GeneXproServer 4.3

Index

Index....2
GeneXproServer 4.3....3
Workflow....3
GeneXproServer Windows and Console Modes....5
The Output Files....6
Original Run File....6
Job Definition....7
Job Node....7
Run Node....8
Datasets Node....9
File Connection Node....9
Internal Connection Node....10
Database Connection Node....10
Settings Node....11
Functions Node....11
Preprocessing and Postprocessing Nodes....11
Test Node....12
Appendix A: List of Adjustable Settings....13
Appendix B: List of Mathematical Functions....15
Appendix C: List of Logical Functions....24
Appendix D: Installation (Add-on and Standalone)....32

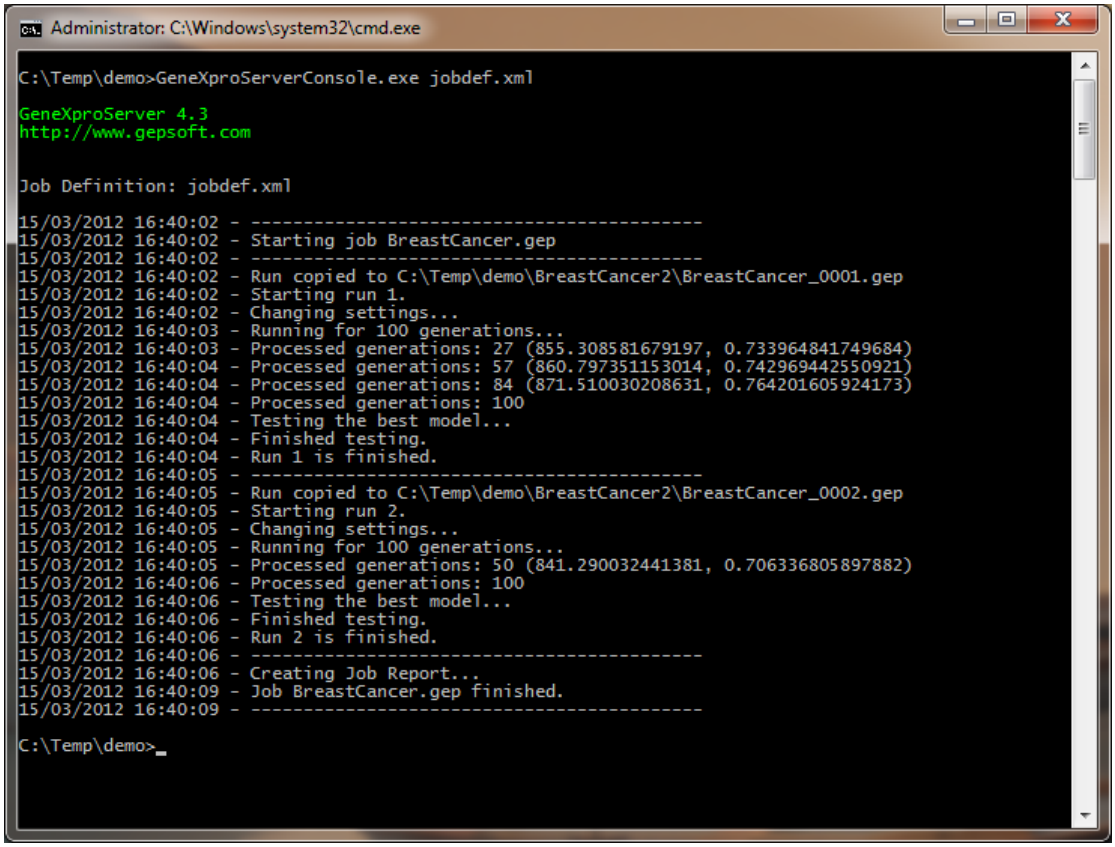
GeneXproServer 4.3

GeneXproServer is a batch processor of GeneXproTools runs that can easily be integrated with any infrastructure and process to enable the discovery and management of hundreds of models over different datasets and generated under different settings.

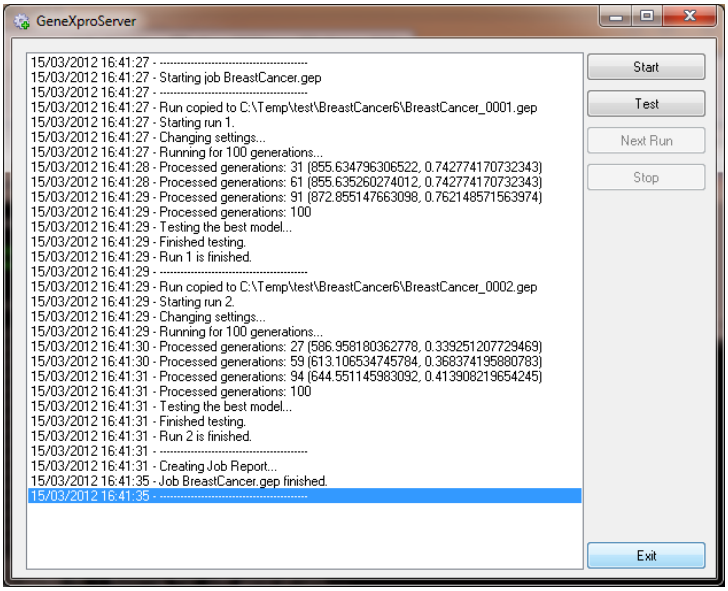
Workflow

The process starts with an initial run created in the client application (GeneXproTools 4.3) and a job definition file that can be created from scratch or from one of the several templates supplied. The job definition file must contain the initial run's name and details about the number of runs, the data to load and the settings to vary over the course of the job. Loading the job definition file into GeneXproServer starts the execution of the job. Each job contains one or more runs. Each run can last a finite number of generations or a number of minutes or hours.

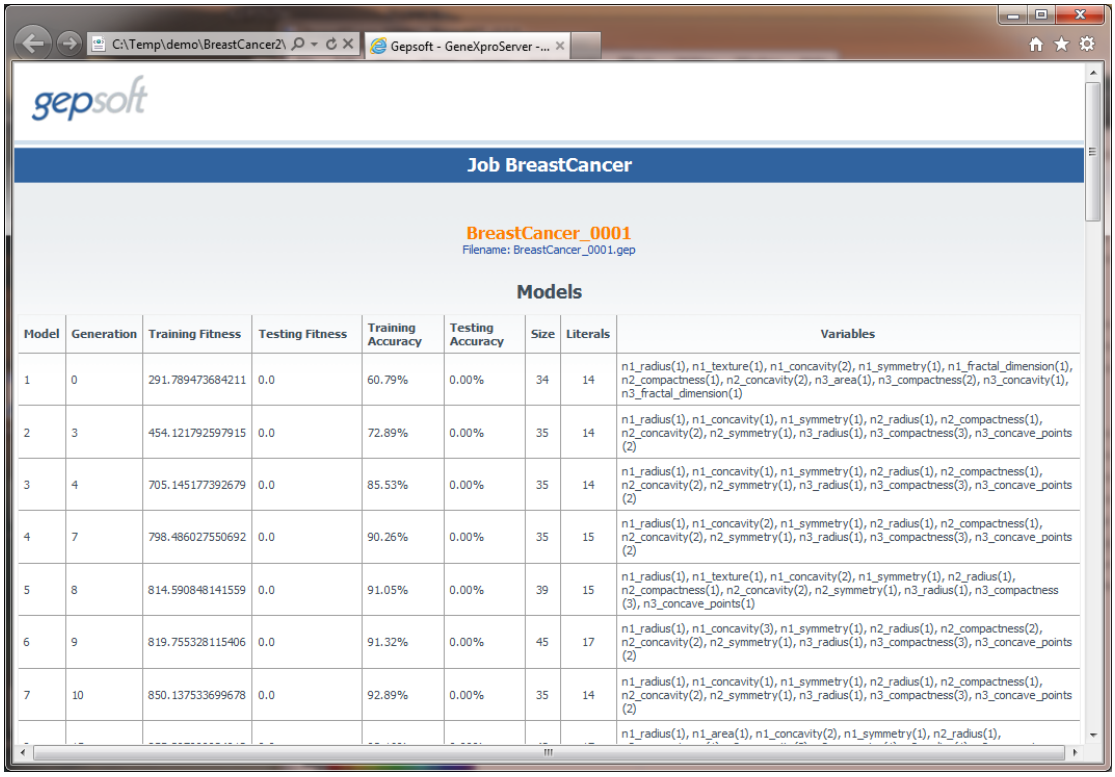
GeneXproServer is shipped in two different configurations: Windows and console mode. The former is appropriate for desktop computers and to create job definitions, whereas the console mode can be easily integrated with other systems and workflows. Either mode provides facilities to trigger external applications before and after each run and at the beginning and end of the job.



The Windows mode of GeneXproServer has a very simple interface with Start and Stop buttons, plus a Test button for checking the settings of all the runs in the job definition in order to prevent failure during execution and a Next Run button that might be useful for skipping a particular run.

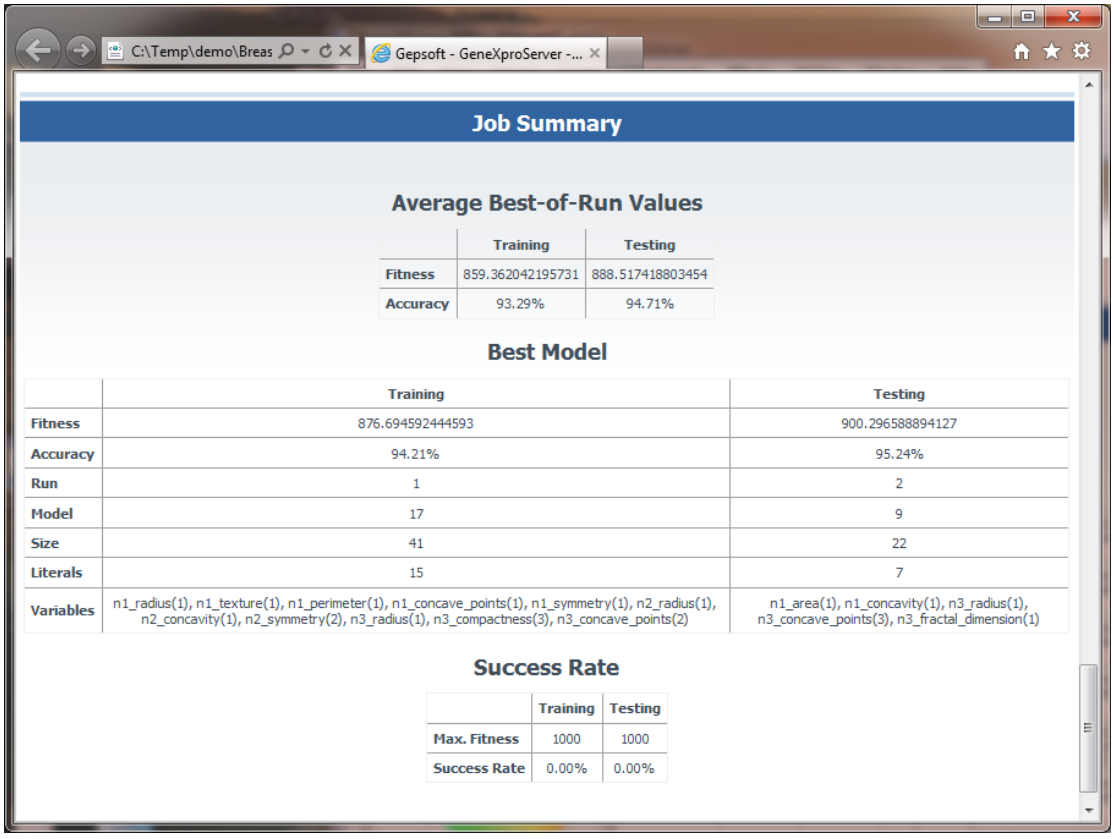


During a job execution GeneXproServer creates a run file for each run as well as a group of XML files with details about each run history and model performance including training and testing fitnesses, accuracies, program sizes, used variables, and so forth. These files can be processed and inserted into enterprise data stores for later analysis.



Model	Generation	Training Fitness	Testing Fitness	Training Accuracy	Testing Accuracy	Size	Literals	Variables
1	0	291.789473684211	0.0	60.79%	0.00%	34	14	n1_radius(1), n1_texture(1), n1_concavity(2), n1_symmetry(1), n1_fractal_dimension(1), n2_compactness(1), n2_concavity(2), n3_area(1), n3_compactness(2), n3_concavity(1), n3_fractal_dimension(1)
2	3	454.121792597915	0.0	72.89%	0.00%	35	14	n1_radius(1), n1_concavity(1), n1_symmetry(1), n2_radius(1), n2_compactness(1), n2_concavity(2), n2_symmetry(1), n3_radius(1), n3_compactness(3), n3_concave_points (2)
3	4	705.145177392679	0.0	85.53%	0.00%	35	14	n1_radius(1), n1_concavity(1), n1_symmetry(1), n2_radius(1), n2_compactness(1), n2_concavity(2), n2_symmetry(1), n3_radius(1), n3_compactness(3), n3_concave_points (2)
4	7	798.486027550692	0.0	90.26%	0.00%	35	15	n1_radius(1), n1_concavity(2), n1_symmetry(1), n2_radius(1), n2_compactness(1), n2_concavity(2), n2_symmetry(1), n3_radius(1), n3_compactness(3), n3_concave_points (2)
5	8	814.590848141559	0.0	91.05%	0.00%	39	15	n1_radius(1), n1_texture(1), n1_concavity(2), n1_symmetry(1), n2_radius(1), n2_compactness(1), n2_concavity(2), n2_symmetry(1), n3_radius(1), n3_compactness (3), n3_concave_points(1)
6	9	819.755328115406	0.0	91.32%	0.00%	45	17	n1_radius(1), n1_concavity(3), n1_symmetry(1), n2_radius(1), n2_compactness(2), n2_concavity(2), n2_symmetry(1), n3_radius(1), n3_compactness(3), n3_concave_points (2)
7	10	850.137533699678	0.0	92.89%	0.00%	35	14	n1_radius(1), n1_concavity(1), n1_symmetry(1), n2_radius(1), n2_compactness(1), n2_concavity(2), n2_symmetry(1), n3_radius(1), n3_compactness(3), n3_concave_points (2)
								n1_radius(1), n1_area(1), n1_concavity(2), n1_symmetry(1), n2_radius(1),

GeneXproServer also provides a simple report that shows all the information generated as well as a simple summary of the job with average best-of-run values, best model information, and success rate both for the training and testing sets.



Average best-of-run values both for the training and testing sets are evaluated using the fitness and accuracy/R-square values obtained for the best model (in terms of training fitness) of each run. The best-of-job model is identified in terms of the highest fitness value obtained both for the training and testing sets. The success rate corresponds to the percentage of processed runs in which max fitness was reached. This report is only compatible with Internet Explorer 8.0 and with Internet Explorer 9.0 in Compatibility Mode.

GeneXproServer Windows and Console Modes

In Windows mode it is possible to test a job definition, skip runs during a job or cancel a job at any time, whereas in console mode it is not possible to interrupt a job gracefully. Both modes share the same implementation and the differences in performance should not be noticeable but the console mode is more appropriate for very long runs as it consumes less resources. The console application takes a single parameter with the path to the job definition, for example:

```
C:\Progra..>GeneXproServerConsole.exe d:\jobs\jobdef.xml
```

The Output Files

GeneXproServer creates the following files for each run:

[run name]_xxxxx.gep: a copy of the original run file.
[run name]_xxxxx.xml: the output of the application during the run.
[run name]_xxxxx_definition.xml: the settings applied to this run.

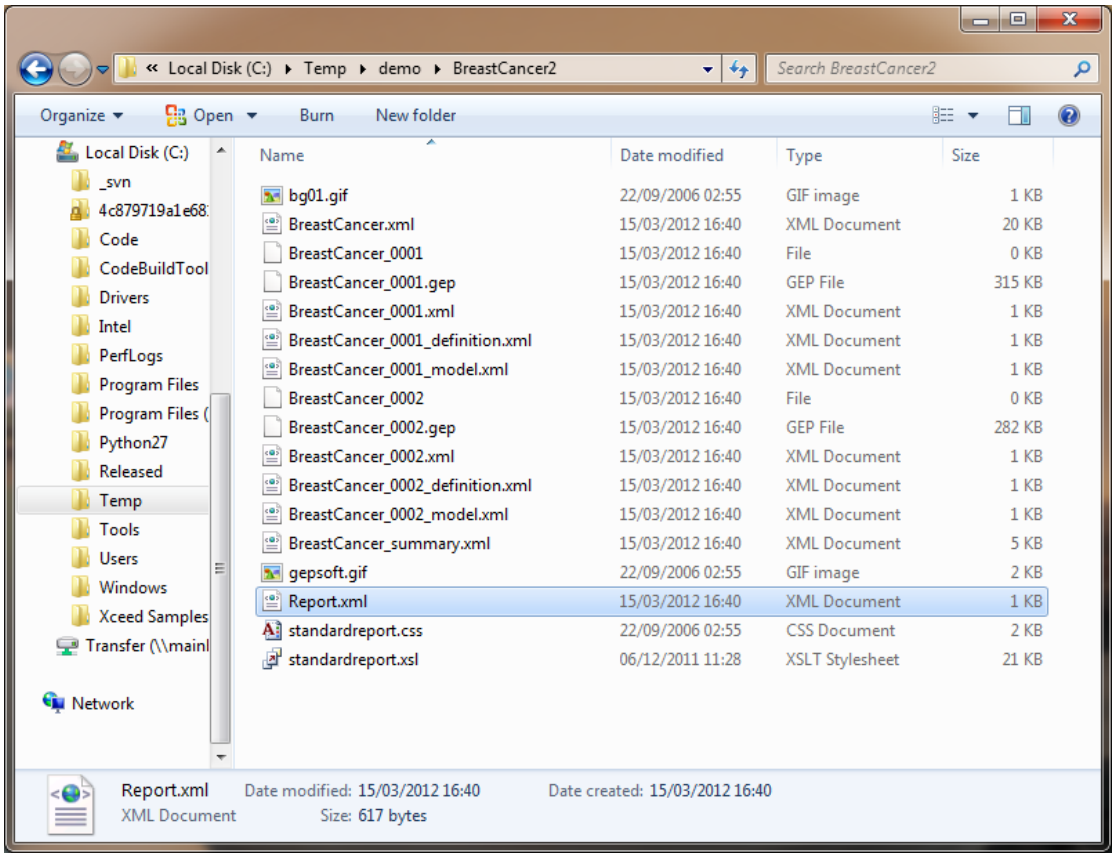
For the whole job GeneXproServer also generates the following files:

[job name]_summary.xml: cumulative values used for the run summary.
Report.xml: XML file that aggregates all the other XML files.

The following files are also copied to the output folder:

gepsoft.gif
bg01.gif
standardreport.css
standardreport.xsl

These are support files used for the transformation of the XML into HTML for the report.



Original Run File

The original run file is a gep file created with GeneXproTools. This run serves as a template for the whole job and determines the type of problem (Function Finding, Classification, Time Series Prediction, or Logic Synthesis) and the number of variables in the problem at hand.

Job Definition

The job definition is an XML file that controls all aspects of the job. GeneXproServer 1.0 job definition file supports the following operations:

1. Loading training and testing datasets
    a. From text files
    b. From any database type
    c. From the job definition itself (embedded data)
2. Changing many of the run settings (head size, number of genes, etc.)
3. Add functions and change function weights
4. Control which models are evaluated on the testing set
5. Launch external applications synchronously and asynchronously
6. Control the frequency of UI feedback
7. Setting the output folder for the job

A very simple job definition could be like this example:

```
<job filename="BreastCancer.gep" path="C:\work" feedback="500">
    <run id="1" stopcondition="generations" value="1000"/>
    <run id="2" stopcondition="generations" value="1000"/>
    <run id="3" stopcondition="generations" value="1000"/>
    <run id="4" stopcondition="generations" value="1000"/>
    <run id="5" stopcondition="generations" value="1000"/>
    <run id="6" stopcondition="generations" value="1000"/>
    <run id="7" stopcondition="generations" value="1000"/>
    <run id="8" stopcondition="generations" value="1000"/>
</job>
```

It executes eight different runs, each with 1000 generations and gives feedback each 500 milliseconds.

Job Node

The job node contains three attributes: `filename`, `path`, and `feedback`.

```
<job filename="BreastCancer.gep" path="C:\work" feedback="500">
```

The filename is the name of the original run file. The path is the folder where the input files are. GeneXproServer assumes that the folder already exists and that it contains the original run file. The feedback attribute is the value in milliseconds between screen updates during a run. If this attribute is set to zero GeneXproServer does not update the user interface during the run. All the output files will be created in a folder under the defined path. The name of the folder is derived from the filename attribute and it is consecutively numbered with each job processing (e.g. c:\work\BreastCancer1\, c:\work\BreastCancer2\, etc.).

Run Node

The run node contains a run identifier `id` that must be unique and preferably sequential within runs. The `stopcondition` can take three values: `generations`, `hours`, or `minutes`.

```
<run id="1" stopcondition="generations" value="500">
```

The `value` attribute denotes either generations or time and determines the effective duration of the run. If the value is set to -1 the run is not processed.

The run node contains three main sections and three control nodes. The sections are:

1. Datasets node
2. Settings node
3. Functions node

The control nodes are:

1. Preprocessing node
2. Postprocessing node
3. Test node

All the sections and control nodes are optional.

The following job definition example contains one run with all the supported node types:

```
<job filename="BreastCancer.gep" path="C:\work" feedback="500">
    <run id="1" stopcondition="generations" value="500">
       <preprocessing path="" hide="yes" synchro="yes"/>/>
        <datasets>
            <dataset type="training" samples="all">
                <connection type="file" format="normal">
                      <path separator="space" haslabels="no">C:\data\cancer_train.txt
                      </path>
                </connection>
            </dataset>
            <dataset type="testing" samples="5000">
                <connection type="database" format="normal">
                    <oledbconnectionstring>Provider=Microsoft.Jet.OLEDB.4.0;Data
                    Source=C:\data\datasets.mdb;User Id=admin;Password=;"
                    </oledbconnectionstring>
                    <sqlstatement>SELECT TOP 5000 * FROM Cancer_Test;</sqlstatement>
                </connection>
            </dataset>
        </datasets>
        <settings>
            <setting key="Genes" value="7"/>
        </settings>
        <functions>
            <function action="add" symbol="Pow" weight="2"/>
        </functions>
        <test whichmodels="all"/>
        <postprocessing path="" hide="yes" synchro="no"/>/>
    </run>
</job>
```

Datasets Node

The datasets node can contain nodes of type <dataset/> and is optional.

```
<dataset type="training" samples="all">

<dataset type="testing" samples="50000">

<dataset type="timeseries" samples="100">
```

The `type` attribute can assume the values `training`, `testing`, or `timeseries` and the `samples` attribute limits the number of data samples to be loaded. If this number is higher than what is available in the data source then this value is ignored. If this parameter is set to `all` then all the samples in the data source are loaded.

File Connection Node

Each dataset must contain a connection sub node that specifies the type of data source:

```
<connection type="file" format="standard">

<connection type="internal" format="standard">

<connection type="database" format="standard">
```

The `type` attribute specifies the type of data source, whereas the `format` specifies the data matrix format.

The file connections must contain a path and support three formats: `standard`, `gem`, or `timeseries`.

```
<connection type="file" format="standard">
      <path separator="space" haslabels="no">
            C:\data\cancer_train.txt
      </path>
</connection>
```

The `standard` format corresponds to the standard Samples x Variables format where samples are in rows and variables in columns, with the dependent variable or class occupying the rightmost position.

The `gem` format corresponds to the Gene Expression Matrix format commonly used in DNA microarrays studies where samples are in columns and variables in rows, with the dependent variable or class occupying the topmost position.

The `timeseries` format is obviously used in Time Series Prediction and consists of a single column of numeric values.

The path node must contain a `separator` attribute that supports the following values:

1. space
2. pipe
3. semicolon
4. comma
5. tab

It must also contain a node `haslabels` with `yes` or `no` values.

These attributes are not validated when the dataset is loaded so it is essential that the data is clean and that it matches all the attributes of the path node. If the data load fails the run cannot continue.

Internal Connection Node

The connection of type internal node is used to embed data in the job definition file itself:

```
<connection type="internal" format="standard">
    <data separator="space" haslabels="no">
        0.2 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1 0
        0.2 0.1 0.1 0.1 0.2 0.1 0.3 0.1 0.1 0
        0.5 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1 0
        0.5 0.4 0.6 0.8 0.4 0.1 0.8 1.0 0.1 1
        0.5 0.3 0.3 0.1 0.2 0.1 0.2 0.1 0.1 0
        0.2 0.3 0.1 0.1 0.3 0.1 0.1 0.1 0.1 0
        0.3 0.5 0.7 0.8 0.8 0.9 0.7 1.0 0.7 1
        1.0 0.5 0.6 1.0 0.6 1.0 0.7 0.7 1.0 1
        1.0 0.9 0.8 0.7 0.6 0.4 0.7 1.0 0.3 1
        0.4 0.1 0.1 0.1 0.2 0.1 0.3 0.1 0.1 0
    </data>
</connection>
```

This mode is very useful if you prefer not to have external dependencies but it can impact the run load time negatively if the data sets are very large. The dataset cannot include extraneous spaces or lines and it must respect the regional settings of the computer where it is processed regarding the decimal separator.

Database Connection Node

The database connection node is the most complex connection type and it supports any database with an OLEDB driver.

```
<connection type="database" format="standard">
   <oledbconnectionstring>Provider=Microsoft.Jet.OLEDB.4.0;Data
   Source=C:\data\datasets.mdb;User Id=admin;Password=;"
   </oledbconnectionstring>
   <sqlstatement>SELECT * FROM Cancer_Train;</sqlstatement>
</connection>
```

The node `oledbconnectionstring` must contain a valid connection string and the `sqlstatement` node must contain a valid SQL query that returns exactly the number of variables used by the rest of the run.

Settings Node

The settings node contains sub nodes that specify which settings change for the run. This node is optional and should only be present if you want to vary one or more settings from run to run:

```
<settings>
      <setting key="Genes" value="5"/>
      <setting key="HeadSize" value="10"/>
</settings>
```

Each setting must contain an attribute `key` and an attribute `value`. The keys and values correspond to the settings in GeneXproTools with the same name. These values are not validated so you must ensure that the correct type is used or the run file may become corrupted. The list of settings that can be changed is shown in Appendix A.

Functions Node

The functions node allows adding new functions and changing the weight of the functions that are part of the function set.

```
<functions>
    <function action="add" symbol="Pow" weight="2"/>
    <function action="set" symbol="+" weight="5"/>
</functions>
```

Each function must contain the following three attributes:

1. `action`: whether to add the function or change the weight
    a. `add`: adds the function
    b. `set`: changes the weight of the function
2. `symbol`: the representation of the function (see appendixes B and C for a list of supported symbols)
3. `weight`: the weight of the function in the function set

Preprocessing and Postprocessing Nodes

Pre- and post-processing nodes are used to launch external applications and scripts before and after a job or before and after each run. They can be a child of the job node or a child of any run node.

```
<preprocessing path="c:\script\notify.cmd" hide="yes" synchro="yes"/>
<postprocessing path="c:\script\email.cmd" hide="yes" synchro="no"/>
```

It must have three attributes: `path`, `hide`, and `synchro`.

1. `path` - the fully qualified path to the application
2. `hide` - does not show the application if its value is `yes`
3. `synchro` - when its value is `yes` it resumes processing only after the external application returns

Test Node

The test node specifies how many models of each run are evaluated on the testing set after the run finishes. By default the last model of each run is always checked against the testing set and therefore needs not be specified.

```
<test whichmodels="all"/>
<test whichmodels="2"/>
```

The attribute `whichmodels` can take the value `all` or a numeric value *n*. In the first case, all the models in the run history are evaluated against the testing set, whereas in the latter only the last *n* models are evaluated.

Appendix A: List of Adjustable Settings

Key	Data Type	Definition
TrainingSamples	int positive	Number of training samples
TestingSamples	int positive	Number of testing samples
Chromosomes	int positive [1, 4096]	Number of chromosomes
HeadSize	int positive [1, 256]	Head size
Genes	int positive [1, 64]	Number of genes
LinkingFunction	int positive	Linking function (see 1)
FitnessFunction	int positive	Fitness function (see 2 and 3)
WithParsimonyPressure	bit	"1" to apply and "0" to disable
RoundingThreshold	float	Rounding threshold
SelectionRange	positive float	Selection range
Precision	positive float	Precision
MutationRate	positive float [0, 1]	Mutation rate
InversionRate	positive float [0, 1]	Inversion rate
ISTranspRate	positive float [0, 1]	IS transposition rate
RISTranspRate	positive float [0, 1]	RIS transposition rate
GeneTranspRate	positive float [0, 1]	Gene transposition rate
OnePointRecRate	positive float [0, 1]	One-point recombination rate
TwoPointRecRate	positive float [0, 1]	Two-point recombination rate
GeneRecRate	positive float [0, 1]	Gene recombination rate
UseRNC	bit	"1" to use and "0" to disable
ConstantsPerGene	int positive [1, 100]	Number of constants per gene
DataTypeRNC	bit	"0" for integer and "1" for floating-point
LowerBoundRNC	float	RNCs lower bound
UpperBoundRNC	float	RNCs upper bound
MutationRateRNC	positive float [0, 1]	RNCs mutation rate
DcMutationRate	positive float [0, 1]	Dc mutation rate
DcInversionRate	positive float [0, 1]	Dc inversion rate
DcTranspRate	positive float [0, 1]	Dc transposition rate
SaveRunHistory	bit	"1" to save and "0" to disable

(1)

Linking Function	Value
Addition	0
Subtraction	1
Multiplication	2
Division	3
And	10001
Or	10002
Nand	10003
Nor	10004
Xor	10005
Nxor	10006
LT	10022
GT	10023
LOE	10024
GOE	10025

(2)

Function Finding, Logistic Regression & Time Series	Value
Relative with SR	0
Relative/Hits	1
Absolute with SR	2
Absolute/Hits	3
R-square	4
Positive Correl	26
Bounded R-square	29
Bounded Positive Correl	33
MSE	5
RMSE	6
MAE	7
RSE	8
RRSE	9
RAE	10
Enhanced MSE	28
Enhanced MAE	30
Enhanced RSE	31
Enhanced RAE	33
Custom	13

(3)

Classification	Value
Hits with Punishment	11
Cost/Gain Matrix	23
Dual Margin	24
Margin with Penalty	25
Entropy	17
Purity	20
Sensitivity/Specificity	12
PPV/NPV	14
SSPN	22
Positive Correl	15
Bounded Positive Correl	27
Enhanced MSE	16
Enhanced MAE	18
Enhanced RSE	19
Enhanced RAE	21
Custom	13

(4)

Logic Synthesis	Value
Hits with Punishment	34
Cost/Gain Matrix	35
Entropy	36
Purity	37
Accuracy	38
Squared Accuracy	39
Sensitivity/Specificity	40
PPV/NPV	41
SSPN	42
Positive Correl	43
Enhanced Correl	44
Enhanced MSE	45
Enhanced MAE	46
Enhanced RSE	47
Enhanced RAE	48
Custom	13

Appendix B: List of Mathematical Functions

Basic arithmetic functions:

- +: addition
- -: subtraction
- *: multiplication
- /: division
- Add3: addition with 3 inputs
- Sub3: subtraction with 3 inputs
- Mul3: multiplication with 3 inputs
- Div3: division with 3 inputs
- Add4: addition with 4 inputs
- Sub4: subtraction with 4 inputs
- Mul4: multiplication with 4 inputs
- Div4: division with 4 inputs

Logarithmic functions:

- Ln: natural logarithm
- Log: logarithm of base 10
- Log2: logarithm(x,y)

Exponential and power functions:

- Sqrt: square root
- Exp: exponential
- Pow: power
- Pow10: 10^x
- X2: x^2
- X3: x^3
- X4: x^4
- X5: x^5
- 3Rt: cube root
- 4Rt: quartic root
- 5Rt: quintic root
- Logi: logistic(x)
- Logi2: logistic(x,y)
- Logi3: logistic(x,y,z)
- Logi4: logistic(a,b,c,d)
- Gau: Gaussian(x)
- Gau2: Gaussian(x,y)
- Gau3: Gaussian(x,y,z)
- Gau4: Gaussian(a,b,c,d)

Rounding and other simple functions:

- Floor: floor
- Ceil: ceiling
- Mod: floating-point remainder
- Abs: absolute value
- Inv: inverse
- Neg: negation
- NOT: complement
- Nop: no operation

Minimum, maximum, and average functions:

- Min2: minimum of 2 inputs
- Min3: minimum of 3 inputs
- Min4: minimum of 4 inputs
- Max2: maximum of 2 inputs
- Max3: maximum of 3 inputs
- Max4: maximum of 4 inputs
- Avg2: average of 2 inputs
- Avg3: average of 3 inputs
- Avg4: average of 4 inputs

Constant functions:

- Zero: constant zero function
- One: constant one function
- Zero2: 0(*x*,*y*)
- One2: 1(*x*,*y*)
- Pi: number p
- E: Euler's number

Trigonometric functions (*x* is in radians):

- Sin: sine
- Cos: cosine
- Tan: tangent
- Csc: cosecant
- Sec: secant
- Cot: cotangent

Inverse trigonometric functions:

- Asin: arcsine
- Acos: arccosine
- Atan: arctangent
- Acsc: arccosecant
- Asec: arcsecant
- Acot: arccotangent

Hyperbolic functions:

- Sinh: hyperbolic sine
- Cosh: hyperbolic cosine
- Tanh: hyperbolic tangent
- Csch: hyperbolic cosecant
- Sech: hyperbolic secant
- Coth: hyperbolic cotangent

Inverse hyperbolic functions:

- Asinh: inverse hyperbolic sine
- Acosh: inverse hyperbolic cosine
- Atanh: inverse hyperbolic tangent
- Acsch: inverse hyperbolic cosecant
- Asech: inverse hyperbolic secant
- Acoth: inverse hyperbolic cotangent

Comparison 0/1 functions of two arguments (OR series):

- OR1: if *x* < 0 OR *y* < 0, then 1, else 0
- OR2: if *x* >= 0 OR *y* >= 0, then 1, else 0
- OR3: if *x* <= 0 OR *y* <= 0, then 1, else 0
- OR4: if *x* < 1 OR *y* < 1, then 1, else 0
- OR5: if *x* >= 1 OR *y* >= 1, then 1, else 0
- OR6: if *x* <= 1 OR *y* <= 1, then 1, else 0

Comparison 0/1 functions of two arguments (AND series):

- AND1: if *x* < 0 AND *y* < 0, then 1, else 0
- AND2: if *x* >= 0 AND *y* >= 0, then 1, else 0
- AND3: if *x* <= 0 AND *y* <= 0, then 1, else 0
- AND4: if *x* < 1 AND *y* < 1, then 1, else 0
- AND5: if *x* >= 1 AND *y* >= 1, then 1, else 0
- AND6: if *x* <= 1 AND *y* <= 1, then 1, else 0

Comparison IF THEN ELSE functions of two arguments (series A):

- LT2A: if *x* < *y*, then *x*, else *y*
- GT2A: if *x* > *y*, then *x*, else *y*
- LOE2A: if *x* <= *y*, then *x*, else *y*
- GOE2A: if *x* >= *y*, then *x*, else *y*
- ET2A: if *x* = *y*, then *x*, else *y*
- NET2A: if *x* != *y*, then *x*, else *y*

Comparison 0/1 IF THEN ELSE functions of two arguments (series B):

- LT2B: if *x* < *y*, then 1, else 0
- GT2B: if *x* > *y*, then 1, else 0
- LOE2B: if *x* <= *y*, then 1, else 0
- GOE2B: if *x* >= *y*, then 1, else 0
- ET2B: if *x* = *y*, then 1, else 0
- NET2B: if *x* != *y*, then 1, else 0

Comparison IF THEN ELSE functions of two arguments (series C):

- LT2C: if *x* < *y*, then (*x*+*y*), else (*x*-*y*)
- GT2C: if *x* > *y*, then (*x*+*y*), else (*x*-*y*)
- LOE2C: if *x* <= *y*, then (*x*+*y*), else (*x*-*y*)
- GOE2C: if *x* >= *y*, then (*x*+*y*), else (*x*-*y*)
- ET2C: if *x* = *y*, then (*x*+*y*), else (*x*-*y*)
- NET2C: if *x* != *y*, then (*x*+*y*), else (*x*-*y*)

Comparison IF THEN ELSE functions of two arguments (series D):

- LT2D: if *x* < *y*, then (*x***y*), else (*x*/*y*)
- GT2D: if *x* > *y*, then (*x***y*), else (*x*/*y*)
- LOE2D: if *x* <= *y*, then (*x***y*), else (*x*/*y*)
- GOE2D: if *x* >= *y*, then (*x***y*), else (*x*/*y*)
- ET2D: if *x* = *y*, then (*x***y*), else (*x*/*y*)
- NET2D: if *x* != *y*, then (*x***y*), else (*x*/*y*)

Comparison IF THEN ELSE functions of two arguments (series E):

- LT2E: if *x* < *y*, then (*x*+*y*), else (*x***y*)
- GT2E: if *x* > *y*, then (*x*+*y*), else (*x***y*)
- LOE2E: if *x* <= *y*, then (*x*+*y*), else (*x***y*)
- GOE2E: if *x* >= *y*, then (*x*+*y*), else (*x***y*)
- ET2E: if *x* = *y*, then (*x*+*y*), else (*x***y*)
- NET2E: if *x* != *y*, then (*x*+*y*), else (*x***y*)

Comparison IF THEN ELSE functions of two arguments (series F):

- LT2F: if *x* < *y*, then (*x*+*y*), else sin(*x***y*)
- GT2F: if *x* > *y*, then (*x*+*y*), else sin(*x***y*)
- LOE2F: if *x* <= *y*, then (*x*+*y*), else sin(*x***y*)
- GOE2F: if *x* >= *y*, then (*x*+*y*), else sin(*x***y*)
- ET2F: if *x* = *y*, then (*x*+*y*), else sin(*x***y*)
- NET2F: if *x* != *y*, then (*x*+*y*), else sin(*x***y*)

Comparison IF THEN ELSE functions of two arguments (series G):

- LT2G: if *x* < *y*, then (*x*+*y*), else atan(*x***y*)
- GT2G: if *x* > *y*, then (*x*+*y*), else atan(*x***y*)
- LOE2G: if *x* <= *y*, then (*x*+*y*), else atan(*x***y*)
- GOE2G: if *x* >= *y*, then (*x*+*y*), else atan(*x***y*)
- ET2G: if *x* = *y*, then (*x*+*y*), else atan(*x***y*)
- NET2G: if *x* != *y*, then (*x*+*y*), else atan(*x***y*)

Comparison IF THEN ELSE functions of three arguments (series A):

- LT3A: if *x* < 0, then *y*, else *z*
- GT3A: if *x* > 0, then *y*, else *z*
- LOE3A: if *x* <= 0, then *y*, else *z*
- GOE3A: if *x* >= 0, then *y*, else *z*
- ET3A: if *x* = 0, then *y*, else *z*
- NET3A: if *x* != 0, then *y*, else *z*

Comparison IF THEN ELSE functions of three arguments (series B):

- LT3B: if (*x*+*y*) < *z*, then (*x*+*y*), else *z*
- GT3B: if (*x*+*y*) > *z*, then (*x*+*y*), else *z*
- LOE3B: if (*x*+*y*) <= *z*, then (*x*+*y*), else *z*
- GOE3B: if (*x*+*y*) >= *z*, then (*x*+*y*), else *z*
- ET3B: if (*x*+*y*) = *z*, then (*x*+*y*), else *z*
- NET3B: if (*x*+*y*) != *z*, then (*x*+*y*), else *z*

Comparison IF THEN ELSE functions of three arguments (series C):

- LT3C: if (*x*+*y*) < *z*, then (*x*+*y*), else (*x*+*z*)
- GT3C: if (*x*+*y*) > *z*, then (*x*+*y*), else (*x*+*z*)
- LOE3C: if (*x*+*y*) <= *z*, then (*x*+*y*), else (*x*+*z*)
- GOE3C: if (*x*+*y*) >= *z*, then (*x*+*y*), else (*x*+*z*)
- ET3C: if (*x*+*y*) = *z*, then (*x*+*y*), else (*x*+*z*)
- NET3C: if (*x*+*y*) != *z*, then (*x*+*y*), else (*x*+*z*)

Comparison IF THEN ELSE functions of three arguments (series D):

- LT3D: if (*x*+*y*) < *z*, then (*x*+*y*), else (*x*-*z*)
- GT3D: if (*x*+*y*) > *z*, then (*x*+*y*), else (*x*-*z*)
- LOE3D: if (*x*+*y*) <= *z*, then (*x*+*y*), else (*x*-*z*)
- GOE3D: if (*x*+*y*) >= *z*, then (*x*+*y*), else (*x*-*z*)
- ET3D: if (*x*+*y*) = *z*, then (*x*+*y*), else (*x*-*z*)
- NET3D: if (*x*+*y*) != *z*, then (*x*+*y*), else (*x*-*z*)

Comparison IF THEN ELSE functions of three arguments (series E):

- LT3E: if (*x*+*y*) < *z*, then (*x*+*y*), else (*x**z*)
- GT3E: if (*x*+*y*) > *z*, then (*x*+*y*), else (*x**z*)
- LOE3E: if (*x*+*y*) <= *z*, then (*x*+*y*), else (*x**z*)
- GOE3E: if (*x*+*y*) >= *z*, then (*x*+*y*), else (*x**z*)
- ET3E: if (*x*+*y*) = *z*, then (*x*+*y*), else (*x**z*)
- NET3E: if (*x*+*y*) != *z*, then (*x*+*y*), else (*x**z*)

Comparison IF THEN ELSE functions of three arguments (series F):

- LT3F: if (*x*+*y*) < *z*, then (*x*+*y*), else (*x*/*z*)
- GT3F: if (*x*+*y*) > *z*, then (*x*+*y*), else (*x*/*z*)
- LOE3F: if (*x*+*y*) <= *z*, then (*x*+*y*), else (*x*/*z*)
- GOE3F: if (*x*+*y*) >= *z*, then (*x*+*y*), else (*x*/*z*)
- ET3F: if (*x*+*y*) = *z*, then (*x*+*y*), else (*x*/*z*)
- NET3F: if (*x*+*y*) != *z*, then (*x*+*y*), else (*x*/*z*)

Comparison IF THEN ELSE functions of three arguments (series G):

- LT3G: if (*x*+*y*) < *z*, then (*x**y*), else (*x*+*z*)
- GT3G: if (*x*+*y*) > *z*, then (*x**y*), else (*x*+*z*)
- LOE3G: if (*x*+*y*) <= *z*, then (*x**y*), else (*x*+*z*)
- GOE3G: if (*x*+*y*) >= *z*, then (*x**y*), else (*x*+*z*)
- ET3G: if (*x*+*y*) = *z*, then (*x**y*), else (*x*+*z*)
- NET3G: if (*x*+*y*) != *z*, then (*x**y*), else (*x*+*z*)

Comparison IF THEN ELSE functions of three arguments (series H):

- LT3H: if (*x*+*y*) < *z*, then (*x***y*), else (*x*-*z*)
- GT3H: if (*x*+*y*) > *z*, then (*x***y*), else (*x*-*z*)
- LOE3H: if (*x*+*y*) <= *z*, then (*x***y*), else (*x*-*z*)
- GOE3H: if (*x*+*y*) >= *z*, then (*x***y*), else (*x*-*z*)
- ET3H: if (*x*+*y*) = *z*, then (*x***y*), else (*x*-*z*)
- NET3H: if (*x*+*y*) != *z*, then (*x***y*), else (*x*-*z*)

Comparison IF THEN ELSE functions of three arguments (series I):

- LT3I: if (*x*+*y*) < *z*, then (*x***y*), else (*x***z*)
- GT3I: if (*x*+*y*) > *z*, then (*x***y*), else (*x***z*)
- LOE3I: if (*x*+*y*) <= *z*, then (*x***y*), else (*x***z*)
- GOE3I: if (*x*+*y*) >= *z*, then (*x***y*), else (*x***z*)
- ET3I: if (*x*+*y*) = *z*, then (*x***y*), else (*x***z*)
- NET3I: if (*x*+*y*) != *z*, then (*x***y*), else (*x***z*)

Comparison IF THEN ELSE functions of three arguments (series J):

- LT3J: if (*x*+*y*) < *z*, then (*x***y*), else (*x*/*z*)
- GT3J: if (*x*+*y*) > *z*, then (*x***y*), else (*x*/*z*)
- LOE3J: if (*x*+*y*) <= *z*, then (*x***y*), else (*x*/*z*)
- GOE3J: if (*x*+*y*) >= *z*, then (*x***y*), else (*x*/*z*)
- ET3J: if (*x*+*y*) = *z*, then (*x***y*), else (*x*/*z*)
- NET3J: if (*x*+*y*) != *z*, then (*x***y*), else (*x*/*z*)

Comparison IF THEN ELSE functions of three arguments (series K):

- LT3K: if (*x*+*y*) < *z*, then (*x*+*y*+*z*), else sin(*x***y***z*)
- GT3K: if (*x*+*y*) > *z*, then (*x*+*y*+*z*), else sin(*x***y***z*)
- LOE3K: if (*x*+*y*) <= *z*, then (*x*+*y*+*z*), else sin(*x***y***z*)
- GOE3K: if (*x*+*y*) >= *z*, then (*x*+*y*+*z*), else sin(*x***y***z*)
- ET3K: if (*x*+*y*) = *z*, then (*x*+*y*+*z*), else sin(*x***y***z*)
- NET3K: if (*x*+*y*) != *z*, then (*x*+*y*+*z*), else sin(*x***y***z*)

Comparison IF THEN ELSE functions of three arguments (series L):

- LT3L: if (*x*+*y*) < *z*, then (*x*+*y*+*z*), else atan(*x***y***z*)
- GT3L: if (*x*+*y*) > *z*, then (*x*+*y*+*z*), else atan(*x***y***z*)
- LOE3L: if (*x*+*y*) <= *z*, then (*x*+*y*+*z*), else atan(*x***y***z*)
- GOE3L: if (*x*+*y*) >= *z*, then (*x*+*y*+*z*), else atan(*x***y***z*)
- ET3L: if (*x*+*y*) = *z*, then (*x*+*y*+*z*), else atan(*x***y***z*)
- NET3L: if (*x*+*y*) != *z*, then (*x*+*y*+*z*), else atan(*x***y***z*)

Comparison IF THEN ELSE functions of four arguments (series A):

- LT4A: if *a* < *b*, then *c*, else *d*
- GT4A: if *a* > *b*, then *c*, else *d*
- LOE4A: if *a* <= *b*, then *c*, else *d*
- GOE4A: if *a* >= *b*, then *c*, else *d*
- ET4A: if *a* = *b*, then *c*, else *d*
- NET4A: if *a* != *b*, then *c*, else *d*

Comparison IF THEN ELSE functions of four arguments (series B):

- LT4B: if (*a*+*b*) < (*c*+*d*), then *c*, else *d*
- GT4B: if (*a*+*b*) > (*c*+*d*), then *c*, else *d*
- LOE4B: if (*a*+*b*) <= (*c*+*d*), then *c*, else *d*
- GOE4B: if (*a*+*b*) >= (*c*+*d*), then *c*, else *d*
- ET4B: if (*a*+*b*) = (*c*+*d*), then *c*, else *d*
- NET4B: if (*a*+*b*) != (*c*+*d*), then *c*, else *d*

Comparison IF THEN ELSE functions of four arguments (series C):

- LT4C: if (*a*+*b*) < (*c*+*d*), then (*a*+*b*), else (*c*+*d*)
- GT4C: if (*a*+*b*) > (*c*+*d*), then (*a*+*b*), else (*c*+*d*)
- LOE4C: if (*a*+*b*) <= (*c*+*d*), then (*a*+*b*), else (*c*+*d*)
- GOE4C: if (*a*+*b*) >= (*c*+*d*), then (*a*+*b*), else (*c*+*d*)
- ET4C: if (*a*+*b*) = (*c*+*d*), then (*a*+*b*), else (*c*+*d*)
- NET4C: if (*a*+*b*) != (*c*+*d*), then (*a*+*b*), else (*c*+*d*)

Comparison IF THEN ELSE functions of four arguments (series D):

- LT4D: if (*a*+*b*) < (*c*+*d*), then (*a*+*b*), else (*c*-*d*)
- GT4D: if (*a*+*b*) > (*c*+*d*), then (*a*+*b*), else (*c*-*d*)
- LOE4D: if (*a*+*b*) <= (*c*+*d*), then (*a*+*b*), else (*c*-*d*)
- GOE4D: if (*a*+*b*) >= (*c*+*d*), then (*a*+*b*), else (*c*-*d*)
- ET4D: if (*a*+*b*) = (*c*+*d*), then (*a*+*b*), else (*c*-*d*)
- NET4D: if (*a*+*b*) != (*c*+*d*), then (*a*+*b*), else (*c*-*d*)

Comparison IF THEN ELSE functions of four arguments (series E):

- LT4E: if (*a*+*b*) < (*c*+*d*), then (*a*+*b*), else (*c**d*)
- GT4E: if (*a*+*b*) > (*c*+*d*), then (*a*+*b*), else (*c**d*)
- LOE4E: if (*a*+*b*) <= (*c*+*d*), then (*a*+*b*), else (*c**d*)
- GOE4E: if (*a*+*b*) >= (*c*+*d*), then (*a*+*b*), else (*c**d*)
- ET4E: if (*a*+*b*) = (*c*+*d*), then (*a*+*b*), else (*c**d*)
- NET4E: if (*a*+*b*) != (*c*+*d*), then (*a*+*b*), else (*c**d*)

Comparison IF THEN ELSE functions of four arguments (series F):

- LT4F: if (*a*+*b*) < (*c*+*d*), then (*a*+*b*), else (*c*/*d*)
- GT4F: if (*a*+*b*) > (*c*+*d*), then (*a*+*b*), else (*c*/*d*)
- LOE4F: if (*a*+*b*) <= (*c*+*d*), then (*a*+*b*), else (*c*/*d*)
- GOE4F: if (*a*+*b*) >= (*c*+*d*), then (*a*+*b*), else (*c*/*d*)
- ET4F: if (*a*+*b*) = (*c*+*d*), then (*a*+*b*), else (*c*/*d*)
- NET4F: if (*a*+*b*) != (*c*+*d*), then (*a*+*b*), else (*c*/*d*)

Comparison IF THEN ELSE functions of four arguments (series G):

- LT4G: if (*a*+*b*) < (*c*+*d*), then (*a**b*), else (*c*+*d*)
- GT4G: if (*a*+*b*) > (*c*+*d*), then (*a**b*), else (*c*+*d*)
- LOE4G: if (*a*+*b*) <= (*c*+*d*), then (*a**b*), else (*c*+*d*)
- GOE4G: if (*a*+*b*) >= (*c*+*d*), then (*a**b*), else (*c*+*d*)
- ET4G: if (*a*+*b*) = (*c*+*d*), then (*a**b*), else (*c*+*d*)
- NET4G: if (*a*+*b*) != (*c*+*d*), then (*a**b*), else (*c*+*d*)

Comparison IF THEN ELSE functions of four arguments (series H):

- LT4H: if (*a*+*b*) < (*c*+*d*), then (*a***b*), else (*c*-*d*)
- GT4H: if (*a*+*b*) > (*c*+*d*), then (*a***b*), else (*c*-*d*)
- LOE4H: if (*a*+*b*) <= (*c*+*d*), then (*a***b*), else (*c*-*d*)
- GOE4H: if (*a*+*b*) >= (*c*+*d*), then (*a***b*), else (*c*-*d*)
- ET4H: if (*a*+*b*) = (*c*+*d*), then (*a***b*), else (*c*-*d*)
- NET4H: if (*a*+*b*) != (*c*+*d*), then (*a***b*), else (*c*-*d*)

Comparison IF THEN ELSE functions of four arguments (series I):

- LT4I: if (*a*+*b*) < (*c*+*d*), then (*a***b*), else (*c***d*)
- GT4I: if (*a*+*b*) > (*c*+*d*), then (*a***b*), else (*c***d*)
- LOE4I: if (*a*+*b*) <= (*c*+*d*), then (*a***b*), else (*c***d*)
- GOE4I: if (*a*+*b*) >= (*c*+*d*), then (*a***b*), else (*c***d*)
- ET4I: if (*a*+*b*) = (*c*+*d*), then (*a***b*), else (*c***d*)
- NET4I: if (*a*+*b*) != (*c*+*d*), then (*a***b*), else (*c***d*)

Comparison IF THEN ELSE functions of four arguments (series J):

- LT4J: if (*a*+*b*) < (*c*+*d*), then (*a***b*), else (*c*/*d*)
- GT4J: if (*a*+*b*) > (*c*+*d*), then (*a***b*), else (*c*/*d*)
- LOE4J: if (*a*+*b*) <= (*c*+*d*), then (*a***b*), else (*c*/*d*)
- GOE4J: if (*a*+*b*) >= (*c*+*d*), then (*a***b*), else (*c*/*d*)
- ET4J: if (*a*+*b*) = (*c*+*d*), then (*a***b*), else (*c*/*d*)
- NET4J: if (*a*+*b*) != (*c*+*d*), then (*a***b*), else (*c*/*d*)

Comparison IF THEN ELSE functions of four arguments (series K):

- LT4K: if (*a*+*b*) < (*c*+*d*), then sin(*a***b*), else sin(*c***d*)
- GT4K: if (*a*+*b*) > (*c*+*d*), then sin(*a***b*), else sin(*c***d*)
- LOE4K: if (*a*+*b*) <= (*c*+*d*), then sin(*a***b*), else sin(*c***d*)
- GOE4K: if (*a*+*b*) >= (*c*+*d*), then sin(*a***b*), else sin(*c***d*)
- ET4K: if (*a*+*b*) = (*c*+*d*), then sin(*a***b*), else sin(*c***d*)
- NET4K: if (*a*+*b*) != (*c*+*d*), then sin(*a***b*), else sin(*c***d*)

Comparison IF THEN ELSE functions of four arguments (series L):

- LT4L: if (*a*+*b*) < (*c*+*d*), then atan(*a***b*), else atan(*c***d*)
- GT4L: if (*a*+*b*) > (*c*+*d*), then atan(*a***b*), else atan(*c***d*)
- LOE4L: if (*a*+*b*) <= (*c*+*d*), then atan(*a***b*), else atan(*c***d*)
- GOE4L: if (*a*+*b*) >= (*c*+*d*), then atan(*a***b*), else atan(*c***d*)
- ET4L: if (*a*+*b*) = (*c*+*d*), then atan(*a***b*), else atan(*c***d*)
- NET4L: if (*a*+*b*) != (*c*+*d*), then atan(*a***b*), else atan(*c***d*)

Appendix C: List of Logical Functions

Basic logical functions with 1 and 2 inputs:

- Not: A'
- And: A·B
- Or: A+B
- Nand: (A·B)'
- Nor: (A+B)'
- Xor: A^B
- Nxor: (A^B)'

Basic logical functions with 3 inputs:

- And3: A·B·C
- Or3: A+B+C
- Nand3: (A·B·C)'
- Nor3: (A+B+C)'
- Odd3: Odd-3-parity
- Even3: Even-3-parity

Basic logical functions with 4 inputs:

- And4: A·B·C·D
- Or4: A+B+C+D
- Nand4: (A·B·C·D)'
- Nor4: (A+B+C+D)'
- Odd4: Odd-4-parity
- Even4: Even-4-parity

Additional logical functions with 1 input:

- Id: Id(A) = A
- Zero: 0(A) = 0
- One: 1(A) = 1

Additional logical functions with 2 inputs:

- LT: A < B
- GT: A > B
- LOE: A <= B
- GOE: A >= B
- NotA: NOT(A,B) = NOT A
- NotB: NOT(A,B) = NOT B
- IdA: IdA(A,B) = A
- IdB: IdB(A,B) = B
- Zero2: 0(A,B) = 0
- One2: 1(A,B) = 1

Derived logical functions with 3 inputs:

- LT3: A < B < C
- GT3: A > B > C
- LOE3: A <= B <= C
- GOE3: A >= B >= C

Common logical functions with 3 inputs:

- Mux: 3-Multiplexer
- If: If A = 1, then B, else C
- Maj: Majority(A,B,C)
- Min: Minority(A,B,C)
- 2Off: Exactly two off
- 2On: Exactly two on

Universal logical modules with 3 inputs (series A):

- LM3A1: AC'+BC
- LM3A2: AC'+B'C
- LM3A3: A'C'+BC
- LM3A4: A'C'+B'C

Universal logical modules with 3 inputs (series B):

- LM3B1: (A+C')·(B+C)
- LM3B2: (A+C')·(B'+C)
- LM3B3: (A'+C')·(B+C)
- LM3B4: (A'+C')·(B'+C)

Universal logical modules with 3 inputs (series C):

- LM3C1: AB'+BC
- LM3C2: AB'+ BC'
- LM3C3: A'B'+BC
- LM3C4: A'B'+ BC'

Universal logical modules with 3 inputs (series D):

- LM3D1: (A+B')·(B+C)
- LM3D2: (A+B')·(B+C')
- LM3D3: (A'+B')·(B+C)
- LM3D4: (A'+B')·(B+C')

Universal logical modules with 3 inputs (series E):

- LM3E1: A'C+ AB'
- LM3E2: A'C'+AB
- LM3E3: A'C'+ AB'

Universal logical modules with 3 inputs (series F):

- LM3F1: (A'+C)·(A+B')
- LM3F2: (A'+C')·(A+B)
- LM3F3: (A'+C')·(A+B')

Universal logical modules with 3 inputs (series G):

- LM3G1: (A^C')·(B^C)
- LM3G2: (A^C')·(B'^C)
- LM3G3: (A'^C')·(B^C)
- LM3G4: (A'^C')·(B'^C)

Universal logical modules with 3 inputs (series H):

- LM3H1: ((A·B)'· C)'
- LM3H2: (A·(B·C)')'
- LM3H3: ((A+B)'+ C)'
- LM3H4: (A+(B+C)')'

Comparison IF THEN ELSE functions with 3 inputs (series A):

- LT3A: If A < B, then (A·C), else (B·C)'
- GT3A: If A > B, then (A·C), else (B·C)'
- LOE3A: If A <= B, then (A·C), else (B·C)'
- GOE3A: If A >= B, then (A·C), else (B·C)'
- ET3A: If A = B, then (A·C), else (B·C)'
- NET3A: If A != B, then (A·C), else (B·C)'

Comparison IF THEN ELSE functions with 3 inputs (series B):

- LT3B: If A < B, then (A·C)', else (A·C)
- GT3B: If A > B, then (A·C)', else (A·C)
- LOE3B: If A <= B, then (A·C)', else (A·C)
- GOE3B: If A >= B, then (A·C)', else (A·C)
- ET3B: If A = B, then (A·C)', else (A·C)
- NET3B: If A != B, then (A·C)', else (A·C)

Comparison IF THEN ELSE functions with 3 inputs (series C):

- LT3C: If A < B, then (A+C)', else (B+C)
- GT3C: If A > B, then (A+C)', else (B+C)
- LOE3C: If A <= B, then (A+C)', else (B+C)
- GOE3C: If A >= B, then (A+C)', else (B+C)
- ET3C: If A = B, then (A+C)', else (B+C)
- NET3C: If A != B, then (A+C)', else (B+C)

Additional universal logical modules with 3 inputs:

- T004: 00000100
- T008: 00001000
- T009: 00001001
- T032: 00100000
- T033: 00100001
- T041: 00101001
- T055: 00110111
- T057: 00111001
- T064: 01000000
- T065: 01000001
- T069: 01000101
- T073: 01001001
- T081: 01010001
- T089: 01011001
- T093: 01011101
- T096: 01100000
- T101: 01100101
- T109: 01101101
- T111: 01101111
- T121: 01111001
- T123: 01111011
- T125: 01111101
- T154: 10011010
- T223: 11011111
- T239: 11101111
- T249: 11111001
- T251: 11111011
- T253: 11111101

Derived logical functions with 4 inputs:

- LT4: A < B < C < D
- GT4: A > B > C > D
- LOE4: A <= B <= C <= D
- GOE4: A >= B >= C >= D

Common logical functions with 4 inputs:

- Tie: Tie
- Ntie: Not tie
- 3Off: Exactly three off
- 3On: Exactly three on

Universal logical modules with 4 inputs (series A):

- LM4A1: AD'+BD+CD
- LM4A2: AD'+B'D+CD
- LM4A3: AD'+BD+C'D
- LM4A4: AD'+B'D+C'D
- LM4A5: A'D'+BD+CD
- LM4A6: A'D'+B'D+CD
- LM4A7: A'D'+BD+C'D
- LM4A8: A'D'+B'D+C'D

Universal logical modules with 4 inputs (series B):

- LM4B1: (A+D')·(B+D)·(C+D)
- LM4B2: (A+D')·(B'+D)·(C+D)
- LM4B3: (A+D')·(B+D)·(C'+D)
- LM4B4: (A+D')·(B'+D)·(C'+D)
- LM4B5: (A'+D')·(B+D)·(C+D)
- LM4B6: (A'+D')·(B'+D)·(C+D)
- LM4B7: (A'+D')·(B+D)·(C'+D)
- LM4B8: (A'+D')·(B'+D)·(C'+D)

Universal logical modules with 4 inputs (series C):

- LM4C1: AB'+BC+BD
- LM4C2: AB'+B'C+BD
- LM4C3: AB'+BC+B'D
- LM4C4: AB'+B'C+B'D
- LM4C5: A'B'+BC+BD
- LM4C6: A'B'+B'C+BD
- LM4C7: A'B'+BC+B'D
- LM4C8: A'B'+B'C+B'D

Universal logical modules with 4 inputs (series D):

- LM4D1: (A+B')·(B+C)·(B+D)
- LM4D2: (A+B')·(B'+C)·(B+D)
- LM4D3: (A+B')·(B+C)·(B'+D)
- LM4D4: (A+B')·(B'+C)·(B'+D)
- LM4D5: (A'+B')·(B+C)·(B+D)
- LM4D6: (A'+B')·(B'+C)·(B+D)
- LM4D7: (A'+B')·(B+C)·(B'+D)
- LM4D8: (A'+B')·(B'+C)·(B'+D)

Universal logical modules with 4 inputs (series E):

- LM4E1: AC'+BC+CD
- LM4E2: AC'+B'C+CD
- LM4E3: AC'+BC+C'D
- LM4E4: AC'+B'C+C'D
- LM4E5: A'C'+BC+CD
- LM4E6: A'C'+B'C+CD
- LM4E7: A'C'+BC+C'D
- LM4E8: A'C'+B'C+C'D

Universal logical modules with 4 inputs (series F):

- LM4F1: (A+C')·(B+C)·(C+D)
- LM4F2: (A+C')·(B'+C)·(C+D)
- LM4F3: (A+C')·(B+C)·(C'+D)
- LM4F4: (A+C')·(B'+C)·(C'+D)
- LM4F5: (A'+C')·(B+C)·(C+D)
- LM4F6: (A'+C')·(B'+C)·(C+D)
- LM4F7: (A'+C')·(B+C)·(C'+D)
- LM4F8: (A'+C')·(B'+C)·(C'+D)

Universal logical modules with 4 inputs (series G):

- LM4G1: A'D+AB+AC
- LM4G2: A'D+AB'+AC
- LM4G3: A'D+AB+ AC'
- LM4G4: A'D+AB'+ AC'
- LM4G5: A'D'+AB+AC
- LM4G6: A'D'+AB'+AC
- LM4G7: A'D'+AB+ AC'
- LM4G8: A'D'+AB'+ AC'

Universal logical modules with 4 inputs (series H):

- LM4H1: (A'+D)·(A+B)·(A+C)
- LM4H2: (A'+D)·(A+B')·(A+C)
- LM4H3: (A'+D)·(A+B)·(A+C')
- LM4H4: (A'+D)·(A+B')·(A+C')
- LM4H5: (A'+D')·(A+B)·(A+C)
- LM4H6: (A'+D')·(A+B')·(A+C)
- LM4H7: (A'+D')·(A+B)·(A+C')
- LM4H8: (A'+D')·(A+B')·(A+C')

Universal logical modules with 4 inputs (series I):

- LM4I1: (((A·B)'·C)'· D)'
- LM4I2: (A·(B·(C·D)')')'
- LM4I3: ((A·(B· C)')'·D)'
- LM4I4: (A·((B·C)'· D)')'
- LM4I5: (((A+B)'+C)'+ D)'
- LM4I6: (A+(B+(C+D)')')'
- LM4I7: ((A+(B+ C)')'+D)'
- LM4I8: (A+((B+C)'+ D)')'

Comparison IF THEN ELSE functions with 4 inputs (series A):

- LT4A: If A < B, then C, else D
- GT4A: If A > B, then C, else D
- LOE4A: If A <= B, then C, else D
- GOE4A: If A >= B, then C, else D
- ET4A: If A = B, then C, else D
- NET4A: If A != B, then C, else D

Comparison IF THEN ELSE functions with 4 inputs (series B):

- LT4B: If A < B, then (C·D), else D'
- GT4B: If A > B, then (C·D), else D'
- LOE4B: If A <= B, then (C·D), else D'
- GOE4B: If A >= B, then (C·D), else D'
- ET4B: If A = B, then (C·D), else D'
- NET4B: If A != B, then (C·D), else D'

Comparison IF THEN ELSE functions with 4 inputs (series C):

- LT4C: If A < B, then (C+D), else D'
- GT4C: If A > B, then (C+D), else D'
- LOE4C: If A <= B, then (C+D), else D'
- GOE4C: If A >= B, then (C+D), else D'
- ET4C: If A = B, then (C+D), else D'
- NET4C: If A != B, then (C+D), else D'

Comparison IF THEN ELSE functions with 4 inputs (series D):

- LT4D: If A < B, then (A·D), else (C·D)'
- GT4D: If A > B, then (A·D), else (C·D)'
- LOE4D: If A <= B, then (A·D), else (C·D)'
- GOE4D: If A >= B, then (A·D), else (C·D)'
- ET4D: If A = B, then (A·D), else (C·D)'
- NET4D: If A != B, then (A·D), else (C·D)'

Comparison IF THEN ELSE functions with 4 inputs (series E):

- LT4E: If A < B, then (A·D)', else (A·C)
- GT4E: If A > B, then (A·D)', else (A·C)
- LOE4E: If A <= B, then (A·D)', else (A·C)
- GOE4E: If A >= B, then (A·D)', else (A·C)
- ET4E: If A = B, then (A·D)', else (A·C)
- NET4E: If A != B, then (A·D)', else (A·C)

Additional universal logical modules with 4 inputs:

- Q0002: 0000000000000010
- Q001C: 0000000000011100
- Q0048: 0000000001001000
- Q0800: 0000100000000000
- Q3378: 0011001101111000
- Q3475: 0011010001110101
- Q3CB0: 0011110010110000
- Q3DEF: 0011110111101111
- Q3DFF: 0011110111111111
- Q4200: 0100001000000000
- Q4C11: 0100110000010001
- Q5100: 0101000100000000
- Q5EEF: 0101111011101111
- Q5EFF: 0101111011111111
- Q6A6D: 0110101001101101
- Q6F75: 0110111101110101
- Q74C4: 0111010011000100
- Q7DA3: 0111110110100011
- Q8304: 1000001100000100
- Q8430: 1000010000110000
- Q8543: 1000010101000011
- Q9D80: 1001110110000000
- QA092: 1010000010010010
- QB36A: 1011001101101010
- QCBCF: 1100101111001111
- QEEB1: 1110111010110001
- QEFFF: 1110111111111111
- QFF7B: 1111111101111011
- QFFF6: 1111111111110110
- QFFFB: 1111111111111011

Appendix D: Installation (Add-on and Standalone)

GeneXproServer ships as an executable that must be installed in a computer where GeneXproTools is already installed. The installation creates another installation executable (GeneXproServerStandAloneSetup.exe), which can be used to install GeneXproServer in machines that don't have GeneXproTools installed. Essentially, this last installer contains all the files needed for GeneXproServer but it cannot be run in a computer where GeneXproTools is installed. This installation can be used to install GeneXproServer on a server cluster, for example. You must own one license of GeneXproServer for each computer where you install it, either as an add-on or standalone.