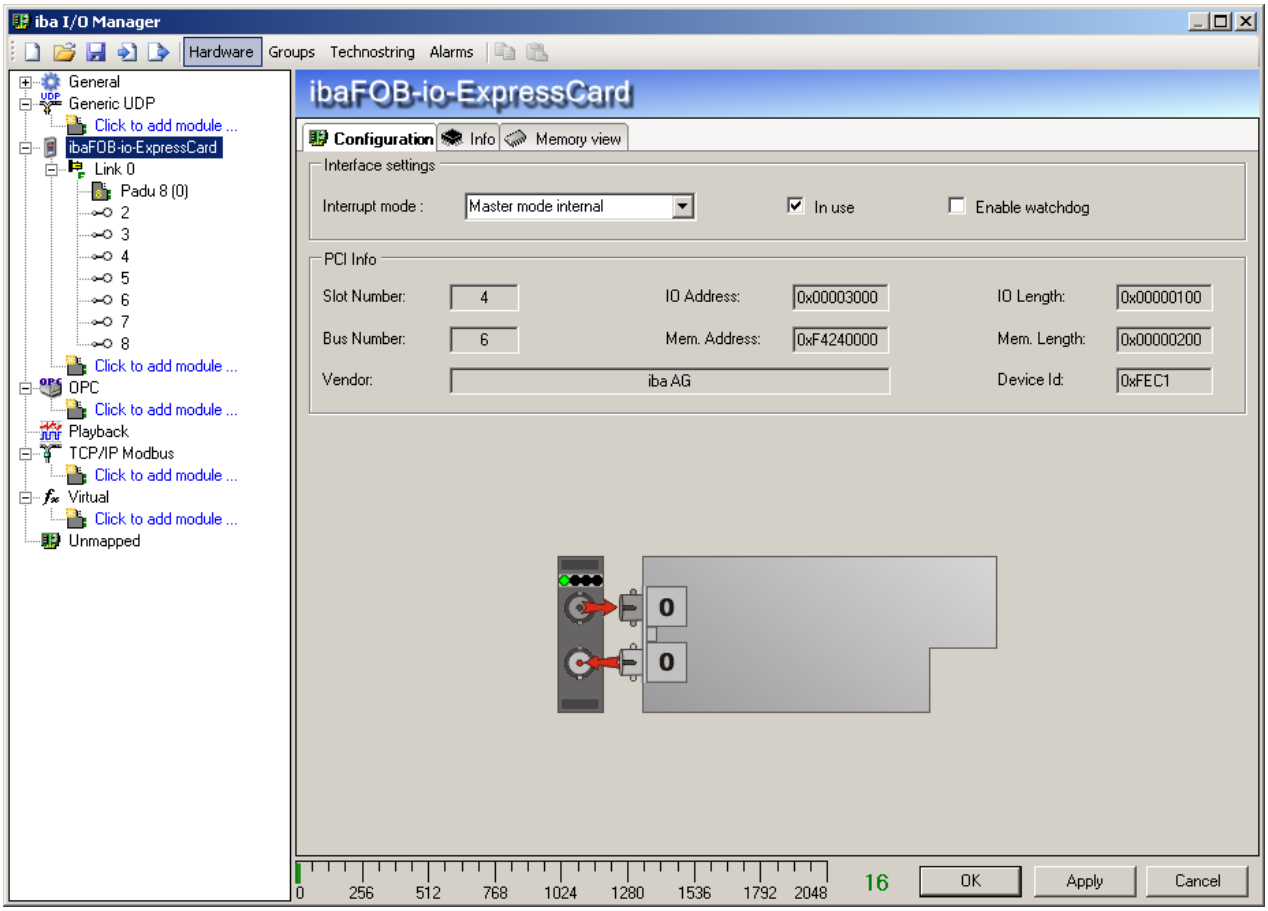
New features in ibaPDA v6.24.0

1 Windows 7 (x86) support

The 32 bit version of windows 7 is supported. In ibaPDA there is nothing really different compared to windows XP and windows Vista. The 64 bit versions of all operating systems are not supported. We have also stopped the support of windows 2000.

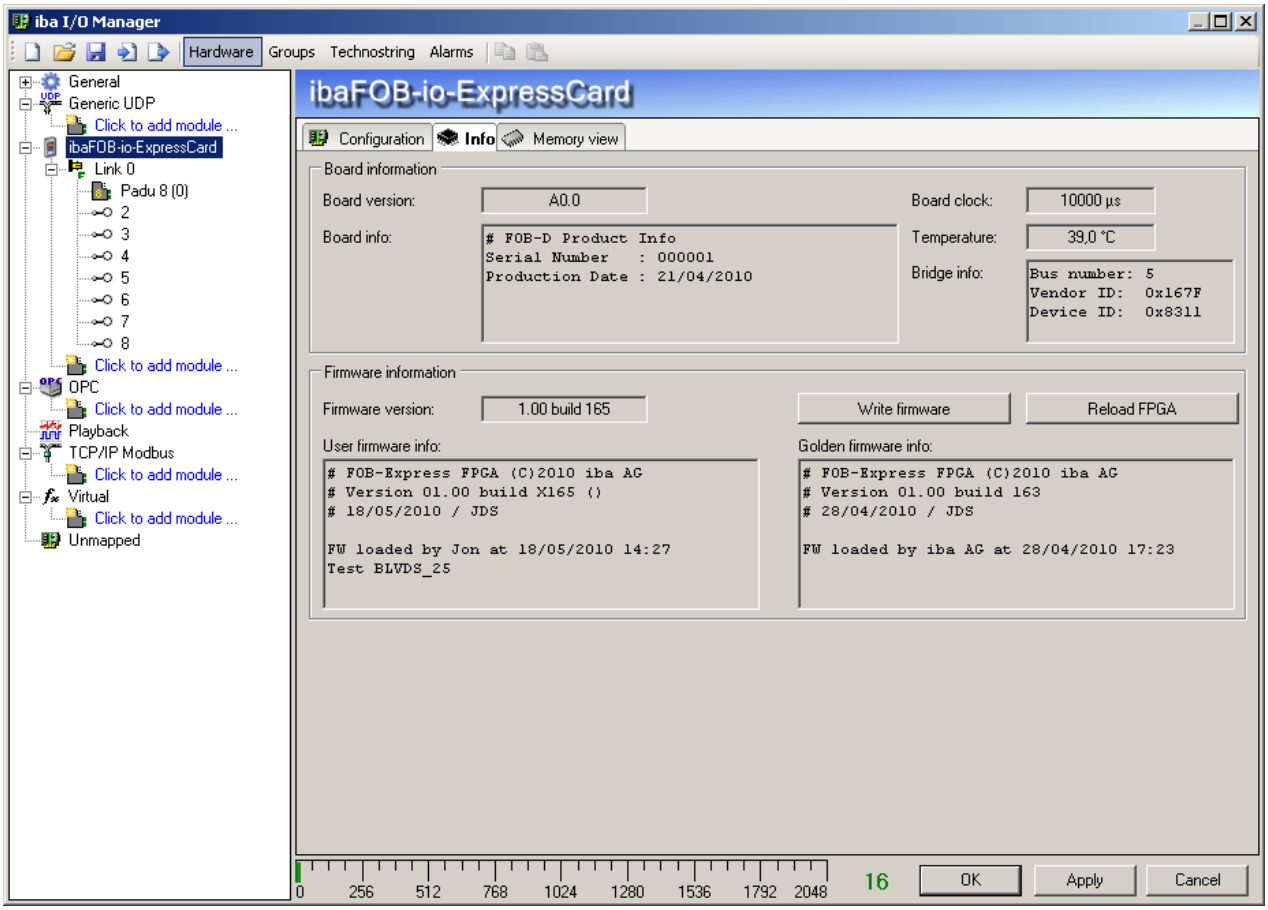
2 Support for ibaFOB-io-ExpressCard

The ibaFOB-io-ExpressCard is a card that can be inserted in the express card slot of a notebook. It has a FO input and a FO output. It behaves almost exactly the same as an ibaFOB-io-D board. The only difference compared to a ibaFOB-io-D board is that there is no sync connector. This means that the express card needs to generate its own clock. So the interrupt mode of an express card must be either master internal or master external. It can't be slave.



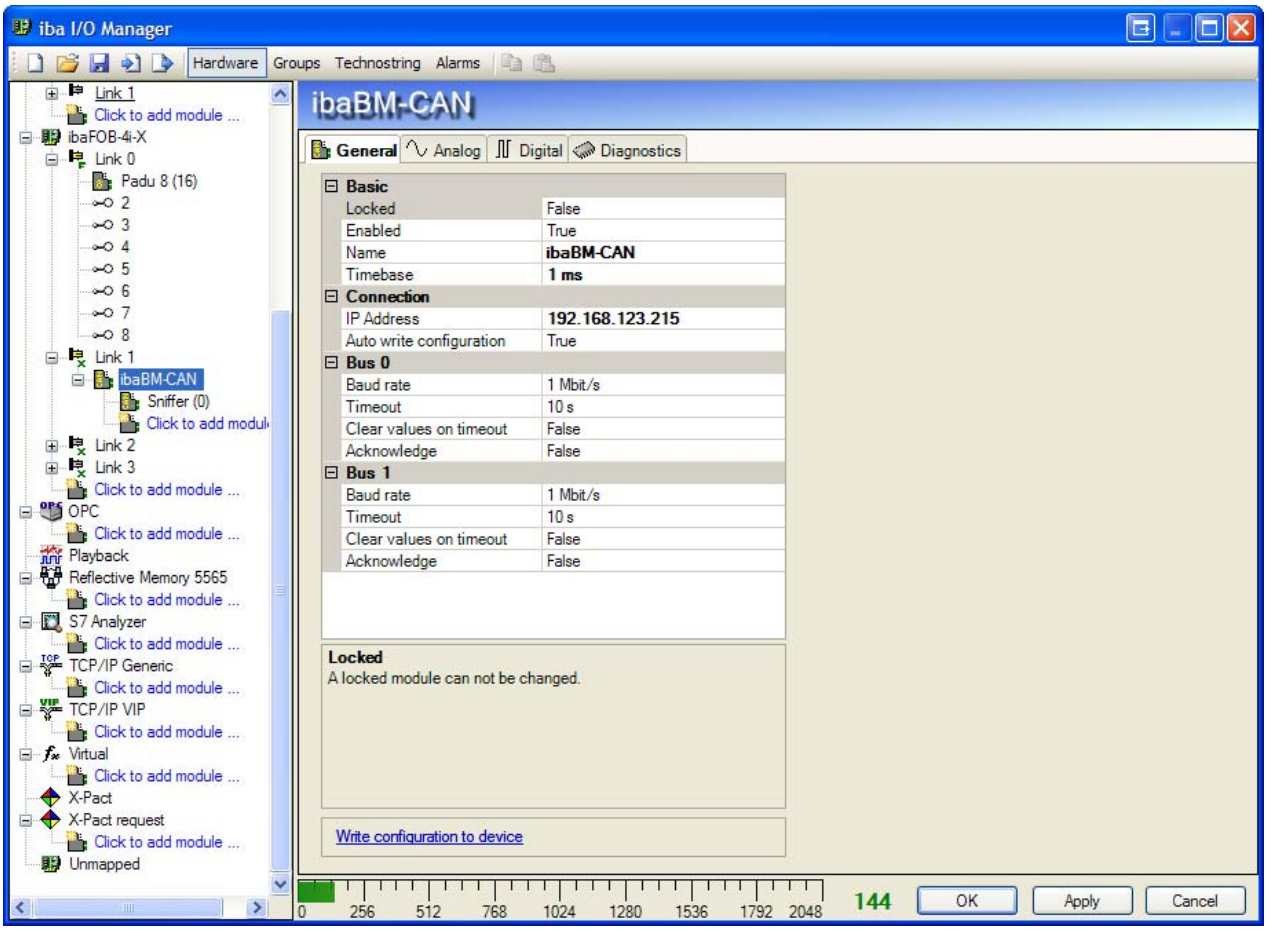
It supports all FO protocols: 2MBit/s, 3.3MBit/s, 5MBit/s and 32MBit/s. All module types that are supported on an ibaFOB-io-D board are also supported on the express card.

The info tab shows the temperature of the express card. All the other information is the same as on the ibaFOB-io-D board.


iba I/O Manager
Hardware
Groups
Technostring
Alarms
General
Generic UDP
Click to add module ...
ibaFOB-io-ExpressCard
Link 0
Padu 8 (0)
2
3
4
5
6
7
8
Click to add module ...
OPC
Click to add module ...
Playback
TCP/IP Modbus
Click to add module ...
Virtual
Click to add module ...
Unmapped
ibaFOB-io-ExpressCard
Configuration
Info
Memory view
Board information
Board version:
A0.0
Board info:
FOB-D Product Info
Serial Number : 000001
Production Date : 21/04/2010
Board clock:
10000 µs
Temperature:
39,0 °C
Bridge info:
Bus number: 5
Vendor ID: 0x167F
Device ID: 0x8311
Firmware information
Firmware version:
1.00 build 165
Write firmware
Reload FPGA
User firmware info:
FOB-Express FPGA (C)2010 iba AG
Version 01.00 build X165 ()
18/05/2010 / JDS
FW loaded by Jon at 18/05/2010 14:27
Test BLVDS_25
Golden firmware info:
FOB-Express FPGA (C)2010 iba AG
Version 01.00 build 163
28/04/2010 / JDS
FW loaded by iba AG at 28/04/2010 17:23
0
256
512
768
1024
1280
1536
1792
2048
16
OK
Apply
Cancel

3 Support for ibaBM-CAN device

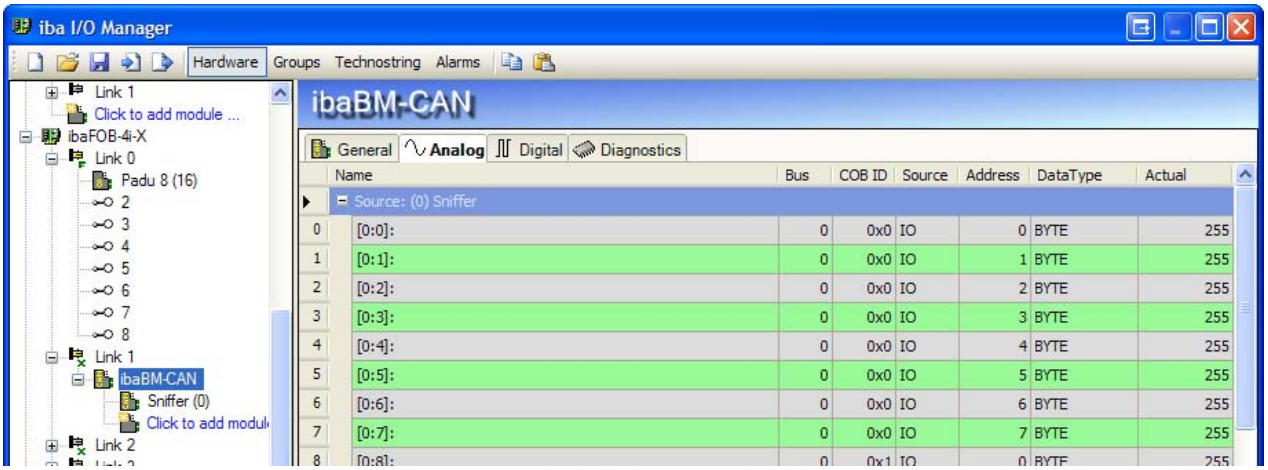
The ibaBM-CAN device is a CAN bus sniffer. It can be connected to 2 CAN buses.



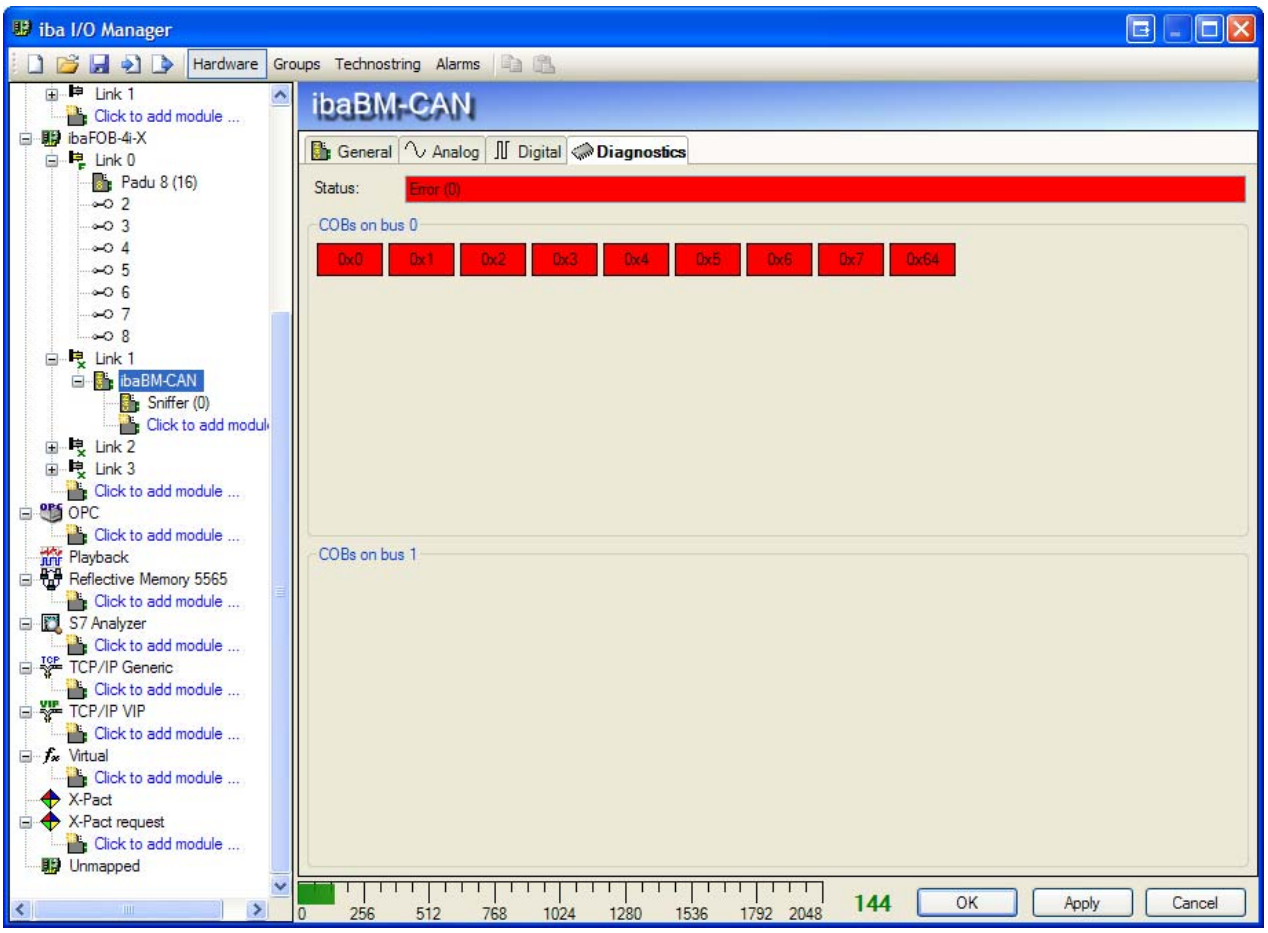
In ibaPDA you have to add a ibaBM-CAN device to a FOB-X or FOB-D board. On the general tab you can enter the IP address of the device. IbaPDA uses TCP to send the configuration to the device. It is also used to retrieve diagnostics information. If the *Auto write configuration* property is true then ibaPDA will write the configuration to the device each time the acquisition is started.

There are 4 properties you have to configure for each CAN bus. The first is the baud rate. The ibaBM-CAN supports all standard CAN baud rates from 10 kbit/s to 1 Mbit/s. The CAN bus is an event based bus. This means that not all COBs (these are the devices connected to the CAN bus) continuously send messages. Usually they only send a message when some event occurs. Therefore the sniffer can't really know if a COB is available on the bus or not. The sniffer uses a timeout to determine if a COB is available or not. If the sniffer doesn't see a message from the COB for the duration of the timeout then it considers the COB to be unavailable. You can configure the value of that timeout per bus. When a timeout occurs on a COB then the sniffer can either keep the current signals values it has for that COB or it can clear the values to zero. You can configure the behavior via the *Clear values on timeout* property. The fourth property is the *Acknowledge* property. This determines if the sniffer acknowledges each message that it receives. This should only be enabled if there are only 2 devices on the bus, the sniffer and another one.

The *Write configuration to device* hyperlink can be used to write the current configuration to the ibaBM-CAN device.

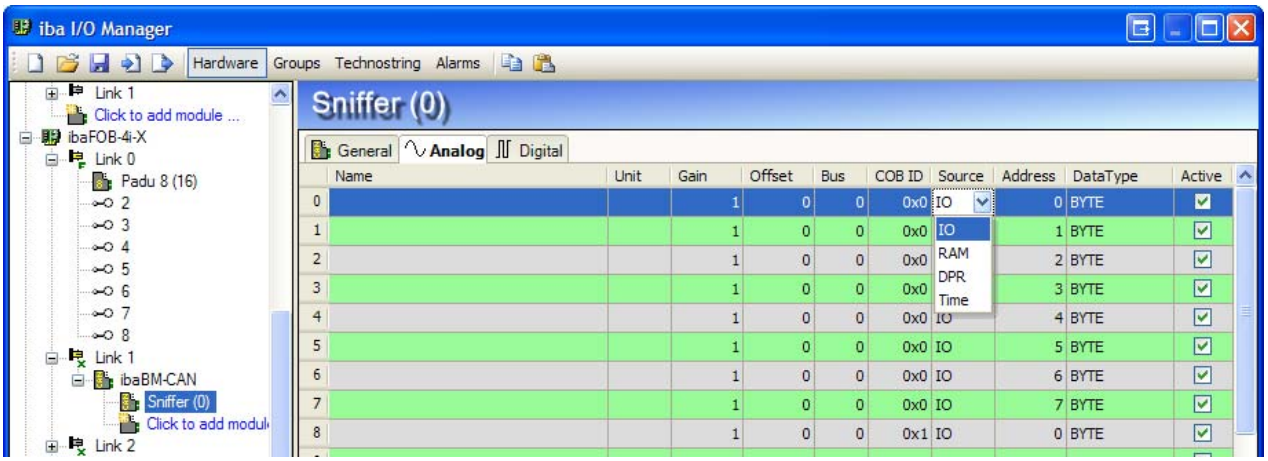


The analog and digital tabs show the current configuration. You can also see the actual values of the signals here.

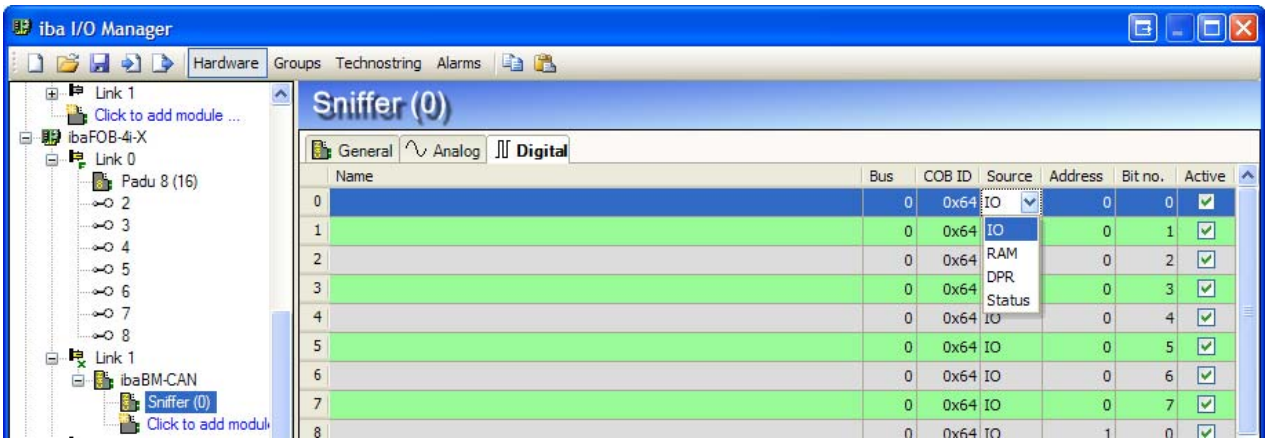


The diagnostics tab shows the status of the connected device. It also shows per bus the available COBs and the COBs that are being sniffed. The unavailable COBs are red, the available COBs are green and the available COBs that are being sniffed are green with a thick border. (The current screenshot only shows red COBs because I didn't have a CAN sniffer device anymore when I wrote this document.)

You can add different sniffer modules to the ibaBM-CAN device. This is the same principle as on the ibaBM-DPMS-S device. You can specify the number of analog and digital signals on a sniffer module.



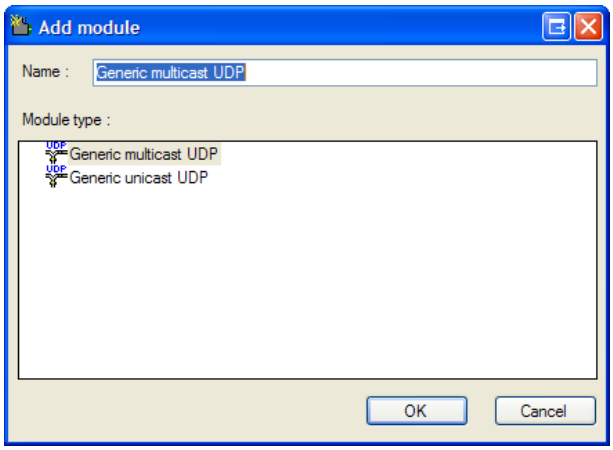
The analog signals have bus, COB ID, source, address and datatype properties. The bus is 0 or 1. The COB ID is a value between 0 and 2047. Each COB has an address space of only 8 bytes. These 8 bytes can be measured by selecting IO as the source. The address and datatype determine then how the data will be interpreted. The source property can also be set to RAM and DPR. These sources return values from the internal memory of the ibaBM-CAN device. They are only supposed to be used for iba support. The time source returns the time in microseconds between the last 2 messages for this COB.



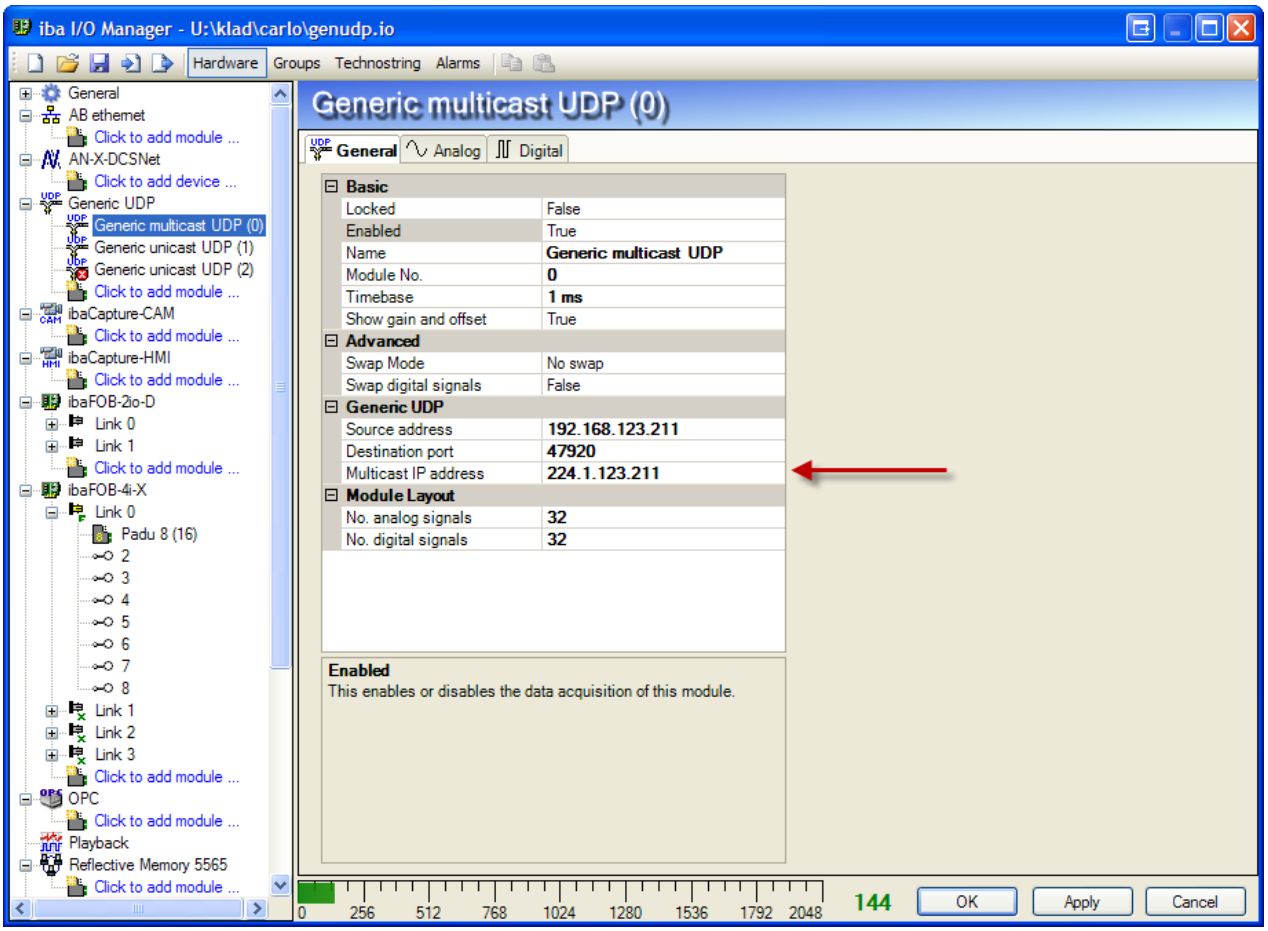
The digital signals have bus, COB ID, source, address and bit number properties. These properties are the same as for the analog signals. The digital signals have a source called status. This returns 1 if the COB is available and 0 if not.

4 Generic UDP multicast

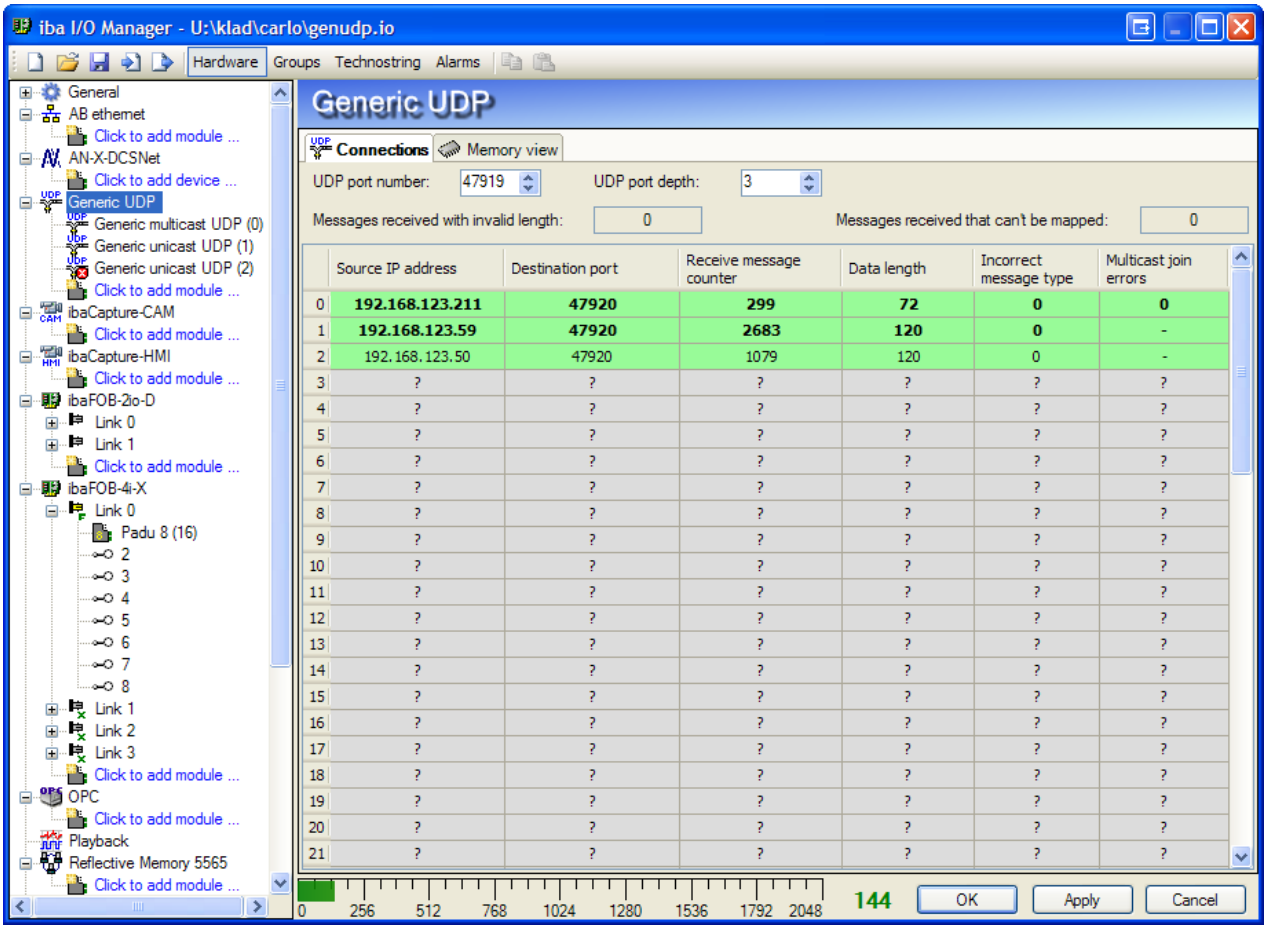
The generic UDP interface now supports 2 module types: unicast and multicast. The unicast module already existed in previous versions.



The only difference between the multicast and unicast modules is the destination IP address. For the unicast module this is the IP address of the pda server PC. For the multicast module this is a multicast address.



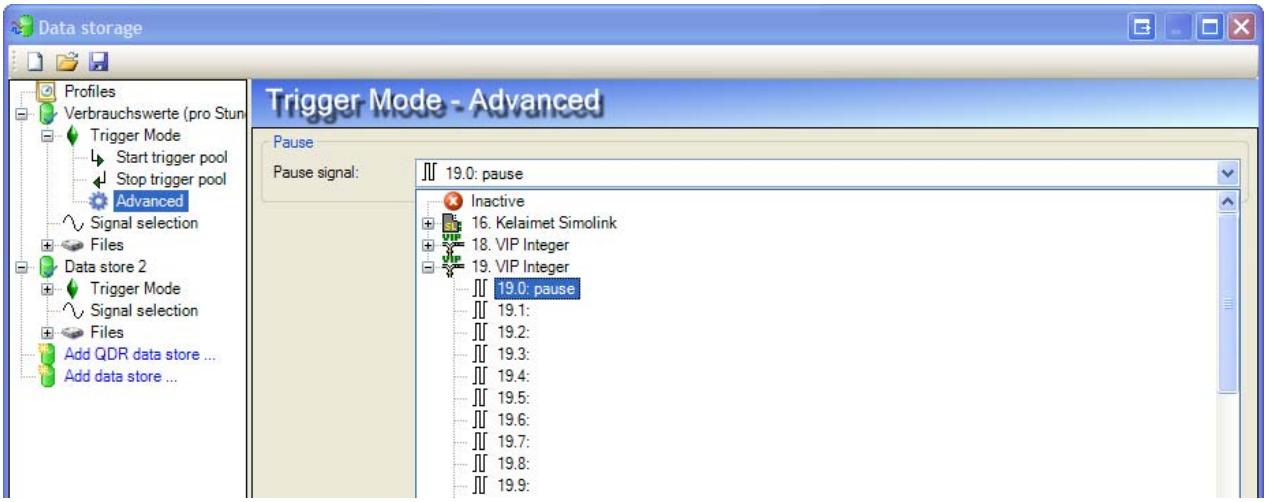
There are a few limitations for the multicast modules. The combination of source address and destination port number of a multicast module must be different than that combination of all the other unicast and multicast modules. So you can't have 1 sender sending to different multicast addresses with the same port number. You also can't have a sender that is sending both unicast and multicast messages to the same port number.



The generic UDP diagnostics have 2 extra columns. The incorrect message type column counts unicast messages on connections configured for multicast and counts multicast messages on connections configured for unicast. The multicast join errors shows how many times the multicast join failed on a multicast connection. This column is a '-' for unicast connections.

5 Datastore pause

A datastore can be paused by a digital signal. When a datastore is paused then all the data is replaced with invalid data which will be shown as a gap in ibaAnalyzer. Signals with datatypes INT, WORD and BYTE are stored as 16 bit integer values in the dat file. There doesn't exist an invalid bit for this datatype in the dat file so these values will be replaced with 0.



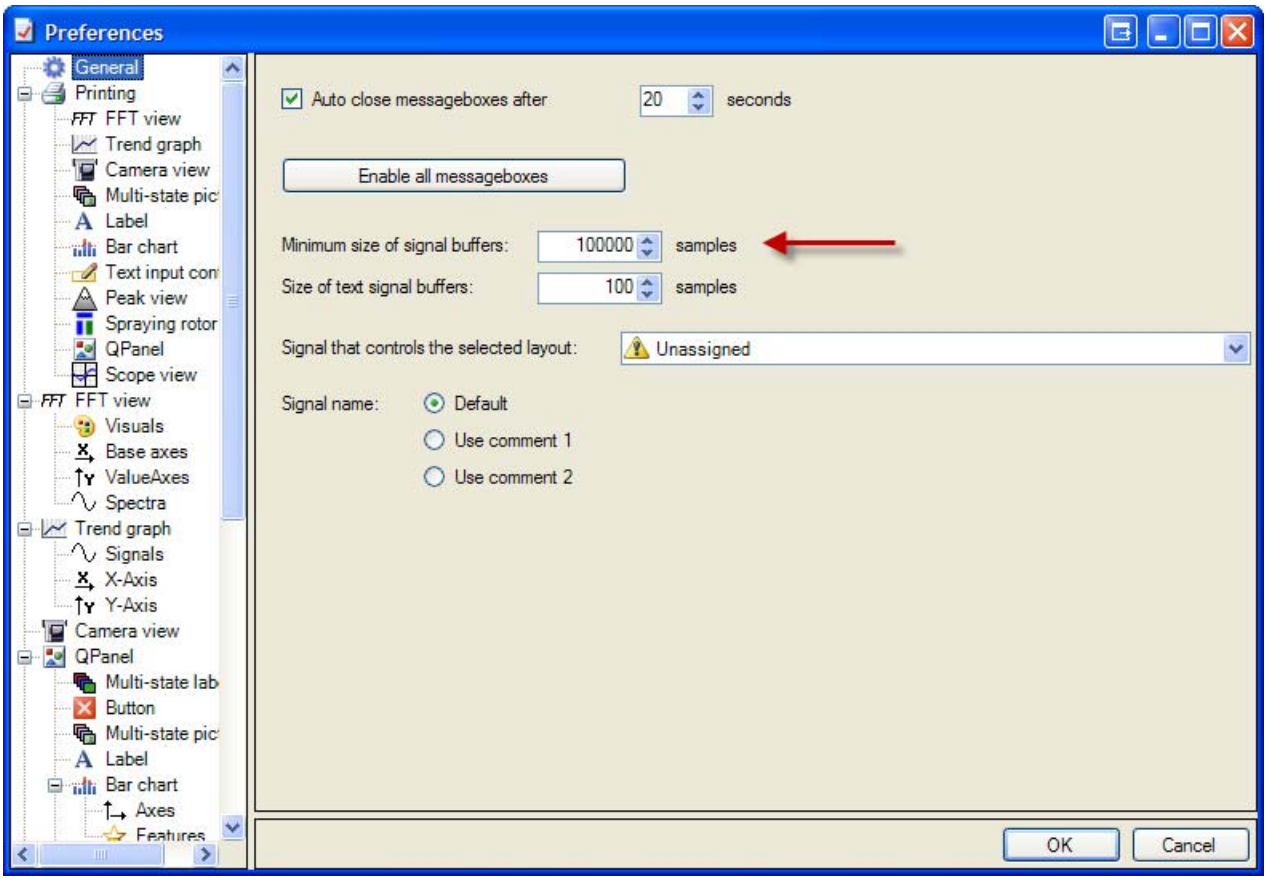
The period of the pda server acquisition loop is equal to the least common multiple of all module timebases with a minimum of 50 ms. The pause signal is evaluated only once per acquisition loop. It is considered to be 1 when it is 1 for the whole duration of the acquisition loop period. It is 0 as soon as 1 sample is 0 during the acquisition loop period. This means that the datastore will be paused some time x after the rising edge of the pause signal. The time x is between 0 and the acquisition loop period. This also means that the datastore will be resumed some time x before the falling edge of the pause signal.

The data storage status tree shows pause icons when the datastore is in pause.

6 Memory usage of client

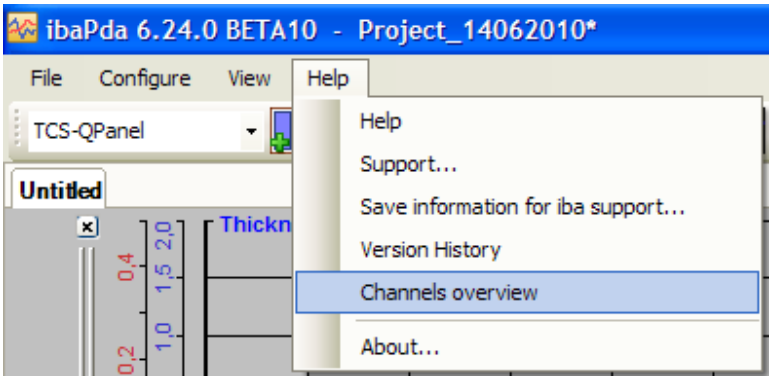
I have recently received several out of memory exceptions in QPanel from different customers. I haven't found any real memory leaks so far. The out of memory exceptions are usually caused by bad configurations. That is why I want to explain a bit the different settings that control how much memory the ibaPDA client and in particular QPanel uses.

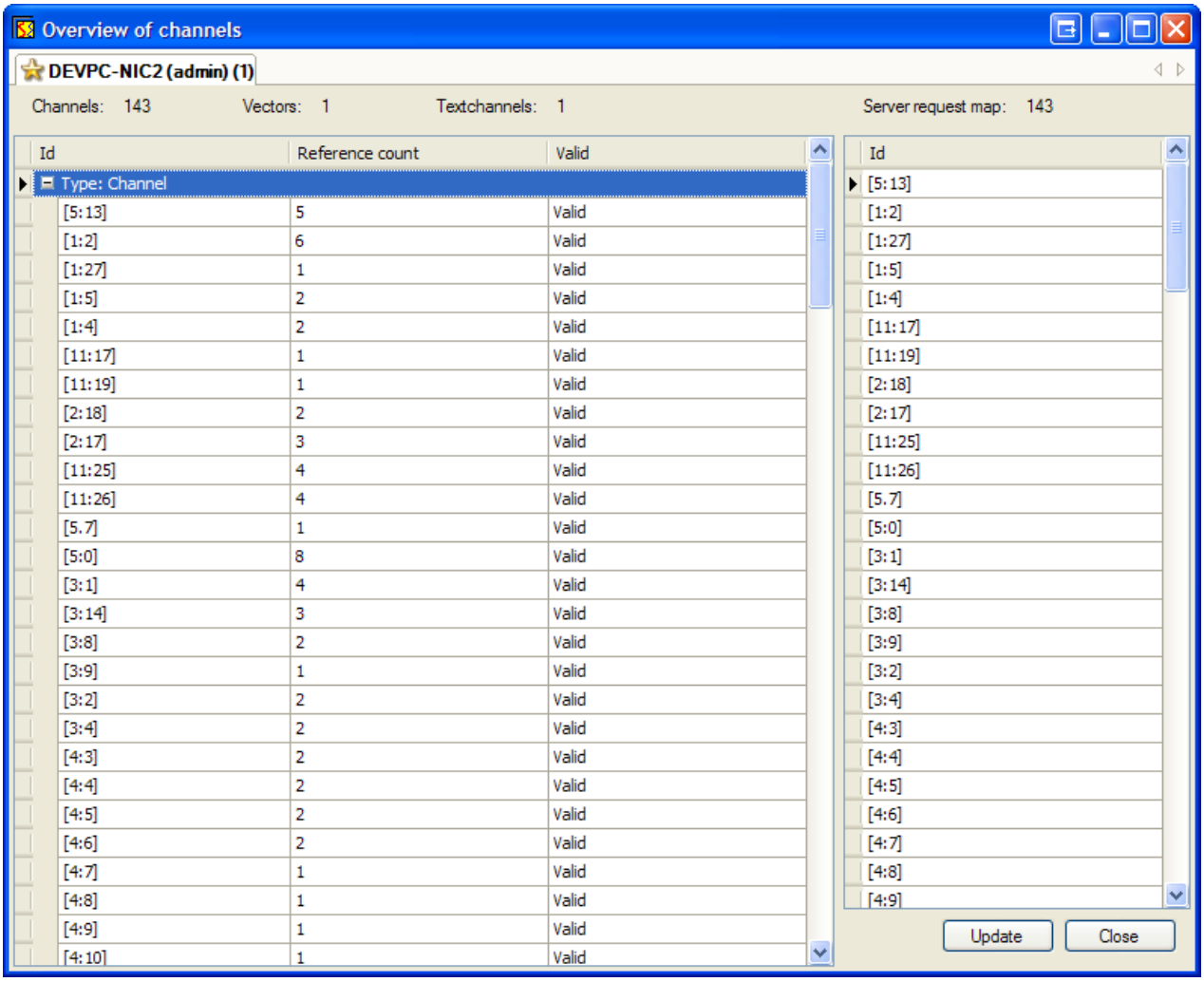
Each signal that is displayed in the pda client get its own buffer. The size of the buffer is determined by the minimum buffer size configured in the preferences.



The setting is called minimum buffer size because the data is stored compressed so depending on the nature of the signal the buffer can hold more samples. The analog signals use 5 bytes per sample and the digital signals 3 bytes per signal. So with the default setting of 100000 samples each analog signal will require about 500 kB and each digital signal about 300 kB.

The pda client requests data from the server for all the signals on all the layouts. This means that also the signals on different layouts than the currently visible one are being requested. So the client creates buffer for the union of all the signals in all the layouts. If you have multiple layouts and/or if you are using vectors then the number of signals can increase very quickly. You can see how many signals are requested by the client on the channels overview dialog. Hold SHIFT and open the help menu. An extra item called channels overview appears now. It can also be opened via the shortcut CTRL+SHIFT+O.





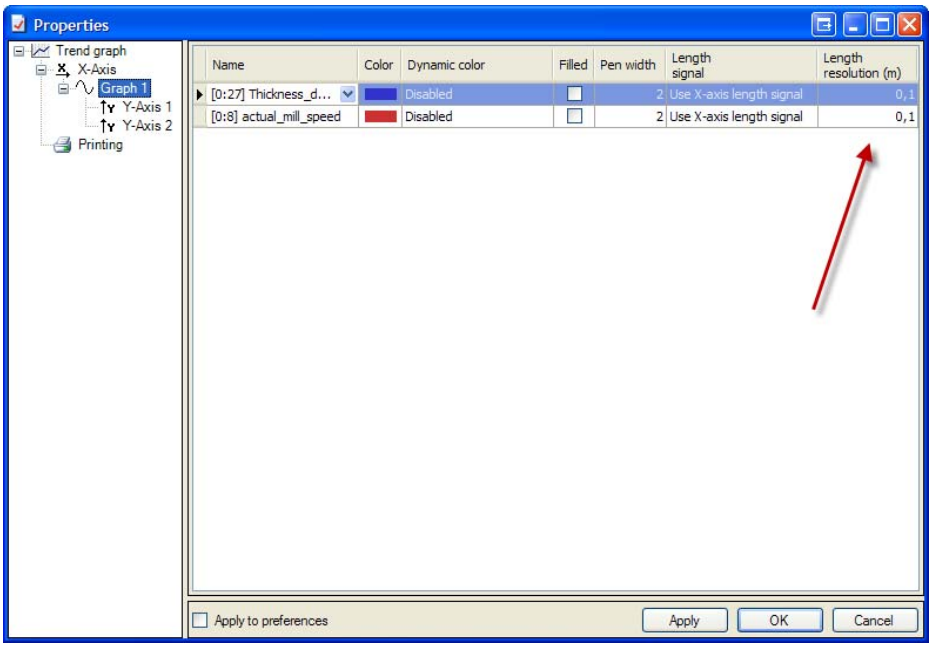
Id	Reference count	Valid
Type: Channel		
[5:13]	5	Valid
[1:2]	6	Valid
[1:27]	1	Valid
[1:5]	2	Valid
[1:4]	2	Valid
[11:17]	1	Valid
[11:19]	1	Valid
[2:18]	2	Valid
[2:17]	3	Valid
[11:25]	4	Valid
[11:26]	4	Valid
[5.7]	1	Valid
[5:0]	8	Valid
[3:1]	4	Valid
[3:14]	3	Valid
[3:8]	2	Valid
[3:9]	1	Valid
[3:2]	2	Valid
[3:4]	2	Valid
[4:3]	2	Valid
[4:4]	2	Valid
[4:5]	2	Valid
[4:6]	2	Valid
[4:7]	1	Valid
[4:8]	1	Valid
[4:9]	1	Valid
[4:10]	1	Valid

Id
[5:13]
[1:2]
[1:27]
[1:5]
[1:4]
[11:17]
[11:19]
[2:18]
[2:17]
[11:25]
[11:26]
[5.7]
[5:0]
[3:1]
[3:14]
[3:8]
[3:9]
[3:2]
[3:4]
[4:3]
[4:4]
[4:5]
[4:6]
[4:7]
[4:8]
[4:9]

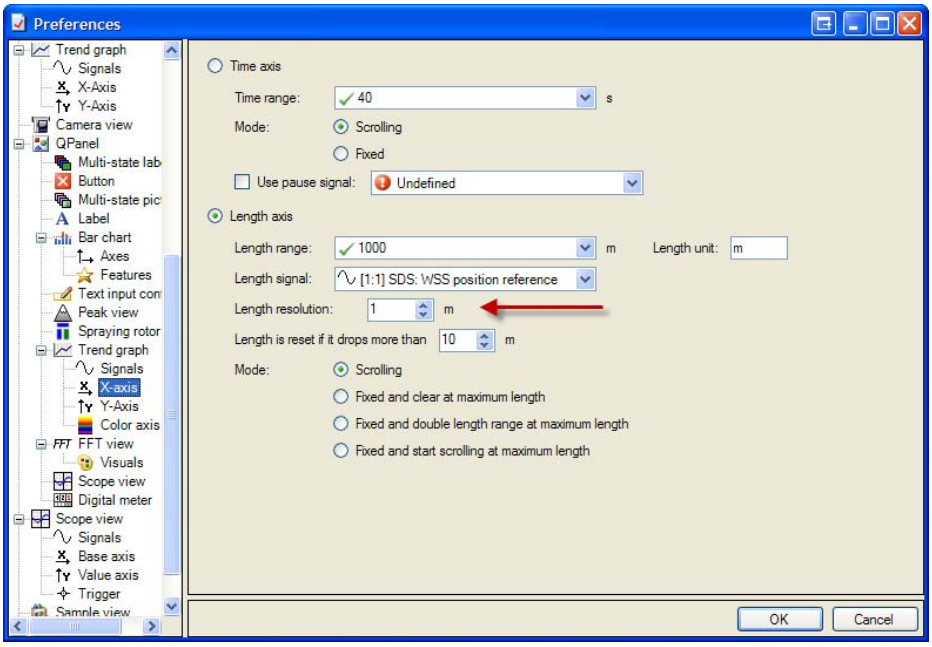
The channels overview dialog has 1 tab per connected server. This will always be 1 in the case of the normal ibaPDA client. It can be multiple ones when using the ibaPDA client Active-X control. The table on the left shows the signals configured in all the layouts. The reference count determines how many times a signal is used on the different layouts. The table on the right shows the signals that are being requested from the server. The number of channels shown in the left top corner determines how many buffers will be allocated.

The previous comments applied to all signals independent of which view type they were used by. Some view types require additional buffers per signal. The FFT view e.g. stores multiple FFT results when used in 3D mode.

The lengthbased trend graph stores lengthbased data for each signal displayed on it. The timebased data of a signal is converted to lengthbased data via a length signal. Whenever the length signal changes more than the configured length resolution a X,Y pair is stored. X is the length position as a float and Y is the signal value as float (also for digital signals). The length resolution of a signal can be set on the signal properties of the trend graph.



Since version 6.24.0 you can specify the default length resolution in the preferences. In the previous versions the default length resolution was fixed to 0,1m.



Each lengthbased sample uses 8 bytes of memory. The number of lengthbased samples is determined by the length of the coil, by how fast the length signal changes and by the configured signal length resolution. If you have e.g. a coil of 5000m length with a length resolution of 0,1m and a length signal that is generated by integrating a speed signal then you will have 50000 samples that use about 400 kB of memory.

The lengthbased data for a vector is stored as X,Y1,Y2,Y3,…Yn where X is the length position and Yx is the value of element x. So the length position is only stored once instead of for each element.

The lengthbased trend graph in versions prior to 6.24.0 stored lengthbased data for the 3 last coils plus the current coil, so 4 coils in total. Since version 6.24.0 the trend graph only stores the current coil when a fixed x-axis is used. When a scrolling x-axis is used then the current and the last coil are stored and all other coils that are currently in view. So if you have a large x-axis range then more coils will be stored in scrolling mode.