Overflow User's Manual

Jean-Marc Valin
Dominic Létourneau

17th June 2002

Contents

List of Figures

Chapter 1

Introduction

Overflow is a free (GPL/LGPL) "data-flow oriented" development environment. It can be used to build complex applications by combining small, reusable building blocks. In some way, it has similarities with *Simulink* and *LabView*, although it is not designed (and far) to be a "clone" of any of them.

Overflow features a GUI that allows rapid application development and includes a visual debugger. Although Overflow can be seen as a rapid prototyping tool, it can also be used for building real-time applications, such as audio effects processing. Since overflow is not really an "interpreted language", it can be quite fast.

It is written in C++ and features a plug-in mechanism that allows new nodes/toolboxes to be easily added. Overflow currently includes the following toolboxes:

- Signal processing
- Speech processing (part of the Open Mind Speech project)
- Vector quantization (VQ)
- Neural network (MLP)
- Fuzzy logic
- Real-time audio effect processing (early stage)
- Linear algebra using LAPACK/BLAS/ATLAS (early stage)
- Image processing (early stage)

1.1 The Idea Behind Overflow

Overflow was designed with the following goals in mind:

- Ease of use

- Speed
- Flexibility
- Extendibility
- Modularity

One thing to note with respect to speed is that we tried the same approach as for the C++ language which can be summarized by “you don’t pay for the features you don’t use”.

1.2 Terminology

This section defines the concepts and terms used by Overflow and shows how they relate to C or Matlab constructs.

1.2.1 Nodes

The basic processing using in Overflow is a Node, a Node is in all ways similar to a C or Matlab function. It takes some input data, performs some operations and send data out.

Built-in nodes

A built-in Overflow node is written in C++ and is part of the Overflow code (or compiled in an Overflow toolbox, like Matlab’s .mex files). In Overflow, all nodes are implemented as a class that derive (directly or indirectly) from a base class called "Node" (note that most nodes derive from "BufferedNode"). Although the Overflow implementation of different nodes uses C++ inheritance mechanism (using classes), there’s no reason for the user to be aware of that. For that reason, it’s not recommended to refer to nodes as "types" or “classes” (e.g.. if Overflow were written in C, nodes would be implemented as functions).

Sub-networks (composite nodes)

An Overflow sub-network (or subnet) is a collection of connected nodes that can be used as if they were a single node. Most Overflow subnets will be saved into .n files, which are almost the exact equivalent of Matlab’s .m files. There’s no real C equivalent because C is a compiled language (although it could be seen as a C function calling other C function).

Node terminals (inputs/outputs)

The inputs of an Overflow nodes are equivalent to the arguments to a Matlab/C function. The same for outputs, but while C is restricted to one return value, Overflow and Matlab can have several outputs. Node inputs and outputs are sometimes referred to as “terminals”.

Node parameters

Overflow node parameters are also equivalent to C/Matlab function arguments. The difference between node parameters and node inputs is that parameters cannot change at run-time. They are specified at "build-time" and stay constant throughout the run. For instance, the "Constant" node has no input, but has a parameter called "VALUE" that is returned as the output of the node. Using constants, you can always "transform" another node's input into a parameter (to the constant). The reverse is not true, however. Why then have parameters and not define every argument as an input? Mostly simplicity and run-time performance. Sometimes, it is just a lot easier to know certain arguments in advance and be sure that they don't change during the run. However, when possible, it is better to implement arguments as inputs, as this allows more flexibility.

1.2.2 Links

There's no real correspondence between Overflow links and C or Matlab constructs. The best analogy would be to say that Links represent the order of the lines in a C/Matlab function. You also need to keep in mind that Overflow uses a "pull method" in order to compute data. What does that mean? When you run a network, the last node (output node) of the main network (called "MAIN" – how original!) is asked for its output. In order to compute its output, it needs to ask its input nodes for their output. That way everything propagates from the end to the beginning recursively.

Now, why going backwards like that? That's a bit long to explain. The quick answer is "because". The longer answer involves faster handling of dependencies, faster processing, buffer management and things like that.

1.2.3 Data Types

Unlike Matlab, that only supports the complex-double-matrix type (well, that's not totally true, but...), Overflow (like C and C++) has support for many different types. The “basic” Overflow types are: Bool, Int, Float, Stream, String and Vector. There are also toolbox-specific types, like FFNet (neural network), VQ (Vector Quantizer), GMM (Gaussian Mixture Model), ...

Right now, the only way to define a new type in Overflow is by adding C++ code for it in a toolbox (or the core). Eventually, there will (could?) be a way to pack data in a "struct" using Overflow nodes, but this is not implemented yet.

Some Overflow Nodes expect a certain type of data as input/parameter and will generate a run-time exception (which will abort execution) if the wrong data type is used (e.g.. a Load node expects a Stream as input and nothing else). Some nodes, like the NOP (no-op) node, can take any type as input. Some node have more complex behavior, like the Add node that can add two floats, two Vectors of the same dimension, but cannot add a Bool and a Vector.

Chapter 2

Getting Started with *vflow*

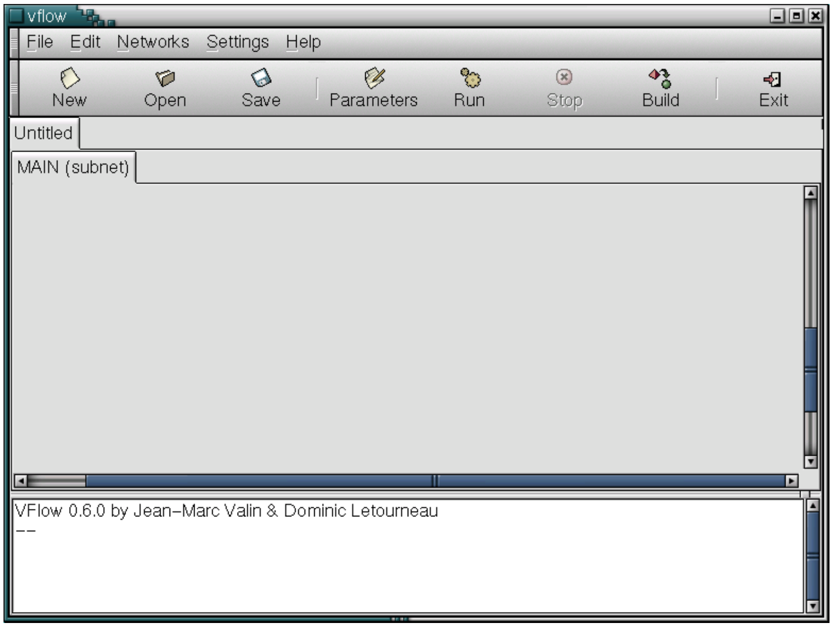
If called with no argument, the *vflow* program will start with a new empty Overflow document (fig. 2.1). It will already have a network named "MAIN". It is important that every "program" contain a network called "MAIN", which is equivalent to the *main()* function in a C program.

2.1 Basic GUI Controls

You can then add nodes to your network by clicking on the right button in the background, select *New Node* and the type of node you want. The node inputs are displayed as dots on the left side of each node, while the outputs are displayed on the right side. Inputs and outputs are called terminals. You can connect two nodes by clicking on a terminal and dragging the mouse to another terminal. Note that you cannot connect two outputs together, nor can you connect two inputs together. Except by using the *Feedback* node (see the "advanced" features section), you should not have feedback loops in your network. Links can be deleted by clicking on them with the SHIFT modifier on.

Right-clicking on a node brings up the node menu. Selecting "Properties" in the node menu brings a dialog with parameters used by the node. Each parameter has a name a type and a value. Some of the parameters are mandatory, while some others are optional. See the node documentation for a description of all the parameters.

All networks must have at least one output. Any network that is not a top-level network (MAIN) may also have inputs. Inputs and outputs names are added by left clicking on a terminal with the SHIFT modifier on. You will be asked to provide the input/output name.



```
VFlow 0.6.0 by Jean-Marc Valin & Dominic Letourneau
--
```

Figure 2.1: vflow main window

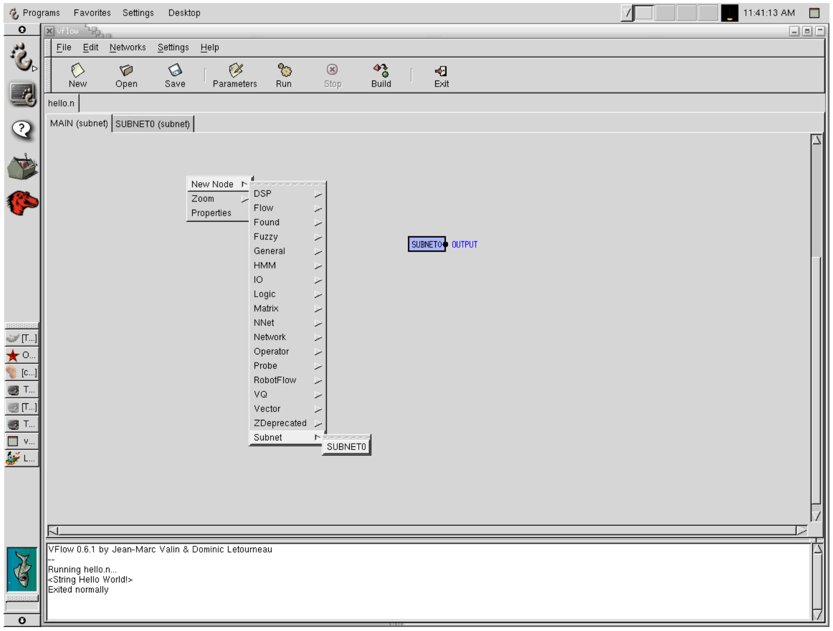
GUI Controls	Actions to do
Add nodes to the network	Right-click on the background and select *New Node* to add the node you want.
Connect two nodes	Click on the terminal and drag the mouse the other terminal.
Delete links between nodes	Click on the link with the SHIFT modifier on.
Set a node parameters	Double-click on the node OR right-click on it and select *Properties*.
Add output/input name to a node	Click on the terminal with the SHIFT modifier on.
Get information about the node	Middle-click on it

Basic GUI Controls

2.2 Using Sub-networks (subnets), Iterators and Threaded Iterators

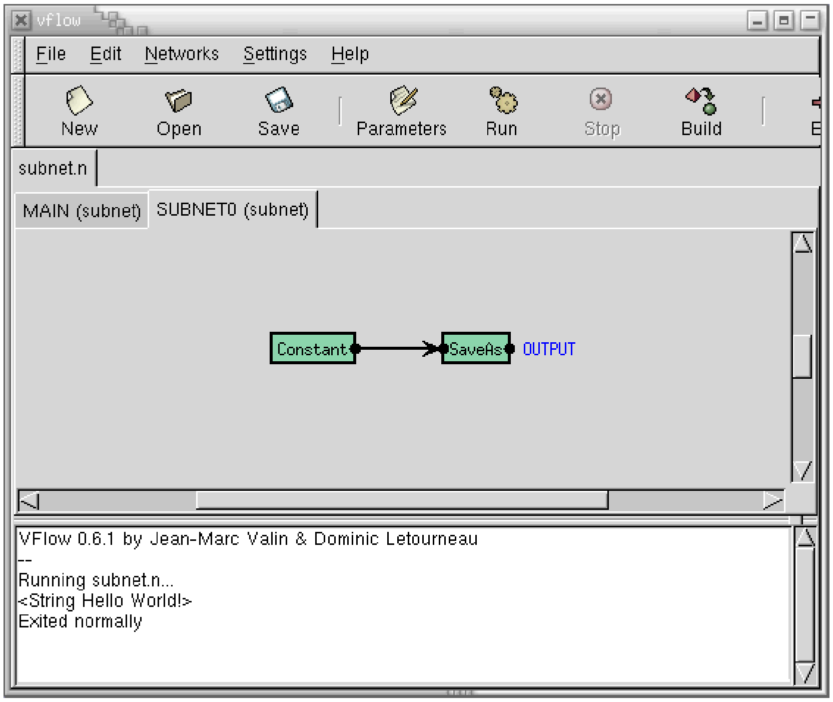
2.2.1 Sub-Networks

As mentionned earlier, every "program" contain a network called "MAIN", which is equivalent to the *main()* function in a C program. However, you can add more sub-networks (equivalent of sub-routines) from the main menu *(Networks->Add Network*) that can contain several nodes connected together. That way, You simplify programming and you can reuse those networks as subnets in a higher level network. It is very important to name the newly created network a different name than "MAIN" for obvious reasons. Those networks must absolutely have "named" inputs and outputs in order to be used in higher level networks as explained in the previous section. To add sub-networks into a network of higher level, right-click on the background and select the sub-network you want to add from the menu (New Node-> Subnet*)*.



```
VFlow 0.6.1 by Jean-Marc Valin & Dominic Letourneau
--
Running hello.n...
<String Hello World!>
Exited normally
```

Figure 2.2: Including a subnet with Overflow



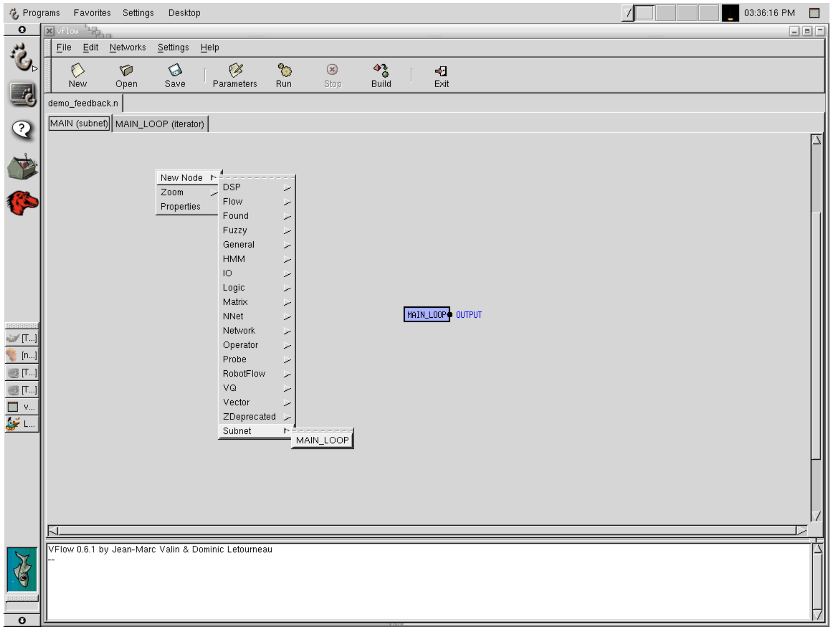
```
VFlow 0.6.1 by Jean-Marc Valin & Dominic Letourneau
--
Running subnet.n...
<String Hello World!>
Exited normally
```

Figure 2.3: Subnet with Overflow

Note that you can try out this program by clicking on the *open* icon and select: *FreeSpeech/examples/subnet.n* .

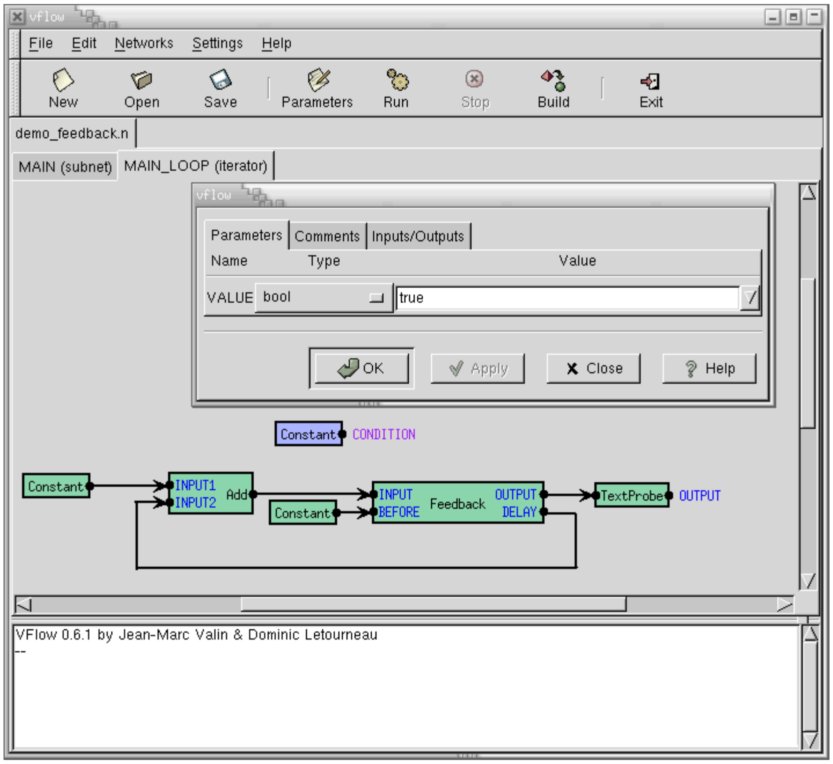
2.2.2 Iterators

Another useful type of network you can create is the Iterator *(main menu Network->Add iterator)*. An iterator, is a control structure that performs a loop. It stops looping when a certain "control condition" is met. The condition is a boolean value the iterator gets from a node. To define the iterator's condition, left click on a node output while holding the CONTROL (or ALT) modifier. Note that there is a bug in some versions of gnome for which CONTROL does not work with the canvas, so you'll have to use ALT.



```
VFlow 0.6.1 by Jean-Marc Valin & Dominic Letourneau
--
```

Figure 2.4: Including an iterator with Overflow



Name	Type	Value
VALUE	bool	true

```
VFlow 0.6.1 by Jean-Marc Valin & Dominic Letourneau
--
```

Figure 2.5: Iterator with Overflow

Note that you can try out this program by clicking on the *open* icon and select: *FreeSpeech/examples/demo_feedback.n* .

2.2.3 Threaded Iterators

For now, Threaded Iterators are experimental. They are a special kind of subnets that provide a different level of multi-threading. You should not use them, unless you REALLY know what you are doing.

2.3 Executing an Overflow Program

When your program is complete, you can execute it by clicking "Run" in the toolbar. If an error occurs, the program will abort and the error will be printed in the text box in the bottom pane. Node that as of 0.6.0, the created documents have execute permission and can be executed as a like a script, provided that the batchflow executable is in the path.

Chapter 3

Using nodes

This section presents different Overflow nodes categories. The nodes categories are divided in two by level of diffiuclty: the *Basic Nodes* and the *Advanced Features*. For every category (no matter its level of difficulty), at least one of his node is explained and given in example. For every example, you can find the corresponding program in the directorie: *FreeSpeech/examples* .

3.1 Basics nodes

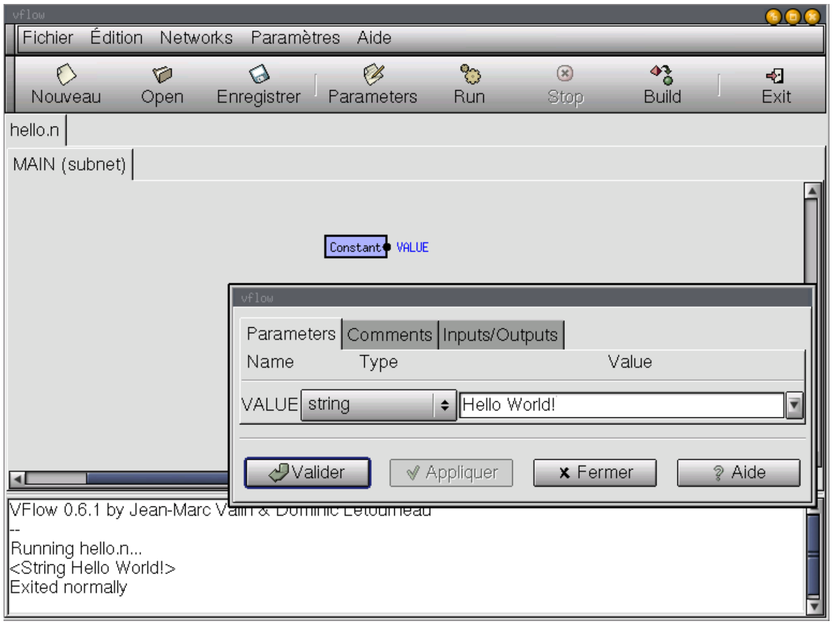
Lets get started with the nodes that are the most popular for new users, that is, the most important to understand.

3.1.1 Category: General

As the name mentions, this category contains nodes that you find frequently in programs.

Constant

Among the nodes of this category, the most often used is certainly the node "Constant". Figure 3.1 shows how you can make Overflow write "Hello World!" in the status window. Yes, it's simply a constant string (using *Constant* node) with an output label attached to it.


vflow
Fichier Édition Networks Paramètres Aide
Nouveau Open Enregistrer Parameters Run Stop Build Exit
hello.n
MAIN (subnet)
Constant VALUE
vflow
Parameters Comments Inputs/Outputs
Name Type Value
Valider Appliquer Fermer Aide

Name	Type	Value
VALUE	string	Hello World!

```
VFlow 0.6.1 by Jean-Marc Va... & Dominic Letourneau
--
Running hello.n...
<String Hello World!>
Exited normally
```

Figure 3.1: Hello World!

Note that you can try out this program by clicking on the *open* icon and select: *FreeSpeech/examples/hello.n* .

In the previous example, the node “Constant” took the data type “string”, however, the node can take other data types. Here is a list of all data types that a node “Constant”can take:

int: This data type has a representation that can take the value of -2 147 483 648 to 2 147 483 647.

float: Floats have the same size as “int”, but are read as floating point numbers. They range from the value -1.4013e-45 to 1.4013e-45 and from -3.4e38 to 3.4e38.

bool: Object that can take the values: “true” or “false”. This data type is often used for condition input. For example, in a node IF, the input “COND” is set by a constant of type bool. When the constant is “true”, the input THEN is pulled and when the constant is “false”, the input ELSE is pulled.

object: An object is a data type defined by developpers. However, you can create an exiting object by typing its corresponding command. For instance, you can create a string “hello world” by typing “<String hello world>” (“<Vector 1 2 3 4 5 6 7 8>” , “<Int 23>” and “<Float 23.67>” are some other examples).

string: Strings are series of characters ('a'-'z', '0'-'9','A'-'Z', or some other symbol)

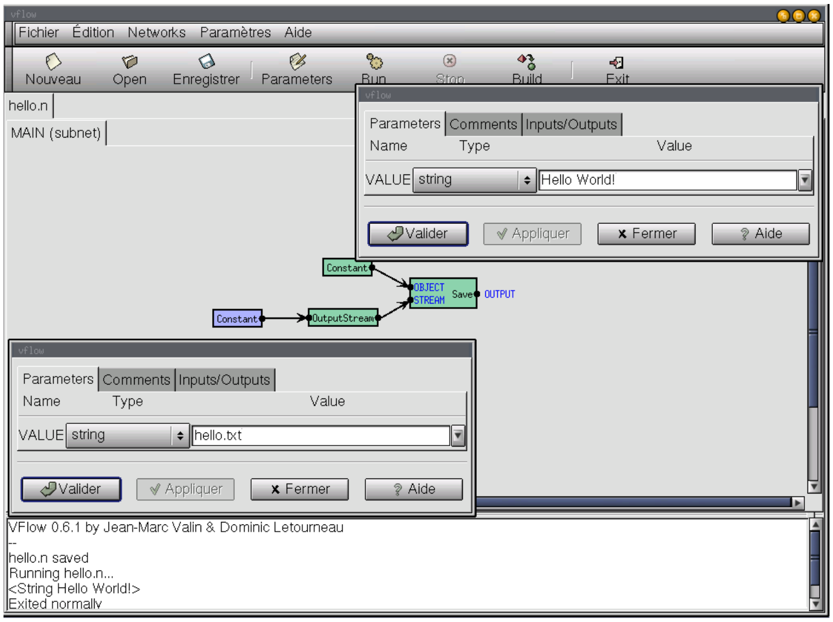
subnet_param: Subnet_param set parameter's name of higher level. For example, create a network SUBNET0 in a file. In SUBNET0 add and set a subnet_param to "hello". Add the node SUBNET0 to the main network and double-click on it. Then, you will see in the tab "Parameters" a parameter to set named "hello".

3.1.2 Category: IO

This category concerns mainly the use of streams for reading, loading and saving data of any kind.

Save

This example is used to save an object in a file or a network stream. Figure 3.2 shows that the nod *Save* takes two inputs: an object to save and an Overflow write stream where to save the object. The write stream is opened using the *OutputStream* node which takes a file name as input (allo.sdw).



Name	Type	Value
VALUE	string	Hello World!

Name	Type	Value
VALUE	string	hello.txt

```
VFlow 0.6.1 by Jean-Marc Valin & Dominic Letourneau
--
hello.n saved
Running hello.n...
<String Hello World!>
Exited normally
```

Figure 3.2: I/O with Overflow

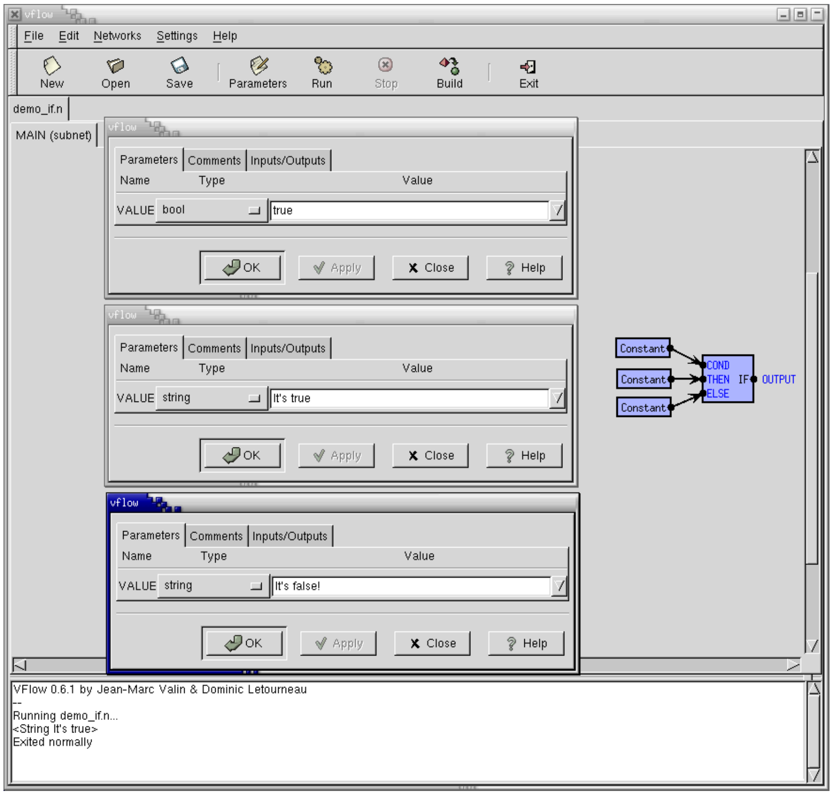
Note that you can try out this program by clicking on the *open* icon and select: *FreeSpeech/examples/save.n* .

3.1.3 Category: Logic

This categroy contains nodes that proceed logical operations such as: AND, OR, NOT, IF etc...

IF

On of the most important control structure in a language is conditional branching. Figure 3.3 shows an *IF* statement in Overflow. The node *IF* pulls his input from the terminal *then* or *else* depending on whether the condition is true or false. When the condition is true, it pulls the terminal *then* and when the condition is false, it pulls the terminal *else*. In both cases, the terminal pulled is returned as the output of the *IF*.



Name	Type	Value
VALUE	bool	true

Name	Type	Value
VALUE	string	It's true

Name	Type	Value
VALUE	string	It's false!

```
VFlow 0.6.1 by Jean-Marc Valin & Dominic Letourneau
--
Running demo_if.n...
<String It's true>
Exited normally
```

Figure 3.3: If/then/else with Overflow

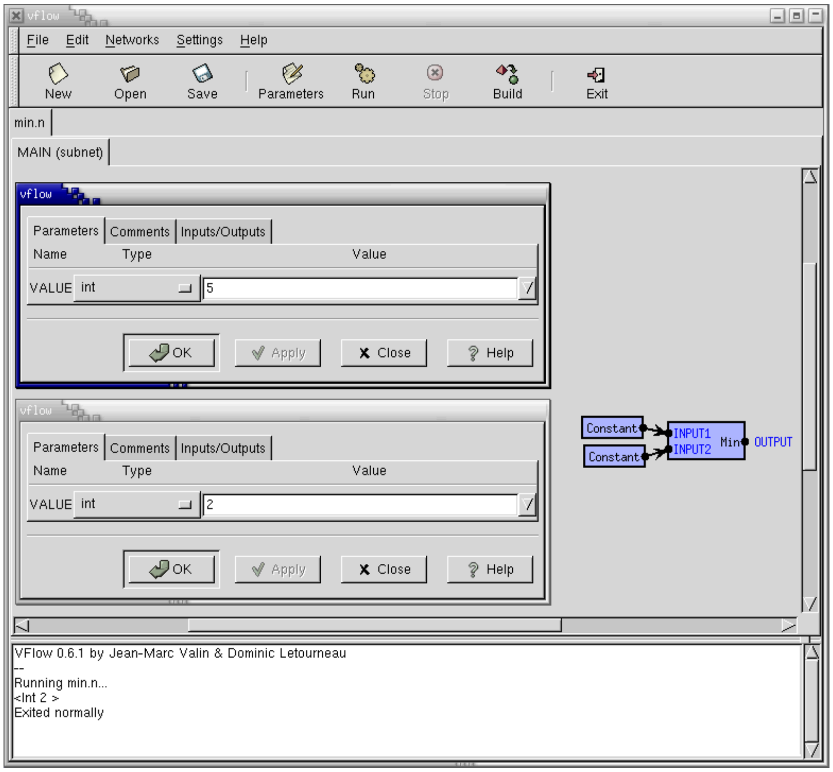
Note that you can try out this program by clicking on the *open* icon and select: *FreeSpeech/examples/demo_ if.n* .

3.1.4 Category: Operator

This categroy contains nodes that proceed arithmetic operations such as: ADD, DIV, EQUAL, MIN, etc...

MIN

Figure 3.4 shows a *MIN* statement in Overflow. The node *MIN* compares its two inputs and return the minimum between the two values.



Name	Type	Value
VALUE	int	5

Name	Type	Value
VALUE	int	2

```
VFlow 0.6.1 by Jean-Marc Valin & Dominic Letourneau
--
Running min.n...
<Int 2 >
Exited normally
```

Figure 3.4: Min with Overflow

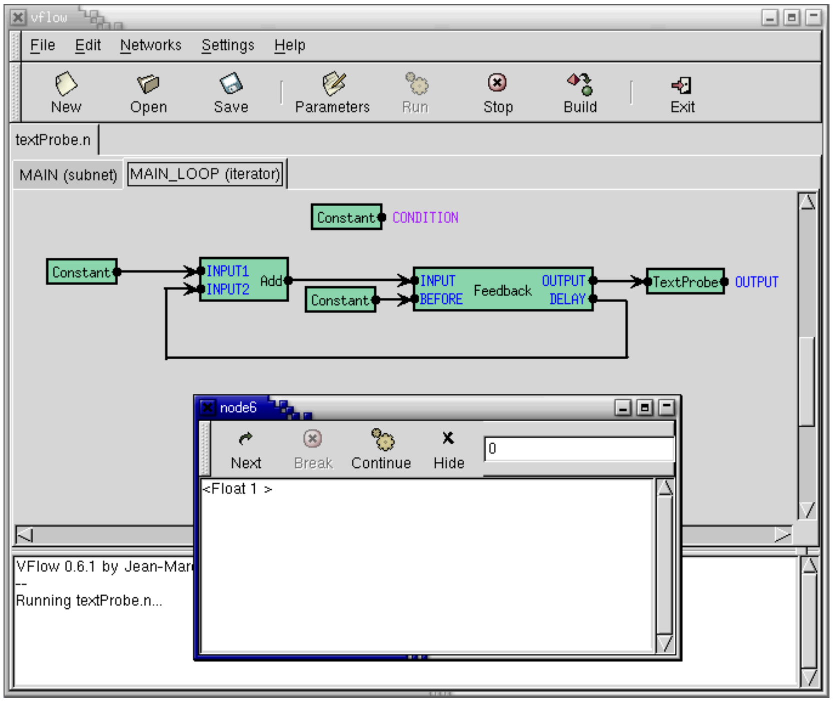
Note that you can try out this program by clicking on the *open* icon and select: *FreeSpeech/examples/min.n* .

3.1.5 Category: Probes

This categroy contains nodes that proceeds data plotting and printing. It also helps for debugging by allowing to “trace” programs initializations, requests, inputs and exceptions.

textProbe

Figure 3.5 shows an Overflow program that uses *textProbe.* That way, *textProbe* shows the program results as text in a box. In iterators, textProbe allows to show the program result for every new iteration by clicking on the icon *Next.*



```
0
```

```
<Float 1 >
```

```
VFlow 0.6.1 by Jean-Mar
--
Running textProbe.n...
```

Figure 3.5: textProbe with Overflow

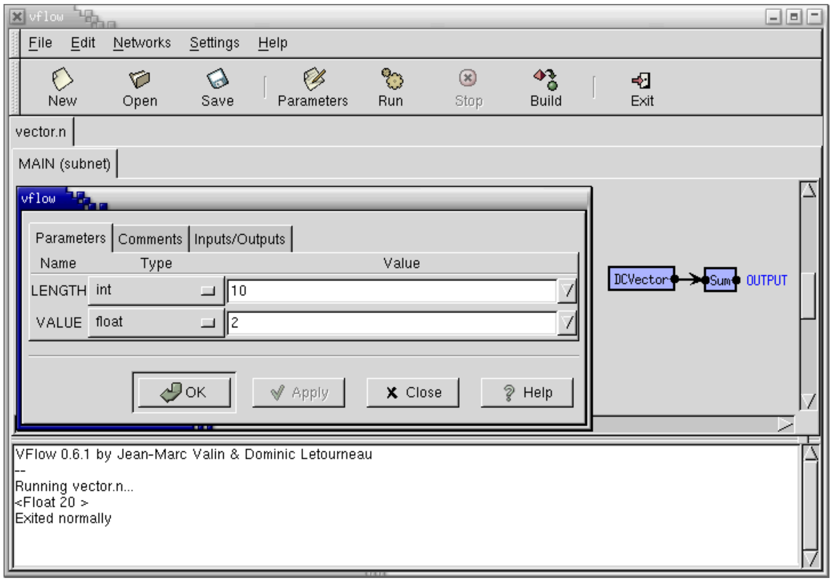
Note that you can try out this program by clicking on the *open* icon and select: *FreeSpeech/examples/textProbe.n* .

3.1.6 Category: Vector

This categroy contains nodes that proceed operations on vectors such as: SUM, CONCATENATE, LENGTH, DCVECTOR, etc...

DCVector and SUM

Figure 3.6 shows an Overflow program that creates a vector and make the summation of all its objects. For creating a *DCVector* you must specify the number of objects you want it to contain (the length) and the value you want all the objects to take (the value). The node *SUM,* add all of the objects given by the vector at his input and return the result.



Name	Type	Value
LENGTH	int	10
VALUE	float	2

```
VFlow 0.6.1 by Jean-Marc Valin & Dominic Letourneau
--
Running vector.n...
<Float 20 >
Exited normally
```

Figure 3.6: Using a vector with Overflow

Note that you can try out this program by clicking on the *open* icon and select: *FreeSpeech/examples/vector.n* .

3.2 Advanced features

The following categories of nodes presented are more complex than the others and not necesserely useful for every user. That's the reason why they are placed in this section.

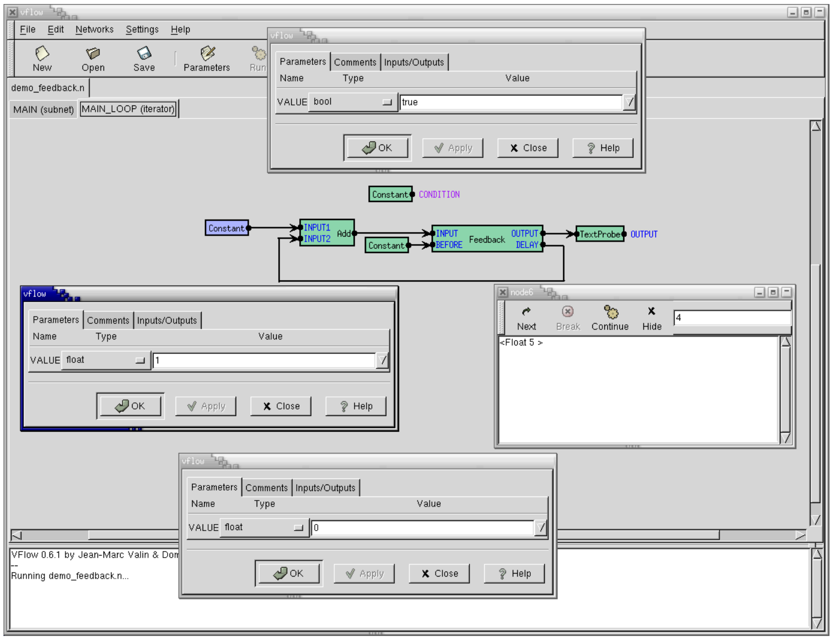
3.2.1 Category: DSP

This category was created for Speech recognition and voice treatment.

3.2.2 Category: Flow

Feed-back Loops

In some circumstances, it is desirable to insert feed-back loops into a program. Normally, Overflow only supports acyclic graphs, but feed-back loops can be made using the special *Feedback* node. Figure 3.7 shows an Overflow program that uses a *Feedback* node. At the beginning, the constant “1” is added to “0” and then, at every iteration, the result of the addition is added to the same constant “1”.



VALUE bool true

VALUE float 1

VALUE float 0

node6: 4

```
<Float 5 >
```

```
VFlow 0.6.1 by Jean-Marc Valin & Dom
--
Running demo_feedback.n...
```

Figure 3.7: Feed-back Loop with Overflow

Note that you can try out this program by clicking on the *open* icon and select: *FreeSpeech/examples/demo_feedback.n* .

Multi-threading

It must first be noted that multi-threading in Overflow is still an experimental feature and not all structures are fully MT-safe. Overflow multi-threading is provided through three special nodes: *SerialThread*, *ParallelThread* and *ThreadJoin*.

SerialThread

The *SerialThread* node provides pipeline-type multi-threading. A thread is started and computes inputs before they are needed by the output node.

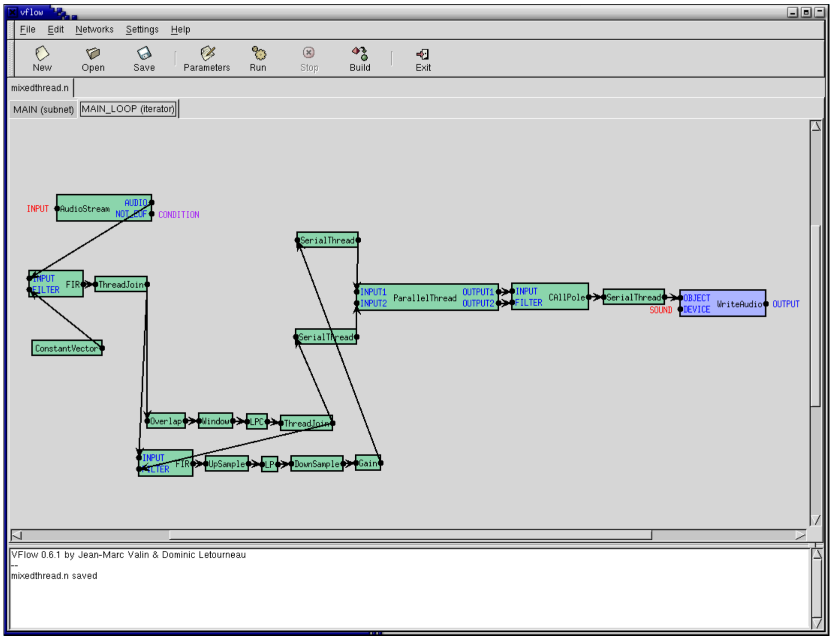
ParallelThread

The *ParallelThread* node provides parallelism-type multi-threading. When asked for an input, it computes both inputs at the same time and caches the other.

ThreadJoin

The *ThreadJoin* node acts like a mutex and prevents two Overflow threads from accessing the same (input) node at the same time.

Figure 3.8 shows an example of Overflow program that uses multi-threading.



```
VFlow 0.6.1 by Jean-Marc Valin & Dominic Letourneau
--
mixedthread.n saved
```

Figure 3.8: Feed-back Loop with Overflow

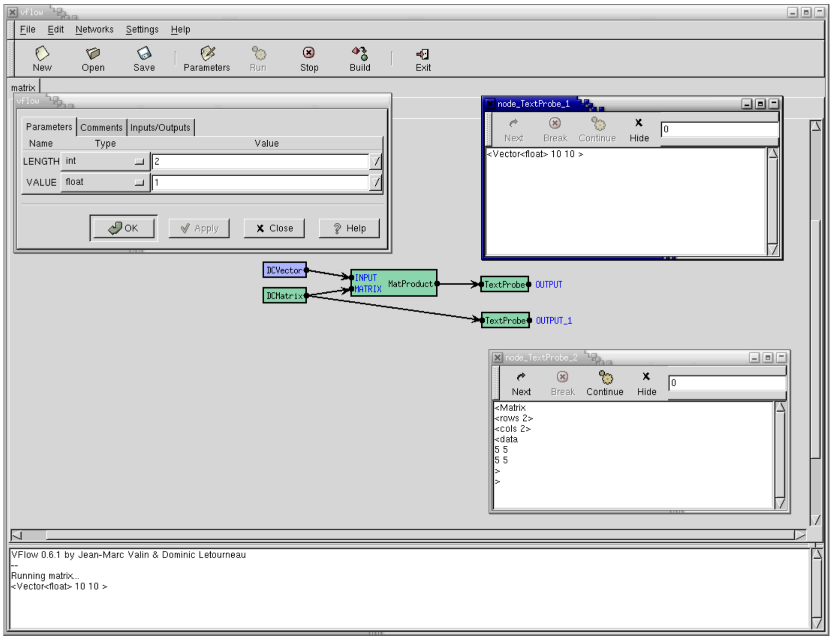
Note that you can try out this program by clicking on the *open* icon and select: *FreeSpeech/examples/mixedthread.n* .

3.2.3 Category: Fuzzy

3.2.4 Category: HMM

3.2.5 Category: Matrix

As the name mentions, this category is about matrix, that is, two dimensions vector. Figure 3.9 shows an example of Overflow program that create matrix and use it with the node *MProduct*.



Name	Type	Value
LENGTH	int	2
VALUE	float	1

node_TextProbe_1: 0

```
<Vector<float> 10 10 >
```

node_TextProbe_2: 0

```
<Matrix
<rows 2>
<cols 2>
<data
5 5
5 5
>
>
```

```
VFlow 0.6.1 by Jean-Marc Valin & Dominic Letourneau
--
Running matrix...
<Vector<float> 10 10 >
```

Figure 3.9: Matrix with Overflow

Note that you can try out this program by clicking on the *open* icon and select: *FreeSpeech/examples/matrix.n* .

3.2.6 Category: NNet

3.2.7 Category: Network

3.2.8 Category: RobotFlow

3.2.9 Category: VQ

Chapter 4

Exceptions

4.1 Run-time Exceptions

First, it is important to note that there are two types of exceptions that can happen. The first type, which we'll call run-time exceptions, is thrown (usually by a node) when an error happens during processing by Overflow. Such type of exception can be thrown when a node receives an object of an unexpected type, but there can be many other causes. Run-time exceptions usually terminate the current Overflow program with an error message indicating where the error happened. They are analogous to run-time errors in most interpreted languages.

It is possible to prevent a run-time exception from stopping a program. This can be done with the Recover node, that catches all run-time exceptions.

4.2 User Exceptions

The second type of exceptions, user exceptions, can be thrown and caught by a user program using the Throw and Catch nodes. They are serve the same function as the throw and catch statements in a C++ program.

Chapter 5

Automatic Code Generation

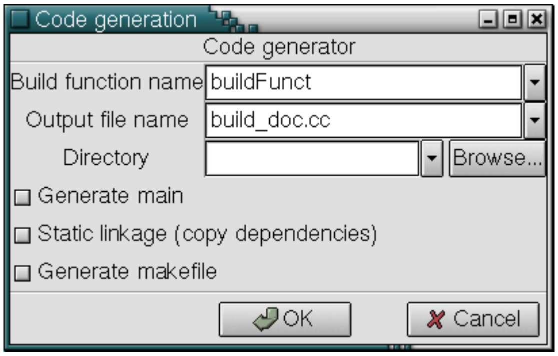
Is is now possible to generate C++ code from an Overflow .n file. You can do that by clicking the "Build" button on the toolbar. Doing so brings up the code generation dialog (fig. 5.1).

The "Build function name" is the name given to the function that will return the network (`Network *`) corresponding to the XML file. You will only care if you want to build a library or if you want multiple networks. The "Output file name" option specifies the name of the C++ file that builds the network. Of course, "Directory" specifies where to put all that.

There are 3 code generation options:

- Generate main: Check that if you want to build an application (as opposed to a library)

- Static linkage: This will copy all the required Overflow files in the same directory. This will allow to build an application/library without linking



Build function name: buildFunct

Output file name: build_doc.cc

Directory:

□ Generate main

□ Static linkage (copy dependencies)

□ Generate makefile

Figure 5.1: Code generation dialog

to Overflow. Note that the Overflow license (LGPL) still applies to the copied files.

- Generate makefile: Unimplemented yet

Note: the VFLOW_SOURCE environment variable must be set to your Overflow source directory in order for the "static linkage" option to work.

Chapter 6

Software Architecture

This chapter explains the software architecture used for the Overflow base library and the mechanisms used for blocks initialization.

6.1 Overflow Internals

6.1.1 Nodes

A node is the smallest processing unit in Overflow. Once it has been initialized, the only method it understands is `getOutput(int output_id, int count)`. In other words, all you can do with it is to ask it for its output. One obvious consequence of that is that all node must have at least one output, but it can have more than one. A node can have any number of input, including zero (examples of node with no inputs are constants and random generators).

If a node requires input data in order to perform some calculations, it will call `getOutput(...)` on its input node(s). Computation is hence performed in a recursive manner until everything is calculated and the last node returns its output. The count argument to the `getOutput(...)` method is used when loops are involved. It specifies the number of the iteration. Note that it is possible for a node to ask its inputs for a different count that the one received. It is even possible to ask for many different count values in a row.

The Node class is an abstract class from which all types of nodes must derive (directly or indirectly). Information on how to derive new types of nodes is given in Extending Overflow chapter 7.

6.1.2 Networks

A network is a graph containing nodes that are linked together in order to perform some operation and/or return a result. Most of the time, the graph will be acyclic that is, it will not contain loops. It is now (as of march 2001) possible to have feedback loops using the FeedBack node, but this is a more advanced topic. One thing worth mentioning is that the Network class derives

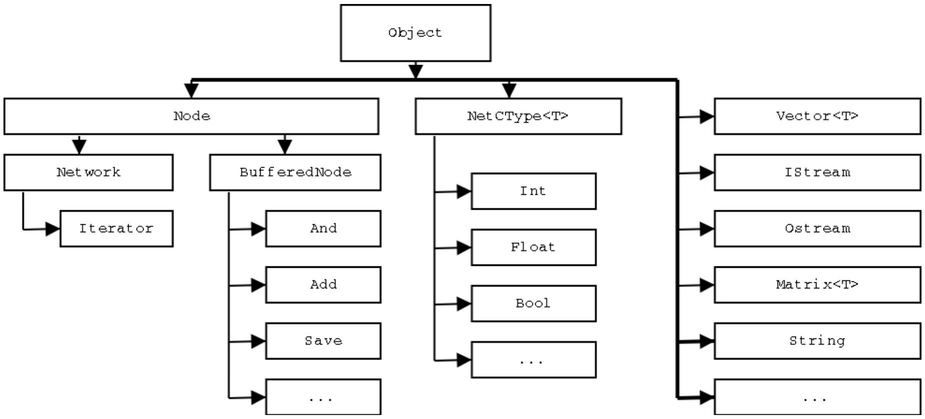


Figure 6.1: Main Overflow classes

directly from the Node class. This means that any network can be used just as if it were a single Node. This makes it possible to include networks in other networks. The included network is often referred to as sub-network, or subnet.

In order for a network to be valid, it must meet the following criteria

- It must have at least one output
- Every node it contains must have a connection to at least one of its output
- All node must have all their inputs connected
- A toplevel network may not have inputs
- There should be no loop (except by using a FeedBack node)

6.2 Class Diagrams

6.3 Data Types

stream: Fd (fd), fptr (FILE) or cpp (stream)

vector: Series of object references. For instance: “<Vector 1 2 3 4 5 6 7 8>”.

matrix: Two dimensions vector.

Chapter 7

Extending Overflow

Overflow is designed to be extendible in many areas, so that it is possible to create new: node types, operators and data types.

7.1 Writing New Nodes

Most of the new nodes will derive from either the Node abstract class or the BufferedNode abstract class. You should use public inheritance when deriving your new class. In all cases, you will need to define a constructor for your new node class. The parameters for this constructors are: `(string nodeName, const ParameterSet &params)`, which are used to initialize the base class, e.g.

```
class MyNode : public BufferedNode {
public:
   MyNode(nodeName, params) : BufferedNode(nodeName, params)
   ...
};
```

Also, if you derive from BufferedNode, you need to define the `virtual void calculate(int output_id, int count, Buffer &out)` method. The arguments are the ID of the input requested (`output_id`), the iteration ID (`count`) and the output buffer for the requested output (`out`). The `calculate` method is expected to assign an object to `out[count]`.

If you derive directly from the Node class, you will need to override the `ObjectRef getOutput(int output_id, int count)` method. The meaning of `output_id` and `count` is the same as for the BufferedNode equivalent, and the result should be returned as an ObjectRef.

Here are some other methods you might want to define too:

- `void initialize()`: As the name implies, it is meant to perform some initialization that cannot be done within the constructor. This method is called only once, starting the processing, but after all the `request()` have been made. In most (all) cases `initialize()` should start by calling the base class implementation (e.g. `BufferedNode::initialize()`).

- `void reset()`: This method should return the node to the same state it was after `initialize()` was first called. In most (all) cases reset() should start by calling the base class implementation (e.g. `BufferedNode::reset()`).

In some rare cases, you will want to define the following method:

- `void request(int outputID, const ParameterSet &req)`: This method is meant to pass on *special requests* to input nodes. For now, this is mainly used by the *BufferedNode* class to compute the size needed for the output buffers. Remember that if you override this method, you **must** make sure that it propagates the request to **all** its input nodes. Otherwise, the nodes that won't be reached will have incorrect buffer size.

At last for a new node to be visible in *vflow*, a special header must be present. An example of this is:

```
class MyNode;
DECLARE_NODE(MyNode)
/*Node
 *
 * @name MyNode
 * @category MyCategory:MySubCategory
 * @description Some description of what MyNode does
 *
 * @input_name SOME_INPUT_NAME
 * @input_type this_input_type
 * @input_description Description of this input
 *
 * @input_name SOME_OTHER_INPUT
 * @input_type that_input_type
 * @input_description Description of that output
 *
 * @output_name SOME_OUTPUT
 * @output_type this_output_type
 * @output_description Description of the output
 *
 * @parameter_name SOME_PARAMETER
 * @parameter_type this_parameter_type
 * @parameter_description The description of the parameter
 * END*/
```

Although this header is only a C++ comment, it is parsed by a PERL script to produce an XML description of each toolbox. The DECLARE_NODE(MyNode) macro is used to register the node in a dictionary when the toolbox is dynamically loaded.

7.2 Example: VAdd.cc

Most nodes must include BufferedNode.h. Also, since this node deals with vectors, we need Vector.h

```
#include "BufferedNode.h"
#include "Vector.h"
```

forward declaration of class VAdd for use with the DECLARE_NODE macro

```
class VAdd;
```

Declaration of the node. This definition is transformed into XML data for the GUI, as well as documentation for the node

```
DECLARE_NODE(VAdd)
/*Node
*
* @name VAdd
* @category DSP:Base
* @description Adds two vectors of same length
*
* @input_name INPUT1
* @input_type Vector<float>
* @input_description First vector
*
* @input_name INPUT2
* @input_type Vector<float>
* @input_description Second vector
*
* @output_name OUTPUT
* @output_type Vector<float>
* @output_description Result vector
*
END*/
```

Class definition/implementation. Note that because we won't need to derive from this class, we don't need a header file (.h) and we can put everything in the .cc. Our node, like most other nodes, derives from BufferedNode.

```
class VAdd : public BufferedNode {
   int input1ID;
   int input2ID;
   int outputID;
public:
   VAdd(string nodeName, ParameterSet params)
   : BufferedNode(nodeName, params)
   {
```

In the constructor, we create both the inputs and outputs.

```
          input1ID = addInput("INPUT1");
          input2ID = addInput("INPUT2");
          outputID = addOutput("OUTPUT");
       }
```

This is the main method for the node, it is called from the BufferedNode class each time a result needs to be calculated.

```
       void calculate(int output_id, int count, Buffer &out)
       {
```

Get input data from previous node(s).

```
          ObjectRef input1Value = getInput(input1ID, count);
          ObjectRef input2Value = getInput(input2ID, count);
```

We cast the generic objects (received through ObjectRefs) into a reference to a Vector<float>. If the cast fails, an exception will automatically be thrown.

```
          const Vector<float> &in1 = object_cast<Vector<float> > (input1Value);
          const Vector<float> &in2 = object_cast<Vector<float> > (input2Value);
```

Check that the size of the two vectors match. Otherwise, throw an exception. Here __FILE__ and __LINE__ are pre-processor macros that will print the file and line where this exception was thrown.

```
          if (in1.size() != in2.size())
              throw new NodeException(this,
                                      "Input vectors must be of same length",
                                       __FILE__, __LINE__);
          int inputLength = in1.size();
```

Allocate a new Vector<float> from the pool of free vectors (that's why we don't use new).

```
          Vector<float> = &output =
                          *Vector<float>::alloc(inputLength);
```

Put the new Vector<float> in the return buffer.

```
          out[count] = &output;
```

Compute the result of the sum.

```
          for (int i=0;i<inputLength;i++)
             output[i]=in1[i]+in2[i];
       }
    };
```

7.3 Creating New Operators

7.3.1 Double Dispatched Operators

It is possible to define binary operators that can act on different kinds of input. One example is the "add" operator, which can be used to add two ints, two floats, two vectors, or an int and a float, ... See data-flow/include/operators.h

7.4 Adding New Data Type

It is possible to define new types in Overflow. In order to be used in new nodes, new types must derive from the Object base class. That the only absolute requirement. However, if you want the new type to integrate more closely with Overflow, there are several things you can do:

- Implement the void `printOn(ostream &out) const` method. This method writes the object to the out stream.
- Implement the void readFrom (istream &in).
- Add the macro `DECLARE_TYPE(`*MyType*`)` to the C++ file where the object is implemented.

There is a certain format which all Object must respect. The object should start with "<MyType" and end with ">" (without the quotes). Usually, every field will be inside < and > signs.

Appendix A

Compiling and Installing

What you need

- An ANSI C++ compiler
  - gcc 2.95.x is OK
  - most of gcc 2.96 variants are OK
  - gcc 3.0.x compiles, but there are run-time glitches
  - egcs 1.1.2 is untested (probably doesn't work)
  - MSVC++ is completely broken, but it possible build a subset of Overflow with it (see here"Compiling on Win32")
  - HP's aCC should work after some minor modifications
- autoconf, automake, libtool (which require perl and m4)
- GNU make
- FFTW (now optional, but recommended) compiled with –enable-float
- gnome (including the development libraries and libxml)
- pthreads (now part of libc in most Linux distributions)

Compilation flags

If you are using gcc, you can control optimization with the CFLAGS and CXXFLAGS environment variables. For use on a Pentium III or an Athlon XP, we suggest to set both CFLAGS and CXXFLAGS to: '-O3 -march=pentiumpro -D_ENABLE_SSE'. For T-bird Athlon, we suggest replacing -D_ENABLE_SSE by -D_ENABLE_3DNOW. This must be done **before running configure**. Also, note that the default flags used if CFLAGS and CXXFLAGS are not set are '-O2 -g'. **It is strongly recommended not to compile Overflow with -g** unless you're really desperate, as the binaries might take up to 600 MB of disk space (instead of 6-10 MB otherwise) due to the C++ name mangling.

Configure options

- –with-libtool-ld=<c++ compiler> You need to specify this option if libtool tries to use ld to link the C++ libraries and executables. These need to be linked with the C++ compiler (e.g. g++) because of initializations that must be performed before the main() starts (On Linux you most likely don't need that).
- –with-fftw-dir=<fftw path> If FFTW is not installed in a standard path, you will need to specify this option.
- –disable-static This option is required. Overflow does not work with static libraries (because it uses dlopen).
- –disable-<package> Doesn't build a certain package (<package> is HMM, VQ, NNet, ...)

Compiling & Installing the software

To compile, type:

```
% ./configure –disable-static –prefix=<your install directory>
% make
% make install
```

Notes:

- As of version 0.5.1, it is now recommended to set the install prefix to /usr or /usr/local, unless you want to keep more than one version installed at the same time.
- If you are using a CVS tarball, you need to use ./autogen.sh instead of ./configure
- You might also need to set your LD_LIBRARY_PATH to <overflow prefix>/lib

You can now start the Overflow environment by typing :

```
% vflow (assuming <overflow prefix>/bin is in your path)
```

Compiling on Win32

Some parts of Overflow (sorry, no GUI yet!) have been ported to Win32 (w/ MSVC++). Using the code generation feature (the "Build" button on the toolbar), it is now possible to compile an overflow application on Windows. Note that this has not been fully tested yet.

One important thing to note with MSVC++ (version 6.0) is that it is a very buggy compiler, mostly when it comes to templates. For example, it does not support template partial specialization and it chokes on a lot of valid template code (static template member functions, pointer to template functions, ...). For this reasons some Overflow features need to be switched off.

So here are the settings you need for Overflow. First, you need to turn on RTTI (which is not enabled by default). Also, I suggest you turn the warnings off. The preprocessor flags (define) you need to set are: BROKEN_TEMPLATES, HAVE_FLOAT_H, NO_HASH_MAP, STUPID_COMPLEX_KLUDGE and (if not already defined), WIN32.

Appendix B

Troubleshooting

1) The binary distribution I downloaded crashes on startup

There can be many causes of that. The most common is that you have a different libstdc++ than the one Overflow was compiled with. Another cause could be that you have FFTW compiled without –enable-float, while Overflow was linked with a float version of FFTW. Overflow has no way to detect that so it crashes. In both cases, the best thing to do is to build Overflow yourself.

2) I compiled Overflow myself and it crashes on startup

The main cause for this is a bug/missing feature in libtool that prevents it from working correctly with C++ on some platforms. This happens mostly on non-Linux platforms though not always. If you suspect that's your problem, try running configure with the `--libtool-ld=g++` option.

3) Overflow tells me it cannot find libflow.so

This can happen if you compile Overflow and then move the installation directory (It can sometimes happen for other reasons). You can set the LD_LIBRARY_PATH to <overflow install dir>/lib. Note that if you moved the Overflow directory, you'll also need to set VFLOW_HOME.

4) The "New Node" menu is empty

You probably moved the Overflow install directory, see 3). Another possibility is if you compiled with `--enable-static` flag. Because Overflow toolboxes are dynamically loaded, everything must be compiled as shared libraries (which is the default in configure).

5) Overflow doesn't compile on my box

First, make sure you have the latest release version. If it doesn't work, you should try the CVS version. If it fails too, please contact us and we'll do our best to make Overflow compile on your platforms.

6) I downloaded a more recent version and it doesn't even compile

This can be due to the fact that you installed an earlier version (0.5.0 and earlier) in a path like /usr or /usr/local. The problem is that the old Overflow includes ended up somewhere like /usr/include so when you try compiling a newer version, g++ sees the old includes (because they are in the include path) instead of the new ones.

Appendix C

Node Documentation

This documentation is generated automatically from the comments included in the Overflow C++ source code.

AND (Logic)

Logical AND between two inputs

	Name	Type	Description
Inputs	INPUT1	bool	First boolean input
	INPUT2	bool	Second boolean input
Outputs	OUTPUT	bool	Boolean output
Parameters	PULL_ANYWAY	bool	Pull on INPUT2 even if INPUT1 is false

Abs (DSP:Base)

Computes the absolute value of each element of a vector

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input vector
Outputs	OUTPUT	Vector<float>	Output vector
Parameters	none		

Accept (Network)

Create a network socket of any type

	Name	Type	Description
Inputs	SOCKET	socket	The socket to listen to
Outputs	SOCKET	stream	The socket to be used for input/output operations
Parameters	none		

Accumulate (General)

Accumulation of objects into a buffer, that is, a vector of Objects References. When the node is in the main network or in a sub-network, his input is packed in the vector only once. However while in iterators, his input is packed (added) in the vector at every iteration. As well, his other input "ACCUM" must be connected to a node: "NewAccumulator"(General).

	Name	Type	Description
Inputs	INPUT	any	Input object
	ACCUM	Vector<ObjectRef>	Accumulator where to put the input
Outputs	OUTPUT	Vector<ObjectRef>	The input accumulator
Parameters	none		

Action (General)

Pulls in order the inputs: BEFORE, INPUT and AFTER.

	Name	Type	Description
Inputs	INPUT	any	The input
	BEFORE	any	To be pulled before
	AFTER	any	To be pulled after
Outputs	OUTPUT	any	The output = The input
Parameters	none		

AdaptMAP (HMM) (require: GMM) Performs MAP adaptation (well, almost!)

	Name	Type	Description
Inputs	FRAMES	Vector<float>	Frame buffer
	GMM	GMM	GMM to be adapted
Outputs	OUTPUT	GMM	Adapted GMM
Parameters	none		

Add (Operator)

Adds two input values and returns the result

	Name	Type	Description
Inputs	INPUT1	any	First value
	INPUT2	any	Second value
Outputs	OUTPUT	any	Result of the addition
Parameters	none		

Amplitude (ZDeprecated)

Deprecated, use RMS instead

	Name	Type	Description
Inputs	INPUT	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	none		

ArgMax (DSP:Base)

Finds the maximum value in a vector

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input vector
Outputs	OUTPUT	Vector<float>	Index 0 contains the maximum value, index 1 contains the index where the maximum is found
Parameters	START	int	Index where search is started
	END	int	Index where search ends

AudioStream (DSP:Audio)

Reads an audio stream and outputs frames

	Name	Type	Description
Inputs	INPUT	Stream	An audio input stream (IStream, fd or FILE *)
Outputs	AUDIO	Vector<float>	Frames read
	NOT_EOF	bool	True if we haven't reach the end of file yet
Parameters	LENGTH	int	Length of the frames (in samples)
	ADVANCE	int	Offset beween frames (in samples)
	ENCODING	string	Type of encoding (LIN16, ULAW, ALAW, LIN8, SPHERE)
	STREAM_TYPE	string	Type of stream (stream, fd, FILE)
	REWIND	bool	If true, the stream rewinds to the beginning of the file when EOF is met

Autocor (DSP:Misc)

Computes the autocorrelation of an input vector with (START <= lag <= END)

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input vector
Outputs	OUTPUT	Vector<float>	Autocorrelation vector
Parameters	START	int	Smallest lag offset (included)
	END	int	Largest lag offset (included)
	CONTINUOUS	bool	Use the previous frame also (cross-correlation) (default false)
	NORMALIZE	bool	Energy normalization (default false)
	NORMALIZE2	bool	Normalize by subtracting the value at lag/2 (default false)

BWExpan (DSP:Adaptive)

Performs bandwidth expansion on an LPC filter, that is, multiplying the radius of the poles by GAMMA

	Name	Type	Description
Inputs	INPUT	Vector<float>	Original LPC filter
Outputs	OUTPUT	Vector<float>	New "bandwidth expanded" LPC filter
Parameters	GAMMA	float	Pole radius factor

BroadcastLoad (IO)
Load an object from file (registered type)

	Name	Type	Description
Inputs	SOCKET	Stream	The stream we are loading from
Outputs	OUTPUT	any	The loaded object
Parameters	none		

BroadcastSave (IO)
Takes an object and saves it using a stream, returns the input object

	Name	Type	Description
Inputs	OBJECT	any	The object that will be saved
	SOCKET	Stream	The output stream where to save
Outputs	OUTPUT	any	The input object
Parameters	none		

BuildDoc (General) (require: UIClasses) Builds a network from a document

	Name	Type	Description
Inputs	INPUT	UIDocument	Loaded document
Outputs	OUTPUT	Network	built network
Parameters	none		

CAllPole (ZDeprecated) No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
	FILTER	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	none		

CGain (DSP:Base)
No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
	GAIN	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	none		

CMCalc (VQ) (require: CMap) No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
	CM	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	OUTPUTLENGTH	any	No description available

CMS (DSP:TimeFreq)

Window-type Cepstram Mean Subtraction (CMS)

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input frames (cepstrum)
Outputs	OUTPUT	Vector<float>	CMS output frames
Parameters	LENGTH	int	Frame length (number of features)
	LOOKBACK	int	CMS window look-back (number of frames)
	LOOKAHEAD	int	CMS window look-ahead (number of frames)

CMTrain (VQ) (require: CMap) Trains a codebook map

	Name	Type	Description
Inputs	TRAIN_IN	Vector<ObjectRef>	Input feature accumulator
	TRAIN_OUT	Vector<ObjectRef>	Output feature accumulator
	VQ	KMeans	Already trained vector quantizer
Outputs	OUTPUT	CodebookMap	Resulting codebook map
Parameters	none		

Catch (Flow)

Catches an exception

	Name	Type	Description
Inputs	INPUT	any	Normal flow
	CATCH	any	Flow to follow if an exception is caught
Outputs	OUTPUT	any	Flow output
	EXCEPTION	any	The exception caught (use only as feedback link)
Parameters	none		

Collector (Flow)

Pass through with unlimited number of input/output pairs

	Name	Type	Description
Inputs	INPUT	any	The input
Outputs	OUTPUT	any	The output = The input (same name)
Parameters	none		

Concat (Operator)

Concatenates two input values and returns the result

	Name	Type	Description
Inputs	INPUT1	any	First value
	INPUT2	any	Second value
Outputs	OUTPUT	any	Result of the concatenation
Parameters	none		

Concatenate (Vector)

Concatenates two vectors together (deprecated, use Concat instead)

	Name	Type	Description
Inputs	INPUT1	Vector<float>	First input vector
	INPUT2	Vector<float>	Second input vector
Outputs	OUTPUT	Vector<float>	Concatenated vector
Parameters	none		

Conj (DSP:Base)

Computes the complex conjugate of a vector

	Name	Type	Description
Inputs	INPUT	Vector<complex>	Input vector
Outputs	OUTPUT	Vector<complex>	Conjugate vector
Parameters	none		

Connect (Network)

Create a network socket of any type

	Name	Type	Description
Inputs	SOCKET	socket	The socket to listen to
	HOST	any	The host we want to connect to.
Outputs	SOCKET	stream	The socket to be used for input/output operations
Parameters	none		

Constant (General)

Defines a constant in terms of type and value. The different types are: int, float, bool, string, objects and subnet_param.

	Name	Type	Description
Inputs	none		
Outputs	VALUE	any	The value (parameter)
Parameters	VALUE	any	The value

ConstantVector (ZDeprecated)

Creates a Constant vector

	Name	Type	Description
Inputs	none		
Outputs	OUTPUT	Vector<float>	The vector
Parameters	VALUE	string	The string representation of the vector

CovarianceAccum (Matrix)
Updates (accumulate) a covariance matrix with an observation vector

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input (observation) vector
	MATRIX	Matrix<float>	Input (covariance) matrix
Outputs	OUTPUT	Matrix<float>	Updated matrix (same object as input)
Parameters	none		

DCMatrix (Matrix)
Creates a matrix of identical values

	Name	Type	Description
Inputs	none		
Outputs	OUTPUT	Matrix<float>	The matrix
Parameters	ROWS	int	Number of rows
	COLUMNS	int	Number of columns
	VALUE	float	Value of each element

DCT (DSP:TimeFreq) (require: FFT) Fast implementation of the discrete cosine transform (DCT) using an FFT

	Name	Type	Description
Inputs	INPUT	Vector<float>	The input vector
Outputs	OUTPUT	Vector<float>	The result of the DCT
Parameters	LENGTH	int	Length of the DCT
	FAST	bool	If true, the DCT is implemented using an FFT
	OUTPUTLENGTH	int	Number of coefficients to calculate (only if FAST=false)

DCVector (Vector)
Creates a vector of identical values

	Name	Type	Description
Inputs	none		
Outputs	OUTPUT	Vector<float>	The vector
Parameters	LENGTH	int	The vector length
	VALUE	float	Value of each element

DTMF (DSP:Audio)
Generates a DTMF signal

	Name	Type	Description
Inputs	INPUT	Vector<int>	DTMF vectors (line/column, starting at 0)
Outputs	OUTPUT	Vector<float>	DTMF frames
Parameters	LENGTH	int	Frame length
	SAMPLING	int	Sampling
	GAIN	float	Value of the gain

Delay (Flow)

Delay the input of DELAY iterations. Therefore, it can only be used in iterators.

	Name	Type	Description
Inputs	INPUT	any	The input object
Outputs	OUTPUT	any	The output object = input object with a delay
Parameters	DELAY	int	The delay

DiagGMMScore (HMM) (require: DGMM) Scores a DiagGMM

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input vector
	GMM	DiagGMM	Input GMM (pdf)
Outputs	OUTPUT	Float	GMM score
Parameters	none		

DiagGMMSetScore (HMM) (require: DGMM) Scores a DiagGMM

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input vector
	GMM	DiagGMM	Input GMM set (pdf's)
Outputs	OUTPUT	Vector<float>	GMM scores
Parameters	none		

Discard (General)

Discards the object pulled

	Name	Type	Description
Inputs	INPUT	any	The input object
Outputs	OUTPUT	NilObject	Always return a NilObject
Parameters	none		

Dist (DSP:Misc)

Calculates the distance between two vectors

	Name	Type	Description
Inputs	INPUT1	Vector<float>	First input vector
	INPUT2	Vector<float>	Second input vector
Outputs	OUTPUT	Vector<float>	Distance between INPUT1 and INPUT2
Parameters	none		

Div (Operator)
Divides a numerator by a denominator

	Name	Type	Description
Inputs	NUM	any	The numerator
	DEN	any	The denominator
Outputs	OUTPUT	any	The result of the division
Parameters	none		

DownSample (DSP:Base)
Downsamples a signal by outputing one sample for every N input samples

	Name	Type	Description
Inputs	INPUT	Vector<float>	Downsampling input
Outputs	OUTPUT	Vector<float>	Downsampled (by N) output
Parameters	FACTOR	int	Downsampling factor N

Entropy (DSP:Misc)
Calculates the entropy of a vector

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input vector
Outputs	OUTPUT	Vector<float>	Entropy value (vector of 1)
Parameters	none		

Equal (Operator)
Returns true if two input values are equal, false otherwise

	Name	Type	Description
Inputs	INPUT1	any	First value
	INPUT2	any	Second value
Outputs	OUTPUT	bool	True or false
Parameters	none		

ExecStream (IO)
A command to be executed (stdout is streamed)

	Name	Type	Description
Inputs	INPUT	string	The command arg
Outputs	OUTPUT	Stream	The stream
Parameters	COMMAND	string	The command

Exp (DSP:Base)

Computes the exponential (base-e) of a vector

	Name	Type	Description
Inputs	INPUT	Vector<float>	The input of the exponential
Outputs	OUTPUT	Vector<float>	Result of the exponential
Parameters	FAST	bool	Should we use exponential approximation

FDSaveFrame (ZDeprecated)

Writes audio frames to the sound card (or any other) file descriptor (deplaced by WriteAudio)

	Name	Type	Description
Inputs	OBJECT	Vector<float>	Audio frames
	FD	FILEDES	(Sound card) File descriptor
Outputs	OUTPUT	Vector<float>	Returning the input audio frames
Parameters	LEAD_IN	int	Number of zero frames to send before starting (for synchronization)

FFT (DSP:TimeFreq) (require: FFT) Computes the real FFT of a float vector

	Name	Type	Description
Inputs	INPUT	Vector<float>	The input vector
Outputs	OUTPUT	Vector<float>	The FFT resuls as [r(0), r(1),..., r(N/2), i(N/2-1), ..., i(2), i(1)]
Parameters	none		

FFTFlip (DSP:Base)

Flips a half-spectrum to produce a symetric spectrum

	Name	Type	Description
Inputs	INPUT	Vector<float>	Half spectrum (real)
Outputs	OUTPUT	Vector<float>	Symetric spectrum
Parameters	none		

FIR (DSP:Filter)

Finite Impulse Response (FIR) filter

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input frame
	FILTER	Vector<float>	Filter coefficients
Outputs	OUTPUT	Vector<float>	Filtered output
Parameters	CONTINUOUS	bool	Should the frames be considered continuous (filter with memory). Default is true
	NONCAUSAL	int	Non-causality in number of samples. Default is causal filter

FLog (DSP:Base)
Computes the natural logarithm of a vector using a *rough* appriximation (only 17 MSB used)

	Name	Type	Description
Inputs	INPUT	Vector<float>	The input of the log
Outputs	OUTPUT	Vector<float>	Result of the log
Parameters	none		

FMapCalc (VQ) (require: FeatureMap) Calculates the result of an hetero-associative map (trained by FMapTrain) for an input vector

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input vector
	FMAP	FeatureMap	The feature map that will be used
Outputs	OUTPUT	Vector<float>	Output features
Parameters	INPUTLENGTH	int	Number of input features
	OUTPUTLENGTH	int	Number of output features

FMapTrain (VQ) (require: FeatureMap) Trains an hetero-associative map based on a decision tree.

	Name	Type	Description
Inputs	TRAIN_IN	Vector<ObjectRef>	An accumulator with input features
	TRAIN_OUT	Vector<ObjectRef>	An accumulator with input features
Outputs	OUTPUT	FeatureMap	The trained 'feature map'
Parameters	LEVELS	any	Number of levels to the decision tree

Feedback (Flow)
Feedback objects with a delay of n iterations.

	Name	Type	Description
Inputs	INPUT	any	The input object
	BEFORE	any	When count < delay, pull the input from here
Outputs	OUTPUT	any	The output object = input object
	DELAY	any	The delayed output of DELAY iteration
Parameters	DELAY	int	Number of iteration for the delay
	BEFORE_LIMIT	int	When count - DELAY is smaller or equal to BEFORE_LIMIT, the input is pulled from BEFORE at (DELAY - count + BEFORE_LIMIT)

Filter (DSP:Filter) No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
	FIR	any	No description available
	IIR	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	none		

Float2Vect (Vector)

Converts float values to a vector of elements in past and future

	Name	Type	Description
Inputs	INPUT	float	The input float
Outputs	OUTPUT	Vector<float>	The vector
Parameters	LOOKAHEAD	int	Number of elements in the future
	LOOKBACK	int	Number of elements in the past

Floor (DSP:Base)

Floors vector values

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input vector
Outputs	OUTPUT	Vector<float>	Output vector (after flooring)
Parameters	THRESH	float	Threshold

FrameLabel (DSP:Audio)

Applies a gain to a vector

	Name	Type	Description
Inputs	INPUT	Stream	Input stream
Outputs	OUTPUT	String	Frame label
Parameters	FRAME_ADVANCE	int	Frame advance to use

FuzzyModelExec (Fuzzy)

FuzzyModelExec takes a FuzzyModel and find its output according the the specified intput.

	Name	Type	Description
Inputs	MODEL	any	The model to use
	INPUT	any	The input values to calculate
Outputs	OUTPUT	any	The output of the fuzzy model
Parameters	none		

FuzzyRule (Fuzzy)

A Rule containing ANTECEDANTS (IF) and CONSEQUENTS(THEN)

	Name	Type	Description
Inputs	none		
Outputs	RULE	Vector<ObjectRef>	The FuzzyRule Object
Parameters	IF	string	Antecedant of the rule seperated by spaces
	THEN	string	Consequent of the rule seperated by spaces

FuzzySet (Fuzzy)

A FuzzySet containing functions associated with names

	Name	Type	Description
Inputs	FUNCTIONS	Vector<ObjectRef>	The Fuzzy Functions
Outputs	SET	Vector<ObjectRef>	The FuzzySet with multiple Fuzzy Functions
Parameters	NAME	string	The name of the set

GCMS (DSP:TimeFreq)

Growing-Window Cepstral Mean Subtraction

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input frames
Outputs	OUTPUT	Vector<float>	CMS output
Parameters	LENGTH	int	Frame lentgh (features)

GCMS2 (DSP:TimeFreq)

Growing-Window Cepstral Mean Subtraction, counting only speech

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input frames
	IS_SPEECH	bool	Whether the frame is speech
Outputs	OUTPUT	Vector<float>	CMS output
Parameters	LENGTH	int	Frame lentgh (features)

GMMScore (HMM)

Scores a GMM for a given frame

	Name	Type	Description
Inputs	FRAMES	Vector<float>	Frames that will be scored
	GMM	GMM	GMM used as dpf
Outputs	OUTPUT	Vector<float>	Log-score (as a vector of 1)
Parameters	none		

GMMTrain (HMM)

Trains a GMM using an accumulator of frames

	Name	Type	Description
Inputs	FRAMES	Vector<ObjectRef>	Frame Accumulator
Outputs	OUTPUT	GMM	The trained GMM
Parameters	SPLIT_LEVELS	int	Number of times to perform the split = log2 (number of gaussians)

Gain (DSP:Base)

Applies a gain to a vector

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input vector
Outputs	OUTPUT	Vector<float>	Output vector (after gain)
Parameters	GAIN	float	Value of the gain

GenericModel (Fuzzy)

A generic Fuzzy controller

	Name	Type	Description
Inputs	RULES	Vector<ObjectRef>	The Rules to use
	ANTECEDENT_SETS	Vector<ObjectRef>	The Sets to use
	CONSEQUENT_SETS	Vector<ObjectRef>	The Sets to use
	INPUT	Vector<float>	The input value of the variables
Outputs	MODEL	Model	The model (cloned)
	OUTPUT	Vector<float>	The defuzzified values
Parameters	none		

GetComposite (General)

Split up a composite object. This node makes just the opposite of the node "MakeComposite", that is, split up his compressed input (the composite object) into several outputs. However, the outputs must be added manually by by users. To add outputs to the node: double-click on it and click on the tab "Input/Outputs". Give a name to the output and press "Add". Repeat as long as you wish. Therefore, you can regroup inputs with "MakeComposite", send them in one output (the composite object) and get them back with "GetComposite". However, if you want to do so, inputs of "MakeComposite" and outputs of "GetComposite" must have corresponding names.

	Name	Type	Description
Inputs	INPUT	CompositeType	Composite object
Outputs	none		
Parameters	none		

Greater (Logic)

Verifies if INPUT1 > INPUT2

	Name	Type	Description
Inputs	INPUT1	any	The first operand
	INPUT2	any	The second operand
Outputs	OUTPUT	bool	bool value
Parameters	none		

GtkPlotProbe (Probe)

Plots a vector using GtkPlot

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input vector
Outputs	OUTPUT	Vector<float>	Same as input
Parameters	BREAK_AT	int	If set, the probe runs until (count = BREAK_AT)
	SHOW	bool	Whether or not to show the the data by default
	SKIP	int	Count increment for each "Next"
	PROBE_NAME	string	Name (title) of the probe

HP (DSP:Filter) No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	DELAY	int	No description available
	FILTER_LENGTH	int	No description available
	FREQ	float	No description available

HistoVect (DSP:Manip)

No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	LENGTH	any	No description available

IDCT (DSP:TimeFreq) (require: FFT) Fast implementation of the inverse discrete cosine transform (IDCT) using an FFT

	Name	Type	Description
Inputs	INPUT	Vector<float>	The input vector
Outputs	OUTPUT	Vector<float>	The result of the DCT
Parameters	LENGTH	int	Length of the input and output vectors

IF (Logic)

Takes a branch or another depending on a condition (Bool value).

	Name	Type	Description
Inputs	COND	bool	The condition for the if statement
	THEN	any	What to do if the condition is true
	ELSE	any	What to do if the condition is false
Outputs	OUTPUT	any	The object from THEN or ELSE depending on COND
Parameters	PULL_ANYWAY	bool	If true, the IF statement pulls also on the branch not taken

IIR (DSP:Filter)
All-pole (IIR) filter

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input signal (frames)
	FILTER	Vector<float>	Filter coefficients (denominator)
Outputs	OUTPUT	Vector<float>	Filtered signal
Parameters	none		

ILTF (DSP:Filter)
Inverse (all-pole) long-term (comb) filter

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input frame
	FILTER	Vector<float>	Filter params as [gain, period]
Outputs	OUTPUT	Vector<float>	Filtered signal
Parameters	none		

IRFFT (DSP:TimeFreq) (require: FFT) Inverse FFT, half complex to real vector

	Name	Type	Description
Inputs	INPUT	Vector<complex>	Half complex vector
Outputs	OUTPUT	Vector<float>	Real inverse FFT output
Parameters	none		

Index (Vector)
Returns a float value from a vector

	Name	Type	Description
Inputs	INPUT	Vector<float>	The input vector
	INDEX	int	Index value (if not specified in parameter)
Outputs	OUTPUT	float	Float at a certain index
Parameters	INDEX	int	Index value

Index2Vector (DSP:Manip)
No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	LENGTH	any	No description available

InferenceModel (Fuzzy)
A generic Fuzzy controller

	Name	Type	Description
Inputs	RULES	Vector<ObjectRef>	The Rules to use
	ANTECEDENT_SETS	Vector<ObjectRef>	The Sets to use
	CONSEQUENT_SETS	Vector<ObjectRef>	The Sets to use
	INPUT	Vector<float>	The input value of the variables
Outputs	MODEL	Model	The model (cloned)
	OUTPUT	Vector<float>	The defuzzified values
	OUTPUT_SETS	Vector<ObjectRef>	The copied consequent set(s)
Parameters	none		

InputStream (IO)
Creates a read-only stream from a filename

	Name	Type	Description
Inputs	INPUT	string	The file name
Outputs	OUTPUT	Stream	The Stream
Parameters	TYPE	String	Type of stream: stream, fd, or FILE (default stream)
	RETRY	int	If set to N, InputStream will retry N times on open fail

IterCount (Logic)
Get the iterator count (iteration number)

	Name	Type	Description
Inputs	none		
Outputs	OUTPUT	int	The iteration count
Parameters	none		

IterWall (Flow)
Get the input object only once, compute the result and always give the same output afterwards.

	Name	Type	Description
Inputs	INPUT	any	The input object
Outputs	OUTPUT	any	The output object = the input object (calculated once)
Parameters	none		

Iterate (Flow)
Specify the number of iteration to do (max). Therefore, it can only be used

in iterators and the output must be set to COND (left click on the node output and hold the control).

	Name	Type	Description
Inputs	none		
Outputs	OUTPUT	bool	Return true if count < MAX_ITER
Parameters	MAX_ITER	int	Number of iteration to do (max)

KeyPad (Probe)
No description available

	Name	Type	Description
Inputs	none		
Outputs	KEYPAD	Vector<int>	A vector of size 2 representing the tuple Line/column of the pressed key.
	KEYPAD_ID	int	The Id of the key that is pressed
	KEYPAD_NAME	Char	The Char description of the key that is pressed
	ACTIVATED	bool	True if the user is pressing a button, else false.
Parameters	none		

LP (DSP:Filter) No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	DELAY	int	No description available
	FILTER_LENGTH	int	No description available
	FREQ	float	No description available

LPC (DSP:Adaptive)
Performs LPC (Linear predictive coefficient) analysis

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input (audio) vector
Outputs	OUTPUT	Vector<float>	LPC coefficients (including a[0]=1)
Parameters	OUTPUTLENGTH	int	Number of LPC coefficients (order = OUTPUTLENGTH-1)
	RADIUS	float	Maximum radius of the poles (used for bandwidth expansion)
	LAG_THETA	float	Minimum resonnance bandwidth allowed (with lag-windowing, approximative)

LPC2PS (DSP:Adaptive) (require: FFT) Calculates the spectral envelope corresponding to an all-pole filter (LPC coefficients)

	Name	Type	Description
Inputs	INPUT	Vector<float>	LPC coefficients (including the '1' as first coefficient)
Outputs	OUTPUT	Vector<float>	Points of the spectral envelope
Parameters	OUTPUTLENGTH	int	Number of points for the spectral envelope

LPC_DECOMP (DSP:Adaptive) No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
Outputs	LPC	any	No description available
	EXC	any	No description available
Parameters	LPC_SIZE	int	No description available
	LAG_THETA	float	No description available
	FRAME_SIZE2	int	No description available

LPFilter (DSP:Filter)
No description available

	Name	Type	Description
Inputs	none		
Outputs	OUTPUT	any	No description available
Parameters	LENGTH	any	No description available
	THETA	any	No description available
	HP	any	No description available

LTF (DSP:Filter)
Long-term (comb) filter

	Name	Type	Description
Inputs	INPUT	any	No description available
	FILTER	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	none		

LTP (DSP:Adaptive)
Long-term predictor, finds best correlation (pitch) within (START <= sample delay <= END)

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input frame
Outputs	OUTPUT	Vector<float>	[pitch gain, pitch period]
Parameters	START	int	Smallest pitch allowed
	END	int	Largest pitch allowed

Length (Vector)
Get the length of a vector

	Name	Type	Description
Inputs	INPUT	any	The vector input
Outputs	OUTPUT	int	The length of the vector
Parameters	none		

List (General)

Load a string from a file (seperated into chunks of 256 bytes)

	Name	Type	Description
Inputs	STREAM	Stream	The stream to load from
Outputs	OUTPUT	Vector<ObjectRef>	The vector output
Parameters	none		

Listen (Network)

Create a network socket of any type

	Name	Type	Description
Inputs	SOCKET	socket	The socket to listen to
Outputs	SOCKET	socket	The socket to be used for input/output operations
Parameters	BACKLOG	int	Number of incoming connections allowed
	BLOCKING	bool	Blocking call to accept.

Load (IO)

Load an object from file (registered type)

	Name	Type	Description
Inputs	STREAM	Stream	The stream we are loading from
Outputs	OUTPUT	any	The loaded object
Parameters	none		

LoadDoc (General) (require: UIClasses) Loads an Overflow XML document

	Name	Type	Description
Inputs	INPUT	string	Document name
Outputs	OUTPUT	UIDocument	loaded document
Parameters	none		

LoadFile (IO) No description available

	Name	Type	Description
Inputs	none		
Outputs	OUTPUT	any	No description available
Parameters	FILENAME	string	No description available

Log (DSP:Base)

Computes the natural logarithm of a vector

	Name	Type	Description
Inputs	INPUT	Vector<float>	The input of the log
Outputs	OUTPUT	Vector<float>	Result of the log
Parameters	FAST	bool	Should we use fast log approximation

MDCT (DSP:TimeFreq) (require: MDCT) MDCT implementation (taken from Vorbis)

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input frame
Outputs	OUTPUT	Vector<float>	MDCT result
Parameters	LENGTH	int	Frame (not window) size

MFCC (ZDeprecated)

Calculates MFCC coefficients from an audio frame (all in one)

	Name	Type	Description
Inputs	INPUT	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	INPUTLENGTH	any	No description available
	OUTPUTLENGTH	any	No description available
	WINDOW	any	No description available
	SAMPLING	any	No description available
	LOW	any	No description available
	HIGH	any	No description available

MMIScore (VQ)

No description available

	Name	Type	Description
Inputs	FRAMES	any	No description available
	MMI	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	none		

MMITrain (VQ)

Train Maximum Mutual Information (MMI) Tree

	Name	Type	Description
Inputs	FRAMES	Vector<ObjectRef>	No description available
Outputs	OUTPUT	Cell	MMI tree
Parameters	LEVELS	int	Number of levels for the tree

MSVQTrain (VQ) (require: MSVQ) Training of a multi-stage vector quantizer

	Name	Type	Description
Inputs	FRAMES	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	STAGES	any	No description available
	BINARY	any	No description available

MakeComposite (General)

Creates a composite object. A composite object is somewhere like a structure in C++. Indeed, a composite object is a regrouping of inputs like a structure is a regrouping of variables. You can chose the name and the number of inputs that you want your node to containt. To add inputs in the node: double-click on it and click on the tab "Input/Outputs". Give a name to the input and press "Add". Repeat as long as you wish. Then, the node will regoup all of these inputs together in only one output (a composite object).

	Name	Type	Description
Inputs	none		
Outputs	OUTPUT	any	Create a new composite object.
Parameters	none		

MakeDiagGMM (HMM) (require: GMM) Transforms a GMM into a DiagGMM

	Name	Type	Description
Inputs	INPUT	GMM	Input GMM
Outputs	OUTPUT	DiagGMM	Output DiagGMM
Parameters	none		

MarkovProb (HMM)

Calculates the Markov chain probability

	Name	Type	Description
Inputs	INPUT	any	State probability
	MATRIX	any	Transition probability matrix
Outputs	OUTPUT	any	A posteriori probability
Parameters	none		

MatProduct (Matrix)

Matrix x vector product

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input vector
	MATRIX	Matrix<float>	Matrix
Outputs	OUTPUT	Vector<float>	Result
Parameters	none		

Max (Operator)

The maximum value

	Name	Type	Description
Inputs	INPUT1	any	First value
	INPUT2	any	Second value
Outputs	OUTPUT	any	The maximum value between INPUT1 and INPUT2
Parameters	none		

Mel (DSP:TimeFreq)
calculates Mel-scale channel energies from power-spectrum

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input power-spectrum
Outputs	OUTPUT	Vector<float>	Mel-scale channel energies
Parameters	INPUTLENGTH	int	Power-spectrum size
	OUTPUTLENGTH	int	Number of channel energies
	SAMPLING	int	Sampling rate used (used for power-spectrum range)
	LOW	int	Lowest frequency
	HIGH	int	Highest frequency

MergeChannels (DSP:Audio)
No description available

	Name	Type	Description
Inputs	LEFT	any	No description available
	RIGHT	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	none		

Min (Operator)
Selects the minimum between two values

	Name	Type	Description
Inputs	INPUT1	any	The first value
	INPUT2	any	The second value
Outputs	OUTPUT	any	The minimum value between INPUT1 and INPUT2
Parameters	none		

Mul (Operator)
Multiplication between two values, vectors, objects (operator* must be defined)

	Name	Type	Description
Inputs	INPUT1	any	The first operand
	INPUT2	any	The second operand
Outputs	OUTPUT	any	The result of INPUT1 * INPUT2
Parameters	none		

MultiPlotProbe (Probe)

Plots multiple vectors using GtkPlot

	Name	Type	Description
Inputs	INPUT	Vector<ObjectRef>	Input vectors
Outputs	OUTPUT	Vector<float>	Same as input
Parameters	BREAK_AT	int	If set, the probe runs until (count = BREAK_AT)
	SHOW	bool	Whether or not to show the the data by default
	SKIP	int	Count increment for each "Next"
	PROBE_NAME	string	Name (title) of the probe

NLMS (DSP:Adaptive)

Normalized LMS algorithm

	Name	Type	Description
Inputs	INPUT	any	The input of the adaptive FIR filter
	REF	any	The signal being tracked
Outputs	OUTPUT	any	The output of the adaptive FIR filter (not the residue)
Parameters	FILTER_LENGTH	any	Length of the adaptive FIR filter
	ALPHA	any	Adaptation rate of the filter coefficients
	BETA	any	Adaptation rate of the normalization energy estimate
	POWER	any	Normalization power

NNetExec (NNet) (require: FFNet) No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
	NNET	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	OUTPUTLENGTH	any	No description available

NNetExecRecurrent (NNet)

No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
	NNET	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	OUTPUTLENGTH	any	No description available

NNetInit (NNet) (require: FFNet) Initialized the neural network weights to fit the input/output set

	Name	Type	Description
Inputs	TRAIN_IN	Vector<ObjectRef>	Training input data
	TRAIN_OUT	Vector<ObjectRef>	Training output data
Outputs	OUTPUT	FFNet	Initialized feed-forward neural network
Parameters	TOPO	string	Number of units on each layer (including input and output layers)
	FUNCTIONS	string	Activation functions for each layer (except the input layer)
	RAND_SEED	int	Sets to random seed to RAND_SEED before initialization

NNetNew (NNet)

Returns a new (MLP) neural network

	Name	Type	Description
Inputs	none		
Outputs	OUTPUT	NNet	New (MLP) neural network
Parameters	TOPO	Vector<string>	No description available

NNetSetCalc (NNet)

No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
	ID	any	No description available
	NNET	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	OUTPUTLENGTH	any	No description available

NNetSetChooseBest (NNet)

Initialized the neural network weights to fit the input/output set

	Name	Type	Description
Inputs	TRAIN_IN	any	No description available
	TRAIN_OUT	any	No description available
	TRAIN_ID	any	No description available
	NET1	any	No description available
	NET2	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	none		

NNetSetInit (NNet)

Initialized the neural network weights to fit the input/output set

	Name	Type	Description
Inputs	TRAIN_IN	any	No description available
	TRAIN_OUT	any	No description available
	TRAIN_ID	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	NB_NETS	any	No description available
	TOPO	any	No description available
	FUNCTIONS	any	No description available
	RAND_SEED	any	No description available

NNetSetTrain (NNet)

No description available

	Name	Type	Description
Inputs	TRAIN_IN	any	No description available
	TRAIN_OUT	any	No description available
	TRAIN_ID	any	No description available
	NNET	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	MAX_EPOCH	any	No description available
	LEARN_RATE	any	No description available
	MOMENTUM	any	No description available
	INCREASE	any	No description available
	DECREASE	any	No description available
	ERR_RATIO	any	No description available
	BATCH_SETS	any	No description available

NNetSetTrainCGB (NNet)

No description available

	Name	Type	Description
Inputs	TRAIN_IN	any	No description available
	TRAIN_OUT	any	No description available
	TRAIN_ID	any	No description available
	NNET	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	MAX_EPOCH	any	No description available
	SIGMA	any	No description available
	LAMBDA	any	No description available

NNetSetTrainDBD (NNet)

No description available

	Name	Type	Description
Inputs	TRAIN_IN	any	No description available
	TRAIN_OUT	any	No description available
	TRAIN_ID	any	No description available
	NNET	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	MAX_EPOCH	any	No description available
	LEARN_RATE	any	No description available
	INCREASE	any	No description available
	DECREASE	any	No description available

NNetTrain (NNet) (require: FFNetTrain) No description available

	Name	Type	Description
Inputs	TRAIN_IN	any	No description available
	TRAIN_OUT	any	No description available
	NNET	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	MAX_EPOCH	any	No description available
	LEARN_RATE	any	No description available
	MOMENTUM	any	No description available
	INCREASE	any	No description available
	DECREASE	any	No description available
	ERR_RATIO	any	No description available
	BATCH_SETS	any	No description available

NNetTrainCGB (NNet) (require: FFNetTrain) No description available

	Name	Type	Description
Inputs	TRAIN_IN	any	No description available
	TRAIN_OUT	any	No description available
	NNET	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	MAX_EPOCH	any	No description available
	SIGMA	any	No description available
	LAMBDA	any	No description available

NNetTrainDBD (NNet) (require: FFNetTrain) Neural network (MLP) training unsing the Delta-bar-delta algorithm

	Name	Type	Description
Inputs	TRAIN_IN	Vector<ObjectRef>	Input data accumulator
	TRAIN_OUT	Vector<ObjectRef>	Output data accumulator
	NNET	FFNet	Neural network that will be trained
Outputs	OUTPUT	FFNet	Trained network
Parameters	MAX_EPOCH	int	Number of training epoch (default 2000)
	LEARN_RATE	float	Initial learning rate (default 0.000001)
	INCREASE	float	Learning rate increment (> 1.0) factor (default 1.04)
	DECREASE	float	Learning rate decrement (< 1.0) factor (default 0.6)
	NB_SETS	int	Number of batch subsets for accelerated training (default 1)
	ALLOC_CHUNK	bool	If true, a big vector is allocated to store all the inputs (default false)
	RPROP	bool	If true, use the RProp variant of delta-bar-delta (default false)

NNetTrainQProp (NNet) (require: FFNetTrain) Neural network (MLP) training unsing the Quickprop algorithm

	Name	Type	Description
Inputs	TRAIN_IN	Vector<ObjectRef>	Input data accumulator
	TRAIN_OUT	Vector<ObjectRef>	Output data accumulator
	NNET	FFNet	Neural network that will be trained
Outputs	OUTPUT	FFNet	Trained network
Parameters	MAX_EPOCH	int	Number of training epoch (default 2000)
	LEARN_RATE	float	Initial learning rate (default 0.000001)
	INCREASE	float	Learning rate increment (> 1.0) factor (default 1.04)
	DECREASE	float	Learning rate decrement (< 1.0) factor (default 0.6)

NNetTrainRecurDBD (NNet) (require: FFNetTrain) No description available

	Name	Type	Description
Inputs	TRAIN_IN	any	No description available
	TRAIN_OUT	any	No description available
	NNET	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	MAX_EPOCH	any	No description available
	LEARN_RATE	any	No description available
	MOMENTUM	any	No description available
	INCREASE	any	No description available
	DECREASE	any	No description available
	BATCH_SETS	any	No description available

NNetTrainRecurrent (NNet) (require: FFNetTrain) No description available

	Name	Type	Description
Inputs	TRAIN_IN	any	No description available
	TRAIN_OUT	any	No description available
	NNET	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	MAX_EPOCH	any	No description available
	LEARN_RATE	any	No description available
	MOMENTUM	any	No description available
	INCREASE	any	No description available
	DECREASE	any	No description available

NNetTrainSCG (NNet) (require: FFNetTrain) Neural network (MLP) training unsing the scaled conjugate gradient algorithm

	Name	Type	Description
Inputs	TRAIN_IN	Vector<ObjectRef>	Input data accumulator
	TRAIN_OUT	Vector<ObjectRef>	Output data accumulator
	NNET	FFNet	Neural network that will be trained
Outputs	OUTPUT	FFNet	Trained network
Parameters	MAX_EPOCH	int	Number of training epoch (default 2000)
	SIGMA	float	Sigma parameter
	LAMBDA	float	Lambda parameter

NNetTrainWeightDBD (NNet) (require: FFNetTrain) Neural network (MLP) training unsing the Delta-bar-delta algorithm

	Name	Type	Description
Inputs	TRAIN_IN	Vector<ObjectRef>	Input data accumulator
	TRAIN_OUT	Vector<ObjectRef>	Output data accumulator
	TRAIN_WEIGHT	Vector<ObjectRef>	Error weights for training
	NNET	FFNet	Neural network that will be trained
Outputs	OUTPUT	FFNet	Trained network
Parameters	MAX_EPOCH	int	Number of training epoch (default 2000)
	LEARN_RATE	float	Initial learning rate (default 0.000001)
	INCREASE	float	Learning rate increment (> 1.0) factor (default 1.04)
	DECREASE	float	Learning rate decrement (< 1.0) factor (default 0.6)

NOP (General)

Pass Through (no operation)

	Name	Type	Description
Inputs	INPUT	any	The input
Outputs	OUTPUT	any	The output = The input
Parameters	none		

NOT (Logic)

Logical NOT of an input

	Name	Type	Description
Inputs	INPUT	bool	Boolean input
Outputs	OUTPUT	bool	Boolean output
Parameters	none		

NewAccumulator (General)

Creates a new Accumulator, that is a vector of Objects References. Accumulators are often used as the input "ACCUM" of the node "Accumulate".

	Name	Type	Description
Inputs	none		
Outputs	OUTPUT	Vector<ObjectRef>	Empty accumulator
Parameters	none		

Noise (DSP:Misc)

Noise generator

	Name	Type	Description
Inputs	none		
Outputs	OUTPUT	Vector<float>	Noise signal (uncorrelated)
Parameters	LENGTH	int	Length of the generated noise signal (frame length)
	TYPE	string	Noise type (UNIFORM, GAUSS, TRIANGLE)
	SD	float	Noise standard deviation

Normalize (DSP:Base)
Normalizes a vector by dividing each element by the sum of all components

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input vector
Outputs	OUTPUT	Vector<float>	Normalized vector
Parameters	none		

OR (Logic)
Logical OR between two inputs

	Name	Type	Description
Inputs	INPUT1	bool	First boolean input
	INPUT2	bool	Second boolean input
Outputs	OUTPUT	bool	Boolean output
Parameters	PULL_ANYWAY	bool	Pull on INPUT2 even if INPUT1 is true

OffsetMatrix (DSP:Manip)
Returns a matrix of frames with offset

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input frame
Outputs	OUTPUT	Matrix<float>	Matrix (ready for SVD, ...)
Parameters	COLUMNS	int	Number of columns (subframe length)
	ROWS	int	Number of rows (number of offsets)

OutputStream (IO)
Creates a write-only stream from a filename

	Name	Type	Description
Inputs	INPUT	string	The file name
Outputs	OUTPUT	Stream	The Stream
Parameters	TYPE	string	Type of stream: stream, fd, or FILE (default stream)

Overlap (DSP:Manip)
Outputs overlapping frames from non-overlapping ones

	Name	Type	Description
Inputs	INPUT	Vector<float>	(Non-overlapped) input frames
Outputs	OUTPUT	Vector<float>	Overlapped output frames
Parameters	OUTPUTLENGTH	int	Frame length for output overlapped frames

OverlapAndAdd (DSP:Audio)
No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	none		

PS (DSP:TimeFreq)
Converts the output of the FFT (not RFFT) node to a power spectrum

	Name	Type	Description
Inputs	INPUT	Vector<float>	Spectrum output from FFT
Outputs	OUTPUT	Vector<float>	Power spectrum (half the input length)
Parameters	none		

PS2LPC (DSP:Adaptive) (require: FFT) Computes LPC coefficients from the spectral envelope of the all-pole filter

	Name	Type	Description
Inputs	INPUT	Vector<float>	Spectral envelope
Outputs	OUTPUT	Vector<float>	LPC coefficients
Parameters	INPUTLENGTH	int	Number of points in the spectral envelope
	OUTPUTLENGTH	int	Number of LPC coefficients (order + 1)
	LAG_THETA	float	Lag-windowing parameter (roughly the minimum bandwidth of resonances)

Pack (Flow)
Pack Data into a vector of Objects references. When the node is in the main network or in a sub-network, his input is packed in the vector only once. However while in iterators, his input is packed (added) in the vector at every iteration.

	Name	Type	Description
Inputs	INPUT	any	Objects to be packed (until processCount reached)
Outputs	OUTPUT	Vector<ObjectRef>	A vector of ObjectRef(s)
Parameters	none		

PackFrames (DSP:Base)

No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	LENGTH	any	No description available
	BACK	any	No description available
	FRONT	any	No description available

ParallelThread (Flow)

Provides parallelism-type threading. When asked for an input, it computes both inputs at the same time and caches the other.

	Name	Type	Description
Inputs	INPUT1	any	First parallelized input
	INPUT2	any	Second parallelized input
Outputs	OUTPUT1	any	First parallelized output
	OUTPUT2	any	Second parallelized output
Parameters	none		

PlotProbe (Probe)

No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	BREAK_AT	any	No description available
	SHOW	any	No description available
	SKIP	any	No description available

Poly (DSP:Base)

No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
	COEF	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	none		

Pow (DSP:Base)

Raises the input vector to a certain power

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input vector
Outputs	OUTPUT	Vector<float>	Result: INPUT.
Parameters	EXP	float	Exponent

Print (IO) No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	none		

Probe (Probe)

Helps debugging by allowing to "trace" programs

	Name	Type	Description
Inputs	INPUT	any	Any data
Outputs	OUTPUT	any	Pass through
Parameters	BREAK_AT	int	If set, the probe runs until (count = BREAK_AT)
	SHOW	bool	Whether or not to show the the data by default
	SKIP	int	Count increment for each "Next"
	PROBE_NAME	string	Name (title) of the probe

RBFTrain (VQ)

No description available

	Name	Type	Description
Inputs	FRAMES	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	NB_GAUSSIANS	any	No description available

RFFT (DSP:TimeFreq) (require: FFT) Computes the FFT of a real input vector and output a complex result

	Name	Type	Description
Inputs	INPUT	Vector<float>	Real vector
Outputs	OUTPUT	Vector<complex>	Complex FFT output
Parameters	none		

RMS (DSP:Misc)

Root mean squared (RMS) value of a signal

	Name	Type	Description
Inputs	INPUT	Vector<float>	The input signal
Outputs	OUTPUT	Vector<float>	The RMS value
Parameters	none		

ReadInt (IO)

ReadInt an integer from file

	Name	Type	Description
Inputs	STREAM	Stream	The stream we are loading from
Outputs	OUTPUT	int	The (next) integer in the stream
Parameters	none		

Receive (IO)

Receive data from a TCP/IP network (Not working yet)

	Name	Type	Description
Inputs	none		
Outputs	OUTPUT	any	No description available
Parameters	VALUE	any	No description available

Recover (Flow)

Recovers from an error (BaseException)

	Name	Type	Description
Inputs	INPUT	any	Normal flow
	CATCH	any	Flow to follow if an error (exception) is caught
Outputs	OUTPUT	any	Flow output
	EXCEPTION	String	The error message caught (use only as feedback link)
Parameters	none		

Reframe (DSP:Manip)

Applies a window on a frame

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input frame
Outputs	OUTPUT	Vector<float>	Reframeed frame
Parameters	LENGTH	int	Length of the frames
	ADVANCE	int	Frame advance (offset)

Reverb (DSP:Effects) (require: Reverb) Stereo reverb effect

	Name	Type	Description
Inputs	LEFT	Vector<float>	Right Input Channel
	RIGHT	Vector<float>	Left Input Channel
Outputs	LEFT	Vector<float>	Right Output Channel
	RIGHT	Vector<float>	Left Output Channel
Parameters	ROOMSIZE	float	0 < Room Size < 1 (default = .5)
	DAMP	float	0 < Damp < 1 (default = .5)
	WET	float	0 < Wet < 1 (default = 1/3)
	DRY	float	0 < Dry < 1 (default = 0)
	WIDTH	float	0 < Width < 1 (default = 1)

Round (Found)

Rounds a float values to the nearest integer

	Name	Type	Description
Inputs	INPUT	float	The input float
Outputs	OUTPUT	int	Nearest integer
Parameters	none		

SampleAndHold (Flow)

Downsamples in the "count" domain

	Name	Type	Description
Inputs	INPUT	any	The input x[count]
Outputs	OUTPUT	any	x[count - (count "modulo" FACTOR)]
Parameters	DOWNSAMPLING	int	The downsampling factor

SampleDelay (DSP:Base)

No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
	DELAY	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	LENGTH	any	No description available
	DELAY	any	No description available
	LOOKBACK	any	No description available
	LOOKAHEAD	any	No description available

Saturate (DSP:Effects)

Saturation (distortion) effect

	Name	Type	Description
Inputs	INPUT	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	SATURATION	any	No description available
	THRESHOLD	any	No description available

Save (IO)

Takes an object and saves it using a stream, returns the input object

	Name	Type	Description
Inputs	OBJECT	any	The object that will be saved
	STREAM	Stream	The output stream where to save
Outputs	OUTPUT	any	The input object
Parameters	PRETTY_PRINT	bool	If we want to print human readable output (and Matlab)

SaveAs (IO) No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	FILENAME	string	No description available

Select (Vector)

Selects an index range in an input vector

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input vector
Outputs	OUTPUT	Vector<float>	Output vector (size = END-START+1)
Parameters	START	int	Start index (inculded)
	END	int	End index (included)

Send (General)

Send data through network (Not Working Yet)

	Name	Type	Description
Inputs	none		
Outputs	OUTPUT	any	No description available
Parameters	VALUE	string	No description available

SeparChannels (DSP:Audio)

No description available

	Name	Type	Description
Inputs	INPUT	Vector<float>	Stero frame (encoded as left, right, left, right, ...)
Outputs	LEFT	Vector<float>	Frame for the left channel
	RIGHT	Vector<float>	Frame for the right channel
Parameters	none		

SerialThread (Flow)

Provides a pipeline-type multi-threading. A thread is started and computes inputs before the are needed by the output node.

	Name	Type	Description
Inputs	INPUT	any	Flow input (asynchronous flow)
Outputs	OUTPUT	any	Output flow (synchronous)
Parameters	LOOKAHEAD	int	Pipeline look-ahead

Serialize (IO)

Takes an object and saves it using a stream, returns the input object

	Name	Type	Description
Inputs	OBJECT	any	The object that will be saved
	STREAM	Stream	The output stream where to save
Outputs	OUTPUT	any	The input object
Parameters	none		

SerializeAs (IO) No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	FILENAME	string	No description available

Sleep (Flow)

Sleep a certain amount of time.

	Name	Type	Description
Inputs	none		
Outputs	VALUE	any	Always return TRUE.
Parameters	SECONDS	float	Sleep x seconds.

Smaller (Logic)

Verifies if INPUT1 is smaller than INPUT2

	Name	Type	Description
Inputs	INPUT1	any	The first operand
	INPUT2	any	The second operand
Outputs	OUTPUT	any	Boolean output
Parameters	none		

SmoothAdd (DSP:Manip)

No description available

	Name	Type	Description
Inputs	LEFT	any	No description available
	CENTER	any	No description available
	RIGHT	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	LENGTH	any	No description available

Socket (Network)

Create a network socket of any type

	Name	Type	Description
Inputs	none		
Outputs	OUTPUT	socket	The socket to be used for input/output operations
Parameters	TYPE	string	Type of socket : BROADCAST, TCP_STREAM, etc.
	PORT	int	Communication port

Sort (DSP:Base)

Sorts an input vector

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input vector
Outputs	OUTPUT	Vector<float>	Sorted output vector
Parameters	none		

Sound (DSP:Audio)

Opens a sound device

	Name	Type	Description
Inputs	none		
Outputs	OUTPUT	any	A file descriptor to the sound device
Parameters	DEVICE	string	Path to the sound device
	RATE	int	Sampling rate
	STEREO	int	1 for stereo, 0 for mono
	MODE	string	R for sound input, W for sound output, RW for full-duplex mode
	BUFFER	int	Length of the audio buffer to allocate (not reliable)
	DUMMY	any	Put something here to output to a file

SpectrumProbe (Probe) No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	BREAK_AT	int	No description available
	SHOW	bool	No description available
	SKIP	int	No description available
	SQRT	bool	No description available
	LOG	bool	No description available

Sqrt (DSP:Base)
Square root of a vector

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input vector
Outputs	OUTPUT	Vector<float>	Result vector of square root
Parameters	none		

Stderr (IO)
Returns the stderr stream (cerr)

	Name	Type	Description
Inputs	none		
Outputs	OUTPUT	Stream	Stderr stream
Parameters	none		

Stdin (IO)
Returns the stdin stream (cin)

	Name	Type	Description
Inputs	none		
Outputs	OUTPUT	Stream	Stdin stream
Parameters	none		

Stdout (IO)

Returns the stdout stream (cout)

	Name	Type	Description
Inputs	none		
Outputs	OUTPUT	Stream	Stdout stream
Parameters	none		

StopRecord (DSP:Misc)

For SV

	Name	Type	Description
Inputs	INPUT	bool	frame by frame
Outputs	OUTPUT	bool	false when should stop
Parameters	START	int	Number of true frames before start
	TIMEOUT	int	Number of false frames before end

StrCat (ZDeprecated)

Concatenates two strings together (deprecated, use Concat instead)

	Name	Type	Description
Inputs	INPUT1	String	First input string
	INPUT2	String	Second input string
Outputs	OUTPUT	String	Concatenated strings
Parameters	none		

Sub (Operator)

Subtracts two values, Vectors, Objects

	Name	Type	Description
Inputs	INPUT1	any	The value to subtract from
	INPUT2	any	The subtracted value
Outputs	OUTPUT	any	The result of the subtraction
Parameters	none		

Sum (Vector)

Sum of all the elements of a vector

	Name	Type	Description
Inputs	INPUT	Vector<float>	The input vector
Outputs	OUTPUT	float	The sum
Parameters	none		

Sync (Flow)

No-op node for which count ratio (getInput/getOutput) = RATIO

	Name	Type	Description
Inputs	INPUT	any	Input
Outputs	OUTPUT	any	Output (no-op) same as input with different count
Parameters	RATIO	any	(input/output) count ratio

TextProbe (Probe)

Prints the data as text

	Name	Type	Description
Inputs	INPUT	any	Any data
Outputs	OUTPUT	any	Pass through
Parameters	BREAK_AT	int	If set, the probe runs until (count = BREAK_AT)
	SHOW	bool	Whether or not to show the the data by default
	SKIP	int	Count increment for each "Next"
	PROBE_NAME	string	Name (title) of the probe

ThreadJoin (Flow)

Acts like a mutex and prevents two Overflow threads from accessing the same (input) node at the same time.

	Name	Type	Description
Inputs	INPUT	any	The input
Outputs	OUTPUT	any	The output = The input
Parameters	none		

Throw (Flow)

Throw a FlowException

	Name	Type	Description
Inputs	INPUT	any	The Object included in the FlowException
Outputs	OUTPUT	any	Will automatically throw a FlowException if pulled
Parameters	none		

TimeAutocorr (DSP:Misc)

Autocorrelation across vectors (frames)

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input vectors (frames)
Outputs	OUTPUT	Vector<float>	Autocorrelations (summed) for each delay
Parameters	INPUTLENGTH	int	Length ov input vectors
	LOOKAHEAD	int	Maximum forward (non-causal) delay
	LOOKBACK	int	Maximum backward (causal) delay

TimeEntropy (DSP:Misc)

Non-stationnarity (pseudo-entropy) measure across vectors (frames)

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input vectors (frames)
Outputs	OUTPUT	Vector<float>	Value of the non-stationnarity measure (as a vector of 1 component)
Parameters	LOOKAHEAD	int	Maximum forward (non-causal) delay
	LOOKBACK	int	Maximum backward (causal) delay

TimeFilter (DSP:Filter)

Filters across vectors (frames)

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input vectors (frames)
Outputs	OUTPUT	Vector<float>	Filtered vectors
Parameters	LENGTH	int	Vector length
	FIR	string	FIR part as <Vector<float> ... >
	IIR	string	IIR part as <Vector<float> ... >
	LOOKAHEAD	int	Non-causality (in frames)

TimeMedian (DSP:Filter)

Performs median filtering across vectors (frames)

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input vectors (frames)
Outputs	OUTPUT	Vector<float>	Median-filtered vectors
Parameters	LENGTH	int	Vector size
	LOOKAHEAD	int	Median look back (number of frames)
	LOOKBACK	int	Median look ahead (number of frames)

Trace (Probe)

Pass Through, tracing initialization, requests, inputs and exceptions

	Name	Type	Description
Inputs	INPUT	any	The input
Outputs	OUTPUT	any	The output = The input
Parameters	TAG	string	Tag to put on the lines

TransMatrix (HMM)

No description available

	Name	Type	Description
Inputs	INPUT	any	state numbers in a frame buffer
Outputs	OUTPUT	any	No description available
Parameters	NB_STATES	any	Number of HMM states
	THRESHOLD	any	The minimum transition probability allowed

TrapezoidalFunction (Fuzzy)

A Fuzzy Function to be included in a FuzzySet

	Name	Type	Description
Inputs	none		
Outputs	FUNCTION	Vector<ObjectRef>	The FuzzyFunction object
Parameters	A	float	A value
	B	float	B value
	C	float	C value
	D	float	D value
	NAME	string	The name of the function

TriangularFunction (Fuzzy)

No description available

	Name	Type	Description
Inputs	none		
Outputs	FUNCTION	Vector<ObjectRef>	The FuzzyFunction object
Parameters	A	float	A value
	B	float	B value
	C	float	C value
	NAME	string	The name of the function

UnPack (Flow)

Unpack data already packed. This node makes just the opposite of "Pack" and is often used with it.

	Name	Type	Description
Inputs	INPUT	any	The packed vector
Outputs	OUTPUT	any	The single unpacked Object
	NOT_END	any	True if there's still data
Parameters	none		

UpSample (DSP:Base)
Upsamples a signal by inserting zeros at regular interval

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input frames
Outputs	OUTPUT	Vector<float>	Upsampled output
Parameters	FACTOR	int	Upsampling factor

VMethod (General)

Applies a certain method on an object int or float. The name of the method to call can be: log, exp, sin or cos.

	Name	Type	Description
Inputs	INPUT	int or float	Object on wich the method will be applied
Outputs	OUTPUT	int or float	Return value of the method
Parameters	METHOD	string	The name of the method to call

VQClass (VQ) (require: VQ) No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
	VQ	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	none		

VQCloseness (VQ)
No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
	VQ	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	none		

VQTrain (VQ) (require: VQ) No description available

	Name	Type	Description
Inputs	FRAMES	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	MEANS	any	No description available
	BINARY	any	No description available

VQWeightMeans (VQ)
No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
	VQ	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	OUTPUTLENGTH	any	No description available

VQuantize (VQ) (require: VQ) No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
	VQ	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	none		

VQuantizeDiff (VQ)

No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
	VQ	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	LENGTH	any	No description available

VarLoad (General)

Load a variable. The variable is pulled from a node of type: " VarStore "(General) which has the same given name. The node: " VarStore " can be declared in a different Overflow file, but must be declared before.

	Name	Type	Description
Inputs	none		
Outputs	OUTPUT	any	The variable value
Parameters	VARIABLE	string	The name of the variable t be loaded

VarStore (General)

Store a variable under a specified name. The variable may be used in other Overflow files by the node: " VarLoad "(General).

	Name	Type	Description
Inputs	INPUT	any	The value of the variable
Outputs	OUTPUT	any	The value of the variable
Parameters	VARIABLE	string	The variable name

VectorCode (Vector)

Modifies a vector using C++ code

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input vector
Outputs	OUTPUT	Vector<float>	Output vector (after gain)
Parameters	CODE	string	C++ code inside function [void func(const float *x, float *y, int length)]

Window (DSP:Base)

Applies a window on a frame

	Name	Type	Description
Inputs	INPUT	Vector<float>	Input frame
Outputs	OUTPUT	Vector<float>	Windowed frame
Parameters	LENGTH	int	Length of the frames/window
	WINDOW	string	Window type (HANNING, HAMMING, HALF_HANNING)
	SYMETRIC	bool	Symetric window, uses (length-1) for normalization

WriteAudio (DSP:Audio)
Writes audio frames to the sound card (or any other) stream

	Name	Type	Description
Inputs	OBJECT	Vector<float>	Audio frames
	DEVICE	Stream	(Sound card) stream
Outputs	OUTPUT	Vector<float>	Returning the input audio frames
Parameters	LEAD_IN	int	Number of zero frames to send before starting (for synchronization)

ZCrossing (DSP:Misc)
Number of zero-crossing in a vector: count(v[i]*v[i+1]<0)

	Name	Type	Description
Inputs	INPUT	Vector<float>	The input vector
Outputs	OUTPUT	float	Number of zero-crossing
Parameters	none		

dB (DSP:Base) No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	none		

undB (DSP:Base) No description available

	Name	Type	Description
Inputs	INPUT	any	No description available
Outputs	OUTPUT	any	No description available
Parameters	none		