free Studio

Connection
User Manual

Revision 1.2 - July 2011

Connection User Manual
Revision 1.2 - 2011-07-12
Published by Eliwell Controls S.r.l.
Via dell’Industria, 15 Z.I. Paludi
32010 Pieve d’Alpago (BL)

Contents

1. BASIC CONCEPTS

1.1 ENTRY POINT AND CONTAINER

FREE Studio Connection is an important piece in the FREE Studio software suite.

It is designed to be the "entry point" to create and manage complex projects, made of different devices and sub-projects; its main purpose is to keep all the pieces together and to simplify the task of linking the various elements (software components or physical devices).

For example, with Connection you can create a project (that can be seen as a big "workspace") that consists in two or more devices physically linked together on the same network, that have both a PLC and HMI project, that act as a master and exchange data with remote slaves, and moreover exchange data between each other, in a peer-to-peer relationship.

At the end of the developing process, Connection will create a single file containing ALL the PLC programs, HMI pages, parameters and settings of ALL the devices; then, using FREE Studio Device, you can distribute and deploy your complex application in the product environment with a single click.

Connection can also be seen as the starting point from which all the other tools of the Suite can be launched, opening their respective documents and projects: Application, UserInterface, and Device.

1.2 COMPOSITE APPLICATIONS AND FIELD I/O

Rich and advanced devices (such as FREE Evolution) can both run a PLC program and show HMI pages on the same hardware.

With Connection you can create the two separate sub-projects by launching the corresponding program (FREE Studio Application for PLC and FREE Studio UserInterface for HMI) and keep them together in a single folder on disk.

If the device can act as a master (that is the case of FREE Evolution), it can exchange data on a local bus talking with one or more slaves, with a protocol like Modbus or CANopen; with Connection you can describe those master/slave networks, by inserting the slaves into the project and connecting their remote objects to local PLC variables, making the PLC program aware of them.

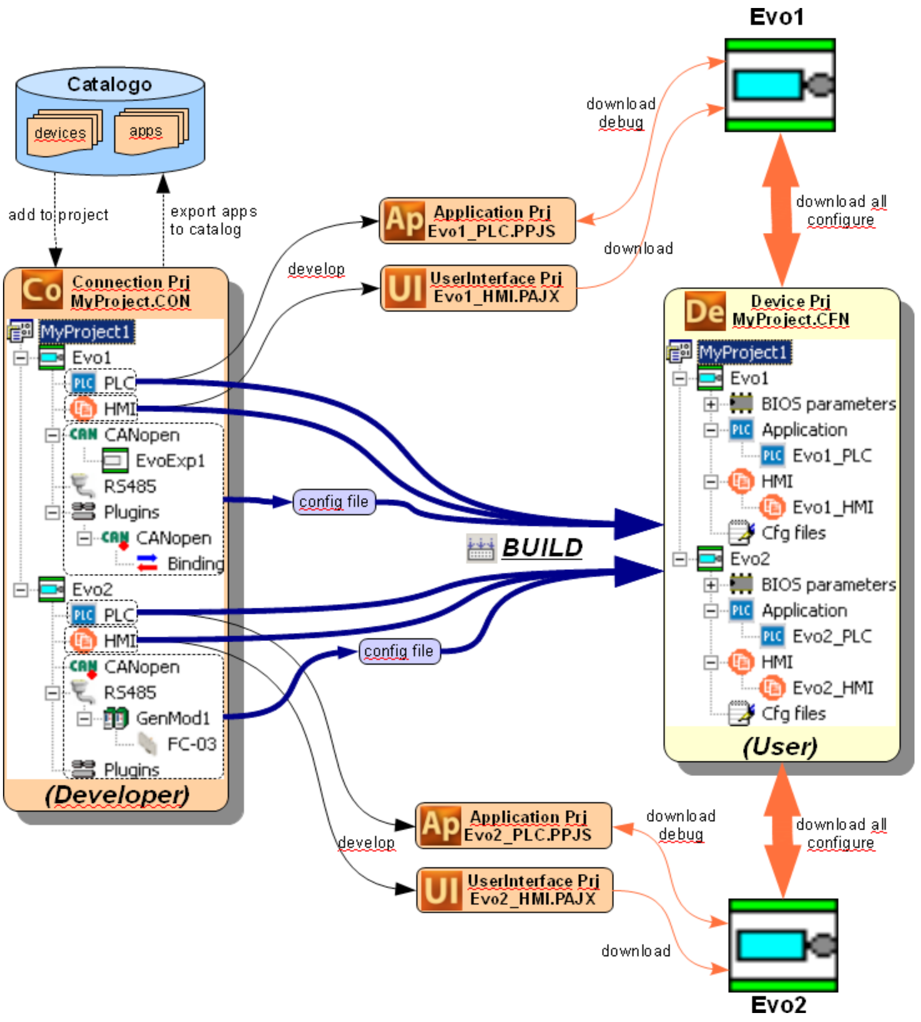
We call this architecture "Field I/O".

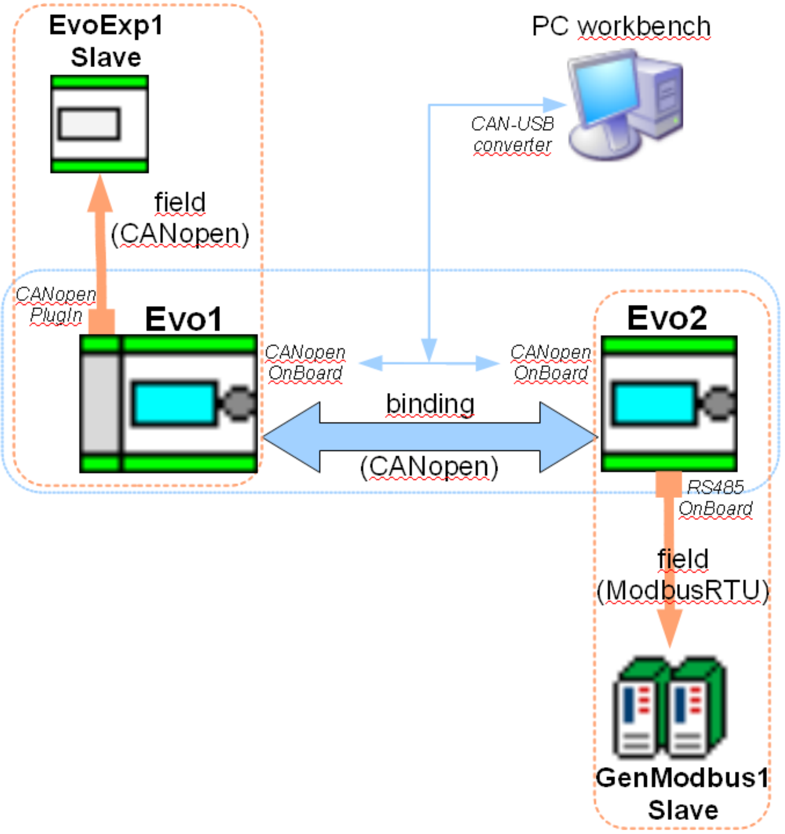
1.3 DISTRIBUTED APPLICATIONS AND BINDING I/O

Sometimes a single application on a single device is not enough to solve complex problems; sometimes it is necessary to create two or more applications that will act together, communicating and exchanging data on a network to take decisions and cooperate.

This scenario is different from "Field I/O" because there is no master or slave, but a group of devices of the same kind (like a group of FREE Evolution) that talk to each other in a peer-to-peer way on a common network, exchanging objects (parameters and values.)

We call this architecture "Binding I/O", because the various elements are bound to each other to operate together.

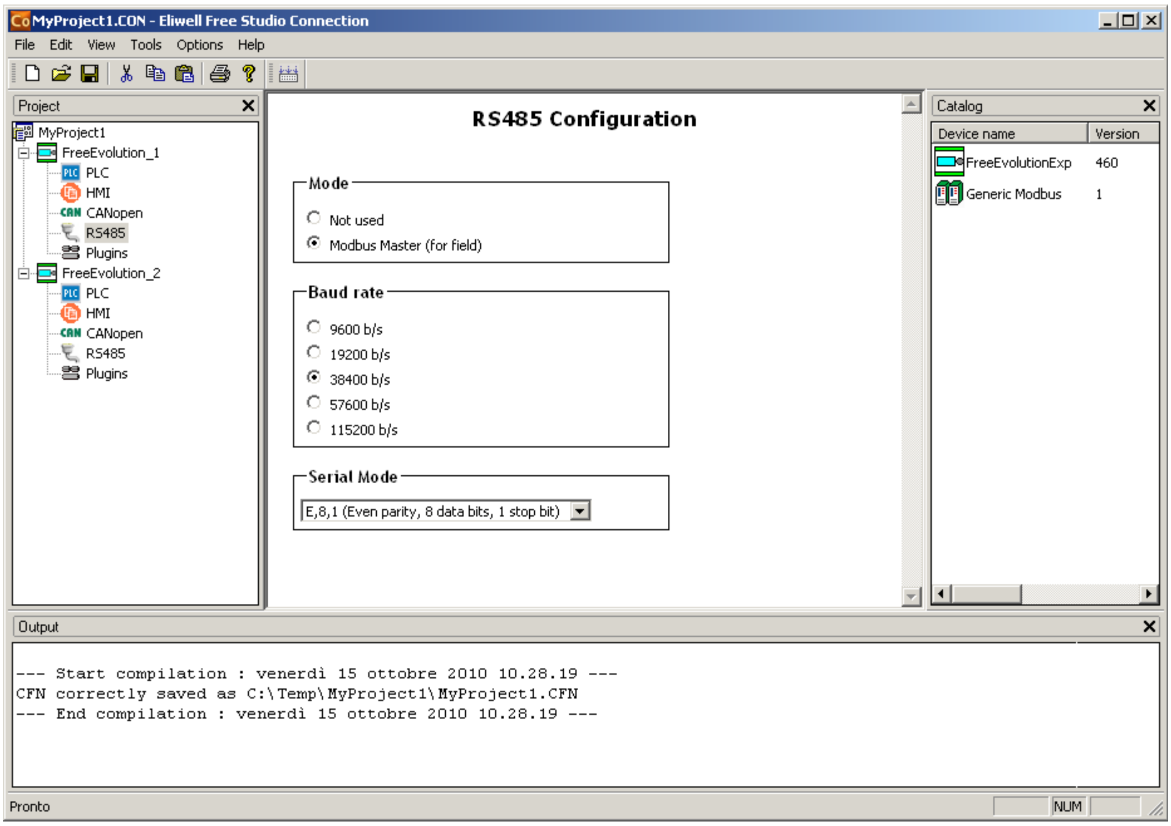

Catalogo
devices
apps
add to project
export apps
to catalog
Co Connection Prj
MyProject.CON
MyProject1
Evo1
PLC
HMI
CANopen
EvoExp1
RS485
Plugins
CANopen
Binding
Evo2
PLC
HMI
CANopen
RS485
GenMod1
FC-03
Plugins
(Developer)
develop
Ap Application Prj
Evo1_PLC.PPJS
UI UserInterface Prj
Evo1_HMI.PAJX
config file
BUILD
config file
develop
Ap Application Prj
Evo2_PLC.PPJS
UI UserInterface Prj
Evo2_HMI.PAJX
Evo1
download
debug
download
download all
configure
De Device Prj
MyProject.CFN
MyProject1
Evo1
BIOS parameters
Application
Evo1_PLC
HMI
Evo1_HMI
Cfg files
Evo2
BIOS parameters
Application
Evo2_PLC
HMI
Evo2_HMI
Cfg files
(User)
download all
configure
download
debug
download
Evo2


EvoExp1
Slave
PC workbench
CAN-USB
converter
field
(CANopen)
CANopen
PlugIn
Evo1
CANopen
OnBoard
CANopen
OnBoard
Evo2
binding
(CANopen)
RS485
OnBoard
field
(ModbusRTU)
GenModbus1
Slave

2. USING THE ENVIRONMENT

2.1 THE WORKSPACE

The figure below shows a view of Connection workspace, including many of its more commonly used components.



RS485 Configuration

Mode
○ Not used
● Modbus Master (for field)

Baud rate
○ 9600 b/s
○ 19200 b/s
● 38400 b/s
○ 57600 b/s
○ 115200 b/s

Serial Mode
E,8,1 (Even parity, 8 data bits, 1 stop bit)

Device name	Version
FreeEvolutionExp	460
Generic Modbus	1

```
--- Start compilation : venerdì 15 ottobre 2010 10.28.19 ---
CFN correctly saved as C:\Temp\MyProject1\MyProject1.CFN
--- End compilation : venerdì 15 ottobre 2010 10.28.19 ---
```

The following paragraphs give an overview of these elements.

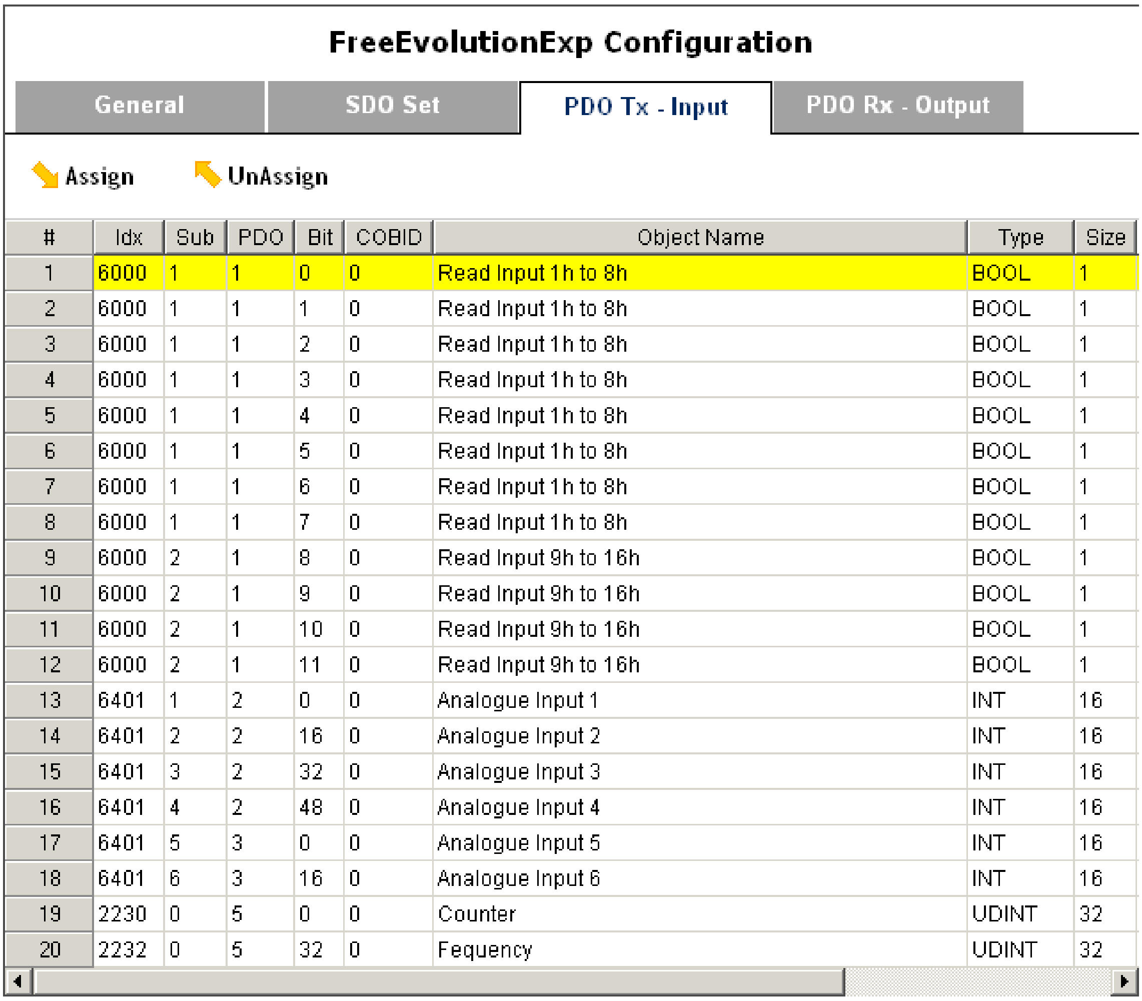
2.1.1 THE MAIN WINDOW

The *Main window* is the central part of the program window, that is surrounded by toolbars and docking windows.

It shows information and configuration pages in a graphical and user-friendly form; the current page is always determined by the selected item in *Project* window.

For example, in the previous image you can see that the *RS485* node is selected (and highlighted) in the *Project* tree and so the *Main window* shows the *RS485 Configuration* page.

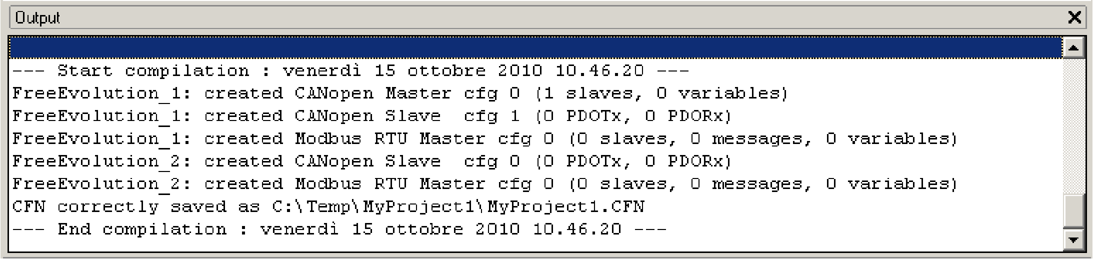
To change the current selected item and so the current page, just do a single click in the *Project* tree.



#	Idx	Sub	PDO	Bit	COBID	Object Name	Type	Size
1	6000	1	1	0	0	Read Input 1h to 8h	BOOL	1
2	6000	1	1	1	0	Read Input 1h to 8h	BOOL	1
3	6000	1	1	2	0	Read Input 1h to 8h	BOOL	1
4	6000	1	1	3	0	Read Input 1h to 8h	BOOL	1
5	6000	1	1	4	0	Read Input 1h to 8h	BOOL	1
6	6000	1	1	5	0	Read Input 1h to 8h	BOOL	1
7	6000	1	1	6	0	Read Input 1h to 8h	BOOL	1
8	6000	1	1	7	0	Read Input 1h to 8h	BOOL	1
9	6000	2	1	8	0	Read Input 9h to 16h	BOOL	1
10	6000	2	1	9	0	Read Input 9h to 16h	BOOL	1
11	6000	2	1	10	0	Read Input 9h to 16h	BOOL	1
12	6000	2	1	11	0	Read Input 9h to 16h	BOOL	1
13	6401	1	2	0	0	Analogue Input 1	INT	16
14	6401	2	2	16	0	Analogue Input 2	INT	16
15	6401	3	2	32	0	Analogue Input 3	INT	16
16	6401	4	2	48	0	Analogue Input 4	INT	16
17	6401	5	3	0	0	Analogue Input 5	INT	16
18	6401	6	3	16	0	Analogue Input 6	INT	16
19	2230	0	5	0	0	Counter	UDINT	32
20	2232	0	5	32	0	Fequency	UDINT	32

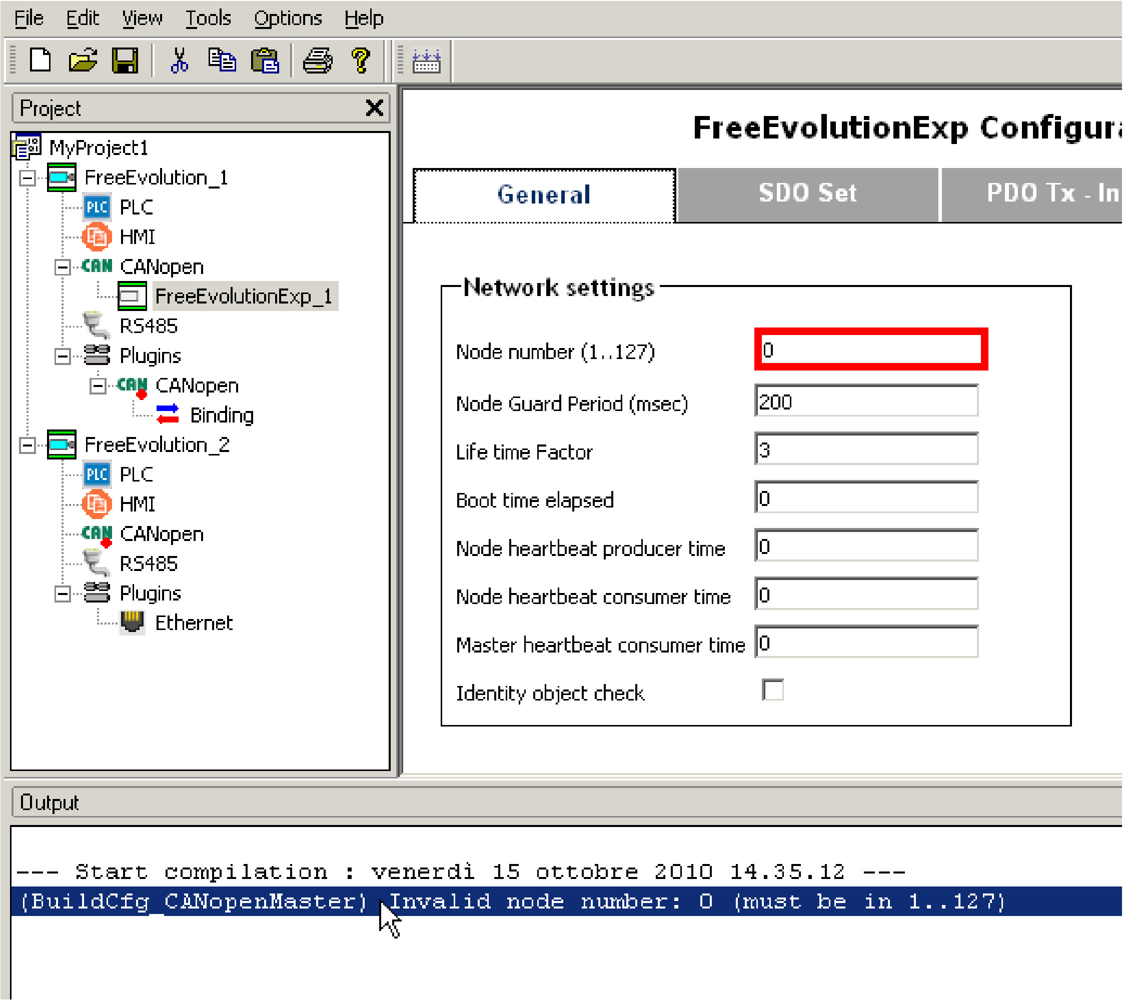
2.1.2 THE OUTPUT WINDOW

The `Output` window is the place where Connection prints its output messages: errors, informations, debug informations, and compilation results.



```
--- Start compilation : venerdì 15 ottobre 2010 10.46.20 ---
FreeEvolution_1: created CANopen Master cfg 0 (1 slaves, 0 variables)
FreeEvolution_1: created CANopen Slave  cfg 1 (0 PDOTx, 0 PDORx)
FreeEvolution_1: created Modbus RTU Master cfg 0 (0 slaves, 0 messages, 0 variables)
FreeEvolution_2: created CANopen Slave  cfg 0 (0 PDOTx, 0 PDORx)
FreeEvolution_2: created Modbus RTU Master cfg 0 (0 slaves, 0 messages, 0 variables)
CFN correctly saved as C:\Temp\MyProject1\MyProject1.CFN
--- End compilation : venerdì 15 ottobre 2010 10.46.20 ---
```

In some situations (for example compilation errors) you can double-click on the error in the output window and you will be brought just at the source of the error, that will be highlighted with a red box.



Network settings

Node number (1..127) 0

Node Guard Period (msec) 200

Life time Factor 3

Boot time elapsed 0

Node heartbeat producer time 0

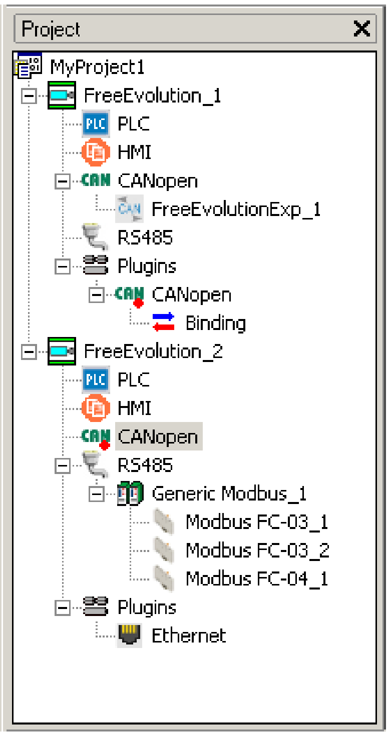
Node heartbeat consumer time 0

Master heartbeat consumer time 0

Identity object check ☐

```
--- Start compilation : venerdì 15 ottobre 2010 14.35.12 ---
(BuildCfg_CANopenMaster) Invalid node number: 0 (must be in 1..127)
```

2.1.3 THE PROJECT WINDOW



The *Project window* shows the elements of the current project in the form a tree, making easy to see the master/slave and parent/child relationship between them.

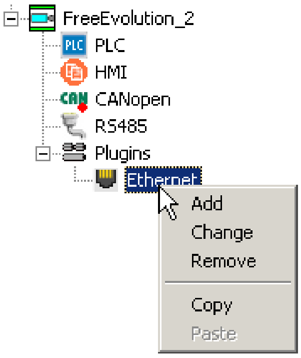
Click on the + and - icons next to each item (or press to corresponding keys) to expand or collapse each item; or press the * key to expand all children of the current item in depth.

Left-clicking an item opens its configuration page in the *Main window* (if there is one), and shows in the *Catalog window* all objects that can be inserted under the current item (if there are).

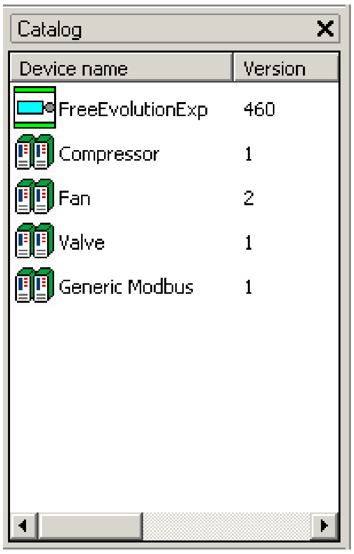
Right-clicking an item selects it and opens its context menu (if there is one), showing some operations you can do on the current tree item, like *Add/Remove/Copy/Paste* and so on.

Pressing the *Delete* key also triggers the *Remove* command.

A single left-click of the item name (or the *F2* key) triggers the in-place rename of the object (if it supports it).



2.1.4 THE CATALOG WINDOW

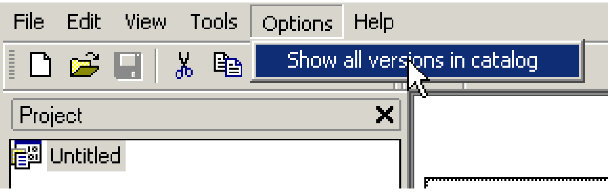


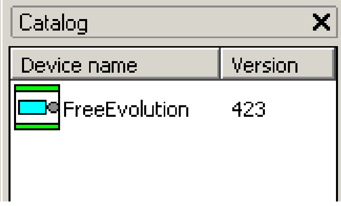
Device name	Version
FreeEvolutionExp	460
Compressor	1
Fan	2
Valve	1
Generic Modbus	1

This window shows a list of objects that can be inserted in the project under the currently selected item in the *Project window*; selecting a different item in *Project window* refreshes this list.

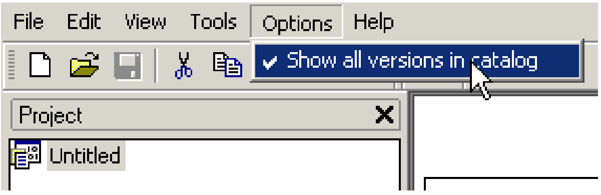
By default, only the "major" version number of each device is shown, and the highest minor version number is implicitly selected; for example, if three different versions of the same device are present in the catalog (10.0, 10.1, 10.2), the *Catalog* will show only the *10* (without the minor version) but will select the 10.2 (the highest).

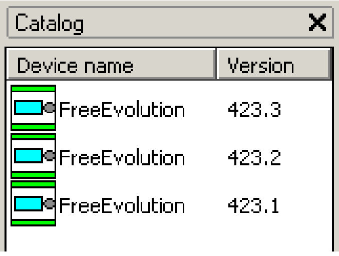
This behavior can be changed by selecting the *Show all versions in catalog* option in the *Options* menu in the menu bar; if you activate this option ALL the available versions (even the older ones) will be shown in the *Catalog* and you will have the chance to manually select and add in the project older versions of each device.





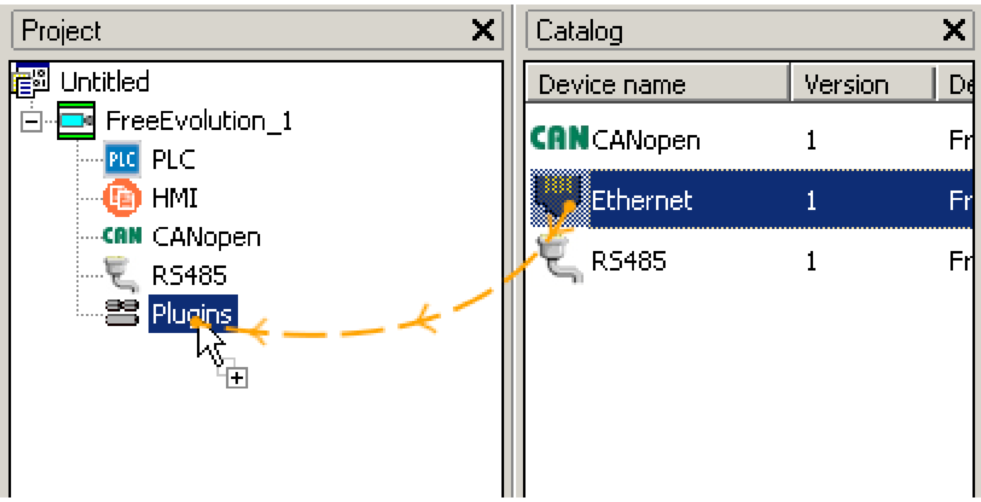
Device name	Version
FreeEvolution	423





Device name	Version
FreeEvolution	423.3
FreeEvolution	423.2
FreeEvolution	423.1

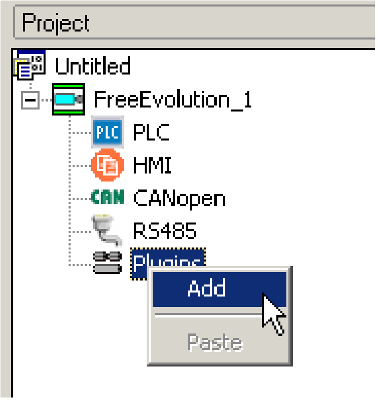
To add an object, drag and drop it from the *Catalog window* to the *Project window*, over the currently selected item (a + icon will appear); it will be added as its last child.

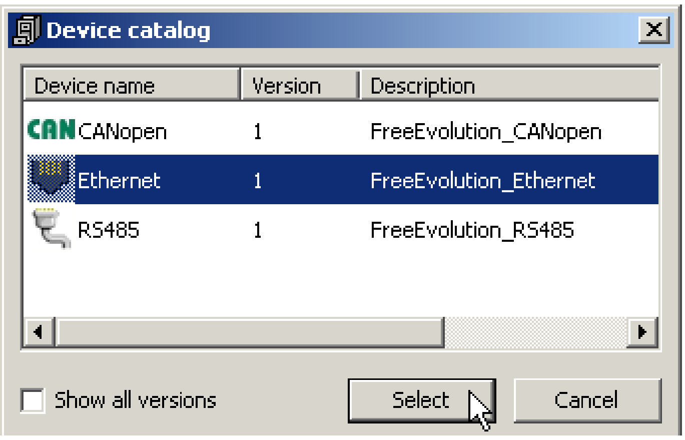


Device name	Version	De
CANopen	1	Fr
Ethernet	1	Fr
RS485	1	Fr

Another way to add an object is to right-click an item in the *Project window* and choose *Add*; a pop-up window will appear, showing the same list of the *Catalog window*. In this way you can add an object without having the *Catalog* visible, useful for example if you are working with a very small screen.

This window also has a *Show all versions* option, that behaves like the flag in the *Options* menu described before.





Device name	Version	Description
CANopen	1	FreeEvolution_CANopen
Ethernet	1	FreeEvolution_Ethernet
RS485	1	FreeEvolution_RS485

□ Show all versions

2.2 LAYOUT CUSTOMIZATION

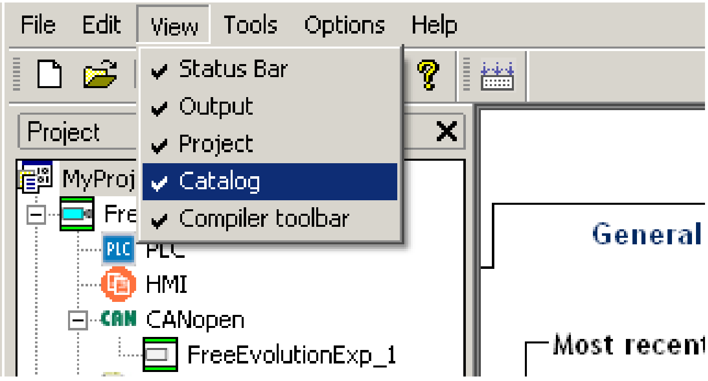
The layout of Connection workspace can be freely customized in order to suit your needs.

Connection takes care of saving the layout configuration on application exit, in order to persist your preferences between different working sessions.

2.3 TOOLBARS AND DOCKING WINDOWS

2.3.1 SHOWING/HIDING

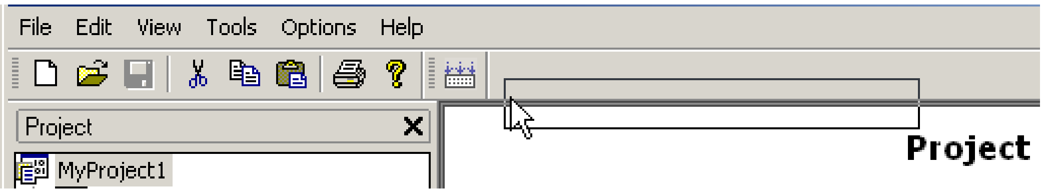
To show (or hide) a toolbar, open the `View` menu and select the desired toolbar or docking window (for example, the `Catalog` dock window).



The element is then shown or hidden.

2.3.2 MOVING TOOLBARS

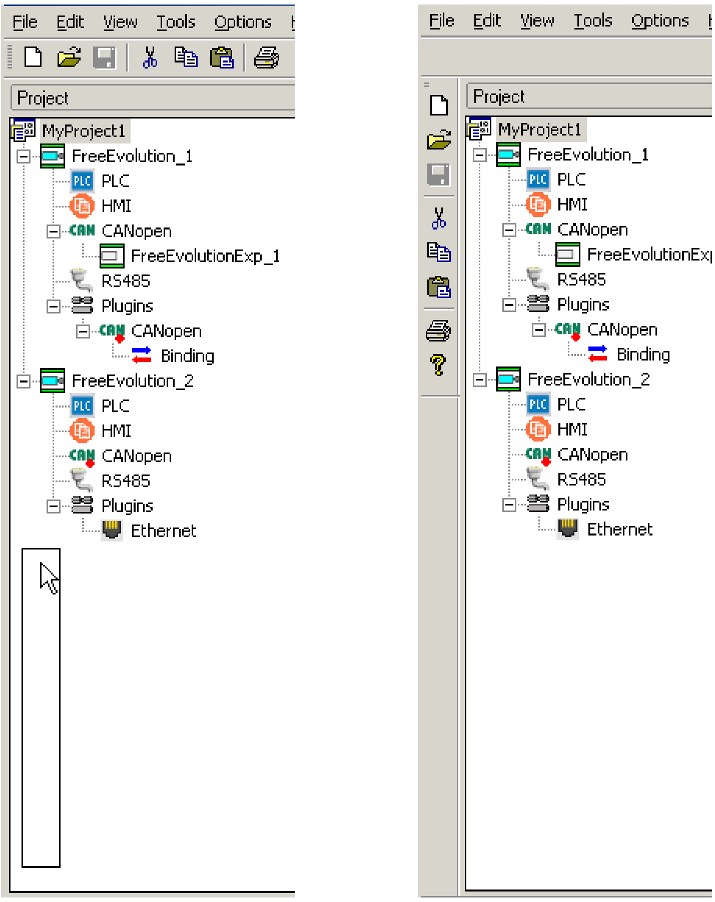
You can move a toolbar by clicking on its left border and then dragging and dropping it to the destination.



The toolbar shows up in the new position.

You can change the shape of the toolbar, from horizontal to vertical, either by pressing the

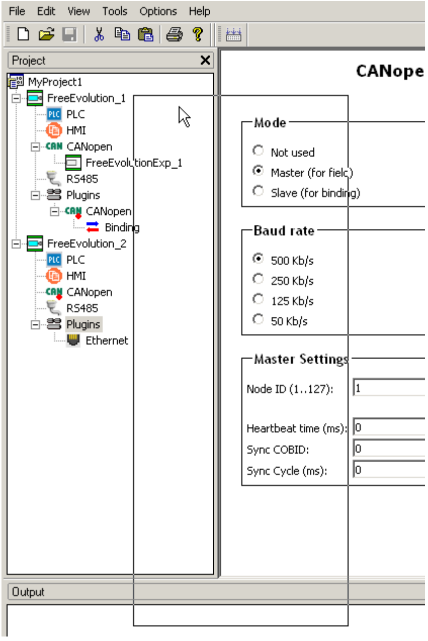
Shift key or by moving the toolbar next to the vertical border of any window.



You can also make the toolbar float, either by pressing the *CTRL* key or by moving the toolbar away from any window border.

2.3.3 MOVING DOCKING WINDOWS

In order to move a docking window, click on its name (at the top of the window) and then drag and drop it to the destination.



Mode

○ Not used
● Master (for field)
○ Slave (for binding)

Baud rate

● 500 Kb/s
○ 250 Kb/s
○ 125 Kb/s
○ 50 Kb/s

Master Settings

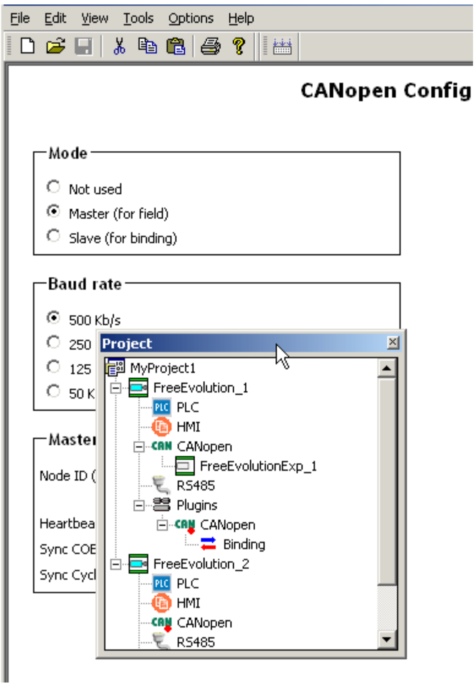
Node ID (1..127): 1

Heartbeat time (ms): 0

Sync COBID: 0

Sync Cycle (ms): 0

You can make the tool window float, by double-clicking on its name, or by pressing the *CTRL* key, or by moving the tool window away from the main window borders.



CANopen Config

Mode
○ Not used
● Master (for field)
○ Slave (for binding)

Baud rate
● 500 Kb/s
○ 250
○ 125
○ 50 K

Maste
Node ID (
Heartbea
Sync COE
Sync Cycl

A tool window can be resized by clicking-and-dragging on its border until the desired size is reached.

3. MANAGING PROJECTS

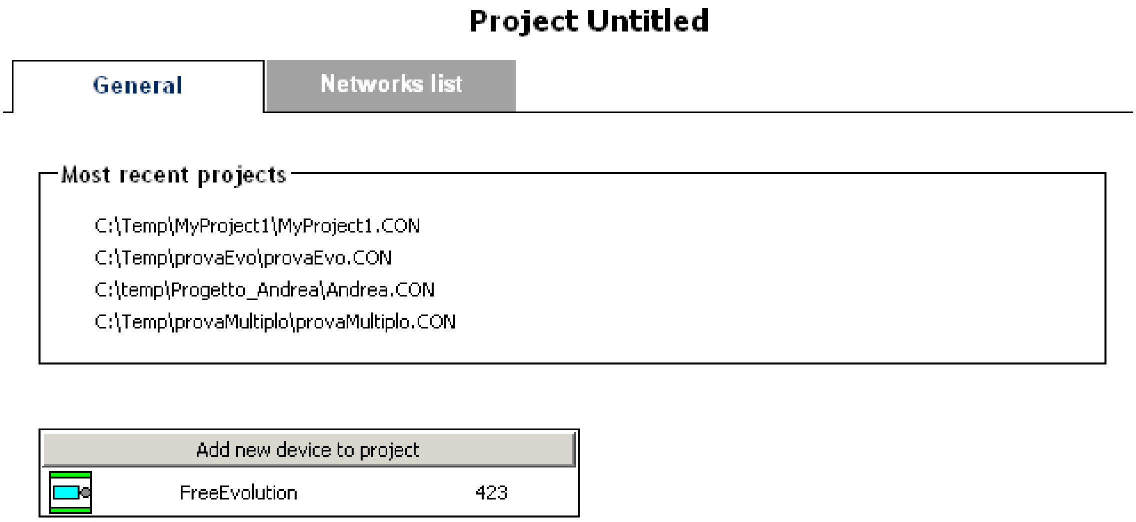
3.1 CREATING A NEW PROJECT AND MAIN PAGE

When you open FREE Studio Connection, you are presented with the `Main page`.

In the `General` tab you can open the last recently opened projects (shown in upper section) or insert a new device in the project, selecting it in the lower panel.

Here you can see all the "top level" devices that you can add, and this window shows the same content of the `Catalog window` when the root item is selected in the `Project window`; therefore it follows the same behavior with respect to the `Show all versions in catalog` flag.

With a just a click in the list, a new device is inserted in the project tree, ready to be configured and programmed.



Most recent projects

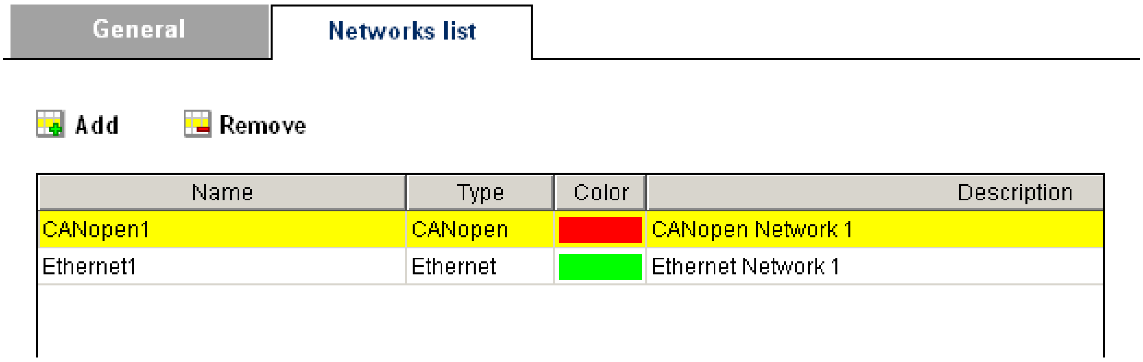
C:\Temp\MyProject1\MyProject1.CON
C:\Temp\provaEvo\provaEvo.CON
C:\temp\Progetto_Andrea\Andrea.CON
C:\Temp\provaMultiplo\provaMultiplo.CON

Add new device to project		
	FreeEvolution	423

In the second tab of the `Main page`, named `Networks list`, you can manage a list of all the "virtual networks" to be used in your project with the devices that will be connected with Binding I/O.

For each network you have to choose a name, the protocol to use (CANopen or Ethernet/ModbusTCP) and symbolic color to show as a small circle in the project tree; each device connected to the same network will be shown with the same color.

While by default you already have two predefined networks (one CANopen and one Ethernet) you can add any number of other networks, to build complex scenarios.



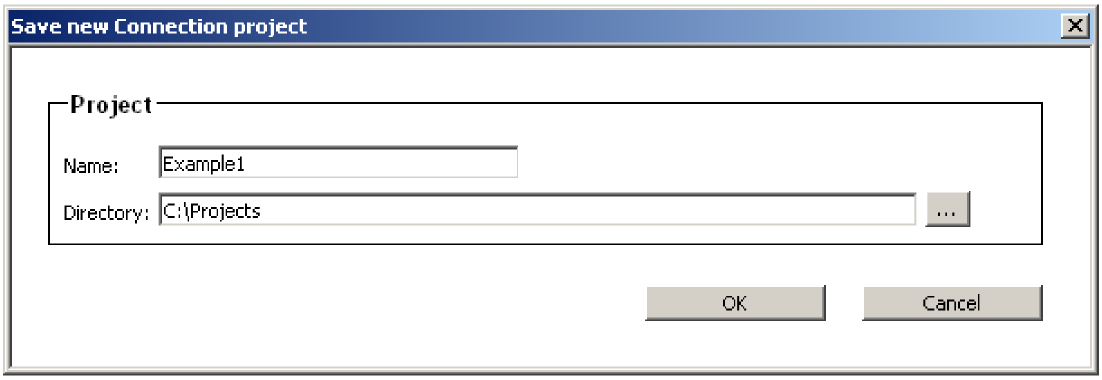
Name	Type	Color	Description
CANopen1	CANopen	🟥	CANopen Network 1
Ethernet1	Ethernet	🟩	Ethernet Network 1

3.2 SAVING THE PROJECT

To save the project, you can select the corresponding item of the menu `File` or the `Main` toolbar.

The Connection project is a single file that has `.CON` extension; it links other sub-components (like PLC application or HMI pages) that typically reside in the same containing folder.

If you are saving a new project (that is still `Untitled`), you are presented with a dialog that asks you the new name for the project and the directory where to save it; the program will create a folder of the chosen `Name` under the chosen `Directory`, and will save a file named like `Name.CON` under it.



Name: Example1

Directory: C:\Projects

In the above example, the folder `C:\Projects\Example1\` will be created and the project will be saved as `C:\Projects\Example1\Example1.CON`.

If you want to save the project with another name, you can choose the command `File / Save as...` and specify a new name and location for the `.CON` file.

IMPORTANT: only the `.CON` project file is saved, no folder is created nor the linked components (PLC or HMI) are copied or moved.

3.3 MANAGING EXISTING PROJECTS

3.3.1 OPENING AN EXISTING PROJECT

To open an existing project, click `Open` in the `File` menu of Connection's main window, or in the `Main` toolbar. This will open a dialog box, which lets you browse to the directory containing the project and select the relative project file.

Otherwise, you can select one of recently opened projects from the `File` menu or in the `Main page`.

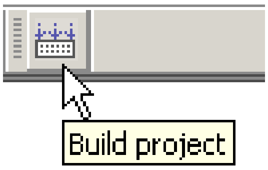
3.3.2 CLOSING THE PROJECT

You can terminate the working session either by:

- starting a new project, with the `File / New` command, or the button in the toolbar;
- explicitly closing the current project with `File / Close` command;
- by exiting Connection.

In all cases, when there are changes not yet saved to file, the program asks you to choose between saving and discarding them.

3.4 BUILDING PROJECTS



When you press the *Build project* button in the toolbar (or the *F7* key), Connection will examine the current project and will:

- Print in the output window any error it found in the checking process; you can then double-click the error to see the source position.
- Generate specific configuration files for each device (for example *CONNEC.PAR* for FREE Evolution, with Field and Binding configuration settings).
- Generate a single *.CFN* file to be used in FREE Studio Device; this file will contain all the sub-components of the current project (devices configurations, PLC applications and HMI pages) all contained inside the *CFN*, in a redistributable form; this file will have the same name of the *.CON* project and will reside in the same folder.

Choosing the *Tools / Open with FREE Studio Device* command, Device will be launched with the generated *CFN* file opened.

IMPORTANT: before executing compilation, please make sure that all the PLC and HMI sub-project have been built with the respective tools (FREE Studio Application and User-Interface). In fact Connection will include in the CFN the last compilation output of each sub-component, so you have to build them BEFORE compiling the Connection project.

3.5 DISTRIBUTING PROJECTS

This topic should be discussed in two different parts:

3.5.1 DISTRIBUTING TO OTHER DEVELOPERS

To distribute the full Connection project to other developers (for example for further development or debugging) you can give the entire folder containing the *.CON* file, that has been created by Connection with the first *Save* command.

In this way all the sub-components created by Connection (PLC, HMI, CFN file) are all contained inside, and since the file paths are maintained as relatives the project can be moved around; so other developers can open the Connection project anywhere and work normally.

One important exception is for *.CON* projects that link external components, for example external PLC projects (on an external directory, or taken from catalog); in this scenario you will have to distribute all the external components manually, because they are not self-contained in the main project folder.

3.5.2 DISTRIBUTING TO USERS OR INSTALLERS

In this scenario, it is enough to distribute the *CFN* file (FREE Studio Device document) created by Connection; you will be able to download everything (PLC, HMI, config files, parameters values) only using Device with a single click.

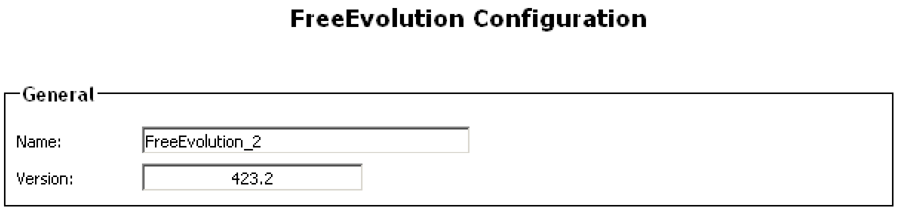
One important exception is for *.CON* projects that link external components from the catalog (PLC and HMI), in this case the produced *CFN* file will not include them; they must be distributed manually.

4. MANAGING PROJECT ELEMENTS

4.1 FREE EVOLUTION

FREE Evolution is one of the top-level devices that you can insert in the project.

On its main page you can change its name and see a picture of it.

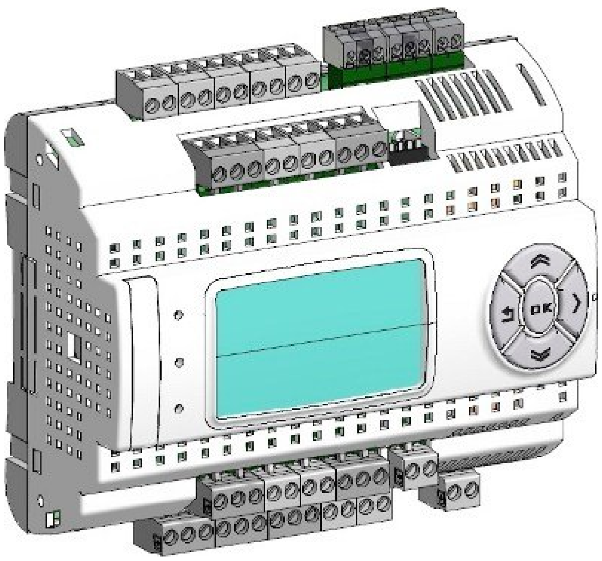


FreeEvolution Configuration

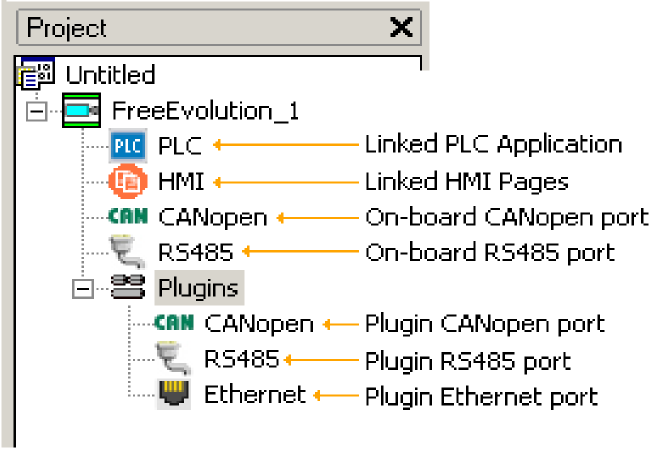
General

Name: FreeEvolution_2

Version: 423.2



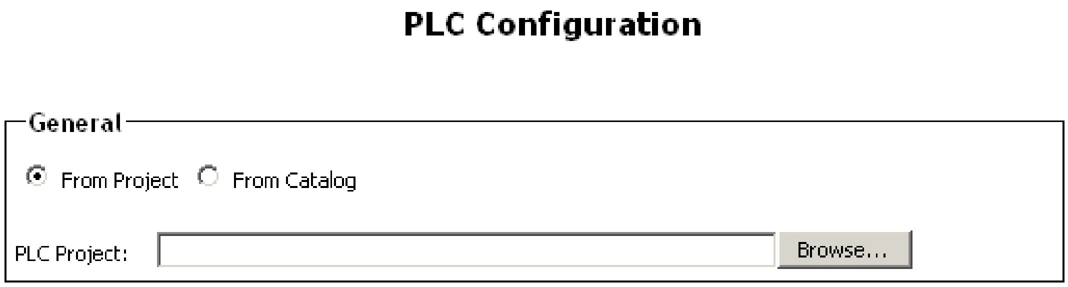
It can run both a PLC application and HMI pages on the same CPU and has a lot of connectivity capabilities, in terms of on-board connectors and may optional plug-ins.



Follows detailed description of each element.

4.1.1 PLC

This tree item lets you create or associate a PLC project to the FREE Evolution; the associated page shows the relative path of the associated *PPJS* file (Application project).



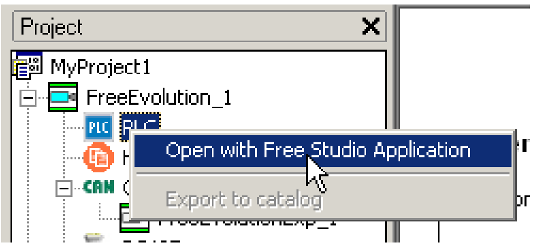
PLC Configuration

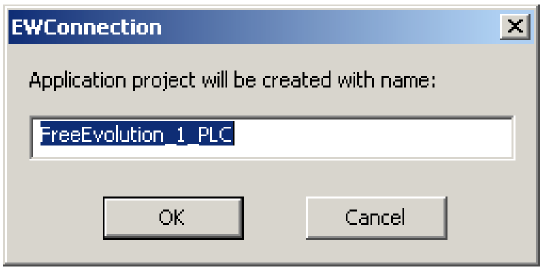
General

● From Project ○ From Catalog

PLC Project:

If you do a right-click on the *PLC*, a pop-up menu will appear with the command *Open with FREE Studio Application*; if the device has no associated project, you will be prompted for the name to give to the new application (by default, the name of the device with the *_PLC* suffix).





EWConnection

Application project will be created with name:

FreeEvolution_1_PLC

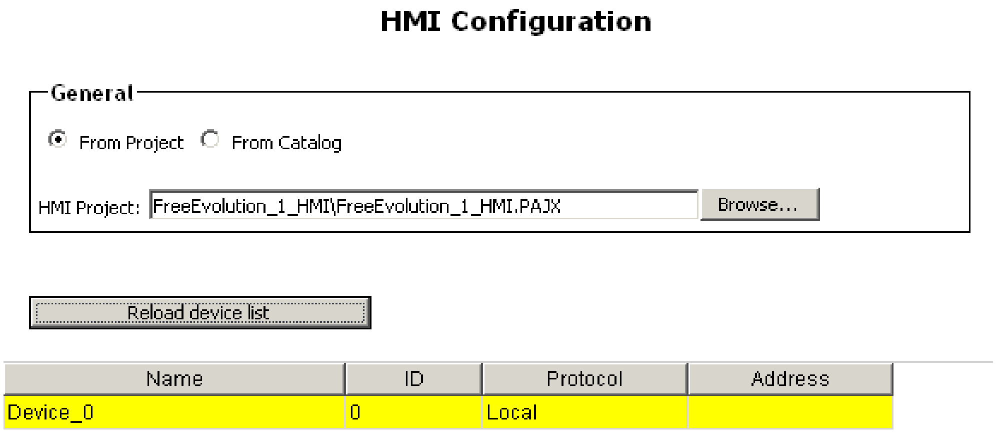
Otherwise if a PLC project has already been associated, Application will be launched and the existing PLC project opened.

If you want to manually associate an existing PLC project to the device, you can choose between a project on the disk in a particular folder or choosing from the local catalog of applications.

If a PLC project has been associated, the *Export to catalog* command in the pop-up menu will be enabled, allowing you to export the application in the catalog for further reuse.

4.1.2 HMI

This tree item lets you create or associate a HMI project to the FREE Evolution; the associated page shows the relative path of the associated *PAJX* file (UserInterface project).



HMI Configuration

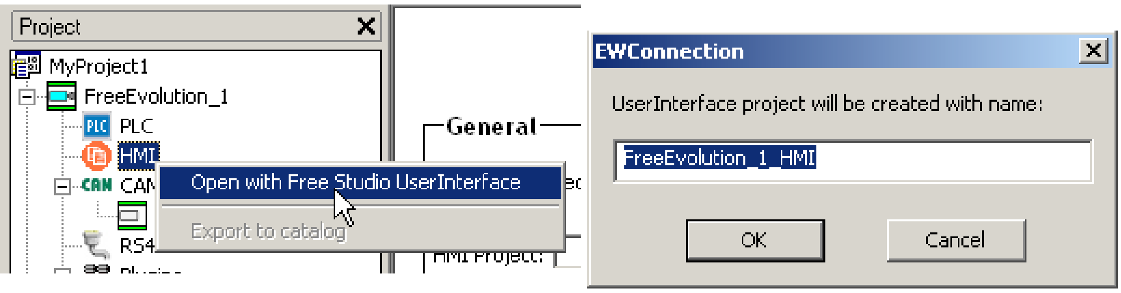
General

● From Project ○ From Catalog

HMI Project: FreeEvolution_1_HMI\FreeEvolution_1_HMI.PAJX

Name	ID	Protocol	Address
Device_0	0	Local	

If you do a right-click on the *HMI*, a pop-up menu will appear with the command *Open with FREE Studio UserInterface*; if the device has no associated project, you will be prompted for the name to give to the new application (by default, the name of the device with the *_HMI* suffix).



UserInterface project will be created with name:

FreeEvolution_1_HMI

Otherwise if a HMI project has already been associated, UserInterface will be launched and the existing HMI project opened.

If you want to manually associate an existing HMI project to the device, you can choose between a project on the disk in a particular folder or choosing from the local catalog of applications.

If a HMI project has been associated, the *Export to catalog* command in the pop-up menu will be enabled, allowing you to export the application in the catalog for further reuse.

4.1.2.1 RETRIEVING REMOTE DATA FROM LOCAL HMI PAGES

If in your HMI pages project you have imported one or more parameter map, you can configure the real address of the remote device here.

In fact by default any parameter map is considered as “local”, and if you want to view in your page any parameter of a remote device you have to insert here (and so outside and independently from UserInterface) the used protocol (Modbus RTU, Modbus TCP or CANopen) and address.

In this way you can design the HMI pages in UserInterface as they were “local” and then later change the real address of the remote device without even recompiling the *PAJX* project (the change is made only in Connection).

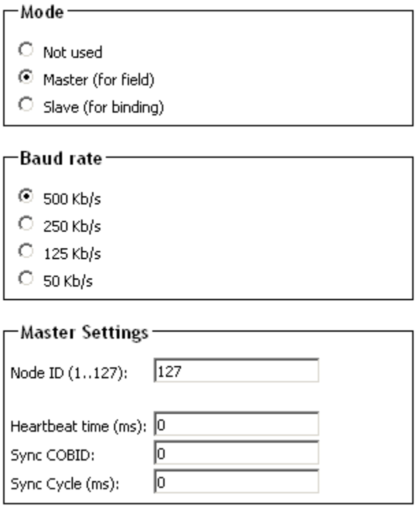
To load or update the list of remote devices (parameter maps) inserted in the UserInterface project, press the *Reload device list* button; please remember to build the *PAJX* project with UserInterface to have an updated list before doing this.

4.1.3 CANOPEN

FREE Evolution has one on-board CANopen port, plus another one available as an external plug-in. Each port can be configured as *Not used* (disabled), *Master* (field), *Slave* (binding).

4.1.3.1 FIELD

When you configure the CANopen port as *Master* the FREE Evolution will act as a CANopen master on this port, so you can attach any number of CANopen slave devices here and exchange data with Field I/O.



Mode

○ Not used

● Master (for field)

○ Slave (for binding)

Baud rate

● 500 Kb/s

○ 250 Kb/s

○ 125 Kb/s

○ 50 Kb/s

Master Settings

Node ID (1..127): 127

Heartbeat time (ms): 0

Sync COBID: 0

Sync Cycle (ms): 0

For a CANopen master port, you have to configure (see 5.1 for further informations):

- Baud rate used in this CANopen network (in Kb/s).
- Node ID for the master (1..127), by default is 127.
- Heartbeat time in ms, by default 0 (heartbeat producer disabled): it is the master producer heartbeat time.
- The SYNC COBID to use, by default 128.
- The period for the SYNC cycle in ms, by default 0 (sync disabled).

Example of possible slaves are the FREE Evolution Expansion module (see 4.5) or generic custom devices imported from their EDS files (CAN custom, see 4.4).

After you added and configured the various CANopen slaves, you can establish the "link" between the remote objects of the slave and the internal PLC variables to read or write.

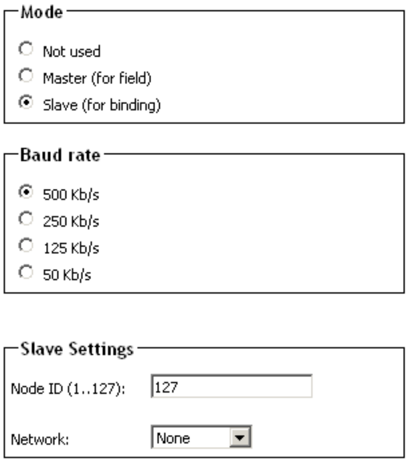
The set of PLC objects you can read or write is made of:

- Status variables, created with FREE Studio Application (not BIOS).
- Field variables, created with FREE Studio Application.

4.1.3.2 BINDING

When you configure the CANopen port as `Slave` the FREE Evolution will act as a CANopen slave on this port, so you can exchange data with Binding I/O with other FREE Evolution devices on the CANopen network.

Configuring the port



Mode

○ Not used

○ Master (for field)

● Slave (for binding)

Baud rate

● 500 Kb/s

○ 250 Kb/s

○ 125 Kb/s

○ 50 Kb/s

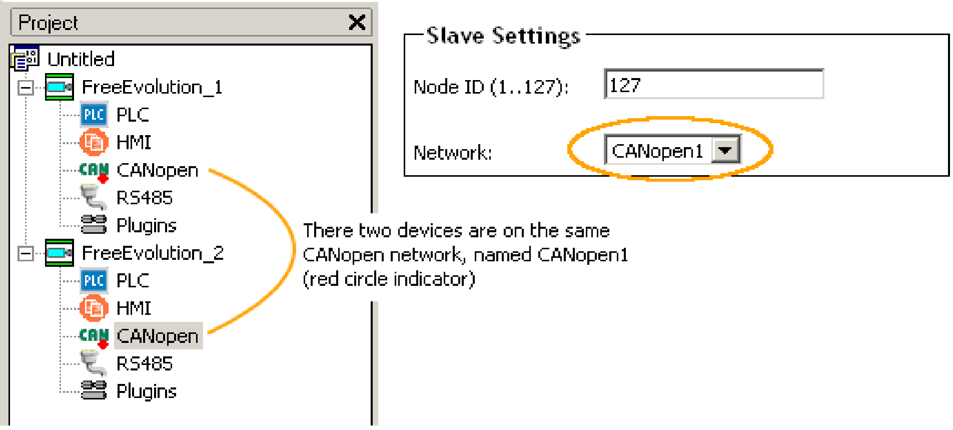
Slave Settings

Node ID (1..127): 127

Network: None

For a CANopen slave port, you have to configure:

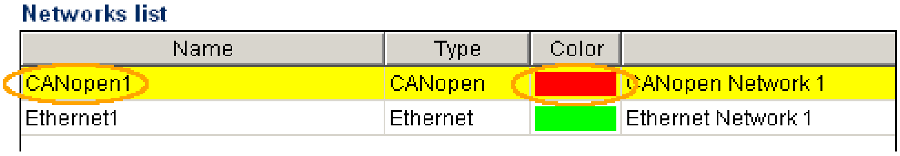
- Baud rate used in this CANopen network (in Kb/s).
- Node ID for the slave (1..127), by default is 127.
- The "virtual network" where this FREE Evolution is attached; in the tree will appear a small colored circle of same color of the chosen network (same color means same network).



Node ID (1..127): 127

Network: CANopen1

There two devices are on the same CANopen network, named CANopen1 (red circle indicator)



Networks list

Name	Type	Color	
CANopen1	CANopen	🟥	CANopen Network 1
Ethernet1	Ethernet	🟩	Ethernet Network 1

The *Binding* object

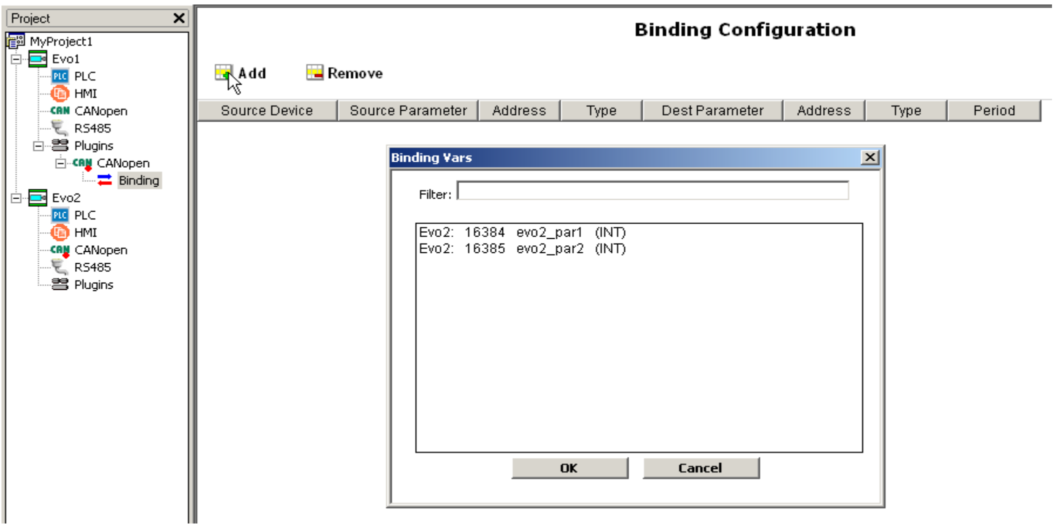
When you configure a CANopen port as *Slave*, you can add under it a *Binding* object: add it if a device wants to READ objects from other ones, while it not needed if the device only SEND objects on the network.

The set of PLC objects you can send or receive is made of:

- EEPROM parameters, created with FREE Studio Application (not BIOS).
- Status variables, created with FREE Studio Application (not BIOS).

Clicking the *Binding* object shows its configuration page: here is a grid where you have to insert all the remote objects to read, and link them to the local destinations.

To do this click the *Add* button, a window showing all the "public" objects from all other devices on this same network will appear; here you can apply search filters and choose which objects to read from (multi-selection is also supported).



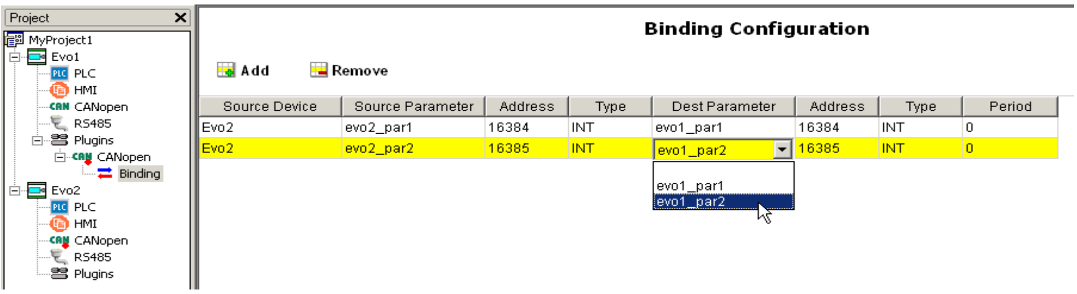
Source Device	Source Parameter	Address	Type	Dest Parameter	Address	Type	Period

Filter:

```
Evo2:  16384  evo2_par1  (INT)
Evo2:  16385  evo2_par2  (INT)
```

Once you inserted the remote objects to read, you have to assign the local destination locations to write to, choosing with the list in the `Dest parameter` column or manually inserting the `Address`.

IMPORTANT: please remember to rebuild the PLC project with Application to see an updated list of public objects here.



Binding Configuration

Source Device	Source Parameter	Address	Type	Dest Parameter	Address	Type	Period
Evo2	evo2_par1	16384	INT	evo1_par1	16384	INT	0
Evo2	evo2_par2	16385	INT	evo1_par2	16385	INT	0

In the above example:

- *Evo1* will read from *Evo2* the *evo2_par1* object and will put it in its local *evo1_par1* object.
- *Evo1* will read from *Evo2* the *evo2_par2* object and will put it in its local *evo1_par2* object.

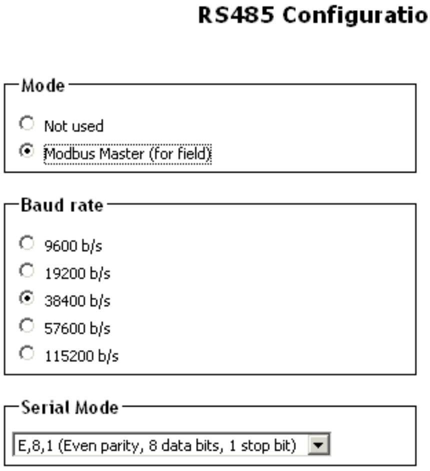
In the *Period* you can configure in detail the single period for each parameter; the object will be updated every “period” ms.

4.1.4 RS485

FREE Evolution has one on-board RS485 port, plus another one available as an external plug-in. Each port can be configured as *Not used* (disabled) or *Master* (field).

4.1.4.1 FIELD

When you configure the RS485 port as *Master* the FREE Evolution will act as a ModbusRTU master on this port, so you can attach any number of Modbus slave devices here and exchange data with Field I/O.



RS485 Configuration

Mode
○ Not used
● Modbus Master (for field)

Baud rate
○ 9600 b/s
○ 19200 b/s
● 38400 b/s
○ 57600 b/s
○ 115200 b/s

Serial Mode
E,8,1 (Even parity, 8 data bits, 1 stop bit)

For a Modbus master port, you have to configure:

- Baud rate used in this Modbus network (in b/s).
- Serial mode (parity, data bits, stop bits).

Example of possible slaves are the FREE Evolution Expansion module (see 4.5), `Generic Modbus` devices (see 4.2), or custom devices created with the ModbusCustomEditor tool (see 4.3).

After you added and configured the various Modbus slaves, you can establish the “link” between the remote objects of the slave and the internal PLC variables to read or write.

The set of PLC objects you can read or write is made of:

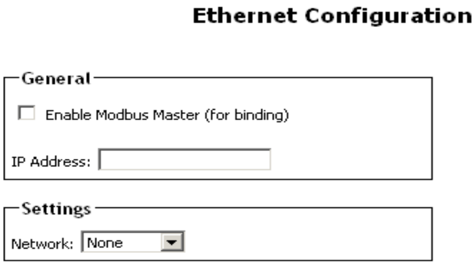
- Status variables, created with FREE Studio Application (not BIOS).
- Field variables, created with FREE Studio Application.

4.1.5 ETHERNET

FREE Evolution can have one Ethernet port, available as an external plug-in. The port always acts a Modbus TCP slave, and additionally can be configured also as `Master` (binding).

4.1.5.1 BINDING

Configuring the port



Ethernet Configuration

General

□ Enable Modbus Master (for binding)

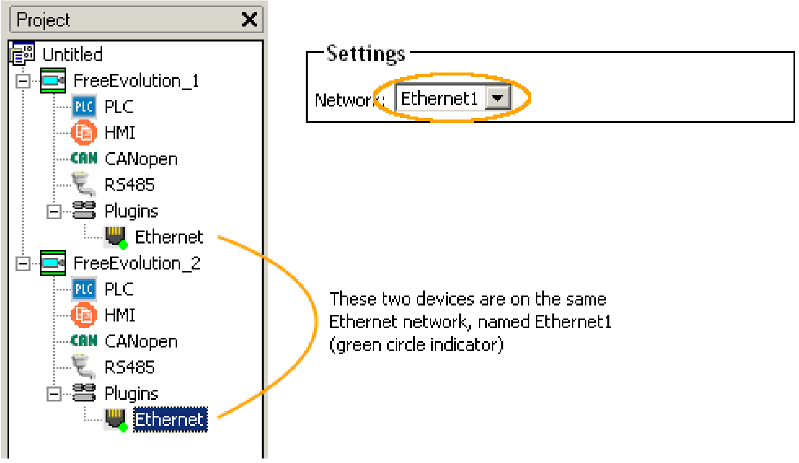
IP Address:

Settings

Network: None

For an Ethernet port, you have to configure:

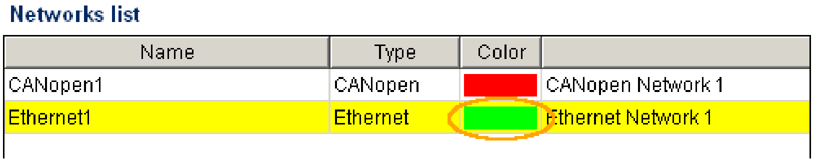
- if it acts also a `Master` (otherwise only `Slave` is implied);
- its IP address;
- the “virtual network” where this FREE Evolution is attached; in the tree will appear a small colored circle of same color of the chosen network (same color means same network).



Settings

Network: Ethernet1

These two devices are on the same Ethernet network, named Ethernet1 (green circle indicator)



Networks list

Name	Type	Color	
CANopen1	CANopen	🟥	CANopen Network 1
Ethernet1	Ethernet	🟩	Ethernet Network 1

The *Binding* object

When you configure a Ethernet port as *Master*, you can add under it a *Binding* object: add it if a device wants to READ objects from other ones, while it not needed if the device only SEND objects on the network (in this case you do not even need to activate the *Master* feature).

The set of PLC objects you can send or receive is made of:

- EEPROM parameters, created with FREE Studio Application (not BIOS).
- Status variables, created with FREE Studio Application (not BIOS).

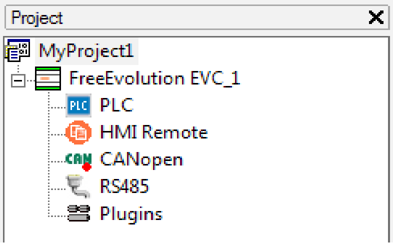
The configuration page for the *Binding* object in Modbus TCP is the same of CANopen, so see 4.1.3.2 for a description and usage of this page.

Because the interface is the same between the two protocols, you can focus on designing your distributed application without knowing the specific communication protocol details.

The only difference from CANopen Binding is that here you have one more column named *Timeout*, where you can configure the specific time-out in ms for each object exchanged.

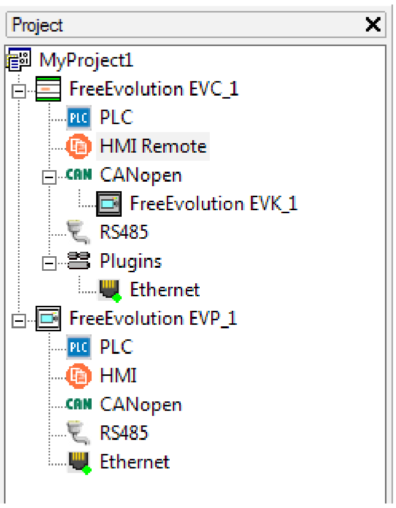
4.2 FREE EVOLUTION EVC

FREE Evolution EVC is a top-level device that has the same characteristics and network behaviour of a FREE Evolution device but does not support local HMI. In fact it has no on-board display to show its own pages. FREE Evolution EVC supports HMI Remote so its pages can be downloaded and shown by FREE Evolution EVK or FREE Evolution EVP keyboards.



Please refer to 4.1 - FREE Evolution chapter for a full description of all FREE Evolution EVC features.

Usage example



In this scenario FreeEvolution EVC_1 device has a PLC project and has an HMI Remote project that makes available EVC pages for linked keyboards.

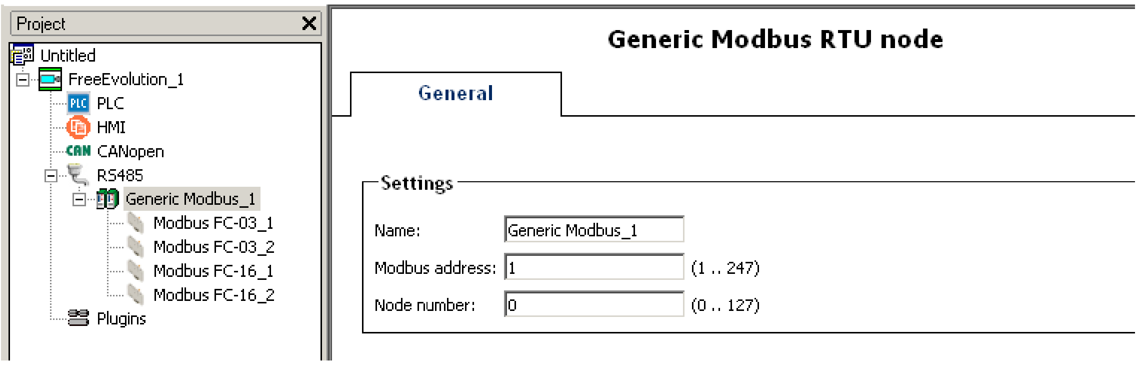
EVC HMI Remote pages can be remoted and shown by FreeEvolution EVK_1 via CANopen field and by FreeEvolution EVP_1 via Ethernet network.

4.3 GENERIC MODBUS

The `Generic Modbus` object is a generic Modbus slave that can be inserted under the RS485 port of the FREE Evolution, when configured as `Modbus master`.

You can use the `Generic Modbus` when you want to manually configure and have full control over the single Modbus messages to send to the slave.

Another typical usage is for third-party devices that you plan to use just once in your projects, and you do not want to put in the catalog for future reuse.



Name: Generic Modbus_1

Modbus address: 1 (1 .. 247)

Node number: 0 (0 .. 127)

In the main page of the `Generic Modbus` you can configure:

- A name for the object in the project.
- Its Modbus address (in the range 1..247).
- Its Node number (in the range 0..127); this value is incremented automatically, and can be used in the PLC program to index the `SysMbMRtuNodeStatus[]` array, that cointains diagnostic information about each slave node.

4.3.1 MODBUS MESSAGES

The `Generic Modbus` object alone will do nothing; you have to add under it one or more `Modbus messages`, that are specific Modbus function requests that will be sent on the bus.

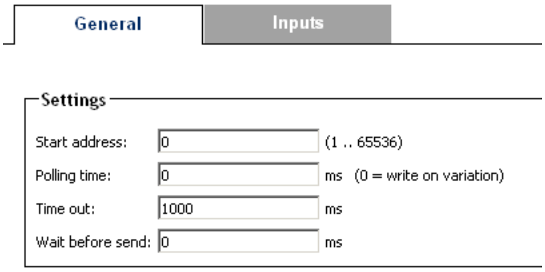
The following messages are supported:

- Function 2 (Read discrete inputs, 0x2): reads one or more read-only digital input (1-bit objects).
- Function 3 (Read holding registers, 0x3): reads one or more read-write register (16-bit objects).
- Function 4 (Read input registers, 0x4): reads one or more read-only register (16-bit objects).
- Function 15 (Write multiple coils, 0xF): writes one or more digital output (1-bit objects).
- Function 16 (Write multiple registers, 0x10): writes one or more register (16-bit objects).

The messages will be processed in the order they are inserted in the tree.

4.3.1.1 GENERAL TAB

For each message, in its `General` tab you can configure.



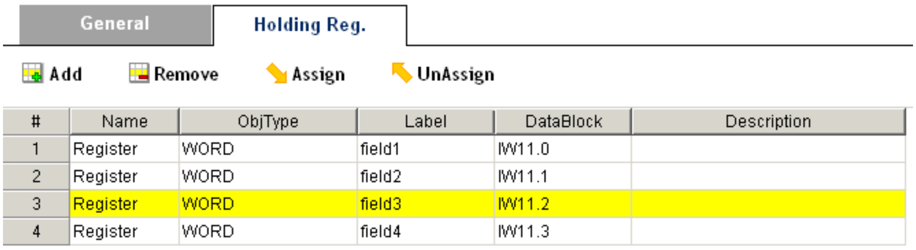
Settings

Start address: 0 (1 .. 65536)
Polling time: 0 ms (0 = write on variation)
Time out: 1000 ms
Wait before send: 0 ms

- Start address: address of the first modbus object to read or write (1..65536).
- Polling time: the message will be processed with this period (ms); for writing operations, `0` means to write it only on variation of the value, for reading operations `0` means maximum speed.
- Timeout: the operation will fail when this time-out expires (ms).
- Wait before send: this is an additional timeout, to be used with slow slaves that do not answer if the messages are sent too fast.

4.3.1.2 REGISTERS TAB

Beside the `General` tab, each different message has a second tab where you can configure the list of objects to read or write.

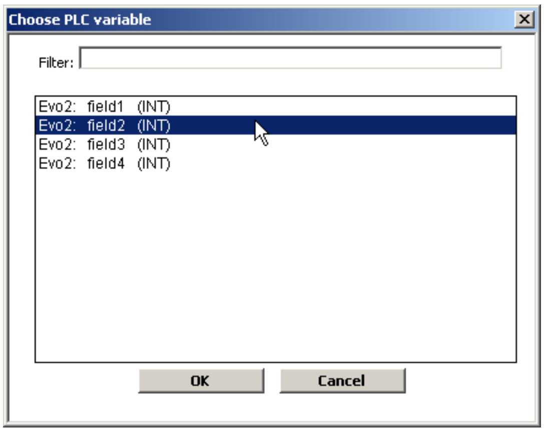


#	Name	ObjType	Label	DataBlock	Description
1	Register	WORD	field1	IW11.0	
2	Register	WORD	field2	IW11.1	
3	Register	WORD	field3	IW11.2	
4	Register	WORD	field4	IW11.3	

Using the `Add` button, insert one row for each Modbus object to read or write, up to 16 elements; the first row has the address configured in the `Start address` box in the `General tab`, and the other rows increment and follow.

For each row, press the `Assign` button to choose the PLC object to link and to be read or written with this Modbus message; you can not leave unassigned rows in the message.

IMPORTANT: please remember to rebuild the PLC project with Application to see an updated list of PLC variables here.



```
Evo2:  field1  (INT)
Evo2:  field2  (INT)
Evo2:  field3  (INT)
Evo2:  field4  (INT)
```

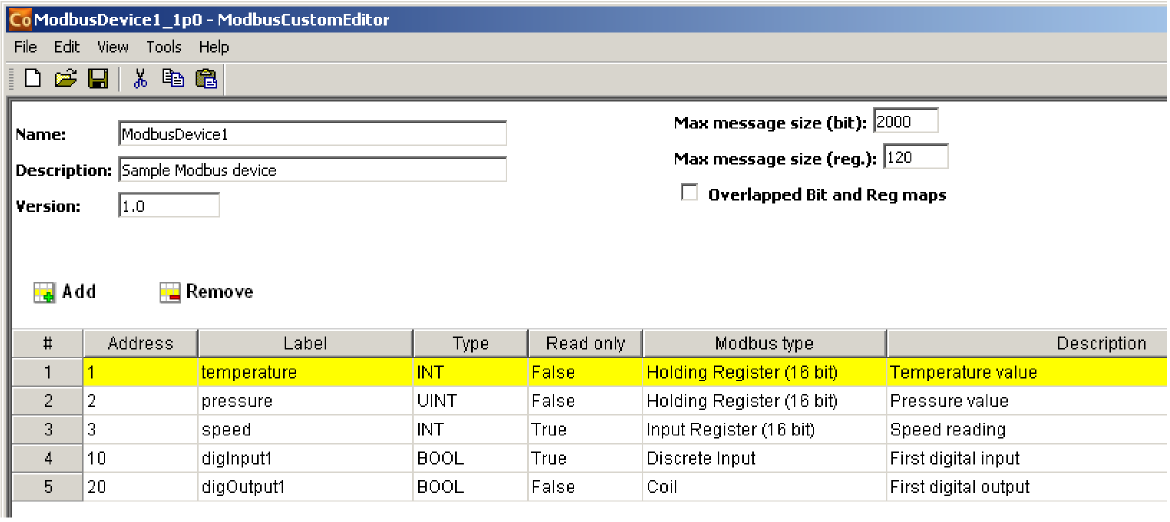
4.4 MODBUS CUSTOM

Modbus custom devices can be created and edit directly by the user.

In this way you can use in your project and add in the catalog for future reuse any third-party Modbus slave, characterizing its Modbus map only the first time and simplifying its further use, because you do not have to care about Modbus messages and functions anymore.

4.4.1 CREATING A NEW MODBUS CUSTOM DEVICE

To create a new *Modbus custom* device, choose *Tools / Run ModbusCustomEditor*; the external ModbusCustomEditor tool will be launched, with a new empty document.



Name: ModbusDevice1

Description: Sample Modbus device

Version: 1.0

Max message size (bit): 2000

Max message size (reg.): 120

□ Overlapped Bit and Reg maps

#	Address	Label	Type	Read only	Modbus type	Description
1	1	temperature	INT	False	Holding Register (16 bit)	Temperature value
2	2	pressure	UINT	False	Holding Register (16 bit)	Pressure value
3	3	speed	INT	True	Input Register (16 bit)	Speed reading
4	10	digInput1	BOOL	True	Discrete Input	First digital input
5	20	digOutput1	BOOL	False	Coil	First digital output

Here you can configure:

- Name of the device.
- Long description for the device.
- A version number.
- Overlapping of bit and register maps: check this if the device has both a *0* register and a *0* bit (in other words it has different addressing of 16-bit and 1-bit objects), uncheck this if the address is unique and so duplicated are not allowed, even if the type is different.
- Max message size: insert here the maximum number of registers per message supported by the device.

Then, using the *Add* button, add one row for each Modbus object of the device; you have to insert its address, name, type (note that *Type* and *Read only* columns are linked with the *Modbus type* column) and optionally a long description.

When you finish, save the current device definition; you will be prompted for a file name with *.PCT* extension, by default it will be proposed the current name+version.

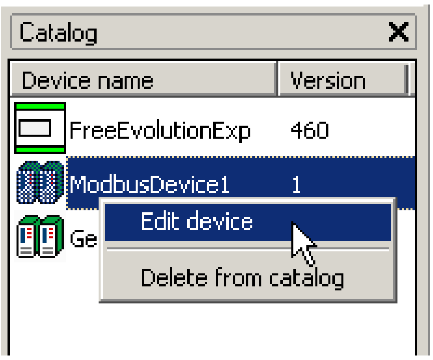
The file will be saved in the special *ModbusCustom* folder in the catalog; now you can close the ModbusCustomEditor and go back in Connection to use the new device.

4.4.2 EDITING AN EXISTING MODBUS CUSTOM DEVICE

To edit an existing *Modbus custom* device, you can:

- Run the ModbusCustomEditor with the *Tools / Run ModbusCustomEditor* command, and then manually open the PCT file with the standard *File / Open* command.

- When the device you want to edit is visible in the *Catalog window* (for example when a RS485 node is selected and is in *Master* mode), you can right-click on it and choose the *Edit device* command; the ModbusCustomEditor will be launched and the selected device opened.



IMPORTANT: when the ModbusCustomEditor is running, Connection is blocked waiting for it to be closed.

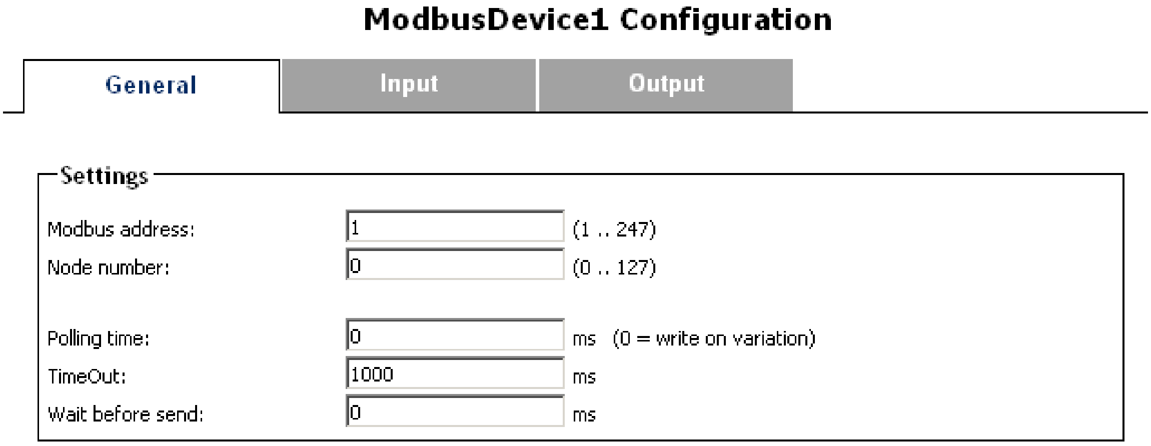
4.4.3 DELETING A MODBUS CUSTOM DEVICE

To delete an existing *Modbus custom* device, when the device is visible in the *Catalog window* do a right-click and choose *Delete from catalog* (see previous paragraph).

4.4.4 USING A MODBUS CUSTOM DEVICE

When you insert the *Modbus custom* device as a Modbus slave (for example under a RS485 port) and click on it on the tree, you will see a page with three tabs.

4.4.4.1 GENERAL TAB



ModbusDevice1 Configuration

Modbus address:	1	(1 .. 247)
Node number:	0	(0 .. 127)
Polling time:	0	ms (0 = write on variation)
TimeOut:	1000	ms
Wait before send:	0	ms

In the *General* tab you can configure:

- Its *Modbus address* (in the range 1..247).
- Its *Node number* (in the range 0..127); this value is incremented automatically, and can be used in the PLC program to index the `SysMbMRtuNodeStatus[]` array, that cointains diagnostic information about each slave node.
- *Polling time*: the Modbus messages will be processed with this period (ms); for writing operations, *0* means to write it only on variation of the value, for reading operations *0* means maximum speed.
- *Timeout*: the operation will fail when this time-out expires (ms).

- `Wait before send`: this is an additional timeout, to be used with slow slaves that do not answer if the messages are sent too fast.

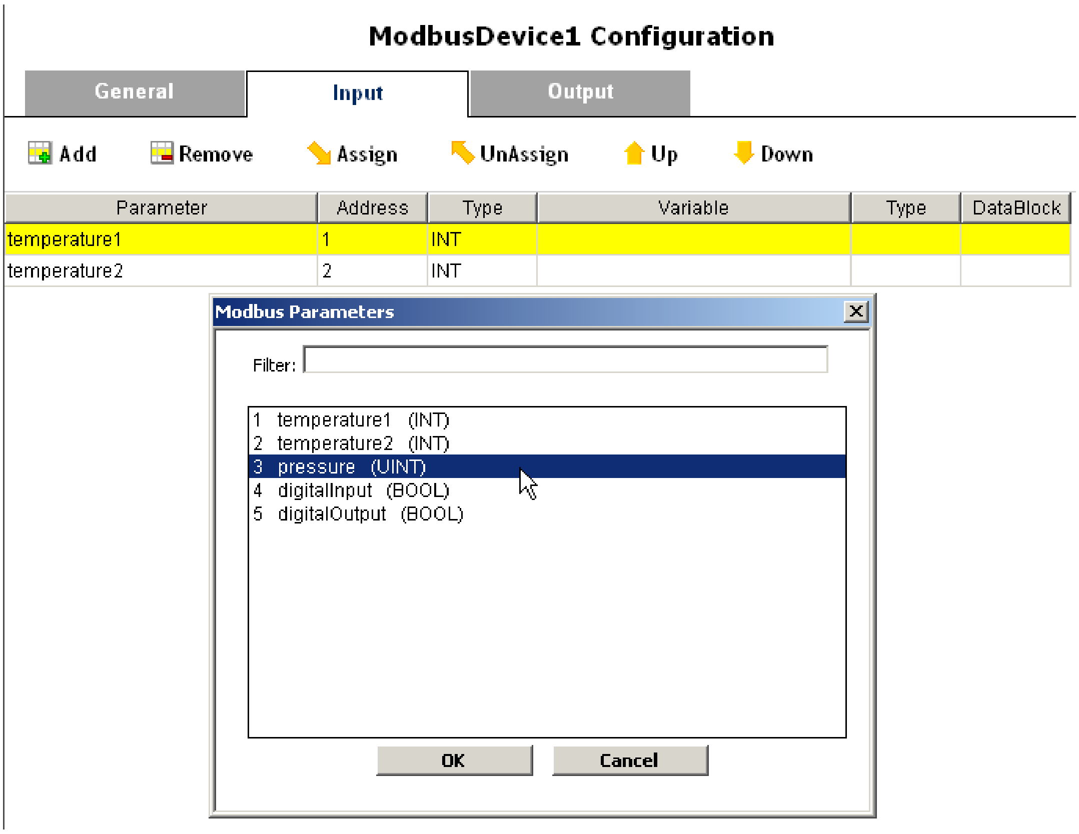
Here you can notice that for `Modbus custom` the `Polling time`, `Timeout` and `Wait before send` are generic for the whole device, while for the `Generic Modbus` you can put specific different values for each single message. This is because with the `Modbus custom` the low-level Modbus messages are automatically calculated and you do not have to worry about them, but as a side-effect you can not “fine-tune” them, because these settings are global.

4.4.4.2 INPUT/OUTPUT TAB

Then, in the `Input` and `Output` tabs you can insert one row for each Modbus object to read or write; press the `Add` button and choose the parameters to exchange (multi selection is supported), and use the `Assign` button to link them to the PLC object to be read or written to.

Insert in the `Input` tab the Modbus objects to READ from the Modbus slave (and to put into PLC variables), and insert in the `Output` tab the Modbus objects to WRITE to the Modbus slave (and to get from the PLC variables).

IMPORTANT: please remember to rebuild the PLC project with Application to see an updated list of PLC variables here.



Parameter	Address	Type	Variable	Type	DataBlock
temperature1	1	INT			
temperature2	2	INT			

Filter:

```
1  temperature1  (INT)
2  temperature2  (INT)
3  pressure  (UINT)
4  digitalInput  (BOOL)
5  digitalOutput  (BOOL)
```

FREE Studio Connection will create the correct Modbus messages analyzing the sequence of addresses and types; if the addresses are consequent and the types are homogenous, different objects will be grouped in single messages to optimize the communication.

The maximum number of registers configured with the ModbusCustomEditor is also considered, along with the maximum number of registers per message of the master (that is 16 for the FREE Evolution).

The grouping and generation of the Modbus messages is totally automatic and recalculated at each compilation, so you do not have to know technical details of the Modbus protocol.

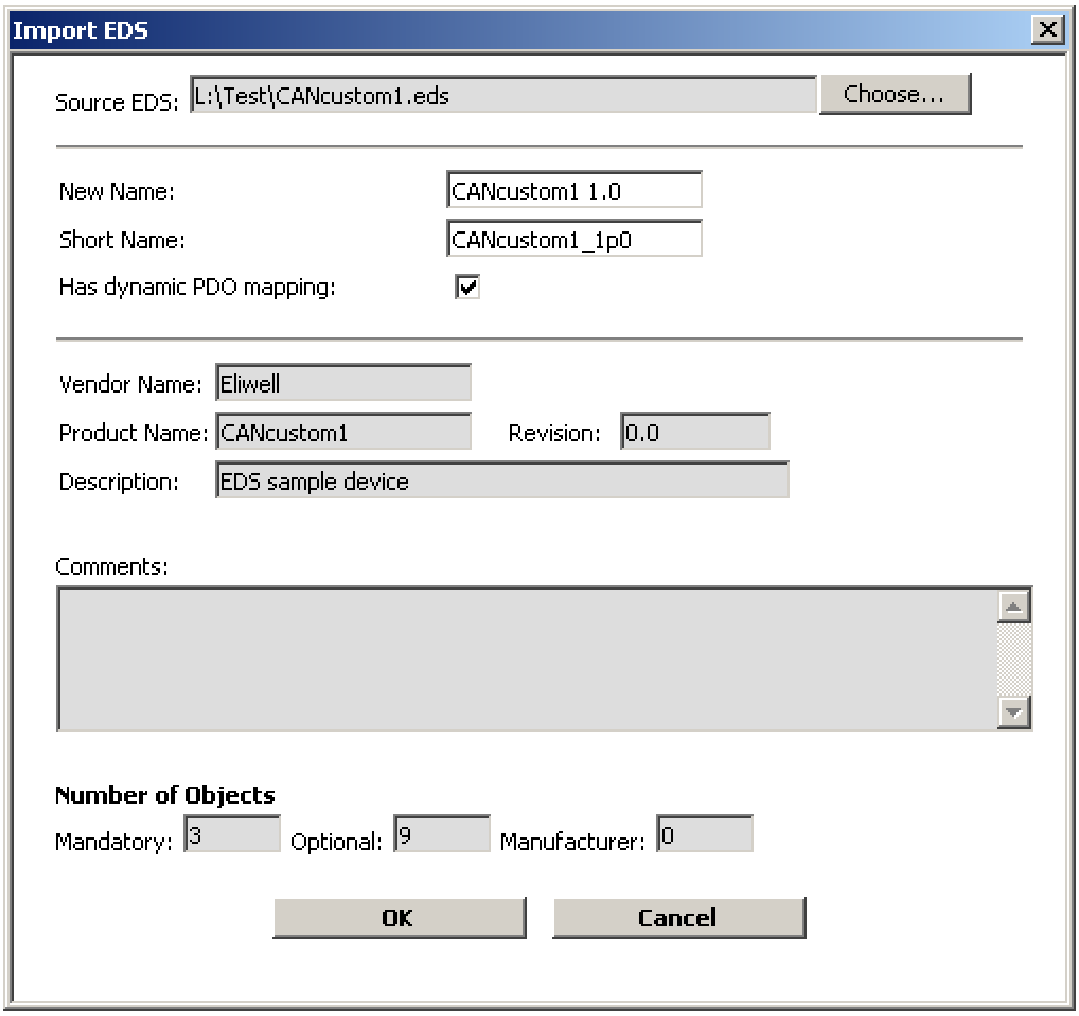
4.5 CAN CUSTOM

CAN custom device can be created and added to the Catalog by importing their *EDS* file. In this way you can use any third-party CANopen device as a slave, if it provides a standard-compliant *EDS* file (Electronic Data Sheet), that follows the DS306 CiA specification.

4.5.1 IMPORTING A NEW CAN CUSTOM DEVICE

To import a new *CAN custom* device, choose *Tools / Import from EDS* command.

The *Import EDS* window will appear.



Import EDS

Source EDS: L:\Test\CANcustom1.eds

New Name: CANcustom1 1.0

Short Name: CANcustom1_1p0

Has dynamic PDO mapping: ☑

Vendor Name: Eliwell

Product Name: CANcustom1 Revision: 0.0

Description: EDS sample device

Comments:

Number of Objects

Mandatory: 3 Optional: 9 Manufacturer: 0

Here you have to configure:

- The source *EDS* file to import, using the *Choose...* button.
- The full name of the device (by default is *Product name + Revision*).
- The short name, this must not include any special character or spaces.
- If the device supports dynamic PDO mapping or not: if you activate this option, you will be able to manually configure and change the default PDO mapping read from the EDS to match the actual mapping of the slave, otherwise the PDO mapping will be read-only and determined only by the EDS default values

After you have chosen the *EDS* file, the window will show a resume of the device characteristic and number of objects (detailed in mandatory, optional, manufacturer).

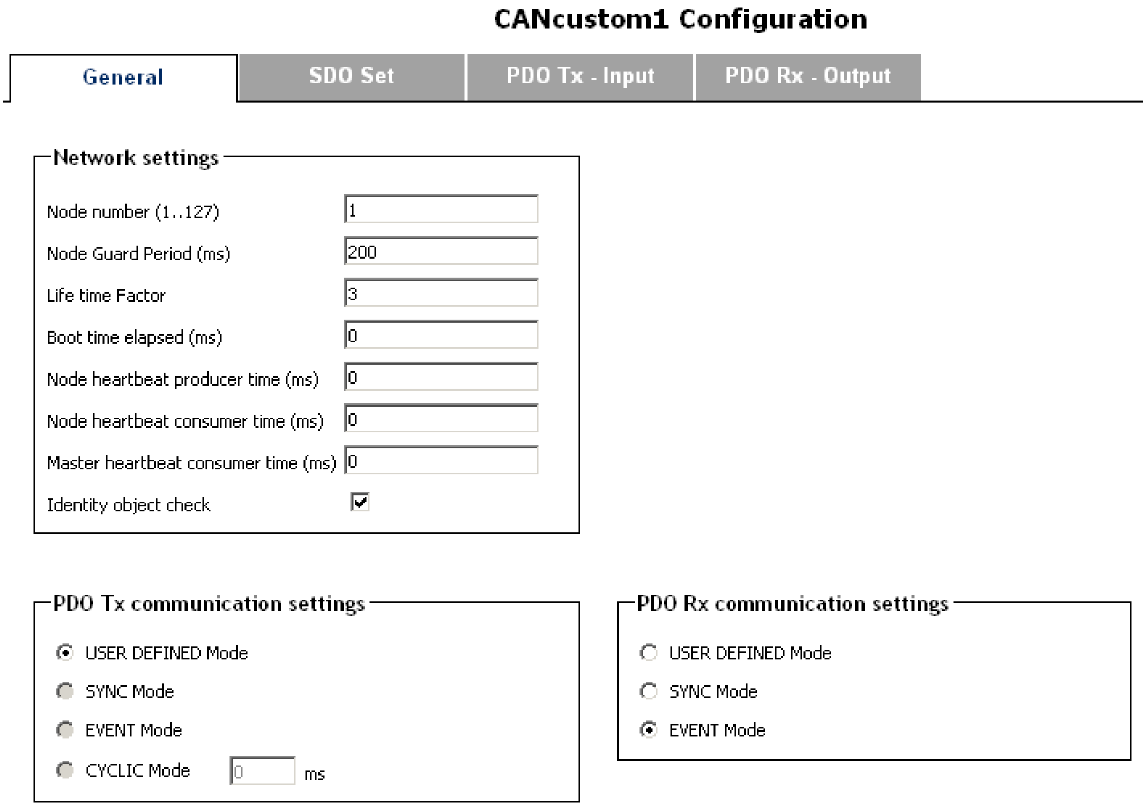
4.5.2 DELETING A CAN CUSTOM DEVICE

When the device you want to delete is visible in the *Catalog window* (for example when a CANopen port is selected and is in *Master* mode), you can right-click on it and choose the *Delete from catalog* command.

4.5.3 USING A CAN CUSTOM DEVICE

When you insert a *CAN custom* device as a CANopen slave (for example under a CANopen port) and click on it on the tree, you will see the following page.

4.5.3.1 GENERAL TAB



CANcustom1 Configuration

Network settings

Node number (1..127): 1
Node Guard Period (ms): 200
Life time Factor: 3
Boot time elapsed (ms): 0
Node heartbeat producer time (ms): 0
Node heartbeat consumer time (ms): 0
Master heartbeat consumer time (ms): 0
Identity object check ☑

PDO Tx communication settings

● USER DEFINED Mode
○ SYNC Mode
○ EVENT Mode
○ CYCLIC Mode 0 ms

PDO Rx communication settings

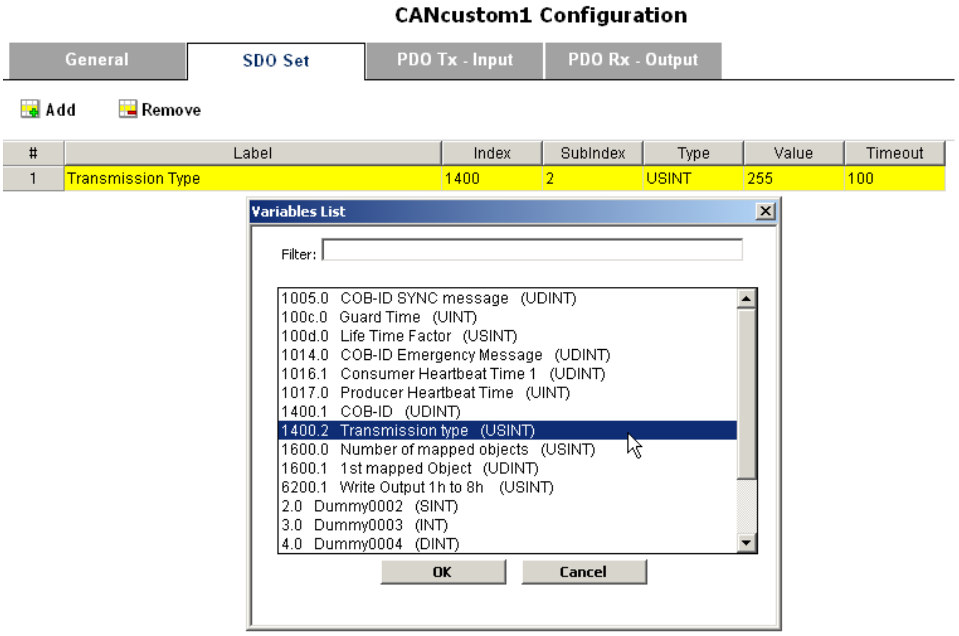
○ USER DEFINED Mode
○ SYNC Mode
● EVENT Mode

In the *General* tab you can configure (see 5.1 for further informations):

- *Node number (1..127)*.
- *Node guard period* in ms (default 200ms), *0* to disable node guard for this slave; if not zero is the interval of node guarding packets sent by the master to the slave.
- *Life time factor* (default 3x), *0* to disable node guard for this slave; if not zero, multiplied for the *Node guarding period* is the maximum amount of time the master will wait for the slave answer of the node guard.
- *Boot time elapsed*: this is the maximum amount of time in ms that the master will wait for the slave to become pre-operational at boot (default 10s), before signaling an error.
- *Node heartbeat producer time* in ms, default is *0* (heartbeat disabled); if not zero the master will enable the heartbeat error handling check for this node.
- *Node heartbeat consumer time* in ms, default is *0* (heartbeat disabled); it is the maximum amount of time the slave will wait for the heartbeat produced by the master, before timing out. This should be greater than the *Heartbeat time* of the master.

- *Master heartbeat consumer time* in ms, default is *0* (heartbeat disabled); it is the maximum amount of time the master will wait for the heartbeat sent by the slave, before timing out. This should be greater than the *Node heartbeat producer time*.
- *Identity object check*: when this option is enabled (the default) the master at boot will check the slave for his identity, verifying that the *Identity object* fields (object 0x1018) match with EDS default values (Vendor ID, Product code, Revision, Serial); if the option is not enabled, no check will be done (this is useful for example with slaves not totally CANopen-compliant or incorrect EDS files).
- *PDO Tx comm settings*: configure here the transmission mode for PDO Tx; depending on the device features (determined from EDS values), not all options may be available.
- *PDO Rx comm settings*: configure here the transmission mode for PDO Rx; depending on the device features (determined from EDS values), not all options may be available.

4.5.3.2 SDO SET TAB



#	Label	Index	SubIndex	Type	Value	Timeout
1	Transmission Type	1400	2	USINT	255	100

Filter:

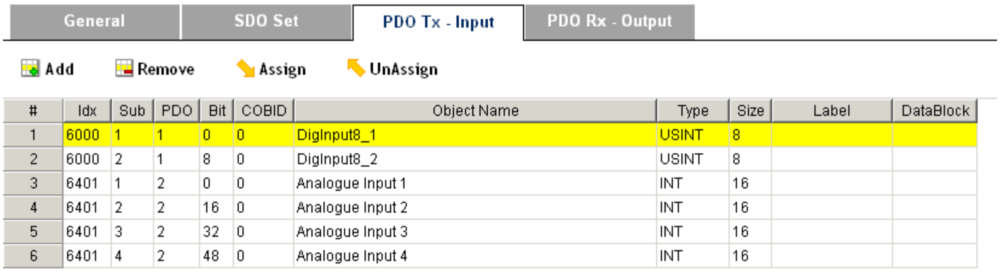
```
1005.0  COB-ID SYNC message  (UDINT)
100c.0  Guard Time  (UINT)
100d.0  Life Time Factor  (USINT)
1014.0  COB-ID Emergency Message  (UDINT)
1016.1  Consumer Heartbeat Time 1  (UDINT)
1017.0  Producer Heartbeat Time  (UINT)
1400.1  COB-ID  (UDINT)
1400.2  Transmission type  (USINT)
1600.0  Number of mapped objects  (USINT)
1600.1  1st mapped Object  (UDINT)
6200.1  Write Output 1h to 8h   (USINT)
2.0  Dummy0002  (SINT)
3.0  Dummy0003  (INT)
4.0  Dummy0004  (DINT)
```

In this page you can insert a list of objects and values to send to the slave at boot for configuration purpose, using SDO packets.

Press the *Add* button, choose the objects to send and then insert their *Value* in the grid.

Some objects are handled automatically, for example the *Transmission type* and *Event timer* are configured automatically depending on the *PDO Tx comm settings* and *PDO Rx comm settings* in the *General tab*.

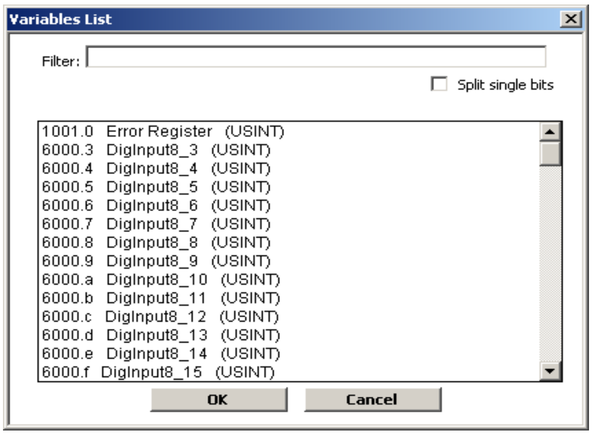
4.5.3.3 PDO TX AND PDO RX TABS



#	Idx	Sub	PDO	Bit	COBID	Object Name	Type	Size	Label	DataBlock
1	6000	1	1	0	0	DigInput8_1	USINT	8		
2	6000	2	1	8	0	DigInput8_2	USINT	8		
3	6401	1	2	0	0	Analogue Input 1	INT	16		
4	6401	2	2	16	0	Analogue Input 2	INT	16		
5	6401	3	2	32	0	Analogue Input 3	INT	16		
6	6401	4	2	48	0	Analogue Input 4	INT	16		

In the *PDO Tx - Input* tab you configure the PDOs (Process Data Object) that the slave transmits, and so the master will receive in input; in the *PDO Rx - Output* you configure the PDOs that the slave receives, and so the master will send in output.

If the *CAN custom* device was imported with the *Dynamic PDO mapping* enabled, you will be able to edit the PDO mapping by adding and removing objects and manually edit the *PDO* and *Bit* columns; otherwise, the *Add* and *Remove* buttons will not be available and you have to use the PDO configuration as-is.



```
1001.0  Error Register  (USINT)
6000.3  DigInput8_3  (USINT)
6000.4  DigInput8_4  (USINT)
6000.5  DigInput8_5  (USINT)
6000.6  DigInput8_6  (USINT)
6000.7  DigInput8_7  (USINT)
6000.8  DigInput8_8  (USINT)
6000.9  DigInput8_9  (USINT)
6000.a  DigInput8_10  (USINT)
6000.b  DigInput8_11  (USINT)
6000.c  DigInput8_12  (USINT)
6000.d  DigInput8_13  (USINT)
6000.e  DigInput8_14  (USINT)
6000.f  DigInput8_15  (USINT)
```

If you check the *Split single bits* option, the object you choose will be inserted as splitted single bits to be linked to BOOL variables (that is the default for digital I/O objects in the DS401 standard).

IMPORTANT: please note that the PDO mapping configuration you enter here is NOT sent to the device, its only purpose is to match an already configured PDO mapping on the device.

Then with the *Assign* button you can link each CAN object with the PLC variable to read (PDO Tx) or write (PDO Rx).

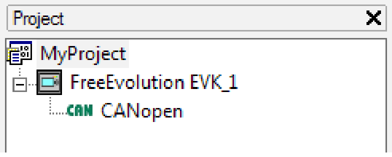
IMPORTANT: please remember to rebuild the PLC project with Application to see an up-dated list of PLC variables here.

4.6 FREE EVOLUTION EVK

FREE Evolution EVK is a keyboard with a display. It is used to show HMI Remote pages that are made available by FREE Evolution or FREE Evolution EVC devices.

EVK keyboard has not PLC, HMI and HMI Remote features and it has one on-board CANopen port.

EVK can maintain on-board no more than one HMI set of remote device pages.



4.6.1 CANOPEN

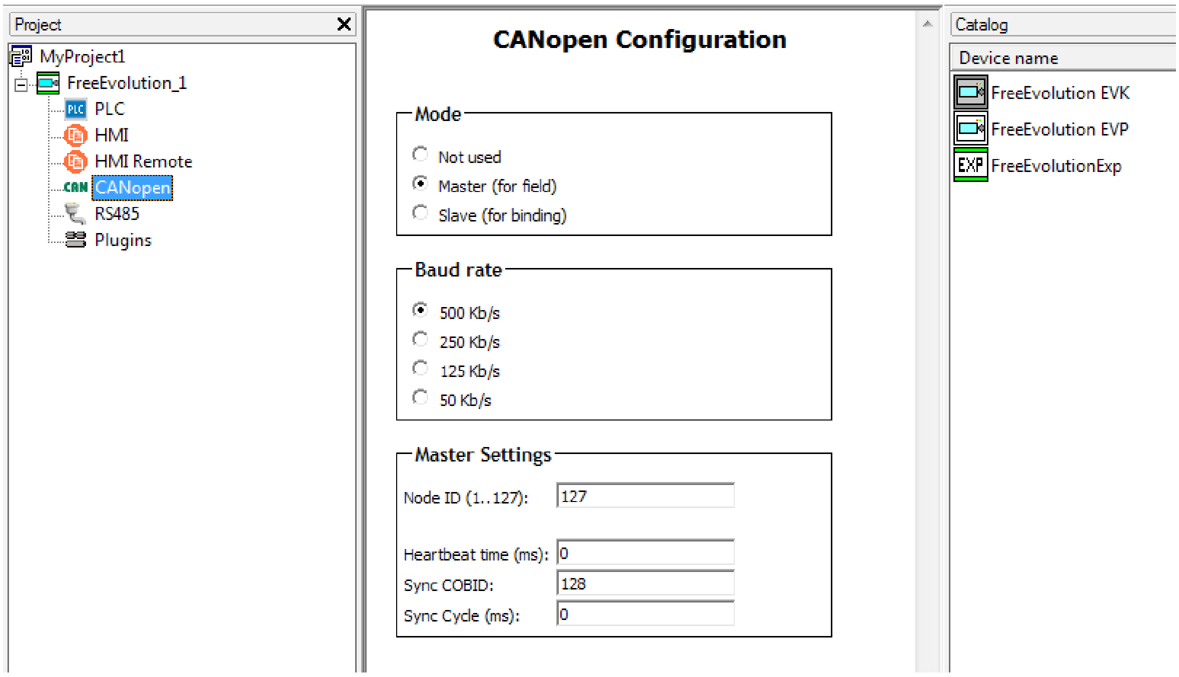
FREE Evolution EVK can be connected to FREE Evolution or to FREE Evolution EVC in field mode or in network mode.

4.6.1.1 FIELD MODE

In this connection mode FREE Evolution EVK has to be considered a slave of FREE Evolution. EVK will be able to show only the remote pages of the master device which is linked to.

To configure network in this way select FREE Evolution or FREE Evolution EVC and add it as first level node; then configure its PLC, HMI, and HMI Remote projects normally.

The project associated to FreeEvolution_1 - HMI Remote node will be shown by FREE Evolution EVK device.



CANopen Configuration

Mode
- ○ Not used
- ● Master (for field)
- ○ Slave (for binding)

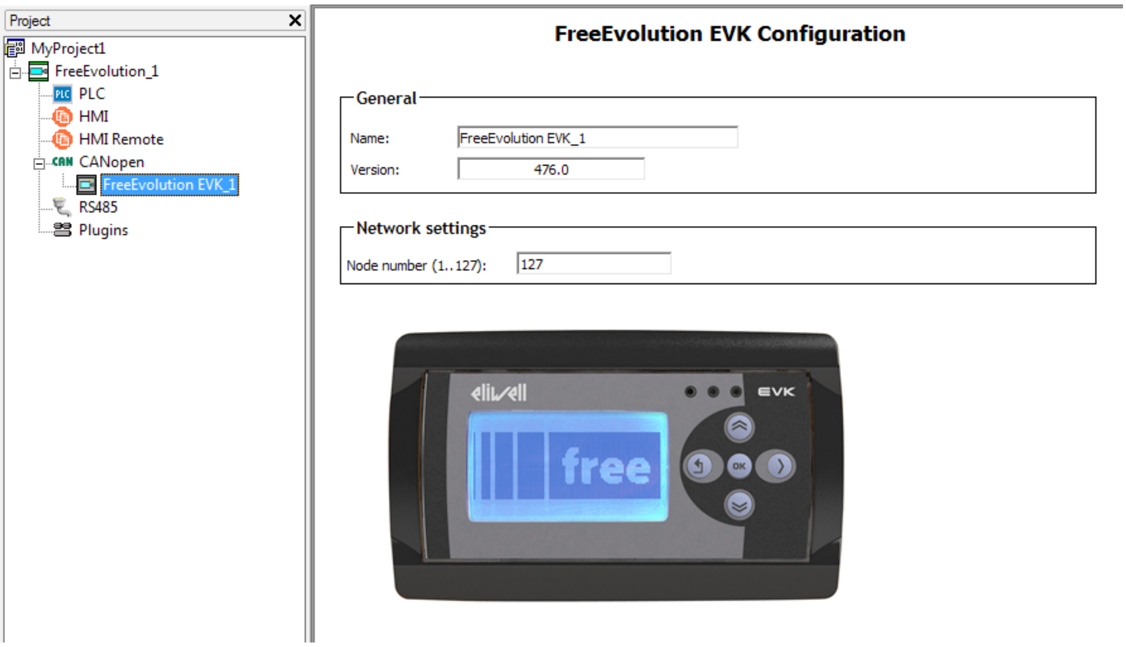
Baud rate
- ● 500 Kb/s
- ○ 250 Kb/s
- ○ 125 Kb/s
- ○ 50 Kb/s

Master Settings

Node ID (1..127): 127
Heartbeat time (ms): 0
Sync COBID: 128
Sync Cycle (ms): 0

Click on *CANopen* and select *Master* (for field) option in *Mode* tab and configure CANopen settings. Then FREE Evolution EVK device can be selected from *Catalog*, dragged and dropped over CANopen node.

Select FreeEvolution EVK_1 device child node and adjust network settings.



FreeEvolution EVK Configuration

General

Name: FreeEvolution EVK_1
Version: 476.0

Network settings

Node number (1..127): 127

As resulting output of the compiling process we see the output line:

```
FreeEvolution_1: added field CAN keyboard 'FreeEvolution EVK_1' (with
virtual master nodeID 124)
```

FreeEvolution EVK_1 device communicates with FreeEvolution_1 using this CAN nodeID. This node ID will be used for navigating remote pages.

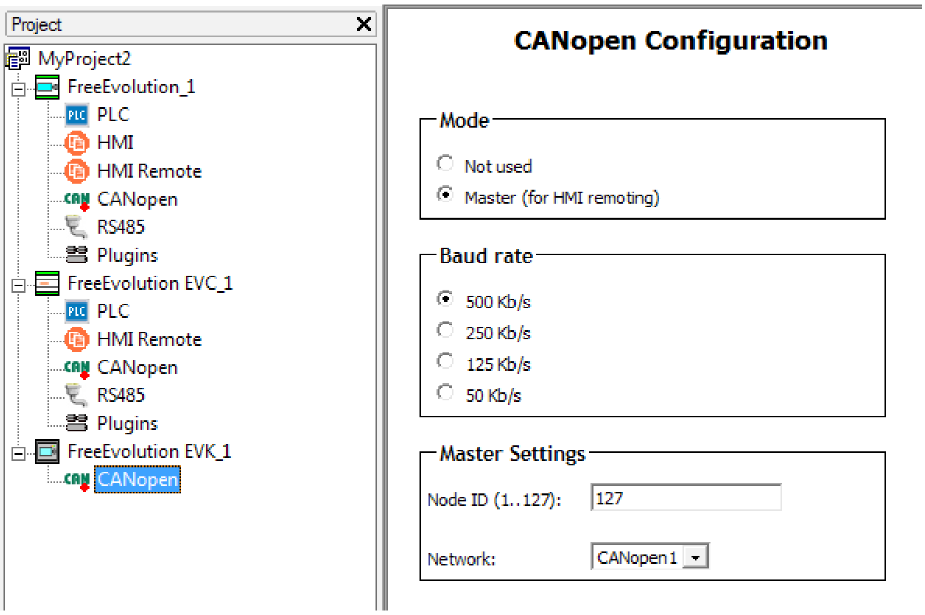
4.6.1.2 NETWORK MODE

In this connection mode free Evolution EVK can be linked to one of the remote devices that are available on the network to navigate HMI Remote pages provided by other devices.

Using Connection it is possible to do so indicating one of the available HMI Remote device of the network. Let's see how with an example.

We have a CANOpen network with FreeEvolution_1 and FreeEvolution EVC_1, then add as first level node FreeEvolution EVK_1 to the network taking it from `Catalog` panel.

Click on `CANOpen node` of FreeEvolution EVK_1 and select `Master` (for HMI remoting) node, assign univoque Node ID and select network `CANOpen1`.



CANopen Configuration

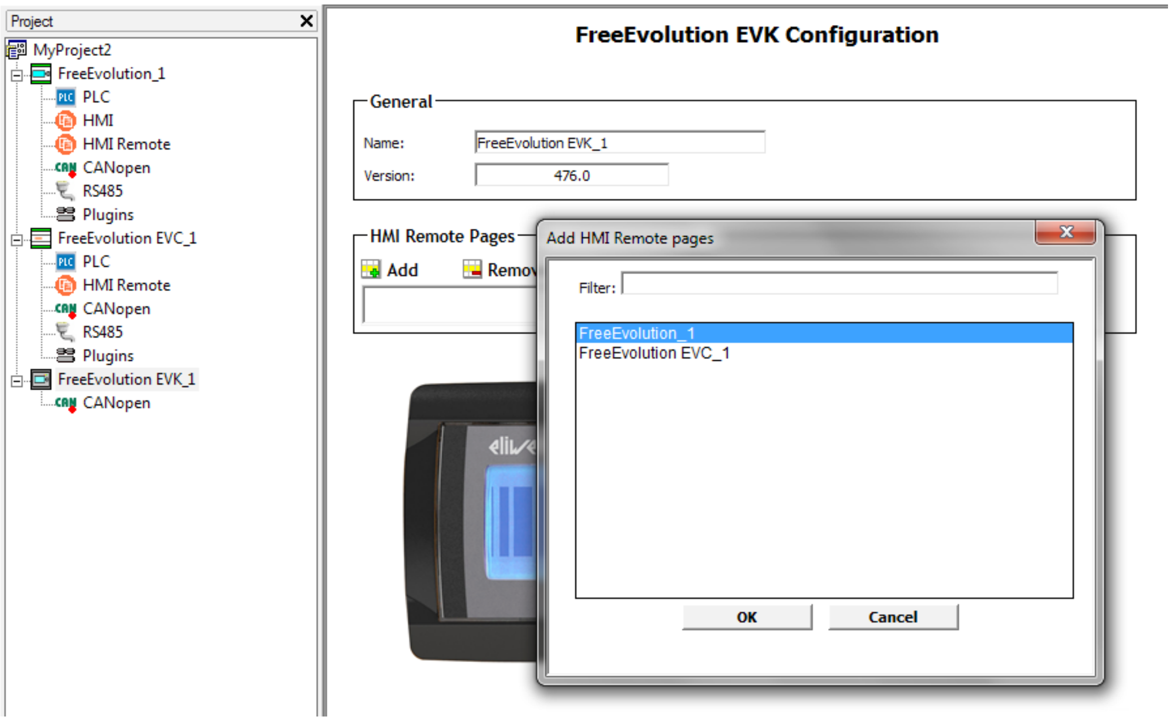
Mode
○ Not used
● Master (for HMI remoting)

Baud rate
● 500 Kb/s
○ 250 Kb/s
○ 125 Kb/s
○ 50 Kb/s

Master Settings
Node ID (1..127): 127
Network: CANopen1

Once linked to the CANopen1 network it is possible to select the HMI remote pages to navigate with FreeEvolution EVK_1.

Click on the device node then click on `Add`. Window `Add HMI Remote pages` will be shown. It is possible to select to navigate pages of FreeEvolution_1 or FreeEvolution EVC_1 because they are on the same network of the keyboard and provide HMI Remote pages.



Name: FreeEvolution EVK_1

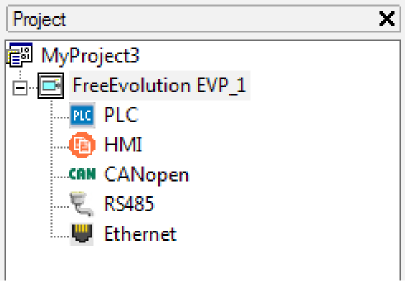
Version: 476.0

FreeEvolution_1
FreeEvolution EVC_1

Click on *FreeEvolution_1*, then click on *OK*. Selected device will be added to HMI Remote Pages. It is not possible to navigate more than one remote device at a time.

4.7 FREE EVOLUTION EVP

FREE Evolution EVP is an advanced keyboard with display that can be used to navigate HMI Remote pages and offers more connectivity (CANopen, RS485, Ethernet) than FREE Evolution EVK. It can also run PLC and local HMI pages and it is also provided with probes.



4.7.1 PLC

FREE Evolution EVP can run PLC. This configuration step can be done in the same way of FREE Evolution and is fully described in section 4.1.1 - PLC.

4.7.2 HMI

FREE Evolution EVP can run local HMI project with its own pages. This configuration step can be done in the same way of FREE Evolution and is fully described in section 4.1.2 - HMI.

4.7.3 PROVIDING HMI PAGES

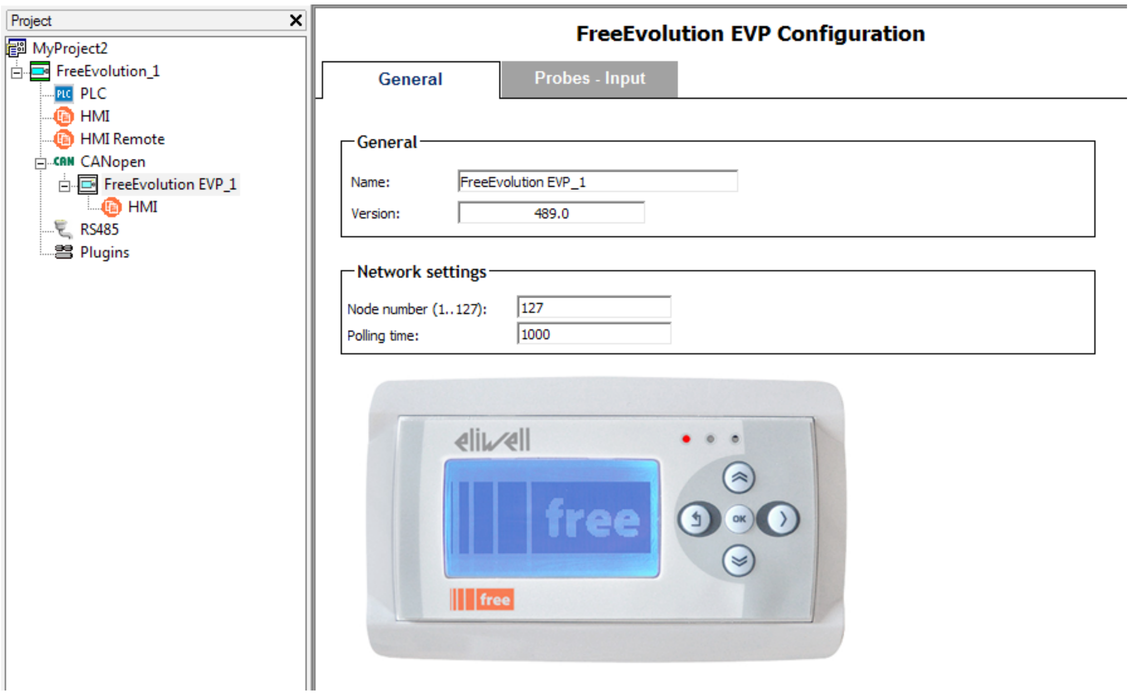
This feature is not supported by FREE Evolution EVP. No linked device can upload HMI pages from FREE Evolution EVP device.

4.7.4 CANOPEN

FREE Evolution EVP can be connected using CANopen in field mode or in network mode.

4.7.4.1 FIELD MODE

To connect FREE Evolution EVP in this mode select *FREE Evolution* or FREE *Evolution EVC* CANopen node and select the option *Master* (for field) then take FREE Evolution EVP device from *Catalog* tab and drop it over CANopen node.



General

Name: FreeEvolution EVP_1

Version: 489.0

Network settings

Node number (1..127): 127

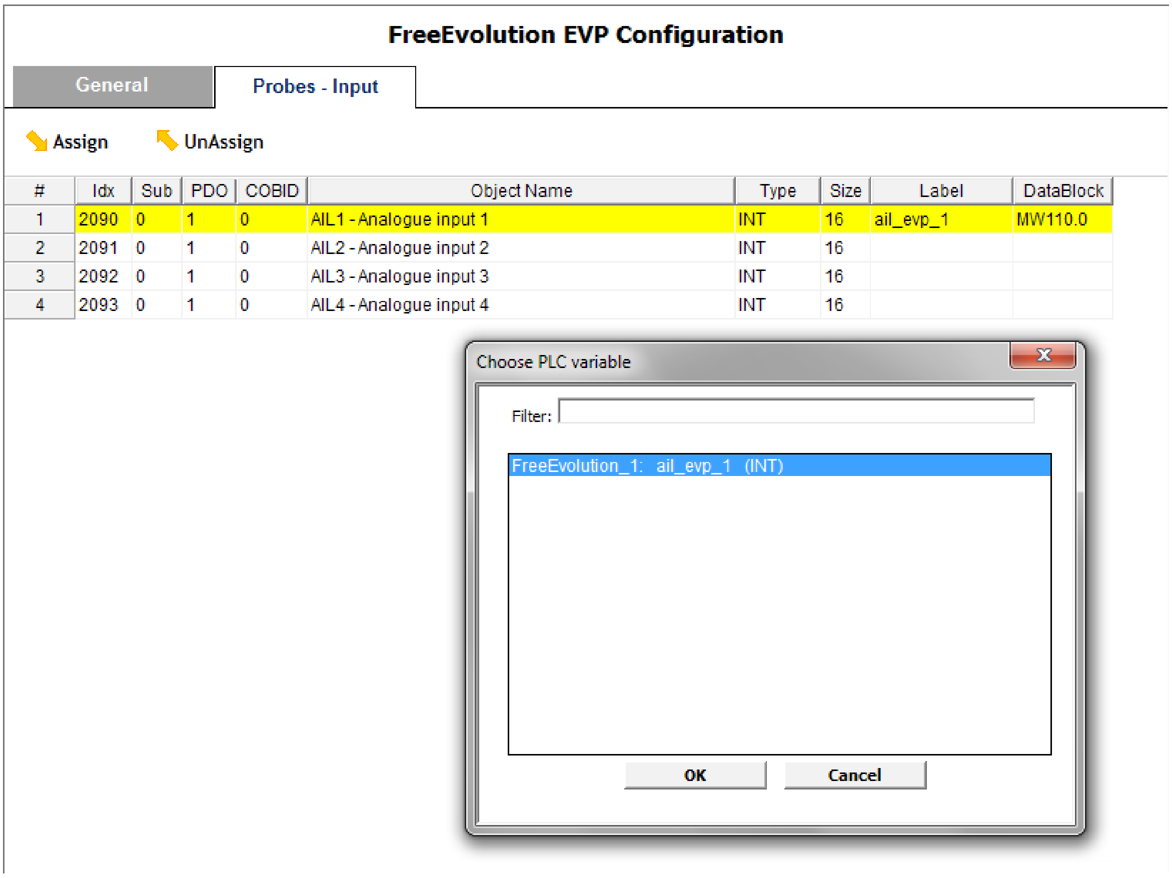
Polling time: 1000

Select *FreeEvolution EVP_1* child node and configure *Network* settings.

Probes

FreeEvolution_1 can access on board FreeEvolution EVP_1 on-board probes. To do so select *Probes-Input* tab then it is possible to map a FreeEvolution_1 parameter to let it obtain the value of an on-board free Evolution probe.

Choose one of the probe and click on *Assign* button. Take one of the FreeEvolution_1 INT parameter and click *OK* button.



FreeEvolution EVP Configuration

#	Idx	Sub	PDO	COBID	Object Name	Type	Size	Label	DataBlock
1	2090	0	1	0	AIL1 - Analogue input 1	INT	16	ail_evp_1	MW110.0
2	2091	0	1	0	AIL2 - Analogue input 2	INT	16		
3	2092	0	1	0	AIL3 - Analogue input 3	INT	16		
4	2093	0	1	0	AIL4 - Analogue input 4	INT	16		

Choose PLC variable

Filter:

FreeEvolution_1: ail_evp_1 (INT)

HMI

It is possible to associate to a FREE Evolution EVP (configured as CANopen field slave) an HMI project with local pages. FREE Evolution EVP would be able to show its own target variables and parameters of the master CANopen which belongs to.

4.7.4.2 NETWORK MODE

In this connection mode FREE Evolution EVP can be linked to one of the remote devices that are available on the network to navigate HMI Remote pages provided by other devices.

Using Connection it is possible to do so by indicating one of the available HMI Remote device of the network. Let's see how with an example.

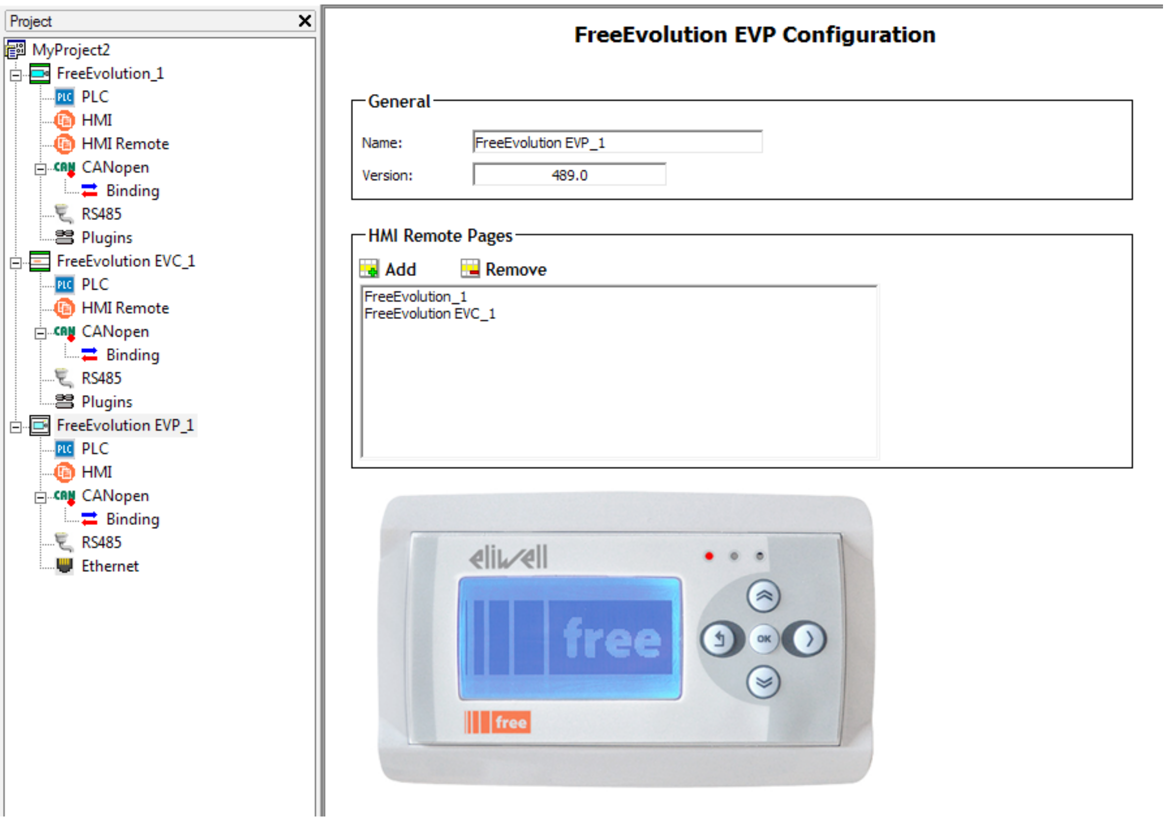
We have a CANopen network with FreeEvolution_1 and FreeEvolution EVC_1 then add as first level node *FreeEvolution EVP_1* to the network taking it from *Catalog* panel.

Click on *CANOpen* node of FreeEvolution EVP_1 and select *Master* (for HMI remoting and binding) node, assign univoque *Node ID* and select network *CANOpen1*.

Binding of variables between FreeEvolution EVP_1 and FreeEvolution EVC_1 and FreeEvolution_1 is allowed in a network of this type (see 4.1.3.2 for more details).

HMI Remote pages

In CANopen network mode it is possible to configure FREE Evolution EVP in order to be linked to 0 to 10 remote devices that can provide HMI Remote pages to the keyboard.



Project

- MyProject2
  - FreeEvolution_1
    - PLC
    - HMI
    - HMI Remote
    - CANopen
      - Binding
    - RS485
    - Plugins
  - FreeEvolution EVC_1
    - PLC
    - HMI Remote
    - CANopen
      - Binding
    - RS485
    - Plugins
  - FreeEvolution EVP_1
    - PLC
    - HMI
    - CANopen
      - Binding
    - RS485
    - Ethernet

FreeEvolution EVP Configuration

General

Name: FreeEvolution EVP_1

Version: 489.0

HMI Remote Pages

Add Remove

FreeEvolution_1
FreeEvolution EVC_1

To add HMI Remote pages select *FreeEvolution EVP_1* node, then press *Add* on the *HMI Remote pages* box thus all available devices will be shown and the user can select the pages to navigate.

4.7.5 RS485

The usage of this communication feature is the same of FREE Evolution (see 4.1.4 - RS485 paragraph).

4.7.6 ETHERNET

FREE Evolution EVP is provided with on-board Ethernet. Ethernet configuration and features for this kind of device is similar to the configuration of the Ethernet plugin of FREE Evolution (see 4.1.5 – Ethernet).

4.8 FREE EVOLUTION EXP

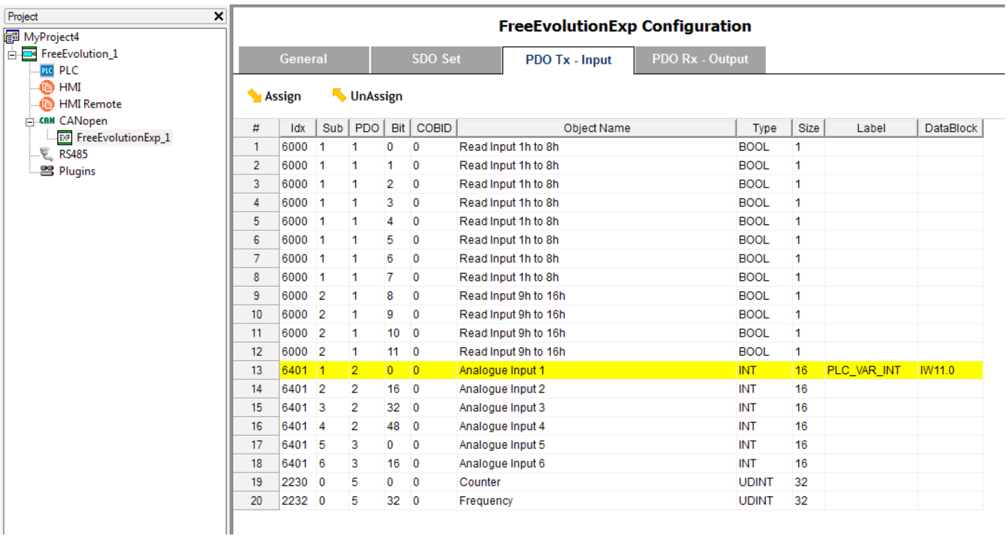
FREE Evolution EXP is a device that can be linked in a CANopen field or Modbus RTU field network whose master can be a FREE Evolution, a FREE Evolution EVC or a FREE Evolution EVP device.

FREE Evolution EXP main feature is to provide a lot of I/O signal to its field master device. I/O signals mapping can be configured by using Connection.

4.8.1 USING FREE EVOLUTION EXP AS CANOPEN FIELD SLAVE

In this configuration sample we want to use FREE Evolution EXP as expansion of a FREE Evolution device. The same can be done for FREE Evolution EVC and FREE Evolution EVP.

Configure FREE Evolution CANopen in *Master* (for field) mode. From the *Catalog* panel it is possible to select *FreeEvolutionExp* node and drop it on the CANopen node.



#	Idx	Sub	PDO	Bit	COBID	Object Name	Type	Size	Label	DataBlock
1	6000	1	1	0	0	Read Input 1h to 8h	BOOL	1		
2	6000	1	1	1	0	Read Input 1h to 8h	BOOL	1		
3	6000	1	1	2	0	Read Input 1h to 8h	BOOL	1		
4	6000	1	1	3	0	Read Input 1h to 8h	BOOL	1		
5	6000	1	1	4	0	Read Input 1h to 8h	BOOL	1		
6	6000	1	1	5	0	Read Input 1h to 8h	BOOL	1		
7	6000	1	1	6	0	Read Input 1h to 8h	BOOL	1		
8	6000	1	1	7	0	Read Input 1h to 8h	BOOL	1		
9	6000	2	1	8	0	Read Input 9h to 16h	BOOL	1		
10	6000	2	1	9	0	Read Input 9h to 16h	BOOL	1		
11	6000	2	1	10	0	Read Input 9h to 16h	BOOL	1		
12	6000	2	1	11	0	Read Input 9h to 16h	BOOL	1		
13	6401	1	2	0	0	Analogue Input 1	INT	16	PLC_VAR_INT	IW11.0
14	6401	2	2	16	0	Analogue Input 2	INT	16		
15	6401	3	2	32	0	Analogue Input 3	INT	16		
16	6401	4	2	48	0	Analogue Input 4	INT	16		
17	6401	5	3	0	0	Analogue Input 5	INT	16		
18	6401	6	3	16	0	Analogue Input 6	INT	16		
19	2230	0	5	0	0	Counter	UDINT	32		
20	2232	0	5	32	0	Frequency	UDINT	32		

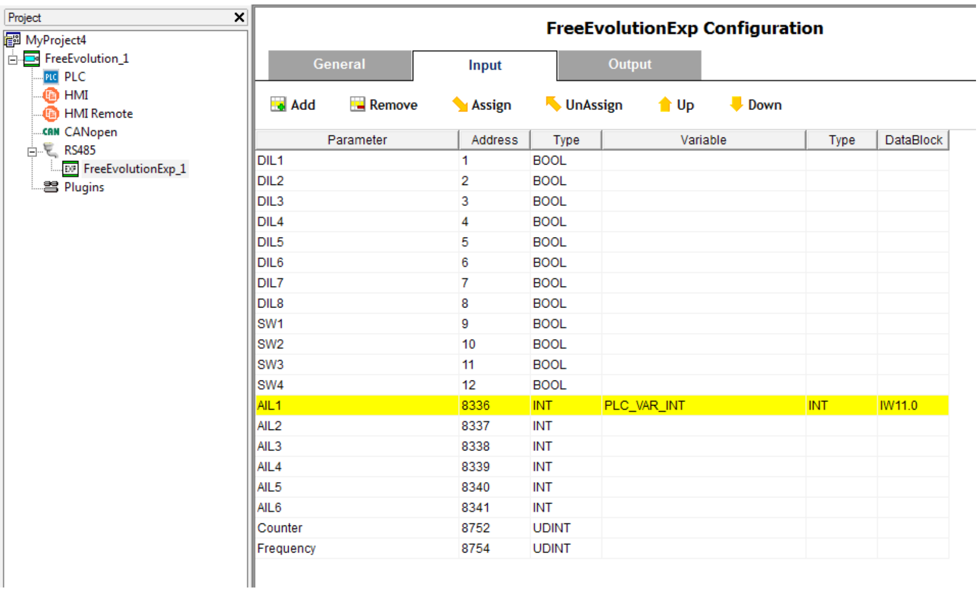
FreeEvolution EXP configuration is quite similar to CAN Custom configuration (see 4.5.3 - Using a CAN custom device) with dynamic PDO mapping feature disabled. Available Input/Output objects that can be mapped on FREE Evolution PLC variables via PDO are listed in *PDO TX-Input* and *PDO RX-Output*.

Connection knows the FREE Evolution EXP dictionary. Each object can be here linked to FreeEvolution_1 PLC variable as it has been done in the above figure for *Analogue Input 1* signal.

4.8.2 USING FREE EVOLUTION EXP AS RS485 FIELD SLAVE

In this configuration sample we want to use FREE Evolution EXP as expansion of a FREE Evolution device. The same can be done for FREE Evolution EVC and FREE Evolution EVP.

Configure FREE Evolution RS485 in *Modbus Master* (for field) mode. From the *Catalog* panel it is possible to select *FreeEvolutionExp* node and drop it on the *RS485* node.



Parameter	Address	Type	Variable	Type	DataBlock
DIL1	1	BOOL			
DIL2	2	BOOL			
DIL3	3	BOOL			
DIL4	4	BOOL			
DIL5	5	BOOL			
DIL6	6	BOOL			
DIL7	7	BOOL			
DIL8	8	BOOL			
SW1	9	BOOL			
SW2	10	BOOL			
SW3	11	BOOL			
SW4	12	BOOL			
AIL1	8336	INT	PLC_VAR_INT	INT	IW11.0
AIL2	8337	INT			
AIL3	8338	INT			
AIL4	8339	INT			
AIL5	8340	INT			
AIL6	8341	INT			
Counter	8752	UDINT			
Frequency	8754	UDINT			

FreeEvolution EXP_1 configuration is quite similar to a Modbus Custom device configuration (see 4.4.4 - Using a Modbus custom device) in which it is possible to assign available FREE Evolution EXP dictionary I/O objects to FreeEvolution_1 PLC variables.

Connection knows the FREE Evolution EXP dictionary. *Input* and *Output* objects can be added, removed, assigned, unassigned or changed in position. Only assigned objects will be requested by FreeEvolution_1 device.

4.9 VIRTUAL CHANNELS ASSIGNMENT CRITERIA

This paragraph concerns the criteria used by Connection to assign virtual node IDs due to the network configuration.

4.9.1 CANOPEN NETWORK - VIRTUAL SDO SERVERS

When CANopen is in use on a FREE Evolution or FREE Evolution EVC device in *slave* mode (network for binding) three SDO servers are activated on it.

First SDO server is used to process requests that arrives to its physical node ID (the ID assigned by user in the configuration box). Supervisor PC should be connected using this node ID. CANopen physical node ID *addr* must be chosen in a range between 1 to 42.

Two other virtual SDO servers are opened on this device and are dedicated to the communication with keyboards (max 2 for each CANopen network). So the device is able to process requests addressed to these node IDs.

Virtual SDO servers node IDs are calculated with this criteria:

```
  ch_1 = 124 - 2 * ( addr - 1 )

  ch_2 = 123 - 2 * ( addr - 1 )
```

The first keyboard on the network communicates to the destination FREE Evolution device using `ch_1`, channel `ch_2` is dedicated to the second.

Example:

```
addr = 1  ->  ch_1 = 124,  ch_2 = 123
addr = 2  ->  ch_1 = 122,  ch_2 = 121
```

4.9.2 ETHERNET - TCP SLAVE CHANNELS

If Ethernet network communication is enabled on a FREE Evolution or FREE Evolution EVC device two TCP slave channels are always opened to support the communication with keyboards.

4.9.3 CANOPEN FIELD - VIRTUAL MASTER CHANNELS

When CANopen is in use on a FREE Evolution or FREE Evolution EVC device in *master* mode (field) three master channels are opened.

First master channel is used to process requests that arrive to its physical node ID (the ID assigned by user in the configuration box). Supervisor PC should be connected using this node ID. CANopen physical node ID `addr` must be chosen in a range between 1 to 122 or 125.

Two other virtual master channels are opened on this device and are dedicated to the communication with keyboards (max 2 for each CANopen network).

Virtual master node IDs have fixed values :

```
ch_1 = 123
ch_2 = 124
```

5. TECHNICAL REFERENCE

5.1 CANOPEN PROTOCOL

5.1.1 OVERVIEW

CANopen realizes a communication model using the serial bus network Controller Area Network (CAN).

Developed originally for passenger cars, the CAN two-wire bus system is already in use in over one million industrial control devices, sensors and actuators.

CiA (CANopen in Automation) maintains the CANopen specifications, including device profiles for I/O modules (CiA DS-401), for electric drive systems (CiA DSP-402) and many more. The process of defining new profiles is continually performed. An independent test and certification process is available at CiA.

A number of CANopen implementations (OEM code) and many CANopen products are already available. CiA regularly publishes an up-to-date catalog of CANopen products and of certified ones.

5.1.2 PHYSICAL STRUCTURE OF A CANOPEN NETWORK

The underlying CAN architecture defines the basic physical structure of the CANopen network. Therefore, a line (bus) topology is used; to avoid reflections of the signals, both ends of the network must be terminated. In addition, the maximum permissible branch line lengths for connection of the individual network nodes must be observed.

Additionally, for CANopen, two additional conditions must be fulfilled:

- all nodes must be configured to the same bit rate and
- no node-ID may exist twice.

Unfortunately there are no mechanisms automatically ensuring these conditions. The system integrator has to check the bit rate and node-ID of every single network node when wiring a network and adjust if necessary.

5.1.3 COB AND COB-ID

CANbus, the physical layer of CANopen, can transmit short packages of data (called COB, Communication Object), that have a 11-bit ID or 29-bit ID (in version CAN 2.0 B); this ID of a CAN-frame is known as Communication Object Identifier, or COB-ID. In case of a transmission collision, the bus arbitration used in the CANbus allows the frame with the smallest ID to be transmitted first and without a delay. Thus giving a low code number for time critical functions ensures the lowest possible delay.

5.1.4 THE OBJECT DICTIONARY

All device parameters are stored in an object dictionary. This object dictionary contains the description, data type and structure of the parameters as well as the address from others point of view. The address is being composed of a 16 bit index and a 8 bit sub-index; the sub-index refers to the elements of complex data types, like arrays and records.

There are a range of mandatory entries in the dictionary which ensures that all CANopen devices of a particular type show the same behavior. The object dictionary concept caters for optional device features which means a manufacturer does not have to provide certain extended functionality on his device, but if he wishes to do so he has to do it in a pre-defined fashion. Additionally, there is sufficient address space for truly manufacturer specific functionality.

5.1.5 THE SERVICE DATA OBJECTS (SDO)

Service Data Messages, in CANopen called Service Data Objects (SDO), are used for read and write access to all entries of the object dictionary of a device. Main usage of this type of messages is the device configuration; SDOs are typically transmitted asynchronously. The requirements towards transmission speed are not as high as for PDOs; the SDO message contains information to address data in the device object dictionary and the data itself.

5.1.6 THE PROCESS DATA OBJECTS (PDO)

Process Data Messages, in CANopen called Process Data Objects (PDO), are used to perform the real-time data transfer between different automation units. PDOs have to be transmitted quickly, without any protocol overhead and within a predefined structure.

The contents of the PDO is encoded in the PDO mapping entries. A PDO can contain up to 8 bytes or 64 single data elements from the object dictionary (in the case of 64, that are bit data); the data are described via its index, sub-index and length. The mapping parameter of a PDO resides also in the object dictionary.

The mapping for the PDO can be static or changeable. If the mapping can be changed, it is called dynamic PDO mapping; changing of mapping can be done in the state pre-operational (default) or operational.

5.1.7 PDO TRANSMISSION MODES

For the PDOs different transmission modes are distinguished:

- SYNC: PDO are transmitted according to the SYNC clock transmitted by the master.
- EVENT: PDO are transmitted when the value changes (asynchronous).
- CYCLIC: PDO transmission is periodic and timer-based.
- RTR: PDO are transmitted only on master request.

The communication parameters of a PDO reside in the object dictionary. The indices for PDOs are built like follow:

- PDO Tx: 0x1800 + PDO number.
- PDO Rx: 0x1400 + PDO number.

The range of the PDO numbers is 1..512. that means up to 512 receive PDOs (RPDO) and up to 512 transmit PDOs (TPDO) are possible for a device.

The communication parameter of PDOs are described with a structure: only sub-index 1 and 2 are mandatory.

Subindex 1 describes the used COB-ID of the PDO: a PDO communication channel between two devices is created by setting the TPDO COB-ID of the first device to the RPDO COB-ID of the second device. For PDOs a 1:1 and a 1:n communication is possible: that means there is always only one transmitter, but an unlimited number of receivers.

The transmission type (sub-index 2) describes the kind of transmission; transmission type 1 means PDO will be triggered with each SYNC Object. If this entry has the value 240, the PDO will be sent/received with each 240th SYNC. If the entry is 255, the transmission is EVENT or CYCLIC, depending on the event timer (see below).

The optional entry inhibit time (sub-index 3) defines a minimum time period between two PDO transmissions. This feature ensures that messages with lower priorities than the actual PDO can be transmitted in the case of continuous transmission of the actual PDO.

The optional entry event timer (sub-index 5) is only relevant for asynchronous Transmit PDOs: if this value is greater then zero, indicates the time to elapse for the CYCLIC; otherwise means EVENT (on variation).

5.1.8 THE EMERGENCY OBJECT

The Emergency Message (EMCY) is a service which signs internal fatal device errors.

The EMCY is transmitted with highest priority; CANopen defines EMCY-Server and EMCY-Clients, the server transmits EMCYs and the clients receive them.

The EMCY telegram consists of 8 bytes: it contains an emergency error code, the contents of object and 5 byte of manufacturer specific error code.

5.1.9 SYNC OBJECT AND TIME STAMP OBJECT

The SYNC Object is a network wide system clock. It is the trigger for synchronous message transmission; the SYNC has a very high priority and contains no data in order to guarantee a minimum of jitter. The SYNC COB-ID is by default 128, but can be configured.

The Time Stamp Object provides a common time reference; it is transmitted with high priority.

5.1.10 ERROR CONTROL: NODE GUARDING

The Node Guarding is the periodical monitoring of certain network nodes; each node can be checked by the master with a certain period called "Node guard period". If the node does not answer after a time calculated as the guard period x "Life time factor", the connection should be considered lost.

This feature is enabled for a slave when both parameters are not zero; please note that when it is enabled it has a big impact on network load.

5.1.11 ERROR CONTROL: HEARTBEAT

The Heartbeat is an error control service without need for remote frames: the Heartbeat producer transmits periodically a heartbeat message; one or more heartbeat consumer receive this message and monitor this indication.

Each heartbeat producer can use a certain period (heartbeat producer time); the heartbeat starts immediately if the heartbeat producer time is zero.

The heartbeat consumer has to monitor the heartbeat producer; it has an entry for each heartbeat producer in its own object dictionary. The heartbeat consumer time can be different for each heartbeat producer but should be greater than the heartbeat producer time.

Heartbeat has a big impact on network load, but in practice the half of the load of the node guarding.

5.1.12 THE NETWORK BEHAVIOR

Devices have four operative states: the `initialization`, the `pre-operational`, the `stopped` and the `operational` one; the difference between master and slave devices is the initiation of the state transitions.

The master controls the state transitions of each device in the network: after power-on a device goes in the `initialization`, and then in the `pre-operational` automatically; in this state reading and writing to its object dictionary via the service data object (SDO) is possible. Therefore the device can now be configured: this means setting of objects or changing of default values in the object dictionary like preparing PDO transmission.

Afterwards the device can be switched into the `operational`" state via the command `Start Remote Node` in order to start PDO communication. PDO communication can be stopped by the network master by simply switching the remote node back to pre-operational by using the `Enter Pre-Operational State` command.

Via the *Stop Remote Node* command the master can force the slave(s) to the *stopped* state. In this state no services besides network and error control mechanism are available.

The command *Reset Communication* resets the communication on the node: all communication parameters will be set to their defaults.

The application will be reset by *Reset Node* command, that resets all application parameter and then calls *Reset Communication* command.

5.1.13 THE BOOT-UP MESSAGE

After a CANopen node has finished its own initialization and entered in the node state *pre-operational* it has to send the Boot-up Protocol Message; this message indicated that the slave is ready for work (e.g. configuration).

The master can wait for this message up to *Boot time elapsed* ms.

5.1.14 THE CANOPEN DEVICE PROFILES

A device profile defines a standard kind of device: for these standard devices a basic functionality has been specified, that every device has to implement. The CANopen Device Profiles ensure a minimum of identical behavior for a kind of devices, and this guarantees an high degree of interoperability and vendor independence.

Each device has to fulfill the requirements on the behavior; furthermore it has to support all mandatory objects: these objects are parameter and data for the device.

Additionally the manufacturer can decide about supported optional objects; all parameters and data, which are not covered by the standardized device profiles can be realized as manufacturer specific objects.

For example, two of the most commonly used Device Profiles are DS401 (Generic I/O Modules) and DS402 (Drives and Motion Control).

5.2 MODBUS PROTOCOL

5.2.1 OVERVIEW

Modbus is a serial communication protocol. In simple terms, it is a method used for transmitting information over serial lines between electronic devices. The device requesting the information is called the Modbus Master and the devices supplying information are Modbus Slaves. In a standard Modbus network, there is one Master and up to 247 Slaves, each with a unique Slave Address from 1 to 247; the Master can also write information to the Slaves.

Address *0* is used as broadcast address.

5.2.2 DATA TYPES

Information is stored in the *Slave* device in four different types: two types are on/off discrete values (coils) and two are numerical values (registers).

- Discrete Input Contacts (read only), 1-bit.
- Discrete Output Coils (read/write), 1-bit.
- Analog Input Registers (read only), 16-bit.
- Analog Output Holding Registers (read/write), 16-bit.

To handle more complex data types (like 32-bit integers or floating point) you have to use two or more following registers and read or write them together.

5.2.3 FUNCTION CODES

The Modbus protocol specifies different “function codes” for each Modbus message:

- 01 (0x01): Read Discrete Output Coils.
- 05 (0x05): Write single Discrete Output Coil.
- 15 (0x0F): Write multiple Discrete Output Coils.
- 02 (0x02): Read Discrete Input Contacts.
- 04 (0x04): Read Analog Input Registers.
- 03 (0x03): Read Analog Output Holding Registers.
- 06 (0x06): Write single Analog Output Holding Register.
- 16 (0x10): Write multiple Analog Output Holding Registers.

5.2.4 ERROR DETECTION AND CRC

CRC stands for Cyclic Redundancy check: it is two bytes added to the end of every Modbus message for error detection. Every byte in the message is used to calculate the CRC. The receiving device also calculates the CRC and compares it to the CRC from the sending device: if even one bit in the message is received incorrectly, the CRCs will be different and an error will result.

5.2.5 PROTOCOL VERSIONS

Versions of the Modbus protocol exist for serial port and for Ethernet and other networks that support the Internet protocol suite. There are many variants of Modbus protocols:

- Modbus RTU: This is used in serial communication (RS232 or RS485) and makes use of a compact, binary representation of the data for protocol communication. The RTU format follows the commands/data with a cyclic redundancy check checksum as an error check mechanism to ensure the reliability of data. Modbus RTU is the most common implementation available for Modbus. A Modbus RTU message must be transmitted continuously without inter-character hesitations. Modbus messages are framed (separated) by idle (silent) periods.
- Modbus ASCII: This is used in serial communication and makes use of ASCII characters for protocol communication. The ASCII format uses a longitudinal redundancy check checksum. Modbus ASCII messages are framed by leading colon (‘:’) and trailing newline (CR/LF).
- Modbus TCP: This is a Modbus variant used for communications over TCP/IP networks. It does not require a checksum calculation as lower layer takes care of the same.