!

Operation
1/25/2004

Operation of logical negation.

```
!x
```

1. x is numbers:
– If the number is non-zero returns 0 (false), otherwise returns 1(true).

Example

```
N/A
```

!=

Operation
1/25/2004

Test for inequality.

```
x!=y
```

1. x and y are both numbers:
– returns 1 if |x-y|/min(|x|,|y|)>=machine epsilon and 0 otherwise
2. x and y are strings
– tests for case sensitive string inequality

Example

```
N/A
```

#include

Batch language command
2/4/2004

File inclusion command. The file specified as the argument of the include command is inserted in place of the #include line.

```
#include "filename";
```

filename must be given relative to the batch file which contains the #include statement.

';' at the end of the command is required.

The inclusion is processed at COMPILE time.

Example

```
#include "Models/F84.def";
```

$

Operation
2/3/2004

Operation of integer division, elementwise matrix multiplication, regular expression matching in strings and tree splitting.

x$y

1. x and y are both numbers:
– rounds both arguments and then returns the result of integer division
2. x and y are matrices of the same dimension
– multiplies matrices element by element
3. x and y are strings
– tries to find a regular expression defined by y in x. Returns a 2x1 matrix which contains 0 based starting and ending offsets of the first match in x. If no match is found, then both offsets are set to -1. Regular expressions are supported by incorporation of a C++ adapted version of GNU POSIX compliant library http://www.gnu.org/directory/regex.html
4. x is a topology or a tree with at least 10 leaves, y is a positive number:
– splits the tree into y subtrees of approximately equal size; returns a matrix of names of internal where split points were.

Example

```
123$4
/* returns 30*/

{{1,2}{3,4}}${{5,10}{15,20}}
/*returns
{{ 5, 20}
{ 45, 80}}*/

"abcdef"$"cd."
/* returns {{2}{4}}*/

"abcdef"$"cd.$"

/* returns {{-1}{-1}}*/
```

%

Operation
3/1/2004

Operation of remainder of integer division. Operation of case insensitive string comparison. Operation of matrix sorting.

x%y

1. x and y are both numbers:
– Rounds both arguments and then returns the remainder of integer division.
2. x and y are strings
– compares x and y lexicographically, ignoring case.
3. x is a matrix and y is an integer
– sorts the matrix on column y

Example

```
7%3
/* returns 1*/
"hAckErS"%"hackers"
/* 1 - true */
m = {{5,3,2,6,10,-1}{0,1,2,3,4,5}};
m = Transpose(m);
ms = Transpose(m%0);
fprintf (stdout, ms);
/* output -
{
{ -1, 2, 3, 5, 6, 10}
{ 5, 2, 1, 0, 3, 4}
}
*/
```

&&

Operation
9/15/2005

Operation of conjunction (logical AND) or change case/character escape in a string.

`x&&y`

1. x and y are both numbers:
– if both arguments are non-zero returns 1(true), otherwise returns 0.
2. x is a string and y is
– 0: returns lower cased x,
– 1: returns upper cased x.
– 2: returns a version of the string with some formatting characters escaped

Example

```
N/A
```

*

Operation
1/24/2004

Operation of numeric and matrix multiplication or string streaming.

`x*y, or , x y(implied multiplication).`

1. x and y are both numbers:
– performs numeric multiplication.
2. x is a matrix and y is a number:
– term wise multiplication of the matrix.
Note: number * matrix will return an error.
3. x and y are matrices, whose dimensions are compatible, i.e the number of columns in the x is the same as the number of rows in y
– matrix multiplication
4. x is a string and y is:
a). Positive number:
– clears x and allocates at least y bytes for storage in x
b). Zero (or negative number):
– finalizes x. MUST be called before x is usable
c). Another string:
– appends y to the end of x, automatically allocating memory if needed.

Note: the purpose of this command is to create large strings by writing small fragments to them and to avoid unneeded memory swapping.

Example

```
2(x+y)
{{1,2}{3,4}}*(-1)
/* returns {{-1,-2}{-3,-4}})*/
{{1,1}}*{{2}{3}}
/* (returns {{5}}) */
largeString = "";
largeString*1024;
for (k=1; k<100; k=k+1)
{
largeString*("The square of "+k+" is "+k^2+" ");
}
largeString*0;
fprintf (stdout, largeString);
```

+

Operation
9/14/2005

Operation of numeric and matrix addition as well as string conversion and concatenation.

`x+y`

1. x and y are both numbers:
– performs numeric addition.
2. x and y are both matrices of the same dimensions:
– performs matrix addition.
3. x and y are both matrices:
– performs matrix addition.
4. x is a string and y is an arbitrary object:
– converts y to a string and concatenates the result to x.
5. x is a number and y is an string
– converts y to a number and adds it to x
6. x is a matrix and y is a number
– Adds y to every element of x. Order is important (e.g. number+matrix won't work). To subtract a number use matrix + (-number)

Example

```
str = "Hello "+"World";
/*String Concatenation:
str' now holds "Hello World"*/
Tree t = ((a,b),c);
tree_string=""+t;
/*Object conversion
tree_string= now holds a string representation of t */
```

-

Operation
1/23/2004

Operation of numeric or matrix subtraction.

`x-y, or -x (unary minus)`

1. x and y are both numbers:
– performs numeric subtraction.
2. x and y are both matrices of the same dimension:
– performs matrix subtraction.

Example

```
N/A
```

/

Operation
1/25/2004

Operation of floating point division or wildcard string comparison

```
x/y
```

1. x and y are both numbers:
– performs numeric division
2. x and y are strings
– compares x to y, allowing for '*' wildcards in y (only in y).

Example

```
"Unix"/"*nix"
/*1 - true*/
"Linux"/"*nix"
/*0 - false*/
```

:=

Batch language command
2/4/2004

Dependance assignment operator. Assigns the formula defined by the rhs to the lhs. This operator should be used to constrain variables before optimization.

```
LHS := RHS;
```

LHS must be an identifier or a matrix element accessed by the bracket operator. If LHS is an identifier of a variable which doesn't exist, the variable will be created.

Example

```
a=2;
b=3;
c:=a+b;
d=a+b;
a=4;
fprintf (stdout, " ", a," ",b," ",c," ",d, " ");
/* outputs 4 3 7 5.*/
```

<

Operation
3/3/2004

Operation 'less'. Also computes log likelihood of a markov chain realization, given the rate matrix.

`x<y`

1. x and y are both numbers:
– returns 1 if x<y (within machine epsilon) and 0 otherwise
2. x and y are both strings:
– performs a case-sensitive < lexicographic comparison
3. x is a 3xN matrix and y is an MxM matrix:
– using MxM as the rate matrix for an M state Markov chain, computes the log-likelihood of the realization specified in x. Each column in x is has the 'from' state in row 1, the 'to' state in row 2 (both must be integers in [0,M-1]), and row 3 is the time over which the step occurs.

Example

```
a = 1;
rates = {{*,a,a,a}{.5*a,*,a,a}{a,a,*,a}{a,a,a,*}};
jumps = {{0,1,2,3}
{1,2,3,0}
{0.1,0.2,0.3,0.05}};
/* chain 0->1->2->3->0 with jump times of 0.1, 0.2, 0.3 and 0.05 */
fprintf (stdout, " ", jumps<rates, " ");
```

<=

Operation
1/25/2004

Operation 'less than or equal to', and tree pattern matching.

`x<=y`

1. x and y are both numbers:
– returns 1 if x<=y (within machine epsilon) and 0 otherwise
2. x and y are both strings:
– performs a case-sensitive <= lexicographic comparison
3. x and y are both topologies or trees:
– tests to see whether tree x matches pattern defined by tree y, i.e. whether y can be obtained from x by collapsing internal branches, sibling swapping and rerooting.

Example

```
Topology t1=(((1,2),3),(4,5),6);
Topology pattern1 = (1,2,3,(4,5,6));
Topology pattern2 = (1,2,3,(6,(5,4)));
Topology pattern3 = ((1,2,4),3,5,6);
t1<=pattern1;
t1<=pattern2;
/* both true */
t1<=pattern3;
/* false */
```

=

Batch language command
2/4/2004

Value assignment operator. Assigns the current value of the rhs to the lhs. To create dependancies use ':='.

```
LHS = RHS;
```

LHS must be an identifier or a matrix element accessed by the bracket operator. If LHS is an identifier of a variable which doesn't exist, the variable will be created.

Example

```
a=2;
b=3;
c:=a+b;
d=a+b;
a=4;
fprintf (stdout, " ", a," ",b," ",c," ",d, " ");
/* outputs 4 3 7 5.*/
```

==

Operation
1/25/2004

Test of equality.

```
x==y
```

1. x and y are both numbers:
– returns 1 if |x-y|/min(|x|,|y|)<machine epsilon and 0 otherwise
2. x and y are strings
– tests for case sensitive string equality
3. x and y are topologies or trees
– compares trees for equality in the sense of unrooted labeled trees, i.e. if one tree can be rerooted and sibling swapped to obtain the other - returns true.

Example

```
Topology t1=((1,2),3,4);
Tree t2=(1,(3,4),2);
Topology t3=(1,4,(2,3));
t1 == t2;
/* 1 - true */
t1 == t3;
/* 0 - false */
```

>

Operation
1/25/2004

Operation 'greater than'. Also can be used to build tree matrix representations from a square distance matrix.

```
x>y
```

1. x and y are both numbers:
– returns 1 if x>y (within machine epsilon) and 0 otherwise

2. x and y are both strings:
– performs a case-sensitive > lexicographic comparison
3. x is an NxN upper triangular matrix:
– performs a neighbor joining algorithm on the matrix and returns a "parent" flat tree representation (an 3x(2N-3) matrix) for more details refer to example files.

Example

```
N/A
```

>=

Operation
1/25/2004

Operation 'greater than or equal to'. Also can be used to build trees from parent node representations.

`x>=y`

1. x and y are both numbers:
– returns 1 if x>=y (within machine epsilon) and 0 otherwise
2. x and y are both strings:
– performs a case-sensitive >= lexicographic comparison
3. x is an Nx3 matrix specifying a tree (see example files for more detail), such as ones returned by the neighbor joining operation, and y is the number of tree leaves (an interger <= (N-3)/2):
– returns a "flat" tree representation as another matrix

Example

```
see example files for more details on case 3.
```

ACCEPT_BRANCH_LENGTHS

Constant
2/5/2004

Whether or not to accept branch lengths in provided in the tree string (ON by default).

`N/A`

If set to '0', branch lengths in the tree string are ignored.

Example

```
N/A
```

ACCEPT_ROOTED_TREES

Constant
2/5/2004

Whether or not to unroot rooted trees (OFF by default).

```
N/A
```

If set to '1', no automatic tree unrooting will be done.

Example

```
Topology unrooted= ((1,2),3);
ACCEPT_ROOTED_TREES=1;
Topology rooted = ((1,2),3);
ACCEPT_ROOTED_TREES=0;
```

ALLOW_BOUNDARY

Constant
2/3/2004

If true (default) then variables are allowed to assume boundary values.

```
ALLOW_BOUNDARY = 0 or 1;
```

If ALLOW_BOUNDARY = 0, then no independent variable will be allowed to assume values closer than $10\hat{(}$-25) to the boundary. You may want to use this setting if the actual boundary value results in a singular behavior. It is better to set boundary values for each parameter individually if needed.

Example

```
N/A
```

ASSIGNED_SEED

Built-in function
2/4/2004

If ASSIGNED_SEED is set, then it is fed to the standard random number generator seed function. Can be used to replicate simulation results. DISABLED IN VERSION 0.99 AND ABOVE.

```
ASSIGNED_SEED=value;
```

Use SetParameter to specify a user value for the seed.

Example

```
N/A
```

AUTO_PARALLELIZE_OPTIMIZE

Constant
9/14/2005

[MPI Only] Tries to automatically spread likelihood function evaluation across multiple nodes.

```
AUTO_PARALLELIZE_OPTIMIZE = 0 (default) or 1;
```

Works only for some types of likelihood functions (will report a warning message if auto-parallelization is not possible). Works best for long sequences; involved substantial overhead and may not scale linearly (but does linearly (with coefficient < 1) if sequences are long enough).

Example

```
AUTO_PARALLELIZE_OPTIMIZE = 1;
Optimize (res, lf);
AUTO_PARALLELIZE_OPTIMIZE = 0;
/* requires an MPI environment to have an effect; flag ignored in single thread or p-thread builds).
```

Abs

Built-in function
9/16/2005

Absolute value, matrix norm, string length and tree flat representation.

`Abs(x)`

1. x is a number:
– returns |x|.
2. x is a vector (column or row matrix):
– returns the Euclidean norm ("length") of x.
3. x is a matrix which is not a vecor:
– returns maximum absolute value among matrix elements.
4. x is a string
– returns character length of x
4. x is a tree or topology:
– returns a 1xN matrix (N is the total number of tree nodes), where each entry corresponds to a node in a post-order traversal of the tree and contains the (0-based) index (in post order traversal) of that node's parent (-1 for the root)

Example

```
Abs({{1,2}{-4,3}})
/*returns 4*/
Abs ("foobar")
/*returns 6.*/
Abs({{1,1}})
/*returns 1.41421 (i.e. the square root of 2).*/
Topology t=((1,2)N1,3,4);
bp = Abs (t);
branchNames = BranchName(t,-1);
for (bi=0; bi<Columns(bp); bi=bi+1)
{
parentIndex = bp[bi];
if (parentIndex >=0)
{
fprintf (stdout, "Node ", branchNames[bi], " <= Parent ", branchNames[parentIndex], " ");
}
else
{
fprintf (stdout, "Node ", branchNames[bi], " is the root ");
}
}
/* returns {{2,2,5,5,5,-1}};
the post order traversal of the tree is:
1,2,N1,3,4,root; the parent of 1 and 2 is traversed 3, the parent of N1,3 and 4 is the root, which is traversed 6th. */
```

Arctan

Built-in function
1/25/2004

Inverse Tangent.

`Arctan(x)`

1. x is a number:
– Returns the inverse tangent of x.

Example

```
N/A
```

BASESET

Data File formatting instruction
2/4/2004

BASESET is used to indicate what characters form the fundamental states of the data.

```
$BASESET:"new set" (new line char)
```

If "new set" = "BASE20", the data is assumed to contain aminoacid info (with standard ambigutity characters).

Must be present in the file before any data begins and before any TOKEN directives are issued. The new character set is simply a string where each character is understood as a new state and the first character is mapped to state '0, the 2nd - to state '1'etc.

Only ASCII characters with codes 30 to 127 should be used for definition of the base set, and the translation is NOT case sensitive.

Does not apply to NEXUS files, which have their own mechanism for defining base character sets.

Example

```
$BASESET:"ACGU" (RNA data)
$BASESET:"01" (binary data)
```

BRACKETING_PERSISTENCE

Constant
2/4/2004

In conjunction with randomized order bracketing, specifies, how many times, on average each variable will be bracketed. Defualt value is 2.5.

```
BRACKETING_PERSISTENCE=value;
```

This options is only meaningful when an appropriate optimization method is chosen by setting OPTIMIZATION_METHOD.

Example

```
N/A
```

Beta

Built-in function
2/3/2004

The 'Beta' function.

```
Beta(x,y)
```

1. x and y are real numbers, which are not negative integers:
– computes Beta(x,y)=Gamma(x)Gamma(y)/Gamma(x+y), where Gamma is the gamma function.

Example

```
N/A
```

Bootstrap

Batch language command
2/4/2004

Can be used to create samples (with replacement) of blocks of columns of 'unit' length of existing data sets or data set filters.

```
DataSetFilter <filterid> = Bootstrap (datasetid, unit, vertical partition);
```

'datasetid' must refer to an existing data set or a data set filter.
'unit' is a positive integer.
The third argument is optional and can be used to permute a subset of the 'filterid'

The specification of vertical partition is identical to that of 'CreateFilter'.

Example

```
N/A
```

BranchCount

Built-in function
2/4/2004

Returns the number of internal branches in the tree.

```
BranchCount(tree_ident);
```

'tree_ident' must be an existing tree or topology variable.

Example

```
N/A
```

BranchLength

Built-in function
7/9/2004

'BranchLength' returns the expected number of substitution along a given branch using the model and parameter value currently attached to the branch. Also can be used to compute path lengths between any two nodes in the tree.

```
BranchLength (treeID, branch index);
```

1. 'treeID':
– is an exisiting tree;
2. 'branch index':
– the integer index of the branch whose length we wish to retrieve, in the post-order traversal sequence.
– if branch index = -1, then a row matrix with branch lengths filled out in the post-order traversal order is returned.
– if branch index is a ';' separated list of two node names in the tree, returns the length of the path between the two nodes.

The value of 0 is returned for undefined cases, such as missing models (and no 'in-string' value). If no models are defined, but the tree was initialized with a string containing branch lengths - those will be returned.

Example

```
/* assuming that aTree includes nodes named
Leaf1 and Node23, the statement below will return the length of the path between these two nodes */
BranchLength (aTree,"Leaf1,Node23");
```

BranchName

Built-in function
3/3/2004

Returns a string containing the name of an internal node of the tree.

```
BranchName(tree_ident, node_index);
```

Valid index range is 0 to BranchCount (tree_ident)-1.

The name of the node is of the form 'Node1' rather than 'Tree.Node1'.

If node_index = -1, then a string row matrix with branch names filled out in the post-order traversal order is returned.

'tree_ident' must be an existing tree or topology variable.

Internal nodes not explicitly named during tree construction will be assigned default names of the form NodeX, where X is the rank of this node during a pre-order traversal of the tree.

Example

```
Tree tr = ((a,b)int1,c,d);
tName = BranchName (tr, 0);
/*Result: "int1".*/
Names = BranchName (tr, -1);
/*Result: {{"a","b","int1","c","d","Node0"}}*/
```

CACHE_SUBTREES

Constant Subtree caching. On by default.
2/4/2004

```
CACHE_SUBTREES= value;
```

1. value = 1:
– (default) HyPhy uses a caching scheme to speed up likelihood caclulations.
2. value = 0;
– caching turned off. A little less memory is but it is not worth the performance loss.

Example

```
N/A
```

CATEGORY_SIMULATION_METHOD

Constant Controls how category variables values are sampled.
2/4/2004

```
CATEGORY_SIMULATION_METHOD=1 or 2;
```

1. value = 1:
– use discretized distibutions to sample values from.
2. value = 2:
– (default) use the underlying continuous distribution whenever possible.

Example

```
N/A
```

CChi2

Built-in function The cumulative Chi squared distribution and fisher exact test.
2/3/2004

```
CChi2 (x,n)
```

1. x and n are positive numbers:
– returns the value of the Chi2 C.D.F (in [0,1]). CChi2(x,n) = P{t<=x}, where t is a random variable distributed as Chi-squared with n degrees of freedom, i.e. as the sum of squares of n independent standard normal distributions

2. x is a matrix and n is a number:
– returns the p-value of the Fisher exact (if n is 0) or a chi-2 approximation (if n is 1) for the contigency table specified by x. This test is computed using the Mehta-Patel network algorithm adapted from
http://portal.acm.org/citation.cfm?id=214326

Example

```
N/A
```

CGammaDist

Built-in function
2/3/2004

The cumulative gamma distribution.

```
CGammaDist(x,a,b)
```

1. x and b are positive numbers, a is not a negative integer:
– returns the value of the Gamma C.D.F (in [0,1]). CGammaDist(x,a,b) = P{t<=x}, where t has gamma distribution with the shape parameter a, and the ‘mean‘ parameter b.

Example

```
N/A
```

CLEAR_FILE

Constant
1/19/2000

Used as an argument to FPRINTF, forces the target file to be cleared.

```
fprintf (file_name,CLEAR_FILE,<more arguments>);
```

Use at the beginning of the file to clear a results file or something of the sort.

Example

```
N/A
```

COVARIANCE_PARAMETER

Constant
9/14/2005

Set this parameter to a string value to direct the function ‘CovarianceMatrix‘ to compute the confidence intervals with this parameter only, treating other parameter estimates as true values.

```
COVARIANCE_PARAMETER = "id"
```

Can either be:
1. A string
– only that parameter is profiled or has empirical Fisher information estimated
2. An associative array
– estimate profile intervals (coordinate wise) or empirical Fisher information for that subset of parameters

Example

```
/* assuming that "T_Scale" is a parameter in the likelihood function 'lf' */
COVARIANCE_PARAMETER = "T_Scale";
COVARIANCE_PRECISION = 0.95;
CovarianceMatrix (cmx, lf);
/* associative array approach; assumes T_Scale and kappa and omega are all model parameters in lf */
COVARIANCE_PARAMETER = {};
COVARIANCE_PARAMETER {"T_Scale"}=1;
COVARIANCE_PARAMETER {"kappa"} = 1;
COVARIANCE_PARAMETER {"omega"} = 1;
COVARIANCE_PRECISION = 0.95;
CovarianceMatrix (cmx, lf);
```

COVARIANCE_PRECISION

Constant
2/4/2004

A constant which determines what method is used by CovarianceMatrix in computing the numerical Hessian.

```
COVARIANCE_PRECISION = value;
```

1. value = 1
– (default) use a first order difference formula for partial derivatives (one function evaluation per matrix cell)
2. value = 2
– use a second order difference formula for partial derivatives (4 function evaluations per cell)
3. value between 0 and 1
– use a likelihood profile approach (each variable is done independently) and a chi$\hat{2}$ approximation with (1-value) serving as the p-value.
4. value is negative
– use a likelihood profile approach (each variable is done independently), with (-value) serving as the cutoff value for the likelihood difference.

Example

```
N/A
```

Category

Batch language command
2/4/2004

Used to define a category (rate variation) variable.

```
category id = (number of intervals, weights, method for representation, density, cumulative, left bound, right
bound, <mean cumulative function>,<hidden transition model>);
```

1. number of intervals:
– number of different categories;

2. weights:
– probability for each interval. Could be set to Equal, a single row matrix of weights, or a formula in terms of _n_, where _n_ runs from 0 to number of intervals:
3. method for representation:
– how to pick a representative for each category. Can be set to MEAN, MEDIAN or SCALED_MEDIAN. Only meaningful for variables coming from continuous distributions. SCALED_MEDIAN is the same as median, except the mean of the discrete approximation is scaled to match the mean of the continuos generator.
4. density or covariate:
– the formula for the generating continuos density function (in terms of _x_). Leave blank to specify a discrete distribution. Alternatively, you can provide the identifier of an existing discrete independent category variable to indicate that the present variable is defined conditiontally.
5. cumulative:
– the formula for the cumulative probability function (in terms of _x_). Leave blank if not known. Specify a row matrix of values for a discrete distribution here. For a conditionally specified variable, provide an MxN rate matrix, where row 'i' defines the values of current category variable in given that the covariate assumes the i-th values from its range.
6. left bound:
– left bound for _x_ in the continuous distribution
7. right bound:
– right bound for _x_ in the continuous distribution
8. optional mean cumulative function:
– if known, provide the formula for mean over interval (left bound,_x_) in terms of _x_
9. hidden transition model:
– (optional) provide the identifier of an existing model, to stipulate that the values of the category variable are spatially correlated via the Markov transition matrix given by that model. The number of states in the Hidden Markov model and the current category variable must be the same.

Example

```
See files in 'Examples'.
```

ChoiceList

Batch language command
2/4/2004

Used to present the user with a list of choices, and guides the user through the process of selecting one (or more) of them. If the user makes a valid selection, then the string value of the selection is stored in the variable SELECTION_STRINGS.

```
ChoiceList (result,"Title",selection length,skip flag,"option 1 name","option 1 description","option 2 name","option
2 description",...,"option N name","option N description"); or
ChoiceList (result,"Title",selection length,skip flag,object)
```

1. 'result':
– holds the choice of the user. If the user selected one option only (triggered by 'selection length flag'), 'result' holds either -1 (the selection was cancelled), or the 0 -based index of the selected option. If more than option were selected 'result' is a vector with indices of selections. If the [0] element of 'result' is -1 then the selection process was cancelled.
2. 'Title':
– is the title of the selection list
3. 'selection length':
– indicates how many choices the user must make. 0 means that the user is allowed to make an arbitrary (1 or more) number of choices
4. 'skip flag':
– either SKIP_NONE, so that all selections are available to the user or a vector of selection indices to skip
5. The list of selections afterwards is simply a collection of available choices.

You may place an object identifier instead of the explicit list of selections. It could be:

1). A dataset (the selections are taxa names)
2). A datasetfilter (the selections are taxa names).

3). If this argument is the literal "LikelihoodFunction" then the list of currently defines likelihood functions will be generated. 4). If this argument is the literal LAST_MODEL_PARAMETER_LIST, then the list of parameters of the model which was last declared will be generated.
5). An Nx2 string matrix containing pairs of choices.

This list will later include other objects.

Example

```
See TemplateBatchFiles.
```

ClearConstraints

Batch language command
2/4/2004

Used to remove constraints on variables.

```
ClearConstraints(var1,var2,...,varN);
```

All var's must reference existing variables.

Call this on category variables if you need to redefine them as 'regular' variables.

Example

```
ClearConstraints(tr.Node1.a,tr.Node1.b);
```

Columns

Built-in function
2/3/2004

The number of columns function.

```
Columns(m)
```

1. m is a matrix:
– returns the number of columns in m

Example

```
N/A
```

Combine

Batch language command
2/4/2004

A command to merge several data sets into one. If the first argument is 'purge', then all the data sets used in the merge operation will be deleted after the operation is finished. The resulting data set contains all the data from the component data sets as stacked columns, i.e. column 1 is a vertical stack of all columns 1 from the component data sets.

```
DataSet datasetid = Combine(purge,datasetid1,...,datasetidn);
```

All 'datasetid' must refer to existing data sets. If the number of sites varies from data set to data set, the missing sites will be padded with deletion symbols. 'purge' is optional.

Example

```
DataSet ds = Combine(data1,data2,data3);
DataSet ds = Combine(purge,data1,data2,data3);
```

Concatenate

Batch language command
2/4/2004

A command to merge several data sets into one. If the first argument is 'purge', then all the data sets used in the merge operation will be deleted after the operation is finished. The resulting data set contains all the data from the component data sets as concatenated rows, i.e. row 1 is the concatenated rows 1 of all component data sets.

```
DataSet datasetid = Concatenate(purge,datasetid1,...,datasetidn);
```

All 'datasetid' must refer to existing data sets. If the number of species varies from data set to data set, the missing rows will be padded with deletion symbols. 'purge' is optional.

Example

```
DataSet ds = Conacatenate(data1,data2,data3);
DataSet ds = Conacatenate(purge,data1,data2,data3);
```

ConstructCategoryMatrix

Batch language command
2/4/2004

Produces a matrix of category assignments for the sites of data analyzed with category variables.

```
ConstructCategoryMatrix(receptacle, likelihood_function, short_or_long);
```

'receptacle' is the matrix to store category assignments,

'likelihood_function' is a likelihood function identifier,

and short_or_long is a flag which indicates whether to produce a matrix which for each site will contain a row of conditional likelihoods for each category, or rather just the index of the maximally likely category.

The resulting matrix will contain a column for each site in the data (left to right).

If short_or_long is set to COMPLETE, the long matrix is produced. Any other value will generate a short matrix.

Example

```
ConstructCategoryMatrix(cm, lf, COMPLETE);
```

Cos

Built-in function
2/3/2004

The cosine function.

```
Cos(x)
```

1. x is a number:
– returns the cosine of angle x (defined in radians).

Example

```
N/A
```

CovarianceMatrix

Batch language command
2/4/2004

CovarianceMatrix computes the approximate covariance matrix for the MLE of a given likelihood function.

```
CovarianceMatrix (result_matrix, likelihood_function_name);
```

Use COVARIANCE_PRECISION to control how the Hessian is computed.

Returns (N is the number of independent variables in likelihood_function_name).:

1. If COVARIANCE_PRECISION is 1 or 2:
– returns an NxN covariance matrix
2. If COVARIANCE_PRECISION < 1:
– returns an Nx3 matrix, where each the first column contains left confidence interval bounds, the second column has parameter MLEs and the third column includes right confidence interval bounds.

Example

```
CovarianceMatrix (covMx, lf);
```

CreateFilter

Batch language command
4/6/2005

Used to select portions of data sets for inclusion in analyses. 'datasetid' is the identifier of the data set to filter. 'unit' is the size of the basic state for analyses in terms of the charchter length.

```
DataSetFilter dataSetFilterid = CreateFilter (datasetid,unit,vertical partition, horizontal partition, alphabet
exclusions);
```

Upon successful completion of this command, 4 variables are automatically created and set:

1. dataSetFilterid species:
– number of species in the data set,
2. dataSetFilterid.sites:
– number of sites(columns) in the data set,
3. dataSetFilterid.unique_sites:
– number of unique sites in the data set,
4. dataSetFilterid.site_freqs:
– the vector of dimension dataSetFilterid.unique_sites, which contains the count of sites of type i, in cell i.
4. dataSetFilterid.site_map:
– the vector of (0-based) indices of sites from the underlying dataset which have been included in the data filter, in the same order they are in the filter.

'datasetid' must refer to an existing data set or a data set filter 'unit' is a positive integer.

'Unit' defines how many consecutive characters are to be treated as a single state and must be a positive integer. E.g, to use codons in analysis of nucleotide data, specify, unit to be 3.

The last three arguments are optional.

Vertical partition designates the sites to be used in analysis. Leaving it blank: "" includes all the sites.

Acceptable partition formats (all indexing is 0-based)

1). Explicit lists
– "0,1,4,5,10", includes sites 1,2,5,6,11 in analysis.
2). Intervals
– "0-99", includes sites 1 through 100 in analysis.
3). Paired intervals
– "0-9&20-29" includes sites 1,21,2,22,..,10,30 in analysis.
4). Composite
– "0-5,7,10-15&20-25".
5). Boolean expression in terms of siteIndex.
– Includes all sites whose index renders the expression true.
6). Combs
– for example "<011>" will select every 2nd and 3rd element of each triplet, "<10101>" - every 1st, 3rd and 5th element of each quintuplet.

7). Regular Expressions
– only sites which match a regular expression will be selected; for instance "/(^[A-Z])\1+$/" will select all constant sites.

The foregoing is also true for horizontal partition, with the exception of item 5, where siteIndex, should be replaced with speciesIndex, and in 7, where regular expression matching is done on sequences. When both vertical and horizontal partitions are specified via regular expressions, the behavior is not well defined.

'Alphabet exclusions' is a comma separated list of state not to be allowed in analysis. Any site which contains excluded characters will be skipped (reported to messages.log)

Example

```
DataSetFilter dsf = CreateFilter (ds,3,"","","TAA,TGA,TAG");
DataSetFilter dsf = CreateFilter (ds,1,"0-100");
DataSetFilter dsf = CreateFilter (ds,1,(siteIndex%3)==0);
DataSetFilter dsf = CreateFilter (ds,1,"/[CG]/"); /* only those sites which have C and G */
```

DATAFILE_TREE

Constant
8/20/2002

Tree string from the most recently read data file.

```
DATAFILE_TREE = Newick tree string;
```

Check IS_TREE_PRESENT_IN_DATA to verify that there were any trees defined in the data file.

Example

```
N/A
```

DATA_FILE_DEFAULT_WIDTH

Constant
5/11/2000

Controls how dataset filters are printed by fprintf. Specifies the number of sites per line. Default is 50.

```
DATA_FILE_DEFAULT_WIDTH=value;
```

N/A

Example

```
N/A
```

DATA_FILE_PARTITION_MATRIX

Constant
2/4/2004

A matrix of partition names and specifications from the most recently read NEXUS data file.

```
DATA_FILE_PARTITION_MATRIX = 2xN string matrix;
```

DATA_FILE_PARTITION_MATRIX is an 2xN matrix of strings, where N is the number of partitions read from the file (check that the matrix has at least two rows - if not, then no partitions were read). The first row contains the identifier of the partition and the second row contains the partition string, in the same format as the one used by CreateFilter.

Example

```
N/A
```

DATA_FILE_PRINT_FORMAT

Constant
2/4/2004

Controls how dataset filters are printed by fprintf. Default is 0.

```
DATA_FILE_PRINT_FORMAT=value
```

1. value = 0:
– FASTA sequential;
2. value = 1:
– FASTA interleaved;
3. value = 2:
– PHYLIP sequential;
4. value = 3:
– PHYLIP interleaved.
5. value = 4:
– NEXUS sequential with sequence labels in the matrix;
6. value = 5:
– NEXUS interleaved with sequence labels in the matrix;
7. value = 6:
– NEXUS sequential without sequence labels in the matrix;
8. value = 7:
– NEXUS interleaved without sequence labels in the matrix;
9. value = 8;
– comma separated character data

Use ConvertDataFile Standard Analysis to change the format of a data file.

Example

```
N/A
```

DEFAULT_FILE_SAVE_NAME

Constant
9/16/2005

Sets the default (save) file name when HyPhy prompts the user for a destination file.

```
DEFAULT_FILE_SAVE_NAME = string;
```

N/A

Example

```
N/A
```

DIRECTORY_SEPARATOR

Constant
2/5/2004

Directory mark in full paths, appropriate for the platform.

```
N/A
```

:, / or depending on the platform.

Example

```
N/A
```

Differentiate

Batch language command
2/4/2004

Find analytical derivative of an expression.

```
Differentiate (receptacle, expression, variable, <times>);
```

1. ‘receptacle‘:
– the name of the variable to receive the derivative;
2. ‘expression‘:
– the expression to differentiate
3. ‘variable‘:
– is the identifier of the variable with respect to which the derivative should be taken
4. ‘times‘:
– an optional parameter of what order derivative should be taken. Default = 1

Supports scalar expressions with +,-,*,/,; Sin, Cos, Arctan, Log, Exp, Erf, Tan in them; correctly treats variable dependancies. Upon execution ‘receptacle‘ will become a dependent variable, set equal to the derivative expression.

HyPhy performs only trivial simplifications,
so the results of Differentiate may be grossly undersimplified.

Example

```
Differentiate (dfx, Exp(x^2),x);
GetString (derivativeStr, dfx, 0);
fprintf (stdout, " ", derivativeStr, " ");
```

DoSQL

Built-in function
2/5/2004

Open, access and close SQL data bases. Implemented via SQLite (www.sqlite.org).

```
DoSQL (arg1, arg2, arg3);
```

DoSQL supports 3 modes of calls:

1. Data Base Open:
– DoSQL (SQL_OPEN, file name, DB_ID): the file in 'file name' is opened (created if needed), and a new database handle is stored in 'DB_ID'.
2. Process an SQL command
– DoSQL (DB_ID, SQL_Command, Callback):
A. 'DB_ID' is a data base handle returned by a data base open command.
B. 'SQL_Command' is a string with SQL commands to execute.
C. 'callback' (optional) is the ID of a user callback function (with a dummy parameter). The callback will be executed once for every record/row matched and can expect to have two variables set:
** SQL_ROW_DATA: the data in the columns selected by the query,
** SQL_COLUMN_NAMES: the name of those columns.
3. Data Base Close:
– DoSQL (SQL_CLOSE, "", DB_ID): the DB referenced by DB_ID handle is closed. The second argument is ignored.

Example

```
N/A
```

END_OF_FILE

Constant
2/4/2004

Set to 0 or 1 by FSCANF to indicate whether the last read reached the end of file.

```
END_OF_FILE
```

Most useful in a loop, when data is read from the same file to check when to terminate.

Example

```
fileName = "list";
while (!END_OF_FILE)
{
fscanf (fileName,"String", str);
fprintf (stdout," ",str);
}
/* Simply reads the file line by line and echoes it to the console. */
```

Eigensystem

Built-in function
3/30/2004

Finds eigenvalues and eigenvectors of a square symmetric matrix

```
Eigensystem(mx)
```

'mx' must be a square symmetric matrix.

The return value is an associative list:
1). the first entry (index 0) is the vector of eigenvalues, sorted in decreasing order;
2). the second entry (index 1) is the matrix of eigenvectors (in columns), corresponding to their respective eigenvalues.

Example

```
/* finding the logarithm of a matrix
mx = Transpose(V) * Diag * V
Log (mx) = Transpose (V) * Log (Diag) * V
*/
mx = {{0.97,0.01,0.02}{0.01,0.90,0.09}{0.02,0.09,0.89}};
eigen = Eigensystem (mx);
evalues = eigen[0];
evectors = eigen[1];
logdiag = {Rows(mx), Rows(mx)};
for (k=Rows(mx)-1; k>=0; k=k-1)
{
logdiag [k][k] = Log(evalues[k]);
}
matrixlog = evectors*logdiag*Transpose(evectors);
fprintf (stdout, " Matrix: ", mx, "Log(Matrix): ", matrixlog, "Exp(Log(Matrix)): ", Exp (matrixlog)," ");
```

Erf

Built-in function
2/3/2004

The error function.

```
Erf(x)
```

1. x is a non-negative number:
– returns Erf(x)=2/Sqrt(Pi) Integral{0,x,Exp[-t2̂]dt}.

Example

```
N/A
```

ExecuteCommands

Batch language command
2/4/2004

Interpret and execute a sequence of HyPhy batch language commands.

```
ExecuteCommands (commands);
```

1. ‘commands‘:
–a string (either a literal or a string variable id), containing commands to execute.

Example

```
ExecuteCommands("fprintf (stdout, Ḧello world ÿ;");
```

Exp

Built-in function
2/3/2004

The exponential function.

```
Exp(x)
```

1. x is a number:
– returns eẋ.
2. x is a square matrix:
– returns Exp[x], implemented as a scaled Taylor series approximation.

Example

```
N/A
```

FORMAT

Data File formatting instruction
2/4/2004

FORMAT is used to indicate what data format a file is in.

```
$FORMAT:"value" (new line char)
```

This instruction has been deprecated, since HyPhy will autodetect file formats

Allowed values:
1. PHYLIPS - PHYLIP sequential;
2. PHYLIPI - PHYLIP interleaved.

Example

```
$FORMAT:"PHYLIPS"
```

FindRoot

Batch language command
2/4/2004

Find a root of an equation in the interval [left_bound, right_bound].

```
FindRoot (receptacle, expression, variable, left_bound, right_bound);
```

1. ‘receptacle‘:
– the name of the variable to receive the root. Upon execution ‘receptacle‘ is set to the root, or if no root could be found - to the value of left_bound.
2. ‘expression‘:
– the expression to define the equation with. The equation solved is ‘expression‘ = 0. It must be a differentiable scalar.
3. ‘variable‘:
– the identifier of the variable with respect to which the equation should be solved.

4. ‘left_bound‘, r‘ight_bound‘:
– define the interval where a root is located. ‘right_bound‘ must be greater than ‘left_bound‘.

If no (or multiple) roots exist in the interval, ‘receptacle‘ will be set to ‘left_bound‘.

Example

```
FindRoot (root, x^3-2 ,x, 0, 2);
fprintf (stdout, " The real cube root of 2 is:", root, " ");
```

Format

Built-in function
9/14/2005

Format is used to print a floating point number with a specified number of places of precision. If given a string argument, will parse and evaluate the formula contained in the string and format the result. Also controls output of tree strings.

```
Format (x,width,places);
```

The function returns a string containing the appropriate representation of the number.

1. ‘width‘:
– is the total length of the number string and should be a non-negative integer. Set ‘width‘ to 0 not to fix the total length
2 ‘places‘:
– is the number of decimal places to be printed and should be a non-negative integer. Set ‘places‘ to 0 to display the default number of decimal places.

Format (x,width,places) is equivalent to C sprintf (”%width.placesf”,x).
Format (x,0,0) is equivalent to C sprintf (”%g”,x).

If ‘x‘ is a tree or topology, then width and places are boolean (0,1) flags with ‘width‘ controlling whether or not to print default internal node names in the tree string and ‘places‘ determining whether or not to include branch lengths in the output string.

If ‘x‘ is a string, ‘x‘ is parsed as a formula, computed, and if the value is a scalar, that result is formatted.

Example

```
x = Arctan(1)*4;
testString = Format ("Arctan(1)*4", 10,10);
fprintf (stdout, " ", Format (x,5,2),
" ", Format (x,0,20),
" ", Format (x,20,0),
" ", Format (x,0,0),
" ", testString, " ");
```

GLOBAL_STARTING_POINT

Constant
8/20/2002

Specifies the default initial value for all independent variables during optimization. Default is 0.1.

```
GLOBAL_STARTING_POINT=value;
```

If distance estimates are being used, then the value of GLOBAL_STARTING_POINT is ignored. If GLOBAL_STARTING_POINT is out of bounds for a variable, the closer bound is used as initial value.

Example

```
N/A
```

Gamma

Built-in function
2/3/2004

The gamma function.

```
Gamma(a)
```

1. a is not a negative integer:
– returns Gamma[a] = Integral{0,Infinity, Exp[-x] x^(a-1)dx}.

Example

```
N/A
```

GammaDist

Built-in function
2/3/2004

The gamma distrufunction.

```
GammaDist(x,a,b)
```

1. x is a number, b is a positive number, a is not a negative integer:
– returns GammaDist[x,a,b] = bâ/Gamma(a) Exp[-bx] x^(a-1).

Example

```
N/A
```

GetDataInfo

Batch language command
7/23/2004

Extract various types of sequence information

```
GetDataInfo(matrix, dataFilterID,<sequence, site | sequence1 , sequence2,AMBIGS>)
```

1. 'matrix':
– upon return contains requested information;
2. 'dataFilterID':
– the data filter ID to extract the information from;
3. One of the 3 options:
I. Blank: returns site duplicate map - i.e. which unique pattern does each site correspond to.
II. Sequence, Site pair: returns a character translation of the state in that position, i.e. a vector with N states (N = number of distinct characters in the data, 4 for nucleotides etc). A '1' in the matrix means that the (sequence,site) character can be mapped to the corresponding basic character. Multiple 1 entries imply an ambiguities.
III. Sequence 1, Sequence 2, AMBIGS: returns an NxN matrix of pariwise differences between sequences (diagonal entries are actually matches). AMBIGS controls how ambiguities are handled (RESOLVE_AMBIGUITIES, AVERAGE_AMBIGUITIES, weighting by column frequencies. SKIP_AMBIGUITIES - ignore pairwise comparisons with ambiguities. Any other value - all resolutions are averaged and treated as equiprobable).

Example

```
N/A
```

GetInformation

Batch language command
10/1/2004

GetInformation is a utility command returns a matrix whose contents vary upon the parameter value.

```
GetInformation (receptacle, object);
```

1. 'receptacle':
– the matrix variable to receive the data

2. 'object':
– the object whose information is to be retrieved. Below is the list of object types and return values:
a. the name of a numeric variable:
** value of the variable, lower and upper bounds on the range as a 1x3 matrix;
b. the name of a category numeric variable:
** a two column matrix with a row per rate class, with column one containing the rates and column 2 containing the probabilities for the rates;
c. the name of a tree node:
** the rate matrix at that node (numeric). If the node has no associated rate matrix, the return value will be a 1x1 matrix whose value is not meaningful;
d. the name of a likelihood function:
** a string column vector containing the names of category variables the function depends on. The order is the same as

that used for building marginal likelihood matrices;
e. the name of a data set filter:
** a string column, where each string is a sequence, ordered the same way they are in the filter.
f. a "" enclosed string literal:
** finds all defined variable names which match the regular expression defined in the string.

If GetInformation was passed an invalid object a 0x0 matrix is returned.

Example

```
N/A
```

GetString

Batch language command
10/7/2004

GetString is a utility command which returns a string, whose meaning varies, given an object.

```
GetString (receptacle, object, index);
```

1. 'receptacle':
– is the string variable to receive the data;
2. 'object':
– is the object whose strings are to be retrieved;
3. 'index':
– is the index of the string to be retrieved.

Following is the list of acceptable objects and corresponding return values (use Rows command to get the total count of objects in C-G).

A. A data set or a data set filter:
– the function returns the name of the taxon at i-th position;
B. A dependent variable:
– GetString returns the formula of the dependance;
C. 'LikelihoodFunction' literal:
– identifier of 'index' likelihood function;
D. 'DataSet' literal:
– identifier of 'index' data set;
E. 'DataSetFilter' literal:
– identifier of 'index' data set filter;
F. 'Tree' literal:
– identifier of 'index' tree;
G. 'LAST_MODEL_PARAMETER_LIST' literal:
– identifier of 'index' parameter of the last defined model;
H. 'HYPHY_VERSION' literal:
– a version string for HyPhy. 'index' can be 0 (version number), 1 (full version), 2 (version/platform);
I. 'TIME_STAMP' literal:
– a time string. 'index' can be 1 (local time in astandard format) or 0 (yyyy/mm/dd hh:mm GMT time string);
J. A likelihood function identifier:
– GetString returns the name of 'index'
independent parameter.
K. 'UserFunction' literal:
– returns an associative array with "ID" key for the function identifier and "Arguments" key pointing to a string column matrix with ordered function arguments

Example

```
N/A
```

GetURL

Batch language command
2/3/2004

Retrieve a url to a string or to a file

```
GetURL (receptacle,URL<,SAVE_TO_FILE>);
```

1. The optional 3rd argument SAVE_TO_FILE is speciefied:
– the URL provided in the string argument URL is retrieved and saved to the file, whose name is given in the string argument receptacle.
2. 3rd argument is omitted:
– the URL provided in the string argument URL is retrieved and stored in the variable 'receptacle'.

Example

```
GetURL (thePage,"http://www.hyphy.org");
/* thePage now contains the HTML code of www.hyphy.org */
GetURL ("test.html","http://www.hyphy.org",SAVE_TO_FILE);
/* the HTML of http://www.hyphy.org is saved to file "test.html" */
```

HYPHY_BASE_DIRECTORY

Constant
2/5/2004

Absolute path to the HyPhy home directory.

```
N/A
```

Has a trailing directory symbol - DIRECTORY_SEPARATOR.

Example

```
N/A
```

HarvestFrequencies

Batch language command
2/4/2004

This function allows one to collect the frequencies of characters (states) in a data set (or filter).

```
HarvestFrequencies (receptacle,dataid,atom,unit,position_dependent_flag);
```

1. 'receptacle':
– is the matrix identifier where to the resulting frequencies table is stored.

2. ‘dataid‘:
– is either a data set or a data set filter.
3. ‘atom‘:
– determines the size of the data atom to count in terms of character length.
4. ‘unit‘:
– is the size (in terms of character length) of the block within each atom to be counted. The ultimate output will count frequencies of units.
5. ‘position_dependent_flag‘:
– determines whether units are to be counted accounting for (or not accounting) for their position within the ‘atom‘.

The resulting matrix will have dimension ‘unit states‘ x 1 (position_dependent_flag is false) or‘unit states‘ x (atom/unit) (position_dependent_flag is true), where ‘unit states‘ is the number of distinct possible units, ordered in terms of the ordering of characters in the data. (e.g. for nucleotides and atom = 3, codon ‘cat‘ will be in row 20.)

dataid must refer to an existing data set.
‘atom‘ and ‘unit‘ must be positive integers, and ‘atom‘ must be divisible by ‘unit‘. ‘position_dependent_flag‘ is either 0 (false) or 1 (true).

Example

```
HarvestFrequencies (freq, ds, 1,1,0);
HarvestFrequencies (freq, ds, 3,1,1);
```

IBeta

Built-in function
2/4/2004

The incomplete beta function.

```
IBeta(x,p,q)
```

1. ‘p‘ and ‘q‘ are non-negative numbers and x should is in [0,1]:
– returns IBeta[x,p,q] = Integral{0,x, Gamma[p+q]/Gamma[p]Gamma[q] t$\hat{}$(p-1) (1-t)$\hat{}$(q-1)};

Example

```
N/A
```

IGamma

Built-in function
2/3/2004

The incomplete gamma function.

```
IGamma(a,x)
```

1. a is not a negative integer, x is a positive number:
– returns IGamma[a,x] = Integral{0,x, Exp[-t] t$\hat{}$(a-1)dt}.

Example

```
N/A
```

INCLUDE_MODEL_SPECS

Constant
2/5/2004

Tree printing control. '0' by default.

N/A

If set to '1', trees and topologies produced by fprintf will include internal branch model specification constructs.

Example

```
m = {{*,a,b,a}{a,*,a,b}{b,a,*,a}{a,b,a,*}};
efv={{1}{1}{1}{1}}*0.25;
Model mm = (m,efv,1);
efv2={{0.4}{0.1}{0.2}{0.3}};
Model mm2 = (m,efv,1);

Tree t = ((1{mm},2{mm}),3,4);
fprintf (stdout, " Without models:",t," ");
INCLUDE_MODEL_SPECS=1;
fprintf (stdout, " With models:",t," ");
INCLUDE_MODEL_SPECS=0;
```

INTEGRATION_PRECISION_FACTOR

Constant
2/4/2004

Designates integration precision used internally for computing cumulative distributions of densities w/o known cumulative distributions formulas. Default is 0.0001.

```
INTEGRATION_PRECISION_FACTOR=value;
```

The actual method implemented for integration is C++ streamlined version of Romberg adaptive integration routines.

Example

```
N/A
```

INTERMEDIATE_PRECISION

Constant
2/4/2004

Designates intermediate precision for optimization methods that combine gradient ascent and bracketing. Not enabled in version after 0.99 - the optimizer automatically chooses such a switching point.

```
INTERMEDIATE_PRECISION=value;
```

INTERMEDIATE_PRECISION is interpreted as relative precision in the likelihood function values, and when it is reached, optimization routines switch from gradient ascent to bracketing. Default is precision/10 or precision when the max dimension of the model matrix is greater than 21.

Example

```
N/A
```

IS_TREE_PRESENT_IN_DATA

Constant
8/20/2002

Was there a tree defined in the last data file read?

```
IS_TREE_PRESENT_IN_DATA = 0 or 1;
```

If there was a tree in the data file, DATAFILE_TREE will be set to the tree string. For NEXUS files, which may define multiple trees, also check NEXUS_FILE_TREE_MATRIX.

Example

```
N/A
```

Integrate

Batch language command
2/5/2004

Numerically integrate an expression over the interval [left_bound, right_bound].

```
Integrate (receptacle, expression, variable, left_bound, right_bound);
```

1. ‘receptacle‘:
– the name of the variable to receive the integral.
2. ‘expression‘:
– the expression to integrate
3. ‘variable‘:
– the identifier of the variable with respect to which the expression should be integrated.

4. ‘left_bound‘, r‘ight_bound‘:
– define the interval over which the integral is evaluated. ‘right_bound‘ must be greater than ‘left_bound‘.

Example

```
Integrate (e,Exp(-x2/2) ,x, 0, 2);
fprintf (stdout, " The integral of Exp[-x2/2] over [0,1] is:", e, " ");
```

InvChi2

Built-in function
2/3/2004

The inverse Chi squared distribution.

```
InvChi2 (x,n)
```

1. x is a number in [0,1] and n is a non-negative number:
– returns y, such that P(t<=y)=x where t is a random variable distributed as Chi-squared with n degrees of freedom.

Example

```
N/A
```

Inverse

Built-in function
2/3/2004

The inverse function.

```
Inverse(x)
```

1. x is a square non-singular matrix:
– computes the matrix inverse of x
2. x is a string:
– flips the ordering of letters in string 'x'.

Example

```
Inverse({{2,1}{1,2}});
/* returns
{
{ 0.666666666667, -0.333333333333}
{ -0.333333333333, 0.666666666667}}
*/
Inverse("HyPhy")
/* returns yhPyH */
```

KEEP_OPTIMAL_ORDER

Constant
2/4/2004

When column optimization is active, this boolean constant allows to keep the optimal order between successive optimizations. Default is false.

```
KEEP_OPTIMAL_ORDER=0 or 1;
```

Meaningful when neither the data nor the tree are changed, for example when multiple models are fitted in succession to the same alignment/tree. Standard analyses set this option automatically when appropriate.

Example

```
N/A
```

LAST_FILE_PATH

Constant
2/4/2004

Set to the last file path returned by a PROMPT_FOR_FILE request.

```
N/A
```

This string variable can be used to retrieve the full path of the last file the user selected when prompted. The value is set after each successful use of to PROMPT_FOR_FILE. Should be saved in another string variable when working with multiple files.

Example

```
N/A
```

LFCompute

Batch language command
3/1/2004

Compute the likelihood function using current parameter values.

```
LFCompute (likelihood_function_id, receptacle);
```

1. 'receptacle':
– LF_START_COMPUTE - must be called before any computations are done to set up the internals,
– is the identifier of the variable which receives the result,
– LF_DONE_COMPUTE - must be called after all the computations are done to clean up the internals.
2. 'likelihood_funtion_id'
– the likelihood function to be evaluated.

Example

```
LFCompute (lf,LF_START_COMPUTE);
LFCompute (lf,res);
LFCompute (lf,LF_DONE_COMPUTE);
```

LF_CALL_COUNT

Constant
8/20/2002

Contains the number of likelihood function evaluations up to this point.

```
N/A
```

N/A

Example

```
N/A
```

LIKELIHOOD_FUNCTION_OUTPUT

Constant
2/4/2004

Controls how likelihood function is printed by fprintf. Default value is 2.

```
LIKELIHOOD_FUNCTION_OUTPUT=value;
```

1. value = 0:
– print just the likelihood function value;
2. value = 1:
– print the likelihood function value followed by a list of parameter values;
3. value = 2:
– 2(default) - print the likelihood function value followed by tree strings with only branch lengths measure in expected substitutions per position.
4. value = 3:
– print the list of parameters and constraints;
5 value = 4:
– print a list of parameters, with values and constraints as a HyPhy batch language statement. May be useful for saving and restoring parameter estimates;
6. value = 5:
– same as 4, except that all tree declarations are also included;
7. value = 6:
– same as (4) with randomized order of variables during bracketing.

Example

```
/* output samples */

/*0*/
-2667.76057913882

/*1*/
Likelihood Function's Current Value = -2667.76057913882
R=0.106575
clockTree.Chimpanzee.a=0.163059
clockTree.Node3.a=0.0544599
clockTree.Gibbon.a=0.420283
clockTree.Node6.a=0.186018
clockTree.Gorilla.a=clockTree.Chimpanzee.a+clockTree.Node3.a=0.217519
clockTree.Human.a=clockTree.Chimpanzee.a=0.163059
clockTree.Orangutan.a=clockTree.Gibbon.a=0.420283

/*2*/
Log Likelihood = -2667.76057913882;
Shared Parameters:
R=0.106575

Tree clockTree=(Gorilla:0.0618842,(Chimpanzee:0.0463904,Human:0.0463904)Node3:0.0154938,(Gibbon:0.119571,Orangutan:0.119571)Node6:0.052922);

/*3*/
Log Likelihood = -2667.76;
Independent Parameters List
Parameter 1 : R
Parameter 2 : clockTree.Chimpanzee.a
Parameter 3 : clockTree.Node3.a
Parameter 4 : clockTree.Gibbon.a
Parameter 5 : clockTree.Node6.a

Constrained Parameters List
Parameter 1 : clockTree.Gorilla.a=clockTree.Chimpanzee.a+clockTree.Node3.a
Parameter 2 : clockTree.Human.a=clockTree.Chimpanzee.a
Parameter 3 : clockTree.Orangutan.a=clockTree.Gibbon.a

/*4*/

global R=0.1065751962763258;
clockTree.Chimpanzee.a=0.1630593312591758;
clockTree.Node3.a=0.05445992293520954;
clockTree.Gibbon.a=0.4202829790973586;
clockTree.Node6.a=0.1860175069882983;
clockTree.Gorilla.a:=clockTree.Chimpanzee.a+clockTree.Node3.a;
clockTree.Human.a:=clockTree.Chimpanzee.a;
clockTree.Orangutan.a:=clockTree.Gibbon.a;

/*5 same as 4 plus */
Tree clockTree=(Gorilla:0.0618842,(Chimpanzee:0.0463904,Human:0.0463904)Node3:0.0154938,(Gibbon:0.119571,Orangutan:0.119571)Node6:0.052922);
```

LIST_ALL_VARIABLES

Constant
8/17/1999

Passed as an argument to fprintf produces a list of all currently defined variables.

```
fprintf(destination, LIST_ALL_VARIABLES);
```

N/A

Example

```
N/A
```

LUDecompose

Built-in function
2/3/2004

The LU matrix decomposition function.

```
LUDecompose(m)
```

1. m is a square matrix.
– Any non-degenerate square matrix can be ‘uniqely‘decomposed into the product of a lower triangular with an upper triangular matrices, where in addition the lower triangular matrix has ones on the diagonal.
The return value of this function is a Nx(N+1) matrix containing the LU decomposition and the last column keeps track of row interchanges used for pivoting during the decomposition. Used in conjunction with LUSolve.

Example

```
LUDecompose({{2,1}{1,2}});
/* returns
{{ 2, 1, 0}
{ 0.5, 1.5, 1}}.
In particular this means that */
{{1,0}{0.5,1}}*{{2,1}{0,1.5}}
/* yields the original matrix */
```

LUSolve

Built-in function
2/3/2004

Solves systems of the form LUx = b.

```
LUSolve(m,b)
```

1. m is is the return value of a call to LUDecompose and b is a vector
– Returns a column-vector containing the solution.

Example

```
/* To solve Ax = b, where A is {{2,1}{1,2}} and b = {{0}{1}} : */
A = {{2,1}{1,2}};
b = {{0}{1}};
m = LUDecompose(A);
x = LUSolve(m,b);
/* the solution is
{{ -0.333333333333}
{ 0.666666666667}}
*/
```

LikelihoodFunction

Batch language command The likelihood function construction operation.
2/4/2004

```
LikelihoodFunction id = (tree1,datafilterid,...,treen,datafilteridn,<optional evaluation template>);
```

Receives pairs (one or more) of existing data set filter and tree as arguments.

'id' must be a valid identifier.

The number of tips in the tree must be equal to the number of species in the data set filter, and, under some options, the names of tree leaves and sequence names must match.

'optional evaluation template' allows the user to alter how the likelihood is assembled from pieces (by default, the log likelihoods coming from each partition are added). Please note that some of the features and operations of the likelihood function may not yet be implemented for 'computational template' functions.

The evaluation template must be one of the three options:

I. A formula in terms of either SITE_LIKELIHOOD or BLOCK_LIKELIHOOD (and, possibly, user parameters). Depending on which variable the formula is defined in terms of, one of the two options is applicable.

1. Block-wise option:
– BLOCK_LIKELIHOOD is a matrix populated with the log likelihoods of each partition after each evaluation.
2. Site-wise option:
– SITE_LIKELIHOOD option is only applicable if every partition has the same number of sites. In this case, for every site, SITE_LIKELIHOOD matrix is populated with site likelihoods (not LOG-likelihoods) and formula is evaluated - it is assumed that the formula returns a LOG-likelihood. Lastly, the total likelihood is obtained by summing the results at every site.

II. An identifier of a hidden Markov category variable.

Only applicable if every partition has the same number of sites AND the number of states in the hidden Markov chain are the same as the number of partitions.

The likelihood is then computed by averaging over all possible realizations of the HMC, where the likelihood of every site in state 'i' of the HMC comes from the i-th partition.

III. A call to a single parameter user defined function, which is passed the matrix of conditional likelihoods at every site, and is expected to return a LOG-likelihood.

Example

```
LikelihoodFunction lf= (tr,df);
LikelihoodFunction lf2= (tr,df,tr2,df2);
```

Log

Built-in function
2/3/2004

The natural (base 'e) logarithm function.

```
Log(x)
```

1. x is a positive number.
– returns the natural log of x.
2. x is a matrix of positive numbers.
– returns a matrix of the same dimension where each entry is the natural log of the corresponding entry in x (note that this is NOT the matrix log function).

Example

```
N/A
```

MAXIMUM_ITERATIONS_PER_VARIABLE

Constant
2/4/2004

If the optimzation precision could not be reached in MAXIMUM_ITERATIONS_PER_VARIABLE*number of independent variables likelihood function evaluations, the routines return with the best value so far. The default value is determined using the formula 500+1000/(number of independent parameters).

```
MAXIMUM_ITERATIONS_PER_VARIABLE=value;
```

Don't change this value unless HyPhy optimization routines repeatedly terminate before optimization precision is met (as reflected in messages.log).

Example

```
N/A
```

MESSAGE_LOG

Constant
8/20/2002

The handle to HyPhy messages file which can be passed to fprintf.

```
N/A
```

N/A

Example

```
fprintf (MESSAGE_LOG,"Some message ");
```

MESSAGE_LOGGING

Constant
2/4/2004

Toggles writing messages to file ‘messages.log‘ on and off.

```
LMESSAGE_LOGGING=0 or 1;
```

N/A

Example

```
N/A
```

MPIReceive

Batch language command
2/5/2004

Receive data from an MPI node.

```
MPIReceive (allowed, sender, data)
```

1. ‘allowed‘:
– only accept messages from this node (set to -1 to accept all messages).
2. ‘sender‘:
– set to the index of the sender
3. ‘data‘:
– the resulting data (always a string) is stored in this ID.

This call will block until message is receieved.

Example

```
N/A
```

MPISend

Batch language command
2/5/2004

Send an MPI instruction to a node.

```
MPISend (node, message);
```

1. ‘node‘:
– index of an MPI node to send the message to. The call may block if the target node doesn‘t receive. ‘0‘ is the master node (running the batch file). ‘node‘ must be between 0 and MPI_NODE_COUNT-1.
2. ‘message‘:
– a string of batch commands to execute withnode ‘node‘ (to return results, use MPISend to calling node in that code)
OR
– the ID of an existing likelihood function. This function will be serialized and sent to node ‘node‘ for optimization. That node will return a string (serialized form) of the function along with parameter MLEs when it is done. Call ‘ExecuteCommands‘ on the return string to gain access to the function and it‘s parameter values.

Example

```
N/A
```

MPI_LAST_SENT_MSG

Constant
2/5/2004

Contents of last MPI message sent.

N/A

N/A

Example

```
N/A
```

MPI_NODE_COUNT

Constant
2/5/2004

Number of available MPI nodes (including self).

N/A

Only meaningful for MPI builds of HyPhy.
Set to 0 for other builds.

Example

```
N/A
```

MPI_NODE_ID

Constant
2/5/2004

ID of current node

N/A

Ranges from 0 to MPI_NODE_COUNT-1.
0 is the 'master' node, i.e. the node executing the batch file.

Only meaningful for MPI builds of HyPhy.

Example

```
N/A
```

Max

Built-in function
2/4/2004

Returns the maximum of its arguments.

```
Max(a,b)
```

a and b must be numbers.

Example

```
N/A
```

Min

Built-in function
2/4/2004

Returns the minimum of its arguments.

```
Min(a,b)
```

a and b must be numbers.

Example

```
N/A
```

Model

Batch language command
2/4/2004

'Model' is used to define an evolution model by specifying substitution rate matrix and equilibrium frequencies.

```
Model model_name = (rate_matrix_ident,freq_vector_ident,<mult_flag>);
```

1. 'rate_matrix':
– must be a square matrix. Diagonal elements are ignored if defined, and reset to ensure that the it is a proper rate matrix.
2. 'freq_vector_ident':
– Equilibrium frequencies matrix must be a column vector of the same dimension as the rate matrix. The frequency vector must be supplied, even when it is included in the structure of the rate matrix (as in F84).
3. 'mult_flag':

– Optional boolean mult_flag is used to indicate whether rate matrix elements should be automatically multiplied by the appropriate equilibrium frequency. Default is 1(true), and if the value of the flag is 0 (false), then the equilibrium frequencies should be integrated into the rate matrix explicitly.

Example

```
See files in the ‘Examples‘ directory.
```

MolecularClock

Batch language command
2/4/2004

Molecular clock is called to set additive length constraints on a tree from a node down.

```
MolecularClock (treeid,var1,var2,...,varn);
```

‘treeid‘ is either a valid tree identifier (in which case the command will set Molecular Clock on the entire tree), or a node identifier (will set Molecular Clock from that node down).

Following ‘treeid‘ is the list of variables to impose the constraint on.

Each node in the part of the tree which MolecularClock is being set on, must depend on the variables in the MolecularClock list.

HyPhy will write a list of generated constraints to messages.log

Example

```
m = {{*,a,b,a}{a,*,a,b}{b,a,*,a}{a,b,a,*}};
efv={{1}{1}{1}{1}}*0.25;
Model mm = (m,efv,1);
Tree tr = ((1,2)12,(3,4)34);
MolecularClock (tr,a);
MolecularClock (tr.12,b);
```

NEXUS_FILE_TREE_MATRIX

Constant
8/20/2002

A matrix of tree strings and identifiers from the most recently read NEXUS data file.

```
NEXUS_FILE_TREE_MATRIX = Nx2 string matrix;
```

NEXUS_FILE_TREE_MATRIX is an Nx2 matrix of strings, where N is the number of trees read from the file (check IS_TREE_PRESENT_IN_DATA to see if any trees were read at all). The first column contains the identifier of the tree, and the second column contains the actual tree string.

Example

```
N/A
```

NICETY_LEVEL

Constant
2/4/2004

Controls how cpu-time greedy and responsive the application is. Default is 1. Maximum(meaningful) value is 6.

```
NICETY_LEVEL=value;
```

Starting with version 0.99 and above, HyPhy automatically selects an appropriate value based on the complexity of an optimization.

This constant has no effect on

Example

```
N/A
```

NO_INTERNAL_LABELS

Constant
2/5/2004

Tree printing control. '0' by default.

```
N/A
```

If set to '1', trees and topologies produced by fprintf will NOT include internal branch names.

Example

```
Tree t = (((1,2)MyInt,3),4,5);
fprintf (stdout, " With labels:",t," ");
NO_INTERNAL_LABELS=1;
fprintf (stdout, " Without labels:",t," ");
NO_INTERNAL_LABELS=0;
```

OPTIMIZATION_METHOD

Constant
2/3/2004

Specifies the optimization method to utilize. Default method is 4 - it is strongly recommended that you do not use non-default methods unless you have reasons to believe that the default converge s to an local extemum.

```
OPTIMIZATION_METHOD = value;
```

1. value = 0:
– sequential single variable bracketing;
2. value = 1:
– simplex method (unreliable and slow);
3. value = 2:
– simulated annealing (currently disabled);
4. value = 3:
– bracketing along conjugate directions obtained without numerical gradients (Powell's method);
5 value = 4:
– bracketing along conjugate directions obtained with numerical gradients (the conjugate gradient method) followed by coordinate wise bracketing
6. value = 5:
– bracketing with randomized order of variables;
7. value = 6:
– same as (4) with randomized order of variables during bracketing.
8. value = 7:
– conjugate gradient (no coordinate wise bracketing).

Example

```
N/A
```

OPTIMIZATION_PRECISION

Constant
1/19/2000

Sets the precision goal for optimization routines. Default is 0.001.

```
OPTIMIZATION_PRECSION=value;
```

N/A

Example

```
N/A
```

OPTIMIZATION_PRECSION_METHOD

Constant
2/4/2004

Optimization precision target. 0 is the default value.

```
OPTIMIZATION_PRECISION_METHOD=value;
```

1. value is 0:
– (default) the optimization precision objective is applied to the likelihood function value;
2. value is 1:
– all the parameter estimates must meat precision goals.

Note that the second option may not be supported by all optimization methods and is not recommended.

Example

```
N/A
```

OPTIMIZE_SUMMATION_ORDER

Constant
2/4/2004

When true, pseudo optimal column ordering is used. Default is true.

```
OPTIMIZE_SUMMATION_ORDER=0 or 1;
```

HyPhy will reorder alignment columns in a way to speed up computations when appropriate. You may want to turn this option off only if you create likelihood functions that are never optimized (for simulation, export etc). Make sure to set the constant BEFORE ikelihood functions are created.

Example

```
N/A
```

OPTIMIZE_SUMMATION_ORDER_PARTITION

Constant
2/4/2004

Used when pseudo optimal column ordering is active. Specifies the subpartition size for column ordering.

```
OPTIMIZE_SUMMATION_ORDER_PARTITION=value;
```

The bigger the value of OPTIMIZE_SUMMATION_ORDER_PARTITION the better savings will be attained, however the time needed for initial column ordering depends quadratically on OPTIMIZE_SUMMATION_ORDER_PARTITION. Default is 1500. This value is sufficient for all but very long (>10000 sites) alignments, and even then should probably never be set about 3000.

Example

```
N/A
```

OpenDataPanel

Batch language command
2/4/2004

This function is used to save and restore states of data panel windows in HyPhy GUI. It is not recommended for direct invokation by the user.

```
OpenDataPanel (dataSetID,species order,display settings,partition settings,<optional likefunc ID>);
```

1. dataSetID:
– identifier of an existing dataset

2. ‘species order‘:
– a string in the same format used by CreateFilter, used to select what species from dataSetID should be included in the data panel. Leave empty to include all.
3. ‘display settings‘:
– a string, containing 7 comma separated numbers, which control additional features of data display. The first number is a flag with the following meaning:
** Flag has bit 1 turned on (i.e. (flag & 0x01) == 1): show consensus sequence
** Flag has bit 3 turned on (i.e. (flag & 0x04) == 1): show translated sequence (the index of the sequence to translate is parameter 3 of 7).
** Flag has bit 4 turned on (i.e. (flag & 0x08) == 1): show reference sequence (the index of the sequence to translate is parameter 2 of 7).
** The fourth number is unused.
** The fifth number selects the format for sequence names.
0: no names dispayed,
1: first 10 characters displayed,
2: full names displayed.
** The sixth number selects the format for repeated characters in a site.
0: display actual character,
1: display dots for repeats of the character in sequence 1.
** The seventh number selects the size of a character group.
9: every 9 sites are grouped (suggested for codon data).
10: every 10 sites are grouped (suggested for nucleotide and aminoacid data).
4.‘partition settings‘:
– a string containing semicolon separated settings for each partition. Each partition specification is a string with 5 comma separated entries.
** First entry is the name of the model attached to the partition. ”-1” indicates that no model is attached.
** Second entry selects model options for the partition. Bits 16-19 select parameter options, 0 for local, 1 for global, 2 for global with rate variation, 3 for model specified. Bits 20-23 choose equilibrium frequency options, 0 for partition, 1 for dataset, 2 for equal, 3 for estimate, 4 for model specified. Bits 24-31 specify the number of rate classes, if applicable. Bits 0-15 are not used.
** Third entry contains partition data settings. The only meaningful value there is the genetic code: bits 16-23 are the internal index of the genetic code table to use.
** Fourth entry is the RGB color for the partitions: bits 0-7 for blue, 8-15 for green and 16-23 for red.
** Fifth entry is the identifier of the tree to attach to the partition. ”No_tree” specifies that no tree be attached.
5. The last optional parameter:
– the identifier of the likelihood function to build based on the info provided in other parameters. If the parameter is not included, no LF is built.

Example

```
DATA_PANEL_SOURCE_PATH="::data:three.seq";
DataSet three2 = ReadDataFile (DATA_PANEL_SOURCE_PATH);
DataSetFilter beginning = CreateFilter (three2,3,"0-161,165-167","","AGA,AGG,TAA,TAG");
DataSetFilter end = CreateFilter (three2,1,"170-368","");
Tree three_tree2=(a,b,og);
Tree three_tree24=(a,b,og);
OpenDataPanel(three2,"","0,0,0,0,2,0,10", "MG94,65537,720896,14443523,three_tree2;F81,1,0,212,three_tree24", three2_LF);
```

OpenWindow

Batch language command
2/5/2004

This command opens windows of HyPhy GUI from batch language files.

```
OpenWindow (window_type, parameters <dimensions>);
```

1. ‘window_type‘:
– can be one of the following literals
** TREEWINDOW (a tree viewer)
** CHARTWINDOW (a chart window)
** DISTRIBUTIONWINDOW (a chart window with Bayesian posterior processing enhancements.

2. ‘parameters‘:
– a column vector of STRINGS which control the appearance and content of the window. The format is determined by ‘window_type‘ as follows.
A. For TREEWINDOW ‘parameters‘ can contain:
[0] - the ID of an existing tree variable, which will be plotted. Must be provided.
[1] - the flag which determines view options. If flag & 8192 > 0, then the tree is scaled as the window is resized. Can be left blank.
[2] - (deprecated) a comma separated string of window coordinates. Use the universal ‘dimensions‘ parameter instead.
[3] - the identifier of the variable to scale tree branches on. Can be left blank.
[4] - a comma separated list of full branch name lengths to be selected. Can be left blank.
B. For CHARTWINDOW ‘parameters‘ can contain:
[0] - title of the window. Must be provided.
[1] - column label matrix ID. It must be a ROW of strings with the same dimension as the count of columns. Optionally, in can have an additional entry, which is a ‘;‘ separated list of row caption and row titles. Must be provided.
[2] - data matrix ID. The matrix is internally evaluated if it consists of formulae. Must have the same number of columns as specified by the label matrix. Must be provided.
[3] - chart type. Must be one of the types supported by the chart window (examine the drop down menu in any chart window for types). Can be left blank.
[4] - x-axis variable. Can be either ‘Index‘ or one of the column names. Can be left blank.
[5] - semicolon separated list of data series. Can be left blank.
[6] - x-axis caption. Can be left blank.
[7] - y-axis caption (value axis or ‘z‘ in 3D plots). Can be left blank.
[8] - aux-axis caption (bottom value axis in contrast plots or ‘y‘ in 3D plots). Can be left blank.
[9] - display legend? 0 - no caption; 1-8 yes and position (top-left,top-mid,top-right,bot-left,bot-mid,bot-right,mid-left,mid-right). Can be left blank.
[10] - overplot graph formula, in terms of _x_ (and _y_ for 3D plots). Can be left blank.
[11] - semi-colon separated series to scale 3D plots x and y-axes on. Can be left blank.
[12] - 3D plot projection settings. Semi-colon separated real coordinated for viewpoint in spherical coordinates (R, xyAngle, zAngle). Can be left blank.
[13] - Semi-colon separated list of fonts. Each font spec must be a colon separates list of font face, font size and numberic font style (flag combo of 0-plain,1-bold, 2-italic). The fonts are for: ticks, axis captions, legend. Can be left blank.
[14] - Semi-colon separated list of series colors. Values are 32 bit RGB. Can be left blank.
[15] - Number of surface divisions in 3D overplots. Can be left blank.
C. For DISTRIBUTIONWINDOW ‘parameters‘ can contain all the same options as CHARTWINDOW plus:
[16] - Semicolon separated list of existing category variables, and derived quantities. Can be left blank.

3. ‘dimensions‘ (optional):
– semicolon separated position spec (in pixels): width, height, left, top.

Example

```
N/A
```

Optimize

Batch language command ‘Optimize‘ performs the optimization of the likelihood function.
2/5/2004

```
Optimize (receptacle, likelihood_function_id);
```

‘receptacle‘ is the identifier of the matrix which receives the results, and ‘likelihood_funtion_id‘ is the likelihood function to be optimized.

Upon completion ‘receptacle‘ will contain 2 rows:
- the first row lists all the parameters (global, followed by independent, followed by dependent - for order, call frpintf on the likelihood function, with LIKELIHOOD_FUNCTION_OUTPUT = 1)
- the second row only has 3 entries - the first element is the value of the likelihood function ([1][0]), the second element is the number of independent parameters that were optimized ([1][1]), the third is the number of global(shared) parameters

that were optimized([1][2]).

'likelihood_funtion_id' must refer to an existing likelihood function OR a user defined function which is to be optimized. In the latter case, the list of independent parameters will be gleaned from function parameter lists.

Example

```
Optimize (res, lf);
function userFunction (x,y)
{
return Exp(-(x-2)^2-(y-3)^2);
}
a = Random(0,10);
b = Random(0,10);
Optimize (res, userFunction (a,b));
fprintf (stdout, " ", res, " ");
```

PRINT_DIGITS

Constant
2/3/2004

Specifies the number of digits to print for floating point numbers. Default is 8, meaningful values are between 1 and 15.

```
PRINT_DIGITS=value;
```

A value less than 0 or greater that 15 results in "%g" behaviour, i.e. the default number of digits. This constant is deprecated - use the Format function instead.

Example

```
N/A
```

PSTreeString

Built-in function
2/5/2004

Returns a string containing a post script rendering of the tree. The length of the branches can be scaled on any model parameter (if it is defined for all branches), or left unscaled (all branches have equal length). The command also takes a page size parameter.

```
res = PSTreeString(tree_ident, parameter_name, page_size);
```

'parameter_name' must be either an empty string (branches have equal lengths) or a parameter name (e.g. "mu"), if that parameter is defined for every branch. The length of the branches can be scaled on any model parameter (if it is defined for all branches), or left unscaled (all branches have equal length).Two special values are also allowed:

1. "EXPECTED_NUMBER_OF_SUBSTITUTIONS":
– scales the tree on the expected number of substitution per site, assuming there is a model attached to every branch of the tree;
2. "STRING_SUPPLIED_LENGTHS":
– scales the tree on the lengths provided in the Newick string itself (assuming there are some).

The page size parameter is a 1x2 matrix which defines the width (first entry) and the height (second entry) of the page in points. 1 point = 1/72 of an inch. If page_size = {{0,0}}, the size of the page is set to 8x11 inches (US Letter size).

The output of the command is PostScript code which should be saved to a file and later rendered by a PS viewer program or sent to a PS printer.

Example

```
Tree primates = (Gorilla:0.0575801,(Chimpanzee:0.05376,Human:0.0413699)Node3:0.0174714,(Gibbon:0.13899,Orangutan:0.100161)Node6:0.0530553);
pString = PSTreeString(primates,"STRING_SUPPLIED_LENGTHS",{{200,200}});
fprintf (stdout,pString);
```

Permute

Batch language command
2/4/2004

Can be used to create permutations of blocks of columns of 'unit' length of existing data sets or data set filters.

```
DataSetFilter <filterid> = Permute (datasetid, unit, vertical partition);
```

'datasetid' must refer to an existing data set or a data set filter.
'unit' is a positive integer.
The third argument is optional and can be used to permute a subset of the 'filterid'

The specification of vertical partition is identical to that of 'CreateFilter'.

Example

```
N/A
```

RANDOM_SEED

Constant
2/4/2004

RANDOM_SEED is used to access the seed of pseudo-random number generating functions.

```
RANDOM_SEED
```

Do not use this constant to SET in versions starting with 0.99. Use SetParameter instead. It is OK to use RANDOM_SEED as a normal variable to GET the value of the random seed.

Example

```
N/A
```

RANDOM_STARTING_PERTURBATIONS

Constant
2/3/2004

Used in conjunction with GLOBAL_STARTING_POINT if small (10% in magnitude) random perturbations are desired. Default value is false (0).

```
RANDOM_STARTING_PERTURBATIONS = 0 or 1;
```

You may use this constant if optimization algorithms seem to converge to local extrema.

Example

```
N/A
```

RELATIVE_PRECISION

Constant
2/4/2004

When RELATIVE_PRECISION is true, the optimization routines use relative error to determine when to return. Default is false.

```
RELATIVE_PRECISION=0 or 1;
```

Doesn‘t apply to all optimization settings and is not recommended.

Example

```
N/A
```

REPEAT

Data File formatting instruction
2/4/2004

REPEAT denotes the repeat character in interleaved data files.

```
$REPEAT:"repeat char" (new line char)
```

A repeat character is used when the 2nd or later species has the same data at a site, as the first species. Note that ‘.‘is the repeat character by default.

Does not apply to NEXUS files (use MATCHCHAR instead)

Example

```
$REPEAT:"-"
$REPEAT:"" (do not use . as repeat)
```

REPLACE_TREE_STRUCTURE

Constant
2/5/2004

What to do when redefining trees. 0 by default.

```
N/A
```

1. '0':
– variables and values from the previous tree are kept;
2. '1':
– variables and values from the previous tree are replaced.

Example

```
N/A
```

Random

Built-in function
6/3/2004

Pseudorandom number generation function. Matrix permutation vector.

```
Random(a,b);
```

1. a and b are numbers, with b>a:
– Returns a (pseudo) random number from a uniform distribution over [a,b]. The numbers are produced via by a 'Mersenne twister' as implemented in http://www.math.keio.ac.jp/matumoto/emt.html. The period of this pseudo-random generator is 2̂19937-1.
2. a is a matrix. b is a number:
– returns a matrix resampled from 'a'. If b==0 the sampling is done with replacement, if b==1 - without (permutation).

Example

```
N/A
```

ReadDataFile

Batch language command
2/4/2004

Data file reading instruction.

```
DataSet datasetid = ReadDataFile(x);
```

The argument of the command designates the file to be read, and must be a string or one of the special values, listed below.

The resulting data set is stored in the receptacle 'datasetid'. Upon successful completion of this command, 'datasetid' contains the data, and 3 variables are automatically created and set:

1. datasetid.species
– number of species in the data set,
2. datasetid.sites
– number of sites(columns) in the data set,
3. datasetid.unique_sites
number of unique sites (patterns) in the data set.

If ‘x‘ is set to ‘PROMPT_FOR_FILE‘, the user will be asked to specify the data file name at the time of execution.

If ‘x‘ is set to ‘USE_NEXUS_FILE_DATA‘, (meaningful only within HYPHY NEXUS blocks), then the data from the NEXUS matrix will be used.

‘x‘ must be given relative to the batch file which contains the ReadDataFile statement (or as an absolute path)

‘datasetid ‘must be a valid identifier (alphanumeric sequence with possible underscores beginning with a letter or an underscore)

Sets IS_TREE_PRESENT_IN_DATA, DATAFILE_TREE, NEXUS_FILE_TREE_MATRIX (for NEXUS files), DATA_FILE_PARTITION_MATRIX.

Example

```
DataSet ds = ReadDataFile("../data/hiv.dat");
```

ReadFromString

Batch language command
9/15/2005

Construct a data file from a string.

```
DataSet datasetid = ReadFromString(x);
```

Effectively the same command as ReadDataFile, but the input is a string (not a file).

See comments for ReadDataFile

Example

```
dss = ">seq1 aaaaaaaa >seq2 cccccccc >seq3 gggggggg";
DataSet ds = ReadFromString(dss);
```

ReconstructAncestors

Batch language command
2/4/2004

Reconstructs the list of ancestral sequences based on current parameter values. The sequences are reconstructed by finding the states of internal tree nodes which maximize the likelihood of the tree.

```
DataSet datasetid = ReconstrucAncestors(likelihood_function_id);
```

1. ‘datasetid‘ is a valid identifier (alphanumeric sequence with possible underscores beginning with a letter or an underscore).
2. ‘likelihood_function_id‘ must reference an already defined likelihood function.

For likelihood functions with rate heterogeneity, the most probable (in the posterior Bayesian sense) is first chosen for every site.

Example

```
DataSet ds = ReconstructAncestors (likFunc);
```

ReplicateConstraint

Batch language command
2/6/2004

ReplicateConstraint exists to facilitate defining tree (or subtree) wide constraints. It allows to replicate a single constraint throughout the matching parts of a tree or several trees.

```
ReplicateConstraint ("constraint", argument_1, ..., argument_N);
```

‘constraint‘ is a quotation-enclosed HyPhy expression, with several special properies:

1. ”thisK” (K is a number starting at 1) will be replaced with argument_K. Therefore the number of arguments must match the number of different ”thisK” in the expression. Please note that thisK identifier should NOT include any underscores.
2. ‘¿ will be match any substring

‘argument_K‘ must be a tree or a tree node.

Example

```
See examples in ‘Examples/ConstraintDefinition‘.
```

RerootTree

Built-in function
2/4/2004

Rerooting command

```
RerootTree(tree_ident, branch_name);
RerootTree (string_ident, 0 - unused 2nd parameter);
```

1. Returns a string containing the Newick style string of the tree rerooted at the specified branch.

2. Can also be used to generate "balanced" tree strings, i.e. reroot a tree in such a way as to balance the number of nodes in each subtree. The latter functionality is useful for speeding up calculations on unrooted trees.

'branch_name' must be a valid tip or internal node name (a string), obtained by a call to 'TipName' or 'BranchName'. The call makes sense if the original tree was rooted.
"string_ident" must be a string containing a valid Newick tree

Example

```
Tree tr = ((a,b),c,d);
newTr = RerootTree (tr, "b");
unbalanced_tree = "(1,(2,(3,(4,(5,(6,7))))))";
rebalanced_tree = RerootTree (unbalanced_tree,0);
fprintf (stdout, " ", newTr, " ", rebalanced_tree," ");
```

Rows

Built-in function
10/7/2004

The number of rows function and object counter. Also returns a vector of keys in an associative array.

`Rows(m)`

1. m is a matrix:
– returns the number of rows in the matrix.
2. m is "LikelihoodFunction":
– returns the number of currently defined likelihood functions.
3. m is "DataSet":
– returns the number of currently defined dataset objects.
4. m is "DataSetFilter":
– returns the number of currently defined
5. m is "Trees:
– returns the number of currently defined trees.
6. m is "Variable":
– returns the number of currently defined variables.
7. m is "LAST_MODEL_PARAMETER_LIST":
– returns the number of parameters in the last defined model.
8. m is an associative array:
– returns a column vector of string keys into the array, in linear (unsorted) order.
9. m is "UserFunction":
– returns the number of user defined functions

Example

```
N/A
```

SCREEN_HEIGHT

Constant
2/4/2004

The height in pixels of the display.

`N/A`

Set in calls to OpenWindow and can‘t be accessed independently. Currently only available in MacOS and Windows

Example

```
N/A
```

SCREEN_WIDTH

Constant
2/4/2004

The width in pixels of the display.

```
N/A
```

Set in calls to OpenWindow and can‘t be accessed independently. Currently only available in MacOS and Windows

Example

```
N/A
```

SKIP_OMISSIONS

Constant
2/4/2004

When true, any data sites which contain any characters w/o information (deletions) are omitted from the dataset filters. Default is false.

```
SKIP_OMISSIONS=0 or 1;
```

Deletions and N-fold ambiguities are NOT the same. Deletions map to NO character, while N-fold ambiguities imply that there is a character there, but an unknown one.

Example

```
N/A
```

SelectTemplateModel

Batch language command
2/4/2004

Prompts the user to select a template model appropriate for the data contained in ‘filterid‘.

```
SelectTemplateModel(filterid);
```

‘filterid‘ must be an existing data set filter.

Example

```
SetDialogPrompt ("Please specify a data file:");
DataSet ds = ReadDataFile(PROMPT_FOR_FILE);
DataSetFilter filteredData = CreateFilter (ds,1);
SelectTemplateModel(filteredData);
```

SetDialogPrompt

Batch language command
8/17/1999

Used to set the prompt string presented to the user when PROMPT_FOR_STRING or PROMPT_FOR_FILE is used.

```
SetDialogPrompt("string");
```

Prompt string must be less than 255 characters.

Example

```
N/A
```

SetParameter

Batch language command
2/4/2004

SetParameter is used to assign a value to a component of an object

```
SetParameter (object, parameterIndex, newValue);
```

1. 'object' = RANDOM_SEED
– 'parameterIndex' = new value to assign the random seed to. (newValue is ignored - set it to 0).
2. 'object' is a likelihood function
– numeric value is 'newValue' is assigned to the independent parameter with index 'parameterIndex'.
3. 'object' is a dataset
– string value (must be a valid identifier) is 'newValue' becomes the name of the sequence with index 'parameterIndex'

Example

```
SetParameter (lf,index,someMatrix[index]);
```

Simulate

Batch language command
2/4/2004

Data file simulation without an existing likelihood function.

```
DataSet datasetid = Simulate(Tree, EF, Alphabet, Sites, <Ancestors, File Name>);
```

1. Tree:
– an existing tree variable with model definitions.
2. EF:
– the matrix of frequencies of characters at the root of the tree.
3. Alphabet:
– A matrix specifying the underlying data alphabet, in the following format:
** First row lists the characters in the same order they are used in the models
** Second row only has two first meaningful entries:
Characters per state (1 for nuc and a.a, 3 for codons)
Comma Separated list of excluded states (such as stop codons)
4. Sites:
– either a number of sites or a character string to assign to the root of the tree (that also determines the lenghts)
5. Ancestors (optional):
– Set to 1 to include ancestral sequences. Default 0.
6. File Name (optional):
– File name to write the data to - only works for simulations without rate categories. With this option, very little of the data is kept in memory.

Example

```
F81M = {{*,t,t,t}
{t,*,t,t}
{t,t,*,t}
{t,t,t,*}};
eqf = {{0.4,0.1,0.2,0.3}};
Model F81 = (F81M,eqf,1);
Tree T = (Chimpanzee:0.0561983,Human:0.0411321,(Gorilla:0.0630279,(Gibbon:0.132441,Orangutan:0.0979926)Node6:0.053019)Node4:0.0143057);
characters = {{"A","C","G","T"}
{"1","","",""}};

DataSet sim = Simulate (T,eqf,characters,10000,1,"testData");
```

SimulateDataSet

Batch language command
2/4/2004

Data file simulation from an existing likelihood function.

```
DataSet datasetid = SimulateDataSet(likelihood_function_id, "comma separated list of excluded states",simulated rates, category variable names);
```

The first argument of the call is an existing likelihood function. ‘Simulate‘ uses current values of parameters in the function to simulate a data set of the same size/type as the one references by the function. If the likelihood function references more than one data set, ‘SimulateDataSet‘ will work with the first data set reference, and all others which have the same number of species/type. The second optional parameter is the list of states which are not permissible as the result of simulation. Whatever states are included there will not be present in the simulated data. ‘datasetid‘ received the simulated data.

‘simulated_rates‘ (optional) - a matrix to receive simulated heterogeneous rates. The dimension of the matrix will be KxN, where K is the number of category variables that the likelihood function depends; N is the total number of data sites in the likelihood function. The entries in a column of the matrix will be the (randomly sampled) category variables that the site depends on.

‘category_variable_names‘ (optional) - a string matrix, of dimension Kx1. The entry in row ‘i‘ is the name of the i-th category variable.
Used in conjunction with ‘simulated_rates‘.

datasetid must be valid identifier (alphanumeric sequence with possible underscores beginning with a letter or an underscore).

likelihood_function_id must reference an already defined likelihood function.

Example

```
DataSet sds = SimulateDataSet(lf);
DataSet sds = SimulateDataSet(lf,"TAA,TGA,TAG");
```

Sin

Built-in function
2/3/2004

The sine function.

```
Sin(x)
```

1. x is a number:
– returns the sine of angle x (defined in radians).

Example

```
N/A
```

Sqrt

Built-in function
2/3/2004

The square root function.

```
Sin(x)
```

1. x is a non-negative number:
– returns the square root of x.

Example

```
N/A
```

TEXTreeString

Built-in function
2/4/2004

Returns a string containing a LaTEX picture description of the tree.

```
TEXTreeString(tree_ident, parameter_name);
```

'parameter_name' must be either an empty string (branches have equal lengths) or a parameter name (e.g. "mu"), if that parameter is defined for every branch. The length of the branches can be scaled on any model parameter (if it is defined for all branches), or left unscaled (all branches have equal length).Two special values are also allowed:
1. "EXPECTED_NUMBER_OF_SUBSTITUTIONS":
– scales the tree on the expected number of substitution per site, assuming there is a model attached to every branch of the tree;
2. "STRING_SUPPLIED_LENGTHS":
– scales the tree on the lengths provided in the Newick string itself (assuming there are some).

The output of the command is LaTEX code which should be saved to a file and later rendered by a LaTEX program.

Example

```
Tree primates = (Gorilla:0.0575801,(Chimpanzee:0.05376,Human:0.0413699)Node3:0.0174714,(Gibbon:0.13899,Orangutan:0.100161)Node6:0.0530553);
tex1 = TEXTreeString (primates,"");
tex2 = TEXTreeString (primates,"STRING_SUPPLIED_LENGTHS");
```

TOKEN

Data File formatting instruction
2/4/2004

TOKEN sets up translation relations between characters.

```
$TOKEN:"new char"=meaning (new line char)
```

Note: this is NOT case sensitive. A new character can be anything, whereas the meaning is defined in terms of base set chars.

Does not apply to NEXUS files (use EQUATE instead).

Example

```
$TOKEN:"B" = "CGT"
(B is either C, G or T)
```

Tan

Built-in function
2/3/2004

The tangent function.

```
Tan(x)
```

1. x is a number:
– returns the tangent of angle x (defined in radians).

Example

```
N/A
```

Time

Built-in function
2/3/2004

The timer function. Returns the value of the system timer in seconds.

```
Time(0 or 1)
```

1. The argument is 0:
– returns the number of seconds since a fixed past date (system dependent)
2. The argument is 1:
– returns system time used up by the HyPhy process since startup (reliable only on *NIX builds).

Example

```
/* To measure how long a sample operation takes */
timer = Time(0);
t = 1;
for (k=2; k<1000000; k=k+1)
{
t = t+Log (k);
}
timer = Time(0)-timer;
fprintf (stdout, " Time taken:", timer, " seconds ");
```

TipCount

Built-in function
2/4/2004

Returns the number of tips(leaves) in the tree.

```
TipCount(tree_ident);
```

'tree_ident' must be an existing tree or topology variable.

Example

```
N/A
```

TipName

Built-in function
2/4/2004

Returns a string containing the name of a tip of the tree.

```
TipName(tree_ident, tip_index);
```

Valid index range is 0 to TipCount (tree_ident)-1.

The name of the tip is of the form 'Tip1' rather than 'Tree.Tip1'.

'tree_ident' must be an existing tree or topology variable.

Example

```
Topology tr = ((a,b),c,d);
tName = TipName (tr, 2);
/* Result: "c" */
```

Topology

Batch language command
2/4/2004

Topology construction command. Topologies are lightweight trees, i.e. the tree structure with labels and associated parameters, but no associated model creation.

```
Topology treeid = X;
```

See notes for 'Tree' for details.

Example

```
N/A
```

Transpose

Built-in function
2/3/2004

The matrix transpose function.

```
Transpose(m)
```

1. m is a matrix:
– returns the transpose m.

Example

```
N/A
```

Tree

Batch language command
2/4/2004

Tree construction command.

```
Tree treeid = X;
```

X can be either the identifier of an existing topology variable or a tree string is a in a parenthetical form (Newick format), with the following extensions:

1. Internal nodes can be named by placing the name after ')' which closes the block of children of the internal node.
2. Each node may have its own model matrix. To attach a matrix to a node place '{matrix name}' after the name of the node.

By default, each node will be associated with the last defined matrix.

If tree string is set to 'PROMPT_FOR_STRING', the user will be prompted to enter the string.

Node names must be alphanumeric ('_' is allowed).
Rooted trees will be automatically 'unrooted', unless ACCEPT_ROOTED_TREES is set to 1 before the call.
One branch trees are allowed.

Example

```
Tree tr = ((1,2),3,4);
Tree tr = ((1{M},2{M})Internal{M},3,(4,5));
```

USE_DISTANCES

Constant
2/4/2004

When 1 (default) distance methods are used (when possible) to get intial parameter value.

```
USE_DISTANCES=0 or 1;
```

This option will only be applied to partitions with no more than 150 sequences. The actual procedure used is to use HKY85 derived distance measure (for nucleotide and codon data), and least squares fitting of branch lengths to pairwise distances.

Example

```
N/A
```

UseModel

Batch language command
2/4/2004

Used to select a model to be attached to branches of trees subsequently.

```
UseModel (model_name);
```

'model_name' must be a
1. valid model defined previously with the 'Model' command, or
2. USE_NO_MODEL - to reset the model used by default.

Example

```
See files in the 'Examples' directory.
```

ZCDF

Built-in function
2/4/2004

Cumulative distribution function of the standard normal random variable

```
ZCDF (x)
```

1. 'x' is a number:
– returns ZCDF(x) = P{N(0,1)<=x}, where N(0,1) is the standard normal.

Example

```
ZCDF(1.96)-ZCDF(-1.96)
/* returns 0.950004 */
```

[]

Operation
9/16/2005

Matrix or string element access operation. Also can be used to access subtrees.

```
Matrix_Ident[row][column];
String_Ident[position];
String_Ident[start_index][end_index];
Topology_ident[index];
Topology_ident["Node Name"];
```

1. For strings, the [] operation returns the character at a given position in the string (indexed from 0). [] can also be used to generate a substring of a given string.
2. In a matrix, can be used to access any given element, indexed by row and column position. Indexing starts at 0. For row or column vectors single index will suffice, i.e. v[0][2] is the same as v[2] if v is a row vector.
3. If row == -1 or column == -1 for a matrix argument, returns a respective row or column (see example)
4. If 'row' is a matrix of the same dimension as the matrix argument and 'column' is skipped, then uses 'row' as a boolean extraction template
5. If 'row' and 'column' are both 1x2 matrices, extracts a rectangular block of elements.
6. For matrices, if 'row' is a string, the string is parsed as a formula and executed for each entry in the matrix. Three predefined variables are populated with appropriate values for each matrix entry:
– _MATRIX_ELEMENT_VALUE_
– _MATRIX_ELEMENT_ROW_
– _MATRIX_ELEMENT_COLUMN_

Remember that all indexing is '0' based. The original matrix is left intact, and a new matrix of the same dimension as the original is returned, populated with the results of formula evaluations.
7. Due to an implementation bug (scheduled for a fix soon), nested indexing (e.g. m1[m2[1]]) is not properly parsed. Use intermediate variables instead.

Example

```
/* String examples */
test_string = "Hello cruel world";
test_substring = test_string[0][4];
/*contains "Hello"*/
See Examples/BatchLanguage/MatrixIndexing.bf for matrix access examples.
```

Raise

`^ - LaTex can't include this symbol in section titles`

Operation
2/3/2004

Operation of raise to power or regular expression search and replace in strings.

`x^y`

1. x and y are both numbers:
– raises x to power y, x must be positive or y must be an integer.
2. x is a string and y is a 2x1 matrix of strings
– searches for the regular expression defined in the first entry of y in x and replaces every found occurrence with the 2nd entry in y (sorry, no back substituitons for now). Regular expressions are supported by incorporation of a C++ adapted version of GNU POSIX compliant library http://www.gnu.org/directory/regex.html

Example

```
Sqrt(2)^Sqrt(2)
/*1.63253*/
(-1.5)^(-2)
/* 0.444444*/
(-1.5)^(-1.5)
/*error*/
"abcbDbeb2bg"^{"b[a-z]"}{"-"}
/*a-bD-b2-*/
```

break

Batch language command
2/4/2004

A loop termination command. When encountered within any for, while or do..while loop, causes the execution of the loop to stop, and transfers control to the first statement following the loop.

`break;`

A loop termination command. When encountered within any for, while or do..while loop, causes the execution of the loop to stop, and transfers control to the first statement following the loop.

Example

```
for (i=5;i<100;i=i+1)
{
if (Random(0,i)>0.9*i)
{
break;
}
else
{
fprintf (stdout, " ",i);
}
}
fprintf (stdout, " ");
```

continue

Batch language command
2/4/2004

A loop 'forward' command. When encountered within any for, while or do..while loop, causes the rest of the commands within the loop to be ignored and forces execution back to the beginning of the loop.

```
continue;
```

N/A

Example

```
for (i=5;i<100;i=i+1)
{
if (Random(0,i)>0.9*i)
{
continue;
}
fprintf (stdout, " ",i);
}
fprintf (stdout, " ");
```

do..while

Batch language command
2/4/2004

The 'do..while'loop. Cond is a logical expression. The loop is executed while Cond evaluates to non-zero.

```
do{..body..}while(Cond);
```

The brackets must be included in the statement.

'break' and 'continue' may be used within loops just as in 'C'.

Example

```
do
{
i = Random (0,105);
fprintf (stdout, " ", i);
}
while (i>=0&&i<=100);
```

for

Batch language command
2/4/2004

The general 'for' loop construct. Very much like the C 'for'. Init consists of one statement (typically assignment). Cond is a logical statement (compound in general). Incr is the statement executed at the end of each loop iteration.

```
for(Init;Cond;Incr){ ..body..}
```

Since '++' is not supported, statements like i=i+1 should be used in the Incr part.

Construcr for (..;..;..); is invalid, the {..body..} part must be present.

'break' and 'continue' may be used within loops just as in 'C'.

Example

```
mx = {61,61};
for (i=0; i<60; i=i+1)
{
for(j=i+1;j<61;j=j+1)
{
mx[i][j]:=a;
mx[j][i]:=b;
}
}
```

fprintf

Batch language command
2/4/2004

Output command. Can be used to print essentially any object to a file or to stdout (console).

```
frpintf(filename or stdout or MESSAGE_LOG, list of variables, expressions or string literals);
```

The first parameter is either a filename, given as a quote-enclosed path (absolute or relative to the file containing 'fprintf') or stdout (console) w/o the quotes;
MESSAGE_LOG can be supplied as the first argument to direct output to 'messages.log' file.

Printing to a file will append rather than rewrite. The variable length list of parameters that follows can contain object identifiers, expressions and quote-enclosed literals, as well as a special meaning constant CLEAR_FILE (which erases existing information in the file).

String literals may contain C-style '\code' control sequences, most notably: '$\backslash n$' for new line and '$\backslash t$' for tab.

Some objects only print to files due to large sizes.

'filename' could be a string variable.

Example

```
x = 12;
fprintf (stdout, x, " squared is ", x^2," ");
m = {{1,0}{0,1}};
fprintf (stdout, m);
```

fscanf

Batch language command
3/23/2004

Input command. Used to read values of numbers, matrices or trees from files or the console.

```
fscanf(filename or stdin, format string, list of variables expressions or string literals);
```

The first parameter is either a filename, given as a quote-enclosed path relative to the file containing ‘fprintf‘ or stdin (console) w/o the quotes. It could also be PROMPT_FOR_FILE if the user is to be prompted for the input file. Format string is a quote-enclosed comma separated list with 5 allowed terms:

1. Number:
– read a number;
2. Matrix :
– read a matrix;
3. Tree:
– read a tree;
4. String:
– read a string;
5. Raw:
– place the previously unread contents of the file into a string.
6 Raw:
– split the previously unread contents of the file into strings and store them in a row vector. A line is terminated by either , ; ôr ;

Fscanf will skip all the characters in between meaningful input (except in ‘Raw‘ mode)

Matrices must enclosed in ‘{..}‘. Trees are enclosed in ‘(..)‘.

‘filename‘ can be a string variable.

Sets END_OF_FILE.

Example

```
/* given string:
0.12; {{0,1}{1,0}} sjdiwj ((1,2),3,4)
*/
fscanf (stdin, "Number,Matrix,Tree",x,m,t);
/*will read number .12 into x
matrix {{0,1}{1,0}} into m
and tree ((1,2),3,4) into t*/
```

function

Batch language command
2/4/2004

User defined function declaration.

```
function id (param1,param2,...,paramN) {...body...}
```

param‘s are passed by value - i.e. changes to parameters within the function will not affect the variables passed as arguments.

To pass parameters by reference use ”param&” in function declaration and pass the name of the variable as a string.

Make sure the function has a ‘return‘ statement.

Example

```
function sqr (by_value)
{
by_value = by_value * by_value;
return by_value;
}

function sqr2(by_ref&)
{
by_ref = by_ref * by_ref;
return by_ref;
}
x = 2;
y = sqr (x);
fprintf (stdout, ," ", x, " ", y, " ");
/* 2 4, x didn‘t change*/
y = sqr 2("x");
fprintf (stdout, x, " ", y, " ");
/* 4 4, x did change*/
```

if..then..else

Batch language command
2/4/2004

Flow control statement.

`if (Cond){..body1..} else {..body2..}`

Cond is a logical expression. If cond evaluates to non-zero, body1 is executed, otherwise, if body2 is present, it is executed (the ‘else‘ clause is optional).

The brackets {} must be included in the statement, even for single statement bodies.

Example

```
i = Random (1,100)$1;
if (i%2)
{
fprintf (stdout, " Generated an odd number ");
}
else
{
fprintf (stdout, " Generated an even number ");
}
```

return

Batch language command
2/4/2004

The ‘return‘ statement for user functions.

`return object;`

‘return‘ must be followed by an object (a constant, variable or formula), except when returning from the ‘main‘ function, in which case the return value is optional.

Example

```
N/A
```

while

Batch language command
2/4/2004

The ‘while‘loop.

```
while(cond){..body..}
```

Cond is a logical expression. The loop is executed while Cond evaluates to non-zero.

The brackets {}must be included in the statement, even for single-statement bodies.

‘break‘ and ‘continue‘ may be used within loops just as in ‘C‘.

Example

```
i=0;
while (i>=0&&i<=100)
{
i = Random (0,105);
fprintf (stdout, " ", i);
}
```

{}

Batch language command
9/14/2005

Used to create empty associative arrays.

```
anArray = {};
```

An associative array (implemented as an AVL tree) is useful for storing (arbitrary elements) under a given key.

Any object which can be converted to a string may be used as a key.

Example

```
anArray = {};
anArray [1] = 3.14;
anArray ["A Matrix"] = {{1,2}};
anArray [{{1,2}}] = "A string";
fprintf (stdout, anArray);
```

||

Operation
2/3/2004

Operation of disjunction (logical OR) or regular expression matching

x||y

1. x and y are both numbers:
– if both arguments are 0 returns 0(false), otherwise returns 1 (true).
2. x and y are both strings:
– find all instances of the regular expression defined by y in x and return a column vector with 0-based match offsets. {{-1}{-1}} is returned when no matches were found. Regular expressions are supported by incorporation of a C++ adapted version of GNU POSIX compliant library http://www.gnu.org/directory/regex.html

Example

```
"aabaaacaaadaaaae"||"a+[â]";
/* returns: */
{
{ 0}
{ 2}
{ 3}
{ 6}
{ 7}
{ 10}
{ 11}
{ 15}}

/* 0-2 is "aab"
3-6 is "aaac"
7-10 is "aaad"
11-15 is "aaaae" */
```