RightSawtoothDistDirect(a;b)"`: Generates a random number based on this distribution.

9.36 Distribution based on empirical values

- `"EmpiricalDensity(x;value1;value2;value3;...;max)"`:
  Calculates the probability density at x. The specified values will be used for the density in the range from 0 to max.
- `"EmpiricalDistribution(x;value1;value2;value3;...;max)"`:
  Calculates the cumulative distribution function at x. The specified values will be used for the density in the range from 0 to max.
- `"EmpiricalRandom(value1;value2;value3;...;max)"`:
  Generates a random number based on this distribution. The specified values will be used for the density in the range from 0 to max.
- `"EmpiricalDistributionMean(value1;value2;value3;...;max)"`:
  Calculates the expected value of the distribution.
- `"EmpiricalDistributionMedian(value1;value2;value3;...;max)"`:
  Calculates the median of the distribution.
- `"EmpiricalDistributionQuantil(value1;value2;value3;...;max;p)"`:
  Calculates the quantil for the probability p of the distribution.
- `"EmpiricalDistributionSD(value1;value2;value3;...;max)"`:
  Calculates the standard deviation of the distribution.
- `"EmpiricalDistributionVar(value1;value2;value3;...;max)"`:
  Calculates the variance of the distribution.
- `"EmpiricalDistributionCV(value1;value2;value3;...;max)"`:
  Calculates the coefficient of variation of the distribution.

Chapter 10
Erlang C calculator

By using the following command some performance indicators can be calculated using the extended Erlang C formula:

- `"ErlangC(lambda;mu;nu;c;K;-1)"`:
  Calculates the average queue length $\mathbf{E}[N_Q]$.
- `"ErlangC(lambda;mu;nu;c;K;-2)"`:
  Calculates the average number of clients in the system $\mathbf{E}[N]$.
- `"ErlangC(lambda;mu;nu;c;K;-3)"`:
  Calculates the average waiting time $\mathbf{E}[W]$.
- `"ErlangC(lambda;mu;nu;c;K;-4)"`:
  Calculates the average residence time $\mathbf{E}[V]$.
- `"ErlangC(lambda;mu;nu;c;K;-5)"`:
  Calculates the average accessibility $1 - P(A)$.
- `"ErlangC(lambda;mu;nu;c;K;t)"`:
  Calculates the the probability for the service level at the t seconds threshold $P(W \leq t)$.

The parameters have the following meanings:

- `lambda`:
  Arrival rate λ (in clients per time unit), i.e. inverse of the mean inter-arrival time.
- `mu`:
  Service rate μ (in clients per time unit), i.e. inverse of the mean service time.
- `nu`:
  Cancelation rate ν (in clients per time unit), i.e. inverse of the mean waiting time tolerance.
- `c`:
  Number of available parallel operating servers.
- `K`:
  Number of available places in the system (waiting and processing places together, i.e. it is $K \geq c$).

Chapter 11
Allen-Cunneen approximation formula

By using the following command some performance indicators can be calculated using the Allen-Cunneen approximation formula:

- `"AllenCunneen(lambda;mu;cvI;cvS;c;-1)"`:
  Calculates the average queue length $E[N_Q]$.
- `"AllenCunneen(lambda;mu;cvI;cvS;c;-2)"`:
  Calculates the average number of clients in the system $\mathbf{E}[N]$.
- `"AllenCunneen(lambda;mu;cvI;cvS;c;-3)"`:
  Calculates the average waiting time $\mathbf{E}[W]$.
- `"AllenCunneen(lambda;mu;cvI;cvS;c;-4)"`:
  Calculates the average residence time $\mathbf{E}[V]$.

The parameters have the following meanings:

- `lambda`:
  Arrival rate λ (in clients per time unit), i.e. inverse of the mean inter-arrival time.
- `mu`:
  Service rate μ (in clients per time unit), i.e. inverse of the mean service time.
- `cvI`:
  Coefficient of variation of the inter-arrival times $\mathbf{CV}[I]$ (small values mean that the inter-arrival times are very homogeneous).
- `cvS`:
  Coefficient of variation of the service times $\mathbf{CV}[S]$ (small values mean that the operations are very homogeneous).
- `c`:
  Number of available parallel operating servers.

Chapter 12
Accessing model properties

12.1 General simulation data

- `"SimTime()"` or `"TNow()"`:
  Gets the current simulation time in seconds.
- `"WarmUp()"` or `"isWarmUp()"`:
  Get 1 if the simulation is in the warm-up phase, otherwise 0.
- `"RepeatCurrent()"`:
  Gets the current repeat number of the simulation (1-based value).
- `"RepeatCount()"`:
  Gets the number of planned repeats of the simulation.
- `"$("Name")"`:
  Gets the ID of the element with the name which is enter between the quotation marks. If there is not station with this name, the function will return -1.
- `"$("Key")"`:
  Returns the value from the map which can be accessed by `getMapGlobal()` at scripting elements.

12.2 Clients in the system

12.2.1 Number of clients in the system

- `"WIP()"` or `"N()"` or `"Station()"`:
  Gets the current total number of clients in the system.
- `"WIP_avg()"` or `"Station_avg()"` or `"N_avg()"` or `"WIP_Mittelwert()"` or `"Station_Mittelwert()"` or `"N_Mittelwert()"`:
  Gets the average number of clients in the system.

  `"WIP_median()"` or `"Station_median()"` or `"N_median()"`:
  Gets the median of the number of clients in the system.

  `"WIP_quantil(p)"` or `"Station_quantil(p)"` or `"N_quantil(p)"`:
  Gets the quantil for the probability p of the number of clients in the system.
- `"WIP_min()"` or `"Station_min()"` or `"N_min()"` or `"WIP_Minimum()"` or `"Station_Minimum()"` or `"N_Minimum()"`:
  Gets the minimal number of clients in the system.

- `"WIP_max()"` or `"Station_max()"` or `"N_max()"` or `"WIP_Maximum()"` or `"Station_Maximum()"` or `"N_Maximum()"`:
  Gets the maximal number of clients in the system.
- `"WIP_var()"` or `"Station_var()"` or `"N_var()"` or `"WIP_Varianz()"` or `"Station_Varianz()"` or `"N_Varianz()"`:
  Gets the variation of the number of clients in the system.
- `"WIP_sd()"` or `"Station_sd()"` or `"N_sd()"` or `"WIP_Standardabweichung()"` or `"Station_Standardabweichung()"` or `"N_Standardabweichung()"`:
  Gets the standard deviation of the number of clients in the system.
- `"WIP_cv()"` or `"Station_cv()"` or `"N_cv()"`:
  Gets the coefficient of variation of the number of clients in the system.
- `"WIP_scv()"` or `"Station_scv()"` or `"N_scv()"`:
  Gets the squared coefficient of variation of the number of clients in the system.
- `"WIP_sk()"` or `"Station_sk()"` or `"N_sk()"`:
  Gets the skewness of the number of clients in the system.
- `"WIP_kurt()"` or `"Station_kurt()"` or `"N_kurt()"`:
  Gets the excess kurtosis of the number of clients in the system.

12.2.2 Number of waiting clients in the system

- `"NQ()"` or `"Queue()"` or `"Schlange()"` or `"Warteschlange()"`:
  Gets the current total number of waiting clients in the system.
- `"NQ_avg()"` or `"Queue_avg()"` or `"Schlange_avg()"` or `"Warteschlange_avg()"` or `"NQ_Mittelwert()"` or `"Queue_Mittelwert()"` or `"Schlange_Mittelwert()"` or `"Warteschlange_Mittelwert()"`:
  Gets the average number of waiting clients in the system.
- `"NQ_median()"` or `"Queue_median()"` or `"Schlange_median()"` or `"Warteschlange_median()"`:
  Gets the median of the number of clients in all queues.
- `"NQ_quantil(p)"` or `"Queue_quantil(p)"` or `"Schlange_quantil(p)"` or `"Warteschlange_quantil(p)"`:
  Gets the quantil for the probability p of the number of clients in all queues.
- `"NQ_min()"` or `"Queue_min()"` or `"Schlange_min()"` or `"Warteschlange_min()"` or `"NQ_Minimum()"` or `"Queue_Minimum()"` or `"Schlange_Minimum()"` or `"Warteschlange_Minimum()"`:
  Gets the minimal number of waiting clients in the system.
- `"NQ_max()"` or `"Queue_max()"` or `"Schlange_max()"` or `"Warteschlange_max()"` or `"NQ_Maximum()"` or `"Queue_Maximum()"` or `"Schlange_Maximum()"` or `"Warteschlange_Maximum()"`:
  Gets the maximal number of waiting clients in the system.
- `"NQ_var()"` or `"Queue_var()"` or `"Schlange_var()"` or `"Warteschlange_var()"` or `"NQ_Varianz()"` or `"Queue_Varianz()"` or `"Schlange_Varianz()"` or `"Warteschlange_Varianz()"`:
  Gets the variation of the number of waiting clients in the system.
- `"NQ_sd()"` or `"Queue_sd()"` or `"Schlange_sd()"` or `"Warteschlange_sd()"` or `"NQ_Standardabweichung()"` or `"Queue_Standardabweichung()"` or

"Schlange_Standardabweichung()" or "Warteschlange_Standardabweichung()":
Gets the standard deviation of the number of waiting clients in the system.

- "NQ_cv()" or "Queue_cv()" or "Schlange_cv()" or "Warteschlange_cv()":
Gets the coefficient of variation of the number of waiting clients in the system.
- "NQ_scv()" or "Queue_scv()" or "Schlange_scv()" or "Warteschlange_scv()":
Gets the squared coefficient of variation of the number of waiting clients in the system.
- "NQ_sk()" or "Queue_sk()" or "Schlange_sk()" or "Warteschlange_sk()":
Gets the skewness of the number of waiting clients in the system.
- "NQ_kurt()" or "Queue_kurt()" or "Schlange_kurt()" or "Warteschlange_kurt()":
Gets the excess kurtosis of the number of waiting clients in the system.

12.3 Clients at the stations

12.3.1 Number of clients at a station

- "WIP(id)" or "N(id)" or "Station(id)":
Gets the current total number of clients at station **id**.
- "WIP(id1;id2)" or "N(id1;id2)" or "Station(id1;id2)":
Gets the current total number of clients at station **id1**. Only clients of the type, whos name appears at the source or the type assignment **id2**, are respected.
- "WIP_avg(id)" or "Station_avg(id)" or "N_avg(id)" or "WIP_Mittelwert(id)" or "Station_Mittelwert(id)" or "N_Mittelwert(id)":
Gets the average number of clients at station **id**.
- "WIP_median(id)" or "Station_median(id)" or "N_median(id)":
Gets the medium of the number of clients at station **id**.
- "WIP_quantil(p;id)" or "Station_quantil(p;id)" or "N_quantil(p;id)":
Gets the quantil for the probability p of the clients at station **id**.
- "WIP_min(id)" or "Station_min(id)" or "N_min(id)" or "WIP_Minimum(id)" or "Station_Minimum(id)" or "N_Minimum(id)":
Gets the minimal number of clients at station **id**.
- "WIP_max(id)" or "Station_max(id)" or "N_max(id)" or "WIP_Maximum(id)" or "Station_Maximum(id)" or "N_Maximum(id)":
Gets the maximal number of clients at station **id**.
- "WIP_var(id)" or "Station_var(id)" or "N_var(id)" or "WIP_Varianz(id)" or "Station_Varianz(id)" or "N_Varianz(id)":
Gets the variation of the number of clients at station **id**.
- "WIP_sd(id)" or "Station_sd(id)" or "N_sd(id)" or "WIP_Standardabweichung(id)" or "Station_Standardabweichung(id)" or "N_Standardabweichung(id)":
Gets the standard deviation of the number of clients at station **id**.
- "WIP_cv(id)" or "Station_cv(id)" or "N_cv(id)":
Gets the coefficient of variation of the number of clients at station **id**.
- "WIP_scv(id)" or "Station_scv(id)" or "N_scv(id)":
Gets the squared coefficient of variation of the number of clients at station **id**.

- "WIP_sk(id)" or "Station_sk(id)" or "N_sk(id)":
  Gets the skewness of the number of clients at station **id**.
- "WIP_kurt(id)" or "Station_kurt(id)" or "N_kurt(id)":
  Gets the excess kurtosis of the number of clients at station **id**.
- "WIP_hist(id;state)" or "Station_hist(id;state)" or "N_hist(id;state)":
  Gets the fraction of time, the system was in the given **state** in relation of the number of clients at stations **id**.
- "WIP_hist(id;stateA;stateB)" or "Station_hist(id;stateA;stateB)" or "N_hist(id;stateA;stateB)":
  Gets the fraction of time, when there are more than **stateA** and at most **stateB** clients at station **id**.

12.3.2 Number of clients at the queue at a station

- "NQ(id)" or "Queue(id)" or "Schlange(id)" or "Warteschlange(id)":
  Gets the current total number of clients in the queue at station **id**.
- "NQ(id;nr)" or "Queue(id;nr)" or "Schlange(id;nr)" or "Warteschlange(id;nr)":
  Gets the current total number of clients in the partial queue <**nr** (1 based) at station **id**. (This command can only be used with "Match" elements.)
- "NQ_avg(id)" or "Queue_avg(id)" or "Schlange_avg(id)" or "Warteschlange_avg(id)" or "NQ_Mittelwert(id)" or "Queue_Mittelwert(id)" or "Schlange_Mittelwert(id)" or "Warteschlange_Mittelwert(id)":
  Gets the average number of clients in the queue at station **id**.
- "NQ_median(id)" or "Queue_median(id)" or "Schlange_median(id)" or "Warteschlange_median(id)":
  Gets the median of the number of clients in the queue at station **id**.
- "NQ_quantil(p;id)" or "Queue_quantil(p;id)" or "Schlange_quantil(p;id)" or "Warteschlange_quantil(p;id)":
  Gets the quantil for the probability p of the number of clients in the queue at station **id**.
- "NQ_min(id)" or "Queue_min(id)" or "Schlange_min(id)" or "Warteschlange_min(id)" or "NQ_Minimum(id)" or "Queue_Minimum(id)" or "Schlange_Minimum(id)" or "Warteschlange_Minimum(id)":
  Gets the minimal number of clients in the queue at station **id**.
- "NQ_max(id)" or "Queue_max(id)" or "Schlange_max(id)" or "Warteschlange_max(id)" or "NQ_Maximum(id)" or "Queue_Maximum(id)" or "Schlange_Maximum(id)" or "Warteschlange_Maximum(id)":
  Gets the maximal number of clients in the queue at station **id**.
- "NQ_var(id)" or "Queue_var(id)" or "Schlange_var(id)" or "Warteschlange_var(id)" or "NQ_Varianz(id)" or "Queue_Varianz(id)" or "Schlange_Varianz(id)" or "Warteschlange_Varianz(id)":
  Gets the variation of the number of clients in the queue at station **id**.
- "NQ_sd(id)" or "Queue_sd(id)" or "Schlange_sd(id)" or "Warteschlange_sd(id)" or "NQ_Standardabweichung(id)" or "Queue_Standardabweichung(id)" or "Schlange_Standardabweichung(id)" or "Warteschlange_Standardabweichung(id)":
  Gets the standard deviation of the number of clients in the queue at station **id**.
- "NQ_cv(id)" or "Queue_cv(id)" or "Schlange_cv(id)" or "Warteschlange_cv(id)":
  Gets the coefficient of variation of the number of clients in the queue at station **id**.

- "NQ_scv(id)" or "Queue_scv(id)" or "Schlange_scv(id)" or "Warteschlange_scv(id)":
  Gets the squared coefficient of variation of the number of clients in the queue at station id.
- "NQ_sk(id)" or "Queue_sk(id)" or "Schlange_sk(id)" or "Warteschlange_sk(id)":
  Gets the skewness of the number of clients in the queue at station id.
- "NQ_kurt(id)" or "Queue_kurt(id)" or "Schlange_kurt(id)" or "Warteschlange_kurt(id)":
  Gets the excess kurtosis of the number of clients in the queue at station id.
- "NQ_hist(id;state)" or "Queue_hist(id;state)" or "Schlange_hist(id;state)" or "Warteschlange_hist(id;state)":
  Gets the fraction of time, when there are state clients in the queue at station id.
- "NQ_hist(id;stateA;stateB)" or "Queue_hist(id;stateA;stateB)" or "Schlange_hist(id;stateA;stateB)" or "Warteschlange_hist(id;stateA;stateB)":
  Gets the fraction of time, when there are more than stateA and at most stateB clients in the queue at station id.

12.3.3 Number of clients just being served at a station

- "Process(id)":
  Gets the current number of clients just being served at station id.

12.3.4 Number of arrivals and departures at a station

- "NumberIn(id)" or "CountIn(id)":
  Gets the number of client arrivals at station id.
- "NumberOut(id)" or "CountOut(id)":
  Gets the number of client departures at station id.

12.4 Clients in system by client type

12.4.1 Number of clients in the system by client type

- "WIP(id)" or "N(id)" or "Station(id)":
  Gets the current total number of clients, whos name appears at the source or the type assignment id.
- "WIP_avg(id)" or "Station_avg(id)" or "N_avg(id)" or "WIP_Mittelwert(id)" or "Station_Mittelwert(id)" or "N_Mittelwert(id)":
  Gets the average number of clients, whos name appears at the source or the type assignment id.
- "WIP_median(id)" or "Station_median(id)" or "N_median(id)":
  Gets the median of the number of clients, whos name appears at the source or the type assignment id, in the system.
- "WIP_quantil(p;id)" or "Station_quantil(p;id)" or "N_quantil(p;id)":
  Gets the quantil for the probability p of the number of clients, whos name appears at the source or the type assignment id, in the system.
- "WIP_min(id)" or "Station_min(id)" or "N_min(id)" or "WIP_Minimum(id)" or "Station_Minimum(id)" or "N_Minimum(id)":
  Gets the minimal number of clients, whos name appears at the source or the type assignment id.

- `"WIP_max(id)"` or `"Station_max(id)"` or `"N_max(id)"` or `"WIP_Maximum(id)"` or `"Station_Maximum(id)"` or `"N_Maximum(id)"`:
  Gets the maximal number of clients, whos name appears at the source or the type assignment `id`.
- `"WIP_var(id)"` or `"Station_var(id)"` or `"N_var(id)"` or `"WIP_Varianz(id)"` or `"Station_Varianz(id)"` or `"N_Varianz(id)"`:
  Gets the variation of the number of clients, whos name appears at the source or the type assignment `id`.
- `"WIP_sd(id)"` or `"Station_sd(id)"` or `"N_sd(id)"` or `"WIP_Standardabweichung(id)"` or `"Station_Standardabweichung(id)"` or `"N_Standardabweichung(id)"`:
  Gets the standard deviation of the number of clients, whos name appears at the source or the type assignment `id`.
- `"WIP_cv(id)"` or `"Station_cv(id)"` or `"N_cv(id)"`:
  Gets the coefficient of variation of the number of clients, whos name appears at the source or the type assignment `id`.
- `"WIP_scv(id)"` or `"Station_scv(id)"` or `"N_scv(id)"`:
  Gets the squared coefficient of variation of the number of clients, whos name appears at the source or the type assignment `id`.
- `"WIP_sk(id)"` or `"Station_sk(id)"` or `"N_sk(id)"`:
  Gets the skewness of the number of clients, whos name appears at the source or the type assignment `id`.
- `"WIP_kurt(id)"` or `"Station_kurt(id)"` or `"N_kurt(id)"`:
  Gets the excess kurtosis of the number of clients, whos name appears at the source or the type assignment `id`.
- `"WIP_hist(id;state)"` or `"Station_hist(id;state)"` or `"N_hist(id;state)"`:
  Gets the fraction of time, the system was in the given `state` in relation of the number of clients, whos name appears at the source or the type assignment `id`.
- `"WIP_hist(id;stateA;stateB)"` or `"Station_hist(id;stateA;stateB)"` or `"N_hist(id;stateA;stateB)"`:
  Gets the fraction of time, when there are more than `stateA` and at most `stateB` clients, whos name appears at the source or the type assignment `id`, in the system.

12.4.2 Number of waiting clients in the system by client type

- `"NQ(id)"` or `"Queue(id)"` or `"Schlange(id)"` or `"Warteschlange(id)"`:
  Gets the current total number of waiting clients, whos name appears at the source or the type assignment `id`.
- `"NQ_avg(id)"` or `"Queue_avg(id)"` or `"Schlange_avg(id)"` or `"Warteschlange_avg(id)"` or `"NQ_Mittelwert(id)"` or `"Queue_Mittelwert(id)"` or `"Schlange_Mittelwert(id)"` or `"Warteschlange_Mittelwert(id)"`:
  Gets the average number of waiting clients, whos name appears at the source or the type assignment `id`.
- `"NQ_median(id)"` or `"Queue_median(id)"` or `"Schlange_median(id)"` or `"Warteschlange_median(id)"`:
  Gets the median of the number of waiting clients, whos name appears at the source or the type assignment id, in the system.
- `"NQ_quantil(p;id)"` or `"Queue_quantil(p;id)"` or `"Schlange_quantil(p;id)"` or `"Warteschlange_quantil(p;id)"`:

Gets the quantil for the probability p of the number of waiting clients, whos name appears at the source or the type assignment id, in the system.

- `"NQ_min(id)"` or `"Queue_min(id)"` or `"Schlange_min(id)"` or `"Warteschlange_min(id)"` or `"NQ_Minimum(id)"` or `"Queue_Minimum(id)"` or `"Schlange_Minimum(id)"` or `"Warteschlange_Minimum(id)"`:
  Gets the minimal number of waiting clients, whos name appears at the source or the type assignment `id`.
- `"NQ_max(id)"` or `"Queue_max(id)"` or `"Schlange_max(id)"` or `"Warteschlange_max(id)"` or `"NQ_Maximum(id)"` or `"Queue_Maximum(id)"` or `"Schlange_Maximum(id)"` or `"Warteschlange_Maximum(id)"`:
  Gets the maximal number of waiting clients, whos name appears at the source or the type assignment `id`.
- `"NQ_var(id)"` or `"Queue_var(id)"` or `"Schlange_var(id)"` or `"Warteschlange_var(id)"` or `"NQ_Varianz(id)"` or `"Queue_Varianz(id)"` or `"Schlange_Varianz(id)"` or `"Warteschlange_Varianz(id)"`:
  Gets the variation of the number of waiting clients, whos name appears at the source or the type assignment `id`.
- `"NQ_sd(id)"` or `"Queue_sd(id)"` or `"Schlange_sd(id)"` or `"Warteschlange_sd(id)"` or `"NQ_Standardabweichung(id)"` or `"Queue_Standardabweichung(id)"` or `"Schlange_Standardabweichung(id)"` or `"Warteschlange_Standardabweichung(id)"`:
  Gets the standard deviation of the number of waiting clients, whos name appears at the source or the type assignment `id`.
- `"NQ_cv(id)"` or `"Queue_cv(id)"` or `"Schlange_cv(id)"` or `"Warteschlange_cv(id)"`:
  Gets the coefficient of variation of the number of waiting clients, whos name appears at the source or the type assignment `id`.
- `"NQ_scv(id)"` or `"Queue_scv(id)"` or `"Schlange_scv(id)"` or `"Warteschlange_scv(id)"`:
  Gets the squared coefficient of variation of the number of waiting clients, whos name appears at the source or the type assignment `id`.
- `"NQ_sk(id)"` or `"Queue_sk(id)"` or `"Schlange_sk(id)"` or `"Warteschlange_sk(id)"`:
  Gets the skewness of the number of waiting clients, whos name appears at the source or the type assignment `id`.
- `"NQ_kurt(id)"` or `"Queue_kurt(id)"` or `"Schlange_kurt(id)"` or `"Warteschlange_kurt(id)"`:
  Gets the excess kurtosis of the number of waiting clients, whos name appears at the source or the type assignment `id`.
- `"NQ_hist(id;state)"` or `"Queue_hist(id;state)"` or `"Schlange_hist(id;state)"` or `"Warteschlange_hist(id;state)"`:
  Gets the fraction of time, when there are `state` waiting clients, whos name appears at the source or the type assignment `id`, are in the system.
- `"NQ_hist(id;stateA;stateB)"` or `"Queue_hist(id;stateA;stateB)"` or `"Schlange_hist(id;stateA;stateB)"` or `"Warteschlange_hist(id;stateA;stateB)"`:
  Gets the fraction of time, when there are more than `stateA` and at most `stateB` waiting clients, whos name appears at the source or the type assignment `id`, in the system.

12.5 Counter and throughput

- `"Counter(id)"` or `"Value(id)"`:
  Gets the value of the counter at station `id`.
  (Can only be applied on "Difference counter", "Counter" and "Throughput" elements.)
- `"Anteil(id)"` or `"Part(id)"`:
  Gets the share of the counter value in the counter group.
  (Can only be applied on "Counter" elements.)
- `"Durchsatz(id)"` or `"Throughput(id)"`:
  Gets the throughput measured in arrivals per second at station `id`.
- `"Durchsatz()"` or `"Throughput()"`:
  Gets the throughput measured in arrivals per second in the system.
- `"DurchsatzMax(id)"` or `"ThroughputMax(id)"`:
  Gets the maximum measured throughput measured in arrivals per second at station `id`.
- `"DurchsatzMaxIntervall(id)"` or `"ThroughputMaxInterval(id)"`:
  Gets the interval length in seconds used to record the maximum throughput at station `id`.

12.6 Waiting times

12.6.1 Waiting times at a station

- `"WaitingTime_sum(id)"` or `"WaitingTime_gesamt(id)"` or `"WaitingTime_summe(id)"`:
  Gets the sum of the waiting times at the station `id` (in seconds).
- `"WaitingTime_avg(id)"` or `"WaitingTime_average(id)"` or `"WaitingTime_Mittelwert(id)"`:
  Gets the average waiting time of the clients at station `id` (in seconds).
- `"WaitingTime_median(id)"`:
  Gets the median of the waiting times of the clients at station `id` (in seconds).
- `"WaitingTime_quantil(p;id)"`:
  Gets the quantil for the probability p of the waiting times of the clients at station `id` (in seconds).
- `"WaitingTime_min(id)"` or `"WaitingTime_Minimum(id)"`:
  Gets the minimum waiting time of the clients at station `id` (in seconds).
- `"WaitingTime_max(id)"` or `"WaitingTime_Maximum(id)"`:
  Gets the maximum waiting time of the clients at station `id` (in seconds).
- `"WaitingTime_var(id)"` or `"WaitingTime_Varianz(id)"`:
  Gets the variation of the waiting times of the clients at station `id` (based on seconds).
- `"WaitingTime_sd(id)"` or `"WaitingTime_Standardabweichung(id)"`:
  Gets the standard deviation of the waiting times of the clients at station `id` (based on seconds).
- `"WaitingTime_cv(id)"`:
  Gets the coefficient of variation of the waiting times of the clients at station `id`.
- `"WaitingTime_scv(id)"`:
  Gets the squared coefficient of variation of the waiting times of the clients at station `id`.
- `"WaitingTime_sk(id)"`:
  Gets the skewness of the waiting times of the clients at station `id`.

- **"WaitingTime_kurt(id)"**:
  Gets the excess kurtosis of the waiting times of the clients at station **id**.
- **"WaitingTime_hist(id;time)"**:
  Gets the fraction of clients for which the waiting time at stations **id** was **time** seconds.
- **"WaitingTime_hist(id;timeA;timeB)"**:
  Gets the fraction of clients for which the waiting time at stations **id** was more than **timeA** and at most **timeB** seconds.

12.6.2 Waiting times over all client types

- **"WaitingTime_avg()"** or **"WaitingTime_average()"** or **"WaitingTime_Mittelwert()"**:
  Get the average waiting time over all clients (in seconds).
- **"WaitingTime_median()"**:
  Get the median of the waiting times over all clients (in seconds).
- **"WaitingTime_quantil(p)"**:
  Gets the quantil for the probability p of the waiting times over all clients (in seconds).
- **"WaitingTime_min()"** or **"WaitingTime_Minimum()"**:
  Get the minimum waiting time over all clients (in seconds).
- **"WaitingTime_max()"** or **"WaitingTime_Maximum()"**:
  Get the maximum waiting time over all clients (in seconds).
- **"WaitingTime_var()"** or **"WaitingTime_Varianz()"**:
  Gets the variation of the waiting times over all clients (based on seconds).
- **"WaitingTime_sd()"** or **"WaitingTime_Standardabweichung()"**:
  Gets the standard deviation of the waiting times over all clients (based on seconds).
- **"WaitingTime_cv()"**:
  Gets the coefficient of variation of the waiting times over all clients.
- **"WaitingTime_scv()"**:
  Gets the squared coefficient of variation of the waiting times over all clients.
- **"WaitingTime_sk()"**:
  Gets the skewness of the waiting times over all clients.
- **"WaitingTime_kurt()"**:
  Gets the excess kurtosis of the waiting times over all clients.
- **"WaitingTime_histAll(time)"**:
  Gets the fraction of clients for which the waiting time was **time** seconds.
- **"WaitingTime_histAll(timeA;timeB)"**:
  Gets the fraction of clients for which the waiting time was more than **timeA** and at most **timeB** seconds.

12.6.3 Waiting times for a specific client type

- **"WaitingTime_sum(id)"** or **"WaitingTime_gesamt(id)"** or **"WaitingTime_summe(id)"**: Gets the sum of the waiting times of the clients, whos name appears at the source or the type assignment **id** (in seconds).

- `"WaitingTime_avg(id)"` or `"WaitingTime_average(id)"` or `"WaitingTime_Mittelwert(id)"`:
  Gets the average waiting time of the clients, whos name appears at the source or the type assignment `id` (in seconds).
- `"WaitingTime_median(id)"`:
  Gets the median of the waiting times of the clients, whos name appears at the source or the type assignment `id` (in seconds).
- `"WaitingTime_quantil(p;id)"`:
  Gets the quantil for the probability p of the waiting times of the clients, whos name appears at the source or the type assignment `id` (in seconds).
- `"WaitingTime_min(id)"` or `"WaitingTime_Minimum(id)"`:
  Gets the minimum waiting time of the clients, whos name appears at the source or the type assignment `id` (in seconds).
- `"WaitingTime_max(id)"` or `"WaitingTime_Maximum(id)"`:
  Gets the maximum waiting time of the clients, whos name appears at the source or the type assignment `id` (in seconds).
- `"WaitingTime_var(id)"` or `"WaitingTime_Varianz(id)"`:
  Gets the variation of the waiting times of the clients, whos name appears at the source or the type assignment `id` (based on seconds).
- `"WaitingTime_sd(id)"` or `"WaitingTime_Standardabweichung(id)"`:
  Gets the standard deviation of the waiting times of the clients, whos name appears at the source or the type assignment `id` (based on seconds).
- `"WaitingTime_cv(id)"`:
  Gets the coefficient of variation of the waiting times of the clients, whos name appears at the source or the type assignment `id`.
- `"WaitingTime_scv(id)"`:
  Gets the squared coefficient of variation of the waiting times of the clients, whos name appears at the source or the type assignment `id`.
- `"WaitingTime_sk(id)"`:
  Gets the skewness of the waiting times of the clients, whos name appears at the source or the type assignment `id`.
- `"WaitingTime_kurt(id)"`:
  Gets the excess kurtosis of the waiting times of the clients, whos name appears at the source or the type assignment `id`.

12.7 Transfer times

12.7.1 Transfer times at a station

- `"TransferTime_sum(id)"` or `"TransferTime_gesamt(id)"` or `"TransferTime_summe(id)"`:
  Gets the sum of the transfer times at the station `id` (in seconds).
- `"TransferTime_avg(id)"` or `"TransferTime_average(id)"` or `"TransferTime_Mittelwert(id)"`:
  Gets the average transfer time of the clients at station `id` (in seconds).
- `"TransferTime_median(id)"`:
  Gets the median of the transfer times of the clients at station `id` (in seconds).

- **"TransferTime_quantil(p;id)"**:
  Gets the quantil for the probability p of the transfer times of the clients at station **id** (in seconds).
- **"TransferTime_min(id)"** or **"TransferTime_Minimum(id)"**:
  Gets the minimum transfer time of the clients at station **id** (in seconds).
- **"TransferTime_max(id)"** or **"TransferTime_Maximum(id)"**:
  Gets the maximum transfer time of the clients at station **id** (in seconds).
- **"TransferTime_var(id)"** or **"TransferTime_Varianz(id)"**:
  Gets the variation of the transfer times of the clients at station **id** (based on seconds).
- **"TransferTime_sd(id)"** or **"TransferTime_Standardabweichung(id)"**:
  Gets the standard deviation of the transfer times of the clients at station **id** (based on seconds).
- **"TransferTime_cv(id)"**:
  Gets the coefficient of variation of the transfer times of the clients at station **id**.
- **"TransferTime_scv(id)"**:
  Gets the squared coefficient of variation of the transfer times of the clients at station **id**.
- **"TransferTime_sk(id)"**:
  Gets the skewness of the transfer times of the clients at station **id**.
- **"TransferTime_kurt(id)"**:
  Gets the excess kurtosis of the transfer times of the clients at station **id**.
- **"TransferTime_hist(id;time)"**:
  Gets the fraction of clients for which the transfer time at stations **id** was **time** seconds.
- **"TransferTime_hist(id;timeA;timeB)"**:
  Gets the fraction of clients for which the transfer time at stations **id** was more than **timeA** and at most **timeB** seconds.

12.7.2 Transfer times over all client types

- **"TransferTime_avg()"** or **"TransferTime_average()"** or **"TransferTime_Mittelwert()"**:
  Get the average waiting time over all clients (in seconds).
- **"TransferTime_median()"**:
  Get the median of the transfer times over all clients (in seconds).
- **"TransferTime_quantil(p)"**:
  Gets the quantil for the probability p of the transfer times over all clients (in seconds).
- **"TransferTime_min()"** or **"TransferTime_Minimum()"**:
  Get the minimum transfer time over all clients (in seconds).
- **"TransferTime_max()"** or **"TransferTime_Maximum()"**:
  Get the maximum transfer time over all clients (in seconds).
- **"TransferTime_var()"** or **"TransferTime_Varianz()"**:
  Gets the variation of the transfer times over all clients (based on seconds).
- **"TransferTime_sd()"** or **"TransferTime_Standardabweichung()"**:
  Gets the standard deviation of the transfer times over all clients (based on seconds).
- **"TransferTime_cv()"**:
  Gets the coefficient of variation of the transfer times over all clients.
- **"TransferTime_scv()"**:
  Gets the squared coefficient of variation of the transfer times over all clients.

- **"TransferTime_sk()"**:
  Gets the skewness of the transfer times over all clients.
- **"TransferTime_kurt()"**:
  Gets the excess kurtosis of the transfer times over all clients.
- **"TransferTime_histAll(time)"**:
  Gets the fraction of clients for which the transfer time was **time** seconds.
- **"TransferTime_histAll(timeA;timeB)"**:
  Gets the fraction of clients for which the transfer time was more than **timeA** and at most **timeB** seconds.

12.7.3 Transfer times for a specific client type

- **"TransferTime_sum(id)"** or **"TransferTime_gesamt(id)"** or **"TransferTime_summe(id)"**:
  Gets the sum of the transfer times of the clients, whos name appears at the source or the type assignment **id** (in seconds).
- **"TransferTime_avg(id)"** or **"TransferTime_average(id)"** or **"TransferTime_Mittelwert(id)"**:
  Gets the average transfer time of the clients, whos name appears at the source or the type assignment **id** (in seconds).
- **"TransferTime_median(id)"**:
  Gets the median of the transfer times of the clients, whos name appears at the source or the type assignment **id** (in seconds).
- **"TransferTime_quantil(p;id)"**:
  Gets the quantil for the probability p of the transfer times of the clients, whos name appears at the source or the type assignment **id** (in seconds).
- **"TransferTime_min(id)"** or **"TransferTime_Minimum(id)"**:
  Gets the minimum transfer time of the clients, whos name appears at the source or the type assignment **id** (in seconds).
- **"TransferTime_max(id)"** or **"TransferTime_Maximum(id)"**:
  Gets the maximum transfer time of the clients, whos name appears at the source or the type assignment **id** (in seconds).
- **"TransferTime_var(id)"** or **"TransferTime_Varianz(id)"**:
  Gets the variation of the transfer times of the clients, whos name appears at the source or the type assignment **id** (based on seconds).
- **"TransferTime_sd(id)"** or **"TransferTime_Standardabweichung(id)"**:
  Gets the standard deviation of the transfer times of the clients, whos name appears at the source or the type assignment **id** (based on seconds).
- **"TransferTime_cv(id)"**:
  Gets the coefficient of variation of the transfer times of the clients, whos name appears at the source or the type assignment **id**.
- **"TransferTime_scv(id)"**:
  Gets the squared coefficient of variation of the transfer times of the clients, whos name appears at the source or the type assignment **id**.
- **"TransferTime_sk(id)"**:
  Gets the skewness of the transfer times of the clients, whos name appears at the source or the type assignment **id**.

- **"TransferTime_kurt(id)"**:
  Gets the excess kurtosis of the transfer times of the clients, whos name appears at the source or the type assignment **id**.

12.8 Process times

12.8.1 Process times at a station

- **"ProcessTime_sum(id)"** or **"ProcessTime_gesamt(id)"** or **"ProcessTime_summe(id)"**:
  Gets the sum of the process times at the station **id** (in seconds).
- **"ProcessTime_avg(id)"** or **"ProcessTime_average(id)"** or **"ProcessTime_Mittelwert(id)"**:
  Gets the average process time of the clients at station **id** (in seconds).
- **"ProcessTime_median(id)"**:
  Gets the median of the process times of the clients at station **id** (in seconds).
- **"ProcessTime_quantil(p;id)"**:
  Gets the quantil for the probability p of the process times of the clients at station **id** (in seconds).
- **"ProcessTime_min(id)"** or **"ProcessTime_Minimum(id)"**:
  Gets the minimum process time of the clients at station **id** (in seconds).
- **"ProcessTime_max(id)"** or **"ProcessTime_Maximum(id)"**:
  Gets the maximum process time of the clients at station **id** (in seconds).
- **"ProcessTime_var(id)"** or **"ProcessTime_Varianz(id)"**:
  Gets the variation of the process times of the clients at station **id** (based on seconds).
- **"ProcessTime_sd(id)"** or **"ProcessTime_Standardabweichung(id)"**:
  Gets the standard deviation of the process times of the clients at station **id** (based on seconds).
- **"ProcessTime_cv(id)"**:
  Gets the coefficient of variation of the process times of the clients at station **id**.
- **"ProcessTime_scv(id)"**:
  Gets the squared coefficient of variation of the process times of the clients at station **id**.
- **"ProcessTime_sk(id)"**:
  Gets the skewness of the process times of the clients at station **id**.
- **"ProcessTime_kurt(id)"**:
  Gets the excess kurtosis of the process times of the clients at station **id**.
- **"ProcessTime_hist(id;time)"**:
  Gets the fraction of clients for which the process time at stations **id** was **time** seconds.
- **"ProcessTime_hist(id;timeA;timeB)"**:
  Gets the fraction of clients for which the process time at stations **id** was more than **timeA** and at most **timeB** seconds.

12.8.2 Process times over all client types

- **"ProcessTime_avg()"** or **"ProcessTime_average()"** or **"ProcessTime_Mittelwert()"**:
  Get the average process time over all clients (in seconds).
- **"ProcessTime_median()"**:
  Get the median of the process times over all clients (in seconds).

- **"ProcessTime_quantil(p)"**:
  Gets the quantil for the probability p of the process times over all clients (in seconds).
- **"ProcessTime_min()"** or **"ProcessTime_Minimum()"**:
  Get the minimum process time over all clients (in seconds).
- **"ProcessTime_max()"** or **"ProcessTime_Maximum()"**:
  Get the maximum process time over all clients (in seconds).
- **"ProcessTime_var()"** or **"ProcessTime_Varianz()"**:
  Gets the variation of the process times over all clients (based on seconds).
- **"ProcessTime_sd()"** or **"ProcessTime_Standardabweichung()"**:
  Gets the standard deviation of the process times over all clients (based on seconds).
- **"ProcessTime_cv()"**:
  Gets the coefficient of variation of the process times over all clients.
- **"ProcessTime_scv()"**:
  Gets the squared coefficient of variation of the process times over all clients.
- **"ProcessTime_sk()"**:
  Gets the skewness of the process times over all clients.
- **"ProcessTime_kurt()"**:
  Gets the excess kurtosis of the process times over all clients.
- **"ProcessTime_histAll(time)"**:
  Gets the fraction of clients for which the process **time** was time seconds.
- **"ProcessTime_histAll(timeA;timeB)"**:
  Gets the fraction of clients for which the process time was more than **timeA** and at most **timeB** seconds.

12.8.3 Process times for a specific client type

- **"ProcessTime_sum(id)"** or **"ProcessTime_gesamt(id)"** or **"ProcessTime_summe(id)"**:
  Gets the sum of the process times of the clients, whos name appears at the source or the type assignment **id** (in seconds).
- **"ProcessTime_avg(id)"** or **"ProcessTime_average(id)"** or **"ProcessTime_Mittelwert(id)"**:
  Gets the average process time of the clients, whos name appears at the source or the type assignment **id** (in seconds).
- **"ProcessTime_median(id)"**:
  Gets the median of the process times of the clients, whos name appears at the source or the type assignment **id** (in seconds).
- **"ProcessTime_quantil(p;id)"**:
  Gets the quantil for the probability p of the process times of the clients, whos name appears at the source or the type assignment **id** (in seconds).
- **"ProcessTime_min(id)"** or **"ProcessTime_Minimum(id)"**:
  Gets the minimum process time of the clients, whos name appears at the source or the type assignment **id** (in seconds).
- **"ProcessTime_max(id)"** or **"ProcessTime_Maximum(id)"**:
  Gets the maximum process time of the clients, whos name appears at the source or the type assignment **id** (in seconds).

- `"ProcessTime_var(id)"` or `"ProcessTime_Varianz(id)"`:
  Gets the variation of the process times of the clients, whos name appears at the source or the type assignment `id` (based on seconds).
- `"ProcessTime_sd(id)"` or `"ProcessTime_Standardabweichung(id)"`:
  Gets the standard deviation of the process times of the clients, whos name appears at the source or the type assignment `id` (based on seconds).
- `"ProcessTime_cv(id)"`:
  Gets the coefficient of variation of the process times of the clients, whos name appears at the source or the type assignment `id`.
- `"ProcessTime_scv(id)"`:
  Gets the squared coefficient of variation of the process times of the clients, whos name appears at the source or the type assignment `id`.
- `"ProcessTime_sk(id)"`:
  Gets the skewness of the process times of the clients, whos name appears at the source or the type assignment `id`.
- `"ProcessTime_kurt(id)"`:
  Gets the excess kurtosis of the process times of the clients, whos name appears at the source or the type assignment `id`.

12.9 Residence times

12.9.1 Residence times at a station

- `"ResidenceTime_sum(id)"` or `"ResidenceTime_gesamt(id)"` or `"ResidenceTime_summe(id)"`:
  Gets the sum of the residence times at the station `id` (in seconds).
- `"ResidenceTime_avg(id)"` or `"ResidenceTime_average(id)"` or `"ResidenceTime_Mittelwert(id)"`:
  Gets the average residence time of the clients at station `id` (in seconds).
- `"ResidenceTime_median(id)"`:
  Gets the median of the residence times of the clients at station `id` (in seconds).
- `"ResidenceTime_quantil(p;id)"`:
  Gets the quantil for the probability p of the residence times of the clients at station `id` (in seconds).
- `"ResidenceTime_min(id)"` or `"ResidenceTime_Minimum(id)"`:
  Gets the minimum residence time of the clients at station `id` (in seconds).
- `"ResidenceTime_max(id)"` or `"ResidenceTime_Maximum(id)"`:
  Gets the maximum residence time of the clients at station `id` (in seconds).
- `"ResidenceTime_var(id)"` or `"ResidenceTime_Varianz(id)"`:
  Gets the variation of the residence times of the clients at station `id` (based on seconds).
- `"ResidenceTime_sd(id)"` or `"ResidenceTime_Standardabweichung(id)"`:
  Gets the standard deviation of the residence times of the clients at station `id` (based on seconds).
- `"ResidenceTime_cv(id)"`:
  Gets the coefficient of variation of the residence times of the clients at station `id`.
- `"ResidenceTime_scv(id)"`:
  Gets the squared coefficient of variation of the residence times of the clients at station `id`.

- **"ResidenceTime_sk(id)"**:
  Gets the skewness of the residence times of the clients at station **id**.
- **"ResidenceTime_kurt(id)"**:
  Gets the excess kurtosis of the residence times of the clients at station **id**.
- **"ResidenceTime_hist(id;time)"**:
  Gets the fraction of clients for which the residence time at stations **id** was **time** seconds.
- **"ResidenceTime_hist(id;timeA;timeB)"**:
  Gets the fraction of clients for which the residence time at stations **id** was more than **timeA** and at most **timeB** seconds.

12.9.2 Residence times over all client types

- **"ResidenceTime_avg()"** or **"ResidenceTime_average()"** or **"ResidenceTime_Mittelwert()"**:
  Get the average residence time over all clients (in seconds).
- **"ResidenceTime_median()"**:
  Get the median of the residence times over all clients (in seconds).
- **"ResidenceTime_quantil(p)"**:
  Gets the quantil for the probability p of the residence times over all clients (in seconds).
- **"ResidenceTime_min()"** or **"ResidenceTime_Minimum()"**:
  Get the minimum residence time over all clients (in seconds).
- **"ResidenceTime_max()"** or **"ResidenceTime_Maximum()"**:
  Get the maximum residence time over all clients (in seconds).
- **"ResidenceTime_var()"** or **"ResidenceTime_Varianz()"**:
  Gets the variation of the residence times over all clients (based on seconds).
- **"ResidenceTime_sd()"** or **"ResidenceTime_Standardabweichung()"**:
  Gets the standard deviation of the residence times over all clients (based on seconds).
- **"ResidenceTime_cv()"**:
  Gets the coefficient of variation of the residence times over all clients.
- **"ResidenceTime_scv()"**:
  Gets the squared coefficient of variation of the residence times over all clients.
- **"ResidenceTime_sk()"**:
  Gets the skewness of the residence times over all clients.
- **"ResidenceTime_kurt()"**:
  Gets the excess kurtosis of the residence times over all clients.
- **"ResidenceTime_histAll(time)"**:
  Gets the fraction of clients for which the residence time was **time** seconds.
- **"ResidenceTime_histAll(timeA;timeB)"**:
  Gets the fraction of clients for which the residence time was more than **timeA** and at most **timeB** seconds.

12.9.3 Residence times for a specific client type

- `"ResidenceTime_sum(id)"` or `"ResidenceTime_gesamt(id)"` or `"ResidenceTime_summe(id)"`: Gets the sum of the residence times of the clients, whos name appears at the source or the type assignment `id` (in seconds).
- `"ResidenceTime_avg(id)"` or `"ResidenceTime_average(id)"` or `"ResidenceTime_Mittelwert(id)"`:
  Gets the average residence time of the clients, whos name appears at the source or the type assignment `id` (in seconds).
- `"ResidenceTime_median(id)"`:
  Gets the median of the residence times of the clients, whos name appears at the source or the type assignment `id` (in seconds).
- `"ResidenceTime_quantil(p;id)"`:
  Gets the quantil for the probability p of the residence times of the clients, whos name appears at the source or the type assignment `id` (in seconds).
- `"ResidenceTime_min(id)"` or `"ResidenceTime_Minimum(id)"`:
  Gets the minimum residence time of the clients, whos name appears at the source or the type assignment `id` (in seconds).
- `"ResidenceTime_max(id)"` or `"ResidenceTime_Maximum(id)"`:
  Gets the maximum residence time of the clients, whos name appears at the source or the type assignment `id` (in seconds).
- `"ResidenceTime_var(id)"` or `"ResidenceTime_Varianz(id)"`:
  Gets the variation of the residence times of the clients, whos name appears at the source or the type assignment `id` (based on seconds).
- `"ResidenceTime_sd(id)"` or `"ResidenceTime_Standardabweichung(id)"`:
  Gets the standard deviation of the residence times of the clients, whos name appears at the source or the type assignment `id` (based on seconds).
- `"ResidenceTime_cv(id)"`:
  Gets the coefficient of variation of the residence times of the clients, whos name appears at the source or the type assignment `id`.
- `"ResidenceTime_scv(id)"`:
  Gets the squared coefficient of variation of the residence times of the clients, whos name appears at the source or the type assignment `id`.
- `"ResidenceTime_sk(id)"`:
  Gets the skewness of the residence times of the clients, whos name appears at the source or the type assignment `id`.
- `"ResidenceTime_kurt(id)"`:
  Gets the excess kurtosis of the residence times of the clients, whos name appears at the source or the type assignment `id`.

12.10 Utilization of the resources

12.10.1 Utilization of a resource

- `"resource_count(id)"` or `"resource_capacity(id)"` or `"MR(id)"`:
  Gets the number of currently existing operators in the specified resource.

- `"resource_down(id)"`:
  Gets the number of operators that are currently in down time in the specified resource.
- `"resource(id)"` or `"utilization(id)"` or `"NR(id)"`:
  Gets the number of currently busy operators in the specified resource.
- `"resource_avg(id)"` or `"resource_average(id)"` or `"resource_Mittelwert(id)"` or `"utilization_avg(id)"` or `"utilization_average(id)"` or `"utilization_Mittelwert(id)"`:
  Gets the average number of busy operators in the specified resource.
- `"resource_min(id)"` or `"resource_Minimum(id)"` or `"utilization_min(id)"` or `"utilization_Minimum(id)"`:
  Gets the minimum number of busy operators in the specified resource.
- `"resource_max(id)"` or `"resource_Maximum(id)"` or `"utilization_max(id)"` or `"utilization_Maximum(id)"`:
  Gets the maximum number of busy operators in the specified resource.
- `"resource_var(id)"` or `"resource_Varianz(id)"` or `"utilization_var(id)"` or `"utilization_Varianz(id)"`:
  Gets the variation of the number of busy operators in the specified resource.
- `"resource_sd(id)"` or `"resource_Standardabweichung(id)"` or `"utilization_sd(id)"` or `"utilization_Standardabweichung(id)"`:
  Gets the standard deviation of the number of busy operators in the specified resource.
- `"resource_cv(id)"` or `"utilization_cv(id)"`:
  Gets the coefficient of variation of the number of busy operators in the specified resource.
- `"resource_scv(id)"` or `"utilization_scv(id)"`:
  Gets the squared coefficient of variation of the number of busy operators in the specified resource.
- `"resource_sk(id)"` or `"utilization_sk(id)"`:
  Gets the skewness of the number of busy operators in the specified resource.
- `"resource_kurt(id)"` or `"utilization_kurt(id)"`:
  Gets the excess kurtosis of the number of busy operators in the specified resource.
- `"resource_hist(id;state)"` or `"utilization_hist(id;state)"`:
  Gets the fraction of time, at which **`state`** of the operators in the specified resource have been busy.
- `"resource_hist(id;stateA;stateB)"` or `"utilization_hist(id;stateA;stateB)"`:
  Gets the fraction of time, at which more than **`stateA`** and at most **`stateB`** of the operators in the specified resource have been busy.

12.10.2 Utilization of all resource together

- `"resource_count()"` or `"resource_capacity()"` or `"MR()"`:
  Gets the number of currently existing operators in all resources.
- `"resource_down()"`:
  Gets the number of operators that are currently in down time in all resources.
- `"resource()"` or `"utilization()"` or `"NR()"`:
  Gets the number of currently busy operators in all resources.
- `"resource_avg()"` or `"resource_average()"` or `"resource_Mittelwert()"` or `"utilization_avg()"` or `"utilization_average()"` or `"utilization_Mittelwert()"`:
  Gets the average number of busy operators in all resources.

- "resource_min()" or "resource_Minimum()" or "utilization_min()" or "utilization_Minimum()":
  Gets the minimum number of busy operators in all resources.
- "resource_max()" or "resource_Maximum()" or "utilization_max()" or "utilization_Maximum()":
  Gets the maximum number of busy operators in all resources.

12.11 Utilization of the transporters

12.11.1 Utilization of a transporter group

- "transporter_count(id)":
  Gets the number of transporter in the specified transporter group.
- "transporter_capacity(id)":
  Gets the number of clients a transporter in the specified transporter group can carry.
- "transporter_down(id)":
  Gets the number of transporters that are currently in down time in the specified transporter group.
- "transporter(id)" or "transporter_utilization(id)":
  Gets the number of currently busy transporters in the specified transporter group.
- "transporter_avg(id)" or "transporter_average(id)" or "transporter_Mittelwert(id)" or "transporter_utilization_avg(id)" or "transporter_utilization_average(id)" or "transporter_utilization_Mittelwert(id)":
  Gets the average number of busy transporters in the specified transporter group.
- "transporter_min(id)" or "transporter_Minimum(id)" or "transporter_utilization_min(id)" or "transporter_utilization_Minimum(id)":
  Gets the minimum number of busy transporters in the specified transporter group.
- "transporter_max(id)" or "transporter_Maximum(id)" or "transporter_utilization_max(id)" or "transporter_utilization_Maximum(id)":
  Gets the maximum number of busy transporters in the specified transporter group.
- "transporter_var(id)" or "transporter_Varianz(id)" or "transporter_utilization_var(id)" or "transporter_utilization_Varianz(id)":
  Gets the variation of the number of busy transporters in the specified transporter group.
- "transporter_sd(id)" or "transporter_Standardabweichung(id)" or "transporter_utilization_sd(id)" or "transporter_utilization_Standardabweichung(id)":
  Gets the standard deviation of the number of busy transporters in the specified transporter group.
- "transporter_cv(id)" or "transporter_utilization_cv(id)":
  Gets the coefficient of variation of the number of busy transporters in the specified transporter group.
- "transporter_scv(id)" or "transporter_utilization_scv(id)":
  Gets the squared coefficient of variation of the number of busy transporters in the specified transporter group.
- "transporter_sk(id)" or "transporter_utilization_sk(id)":
  Gets the skewness of the number of busy transporters in the specified transporter group.
- "transporter_kurt(id)" or "transporter_utilization_kurt(id)":
  Gets the excess kurtosis of the number of busy transporters in the specified transporter group.

- **"transporter_hist(id;state)"** or **"transporter_utilization_hist(id;state)"**:
  Gets the fraction of time, at which **state** of the transporters in the specified transporter group have been busy.
- **"transporter_hist(id;stateA;stateB)"** or **"transporter_utilization_hist(id;stateA;stateB)"**:
  Gets the fraction of time, at which more than **stateA** and at most **stateB** of the transporters in the specified transporter group have been busy.

12.11.2 Utilization of all transporters together

- **"transporter_count()"**:
  Gets the number of currently existing transporters in all transporter groups.
- **"transporter_down()"**:
  Gets the number of transporters that are currently in down time in all transporter groups.
- **"transporter()"** or **"transporter_utilization()"**:
  Gets the number of currently busy transporters in all transporter groups.
- **"transporter_avg()"** or **"transporter_average()"** or **"transporter_Mittelwert()"** or **"transporter_utilization_avg()"** or **"transporter_utilization_average()"** or **"transporter_utilization_Mittelwert()"**:
  Gets the average number of busy transporters in all transporter groups.
- **"transporter_min()"** or **"transporter_Minimum()"** or **"transporter_utilization_min()"** or **"transporter_utilization_Minimum()"**:
  Gets the minimum number of busy transporters in all transporter groups.
- **"transporter_max()"** or **"transporter_Maximum()"** or **"transporter_utilization_max()"** or **"transporter_utilization_Maximum()"**:
  Gets the maximum number of busy transporters in all transporter groups.

12.12 Accessing the Statistics stations records

- **"Statistics(id;nr)"**:
  Gets the current value of record **nr** (1 based) at Statistics station **id**.
- **"Statistics_avg(id;nr)"** or **"Statistics_average(id;nr)"**:
  Gets the average value of record **nr** (1 based) at Statistics station **id**.
- **"Statistics_median(id;nr)"**:
  Gets the median of the value of record **nr** (1 based) at Statistics station **id**. (This command is not available for continuous-time user-defined statistics entries).
- **"Statistics_quantil(id;nr;p)"**:
  Gets the quantil for the probability p of the value of record **nr** (1 based) at Statistics station **id**. (This command is not available for continuous-time user-defined statistics entries).
- or **"Statistics_min(id;nr)"** or **"Statistics_Minimum(id;nr)"**:
  Gets the minimum value of record **nr** (1 based) at Statistics station **id**.
- **"Statistics_max(id;nr)"** or **"Statistics_Maximum(id;nr)"**:
  Gets the maximum value of record **nr** (1 based) at Statistics station **id**.
- **"Statistics_var(id;nr)"**:
  Gets the variance of record **nr** (1 based) at Statistics station **id**.

- **"Statistics_std(id;nr)"**:
  Gets the standard deviation of record **nr** (1 based) at Statistics station **id**.
- **"Statistics_cv(id;nr)"**:
  Gets the coefficient of variation of record **nr** (1 based) at Statistics station **id**.
- **"Statistics_scv(id;nr)"**:
  Gets the squared coefficient of variation of record **nr** (1 based) at Statistics station **id**.
- **"Statistics_sk(id;nr)"**:
  Gets the skewness of record **nr** (1 based) at Statistics station **id**.
- **"Statistics_kurt(id;nr)"**:
  Gets the excess kurtosis of record **nr** (1 based) at Statistics station **id**.
- **"Statistics_hist(id;nr;state)"**:
  Get the part in which the value of record **nr** (1 based) at Statistics station **id** was **state**. (This command is not available for continuous-time user-defined statistics entries).
- **"Statistics_hist(id;nr;stateA;stateB)"**:
  Get the part in which the value of record **nr** (1 based) at Statistics station **id** was bigger than **stateA** and smaller or equal **stateB**. (This command is not available for continuous-time user-defined statistics entries).

12.13 Accessing analog values

- **"AnalogValue(id)"**:
  Gets the current value of the "Analog value" element or the "Tank" element **id**.
- **"AnalogRate(id)"**:
  Gets the current change rate of the value of the "Analog value" element **id**.
- **"ValveMaximumFlow(id;nr)"**:
  Gets the current maximum flow at valve **nr** (1 based) at "Tank" element **id**.

12.14 Accessing the client object specific data fields

- **"WarmUpKunde()"** or **"WarmUpClient()"** or **"isWarmUpClient()"**:
  Gets 0 or 1 depending on whether the client was generated during the warm-up phase (1) or not (0).
- **"KundeInStatistik()"** or **"ClientInStatistics()"** or **"isClientInStatistics()"**:
  Gets 0 or 1 depending on whether the client is to be recorded in the statistics (1) or nor (0). Additionally the client has to be generated after the warm-up phase to be actually recorded.
- **"KundeNummer()"** or **"ClientNumber()"**:
  Returns the 1-based consecutive number of the current client. If using multiple simulation threads this number is thread-local.
- **"ClientData(index)"**:
  Gets the data field at position **index** from the current client object.
  The "Variable" elements can be used to write to these fields.
- **"Alternative()"**:
  Indicates which operator alternative has been chosen at the last process station the client has passed to serve the client. If the client has not passed any process station yet, the function will return 0. Otherwise a value of 1 or larger will be returned.

- `"PreviousStation()"`:
  Gets the ID of the station where the client was before he entered the current station.
- `"CurrentWaitingTime()"`:
  Gets the waiting time of the current client at the current station.
- `"ClientBatchSize()"`:
  If the client object is a temporary batch, the number of clients contained in the batch is returned. Otherwise, 0 is returned.

12.15 Accessing the costs

- `"costs_waiting_sum(id)"`:
  Gets the waiting time costs of the clients, whos name appears at the source or the type assignment `id`.
- `"costs_waiting_avg(id)"` or `"costs_waiting_average(id)"` or `"Kosten_WaitingTime_Mittelwert(id)"`:
  Gets the average waiting time costs of the clients, whos name appears at the source or the type assignment `id`.
- `"costs_waiting_sum()"` or `"Kosten_WaitingTime_Summe()"`:
  Gets the sum of the waiting time costs of all clients.
- `"costs_waiting_avg()"` or `"costs_waiting_average()"`:
  Gets the average waiting time costs of all clients.
- `"costs_waiting()"`:
  Gets the waiting time costs of the current client.
- `"costs_transfer_sum(id)"`:
  Gets the transfer time costs of the clients, whos name appears at the source or the type assignment `id`.
- `"costs_transfer_avg(id)"` or `"costs_transfer_average(id)"`:
  Gets the average transfer time costs of the clients, whos name appears at the source or the type assignment `id`.
- `"costs_transfer_sum()"`:
  Gets the sum of the transfer time costs of all clients.
- `"costs_transfer_avg()"` or `"costs_transfer_average()"`:
  Gets the average transfer time costs of all clients.
- `"costs_transfer()"`:
  Gets the transfer time costs of the current client.
- `"costs_process_sum(id)"`:
  Gets the process time costs of the clients, whos name appears at the source or the type assignment `id`.
- `"costs_process_avg(id)"` or `"costs_process_average(id)"`:
  Gets the average process time costs of the clients, whos name appears at the source or the type assignment `id`.
- `"costs_process_sum()"`:
  Gets the sum of the process time costs of all clients.
- `"costs_process_avg()"` or `"costs_process_average()"`:
  Gets the average process time costs of all clients.

- `"costs_process()"`:
  Gets the process time costs of the current client.
- `"costs(id)"`:
  Provides the station costs, which so far have occurred at station `id`.
- `"costs()"`:
  Provides the station costs, which so far have occurred in total at all stations.
- `"costs_resource(id)"`:
  Returns the costs incurred by the specified resource so far.
- `"costs_resource()"`:
  Returns the costs incurred by all resources so far.

Chapter 13
Comparison

- `"a == b"`:
  Gets true, if a has the same value as b.
- `"a != b"` or `"a <> b"`:
  Gets true, if a and b have different values.
- `"a < b"`:
  Gets true, if a is less than b.
- `"a <= b"` or `"a =< b"`:
  Gets true, if a is less or equal to b.
- `"a > b"`:
  Gets true, if a is larger than b.
- `"a >= b"` or `"a => b"`:
  Gets true, if a is larger or equal to b.
- `"A || B"`:
  Gets true, if A or B (or both) are true.
- `"A && B"`:
  Gets true, if A and B are both true.
- `"!(A)"`:
  Gets true, if A is not true.

The lowercase characters a and b are placeholders for calculation expressions like `"WIP()"`. The capital letters A and B stand for comparisons like `"WIP()<5"`.

13.1 Comparison function

Additionally normal calculation commands an `"If"` function is available. This functions expects an odd number of parameters:

`"If(condition1;value1;condition2;value2;...;valueElse)"`

If `condition1`> 0 the function returns `value1`. Otherwise `condition2`> 0 is tested and if its fulfilled `value2` is returned and so on. If non of the conditions is fulfilled the function returns `valueElse`.

Part II

Javascript commands reference

Scripts can be used at different points in the simulator. The script language is **Javascript** or **Java**.

In this section the additional **Javascript** commands which are available when using Javascript to access the simulation or statistics data and to output filtered data are presented.

Chapter 14
Statistics object

The "Statistics" object offers read access to the xml elements which are the base of the statistics data. The "Statistics" object is only available after the simulation while filtering the results while and when running a parameter series script. The following methods are in this object available:

14.1 Definition of the output format

- "Statistics.setFormat("Format")":
  This command allows to setup the format that is used in "Statistics.xml" for outputing numbers as strings. You can specify whether to use a floating point notation or percent notation or interpreting the value as a time. As default floating point notation is used.
  - "System": Using floating point notation for numbers and percent values.
  -
  - "Fraction": Using floating point notation for numbers (0.375 for example).
  - "Percent": Using percent notation for numbers (35.7% for example).
  - "Time": Interpreting numbers as times (00:03:25,87 for example).
  - "Number": Interpreting time values as normal numbers (format defined by Percent or Fraction).
- "Statistics.setSeparator("Format")":
  This command allows to select the separator to be used when printing out distributions of measured values.
  - "Semicolon": Semicolons as separators
  - "Line": Line break as separators
  - "Tabs": Tabulators as separators

14.2 Accessing statistics xml data

- "Statistics.xml("Path")":
  Loads the xml field which is specified by the parameter and returns the data in the format defined by "Statistics.setFormat" and "Statistics.setSeparator" as a string.

Example:
```
var name=Statistics.xml("Model->ModelName")
```

- "Statistics.xmlNumber("Path")":
  Loads the xml field which is specified by the parameter and returns the value as a number. If the field cannot be interpreted as a number, a string containing an error message will be returned.
- "Statistics.xmlArray("Path")":
  Loads the xml field which is specified by the parameter, interprets it as a distribution and returns the values as an array of numbers. If the field cannot be interpreted as a distribution, a string containing an error message will be returned.

  Example:
  ```
  Statistics.xmlArray("StatisticsProcessTimesClients->ClientType[Type=\"ClientsA\"]->
  [Distribution]")
  ```
- "Statistics.xmlSum("Path")":
  Loads the xml field which is specified by the parameter, interprets it as a distribution and returns the sum of all values as a number. If the field cannot be interpreted as a distribution, a string containing an error message will be returned.

  Example:
  ```
  Statistics.xmlSum("StatisticsProcessTimesClients->ClientType[Type=\"ClientsA\"]->
  [Distribution]")
  ```
- "Statistics.xmlMean("Path")":
  Loads the xml field which is specified by the parameter, interprets it as a distribution and returns the mean of values as a number. If the field cannot be interpreted as a distribution, a string containing an error message will be returned.

  Example:
  ```
  Statistics.xmlMean("StatisticsProcessTimesClients->ClientType[Type=\"ClientsA\"]->
  [Distribution]")
  ```
- "Statistics.xmlSD("Path")":
  Loads the xml field which is specified by the parameter, interprets it as a distribution and returns the standard deviation of values as a number. If the field cannot be interpreted as a distribution, a string containing an error message will be returned.

  Example:
  ```
  Statistics.xmlSD("StatisticsProcessTimesClients->ClientType[Type=\"ClientsA\"]->
  [Distribution]")
  ```
- "Statistics.xmlCV("Path")":
  Loads the xml field which is specified by the parameter, interprets it as a distribution and returns the coefficient of variation of values as a number. If the field cannot be interpreted as a distribution, a string containing an error message will be returned.

  Example:
  ```
  Statistics.xmlCV("StatisticsProcessTimesClients->ClientType[Type=\"ClientsA\"]->
  [Distribution]")
  ```
- "Statistics.translate("de")":
  Translates the statistics data to English ("en") or German ("de") so the preferred xml tag names can be used independent of the language setting under which the statistics file was generated.

14.3 Saving the statistics data to files

- `"Statistics.save("FileName")"`:
  Saves the entry statistics data under the next available file name in the given folder.
  This function is only available in the Run script panel.
- `"Statistics.saveNext("FolderName")"`:
  Saves the entry statistics data under the next available file name in the given folder.
  This function is only available in the Run script panel.
- `"Statistics.filter("FileName")"`:
  Applies the selected script on the statistics data and returns the results.
  This function is only available in the Run script panel.
- `"Statistics.cancel()"`:
  Sets the cancel status. (When output is canceled to further file output will be performed.)

14.4 Accessing station data

- `"Statistics.getStationID("StationName")"`:
  Gets the ID of a station based on its name. If there is no station with a matching name, the function will return -1.

Chapter 15
System object

The "System" object allows to access some general program functions. The "System" object is only available after the simulation while filtering the results or when running parameter series scripts. The following methods are available in this object:

- "System.calc("Expression")":
  Calculates the expression passed as a string by means of the term evaluation function, which is also used in various other places in the program (see part I), and returns the result as a number. If the expression can not be calculated, an error message is returned as a string. The term evaluation function allows access to all known probability distributions, the Erlang C calculator, etc.
- "System.time()":
  Returns the current system time as a milliseconds value. This functions can be used to measure the runtime of a script.
- "System.getInput("http://Adresse",-1)":
  Loads a numerical value via the specified address and returns it. If no value could be loaded, the error value specified in the second parameter is returned.
- "System.execute("program.exe")":
  Executes an external command and returns immediately. Returns true, if the program could be started. Executing external programs by scripts is disabled by default. If can be activated in the program settings dialog.
- "System.executeAndReturnOutput("program.exe")":
  Executes an external command and returns the output. Executing external programs by scripts is disabled by default. If can be activated in the program settings dialog.
- "System.executeAndWait("program.exe")":
  Executes an external command, waits for completion and returns the return code of the program. In case of an error -1 will be returned. Executing external programs by scripts is disabled by default. If can be activated in the program settings dialog.

Chapter 16

`Simulation` object

The `"Simulation"` object allows to access the model data while simulation is running. It is not available for filtering the results after simulation has terminated. The following methods are available in this object:

16.1 Base functions

- `"Simulation.time()"`:
  Gets the current time in the simulation as a seconds numerical value.
- `"Simulation.calc("Expression")"`:
  Calculates the expression passed as a string by means of the term evaluation function, which is also used in various other places in the program (see part I), and returns the result as a number. If the expression can not be calculated, an error message is returned as a string. The term evaluation function allows access to all known probability distributions, the Erlang C calculator, etc.
- `"Simulation.getInput("http://Adresse",-1)"`:
  Loads a numerical value via the specified address and returns it. If no value could be loaded, the error value specified in the second parameter is returned.
- `"Simulation.execute("program.exe")"`:
  Executes an external command and returns immediately. Returns true, if the program could be started. Executing external programs by scripts is disabled by default. If can be activated in the program settings dialog.
- `"Simulation.executeAndReturnOutput("program.exe")"`:
  Executes an external command and returns the output. Executing external programs by scripts is disabled by default. If can be activated in the program settings dialog.
- `"Simulation.executeAndWait("program.exe")"`:
  Executes an external command, waits for completion and returns the return code of the program. In case of an error -1 will be returned. Executing external programs by scripts is disabled by default. If can be activated in the program settings dialog.
- `"Simulation.isWarmUp()"`:
  Gets true of false depending if the simulation is in the warm-up phase.
- `"Simulation.getMapLocal()"`:
  Returns a station-local mapping into which values can be written and from which values can be read. The values stored here are retained beyond the execution of the current script.

- "Simulation.getMapGlobal()":
  Returns a model wide mapping into which values can be written and from which values can be read. The values stored here are retained beyond the execution of the current script.
- "Simulation.pauseAnimation()":
  Switches the animation to single step mode. If the animation is already executed in single step mode or if the model is executed as a simulation, this command has no effect.
- "Simulation.terminateSimulation(message)":
  Beendet die Simulation. Wird als Nachricht **null** übergeben, so wird die Simulation normal beendet. Im Falle einer Nachricht wird die Simulation mit der entsprechenden Fehlermeldung abgebrochen.

16.2 Accessing client-specific data

- "Simulation.clientTypeName()":
  Gets the name of the type of the client who has triggered the processing of the script.
  (If the event was triggered by a client.)
- "Simulation.isWarmUpClient()":
  Gets true of false depending if the current client was generated during the warm-up phase and therefore will not be recorded in the statistics.
  (If the event was triggered by a client.)
- "Simulation.isClientInStatistics()":
  Gets true of false depending if the current client is to be recorded in the statistics. This value is independent of the warm-up phase. A client will only be recorded if he was generated after the warm-up phase and this value is true.
  (If the event was triggered by a client.)
- "Simulation.setClientInStatistics(inStatistics)":
  Sets if a client is to be recorded in the statistics. This value is independent of the warm-up phase. A client will only be recorded if he was generated after the warm-up phase and this value is not set to false.
  (If the event was triggered by a client.)
- "Simulation.clientNumber()":
  Get the 1-based consecutive number of the current client. When using multiple simulation threads this number is thread local.
  (If the event was triggered by a client.)
- "Simulation.clientWaitingSeconds()":
  Gets the current waiting time of the client who has triggered the processing of the script as a seconds numerical value.
  (If the event was triggered by a client.)
- "Simulation.clientWaitingTime()":
  Gets the current waiting time of the client who has triggered the processing of the script as a formated time value as a string.
  (If the event was triggered by a client.)
- "Simulation.clientWaitingSecondsSet(seconds)":
  Sets the current waiting time of the client who has triggered the processing of the script.
  (If the event was triggered by a client.)
- "Simulation.clientTransferSeconds()":
  Gets the current transfer time of the client who has triggered the processing of the script as a seconds

numerical value.
(If the event was triggered by a client.)

- **"Simulation.clientTransferTime()"**:
  Gets the current transfer time of the client who has triggered the processing of the script as a formated time value as a string.
  (If the event was triggered by a client.)
- **"Simulation.clientTransferSecondsSet(seconds)"**:
  Sets the current transder time of the client who has triggered the processing of the script.
  (If the event was triggered by a client.)
- **"Simulation.clientProcessSeconds()"**:
  Gets the current processing time of the client who has triggered the processing of the script as a seconds numerical value.
  (If the event was triggered by a client.)
- **"Simulation.clientProcessTime()"**:
  Gets the current processing time of the client who has triggered the processing of the script as a formated time value as a string.
  (If the event was triggered by a client.)
- **"Simulation.clientProcessSecondsSet(seconds)"**:
  Sets the current processing time of the client who has triggered the processing of the script.
  (If the event was triggered by a client.)
- **"Simulation.clientResidenceSeconds()"**:
  Gets the current residence time of the client who has triggered the processing of the script as a seconds numerical value.
  (If the event was triggered by a client.)
- **"Simulation.clientResidenceTime()"**:
  Gets the current residence time of the client who has triggered the processing of the script as a formated time value as a string.
  (If the event was triggered by a client.)
- **"Simulation.clientResidenceSecondsSet(seconds)"**:
  Sets the current residence time of the client who has triggered the processing of the script.
  (If the event was triggered by a client.)
- **"Simulation.getClientValue(index)"**:
  Gets for the current client the numerical value which is stored by the index **index**.
  (If the event was triggered by a client.)
- **"Simulation.setClientValue(index,value)"**:
  Sets for the current client the numerical **value** for the index **index**.
  (If the event was triggered by a client.)
- **"Simulation.getClientText("key")"**:
  Gets for the current client the string which is stored by the key **key**.
  (If the event was triggered by a client.)
- **"Simulation.setClientText("key","value")"**:
  Sets for the current client string **value** for the key **key**.
  (If the event was triggered by a client.)
- **"Simulation.getAllClientValues()"**: Return all numerical values stored for the current client.
- **"Simulation.getAllTexts()"**: Return all text values stored for the current client.

16.3 Temporary batches

If the current client is a temporary batch, the properties of the inner clients it contains can be accessed in read-only mode:

- **"Simulation.batchSize()"**:
  Returns the number of clients that are in the temporary batch. If the current client is not a temporary batch, the function returns 0.
- **"Simulation.getBatchTypeName(batchIndex)"**:
  Returns the name of one of the clients in the current batch. The passed index is 0-based and must be in the range from 0 to **batchSize()-1**.
- **"Simulation.getBatchWaitingSeconds(batchIndex)"**:
  Returns the previous waiting time of one of the clients in the current batch in seconds as a numerical value. The passed index is 0-based and must be in the range from 0 to **batchSize()-1**.
- mdSimulation.getBatchWaitingTime(batchIndex):
  Returns the previous waiting time of one of the clients in the current batch in formatted form as a string. The passed index is 0-based and must be in the range from 0 to **batchSize()-1**.

  tem mdSimulation.getBatchTransferSeconds(batchIndex):
  Returns the previous transfer time of one of the clients in the current batch in seconds as a numerical value. The passed index is 0-based and must be in the range from 0 to **batchSize()-1**.
- **"Simulation.getBatchTransferTime(batchIndex)"**:
  Returns the previous transfer time of one of the clients in the current batch in formatted form as a string. The passed index is 0-based and must be in the range from 0 to **batchSize()-1**.
- **"Simulation.getBatchProcessSeconds(batchIndex)"**:
  Returns the previous processing time of one of the clients in the current batch in seconds as a numerical value. The passed index is 0-based and must be in the range from 0 to **batchSize()-1**.
- **"Simulation.getBatchProcessTime(batchIndex)"**:
  Returns the previous processing time of one of the clients in the current batch in formatted form as a string. The passed index is 0-based and must be in the range from 0 to **batchSize()-1**.
- **"Simulation.getBatchResidenceSeconds(batchIndex)"**:
  Returns the previous residence time of one of the clients in the current batch in seconds as a numerical value. The passed index is 0-based and must be in the range from 0 to **batchSize()-1**.
- **"Simulation.getBatchResidenceTime(batchIndex)"**:
  Returns the previous residence time of one of the clients in the current batch in formatted form as a string. The passed index is 0-based and must be in the range from 0 to **batchSize()-1**.
- **"Simulation.getBatchValue(batchIndex, index)"**:
  Returns a stored numerical value for one of the clients in the current batch. The passed batch index is 0-based and must be in the range from 0 to **batchSize()-1**.
- **"Simulation.getBatchText(batchIndex, key)"**:
  Returns a stored text value for one of the clients in the current batch. The passed batch index is 0-based and must be in the range from 0 to **batchSize()-1**.

16.4 Accessing parameters of the simulation model

- **"Simulation.set("Name",Value)"**:
  Sets the simulation variable which is specified as the first parameter to the value specified as the second

parameter. **Value** can be a number or a string. The case of a number the value will be assigned directly. Strings will be interpreted like **Simulation.calc** does and the result will be assigned to the variable. **Name** can either be the name of an already defined simulation variable or of a client data field in the form **ClientData(index)** with $index \geq 0$.

- **"Simulation.setValue(id,Value)"**:
  Sets the value at the "Analog value" or "Tank" element with the specified **id**.
- **"Simulation.setRate(id,Value)"**:
  Sets the change rate (per second) at the "Analog value" element with the specified **id**.
- **"Simulation.setValveMaxFlow(id,ValveNr,Vale)"**:
  Sets the maximum flow (per second) at the specified valve (1 based) of the "Tank" element with the specified **id**. The maximum flow has to be a non-negative number.
- **"Simulation.getWIP(id)"**:
  Gets the current number of clients at the station with the specified id.
- **"Simulation.getNQ(id)"**:
  Gets the current number of clients in the queue at the station with the specified id.
- **"Simulation.getWIP()"**:
  Gets the current number of clients in the system.
- **"Simulation.getNQ()"**:
  Gets the current number of waiting clients in the system.

16.5 Accessing the current input value

- **"Simulation.getInput()"**:
  If the Javascript code is being executed from a Input (Script) element, this function returns the current input value. Otherwise it will just return 0.

16.6 Number of operators in a resource

- **"Simulation.getAllResourceCount()"**:
  Returns the current number of operators in all resources together.
- **"Simulation.getResourceCount(id)"**:
  Returns the current number of operators in the resource with the specified id.
- **"Simulation.setResourceCount(id,count)"**:
  Sets the number of operators in the resource with the specified id. To be able to set the number of operators in a resource at runtime, the initial number of operators in the resource has to be a fixed number (not infinite many and not by a time table). Additionally no down times are allowed for this resource. The function returns **true** if the number of operators has successfully been changed. If the new number of operators is less than the previous number, the new number may is not instantly visible in the simulation system because removed but working operators will finish their current tasks before they are actually removed.
- **"Simulation.getAllResourceDown()"**:
  Returns the current number of operators in down time in all resources together.
- **"Simulation.getResourceDown(id)"**:
  Returns the current number of operators in down time in the resource with the specified id.

16.7 Fire signal

- `"Simulation.signal(name)"`:
  Fires the signal with the given name.

16.8 Trigger script execution

- `"Simulation.triggerScriptExecution(stationId,time)"`:
  Triggers the execution of the script at a script or a script hold station at a given time.

16.9 Output message in logging

- `"Simulation.log(message)"`:
  Outputs the passed message to the logging system (if logging is enabled).

16.10 Release clients at delay stations

If a list of clients at a delay station is recorded, this list can be queried using the following function and individual clients can be selectively released before their specified delay time has expired.

- `"getDelayStationData(id)"`:
  Returns an object which offers the methods described in **Accessing client-specific data** for accessing the list of clients at the delay station `id`. If the id is invalid, `null` will be returned.

16.11 Clients in the queue of a process station

- `"getProcessStationQueueData(id)"`: Returns an object which offers the methods described in **Accessing client-specific data** for accessing the list of clients waiting at the process station `id`. If the id is invalid, `null` will be returned. Only the waiting clients can be accessed, not the clients which are already in service process. Also clients cannot be released via the `release` method here.

Chapter 17
`Clients` object

The **`"Clients"`** object is only available within a hold by script condition element and allows to access the waiting clients and to release them.

- **`"Clients.count()"`**:
  Returns the current number of waiting clients. For the other methods a single client can be accessed via the index parameter (valued from 0 to **`count()`**-1).
- **`"Clients.clientTypeName(index)"`**:
  Gets the name of the type of the client.
- **`"Clients.clientWaitingSeconds(index)"`**:
  Gets the current waiting time of the client as a seconds numerical value.
- **`"Clients.clientWaitingTime(index)"`**:
  Gets the current waiting time of the client as a formated time value as a string.
- **`"Clients.clientTransferSeconds(index)"`**:
  Gets the current transfer time of the client as a seconds numerical value.
- **`"Clients.clientTransferTime(index)"`**:
  Gets the current transfer time of the client as a formated time value as a string.
- **`"Clients.clientProcessSeconds(index)"`**:
  Gets the current processing time of the client as a seconds numerical value.
- **`"Clients.clientProcessTime(index)"`**:
  Gets the current processing time of the client as a formated time value as a string.
- **`"Clients.clientResidenceSeconds(index)"`**:
  Gets the current residence time of the client as a seconds numerical value.
- **`"Clients.clientResidenceTime(index)"`**:
  Gets the current residence time of the client as a formated time value as a string.
- **`"Clients.clientData(index,data)"`**:
  Returns the data element which index is specified via the second parameter of the selected client.
- **`"Clients.clientData(index,data,value)"`**:
  Set the numerical value specified by the third parameter for the data element which index is specified via the second parameter of the selected client.
- **`"Clients.clientTextData(index,key)"`**:
  Returns the data element which key is specified via the second parameter of the selected client.

- "Clients.clientTextData(index,key,value)":
  Set the text value specified by the third parameter for the key which is specified via the second parameter of the selected client.
- "Clients.release(index)":
  Causes the forwarding of the specified client.

Chapter 18
`Output` object

The `"Output"` object provides functions for output of filtered results:

- `"Output.setFormat("Format")"`:
  This command allows to setup the format that is used in `"Output.print"` and `"Output.println"` for outputing numbers as strings. You can specify whether to use a floating point notation or percent notation or interpreting the value as a time. As default floating point notation is used.

  - `"Fraction"`:
    Using floating point notation for numbers (0.375 for example).
  - `"Percent"`:
    Using percent notation for numbers (35.7% for example).
  - `"Number"`:
    Interpreting numbers as normal number values (decimal or percent).
  - `"Time"`:
    Interpreting numbers as time values.

- `"Output.setSeparator("Format")"`:
  This command allows to select the separator to be used when printing out arrays.

  - `"Semicolon"`:
    Semicolons as separators.
  - `"Line"`:
    Line break as separators.
  - `"Tabs"`:
    Tabulators as separators.

- `"Output.setDigits(digits)"`:
  This command allows to define the number of digits to be displayed when printing a number in local notation. A negative value means that all available digits are being printed. (If the system notation is used, always all available digits are being printed.)
- `"Output.print("Expression")"`:
  Outputs the passed expression. Strings will be written directly. Numbers are formated according to the format defined via `Output.setFormat`.

- `"Output.println("Expression")"`:
  Outputs the passed expression and adds a line break after the expression. Strings will be written directly. Numbers are formated according to the format defined via **`Output.setFormat`**.
- `"Output.newLine()"`:
  Outputs a line break. This functions is equivalent to calling
  `"Output.println("")"`.
- `"Output.tab()"`:
  Outputs a tabulator.
- `"Output.cancel()"`:
  Sets the cancel status. (When output is canceled to further file output will be performed.)
- `"Output.printlnDDE("Workbook","Table","Cell","Expression")"`:
  This command is only available if DDE is available, i.e. under Windows. It outputs the passed expression via DDE in the specified table in Excel. Numbers are formated according to the format defined via **`Output.setFormat`**.

Chapter 19

`FileOutput` object

The `"FileOutput"` object offers all function the `"Output"` has but is only available when running a parameter series script. In opposite to the `"Output"` object the output of the `"FileOutput"` object is not written to the default output but is appended to a file which has to be specified by `"FileOutput.setFile("Filename")"` before.

Chapter 20
Model object

Th **"Model"** object is only available during parameter series script execution and offers functions for accessing the model properties and for starting simulations.

- **"Model.reset()"**:
  Resets the model to the initial state.
- **"Model.run()"**:
  Simulates the current model. The results can be accessed by the **"Statistics"** object after the simulation.
- **"Model.setDistributionParameter("Path",Index,Value)"**:
  Sets the distribution parameter **Index** (from 1 to 4) of the distribution referred to by **Path**.
- **"Model.setMean("Path",Value)"**:
  Sets the mean of the distribution referred to by **Path** to the specified value.
- **"Model.setSD("Path",Value)"**:
  Sets the standard deviation of the distribution referred to by **Path** to the specified value.
- **"Model.setString("Path","Text)""**:
  Writes the string **Text** to the location referred to by **Path**.
- **"Model.setValue("Path",Value)"**:
  Writes the number **Value** to the location referred to by **Path**.
- **"Model.xml("Path")"**:
  Loads the xml field which is specified by the parameter and returns the data as String. This function is the equivalent of **"Statistics.xml("Path")"** for models.
- **"Model.getResourceCount("ResourceName")"**:
  Gets the number of operations in the resource with name **ResourceName**. If the resource does not exist or is not defined by a fixed number of operators the function will return -1.
- **"Model.setResourceCount("ResourceName",Count)"**:
  Sets the number of operations in the resource with name **ResourceName**.
- **"Model.getGlobalVariableInitialValue("VariableName")"**:
  Gets the expression by which the initial value of the global variable with name **VariableName** is calculated. If the is no global variable with this name the function will return an empty string.
- **"Model.setGlobalVariableInitialValue("VariableName","Expression")"**:
  Sets the expression by which the initial value of the global variable with name **VariableName** is calculated.

- "Model.getGlobalMapInitialValue("VariableName")":
  Gets the initial value of the entry **VariablenName** of the global map. If there is no entry with this name, **null** will be returned.
- "Model.setGlobalMapInitialValue("VariableName","Expression")":
  Sets the initial value (of type **Integer**, **Long**, **Double** oder **String**) of the key **VariablenName** in the global map.
- "Model.cancel()":
  Sets the cancel status. (When processing is canceled to further simulations will be performed.)

20.1 Accessing station data

- "Model.getStationID("StationName")":
  Gets the ID of a station based on its name. If there is no station with a matching name, the function will return -1.

20.2 Retrieve the associated statistics file

- "Statistics.getStatisticsFile()":
  Returns the full path and file name of the statistics file from which the data was loaded. If the statistic data was not loaded from a file, an empty string is returned.
- "Statistics.getStatisticsFileName()":
  Returns the file name of the statistics file from which the data was loaded. If the statistic data was not loaded from a file, an empty string is returned.

Chapter 21
XML selection commands

By the parameters of the functions of the **"Statistics"** object the content of the value or of an attribute of an XML element can be read. The selection of an XML element is done multistaged step by step divided by "->" characters. Between the "->" characters the names of the individual XML nodes are noted. In addition in square brackets names and values of attributes can be specified to filter by whom.

Examples:

- **"Statistics.xml("Model->ModellName")"**:
  Shows the content of the element **ModelName**, which is a child element of **Model**.
- **"Statistics.xml("StatisticsInterArrivalTimesClients->
  Station[Type=\"Source id=1\"]->[Mean]")"**:
  Selects the **Station** sub element of the **StatisticsInterArrivalTimesClients** element, for which the **Type** attribute is set to **Source id=1**. And returns the value of the **Mean** attribute.

Part III

Java commands reference

Scripts can be used at different points in the simulator. The script language is **Javascript** or **Java**.

In this section the additional **Java** commands which are available when using Java to access the simulation or statistics data and to output filtered data are presented.

The **Java** code has to be embedded in a

```
void function(SimulationInterface sim) {

}
```

method. In addition to the standard language commands you can access the simulation or statistics data depending on the context in which the script is executed via the `SimulationInterface` interface which is given as a parameter. The `SimulationInterface` has some methods which allow to get sub-interfaces which offer these data:

Chapter 22

StatisticsInterface accessible via sim.getStatistics()

The `sim.getStatistics()` methods returns a StatisticsInterface interface which offers read access to the xml elements which are the base of the statistics data. The `StatisticsInterface` interface is only available after the simulation while filtering the results while and when running a parameter series scripts. The following methods are in this object available:

22.1 Definition of the output format

- **`"void setFormat(final String format)"`**:
  This command allows to setup the format that is used in **`"Statistics.xml"`** for outputing numbers as strings. You can specify whether to use a floating point notation or percent notation or interpreting the value as a time. As default floating point notation is used.
  - **`"System"`**: Using floating point notation for numbers and percent values.
  -
  - **`"Fraction"`**: Using floating point notation for numbers (0.375 for example).
  - **`"Percent"`**: Using percent notation for numbers (35.7% for example).
  - **`"Time"`**: Interpreting numbers as times (00:03:25,87 for example).
  - **`"Number"`**: Interpreting time values as normal numbers (format defined by **`Percent`** or **`Fraction`**).
- **`"void setSeparator(final String separator)"`**:
  This command allows to select the separator to be used when printing out distributions of measured values.
  - **`"Semicolon"`**: Semicolons as separators
  - **`"Line"`**: Line break as separators
  - **`"Tabs"`**: Tabulators as separators

22.2 Accessing statistics xml data

- **`"String xml(final String path)"`**:
  Loads the xml field which is specified by the parameter and returns the data in the format defined by **`sim.getStatistics().setFormat`** and **`sim.getStatistics().setSeparator`** as a string.

Example: `String name=sim.getStatistics().xml("Model->ModelName")`

- `"Object xmlNumber(final String path)"`:
  Loads the xml field which is specified by the parameter and returns the value as a `Double` number. If the field cannot be interpreted as a number, a string containing an error message will be returned.
- `"Object xmlArray(final String path)"`:
  Loads the xml field which is specified by the parameter, interprets it as a distribution and returns the values as an array of numbers (`double[]`). If the field cannot be interpreted as a distribution, a string containing an error message will be returned.

  Example:
  ```
  sim.getStatistics().xmlArray("StatisticsProcessTimesClients->
  ClientType[Type=\"ClientsA\"]->[Distribution]")
  ```
- `"Object xmlSum(final String path)"`:
  Loads the xml field which is specified by the parameter, interprets it as a distribution and returns the sum of all values as a `Double` number. If the field cannot be interpreted as a distribution, a string containing an error message will be returned.

  Example:
  ```
  sim.getStatistics().xmlSum("StatisticsProcessTimesClients->
  ClientType[Type=\"ClientsA\"]->[Distribution]")
  ```
- `"Object xmlMean(final String path)"`:
  Loads the xml field which is specified by the parameter, interprets it as a distribution and returns the mean of values as a `Double` number. If the field cannot be interpreted as a distribution, a string containing an error message will be returned.

  Example:
  ```
  sim.getStatistics().xmlMean("StatisticsProcessTimesClients->
  ClientType[Type=\"ClientsA\"]->[Distribution]")
  ```
- `"Object xmlSD(final String path)"`:
  Loads the xml field which is specified by the parameter, interprets it as a distribution and returns the standard deviation of values as a `Double` number. If the field cannot be interpreted as a distribution, a string containing an error message will be returned.

  Example: `sim.getStatistics().xmlSD("StatisticsProcessTimesClients->`
  `ClientType[Type=\"ClientsA\"]->[Distribution]")`
- `"Object xmlCV(final String path)"`:
  Loads the xml field which is specified by the parameter, interprets it as a distribution and returns the coefficient of variation of values as a `Double` number. If the field cannot be interpreted as a distribution, a string containing an error message will be returned.

  Example:
  ```
  sim.getStatistics().xmlCV("StatisticsProcessTimesClients->
  ClientType[Type=\"ClientsA\"]->[Distribution]")
  ```
- `"boolean translate(final String language)"`:
  Translates the statistics data to English ("en") or German ("de") so the preferred xml tag names can be used independent of the language setting under which the statistics file was generated.

22.3 Saving the statistics data to files

- `"boolean save(final String fileName)"`:
  Saves the entry statistics data under the next available file name in the given folder.
  This function is only available in the Run script panel.
- `"boolean saveNext(final String folderName)"`:
  Saves the entry statistics data under the next available file name in the given folder.
  This function is only available in the Run script panel.
- `"String filter(final String fileName)"`:
  Applies the selected script on the statistics data and returns the results.
  This function is only available in the Run script panel.
- `"void cancel()"`:
  Sets the cancel status. (When output is canceled to further file output will be performed.)

22.4 Accessing station data

- `"int getStationID(final String name)"`:
  Gets the ID of a station based on its name. If there is no station with a matching name, the function will return -1.

22.5 Retrieve the associated statistics file

- `"String getStatisticsFile()"`:
  Returns the full path and file name of the statistics file from which the data was loaded. If the statistic data was not loaded from a file, an empty string is returned.
- `"String getStatisticsFileName()"`:
  Returns the file name of the statistics file from which the data was loaded. If the statistic data was not loaded from a file, an empty string is returned.

Chapter 23

RuntimeInterface accessible via sim.getRuntime()

The `RuntimeInterface` interface allows to access some general program functions.
The `RuntimeInterface` is always available. The following methods are available in this interface:

- **`"Object calc(final String expression)"`**:
  Calculates the expression passed as a string by means of the term evaluation function, which is also used in various other places in the program (see part I), and returns the result as a **`Double`** number. If the expression can not be calculated, an error message is returned as a string. The term evaluation function allows access to all known probability distributions, the Erlang C calculator, etc.
- **`"long getTime()"`**:
  Returns the current system time as a milliseconds value. This functions can be used to measure the runtime of a script.
- **`"double getInput(final String url, final double errorValue)"`**:
  Loads a numerical value via the specified address and returns it. If no value could be loaded, the error value specified in the second parameter is returned.
- **`"boolean execute(final String commandLine)"`**:
  Executes an external command and returns immediately. Returns true, if the program could be started. Executing external programs by scripts is disabled by default. If can be activated in the program settings dialog.
- **`"String executeAndReturnOutput(final String commandLine)"`**:
  Executes an external command and returns the output. Executing external programs by scripts is disabled by default. If can be activated in the program settings dialog.
- **`"int executeAndWait(final String commandLine)"`**:
  Executes an external command, waits for completion and returns the return code of the program. In case of an error -1 will be returned. Executing external programs by scripts is disabled by default. If can be activated in the program settings dialog.

Chapter 24

SystemInterface accessible via sim.getSystem()

The SystemInterface interface allows to access the model data while simulation is running. It is not available for filtering the results after simulation has terminated. The following methods are available in this interface:

24.1 Base functions

- "double getTime()":
  Gets the current time in the simulation as a seconds numerical value.
- "Object calc(final String expression)":
  Calculates the expression passed as a string by means of the term evaluation function, which is also used in various other places in the program (see part I), and returns the result as a **Double** number. If the expression can not be calculated, an error message is returned as a string. The term evaluation function allows access to all known probability distributions, the Erlang C calculator, etc.
- "boolean isWarmUp()":
  Gets true of false depending if the simulation is in the warm-up phase.
- "Map<String,Object> getMapLocal()":
  Returns a station-local mapping into which values can be written and from which values can be read. The values stored here are retained beyond the execution of the current script.
- "Map<String,Object> getMapGlobal()":
  Returns a model wide mapping into which values can be written and from which values can be read. The values stored here are retained beyond the execution of the current script.
- "void pauseAnimation()":
  Switches the animation to single step mode. If the animation is already executed in single step mode or if the model is executed as a simulation, this command has no effect.
- "void terminateSimulation(final String message)":
  Terminates the simulation. If **null** is passed as message, the simulation is terminated normally. In case of a message, the simulation will be terminated with the corresponding error message.

24.2 Accessing parameters of the simulation model

- "void set(final String varName, final Object varValue)":
  Sets the simulation variable which is specified as the first parameter to the value specified as the second

parameter. **varValue** can be a number or a string. The case of a number the value will be assigned directly. Strings will be interpreted like **calc(final String expression)** does and the result will be assigned to the variable. **varName** can either be the name of an already defined simulation variable or of a client data field in the form **ClientData(index)** with $index \geq 0$.

- **"void setAnalogValue(final Object elementID, final Object value)"**:
  Sets the value at the "Analog value" or "Tank" element with the specified id.
- **"void setAnalogRate(final Object elementID, final Object value)"**:
  Sets the change rate (per second) at the "Analog value" element with the specified id.
- **"void setAnalogValveMaxFlow(final Object elementID, final Object valveNr, final Object value)"**:
  Sets the maximum flow (per second) at the specified valve (1 based) of the "Tank" element with the specified id. The maximum flow has to be a non-negative number.
- **"int getWIP(final int id)"**:
  Gets the current number of clients at the station with the specified id.
- **"int getNQ(final int id)"**:
  Gets the current number of clients in the queue at the station with the specified id.
- **"int getWIP()"**:
  Gets the current number of clients in the system.
- **"int getNQ()"**:
  Gets the current number of waiting clients in the system.

24.3 Number of operators in a resource

- **"int getAllResourceCount()"**:
  Returns the current number of operators in all resources together.
- **"int getResourceCount(final int resourceId)"**:
  Returns the current number of operators in the resource with the specified id.
- **"boolean setResourceCount(final int resourceId, final int count)"**:
  Sets the number of operators in the resource with the specified id. To be able to set the number of operators in a resource at runtime, the initial number of operators in the resource has to be a fixed number (not infinite many and not by a time table). Additionally no down times are allowed for this resource. The function returns **true** if the number of operators has successfully been changed. If the new number of operators is less than the previous number, the new number may is not instantly visible in the simulation system because removed but working operators will finish their current tasks before they are actually removed.
- **"int getAllResourceDown()"**:
  Returns the current number of operators in down time in all resources together.
- **"int getResourceDown(final int resourceId)"**:
  Returns the current number of operators in down time in the resource with the specified id.

24.4 Fire signal

- **"signal(final String signalName)"**:
  Fires the signal with the given name.

24.5 Trigger script execution

- `"boolean triggerScriptExecution(final int stationId, final double time)"`:
  Triggers the execution of the script at a script or a script hold station at a given time.

24.6 Run external code

- `"Object runPlugin(final String className, final String functionName, final Object data)"`:
  Runs the specified method in the specified class and passes the optional parameter `data` to the method. The return value of the method will be returned by `runPlugin`. If calling the external method fails, `runPlugin` will return `null`.

24.7 Output message in logging

- `"void log(final Object obj)"`:
  Outputs the passed message to the logging system (if logging is enabled).

24.8 Release clients at delay stations

If a list of clients at a delay station is recorded, this list can be queried using the following function and individual clients can be selectively released before their specified delay time has expired.

- `"ClientsInterface getDelayStationData(final int id)"`:
  Returns an object implementing `ClientsInterface` which represents the list of clients at the delay station `id`. If the id is invalid, `null` will be returned.

24.9 Clients in the queue of a process station

- `"ClientsInterface getProcessStationQueueData(final int id)"`: Returns an object implementing `ClientsInterface` which represents the list of clients waiting at the process station `id`. If the id is invalid, `null` will be returned. Only the waiting clients can be accessed, not the clients which are already in service process. Also clients cannot be released via `ClientsInterface.release` here.

Chapter 25

`ClientInterface` accessible via `sim.getClient()`

The `ClientInterface` interface allows to access the data of the current client while the simulation is running. It is only available if the execution was triggered by a client. The following methods are available in this interface:

- `"Object calc(final String expression)"`:
  Calculates the expression passed as a string by means of the term evaluation function, which is also used in various other places in the program (see part I), and returns the result as a `Double` number. If the expression can not be calculated, an error message is returned as a string. The term evaluation function allows access to all known probability distributions, the Erlang C calculator, etc.
- `"String getTypeName()"`:
  Gets the name of the type of the client who has triggered the processing of the script.
- `"boolean isWarmUp()"`:
  Gets `true` of `false` depending if the current client was generated during the warm-up phase and therefore will not be recorded in the statistics.
- `"boolean isInStatistics()"`:
  Gets true of false depending if the current client is to be recorded in the statistics. This value is independent of the warm-up phase. A client will only be recorded if he was generated after the warm-up phase and this value is true.
- `"void setInStatistics(final boolean inStatistics)"`:
  Sets if a client is to be recorded in the statistics. This value is independent of the warm-up phase. A client will only be recorded if he was generated after the warm-up phase and this value is not set to false.
- `"long getNumber()"`:
  Get the 1 based consecutive number of the current client. When using multiple simulation threads this number is thread local.
- `"double getWaitingSeconds()"`:
  Gets the current waiting time of the client who has triggered the processing of the script as a seconds numerical value.
- `"String getWaitingTime()"`:
  Gets the current waiting time of the client who has triggered the processing of the script as a formated time value as a string.
- `"void setWaitingSeconds(final double seconds)"`:
  Sets the current waiting time of the client who has triggered the processing of the script.

- **"double getTransferSeconds()"**:
  Gets the current transfer time of the client who has triggered the processing of the script as a seconds numerical value.
- **"String getTransferTime()"**:
  Gets the current transfer time of the client who has triggered the processing of the script as a formated time value as a string.
- **"void setTransferSeconds(final double seconds)"**:
  Sets the current transfer time of the client who has triggered the processing of the script.
- **"double getProcessSeconds()"**:
  Gets the current processing time of the client who has triggered the processing of the script as a seconds numerical value.
- **"String getProcessTime()"**:
  Gets the current processing time of the client who has triggered the processing of the script as a formated time value as a string.
- **"void setProcessSeconds(final double seconds)"**:
  Sets the current processing time of the client who has triggered the processing of the script.
- **"double getResidenceSeconds()"**:
  Gets the current residence time of the client who has triggered the processing of the script as a seconds numerical value.
- **"String getResidenceTime()"**:
  Gets the current residence time of the client who has triggered the processing of the script as a formated time value as a string.
- **"void setResidenceSeconds(final double seconds)"**:
  Sets the current residence time of the client who has triggered the processing of the script.
- **"double getValue(final int index)"**:
  Gets for the current client the numerical value which is stored by the index **index**.
- **"void setValue(final int index, final int value)"**,
  "void setValue(final int index, final double value)",
  "void setValue(final int index, final String value)":
  Sets for the current client the **value** for the index **index**. If **value** is a string, the string is interpreted by **calc(final String expression)** before assigning the result.
- **"String getText(final String key)"**:
  Gets for the current client the string which is stored by the key **key**.
- **"void setText(final String key, final String value)"**:
  Sets for the current client string **value** for the key **key**.
- **"double[] getAllValues()"**: Return all numerical values stored for the current client.
- **"Map<String,String> getAllTexts()"**: Return all text values stored for the current client.

25.1 Temporary batches

If the current client is a temporary batch, the properties of the inner clients it contains can be accessed in read-only mode:

- `"int batchSize()"`:
  Returns the number of clients that are in the temporary batch. If the current client is not a temporary batch, the function returns 0.
- `"String getBatchTypeName(final int batchIndex)"`:
  Returns the name of one of the clients in the current batch. The passed index is 0-based and must be in the range from 0 to `batchSize()-1`.
- `"double getBatchWaitingSeconds(final int batchIndex)"`:
  Returns the previous waiting time of one of the clients in the current batch in seconds as a numerical value. The passed index is 0-based and must be in the range from 0 to `batchSize()-1`.
- `"String getBatchWaitingTime(final int batchIndex)"`:
  Returns the previous waiting time of one of the clients in the current batch in formatted form as a string. The passed index is 0-based and must be in the range from 0 to `batchSize()-1`.
- `"double getBatchTransferSeconds(final int batchIndex)"`:
  Returns the previous transfer time of one of the clients in the current batch in seconds as a numerical value. The passed index is 0-based and must be in the range from 0 to `batchSize()-1`.
- `"String getBatchTransferTime(final int batchIndex)"`:
  Returns the previous transfer time of one of the clients in the current batch in formatted form as a string. The passed index is 0-based and must be in the range from 0 to `batchSize()-1`.
- `"double getBatchProcessSeconds(final int batchIndex)"`:
  Returns the previous processing time of one of the clients in the current batch in seconds as a numerical value. The passed index is 0-based and must be in the range from 0 to `batchSize()-1`.
- `"String getBatchProcessTime(final int batchIndex)"`:
  Returns the previous processing time of one of the clients in the current batch in formatted form as a string. The passed index is 0-based and must be in the range from 0 to `batchSize()-1`.
- `"double getBatchResidenceSeconds(final int batchIndex)"`:
  Returns the previous residence time of one of the clients in the current batch in seconds as a numerical value. The passed index is 0-based and must be in the range from 0 to `batchSize()-1`.
- `"String getBatchResidenceTime(final int batchIndex)"`:
  Returns the previous residence time of one of the clients in the current batch in formatted form as a string. The passed index is 0-based and must be in the range from 0 to `batchSize()-1`.
- `"double getBatchValue(final int batchIndex, final int index)"`:
  Returns a stored numerical value for one of the clients in the current batch. The passed batch index is 0-based and must be in the range from 0 to `batchSize()-1`.
- `"String getBatchText(final int batchIndex, final String key)"`:
  Returns a stored text value for one of the clients in the current batch. The passed batch index is 0-based and must be in the range from 0 to `batchSize()-1`.

Chapter 26

`InputValueInterface` accessible via `sim.getInputValue()`

The `InputValueInterface` interface allows to access the next input value if the processing was triggered by a Input (Script) element. The following methods are available in this interface:

- `"double get()"`:
  This function returns the current input value.

Chapter 27

`ClientsInterface` accessible via `sim.getClients()`

The `ClientsInterface` object is only available within a hold by script condition element and allows to access the waiting clients and to release them.

- **`"int count()"`**:
  Returns the current number of waiting clients. For the other methods a single client can be accessed via the index parameter (valued from 0 to `count()`-1).
- **`"String clientTypeName(final int index)"`**:
  Gets the name of the type of the client.
- **`"double clientWaitingSeconds(final int index)"`**:
  Gets the current waiting time of the client as a seconds numerical value.
- **`"String clientWaitingTime(final int index)"`**:
  Gets the current waiting time of the client as a formated time value as a string.
- **`"double clientTransferSeconds(final int index)"`**:
  Gets the current transfer time of the client as a seconds numerical value.
- **`"String clientTransferTime(final int index)"`**:
  Gets the current transfer time of the client as a formated time value as a string.
- **`"double clientProcessSeconds(final int index)"`**:
  Gets the current processing time of the client as a seconds numerical value.
- **`"String clientProcessTime(final int index)"`**:
  Gets the current processing time of the client as a formated time value as a string.
- **`"double clientResidenceSeconds(final int index)"`**:
  Gets the current residence time of the client as a seconds numerical value.
- **`"String clientResidenceTime(final int index)"`**:
  Gets the current residence time of the client as a formated time value as a string.
- **`"void clientData(final int index, final int data, final double value)"`**:
  Set the numerical value specified by the third parameter for the data element which index is specified via the second parameter of the selected client.
- **`"double clientData(final int index, final int data)"`**:
  Returns the data element which index is specified via the second parameter of the selected client.
- **`"String clientTextData(final int index, final String key)"`**:
  Returns the data element which key is specified via the second parameter of the selected client.

- **`"String clientTextData(final int index, final String key, final String value)"`**:
  Set the text value specified by the third parameter for the key which is specified via the second parameter of the selected client.
- **`"void release(final int index)"`**:
  Causes the forwarding of the specified client.

Chapter 28

`OutputInterface` accessible via `sim.getOutput()`

The `OutputInterface` interface provides functions for output of filtered results:

- **`"void setFormat(final String format)"`**:
  This command allows to setup the format that is used in **`print`** and **`println`** for outputing numbers as strings. You can specify whether to use a floating point notation or percent notation or interpreting the value as a time. As default floating point notation is used.
  - **`"Fraction"`**:
    Using floating point notation for numbers (0.375 for example).
  - **`"Percent"`**:
    Using percent notation for numbers (35.7% for example).
  - **`"Number"`**:
    Interpreting numbers as normal number values (decimal or percent).
  - **`"Time"`**:
    Interpreting numbers as time values.
- **`"void setSeparator(final String separator)"`**:
  This command allows to select the separator to be used when printing out arrays.
  - **`"Semicolon"`**:
    Semicolons as separators.
  - **`"Line"`**:
    Line break as separators.
  - **`"Tabs"`**:
    Tabulators as separators.
- **`"void setDigits(final int digits)"`**:
  This command allows to define the number of digits to be displayed when printing a number in local notation. A negative value means that all available digits are being printed. (If the system notation is used, always all available digits are being printed.)
- **`"void print(final Object obj)"`**:
  Outputs the passed expression. Strings will be written directly. Numbers are formated according to the format defined via **`setFormat`**.

- "**void println(final Object obj)**":
  Outputs the passed expression and adds a line break after the expression. Strings will be written directly. Numbers are formated according to the format defined via **setFormat**.
- "**void newLine()**":
  Outputs a line break. This functions is equivalent to calling **println("")**.
- "**void tab()**":
  Outputs a tabulator.
- "**void cancel()**":
  Sets the cancel status. (When output is canceled to further file output will be performed.)
- "**printlnDDE(final String workbook, final String table, final String cell, final Object obj)**":
  This command is only available if DDE is available, i.e. under Windows. It outputs the passed expression via DDE in the specified table in Excel. Numbers are formated according to the format defined via **setFormat**.

Chapter 29

`FileOutputInterface` accessible via `sim.getFileOutput()`

The `FileOutputInterface` interface offers all function the `OutputInterface` interface has but is only available when running a parameter series script. In opposite to the `OutputInterface` interface the output of the `FileOutputInterface` interface is not written to the default output but is appended to a file which has to be specified by `sim.getFileOutput().setFile("Filename")` before.

Chapter 30

ModelInterface accessible via sim.getModel()

The `ModelInterface` interface is only available during parameter series script execution and offers functions for accessing the model properties and for starting simulations.

- **"void reset()"**:
  Resets the model to the initial state.
- **"void run()"**:
  Simulates the current model. The results can be accessed by the **StatisticsInterface** interface after the simulation.
- **"boolean setDistributionParameter(final String xmlName, final int number, final double value)"**:
  Sets the distribution parameter **number** (from 1 to 4) of the distribution referred to by **xmlName**.
- **"boolean setMean(final String xmlName, final double value)"**:
  Sets the mean of the distribution referred to by **xmlName** to the specified value.
- **"boolean setSD(final String xmlName, final double value)"**:
  Sets the standard deviation of the distribution referred to by **xmlName** to the specified value.
- **"boolean setString(final String xmlName, final String value)"**:
  Writes the string **value** to the location referred to by **xmlName**.
- **"boolean setValue(final String xmlName, final double value)"**:
  Writes the number **value** to the location referred to by **xmlName**.
- **"String xml(final String xmlName)"**:
  Loads the xml field which is specified by the parameter and returns the data as String. This function is the equivalent of **sim.getStatistics().xml(xmlName)** for models.
- **"getResourceCount(final String resourceName)"**:
  Gets the number of operations in the resource with name **resourceName**. If the resource does not exist or is not defined by a fixed number of operators the function will return -1.
- **"boolean setResourceCount(final String resourceName, final int count)"**:
  Sets the number of operations in the resource with name **resourceName**.
- **"String getGlobalVariableInitialValue(final String variableName)"**:
  Gets the expression by which the initial value of the global variable with name **variableName** is calculated. If the is no global variable with this name the function will return an empty string.
- **"boolean setGlobalVariableInitialValue(final String variableName, final String expression)"**:

Sets the expression by which the initial value of the global variable with name **variableName** is calculated.

- "String getGlobalMapInitialValue(final String variableName)":
  Gets the initial value of the entry **VariablenName** of the global map. If there is no entry with this name, **null** will be returned.
- "boolean setGlobalMapInitialValue(final String variableName, final String expression)":
  Sets the initial value (of type **Integer**, **Long**, **Double** oder **String**) of the key **VariablenName** in the global map.
- "void cancel()":
  Sets the cancel status. (When processing is canceled to further simulations will be performed.)

30.1 Accessing station data

- "int getStationID(final String name)":
  Gets the ID of a station based on its name. If there is no station with a matching name, the function will return -1.

Chapter 31
XML selection commands

By the parameters of the functions of the **"StatisticsInterface"** interface the content of the value or of an attribute of an XML element can be read. The selection of an XML element is done multistaged step by step divided by "->" characters. Between the "->" characters the names of the individual XML nodes are noted. In addition in square brackets names and values of attributes can be specified to filter by whom.

Examples:

- **"sim.getStatistics().xml("Model->ModellName")"**:
  Shows the content of the element **ModelName**, which is a child element of **Model**.
- **"sim.getStatistics().xml("StatisticsInterArrivalTimesClients->
  Station[Type=\"Source id=1\"]->[Mean]")"**:
  Selects the **Station** sub element of the **StatisticsInterArrivalTimesClients** element, for which the **Type** attribute is set to **Source id=1**. And returns the value of the **Mean** attribute.