



## Reference Manual

NETRAL 1997-2008





# Neuro Developer Kit

## NETRAL 1997-2008

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: juillet 2008 in Issy-Les-Moulineaux France

# Table of Contents

<b>Part I Neuro Developer Kit 7.0</b>	<b>3</b>
1 NDK installation .....	3
2 How to use NDK with your program .....	3
<b>Part II NDK reference guide</b>	<b>6</b>
1 Presentation .....	6
Conventions .....	6
C / C++ usage .....	6
Delphi usage .....	7
Python usage .....	7
2 Types .....	7
Procedural types .....	7
TCallbackFct .....	8
TDataFunc .....	8
TData .....	9
DataTable .....	10
DataTable: Training and test data set .....	10
TIndexPilotInfo .....	11
TIndexInfoTable .....	11
TLayerData .....	14
TLoopData .....	15
TMergeData .....	15
TModelAction .....	16
ModelActionTable .....	16
TModelData .....	16
TModifyModel .....	17
ModifyModelTable .....	18
TNeuronData .....	18
NeuronDataTable .....	19
TNTText .....	20
TextTable .....	20
TPilotInfo .....	22
InfoTable .....	22
TRealPilot .....	25
RealTable .....	25
TSynapseData .....	26
SynapseDataTable .....	26
TTransfer .....	27
TransferTable .....	27
TransferTableFD .....	28
Other types .....	28
3 Constants .....	28
Differential loop .....	29
Activation functions .....	29
4 Primitives .....	29

<b>Level 0</b> .....	<b>30</b>
CloseLibrary .....	30
CreateNewModel .....	31
DestroyModel .....	31
GetIndexInfo .....	31
GetInfo .....	32
GetRealValue .....	32
GetText .....	33
GetVector .....	33
LicenseLevel .....	34
LoadModelFromFile .....	34
ModelAction .....	34
OpenLibrary .....	35
SetIValue .....	35
SetRandSeed .....	36
SetRealValue .....	36
SetText .....	36
SetTrainingSet .....	37
SetVector .....	37
TestFile .....	38
Train .....	38
Transfer .....	39
<b>Level 1</b> .....	<b>40</b>
DLLQueryingToken .....	40
LoadParameters .....	40
SaveModel .....	40
SaveParameters .....	41
<b>Level 2</b> .....	<b>41</b>
CreateDataAnalyser .....	41
FunctionCompute2D .....	42
FunctionCompute3D .....	42
GetNeuroneInfo .....	43
GetSynapseInfo .....	43
HiddenLayerAnalysis .....	44
InputGrammSchmidt .....	44
LoadOptimizer .....	45
MultiTrain .....	45
NewOptimizer .....	46
OptimizeInputs .....	46
OptimizeInputsEx .....	47
TransferData .....	47
TransferFD .....	48
<b>level 3</b> .....	<b>49</b>
CloneModel .....	49
ModifyModel .....	49
MoveModel .....	50
SetDataProc .....	50



**Part**



# 1 Neuro Developer Kit 7.0

The ©**Neuro Developer Kit (NDK)** allows the neural network engine of <%NEUROONE%> to be used in other applications.

The NDK shares a common model saving format with all the products of the ©**Neuro One Suite**. A model designed with one of these products may be used indifferently by any other product of the Suite. The NDK offers an interface of this format with other custom applications.

The NDK is mainly a Dynamic Linked Library which exports the Netral Neural engine functionalities. This library may be called from many different applications.

The NDK exists in 4 license levels, represented by 4 dlls :

- MonaEx70\_0.dll → level 0
- MonaEx70\_1.dll → level 1
- MonaEx70\_2.dll → level 2
- MonaEx70.dll → level 3

NDK includes the interfaces with <%MATLAB%> under the name of <%NEUROONEML%>

## 1.1 NDK installation

For the level 0 license, you just have to copy then MonaEx70\_0.dll to your computer.

For the higher level licenses,  
first, download the installation program from the NETRAL site  
second, run the installation program and follow the on screen indications.

## 1.2 How to use NDK with your program

NDK is a dynamic link library(dll). This library exposes the interface routines to then Non linear engine from Netral. It can be used like any other dll, depending on your development language.

- **Delphi**  
Interface unit "MonaEx70.pas".
- **C / C++**  
header file "MonaEx70.h";  
Library file "MonaEx70.lib";  
Definition file "MonaEx70.def"

Interface routines are described in the interfaces files. Please find the complete documentation in this book.

### CAUTION

NDK Interface routines are using the standard calling convention **stdcall** . This convention is defined

by :

- Parameters are read from the right to the left
- Fonction cleaning.

The calling routines must use this convention, as it is in the interface files "MonaEx70.pas" or "MonaEx70.h"

**Part**



## 2 NDK reference guide

[Neuro Developer Kit 7.0](#) <sup>3</sup>

[Constants](#) <sup>28</sup>

[Types](#) <sup>7</sup>

[Primitives](#) <sup>28</sup>

### 2.1 Presentation

[Conventions](#) <sup>6</sup>

[C / C++ usage](#) <sup>6</sup>

[Delphi usage](#) <sup>7</sup>

[Python usage](#) <sup>7</sup>

#### 2.1.1 Conventions

The constants, types and routines of NDK are described in this manual. Their declarations are given in three frequent languages :

- C - C ANSI normalisation
- C++ - C++ Builder reference
- Pascal - Delphi reference

The calling convention is always the standard convention (`__stdcall`) :

- Argument-passing order right to left;
- Called function pops its own arguments from the stack;

You must use this calling convention when you write your application in order to call a NDK routine. Depending upon the IDE or the compiler, the declaration modifiers may vary. The examples from the "MonaEx70.h", "MonaEx70.pas" files are not contractual.

#### 2.1.2 C / C++ usage

Import a NDK routine in C/C++ is straightforward :

1. include the "MonaEx70.h" in your application c file;
2. link your application with the library "MonaEx70.lib". This file may be recreated by using a utility like IMPLIB or similar with the "MonaEx70.dll" file.

**Warning** - The library file may depends upon the used compiler. Netral supply the files for Microsoft Visual C++ and Borland compilers. For other compilers, please check the compiler documentation to find how to import a DLL.

### 2.1.3 Delphi usage

Import a NDK routine in Delphi is straightforward.

Include the direct import file "Monallmp.pas" or the dynamic input file "MonallmpDyn.pas" into the application uses clause, and call the routine.

### 2.1.4 Python usage

A NDK routine is imported by Python like any other standard Windows DLL routine, using the ctypes module..

```
from ctypes import windll
monal = windll.LoadLibrary( dllpath )
... use "monal" to call the DLL exposed functions.
```

check the ctypes module help.

The reference and pointers calls use the ctypes.byref() calling.

## 2.2 Types

[TCallbackFct](#) <sup>8</sup>

[TData](#) <sup>9</sup>

[TPilotInfo](#) <sup>22</sup>

[TIndexPilotInfo](#) <sup>11</sup>

[TLayerData](#) <sup>14</sup>

[TLoopData](#) <sup>15</sup>

[TMergeData](#) <sup>15</sup>

[TModelAction](#) <sup>16</sup>

[TModelData](#) <sup>16</sup>

[TModifyModel](#) <sup>17</sup>

[TNeuronData](#) <sup>18</sup>

[TNewDataProc](#) <sup>8</sup>

[TNTxt](#) <sup>20</sup>

[TRealPilot](#) <sup>25</sup>

[TSynapseData](#) <sup>26</sup>

[Autres types](#) <sup>28</sup>

### 2.2.1 Procedural types

Enter topic text here.

### 2.2.1.1 TCallBackFct

Callback function for the time consuming procedures.

#### C / C++ declaration

```
typedef long __stdcall (*TCallBackFct)(long param1, long param2,
double param3, long style);
```

#### Pascal declaration

```
TCallBackFct = function (param1, param2: LongInt; param3: Double;
style: LongInt): LongInt; stdcall;
```

The available calling style are :


```
cbk_None      0
cbk_TrainNumber 1
cbk_TrainProgress 2
cbk_TrainEnd 3
cbk_InitParam 4
cbk_StartCycle 5
cbk_EndCycle 6
cbk_Transfer 7
cbk_Bootstrap 8
```

The messages send by the system are the followings :

param1	param2	param3	style	Quand
0	0	0	cbk_InitParam	Parameter initialization
NAppr	Epoch	Cost	cbk_Bootstrap	After each bootstrap
Nombre	FinAppr	Cost	cbk_TrainEnd	At the end of training
0	NCompute	0	cbk_StartCycle	Multi training cycle beginning
i	NCompute	Sortie[0]	cbk_Transfer	After a transfer in MultiTransfer
i	NCompute	NAN	cbk_Transfer	After a transfer if there is a NAN in the inputs
NCompute	NCompute	0	cbk_EndCycle	At the end of a multitransfer
0	Count	CoutCourant	cbk_TrainProgress	Training start
i	Count	CoutCourant	cbk_TrainProgress	After each training cycle

[Train](#) 

### 2.2.1.2 TDataFunc

**DataFuncs** are function designed to pre-process the data while they are loaded in the model. They are stuck to the model through the [SetDataFunc](#)  procedure.

**C/C++ declaration**

```
typedef long __stdcall (*TTranslateDataFunc)(char *Source:, char
Sep, int Len, void *Target);
typedef long __stdcall (*TModifyDataFunc)(Data: Pointer);
```

**Pascal declaration**

```
TTranslateDataFunc = function(Source: PChar; Sep: Char; Len:
Integer; Target: Pointer): LongInt; stdcall;
TModifyDataFunc = function(Data: Pointer): LongInt; stdCall;
```

Those two procedural types can be used by [SetDataFunc](#)<sup>[50]</sup>, depending of its style. Once affected, they process the data while they are stored in internal memory.

The type TTranslateDataFunc allows a pre processing of the line read in a csv like text file. This function must perform the data translation towards the "Target" vector.

The type TModifyDataFunc allows a pre processing of the real values vector just before its storing in the memory.

these functions must return 0 if succeeded.

If they exist, these functions are applied onr after the other.

**2.2.2 TData****C / C++ declaration**

```
typedef enum {drTrainingData, drTrain, drTest, drOutputVector,
drStateVector, drLocalCost, drErrors, drVariables, drNormalizationSet,
drInOutNormalizationSet, drExperimentalDomain, drOutputDomain,
drParamDomain, drParamIni, drJacobian, drDispersionMatrix,
drConfidenceParam, drConfidenceMultiplier, drParamUpDownList,
drBoolInputs, drTrainingResult, drBootStrapParam, drLeverages,
drResiduals, drInputs, drOutputs, drComputedOutputs, drWeighting,
drComputedOutputsPCI, drComputedOutputsMCI, drParamCorrelMatrix,
drParametersPCI, drParametersMCI} TData;
```

**Pascal declaration**

```
TData = (_TrainingData, _Train, _Test, _OutputVector, _StateVector,
_LocalCost, _Errors, _Variables, _NormalizationSet,
_InOutNormalizationSet, _ExperimentalDomain, _OutputDomain,
_ParamDomain, _ParamIni, _Jacobian, _DispersionMatrix,
_ConfidenceParam, _ConfidenceMultiplier, _ParamUpDownList,
_BooleanInputs, _TrainingResult, _BootStrapParam, _Leverages, _Residuals,
_Inputs, _Outputs, _ComputedOutputs, _Weighting, _ComputedOutputsPCI,
_ComputedOutputsMCI, _ParamCorrelMatrix, _ParametersPCI,
_ParametersMCI);
```

[DataTable](#)<sup>[10]</sup>  
[DataTable: Training and test data set](#)<sup>[10]</sup>  
[GetVector](#)<sup>[33]</sup>  
[SetVector](#)<sup>[37]</sup>

### 2.2.2.1 DataTable

Value	In/Out	level (s)	Index	Type	Meaning
_Errors	O	0	NA	long	Data file line reading error vector
_Variables	I/O	0/0	NA	real	Parameter vector
_NormalizationSet	I/O	1/0	NA	real	Normalization matrix (inputs then outputs)
_InOutNormalizationSet	I/O	1/0	0:In;1:Out	real	Full normalization matrix.
_ExperimentalDomain	I/O	2/2	NA	real	Experimental space
_OutputDomain	I/O	2/2	NA	real	Output space
_ParamDomain	I/O	2/2	NA	real	Parametric space paramétrique
_ParamIni	I/O	0/0	NA	real	Parameters initialization factors
_Jacobian	O	2	output	real	Jacobian matrix
_DispersionMatrix	O	2	output	real	Dispersion matrix
_ConfidenceParam	O	0	output	real	Confidence parameters vector.
_ConfidenceMultiplier	O	0	NA	real	Confidence multiplier vector.
_ParamUpDownList	O	0	NA	long	Parent-Child pairs list.
_BoolInputs	I/O	2/2	NA	long	Inputs freezing in reverse mode ( <a href="#">OptimizeInputs</a> <sup>[46]</sup> )
_TrainingResult	O	2	NA	real	Training results matrix or train index vector
_BootStrapParam	O	2	NA	real	Bootstrap indexed parameters.
_ParamCorrelMatrix	O	2	output	real	Parameter correlation matrix
_ParametersCI	O	2	NA	real	Confidence lower and upper limit of parameter double vector

### 2.2.2.2 DataTable: Training and test data set

Value	In/Out	level (s)	Index	Type	Meaning
_TrainingData	I/O	0/0	data	real	Training and test data
_TrainingDataLogging	O	0	data	real	Training data, test data with computation results
_Train	O	0	NA	real	Training data
_Test	O	0	NA	real	Test data
_OutputVector	O	0	NA	real	Output vector
_StateVector	O	2	NA	real	State vector
_LocalCost	O	0	Output	real	Local cost vector
_Leverages	O	2	Output	real	Leverage vertical vector.
_Residuals	O	0	Output	real	Residual vertical vector.

_Inputs	O	0	Input	real	Inputs vertical vector.
_Outputs	O	0	Output	real	Outputs vertical vector.
_States	I/O	2/2	State	real	Computed or initialisation state vertical vector.
_ComputedOutputs	O	0	Output	real	Computed outputs vertical vector.
_Weighting	I/O	2/2	Output	real	Output weighting vertical vector
_ComputedOutputs CI	O	2	Output	real	confidence lower and upper limit of computed outputs vertical double vector

## 2.2.3 TIndexPilotInfo

### C / C++ declaration

```
typedef enum (iipIsTest, iipNeuronClass, iipLayer, iipNodeLayerIndex,
iipLayerLength, iipSynChildren, iipSynChildChild, iipSynChildType,
iipSynChildNum, iiStateLayer, iiStateNodeLayerIndex, iiNodeNetIndex,
iiNLayer, iiRangJacob, iiBestTrain) TIndexPilotInfo;
```

### Pascal declaration

```
TIndexPilotInfo = (iipIsTest, iipNeuronClass, iipLayer,
iipNodeLayerIndex, iipLayerLength, iipSynChildren, iipSynChildChild,
iipSynChildType, iipSynChildNum, iiStateLayer, iiStateNodeLayerIndex,
iiNodeNetIndex, iiNLayer, iiRangJacob, iiBestTrain);
```

[TIndexInfoTable](#) <sup>[11]</sup>

Used in [GetIndexInfo](#) <sup>[31]</sup>

### 2.2.3.1 TIndexInfoTable

iipIsTest	Is this data line a member of the test set
iipNeuronClass	class index of the indexed node
iipLayer	layer index of the indexed node
iipNodeLayerIndex	in layer index of the indexed node
iipLayerLength	layer length
iipSynChildren	children number of the indexed node
iipSynChildChild	node index of the child node of index hi(index) of the indexed lo(index) node
iipSynChildType	type of the indexed hi(Index) link of the indexed lo(Index) node
iipSynChildNum	number of the indexed hi(Index) link of the indexed lo (Index) node
iiStateLayer	layer of the indexed output state node
iiStateNodeLayerIndex	index in its layer of the indexed output state node
iiNodeNetIndex	model index of the indexed node (Multiple models)
iiNLayer	layer number of the indexed model (Multiple models)
iiRangJacob	jacobian rank of the indexed output
iiBestTrain	index of the best training, following the indexed criterium (note 1)
iKSize	Kohonen map dimension (1 or 2)

<code>iModelClass</code>	Model class index (index < 0) or Included model class for model container.
--------------------------	----------------------------------------------------------------------------

1. Training criterium : [InfoTable](#) <sup>24</sup>

2. Model class index :

- 1 Neuron network
- 2 Compiled model
- 3 Kohonen network
- 4 other node model
- 5 parallel container model
- 6 serial container model
- 7 generic container model
- 8 generic linear model
- 9 matrix model
- 10 normalizer model
- 11 polynôm model
- 12 identity model
- 255 generic model



## 2.2.4 TLayerData

### C / C++ declaration

```
typedef struct
{
    long size;
    short Index;
    short Length;
    short Activation;
    TNeuronType NeuronType;
    TStylePos StylePos;
    short ActivationPlus;
    char LName[256];
    int NetIndex;
} TLayerData;
```

### Pascal declaration

```
PPlayerData = ^TLayerData;
TLayerData = record
    size: LOnGInt;
    Index: SmallInt;
    Length: SmallInt;
    Activation: SmallInt;
    NeuronType: TNeuronType;
    StylePos: TStylePos;
    ActivationPlus: SmallInt;
    LName: Array[0..255] of char;
    NetIndex: Integer;
end;
```

[ModifyModel](#) 

## 2.2.5 TLoopData

### C / C++ declaration

```
typedef struct
{
    short ILayer;
    short INeuron;
    short NState;
} TLoopData;
```

### Pascal declaration

```
TloopData = record
Begin
    Ilayer: SmallInt;
    INeuron: SmallInt;
    NState: SmallInt;
end;
```

State model :

**NState** is the loop order  
**ILayer** and **INeuron** are 0

Input/Output model :

**ILayer** is the layer of the looped node  
**INeuron** is the index in its layer of the looped node  
**NState** is the number of loops

## 2.2.6 TMergeData

### C / C++ declaration

```
typedef struct
{
    long Size;
    char Source[256];
    short NInputMerge;
    short NLayerShift;
    short NNodeShift;
    short AutoMerge;
    short PlugIn;
    short SharedSynapses;
    short ByName;
} TMergeData;
```

### Pascal declaration

```
PMergeData = ^TMergeData
TMergeData = record
    Size: LongInt;
    NetWorkFile: array [0..255] of Char;
    NbInputMerge: Word;
    NbLayerShift: Word;
    NbNeuronShift: Word;
```

```

    AutoMerge: Word;
    PlugIn: Word;
    SharedSynapses: Word;
    ByName: Word;
end;

```

[ModifyModel](#)<sup>[49]</sup>

## 2.2.7 TModelAction

### C / C++ declaration

```

typedef enum {maInitParam, maLoadTrainingData, maNormalizeData,
maClearComment, maConfidenceComputation, maClearTrainModelList}
TModelAction;

```

### Pascal declaration

```

TModelAction = (maInitParams, maLoadTrainingData, maNormalizeData,
maClearComment, maConfidenceComputation, maClearTrainModelList);

```

[ModelActionTable](#)<sup>[16]</sup>

[ModelAction](#)<sup>[34]</sup>

### 2.2.7.1 ModelActionTable

value	Action performed
maInitParams	Parameters initialization. Parameters initialization values have been fixed by a call to <b>SetRealValue</b> with the style rpIniDev.
maLoadTrainingData	Training data loading from a file. File name and data sizes have been fixed by a call to <b>SetTrainingSet</b> .
maNormalizeData	Training data normalization. Normalisation scale has been fixed by a call to <b>SetRealValue</b> with the style rpScale.
maClearComment	Comments clearing.
maConfidenceComputation	Dispersion matrix and confidence multipliers computation. Confidence level has been fixed by a call to <b>SetRealValue</b> with the style rpConfLevel.
maClearTrainModelList	Training model list clearing (Multi training).

## 2.2.8 TModelData

### C / C++ declaration

```

typedef struct
{
    long size;                Record size
    int NModelInput;          Input number
    int NModelOutput;         Output number
    int NHidden;              Hidden nodes number / -polynom order
    short HiddenActiv;        Hidden node activation (default 2)
    short ILayer;
    short INeuron;
    short NState;
    short Redondancy;         1: loop redundancy treated
    short DoNotCreateSynapses; Synapseless creation
    short ExistingNeuron;     Don't create node for looping
}

```

```

short PolyType;
1 : no square in polynoms
2 : no bias on the output
4 : no bias on the first layer

} TModelData ;

```

### Pascal declaration

```

TModelData = record
  Size: LongInt;           Record size
  NModelInput: integer;   Input number
  NModelOutput: Integer;  Output number
  NHidden: Integer;       Hidden nodes number / -polynom order
  AHiddenActiv: SmallInt; Hidden node activation (default 2)
  ILayer: SmallInt;
  INeuron: SmallInt;
  NState: SmallInt;
  Redondancy: Boolean;    1: loop redundancy treated
  DoNotCreateSynapses: Boolean Synapseless creation
  ExistingNeuron: Boolean; Don't create node for looping
  PolyType: SmallInt;    1 : no square in polynoms
                        2 : no bias on the output
                        4 : no bias on the first layer
end;

```

The three datas **ILayer**, **INeuron** et **NState** are used for a looped model.

State model :

**NState** is the loop order  
**ILayer** and **INeuron** are 0

Input/Output model :

**ILayer** is the layer of the looped node  
**INeuron** is the index in its layer of the looped node  
**NState** is the number of loops

## 2.2.9 TModifyModel

### C / C++ declaration

```

typedef enum { mnSolidifyNetwork, mnModifyNeuron, mnAddLoop,
mnDifferentialLoop, mnNewNeuron, mnDestroyNeuron, mnMoveNeuron,
mnMergeNeuron, mnNewSynapse, mnDestroySynapse, mnModifySynapse,
mnNewLayer, mnDestroyLayer, mnMergeNetwork, mnCompactNetwork }
TModifyNetwork;

```

### Pascal declaration

```

TModifyNetwork = (mnSolidifyNetwork, mnModifyNeuron, mnAddLoop,
mnDifferentialLoop, mnNewNeuron, mnDestroyNeuron, mnMoveNeuron,
mnMergeNeuron, mnNewSynapse, mnDestroySynapse, mnModifySynapse,
mnNewLayer, mnDestroyLayer, mnMergeNetwork, mnCompactNetwork);

```

[ModifyModelTable](#) <sup>18</sup>

[ModifyModel](#)<sup>[49]</sup>

## 2.2.9.1 ModifyModelTable

Style	Action	Param
mnSolidifyNetwork	All the synapses of the model are petrified. The model will not be able to be trained. No more gradient computation, leverages, neither confidence intervals.	ND
mnModifyNeuron	Neuron modification.	<a href="#">TNeuronData</a> <sup>[18]</sup> (1) structure address
mnAddLoop	Add a loop to the model.	Pointeur sur un <a href="#">TLoopData</a> <sup>[15]</sup> structure address
mnDifferentialLoop	Add a differential loop to the model	<a href="#">Constante de bouclage</a> <sup>[29]</sup>
mnNewNeuron	Create and insert a new neuron in the model	<a href="#">TNeuronData</a> <sup>[18]</sup> structure address
mnDestroyNeuron	Destroy a neuron in the model	<a href="#">TNeuronData</a> <sup>[18]</sup> structure address
mnMoveNeuron	Move a neuron in the model	<a href="#">TSynapseData</a> <sup>[26]</sup> structure address
mnMergeNeuron	Merge two neurons in the model	<a href="#">TSynapseData</a> <sup>[26]</sup> structure address
mnNewSynapse	Create a synaptic link	<a href="#">TSynapseData</a> <sup>[26]</sup> structure address
mnDestroySynapse	Destroy a synapse	<a href="#">TSynapseData</a> <sup>[26]</sup> structure address
mnModifySynapse	Modify a synapse	<a href="#">TSynapseData</a> <sup>[26]</sup> (2) structure address
mnNewLayer	Create a neuron layer in the model	<a href="#">TLayerData</a> <sup>[14]</sup> structure address
mnDestroyLayer	Destroy a neuron layer in the model	<a href="#">TLayerData</a> <sup>[14]</sup> structure address
mnMergeNetwork	Merge two models	<a href="#">TMergeData</a> <sup>[15]</sup> structure address
mnCompactNetwork	Suppress useless layers, neurons and synapses in the model.	ND

## 2.2.10 TNeuronData

## C / C++ declaration

```

typedef enum { ntNone, ntSigma, ntSigma2, ntPi } TNeuronType;
typedef enum { psBegin, psEnd } TPositionStyle;
typedef enum { anDontCare, anConstant, anInput, anOutPut }
TAddNeuronType;

typedef struct
{
    long size;
    char Name[256];
    short ModelName;
    short LayerIndex;
    short NodeIndex;
    short Activation[28];
    long NeuronID;
    long NeuronType; // TNeuronType
    long StylePos; // TPositionStyle
    long NeuronPosition; // TAddNeuronType

```

```

    long ActivationPlus[29];
    long IndexNet;
    double Value;
} TNeuronData;

```

### Pascal declaration

```

TNeuronType = (ntNone, ntSigma, ntSigma2, ntPi);
TPositionStyle = (psBegin, psEnd);
TAddNeuronType = (anDontCare, anConstant, anInput, anOutPut,
anOutState);

```

```

TNeuroneData = record
    Size: Integer;
    Name: array[0..255] of Char;
    ModelName: LongInt;
    LayerIndex: LongInt;
    NodeIndex: LongInt;
    Activation[29]: LongInt;
    NeuronID: LongInt;
    NeuronType: LongInt;           //TNeuronType
    StylePos: LongInt;           //TPositionStyle
    NeuronPosition: LongInt;     //TAddNeuronType
    ActivationPlus[29]: LongInt;
    IndexNet: LongInt;
    Value: Double;
end;

```

[NeuronDataTable](#)<sup>[19]</sup>

[GetNeuroneInfo](#)<sup>[43]</sup>

[ModifyModel](#)<sup>[49]</sup>

#### 2.2.10.1 NeuronDataTable

TPositionStyle :

psBegin	0	Index counted from beginning
psEnd	1	Index counted from the end

TNeuronType

ntNone	0	Single output, no activation
ntSigma	1	Standard weighted sum neuron
ntSigma2	2	Weighted sum of square neuron
ntPi	3	Weited product neuron

TAddNeuronType

anDontCare	0	NA
anConstant	1	Constant neuron
anInput	2	Input neuron
anOutPut	3	Outpt neuron
anOutState	4	State output neuron

Fields definition:

size	Record size
Name	Node or model name
ModelName	boolean giving the name holder
LayerIndex	Layer index in the model
NodeIndex	Node index in the layer
<a href="#">Activation</a> <sup>[29]</sup>	Activation index
NeuronID	Node ID
NeuronType	Type of node, following TNeuronType
StylePos	Type of positionning, following TPositionStyle
NeuronPosition	Node position, following TAddNeuronType
<a href="#">ActivationPlus</a> <sup>[29]</sup>	Pre activation index (reserved)
IndexNet	Model index in multi models
Value	Node output value

## 2.2.11 TNText

### C / C++ declaration

```
typedef enum {gt_Caller, gt_ProductInfo, gt_Version, gt_FunctionName,
gt_ModelName, gt_Formula, gt_ModelString,gt_Comment, gt_InputName,
gt_OutputName, gt_Sticker, gt_ParameterName, gt_ModelClassName,
gt_StateName, gt_NodeName, gt_SaveDir, gt_MatrixSubDir, gt_ParamFile,
gt_NoiseModelFile, gt_MMLFunction, gt_AddTrainModel, gt_MGL, gt_Excel,
gt_Code} TNText;
```

### Pascal declaration

```
TNText = {gt_Caller, gt_ProductInfo, gt_Version, gt_FunctionName,
gt_ModelName, gt_Formula, gt_ModelString, gt_Comment, gt_InputName,
gt_OutputName, gt_Sticker, gt_ParameterName, gt_ModelClassName,
gt_StateName, gt_NodeName, gt_SaveDir, gt_MatrixSubDir, gt_ParamFile,
gt_NoiseModelFile, gt_MMLFunction, gt_AddTrainModel, gt_MGL, gt_Excel,
gt_Code) ;
```

### Comments

Used with [GetText](#)<sup>[33]</sup> and [SetText](#)<sup>[36]</sup>.

[TextTable](#)<sup>[29]</sup>

### 2.2.11.1 TextTable

Values meaning :

Value	In/Out	level (s)	Handle	Ind0	Ind 1	Signification
gt_Caller	O	0	NA	NA	NA	Library calling program
gt_ProductInf	O	0	NA	NA	NA	Library info

o						
gt_Version	O	0	NA	NA	NA	Library version
gt_FunctionName	O	0	NA	fType	Index	function name
gt_ModelName	I/O	2/0	Handle	NA	NA	Model name
gt_Formula	O	0	Handle	NA	NA	Model computation formula
gt_Comment	O	0	Handle	Index	NA	Indexed comment
gt_InputName	I/O	2/0	Handle	Index	NA	Indexed input name
gt_OutputName	I/O	0	Handle	Index	NA	Indexed output name
gt_Sticker	O	0	Handle	Layer	Node	Sticker
gt_ParameterName	I/O	2/0	Handle	Index	NA	Indexed parameter name
gt_ModelClassName	O	2	Handle	NA	NA	Model class name
gt_StateName	I/O	2/2	Handle	Index	NA	Indexed state variable name.
gt_NodeName	I/O	2/0	Handle	Layer	Node	Node name. Node is defined by its layer and its layer index.
gt_SaveDir	I/O	0/0	NA	NA	NA	Training results saving folders (note 1).
gt_ParamFile	O	1	Handle	Train #		Training saving filename
gt_NoiseModelFile	I	2	Handle	NA	NA	Recording a noise model file.
gt_MMLFunction	I	2	Handle	NA	NA	Recording of a MathML function (Activation, Cost, Proximity, decreasing)
gt_AddTrainModel	O	0	Handle	NA	NA	Recording a model file for multi training.
gt_Diagnostic	O	0	Handle	Level	NA	Diagnostic of the current trained model.
gt_Separator	I/O	0/0	Handle	NA	NA	Data separator in files
gt_Code	O	0	Handle	Lang	Fct	Model code (note 2)

### Notes

1: Saving folder nature is depending upon txttype.

txttype may take three values:

- 0 : Base saving folder. If undefined, NDK uses system defined temporary file ;
- 1 : Folder relative to Base saving folder, for saving temporary training results ;
- 2 : Folder relative to Base saving folder, for saving models for multi-training.

2 : The code language is defined by Ind0 :

- 0 : tcUnknown -> none
- 1 : tcXML -> Model XML code
- 2 : tcC -> Model C code (reserved)
- 3 : tcVBasic -> Model VBasic code (reserved)
- 4 : tcXL -> Excel worksheet code to be pasted in A1 cell.
  - Ind1 = 0 : horizontal
  - Ind1 = 1 : vertical
- 5 : tcMGL -> Excel MGL code.

## 2.2.12 TPilotInfo

### C / C++ declaration

```
typedef enum { ipNInput, ipNOutput, ipNOrder, ipCanLearn,
ipTransposed, ipDimension, ipNbLayer, ipClass, ipNHidden, ipNTrain,
ipBootStrap, ipStatus, ipNNeuron, ipNModelInput, ipNSynapse, ipNData,
ipTrainCost, ipEndTrain, ipDataLoaded, ipIsLin, ipCommentCount,
ipModelClass, ipNFuncAct, ipNFuncCost, ipNFuncProx, ipNFuncDecay,
ipValidHandle, ipBin, ipNTrainModel, ipBootStrapType,
ipTrainingAlgorithm, ipSelectedOutput, ipProximity, ipDecay,
ipProximityDecay, ipReverseMode } TPilotInfo;
```

### Pascal declaration

```
TInfoPilot = (ipNInput, ipNOutput, ipNOrder, ipCanLearn, ipTransposed,
ipDimension, ipNbLayer, ipClass, ipNHidden, ipNTrain, ipBootStrap,
ipStatus, ipNNeuron, ipNModelInput, ipNSynapse, ipNData, ipTrainCost,
ipEndTrain, ipDataLoaded, ipIsLin, ipCommentCount, ipModelClass,
ipNFuncAct, ipNFuncCost, ipNFuncProx, ipNFuncDecay, ipValidHandle,
ipBin, ipNTrainModel, ipBootStrapType, ipTrainingAlgorithm,
ipSelectedOutput, ipProximity, ipDecay, ipProximityDecay,
ipReverseMode);
```

### Comments

Used with [GetInfo](#)<sup>[22]</sup> and [SetlValue](#)<sup>[35]</sup>.

[InfoTable](#)<sup>[22]</sup>

#### 2.2.12.1 InfoTable

Values meaning :

value	In/ Out	level(s)	action
ipNInput	IO	3/0	Model input number.
ipNOutput	O	0	Model output number
ipNOrder	O	2	Model order
ipCanLearn	O	1	Training capacity (0: no; 1: yes)
ipDimension	O	0	Parameter vector size
ipNbLayer	O	0	Model layer number
ipClass	O	3	Model pilot class index(note 1)
ipNHidden	O	0	Hidden nodes number
ipBootStrap	IO	3	Bootstrap number
ipStatus	O	0	Model status (0 ou error code)
ipNNeuron	O	0	Model node number
ipNModelInput	O	0	Model input number before eventual size reduction
ipNSynapse	O	0	Model parameter number.
ipNData	O	0	Training data number de
ipTrainCost	IO	2/0	Cost function index (note 2)
ipEndTrain	O	0	Training stop reason (note 3)

ipDataLoaded	O	0	Training data are loaded (0: no; 1: yes)
ipIsLin	O	0	Parameters linear model (0: no; 1: yes)
ipCommentCount	O	0	Model stored comments number
ipModelCount	O	3	Model count in multi models
ipNFuncAct	O	0	Available activation function number.
ipNFuncCost	O	0	Available cost function number.
ipNFuncProx	O	3	Available proximity function number (Kohonen models) (note 4)
ipNFuncDecay	O	3	Available decreasing function number (Kohonen models) (note 5)
ipValidHandle	O	0	Valid handle (0: no; 1: yes)
ipBin	IO	2/2	Binary storing forcing.
ipNTrainModel	O	2	Stored models stored number for multi training.
ipBootStrapType	IO	2/2	Bootstrap type (note 6)
ipTrainingAlgorithm	I	0/0	Training algorithm (note 7)
ipSelectedOutput	IO	0/0	Selected output (default 0)
ipProximity	IO	0/0	Proximity function index (Kohonen models)
ipDecay	IO	0/0	Decay function index (Kohonen models)
ipProximityDecay	IO	0/0	Distance decay function index (Kohonen models)
ipReverseMode	IO	0/0	Reverse mode (note 8).
ipBestIndex	O	0	Best training index, following the criterium
ipCriterium	IO	0/0	Model selection criterium. (note 9)
ipMultiTrainCount	O	2	Train count for multi train procedure
ipNTest	IO	0/0	Test set size
ipTestGroup	IO	0/0	Grouped test data
ipSaveTemp	IO	1/1	Do save on disk the temporary results
ipLoadTrainedModel	O	1	Load the trained model of index.

## Notes :

## 1. Model pilot class index :

- 1 Usage
- 2 Static training
- 3 Dynamic training
- 4 Reverse mode
- 11 Linear symetric matrix
- 12 Linear square matrix
- 13 Linear matrix
- 14 Linear vector

## 2. Available cost functions index :

- 0 Null cost
- 1 Error square
- 2 Delta values square
- 3 Lessed delta values square
- 4 Weighted squared error
- 5 Badly classified
- 6 bascule
- 7 Relative error square
- 8 Gaussian error square

- 9 Crossed entropy
- 10 Exponential square error

3. Training stop reason :

- 0 Current training
- 1 Reached objective
- 2 Epoch number reached
- 3 Computation accuracy
- 4 Strong parameter
- 5 Computation error
- 6 User ask
- 7 Best validation

4. Available proximity function index :

- 0  $1/(1 + \text{Distance})$
- 1  $1/\text{sqrt}(1 + \text{Distance})$
- 2  $\text{Exp}(-\text{Distance})$

5. Available decreasing function index :

- 0 (Ratio < 1)
- 1  $\text{Exp}(-\text{Ratio}/2)$

6. Bootstrap type :

- 0: nothing (default)
- 1: Standard
- 2: Residual.

7: Training algorithm :

- 0: Gradient; (default)
- 1: BFGS;
- 2: Levenberg-Marquardt.

8. Reverse modes :

- 0: icNone nothing
- 1: icMinimisation output minimization
- 2: icMaximisation output maximization
- 3: icReverse target-output delta minimization
- 4: icReverseSqrMin target-output delta minimization, while minimizing the squared inputs.
- 5: icReverseSqrDeltaMin target-output delta minimization, while minimizing the inputs modifications.

9. Model selection criteria

Value	Name	Meaning
0	Tcr_Cost	Training cost
1	Tcr_Test	Test cost
2	Tcr_Press	PRESS
3	Tcr_Mu	Homogeneity
4	Tcr_StdDevBiasLess	Biasless standard deviation
5	Tcr_R2	R2
6	Tcr_R2Adjust	Adjusted R2
7	Tcr_Correl	Correlation

8	Tcr_DeltaRank	Jacobien rank defect
9	Tcr_Determinant	Determinant
10	Tcr_EigenMin	Higher Dispersion Matrix Eigen value
11	Tcr_EigenMax	Lower Dispersion Matrix Eigen value
12	Tcr_Conditionning	Ratio of the two upper
13	Tcr_Trace	Dispersion Matrix Trace
14	Tcr_CorrelMax	Maximum of parameters absolute correlation
15	Tcr_CorrelMean	Mean of parameters absolute correlation

## 2.2.13 TRealPilot

### C / C++ declaration

```
typedef enum {rpNone, rpNoiseVar, rpInputPotential, rpMaxError,
rpTrainAccuracy, rpStdDev, rpPress, rpMu, rpTrainParam, rpCost,
rpIniDev, rpScale, rpConfLevel, rpWeighting, rpBootStrapAccuracy}
TRealPilot;
```

### Pascal declaration

```
TRealPilot = (rpNone, rpNoiseVar, rpInputPotentiel, rpMaxError,
rpTrainAccuracy, rpStdDev, rpPress, rpMu, rpTrainParam, rpCost,
rpIniDev, rpScale, rpConfLevel, rpWeighting, rpBootStrapAccuracy);
```

### Comments

[RealTable](#) <sup>[25]</sup>

Used with [GetRealValue](#) <sup>[32]</sup> and [SetRealValue](#) <sup>[38]</sup>.

### 2.2.13.1 RealTable

value	In/Out	level(s)	index	signification
rpNoiseVar	O	0	NA	Noide variance
rpInputPotentiel	I/O	3/3	Input	Indexed node potential
rpMaxError	O	0	NA	Maximum residual
rpTrainAccuracy	I/O	3/0	NA	Training accuracy
rpStdDev	O	0	NA	Training standard deviation
rpPress	O	0	NA	PRESS
rpMu	O	0	NA	Homogeneity
rpTrainParam	I/O	3/1	NA	Training cost function parameter
rpCost	O	0	CRIndex	Training cost (CRI a 1: with weighting; CRI a 2: Test)
rpIniDev	I/O	0	Biais 0	Parameter initialization factors
rpScale	O	0/0	NA	Normalization scale
rpConfLevel	I/O	2/2	output	Confidence level
rpWeighting	I/O	2/2	data	Indexed data weighting
rpBootStrapAccura	I/O	3/3	NA	Boostrap accuracy

cy

## 2.2.14 TSynapseData

### C / C++ declaration

```
typedef struct
{
    long Size;
    short OriginLayer;
    short OriginIndex;
    short TargetLayer;
    short TargetIndex;
    short LinkStyle;
    short Inputshort;
    short Outputshort;
    double Value;
    char SName[256];
    long NetIndex;
    long SynIndex
} TSynapseData;
```

### Pascal declaration

```
PSynapseData ^TSynapseData;
TSynapseData = record
    Size: LongInt;
    OriginLayer: SmallInt;
    OriginIndex : SmallInt;
    TargetLayer: SmallInt;
    TargetIndex: SmallInt;
    LinkStyle: SmallInt;
    InputBool: SmallInt;
    OutputBool: SmallInt;
    Value: Double;
    SName: Array[0..255] of Char;
    NetIndex: LongInt;
    SynIndex: LongInt;
end;
```

[SynapseDataTable](#) <sup>26</sup>

[GetSynapseInfo](#) <sup>43</sup>

[ModifyModel](#) <sup>49</sup>

### 2.2.14.1 SynapseDataTable

Fields meaning :

OriginLayer	Origin layer
OriginIndex	Origin node index in its layer
TargetLayer	Target layer
TargetIndex	Target node index in its layer
LinkStyle	Link style
InputBool	Boolean inputs

OutputBool	Boolean outputs
Value	Parameter value
SName	Link name
NetIndex	Model index in multi model
SynIndex	Synaptic link index

## 2.2.15 TTransfer

### C / C++ declaration

```
typedef enum {
    trSimple,           // simple transfer
    trConfidence,      // transfer with confidence
    trGradLev,         // transfer with gradient and leverage
    trGradDir,         // transfer with direct gradient
    trGrads,           // transfer with gradient and input gradient
    trGradHess,        // transfer with gradient and hessian
    trInGradHess,      // transfer with input gradient and input hessian
    trMixHess,         // transfer with mixt hessian
    trBootstrap,       // Transfer with bootstrap
    trKohonen          // Kohonen transfer
} TTransfer;
```

### Pascal declaration

```
TTransfer = (
    trSimple,           // simple transfer
    trConfidence,      // transfer with confidence
    trGradLev,         // transfer with gradient and leverage
    trGradDir,         // transfer with direct gradient
    trGrads,           // transfer with gradient and input gradient
    trGradHess,        // transfer with gradient and hessian
    trInGradHess,      // transfer with input gradient and input hessian
    trMixHess,         // transfer with mixt hessian
    trBootstrap,       // Transfer with bootstrap
    trKohonen          // Kohonen transfer
);
```

### Comments

[TransferTable](#)<sup>[27]</sup>  
[TransferTableFD](#)<sup>[28]</sup>  
Used with [Transfer](#)<sup>[39]</sup> and [TransferFD](#)<sup>[48]</sup>

### 2.2.15.1 TransferTable

Style	Level	V0	V1	V2	V3	Index	Signification
trSimple	0	input	output	NA	NA	NA	Simple transfer

trConfidence	0	input	output	confidence	NA	NA	transfer with confidence interval computation
trGradLev	2	input	output	levier	NA	output	transfer with leverage computation
trGradDir	3	input	output	gradient	NA	output	transfer with direct gradient computation
trGrads	2	input	output	gradient	NA	output	transfer with inputs relative gradient computation
trGradHess	2	input	output	gradient	hessian	output	transfer with gradient and hessian computation
trInGradHess	2	input	output	grad/ln	hess/ln	output	transfer with inputs relative gradient and hessian computation.
trMixHess	3	input	output	mix-hess	NA	output	transfer with mixt hessian
trBootstrap	3	input	output	NA	NA	0	transfer with bootstrap -> all the results
trBootstrap	3	input	moyenne	IC+	IC-	1	transfer with bootstrap -> synthetised
trKohonen	2	input	output	distance2	Coord	NA	transfer of a kohonen model

### 2.2.15.2 TransferTableFD

Style	Level	V0	V1	V2	V3	Index	Meaning
trGradLev	2	input	output	levier	NA	output	transfer with leverage computation
trGradHess	2	input	output	gradient	hessian	output	transfer with gradient and hessian computation
trInGradHess	2	input	output	grad/ln	hess/ln	output	transfer with input relative gradient and hessian
trMixHess	3	input	output	mix-hess	NA	output	transfer with mixt hessian

### 2.2.16 Other types

#### Pascal declaration

```

PLongInt = ^LongInt;
PDouble = ^Double;
ArrayOfDouble = Array [0..1023] of Double;
PArrayOfDouble = ^ArrayOfDouble;

```

## 2.3 Constants

[Differential loop](#) 
  
[Activation functions](#) 

### 2.3.1 Differential loop

- dl\_Direct = 0
- dl\_RK2 = 1
- dl\_RK4 = 2

[ModifyModel](#)<sup>[49]</sup>

### 2.3.2 Activation functions

af_Identity	0	Identity
af_Unit	1	Unit
af_Tanh	2	Hyperbolic tangent
af_Step	3	Step
af_Gauss	4	Gauss
af_Sin	5	Sine
af_Exp	6	Exponential
af_Quad	7	Quadratic ( $X^2/2$ )
af_Atng	8	Inverse tangent
af_M0	9	Moment 0 ( $1/(1+X^2)$ )
af_M1	10	Moment 1 ( $X/(1+X^2)$ )
af_Rev	11	Reverse
af_Log	12	Logarithm
af_Cubic	13	Cubic( $X^3/6$ )
af_QAbs	14	Quasi Absolute
af_Sig	15	Sigmoid
af_Compl	16	Complement to 1
af_Root	17	Root
af_InvHS	18	Inverse Hyperbolic Sine
af_sqr	19	Square ( $X^2$ )
af_Cos	20	Cosine
af_Erf	21	Error Function (Integral of Gaussian)

[TNeuronData](#)<sup>[18]</sup>

## 2.4 Primitives

[CloneModel](#)<sup>[49]</sup>

[CreateDataAnalyser](#)<sup>[41]</sup>

[CreateInputOptimizerNet](#)<sup>[45]</sup>

[CreateNewModel](#)<sup>[31]</sup>

[DLLQueryingToken](#)<sup>[40]</sup>

[DestroyDriver](#)<sup>[31]</sup>

[FunctionCompute2D](#)<sup>[42]</sup>

[FunctionCompute3D](#)<sup>[42]</sup>

[GetInfo](#)<sup>[32]</sup>

[GetNeuroneInfo](#)<sup>[43]</sup>  
[GetOptimizerHandle](#)<sup>[46]</sup>  
[GetRealValue](#)<sup>[32]</sup>  
[GetSynapseInfo](#)<sup>[43]</sup>  
[GetText](#)<sup>[33]</sup>  
[GetVector](#)<sup>[33]</sup>  
[HiddenLayerAnalysis](#)<sup>[44]</sup>  
[InputGrammSchmidt](#)<sup>[44]</sup>  
[LicenseLevel](#)<sup>[34]</sup>  
[LoadModelFromFile](#)<sup>[34]</sup>  
[LoadOptimizer](#)<sup>[45]</sup>  
[LoadParameters](#)<sup>[40]</sup>  
[ModelAction](#)<sup>[34]</sup>  
[ModifyModel](#)<sup>[49]</sup>  
[MultiTrain](#)<sup>[45]</sup>  
[NewOptimizer](#)<sup>[46]</sup>  
[OpenLibrary](#)<sup>[35]</sup>  
[OptimizeInputs](#)<sup>[46]</sup>  
[OptimizeInputsEx](#)<sup>[47]</sup>  
[SaveModel](#)<sup>[40]</sup>  
[SaveParameters](#)<sup>[41]</sup>  
[SetIValue](#)<sup>[35]</sup>  
[SetNewDataProc](#)<sup>[50]</sup>  
[SetRandSeed](#)<sup>[36]</sup>  
[SetRealValue](#)<sup>[36]</sup>  
[SetText](#)<sup>[36]</sup>  
[SetTrainingSet](#)<sup>[37]</sup>  
[SetVector](#)<sup>[37]</sup>  
[TestFile](#)<sup>[38]</sup>  
[Train](#)<sup>[38]</sup>  
[Transfer](#)<sup>[39]</sup>  
[TransferData](#)<sup>[47]</sup>  
[TransferFD](#)<sup>[48]</sup>

## 2.4.1 Level 0

### 2.4.1.1 CloseLibrary

#### C/C++ declaration

```
extern long __stdcall CloseLibrary;
```

#### Pascal declaration

```
function CloseLibrary: Integer; stdcall;
```

Close the access to NDK library and free the license token. This function must be called before the closing of the calling program.

Return → 0 si OK.

[OpenLibrary](#)<sup>[35]</sup>  
[Primitives](#)<sup>[29]</sup>

### 2.4.1.2 CreateNewModel

#### C/C++ declaration

```
extern long __stdcall CreateNewModel(struct TModelData[16] ModelData,
    long Trainable);
```

#### Pascal declaration

```
Function CreateNewModel(ModelData : TModelData[16]; Trainable): LongInt;
stdcall;
```

*New model creation. The [TModelData](#)<sup>[16]</sup> structure is a description of the model to build :*

```
IN      → ModelData : model description.
IN      → Trainable : 0, no training, 1, with training
Return  → Model handle, <= 0 if failed.
```

[Transfer](#)<sup>[39]</sup>

[LoadModelFromFile](#)<sup>[34]</sup>

[GetOptimizerHandle](#)<sup>[46]</sup>

[GetVector](#)<sup>[33]</sup>

[CopyDriver](#)<sup>[49]</sup>

[Primitives](#)<sup>[29]</sup>

### 2.4.1.3 DestroyModel

#### C/C++ declaration

```
extern long __stdcall DestroyModel(long Handle);
```

#### Pascal declaration

```
Function DestroyModel(Handle: LongInt): LongInt; stdcall;
```

*Model and pilot destroying. The memory is freed.*

```
IN      → Handle : Model handle
return  → 0 if OK, -1 if failed.
when returned, Handle variable is undetermined.
```

**Remarque :** *This function must applied to all kinds of model, created with [CreateNewModel](#)<sup>[31]</sup>, [LoadModelFromFile](#)<sup>[34]</sup>, [CreateDataAnalyser](#)<sup>[41]</sup>, [LoadOptimizer](#)<sup>[45]</sup> or [CloneModel](#)<sup>[49]</sup>.*

[Primitives](#)<sup>[29]</sup>

### 2.4.1.4 GetIndexInfo

#### C/C++ declaration

```
long __stdcall GetIndexInfo(long Handle, int Index, TIndexPilotInfo[11]
    Style);
```

#### Pascal declaration

```
function GetIndexInfo(Handle: LongInt; Index: Integer; Style:
TIndexPilotInfo[11]): LongInt; stdcall;
```

*Getting integer indexed information from the object :*

IN → Handle : Model handle.  
 IN → Index: Research index  
 Style → [TIndexPilotInfo](#)<sup>[11]</sup> type of requested information.  
 Return → Requested information. -1 if failed

[TIndexInfoTable](#)<sup>[11]</sup>  
[Primitives](#)<sup>[28]</sup>

### 2.4.1.5 GetInfo

#### C/C++ declaration

```
extern long __stdcall GetInfo(long Handle, TPilotInfo[22] Style);
```

#### Pascal declaration

```
Function GetInfo(Handle: LongInt; Style: TPilotInfo[22]): LongInt;
stdcall;
```

*Getting integer information from the object :*

IN → Handle : Model handle.  
 Style → [TPilotInfo](#)<sup>[22]</sup> type of requested information.  
 Return → Requested information. -1 if failed

[InfoTable](#)<sup>[22]</sup>  
[Primitives](#)<sup>[28]</sup>

### 2.4.1.6 GetRealValue

#### C/C++ declaration

```
long __stdcall GetRealValue(long Handle, int Index, double *Value,
TRealPilot[25] style);
```

#### Pascal declaration

```
function GetRealValue(Handle: LongInt; Index: Integer; var Value:
NReal; Style: TRealPilot[25]): LongInt; stdcall;
```

*Getting real indexed information from the object :*

IN → Handle : Model handle.  
 IN → Index: Research index  
 OUT → Value: Requested value  
 Style → [TRealPilot](#)<sup>[25]</sup> type of requested information.

[TRealPilot](#)<sup>[25]</sup>

[RealTable](#)<sup>[25]</sup>  
[SetRealValue](#)<sup>[36]</sup>  
[Primitives](#)<sup>[29]</sup>

### 2.4.1.7 GetText

#### C/C++ declaration

```
long __stdcall GetText(long Handle, int Ind0, int Ind1, TNText[20]
Style, char* Dest);
```

#### Pascal declaration

```
function GetText(Handle: LongInt; Ind0, Ind1: Integer; Style: TNText[20]
; Dest: PChar): LongInt;
```

*Reading a text on the model.*

IN → Handle : Model handle.  
 IN → Ind0 : Index 0 of the text (cf. [TNText](#)<sup>[20]</sup>)  
 IN → Ind1 : Index 1 of the text (cf. [TNText](#)<sup>[20]</sup>)  
 IN → Dest : Destination buffer.  
 Return → If Dest is Null, then return the requested buffer size.  
 Else, return 0 if OK, or an error code if failed.

[TNText](#)<sup>[20]</sup>  
[TextTable](#)<sup>[20]</sup>  
[SetText](#)<sup>[36]</sup>  
[Primitives](#)<sup>[29]</sup>

### 2.4.1.8 GetVector

#### C/C++ declaration

```
long __stdcall GetVector(long Handle, void* Target, int Index, TData[9]
Style);
```

#### Pascal declaration

```
function GetVector(Handle: LongInt; Target: Pointer; Index: Integer;
Style: TData[9]): Integer; stdcall;
```

*Reading a vector from the model.*

IN → Handle : Model handle.  
 IN → Target : Destination buffer (cf. [TData](#)<sup>[9]</sup>). It is a table of double or long, depending on the style.  
 IN → Index : Vector index (cf. [TData](#)<sup>[9]</sup>)  
 IN → Style Vector to read  
 Return → If Target is Null, the vector length (Element number).  
 otherwise, 0 if OK or error code.

[TDataTable](#)<sup>[10]</sup>  
[SetVector](#)<sup>[37]</sup>

### 2.4.1.9 LicenseLevel

#### C/C++ declaration

```
extern long __stdcall LicenseLevel();
```

#### Pascal declaration

```
Function LicenseLevel: LongInt; stdcall;
```

*Reading the license level.*

Return	→	License level 1, 2 ou 3
		0 Use the free version.
		-1 no installation
		-2 bad installation
		-3 evaluation overtimed
		-4 validity outdated
		-5 no network card.

[Primitives](#)<sup>29</sup>

### 2.4.1.10 LoadModelFromFile

#### C/C++ declaration

```
extern long __stdcall LoadModelFromFile(char * FileName, long  
Trainable);
```

#### Pascal declaration

```
Function LoadModelFromFile(FileName: PChar; Trainable): LongInt;  
stdcall;
```

*Creating a model from a file*

IN	→	FileName : File name.
IN	→	Trainable : 0, no training, 1, with training
Return	→	Model handle, <= 0 if failed.

**Remark :** The model created with **LoadModelFromFile** can be used, depending on the license level, to perform transfer, with or without leverage computation. It can be used in a training process only if **Trainable** is not 0..

[Transfer](#)<sup>47</sup>

[NewOptimizer](#)<sup>46</sup>

[CloneModel](#)<sup>49</sup>

[Primitives](#)<sup>29</sup>

### 2.4.1.11 ModelAction

#### C/C++ declaration

```
extern long __stdcall ModelAction(long Handle, TModelAction16 Style);
```

**Pascal declaration**

```
function ModelAction(Handle: LongInt; Style: TModelAction[16]): LongInt;
stdcall;
```

*Action of the model, depending on the style .:*

```
IN      → Handle      Model handle.
IN      → Style[16]    Action to perform.
Return  → 0 if OK, <> 0 if failed
```

[ModelActionTable](#)<sup>[16]</sup>

[TModelAction](#)<sup>[16]</sup>

[Primitives](#)<sup>[29]</sup>

**2.4.1.12 OpenLibrary****C/C++ declaration**

```
extern long __stdcall OpenLibrary(long CallerHndl);
```

**Pascal declaration**

```
function OpenLibrary(CallerHndl: LongInt): Integer; stdcall;
```

*Open the NDK DLL, and look for the token if necessary. Must be called before any other call.*

```
IN      → CallerHndl : Calling window handle, or 0.
Return  → License level or error code..
```

[CloseLibrary](#)<sup>[30]</sup>

[Primitives](#)<sup>[29]</sup>

**2.4.1.13 SetIValue****C/C++ declaration**

```
long __stdcall SetIValue(long Handle, int Value, TPilotInfo[22] Style);
```

**Pascal declaration**

```
function SetIValue(Handle: LongInt; Value: Integer; Style: TPilotInfo[22]): LongInt; stdcall;
```

*Writing an integer information on the object :*

```
IN      → Handle : Model handle.
Style   → TPilotInfo[22] type of information to write.
Return  → 0 if OK
```

[InfoTable](#)<sup>[22]</sup>

[Primitives](#)<sup>[29]</sup>

#### 2.4.1.14 SetRandSeed

##### C/C++ declaration

```
extern long __stdcall SetRandSeed(long Seed);
```

##### Pascal declaration

```
Function SetRandSeed(Seed: LongInt): LongInt; stacall;
```

*Random generator initialization.*

IN → Seed : Random generator seed

[Primitives](#)<sup>[29]</sup>

#### 2.4.1.15 SetRealValue

##### C/C++ declaration

```
extern long __stdcall SetRealValue(long Handle, int Index, double Value, style: TRealPilot[28]);
```

##### Pascal declaration

```
function SetRealValue(Handle: LongInt; Index: Integer; Value: NReal; style: TRealPilot[28]): LongInt; stdcall;
```

*Writing real indexed information to the object :*

IN → Handle : Model handle.  
 IN → Index: Research index  
 OUT → Value: value to write  
 Style → [TRealPilot](#)<sup>[28]</sup> type of information.  
 Return → 0 if OK.

[TRealPilot](#)<sup>[28]</sup>

[RealTable](#)<sup>[28]</sup>

[GetRealValue](#)<sup>[28]</sup>

[Primitives](#)<sup>[29]</sup>

#### 2.4.1.16 SetText

##### C/C++ declaration

```
long __stdcall SetText(long Handle, int Ind0, int Ind1, TNText[20] Style, char* Source);
```

##### Pascal declaration

```
function SetText(Handle: LongInt; Ind0, Ind1: Integer; Style: TNText[20]; Source: PChar): LongInt;
```

*Writing a text on the model.*

IN → Handle : Model handle.  
 IN → Ind0 : Index 0 of the text (cf. [TNText](#)<sup>[20]</sup>)  
 IN → Ind1 : Index 1 of the text (cf. [TNText](#)<sup>[20]</sup>)

IN → Dest : Source buffer.  
 Return → Return 0 if OK, or an error code if failed.

[TNTxt](#)<sup>[20]</sup>  
[TextTable](#)<sup>[20]</sup>  
[GetText](#)<sup>[33]</sup>  
[Primitives](#)<sup>[29]</sup>

### 2.4.1.17 SetTrainingSet

#### C/C++ declaration

```
extern long __stdcall SetTrainingSet(long Handle, char * DataFileName,
long FirstRow, long LastRow, char Sep);
```

#### Pascal declaration

```
Function SetTrainingSet(Handle: LongInt; DataFileName PChar; FirstRow,
LastRow: LongInt; Sep: Char): LongInt; stdcall;
```

#### Training data definition.

IN → DataFileName : ASCII formatted data file name.  
 IN → FirstRow : First data line in the file (First line is quoted 1).  
 IN → LastRow : Last data line in the file.  
 IN → Sep : Data separator.  
 Return → 0 si OK, -1 if invalid handle, 1 if model is not training capable

**Remark :** Getting memory for training data, and, if DataFileName is valid, data file definition. In the file, data must be ordered by columns. First columns are inputs, then outputs. If a preprocessing function hve been defined with SteDataFunc, data will be pre-processed. Data can then be loaded directly by using ModelActon with the style maLoadTrainingData. The data can also be loaded line by line by using the function [SetVector](#)<sup>[37]</sup> with the style **TrainingData**.

[CreateNewModel](#)<sup>[31]</sup>  
[LoadModelFromFile](#)<sup>[34]</sup>  
[Primitives](#)<sup>[29]</sup>

### 2.4.1.18 SetVector

#### C/C++ declaration

```
long __stdcall SetVector(long Handle, void* VSource, int Index, TData
Style);
```

#### Pascal declaration

```
function SetVector(Handle: LongInt; VSource: Pointer; Index: Integer;
Style: TData): Integer; stdcall;
```

#### writing a vector to the model.

IN → Handle : Model handle.

IN → Target : Destination buffer (cf. [TData](#)<sup>[9]</sup>). It is a table of double or long, depending on the style.  
 IN → Index : Vector index (cf. [TData](#)<sup>[9]</sup>)  
 IN → Style Vector to write  
 Return → 0 if OK or error code.

[TDataTable](#)<sup>[10]</sup>  
[GetVector](#)<sup>[33]</sup>

### 2.4.1.19 TestFile

#### C/C++ declaration

```
extern long __stdcall TestFile(char *FileName);
```

#### Pascal declaration

```
function TestFile(FileName: PChar): Integer; stdCall;
```

*Analyse a file in order to determine if it contains an object readable by NDK.*

IN → FileName: file name or character string to test.  
 Return → 0: unknown  
           1: binary model file  
           2: ASCII model file  
           3: XML model file  
           4: XML model string  
           5: XML meta model file  
           6: Compiled model file  
           7: MGL file  
           8: MGL string  
          128: binary parameter file  
          129: ASCII parameter file  
          130: XML parameter file  
          131: XML parameter string.  
          136: ASCII parameter string

[LoadModelFromFile](#)<sup>[34]</sup>  
[LoadParameters](#)<sup>[40]</sup>  
[Primitives](#)<sup>[29]</sup>

### 2.4.1.20 Train

#### C/C++ declaration

```
extern long __stdcall Train(long Handle, long* Number, long  
TrainStyle, TCallbackFct[8]* ACallback);
```

#### Pascal declaration

```
function Train(Handle: LongInt; var Number: Integer; TrainStyle:  
Integer; ACallback: TCallbackFct[8]): LongInt; stdcall;
```

*Training algorithm has been fixed with [SetValue](#)<sup>[35]</sup> with style ipBootStrapType.  
 Launch a training*

IN → Handle : Model handle.

IN/OUT → Number: long  
           IN → number or requested epochs  
           OUT → number of actual epochs

IN → TrainStyle: Combination of constants :

(0) TrS_Std	Standard training
(1) TrS_InitParam	Training with parameters initialization at the first cycle.
(2) TrS_Quiet	Quiet training (no info send with ACallBack)
(3) TrS_leverage	leverage computation after training.
(4) TrS_Bootstrap	Bootstrap training from an already trained model.

The number of bootstrap is fixed with fonction [SetValue](#)<sup>[35]</sup> with style "ipBootstrap".

The bootstrap type is fixed with [SetValue](#)<sup>[35]</sup> with style "ipBootstrapType".

Available types are :

btStd (1)	: standard bootstrap
btResidual (2)	: residual bootstrap.

IN → ACallBack: [Callback function](#)<sup>[8]</sup>

Return → 0 if OK, else error code:  
           -1 invalid handle

**Remark :** Before using this function, the data must be loaded. parameters may be initialized. In order to follow the training process, use the callback function and follow the returned values.

[CreateNewModel](#)<sup>[31]</sup>  
[LoadModelFromFile](#)<sup>[34]</sup>  
[Primitives](#)<sup>[29]</sup>

### 2.4.1.21 Transfer

#### C/C++ declaration

```
long __stdcall Transfer(long Handle, double* V0, double* V1, double* V2, double* V3, int Index, TTransfer[27] Style);
```

#### Pascal declaration




```
function Transfer(Handle: LongInt; V0, V1, V2, V3: PArrayOfReal; Index: Integer; Style: TTransfer[27]): LongInt;
```

*Transfer through the model. The actual transfer depends upon the style.*

IN → Handle : Model handle.  
 IN/OUT → V0, V1, V2, V3 : Data tables for input and output.  
 IN → Index : depends on style  
 IN → Style  
 Return → 0 if computation OK  
           < 0 if invalid handle  
           > 0 if computation error during transfer.

**Remarque :** Les dimensions doivent être vérifiées avec la fonction GetInfo. L'utilisateur est responsable de fournir suffisamment de datas en input, ainsi que de la dimension des buffers de destination.

[TTransfer](#)<sup>[27]</sup>

[TransferTable](#)   
[TransferFD](#)   
[Primitives](#) 

## 2.4.2 Level 1

### 2.4.2.1 DLLQueryingToken

#### C/C++ declaration

```
long __stdcall DLLQueryingToken();
```

#### Pascal declaration

```
function DLLQueryingToken: Integer; stdcall;
```

*Return 1 as long as the library is looking for a license token.*

### 2.4.2.2 LoadParameters

#### C/C++ declaration

```
long __stdcall LoadParameters(long Handle, char *FileName);
```

#### Pascal declaration

```
function LoadParameters(Handle: LongInt; FileName: PChar): LongInt;  
stdcall;
```

*Loading a parameter vector for a neural model*

IN → Handle : Model handle.  
IN → FileName : target file name.

Formats can be tested with [TestFile](#)  :

- NML extension "nml" ou "xml".
- ASCII extension "txt"

Can be called at any time. The vector saved in the file must have the right size. It can be a TRealVector type vector or a TRealByteVector type vector. in the last case, the byte associated with a parameter value must be different of 0 in order to be trainable.

**Remarque** : No check of the path..

[Primitives](#) 

### 2.4.2.3 SaveModel

#### C/C++ declaration

```
long __stdcall SaveModel(long Handle, char* FileName);
```

#### Pascal declaration

```
function SaveModel(Handle: LongInt; FileName: PChar): LongInt;  
stdcall;
```

*Saving the current model in a file:*

IN → Handle : Model handle.  
 IN → FileName : File name.

**Remark :** No check on the path. No check on eventual exiting file..

[Primitives](#)<sup>29</sup>

#### 2.4.2.4 SaveParameters

##### **C/C++ declaration**

```
long __stdcall SaveParameters(long Handle, char* FileName);
```

##### **Pascal declaration**

```
function SaveParameters(Handle: LongInt; FileName: PChar): LongInt;  
stdcall;
```

*Saving the current parameters in a file:*

IN → Handle : Model handle.  
 IN → FileName : File name.

Format can be :

- NML extension "nml" ou "xml".
- ASCII extension "txt"

**Remark :** No check on the path. No check on eventual exiting file..

### 2.4.3 Level 2

#### 2.4.3.1 CreateDataAnalyser

##### **C/C++ declaration**

```
long __stdcall CreateDataAnalyser(long NInput, long NOutput);
```

##### **Pascal declaration**

```
function CreateDataAnalyser(NInput, NOutput: Integer): LongInt;  
stdcall;
```

*Create a data analysis pilot*

IN → NInput Model input number  
 IN → NOutput Model output number  
 Return → Pilot handle, 0 if failed.

**Remarque :** Necessary before calling [InputGrammSchmidt](#)<sup>44</sup>.

[InputAnalysis](#)<sup>44</sup>  
[Primitives](#)<sup>29</sup>

### 2.4.3.2 FunctionCompute2D

#### C/C++ declaration

```
long __stdcall FunctionCompute2D(long Handle, double* VEntreeOrigine,
int RangEntree, int RangOutput, int RangEntreeDerive, int Diviseur,
int Dim, double* EntreesX, double* TargetY, double* TargetDerivative,
double* TargetYConf, double LeverageThreshold, int HideOverThreshold,
int* Code);
```

#### Pascal declaration

```
function FunctionCompute2D(Handle: LongInt; VEntreeOrigine: Pointer;
RangEntree, RangOutput, RangEntreeDerive, Diviseur, Dim: integer;
EntreesX, TargetY, TargetDerivative, TargetYConf: Pointer;
LeverageThreshold: NReal; HideOverThreshold: Boolean; var Code:
Integer): Integer; stdcall;
```

*Multiple computation in a model in order to draw a 2D curve.*

Handle :	Model handle.
VEntreeOrigine	Initial input vector (NInput)
RangEntree	Variable input index
RangOutput	Output index
RangEntreeDerive	Derivative variable input index.
Diviseur	Over sampling (dynamic)
Dim	Function points number
EntreesX	Successive Inputs (Dim)
TargetY	Computed output targets (Dim)
TargetDerivative	Computed derivative targets (Dim)
TargetYConf	Confidence interval targets (Dim)
LeverageThreshold	Confidence interval threshold for hiding computed values
HideOverThreshold	Boolean -> Values with a too high confidence interval are zeroed Zero value is hold by TargetY[0]
Code	Return code.

### 2.4.3.3 FunctionCompute3D

#### C/C++ declaration

```
long __stdcall FunctionCompute3D(long Handle, double* VEntreeOrigine,
int RangEntreeX, int RangEntreeY, int RangOutput, int
RangEntreeDerive, int Diviseur, int DimX, int DimY, double* EntreesX,
double* EntreesY, double* TargetZ, double* TargetDerivative, double*
TargetZLever, int* Code);
```

#### Pascal declaration

```
function FunctionCompute3D(Handle: LongInt; VEntreeOrigine: Pointer;
RangEntree: Integer2; RangOutput, RangEntreeDerive, Diviseur, DimX,
DimY: Integer; EntreesX, EntreesY, TargetZ, TargetDerivative,
TargetZLever: Pointer; var Code: Integer): Integer; stdcall;
```

*Multiple computation in a model in order to draw a 3D curve.*

Handle : Model handle.

VEntreeOrigine	Initial input vector (NInput)
RangEntree	Variable input index
RangOutput	Output index
RangEntreeDerive	Derivative variable input index.
Diviseur	Over sampling (dynamic)
DimX	X Function points number
DimY	Y Function points number
EntreesX	X successive Inputs (Dim)
EntreesY	Y successive Inputs (Dim)
TargetY	Computed output targets (Dim)
TargetDerivative	Computed derivative targets (Dim)
TargetYConf	Confidence interval targets (Dim)
LeverageThreshold	Confidence interval threshold for hiding computed values
HideOverThreshold	Boolean -> Values with a too high confidence interval are zeroed Zero value is hold by TargetY[0]

Code            Return code.

#### 2.4.3.4 GetNeuroneInfo

##### C/C++ declaration

```
extern int __stdcall GetNeuroneInfo(long Handle, struct TNeuronData[18]
* lParam);
```

##### Pascal declaration

```
Function GetNeuroneInfo(Handle: LongInt; lParam: PNeuronData[18]):
Integer;
```

*Lecture des informations d'un neurone.*

IN            → Handle : Model handle.  
IN/OUT       → lParam : Structure de description du neurone.  
Return        → 0 si OK

The [TNeuronData](#)<sup>[18]</sup> structure describe the targeted node. Befoare the call, LayerIndex and NodeIndex fields must be written. The other fields can be modified after the call, in order to call [ModifyModel](#)<sup>[49]</sup>.

##### [Primitives](#)<sup>[29]</sup>

[NeuronDataTable](#)<sup>[19]</sup>

[TNeuronData](#)<sup>[18]</sup>

[ModifyNetWork](#)<sup>[49]</sup>

#### 2.4.3.5 GetSynapseInfo

##### C/C++ declaration

```
extern int __stdcall GetSynapseInfo(long Handle, struct TSynapseData[26]
* lParam);
```

**Pascal declaration**

```
Function GetSynapseInfo(Handle: LongInt; lParam: PSynapseData[26]):
Integer;
```

*Reading a synapse values*

```
IN      → Handle : Model handle.
IN/OUT  → lParam : Synapse description record
Return  → 0 if OK, -1 otherwise.
```

**Remark :** The fields OriginLayer, OriginIndex, TargetLayer et TargetIndex must be filled before a call.

[Primitives](#)<sup>[29]</sup>[SynapseDataTable](#)<sup>[26]</sup>[TSynapseData](#)<sup>[26]</sup>[ModifyNetWork](#)<sup>[49]</sup>**2.4.3.6 HiddenLayerAnalysis****C/C++ declaration**

```
extern long __stdcall HiddenLayerAnalysis(long Handle, double*
Relevance);
```

**Pascal declaration**

```
function HiddenLayerAnalysis(Handle: LongInt; Relevance: Pointer):
LongInt; stdcall;
```

*Hidden layer analysis.*

```
IN      → Handle : Model handle.
OUT     → Relevance : Target double table for reading the relevance values of the hidden nodes.
Return  → 0 if OK, <> 0 if failed
```

**Remark :** User is responsible for dimensionning the target table.  
Must be called after training.

[CreateNewModel](#)<sup>[31]</sup>[LoadModelFromFile](#)<sup>[34]</sup>[Primitives](#)<sup>[29]</sup>**2.4.3.7 InputGrammSchmidt****C/C++ declaration**

```
extern long __stdcall InputGrammSchmidt(long Handle, double* Weight,
double* Relevance);
```

**Pascal declaration**

```
Function InputGrammSchmidt(Handle: LongInt; Weight, Relevance:
Pointer): LongInt; stdcall;
```

*Input data analysis.*

```
IN      → Handle : Model handle.
OUT     → Weight : Double table of the input weights.
OUT     → Relevance : Double table of the inputs relevance.
Return  → 0 if OK, <> 0 if failed
```

The selected output has been fixed with [SetValue](#)<sup>[38]</sup> with the style "ipSelectedOutput".

**Remark :** Weight et Relevance tables must be sized to NInput.  
Must be called with an appropriate handle coming from [CreateDataAnalyser](#)<sup>[41]</sup>.

[CreateDataAnalyser](#)<sup>[41]</sup>  
[Primitives](#)<sup>[29]</sup>

#### 2.4.3.8 LoadOptimizer

##### C/C++ declaration

```
long __stdcall LoadOptimizer(char * Fichier);
```

##### Pascal declaration

```
function LoadOptimizer(Fichier: PChar): LongInt; stdcall;
```

*Input optimizer pilot creation*

```
IN      → Fichier : Model file name
Return  → The created pilot handle.
```

**Remark :** This function create a model whose variables are the inputs. The training process will modify those inputs in order to get a desired result as outputs.

[OptimizeInputs](#)<sup>[46]</sup>  
[Primitives](#)<sup>[29]</sup>

#### 2.4.3.9 MultiTrain

##### C/C++ declaration

```
long __stdcall MultiTrain(long Handle, long NInit, long *number, long
Style, TCallbackFct* ACallback);
```

##### Pascal declaration

```
function MultiTrain(Handle: LongInt; NInit: LongInt; var number:
LongInt; Style: LongInt; ACallback: TCallbackFct): LongInt; stdcall;
```

*Lancement d'une salve d'apprentissages.*

```
IN      → Handle : Model handle.
IN      → NInit  : number of parameter initialization
```

IN → number : number of training cycles requested for each initialization  
 IN → Style : combination of styles :  
     (0) TrS\_Std               Standard training  
     (1) TrS\_InitParam       Training with parameters initialization at the first cycle.  
     (2) TrS\_Quiet            Quiet training (no info send with ACallBack)  
     (3) TrS\_leverage        leverage computation after training.  
 IN → ACallBack : Call back function.

### 2.4.3.10 NewOptimizer

#### C/C++ declaration

```
long __stdcall NewOptimizer(long Handle);
```

#### Pascal declaration

```
Function NewOptimizer(Handle: LongInt): LongInt;
```

*Create an optimizer driver handle with reference to a model output.*

IN → Handle : Model handle.  
 Return → Optimizer handle.

The selected output can be fixed by calling [SetValue](#)<sup>[35]</sup> with the style "ipSelectedOutput".  
 The computation mode can be fixed bt calling [SetValue](#)<sup>[35]</sup> with the style "ipReverseMode".

[CreateNewModel](#)<sup>[31]</sup>  
[LoadModelFromFile](#)<sup>[34]</sup>  
[Primitives](#)<sup>[29]</sup>

### 2.4.3.11 OptimizeInputs

#### C/C++ declaration

```
long __stdcall OptimizeInputs(long Handle, long *Number, double *OutputTarget);
```

#### Pascal declaration

```
function OptimizeInputs(Handle: LongInt; var Number: LongInt; OutputTarget: PDouble): LongInt; stdcall;
```

*Calcul inverse dans le réseau*

IN → Handle : Optimizer handle.  
 IN/OUT → Number    Epoch number  
 IN → OutputTarget Double table address de doubles with the target outputs.  
 Return → 0 if OK, else error code

The training algorithm can be fixed by calling [SetValue](#)<sup>[35]</sup> with the style "ipTrainingAlgorithm".  
 The computation mode can be fixed bt calling [SetValue](#)<sup>[35]</sup> with the style "ipReverseMode".  
 During the reverse computation, some of the inputs may be keep as constants. The fixed inputs may be fixed by calling the [SetVector](#)<sup>[37]</sup> with the style "\_BoolInputs" and along integer vector with everything set to 0, except the fixed inputs.

**Remark** : an optimizer handle has been obtained with one of the functions : [NewOptimizer](#)<sup>[46]</sup> or [LoadOptimizer](#)<sup>[45]</sup>. For such a handle, variables are the model inputs, and the model dimension is the number of inputs.

[Primitives](#)<sup>[29]</sup>  
[OptimizeInputsEx](#)<sup>[47]</sup>

### 2.4.3.12 OptimizeInputsEx

#### C/C++ declaration

```
long __stdcall OptimizeInputsEx(long Handle, long *Number, double
    *Input, long *InputBool, double *OutputTarget);
```

#### Pascal declaration

```
function OptimizeInputsEx(Handle: LongInt; var Number: LongInt; Input:
    PDouble; InputBool: PLongInt; OutputTarget: PDouble): LongInt;
stdcall;
```

#### Reverse model computation

IN	→ Handle :	Model handle.
IN	→ Mode	Reverse computation mode
IN/OUT	→ Number	Number of requested cycles. Return the actual number of cycles.
IN/OUT	→ Input	Double table holding the input values (initial values / returns values)..
IN	→ InputBool	Long integer table holding the boolean variability of inputs.
IN	→ OutputTarget	Double table holding the target outputs.
Return	→ 0 if OK, error code otherwise.	

The model handle used here is a standard handle. This is not the case for the handle used in [OptimizeInputs](#)<sup>[46]</sup>.

The training algorithm can be fixed by calling [SetValue](#)<sup>[35]</sup> with the style "ipTrainingAlgorithm".

The computation mode can be fixed by calling [SetValue](#)<sup>[35]</sup> with the style "ipReverseMode".

[Primitives](#)<sup>[29]</sup>  
[OptimizeInputs](#)<sup>[46]</sup>

### 2.4.3.13 TransferData

#### C/C++ declaration

```
long __stdcall TransferData(long Handle, double* Target, int
    LineLength, int MaxData, int IsExternal, TCallbackFct* ACallback);
```

#### Pascal declaration

```
function TransferData(Handle: LongInt; Target: PArrayOfReal;
    LineLength, MaxData, IsExternal: Integer; ACallback: TCallbackFct):
    LongInt; stdCall;
```

*Model data transfer.*

IN → Handle : Model handle.  
 IN/OUT → Target : Output double table.  
 IN → LineLength : Output table line length  
 IN → MaxData : Output table column length.  
 IN → IsExternal : External data ?  
                   0: inputs are coming from training data registered in the model  
                   1: inputs data are read in Target.

Return → 0 if OK  
           < 0 Invalid handle  
           > 0 Transfer computation error.

Returned values in an output line are, in the order :

Inputs  
 Target outputs  
 Computed outputs  
 Computed state variables  
 Leverages.

Line number is limited to training data number. If MaxData is negative and IsExternal is 0, then the line number is the number of training data.

**Remark :** Sizes must be verified with [GetInfo](#)<sup>[32]</sup>.

[CreateNewModel](#)<sup>[31]</sup>  
[LoadModelFromFile](#)<sup>[34]</sup>  
[Transfer](#)<sup>[39]</sup>  
[Primitives](#)<sup>[29]</sup>

#### 2.4.3.14 TransferFD

##### C/C++ declaration

```
long __stdcall TransferFD(long Handle, double* V0, double* V1, double*
V2, double* V3, int Index, double PrecDF, TTransfer[27] Style); long
__stdcall GenericTransfer(long Handle, double* V0,
```

##### Pascal declaration

```
function TransferFD(AHandle: LongInt; V0, V1, V2, V3: PArrayOfReal;
Index: Integer; PrecDF: NReal; Style: TTransfer[27]): LongInt;
```

*Transfer through the model. The actual transfer depends upon the style.*

*First and second order differentiation are performed by finite difference.*

IN → Handle : Model handle.  
 IN/OUT → V0, V1, V2, V3 : Data tables for input and output.  
 IN → Index : depends on style  
 IN → Style  
 Return → 0 if computation OK  
           < 0 if invalid handle  
           > 0 if computation error during transfer.

**Remark :** Size can be checked with [GetInfo](#)<sup>[32]</sup> function. The user is responsible of giving enough

room to store the resulting data.

[TTransfer](#)<sup>[27]</sup>  
[TransferTableFD](#)<sup>[28]</sup>  
[Transfer](#)<sup>[39]</sup>  
[Primitives](#)<sup>[29]</sup>

## 2.4.4 level 3

### 2.4.4.1 CloneModel

#### C/C++ declaration

```
extern long __stdcall CloneModel(long Handle);
```

#### Pascal declaration

```
Function CloneModel(Handle: LongInt): LongInt; stdcall;
```

*Duplicate a model and create a new driver for it..*

IN → Handle : Model handle.

[Primitives](#)<sup>[29]</sup>

### 2.4.4.2 ModifyModel

#### C/C++ declaration

```
extern int __stdcall ModifyModel(long Handle, long Param, TModifyModel  
Style);
```

#### Pascal declaration

```
Function ModifyModel(Handle: LongInt; Param: LongInt; Style:  
TModifyModel): Integer;
```

*This function modify the model.*

IN → Handle : Model handle.

IN → Param : Parameterd defined depending upon the value of Style.

IN → Style : define the actual modification..

Table of the possible actions, depending upon [style](#)<sup>[17]</sup>:

[ModifyModelTable](#)<sup>[18]</sup>

#### Notes:

1. Before modifying a neuron with mnModifyNeuron, it is better to read the informations of the target neuron with the function [GetNeuroneInfo](#)<sup>[43]</sup>. The [TNeuronData](#)<sup>[18]</sup> record is then written and has only to be modified. After the modification, a new call to [GetNeuroneInfo](#)<sup>[43]</sup> will verify the action.
2. Before modifying a synapse with mnModifySynapse, it is better to read the target synapse information with the function [GetSynapseInfo](#)<sup>[43]</sup>. The [TSynapseData](#)<sup>[26]</sup> record is then written and

has only to be modified. After the modification a new call to [GetSynapseInfo](#)<sup>[43]</sup> will verify the action.

[NeuronDataTable](#)<sup>[19]</sup>  
[SynapseDataTable](#)<sup>[26]</sup>  
[TModifyModel](#)<sup>[17]</sup>  
[Primitives](#)<sup>[29]</sup>

#### 2.4.4.3 MoveModel

##### C/C++ declaration

```
long __stdcall MoveModel(long Handle, long HandleSource, int DoNorm);
```

##### Pascal declaration

```
function MoveModel(Handle, HandleSource: LongInt; DoNorm: integer):  
LongInt; stdcall;
```

*Duplicate a model towards an existing driver. The actual model in the driver is lost.*

IN → Handle : Model handle.

[Primitives](#)<sup>[29]</sup>

#### 2.4.4.4 SetDataProc

##### C/C++ declaration

```
extern int __stdcall SetDataProc(long Handle, void * Func,  
TDataModifStyle Style);  
typedef enum{ //Pour la fonction SetDataProc  
nds_Modify, // Fonction de type TModifyDataFunc  
nds_Translate // Fonction de type TTranslateDataFunc  
} TDataModifStyle;
```

##### Pascal declaration

```
function SetDataProc(Handle: LongInt; Func: Pointer; Style:  
TDataModifStyle): Integer;  
TDataModifStyle = ( //Pour la fonction SetDataProc  
nds_Modify, // Fonction de type TModifyDataFunc  
nds_Translate // Fonction de type TTranslateDataFunc  
);
```

*Definition of a new data pre processing function.*

IN → Handle : Model handle.

IN → Func: pointer to the function. Two function types are allowed :  
TTranslateDataFunc and TModifyDataFunc.

IN → Style : TDataModifStyle

Return : 0 si OK, -1 if invalid handle, 1 if model has no training capability

##### Remark :

*The type TTranslateDataFunc allows a pre processing of the line read in a csv like text file. This*

*function must perform the data translation towards the "Target" vector.*

*The type TModifyDataFunc allows a pre processing of the real values vector just before its storing in the memory.*

To remove one of the functions, call the procedure with a null parameter.

[TDataFunc](#)   
[Primitives](#) 

# Index

## - A -

Autres types: 28

## - C -

CloneModel 49  
CloseLibrary 30  
Constantes: 28  
Conventions: 6  
CreateDataAnalyser 41  
CreateNewModel: 31

## - D -

DestroyModel: 31

## - F -

FunctionCompute2D 42  
FunctionCompute3D 42

## - G -

GetIndexInfo 31  
GetInfo: 32  
GetNeuronInfo: 43  
GetRealValue 32  
GetSynapseInfo: 43  
GetText 33  
GetVector 33

## - H -

HiddenLayerAnalysis 44

## - I -

InputGrammSchmidt 44  
Installation of MONAL: 3

## - L -

LicenseLevel: 34  
LoadModelFromFile 34  
LoadOptimizer 45

## - M -

ModelAction 34  
ModifyModel: 49  
MoveModel 50  
MultiTrain 45

## - N -

NewOptimizer 46

## - O -

OpenLibrary 35  
OptimizeInputs 46  
OptimizeInputsEx 47

## - S -

SaveModel: 40  
SaveParameters 41  
SetValue 35  
SetNewDataProc: 50  
SetRandSeed: 36  
SetRealValue 36  
SetText 36  
SetTrainingSet: 37  
SetVector 37

## - T -

TestFile 38  
TInfoPilote: 22  
TLoopData: 15  
TModifyNetwork: 17  
TNeuronData: 18  
TNewDataProc: 8  
Train: 38  
Transfer: 39

TransferData 47

TransferFD 48

## - U -

Utilisation des primitives en C / C++: 6

Utilisation des primitives en Delphi: 7

Utilisation du Run-Time MONAL avec vos programmes: 3

Endnotes 2... (after index)

Back Cover