Cogito Instruments CI 9120 manual

1024 Neurons

Address:

Cogito Instruments SA

7, rue de l'Arquebuse

1204 Geneva

Switzerland

contact@cogitoinstruments.com

For support please contact the support email address:

contact@cogitoinstruments.com

Table of Contents

1. CI 9120 overview

Cogito Instruments CI 9120 is a cRIO module that contains an artificial neural network (CogniMem Technologies CM1K). This neural network (NN) of 1024 neurons can compute the distance between any pattern of 256 bytes and its internal knowledge and return a classification in a very short time thanks to its inherent massive parallelism.

Neural network operating principle

The NN is composed of 1024 neurons, each having an internal memory of 256 bytes and 4 internal registers: **Category** (CAT), **Context**, **Active Influence Field** (AIF) and **Minimum Influence Field** (MINIF). The neurons can be in three states: "Idle", "Ready-to-learn" or "Committed". At power-up all neurons are "Idle" except the first neuron which is "Ready-to-learn". Once the first neuron has learned a vector, it becomes "Committed" and the next neuron becomes Ready-to-learn.

When one broadcasts a vector (256 bytes) to the NN, each committed neuron compares the vector to its internal memory and computes the euclidian distance between them. If that distance is smaller than its AIF, it "fires". Three outcomes are possible:

- No neuron fires, the vector has not been recognized.
- Several neurons with different CAT fire at the same time, the vector is recognized but the category is *Uncertain* (UNC).
- All firing neurons have the same CAT, then the vector is *Identified* (ID).

One can then request the network to return the **Distance** (DIST) and **Category** (CAT) of the firing neurons. The NN returns the DIST and CAT in increasing order of DIST (thus, the most probable results comes first).

At power-up, the NN's internal knowledge is blank. The NN can acquire knowledge through **training** or by directly downloading knowledge (from a previous training) to the network.

Training the network consists in broadcasting a vector and subsequently performing a Learn operation, specifying the CAT of the vector. At this point the NN knows if the vector was recognized or not. Two situations can arise:

- If the NN has identified (ID = True) the vector and the new CAT one is sending through the Learn operation is the same as the CAT of the firing neurons, nothing happens
- If the vector has been correctly classified by some but not all of the firing neurons - meaning some firing neurons have a CAT different from the CAT one is trying to Learn - those neurons will reduce their AIF so that such erroneous classification is avoided in the future.
- If no neuron of the right CAT has fired, a new neuron will be committed. Its internal memory will be loaded with the broadcasted vector and its AIF will be set to the smallest DIST of all the other committed neurons to that vector. All erroneous neurons will decrease their AIF.

Neural network settings

The NN has a certain number of settings influencing its learning and classification behavior:

- **MAXIF** (*Maximum Influence Field)* sets the upper bound of AIF. When a new neuron gets committed its AIF cannot exceed MAXIF. Subsequent learning can only decrease this AIF. Default: 16384. Reading this register when the network is full returns the value 65535.
- **MINIF** sets the lower bound of AIF. If a neuron's AIF has reached MINIF, it means that two vectors which differ by less than MINIF and have different CAT exist in the NN. When a neuron with AIF=MINIF fires, a boolean flag reports the fact that the neuron is *degenerated*. Default: 2. Reading this register when the network is full returns the value 65535.
- Using the **Context** register one can divide the NN into several subsets working independently. The **Context** is composed of two things: a **Norm** and a C**ontext** #. The **Norm** sets the type of norm used for the DIST calculations. Two norms are available: L1 (Manhattan distance) and Lsup (maximum distance amongst all dimensions). For each **Norm**, the user can define up to 127 different contexts using **Context #**. Each combination of **Norm** and C**ontext** # is unique and constitutes a given **Context**, hence the pair is grouped in a cluster. When a new neuron is committed, it stores internally the value of the **Context** register at the time of committing. When a new vector is broadcast, only the neurons pertaining to the current C**ontext** respond. If **Context#** is set to 0, all neurons will respond (**Context#** is ignored). If a neuron has been committed with **Context#** = 0, it will only respond when **Context#** = 0. Reading this register when the network is full returns **Norm** = Lsup and **Context#** = 127, which are irrelevant values, but in this case the register can still be written to change the context for recognition.
- **Classifier** sets the type of response returned by the NN. The two options are *Radial Basis Function* (RBF) or *K-Nearest Neighbours* (KNN). In RBF mode, only the neurons with DIST < AIF fire, whereas in KNN mode all the committed neurons in the network return their DIST and CAT.

2. CI 9120 Specifications

Timing characteristics

Minimum time to stream a vector of 256 elements at 70°C…………………………………………..56 us

Minimum time per result…………………………………………………………………………….8 us

Minimum time to save the knowledge of 1024 neurons……………………………………………….220 ms

Minimum time to restore the knowledge of 1024 neurons……………………………………………...50 ms

Temperature range

Temperature range (ambient)…………………………………………………………………..-40°C to +70°C

Power requirements

Power consumption from chassis……………………………………………………………………1 W maximum

Thermal dissipation (at 70°C)…………………………………………………………………...1 W maximum

Physical characteristics

Weight…….131 g

3. Installation

There is a device driver available for the CI 9120 module. The driver can be downloaded and installed using the JKI VI Package Manager.

Software Requirements

Programming an application with the CI 9120 module requires:

- LabVIEW 2014 or higher
- LabVIEW FPGA 14.0 or higher
- NI-RIO 14.0 or higher
- JKI VI Package Manager 14.0 or higher

Hardware Compatibility

The CI 9120 has been tested on the following targets:

cRIO-9030, cRIO-9031, cRIO-9033, cRIO-9035, cRIO-9036, cRIO-9037, cRIO-9038, cRIO-9039, cRIO-9063, cRIO-9064, cRIO-9065, cRIO-9066, cRIO-9067, cRIO-9068, cRIO-9073, cRIO-9074, cRIO-9075, cRIO-9076, cRIO-9081, cRIO-9082.

4. Getting started

This section describes how to modify the “Waveform Recognition Example” example project provided with the drivers in order to run it on a different CompactRIO chassis. Follow the steps:

1. Insert the CI 9120 in the cRIO chassis in the slot of your choice (here we will use slot 1)
2. Power-up your cRIO and connect it to your local network.

3. Install the CI 9120 drivers using the VI Package Manager (VIPM).
4. Start LabVIEW and open the example finder. Menu Help → Find Examples... Then search for “Cogito” and you should find it.
5. In the Project Explorer window, right-click Project → New → Targets and Devices…(see Fig. 1)

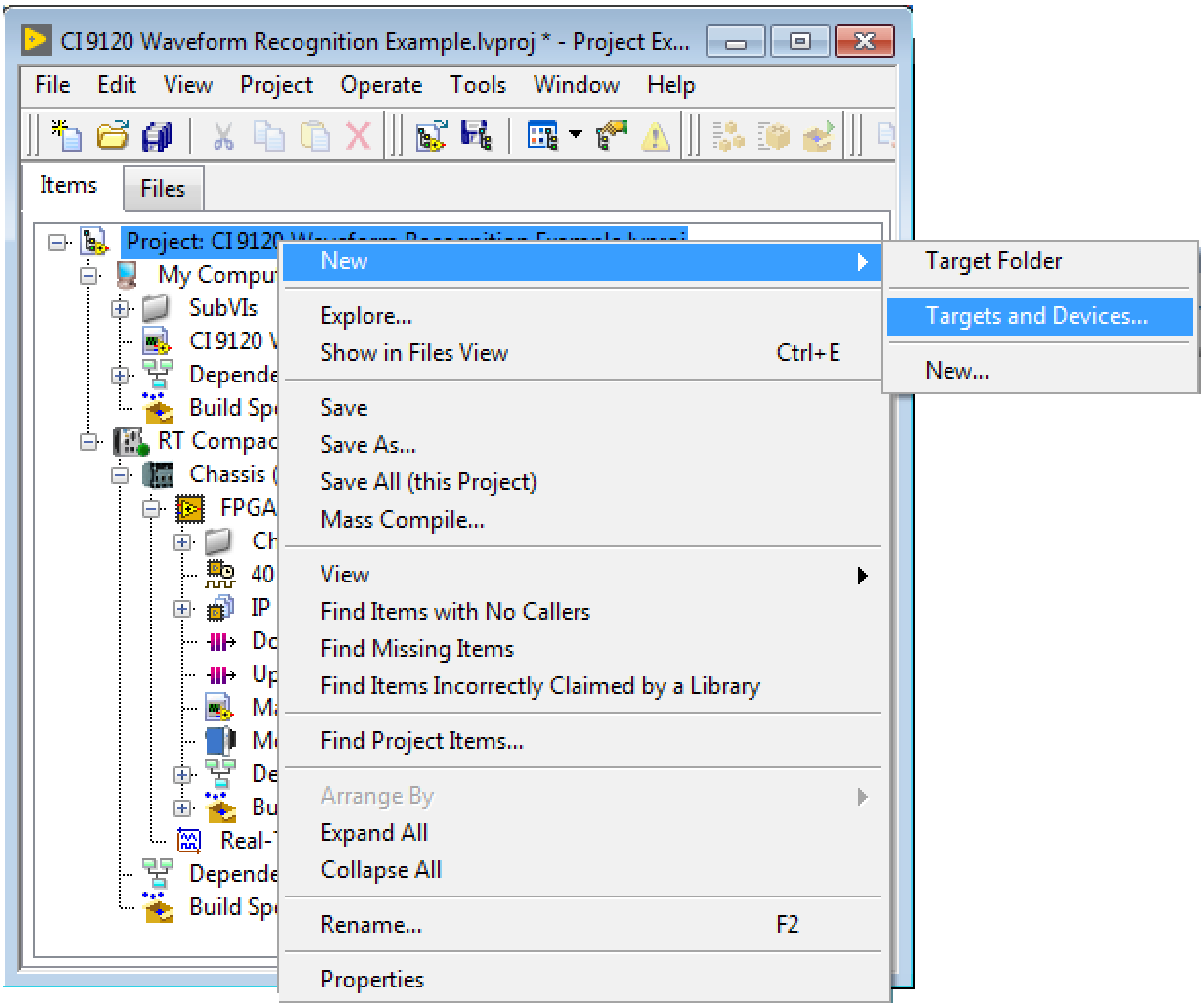

Fig. 1: Adding a new Target to the project.

6. The “Add Targets and Devices” wizard opens. Select “Discover an existing target or device” under “Existing target or device”. In the listbox, expand the section “Real-time CompactRIO”. After a few seconds, your target should show up. Select it and click OK.
7. The wizard will then prompt you to choose between “Scan Interface” and “LabVIEW FPGA Interface”. Choose “LabVIEW FPGA Interface” and click OK.
8. The wizard will now “discover” the chassis, i.e. it will scan the chassis to detect if there are any known modules inserted. While discovering the chassis, a prompt might come up saying that the FPGA target has been programmed and may be in use, asking you if you want to halt its execution to discover the chassis. Click “Discover”. If you do not want to stop the FPGA now, you can always add your module to the project manually later.

9. Your chassis now appears in the project. If you expand the tree under <Your cRIO> → Chassis → FPGA Target, you should see the module CI 9120. If the module was not in the chassis at the time you underwent step 8, or if you chose not to discover the chassis, you can manually add the module by right-clicking FPGA Target → New → C Series Modules…

10. Drag and drop the items "Main_FPGA.vi", DownFIFO and UpFIFO from the original FPGA Target to the newly added one. You can now delete the original FPGA Target (right-click on the Target → Remove from Project)

11. Create a build specification and compile the Main_FPGA.vi on your target

12. Go to the block diagram of the PC vi ("CI 9120 Waveform Recognition Example PC.vi") and right-click the Open FPGA VI Reference SubVI → Configure Open FPGA VI Reference… (See Fig.2)

13. Select "VI" and browse to point to the Main_FPGA.vi on your target. Click OK.

14. Once the FPGA is done compiling, you are ready to run the example.

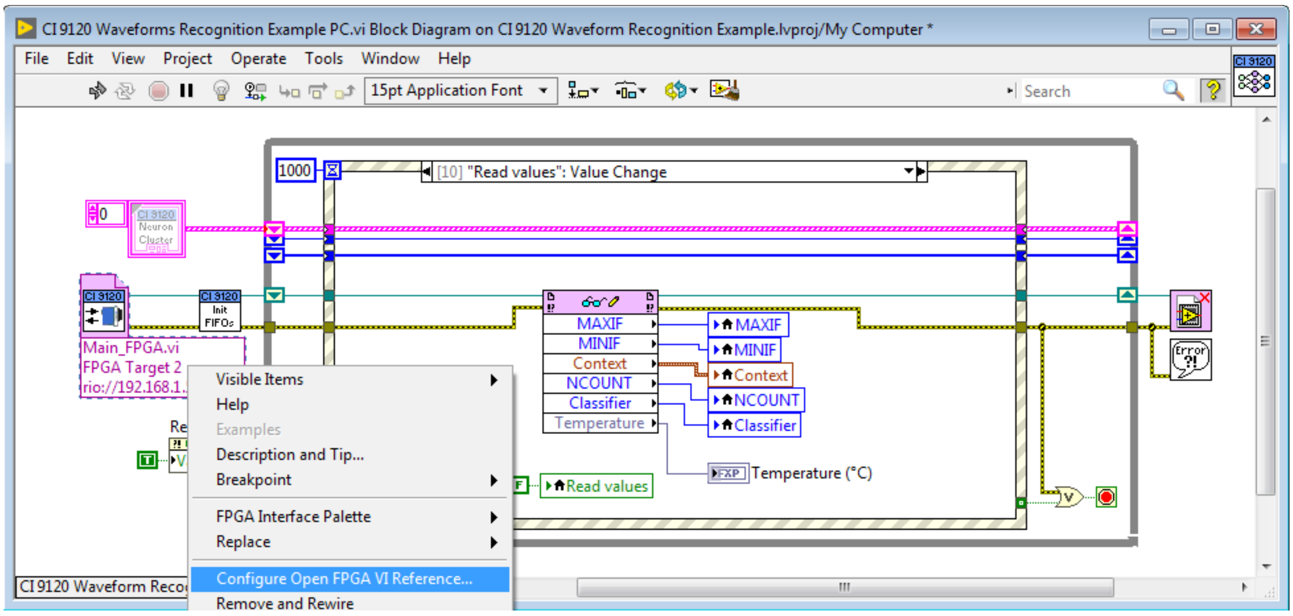

Fig. 2: Configuring the "Open VI Reference..." on the PC side.

5. LabVIEW FPGA API

Once the module is present in a LabVIEW FPGA project, the neural network can be addressed directly using the API described in this section.

As this module is not a standard I/O module, no I/O nodes are available for this module. Instead, the neural network is controlled through a set of method and property nodes.

Module communication modes

In LabVIEW FPGA, the vectors can only be broadcast one element at a time. To allow for fast data transfer and avoid mixing up the data, the CI 9120 module has four communication modes: Normal (Default), Broadcast, Save and Restore mode (see Fig. 3). The different methods/properties are only accessible in a given mode and some methods induce mode transitions. Fig.3 summarizes the scope of the different methods and properties.

If the module is not in the right mode when a property node is called, it will block the execution until the module is in Normal mode.

Methods have three types of behavior: the “start” methods have a timeout, the “data transfer” methods (*Save Element*, *Restore Element*, etc..) return a “Mode error” boolean and the other methods block the execution like the property nodes.

This architecture ensures a robust communication between the FPGA and the module.

The special property node *Module Status* is available all the time. It returns the current communication mode, allowing the user to debug his application.

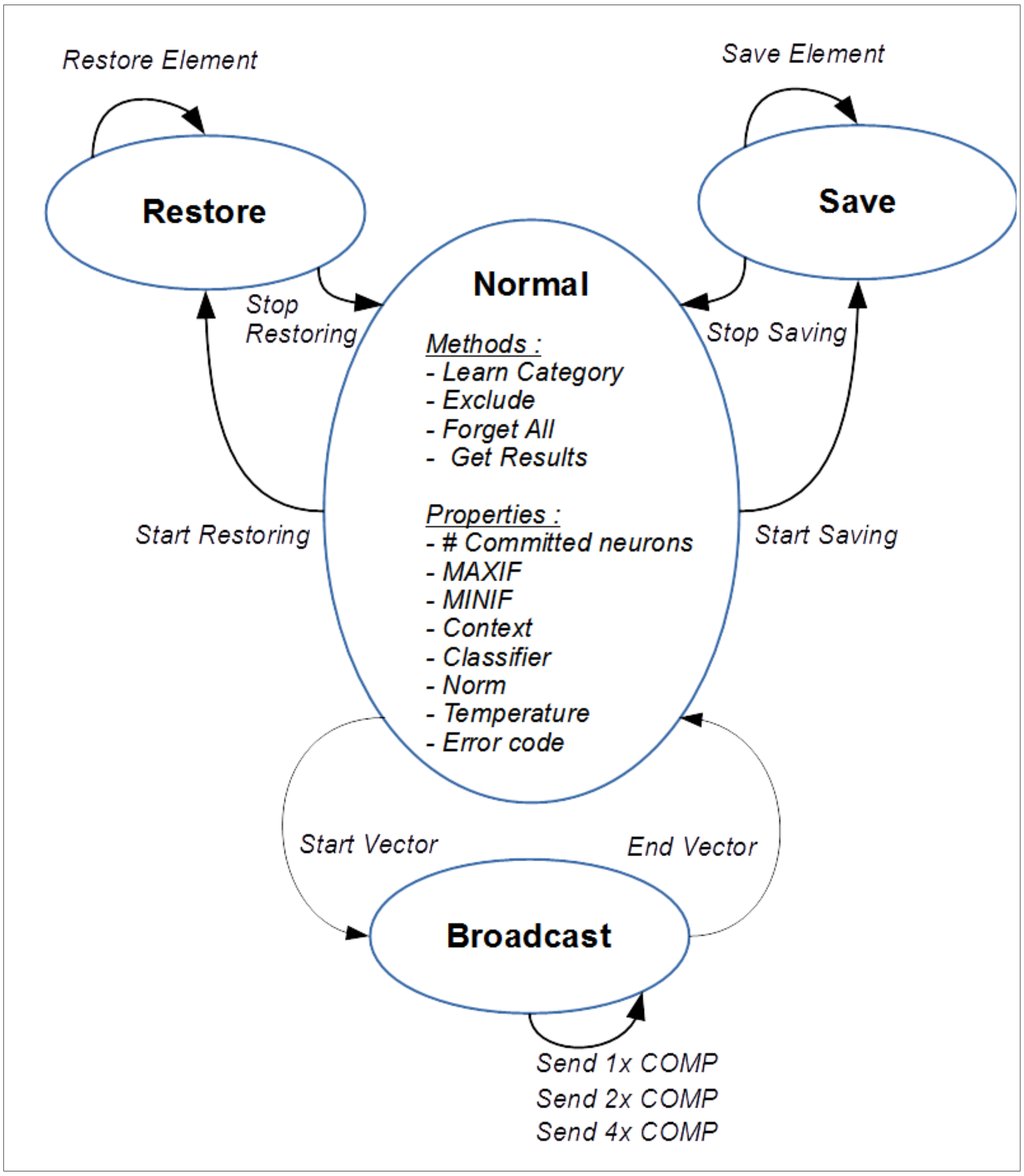

Fig. 3: CI 9120 communication modes and transitions

Broadcasting a vector

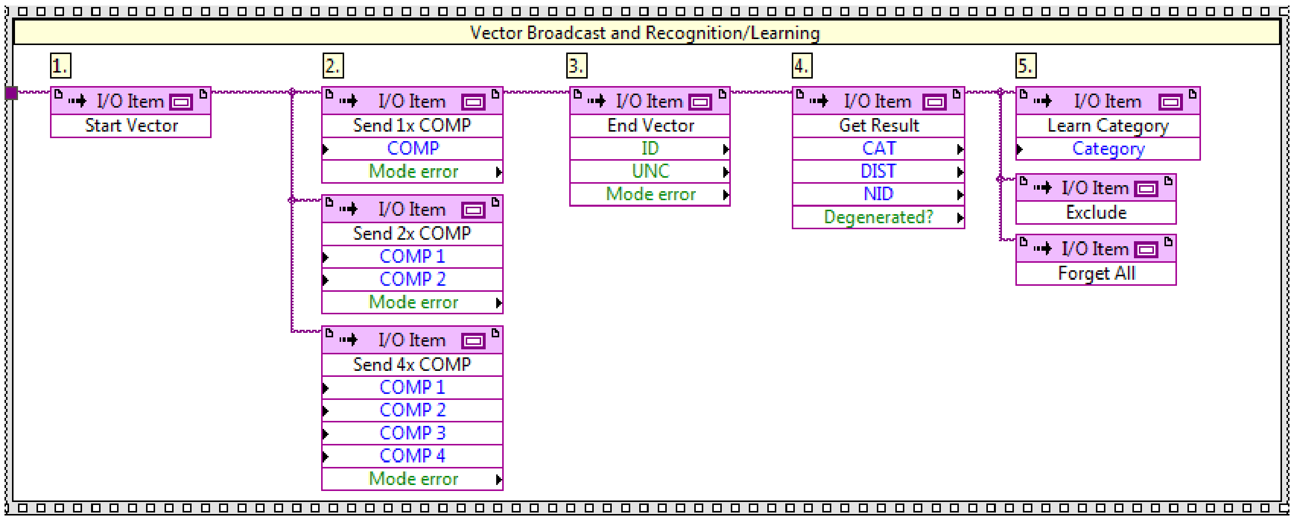

Fig. 4: Vector broadcast and classification/learning methods

To broadcast a vector to the NN, the procedure is as follows (see Fig.4):

1. Run the *Start Vector* method, then

2. Run one of three methods: *Send 1x COMP*, *Send 2x COMP* or *Send 4x COMP* multiple times until all (max. 256) components have been broadcast. These methods return a "Mode error" flag if the module was not in Broadcast mode when they were called (meaning probably that the *Start Vector* method was not called in due time).

3. Run the *End Vector* method. This method returns whether the vector was *Identified* (ID), recognized with *Uncertainty* (UNC) or not recognized at all. Based on this information one can choose what to do next: get more detailed results, learn or do nothing (start next vector). This method puts the module back into Normal mode. This method returns a "Mode error" flag if the module was not in Broadcast mode when it was called (meaning probably that the *Start Vector* method was not called in due time).

4. More detailed results can be obtained by running the *Get Results* method multiple times. Each call to this method returns the DIST and CAT of the next firing neuron by increasing order of DIST. It also returns the *Degenerated?* flag which indicates that the neuron is degenerated. When all firing neurons have returned their result, the method returns DIST = 65535 and CAT = 32768.

5. A category can be assigned to the broadcast vector by running the *Learn Category* method. This method expects a category number in the form of an U16. If Category = 0 is sent to this method, it will preform an exclude (like the *Exclude* method). The *Exclude* method tells the NN that the vector was not part of any of the categories it has learned so far. All the neurons that have fired will therefore decrease their AIF.

Finally, the *Forget All* method can be run anytime (when in normal communication mode). It will simply un-commit all the neurons in the NN. The knowledge will be blank again.

<u>Note about vector broadcast methods:</u> there are three different methods to broadcast vector elements (*Send 1x COMP*, *Send 2x COMP* and *Send 4x COMP)*. The reason for this is to allow for both speed and convenience, depending on the user's requirements. To broadcast at full speed, you must use the *Send 4x COMP* method. However, if speed is not a necessity, the user is free to use the *Send 1x COMP* or *Send 2x COMP* methods.

Saving and restoring the knowledge

When the NN has been trained for a specific application, it can be useful to save the knowledge in order for the NN to be immediately operational at the next power-up (after restoring this knowledge). It can also be useful to review the content of the neurons in order to understand better the behavior of the NN, or even manually edit that content.

To save the knowledge, follow these steps (see Fig.5):

1. Run the *Start Saving* method, then
2. Run the *Save Element* method multiple times. In order to retrieve the entire knowledge, one must call this method 68 times the number of committed neurons in the network (e.g. if 1024 neurons are committed, the method should be called 69632 times). This method returns a "Mode error" flag if the module was not in Save mode when it was called (meaning probably that the *Start Saving* method was not called in due time).
3. Run the *Stop Saving* method when done. This puts the module back into Normal mode.

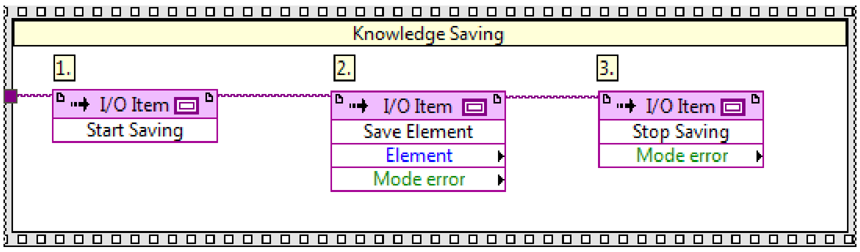

Fig. 5: Knowledge saving methods

To restore the knowledge, follow these steps (see Fig.6):

1. Run the *Start Restoring* method. If it didn't time out (it means the module is now in Restore mode), then

2. Run the *Restore Element* method multiple times. Each neuron contains 68 elements, so one must call this method a multiple of 68 times.

3. Run the *Stop Restoring* method when done. This puts the module back into Normal mode.

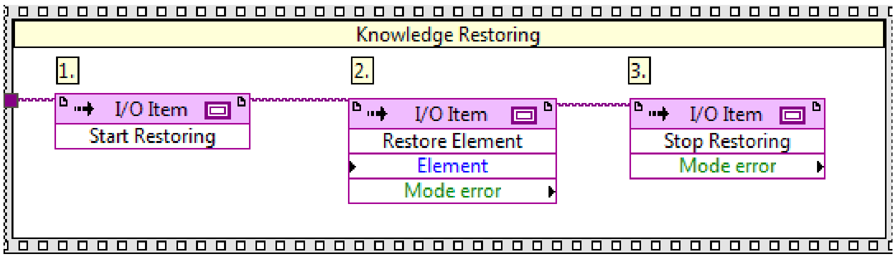

Fig. 6: Knowledge restoring methods

Knowledge data format: the knowledge elements returned by the *Save Element* method can be saved as is and fed back into the *Restore Element* method in the same order to properly restore the knowledge. However, if one wants to display or even edit that knowledge, one needs to know the data format of the knowledge. The first 64 U32 elements are the 256 U8 vector components grouped 4 by 4, followed by Context, MINIF, AIF and CAT.

Property nodes

The property nodes can be devided into three categories (see Fig.7): “Module Identification”, “Neural Network Settings” and “Module Status”.

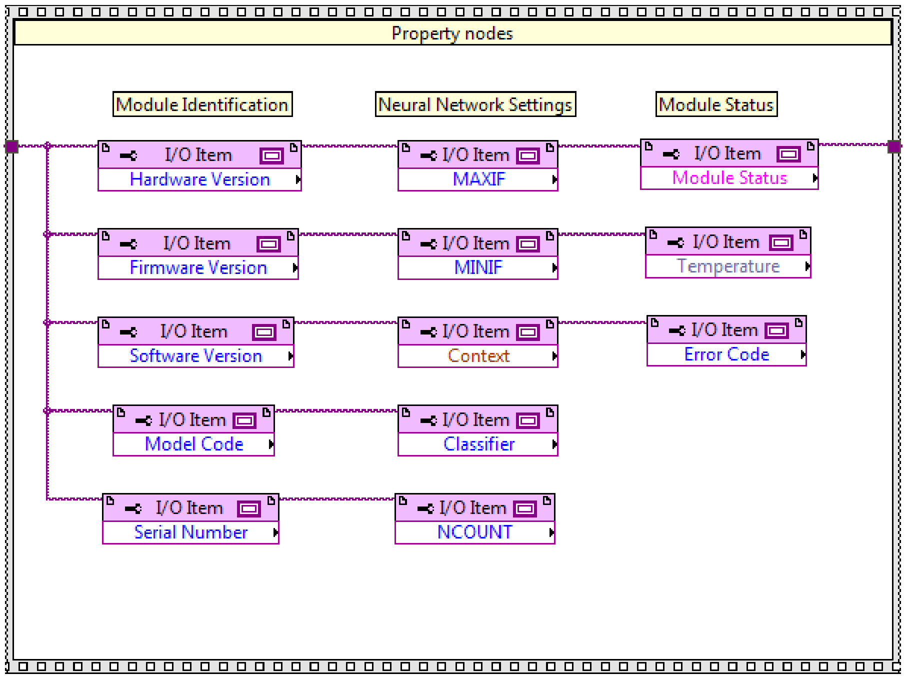

Fig. 7: Property nodes

"Module Identification" properties (Fig.7) are read-only and provide access to the hardware, firmware and software versions, module model code and serial number. These nodes are provided for support purposes.

The NN settings properties described in Section Neural network settings can be accessed (Read/Write) using the property nodes visible under "Neural Network Settings" in Fig.7. In addition to these, the (read-only) property *NCOUNT* returns the number of neurons currently committed in the network. It is useful to interrogate this property after a call to the *Learn Category* method to know if a new neuron was committed, or before calling the *Start Saving* method, in order to save only the knowledge of the committed neurons (the remaining knowledge is irrelevant). Note that if all neurons are committed, the MAXIF, MINIF, Context and Norm will return 65535, 65535, 127 and Lsup respectively when read.

The remaining property nodes belong to the "Module Status" category:

- *Module Status* (read-only) returns a cluster of three elements: Module Status, Module Communication Mode and an Error boolean. The module communication mode enum returns the current communication mode (Normal, Save, Restore or Broadcast, see Section Module communication modes). The error boolean is a flag signalling that the module has output an error. This error can be read and acknowledged by calling the *Error Code* property when the module is in normal mode (see Error codes in section below). The module status enum reports on the current status of the module. If it is anything else than "Normal", none of the property and method nodes can execute. This would typically occur if the module is withdrawn from the chassis or the wrong module was inserted.
- *Error code* (read-only) returns the error code and clears the error flag.
- *Temperature* (read-only) returns the temperature inside the module in °C. The module is not guaranteed to work according to specifications if the temperature exceeds 85°C.

6. Error codes

The following table lists the different error codes that can be returned by the *Error code* property node and their description.

Error code	Description	Possible reason	What should I do ?
65	Internal communication error.	Module temperature too high.	Let the module cool down and re-try. Contact support.
66	Overcurrent (> 200mA) detected	Module temperature too high. Short circuit inside the module.	Let the module cool down and re-try. Contact support.
68	Module temperature exceeds 85°C	Ambient temperature > 70°C	Let the module cool down. Decrease speed of operation.