OPC CANopen Server

User Guide

Issue 3

1 Introduction

This document is a guide to explain to users how to work with the OPC CANopen server version 2.8.1. The OPC Server supports several CAN cards and comes in two parts:

- A card dependant component
- The OPC Server component

The card dependant part is implemented as a COM component. Both the OPC Server component and the card dependant component must be installed and registered on the computer which the OPC Server will be used on.

To enable addressing of OPC items within an OPC client, the address space of the OPC Server must be pre-configured. This is done using a configuration file that maps CANopen objects to OPC items. The user is responsible for creating this configuration file as applications differs greatly, therefore rendering a standard configuration file useless. See section 3, “Configuration of Address space” for details on how to create and modify a configuration file.

The usage of the server depends on the client program as well. This document is aimed towards use with the PVSS OPC Client, and details on how to use any OPC server using PVSS can be found at:

http://itcobe.web.cern.ch/itcobe/Services/Pvss/FAQ/Questions/opcClientConfiguration.html

This version of the OPC Server allows for any of the cards supported to be in the same PC at the same time, e.g. there could be two National Instruments cards and three Kvaser cards installed (assuming availability of PCI slots in the PC).

2 Installing the CANopen Server

Unzip the compressed file containing the OPC Server and components to a temporary directory. Run the setup file names ‘setup.exe’.

3 Configuration of Address space

The CANopen OPC server imports all information needed to define its address space (i.e. the named data available to the OPC client) from a configuration file called OPCCanServer.cfg. This file must be placed in the same directory as the binary executable file of the OPC Server.

The file contains six sections describing different parts of the CAN system, as explained below:

[CANBUS] – Describes all CAN buses that the system has.

[DEVICE] – Contains records about all nodes in each system.

[ELMB] – Contain record about all ELMB nodes in each system. The server creates the

default set of items (see Appendix A, “Default OPC configuration for the ELMB”)

[SDOItem] – Lists all OPC items that are mapped to entries in the object dictionary and hence accessed via SDO.

[PDO] – Define all PDO messages for real time data transmission.

[PDOItem] – Lists all OPC items that are mapped to bytes transmitted in the different PDOs defined in the section **[PDO]**.

[INIT] – Defines any values for OPC items that are to be initialized on startup of the OPC Server.

[ITEM] – Defines analytic formulas to convert raw data to physical units and result put in memory items.

Each section may contain several lines, all having the same syntax. The first word is a name followed by several parameters after the symbol “=”. Each parameter is separated by a space.

[CANBUS]

This section describes the CAN buses where each bus must have a unique name.

The first parameter is the name of the COM (card dependant) component as shown below:

- NICANserver+ This is the name for the National Instruments interface
- KVCANserver+ This is the name for the Kvaser interface

The second parameter is a port name. This is dependant upon the card and each port must be configured using the tools supplied by each supplier:

- (can0, can1…) for NI card
- (0,1….) for Kvaser card

The last parameter is the speed of the bus. For example,

To define a bus named “CAN_NET”, which uses a NI CAN card at 125kBaud on the port named “can0”, the following line must be entered:

CAN_NET = NICANserver+ can0 125000

To define a bus named “CAN_KV”, which uses a Kvaser CAN card at 125kBaud on the port named “0”, the following line must be entered:

CAN_KV = KVCANserver+ 0 125000

[DEVICE]

This section lists the nodes (or devices) on each of the CANopen buses. It specifies a unique name for each node and requires a minimum of two parameters. A third parameter is optional. The two required parameters are the name of the bus to which it is connected and the node identifier number (as required by all CAN nodes). The optional third parameter specifies whether to perform “Node Guarding” on the node, or that the node sends the “Heartbeat” message.

For example,

To define a node called “ELMB_1” on the bus “CAN_NET” with node ID 0x2F that

utilizes Node Guarding, the following line must be entered:

ELMB_1 = CAN_NET 2F NG

For the same node on the same bus, but without Node Guarding, the following line must be entered:

ELMB_1 = CAN_NET 2F

If you would like to receive the heartbeat message from the node, you have to write:

ELMB_1 = CAN_NET 2F HB

[ELMB]

This section is to describe the ELMBs. The syntax is the same as in the previous section. The server creates the predefined set of items for the ELMB. This set is described in Appendix A.

[SDOItem]

This section is used to define the OPC items that are mapped to entries in the object dictionary of a particular node and that are accessed via SDO. Each line in this section must supply an OPC item name and give five parameters:

1. The name of the node (as defined in the **[DEVICE]** or **[ELMB]** section)
2. Index of SDO object as defined in the CANopen object dictionary
3. Sub-index of SDO object as defined in the CANopen object dictionary
4. The direction – this may be IN (for input), OUT (for output) or IO (for bi-directional)
5. The type of data – this must be a type as defined in the OPC specification such as, VT_UI1 (unsigned integer, 1 byte), VT_UI2 (unsigned integer, 2 bytes). See Table 1 for a list of all types.

For example,

To define an SDO with an item ID of “rate”, which is defined in the object dictionary for the node “ELMB_1” having an index of 2100, sub-index 2 that is a bi-directional value of a single byte, unsigned integer, the following line must be entered:

rate = ELMB_1 2100 2 IO VT_UI1

[PDO]

This section is used to define the COB-IDs for the PDOs defined in each bus of the system. The number of parameters depends on the Device Specification Profile (DSP) that the PDO belongs to. Three parameters are mandatory:

1. The bus name as defined in the section **[CANBUS]**
2. The DSP – the OPC Server supports three profiles: 401, 404 and LMB[1]

[1] The LMB DSP is essentially the 404 with the order of bytes reversed, which is used for the LMB

3. The direction of the PDO – this may be IN (input) or OUT (output). PDOs ***cannot*** be bi-directional.

For the profiles 404 and LMB, two additional parameters are required:

4. The zero based index of the multiplexer byte
5. The number of times the PDO is multiplexed

For example,

To define an input PDO with COB-ID 0x1AF on the bus named "CAN_NET". If this uses the DSP 404 (which is therefore multiplexed) with the multiplexer byte being the first byte (index 0) and is multiplexed 64 times (the multiplexer byte may be a value between 0 and 63 inclusive), the following line must be entered:

```
1AF = CAN_NET 404 IN 0 64
```

To define an input PDO with COB-ID 0x1BF on the bus named "CAN_NET". If this uses the DSP 401, the following line must be entered:

```
1BF = CAN_NET 401 IN
```

[PDOItem]

This section defines the OPC items that are mapped to bytes in the PDO defined in the section **[PDO]** i.e. the actual real time data value for the sensors and/or actuators. The number of parameters in the definition depends on the profile defined. There are four mandatory parameters required after the PDO item name:

1. The node name that the PDO applies to as defined in the section **[DEVICE]**
2. The COB-ID of the PDO as defined in the section **[PDO]**
3. The type of data – this must be a type as defined in the OPC specification such as, VT_UI1 (unsigned integer, 1 byte), VT_UI2 (unsigned integer, 2 bytes). See Table 1 for a list of all types.

The following parameters depend upon the profile defined for the PDO and the data type defined for this PDO item.

For DSP 401:

1. The zero based index for where the data value begins in the PDO message
2. If the data type is VT_BOOL (single bit value) then the zero based index for where the bit value is located in the byte

In order to extract several bit into one item, for date type VT_UI1 there are additional parameters.

```
Status = ELMB_1 1AF VT_UI1 3 4 2
```

The last two parameters mean that the server extracts 2 bits, beginning with the 4th bit (in byte 3).

For DSP 404 and LMB:

1. The "multiplex byte" value that applies to this PDOItem. The text **ALL** may be used that creates items for all possible multiplexed values by appending a count (starting at 0) to the end of the PDOItem name. E.g. for a multiplexed PDOItem called **value**, the items created would be **value0**, **value1**, etc.

2. The zero based index for where the data value begins in the PDO message
3. If the data type is VT_BOOL (single bit value) then the zero based index for where the bit value is located in the byte

For example,

Defining a multiplexed PDOItem named “value” for the node “ELMB_1” with COB-ID 0x1AF where all possible values wish to be addressed. The data is an unsigned integer of 2 bytes and the data starts in the fourth byte (index 3). The line must be entered as:

`value = ELMB_1 1AF VT_UI2 ALL 3`

This creates 64 items, from **`value0`**, **`value1`**, to **`value62`**, **`value63`** (assuming the PDO is as defined in the example above and therefore is multiplexed 64 times).

Defining a PDOItem named “setting” for the node “ELMB_1” with COB-ID 0x1BF. The data is an unsigned integer of 1 byte and the data starts in the third byte (index 2). The line must be entered as:

`setting = ELMB_1 1BF VT_UI1 2`

Defining a PDOItem named “switch” for the node “ELMB_1” with COB-ID 0x1BF. The data is a Boolean (1 bit) and the data is the third bit (index 2) of the second byte (index 1). The line must be entered as:

`switch = ELMB_1 1BF VT_BOOL 1 2`

`[INIT]`

This section defines values that are used to initialize any output items in the OPC address space. All values will be sent to the CAN nodes after creating the OPC address space and opening the CAN bus port in the order they are written. This is completed once at startup time of the OPC Server. The syntax for this section is:

<BUS>.<ITEM> = value

or

<BUS>.<NODE>.<ITEM> = value

For example,

To set the Sync Interval item (see section 3.1, “Predefined OPC Items” for information on the Sync Interval item) for the bus named “CAN_NET” to 1 second (1000 milliseconds), the following line is added:

`CAN_NET.SyncInterval = 1000`

To set the NMT item (see section 3.1, “Predefined OPC Items” for information on the NMT item) for the node named “ELMB_1” on bus “CAN_NET” to 1, the following line is added:

`CAN_NET.ELMB_1.NMT = 1`

Table 1 below shows the allowed data types that may be used for the items in the SDOItem and PDOItem sections.

Table 1, Data type identifiers allowed

Type Identifier	Size	C Style Equivalent
VT_BOOL	1 bit	`bool`
VT_I1	1 byte char	
VT_I2	2 bytes short	
VT_UI2	2 bytes unsigned	`short`
VT_I4	4 bytes int	
VT_UI4	4 bytes unsigned	`int`
VT_R4	4 bytes float	
VT_R8	8 bytes double	
VT_I8	8 bytes long	
VT_UI8	8 bytes unsigned	`long`

[ITEM]

This section describes items which can be calculated from the input items.

The syntax is

<Name of item(s)>[*start..end:step*]{*min:max*} = *<analytic formula>*

The name of items can be simple string and analytic formula can be a value:

`Coefficient = 6.2345`

It is possible to define a sequence of items:

`res[0..5:1] = 100.55 100.2 99.98 100.02 99.99 100.03`

The first digit in the bracket (**0**) means the value to start from, the second digit (**5**) is the value to end on and the third digit, after the colon, (**1**) is the step.

The OPC server creates six items (**res0**, **res1**… **res5**) based on this string and it assigns the values written on the left hand to those items.

The next example shows how we can assign the value of an input item to a memory item. On the left side, the name of input items has square brackets which values inside. There are only two digits in these brackets. The first digit (**1**) is the starting value and the second digit (**2**) is the step.

`Voltage[0..5:1] = LaR.ELMB_1.counts[0:2]`

The server creates six items (**Voltage0**, **Voltage1** … **Voltage5**) and these items will refer to the input items **Lar.ELMB_1.counts0**, **Lar.ELMB_1.counts2** … **Lar.ELMB_1.counts10**

The last example shows how to write an analytical formula. To the right of the equal sign, it is possible to use any items which have previously been defined and also four rules of arithmetic standard arithmetic functions (ln, sin, cos, tg, …) can be used.

`Resistance[0..5:1] = (Voltage[0:1]/Current[0:1])*res[0:1]`

We also can write .

Resistance[0..5:1]{20:130} = (Voltage[0:1]/Current[0:1])*res[0:1]

The values in braces defines the minimum and maximum of parameters

3.1 Predefined OPC Items

The OPC address space is a tree-based structure allowing an item to be represented as follows:

<BUS>.<ITEM> for example **ELMB_NET.Port Error**

or

<BUS>.<NODE>.<ITEM> for example **ELMB_NET.ELMB_1.value1**

In order to send the NMT or Sync messages, and to allow the error status of the card to be read for example, the CANopen OPC server has some additional pre-defined OPC items. It is important that no user defined items use the same name as those listed below.

For a bus the server creates the following items:

- Port Error – this indicates if there is an error on the interface
- Port Error Description – this is an explanation about any error on the interface
- NMT – this item allows to send NMT messages over a bus
- SYNC – this item allows to send a single Sync message to a bus
- SyncInterval – this item allows a periodic sync messages to be sent to a bus (value set is in milliseconds)
- NodeGuardingInterval – this item sets the node-guarding interval (value set is in milliseconds)

For a node the server creates the following items:

- NMT – this item is to send NMT messages to a node
- Boot Up Message – this item contains the number of boot up messages from a node and is updated whenever a bootup message is received
- Emergency Error Code – this item contains the value of Emergency Error Code from last emergency message and is updated whenever an emergency message is received
- Error – this item holds the contents of the node error register
- Specific Error Code Byte 1... 5 – these 5 items contain data from corresponding bytes of Emergency Message.

Below you can see the example of a CANopen system that consists of two buses and two nodes (one node per bus).

```
[CANBUS]
ELMB_NET = NICANserver+ can0 125000
WAGO_NET = KVCANserver+ 0 250000

[DEVICE]
ELMB = ELMB_NET 3F NG
```

```
WAGO = WAGO_NET 1

[SDOItem]
AS = ELMB 1800 2 IO VT_UI1

[PDO]
1AF = ELMB_NET ELMB IN 0 16
22F = ELMB_NET 401 OUT
181 = WAGO_NET 401 IN
201 = WAGO_NET 401 OUT

[PDOItem]
rd = ELMB 1AF VT_UI2 ALL 1
Config = ELMB 1AF VT_UI1 ALL 4
ADC1 = ELMB 22F VT_UI1 0 Switch1 = WAGO 201 VT_BOOL 0 0
Switch2 = WAGO 201 VT_BOOL 0 1
LAMP1 = WAGO 181 VT_BOOL 0 0
LAMP2 = WAGO 181 VT_BOOL 0 1
```

4 CANHOST utility

The “canhostplus” utility has been ported from “openhost” (supplied by Henk Boterenbrood, Nikhef) in order to use component access to the CAN bus. This utility can work with either the NI card or the Kvaser card. To use the utility, from a command line interface (DOS window) navigate to the ‘Tools’ subfolder of the installation directory (by default, this is C:\CANopenOPC) and enter:

```
canhostplus NICANserver+:can0 125000
```

for the NI card interface on the port named “can0” at 125kBaud

or

```
canhostplus KVCANserver+:0 125000
```

for the Kvaser card interface on the port named “0” at 125kBaud

The menu and functions are the same as the openhost utility**.**

NOTE: The card dependant components must have been registered for the CANHOSTPLUS utility to work (which should have been done automatically at installation time).

5 Common Questions

How can I check whether my configuration settings have been acknowledged by the ELMB?

The quality bit of the readable item (which must be addressed in a different element to the writeable item) is set to bad when a value is sent to the ELMB from the OPC Server. If the ELMB acknowledges the setting, this quality bit will be set to true.

Note: In PVSS, the quality bit is given by the _online.._aut_inv config for an element. For this bit to be set by the OPC Server, in the "OPC parameterization" panel, the "OPC bits --> PVSS bits" check box must be ticked.

When I send a value to the ELMB via the OPC Server, even when the ELMB does not respond the readable value is updated to the value I just tried to set. Should this happen?

The readable item being updated allows you to tell that the message has been accepted by the OPC Server and sent to the ELMB. If the ELMB does not respond, although the value is updated, the quality flag is set to bad (false) indicating that this value has not been accepted by the ELMB.

How do I check that my ELMB node is still "alive"?

A mechanism called Node Guarding may be used which sends a message to the ELMB at specified intervals and this must be acknowledged by the ELMB. If the ELMB does not respond to five Node Guard messages in a row, then values are set to indicate the ELMB is no longer alive. The indications are that an item named "Error" (defined for each ELMB by default in the OPC Server) is given the value 8 (eight) and that all readable items have the quality bit set to bad (false). See section 3 "Configuration of Address space" under [DEVICE] for details on specifying Node Guarding for an ELMB.

I have specified Node Guarding for an ELMB in the OPC Server configuration file but after some time, I get the indication that the ELMB is not alive. However, I can see values updating so I know it is OK. What is wrong?

Earlier versions of the ELMB software do not have the function implemented to respond to Node Guarding. To check the version, read SDO index 100A, sub-index 0. Version MA32 is known to NOT have Node Guarding. Update the software in the ELMB to the latest version, or remove the specification for Node Guarding from the OPC configuration file.

When I start the OPC Server, the values in the ELMB are not updated into my OPC Client, why is this?

The CANopen standard does not allow for messages containing configuration information to be sent without a specific request. If the values in the ELMB are required to be updated when the OPC Server starts, this must be completed by reading each of the required values from the ELMB after the connection to the server has been made.

6 References

1. OLE for process Control, Data Access Standard, Version 2.05 September 11,2002

2. CANopen Implementation GuideLines, by G.Gruhler and Bernd Dreir, STA Reutiligen, Germany, 1997

Appendix A Default OPC configuration for the ELMB

This section describes the default OPC items that will be created for any ELMB defined in the **[ELMB]** section of the configuration file. This section is optional and is a new section in order to keep backwards compatibility with previous versions of the OPC server. Items in this section have the following syntax:

[ELMB]

<nodeName> = <busName> <nodeId> <nodeGuardingFlag> <setFlag>

where,

- *<nodeName>* is an string representing the device name.
- *<busName>* represents the bus name this ELMB belongs to.
- *<nodeId>* is the ID for this ELMB (hex format)
- *<nodeGuardingFlag>* defines whether the OPC server has to perform node guarding for this particular node or listens for the heartbeat message. If node guarding is defined the value of this flag is **NG**. If heartbeat messages are to be received, the value of this flag is **HG**
- *<setFlag>* is used to specify whether to create OPC items for ADC counts or for microvolts. By default, the ELMB sends values in microvolts and therefore this flag will not be used very often. If the ELMB is configured to send ADC counts rather than microvolts, then the text for this flag is must be **counts**. If this flag is omitted, the default setting is used which is for microvolts.

For example, to define an ELMB called ELMB_3F_1 on a bus called MUON, having a node ID of 3F (hex) with node guarding and giving values in counts, the following line(s) should be entered:

```
[ELMB]
ELMB_3F_1 = MUON 3F NG mv
```

The default configuration of the ELMB comprises OPC items for digital input and output, analog inputs, different items to handle the configuration of the ADC and other frequently used settings. The OPC server follows the convention shown in Table 2 to automatically handle the PDO COB-ID assignment.

Table 2, PDO COB-ID assignment

Channels	PDO	COB-ID	Example[2]
Digital Inputs	TxPDO1	0x180 + NodeId	0x1BF
Digital Outputs	RxPDO1	0x200 + NodeId	0x23F
Analog Inputs (counts)	TxPDO2	0x280 + NodeId	0x2BF
Analog Inputs (mV)	***TxPDO3***	***0x380 + NodeId***	***0x3BF***

Table 3 lists the different OPC items that are created by default. The OPC item names are taken from the above example.

Table 3, OPC items for ELMB default configuration (Assuming ELMB name: ELMB_3F_1 and bus name MUON)

OPC Item	Description	Type	Attr
MUON.ELMB_3F_1.swVersion	ELMB software version	BSTR	RO
MUON.ELMB_3F_1.rate	ADC conversion rate	VT_UI1	RW
MUON.ELMB_3F_1.range	ADC range	VT_UI1	RW
MUON.ELMB_3F_1.mode	ADC measurement mode	VT_UI1	RW
MUON.ELMB_3F_1.save	Saves configuration to EEPROM	VT_UI4	WO
MUON.ELMB_3F_1.load	Loads default configuration from EEPROM	VT_UI4	WO
MUON.ELMB_3F_1.channelMax	Total number of ADC channels	VT_UI1	RW
MUON.ELMB_3F_1.di_F_# (#-0..7)	Digital Input Bits (ELMB Port F)	VT_BOOL	RO
MUON.ELMB_3F_1.di_F_byte	Digital Input Byte (ELMB Port F)	VT_UI1	RO
MUON.ELMB_3F_1.digitalInEnable	Enables transmission of Digital Input signals	VT_BOOL	RW
MUON.ELMB_3F_1.do_C_# (#-0...7)	Digital Output Bits (ELMB Port C)	VT_BOOL	RW
MUON.ELMB_3F_1.do_C_byte	Digital Output Byte (ELMB Port C)	VT_UI1	RW
MUON.ELMB_3F_1.do_A_# (#-0...7)	Digital Output Bits (ELMB Port A)	VT_BOOL	RW
MUON.ELMB_3F_1.do_A_byte	Digital Output Byte (ELMB Port A)	VT_UI1	RW
MUON.ELMB_3F_1.ai_#	Analog Input Channel # - 0...63	VT_UI2/VT_UI4[3]	RO

In addition, the user will be able to add new OPC items map to either SDO or PDO on top of the default configuration. However, the user must not use OPC item names reserved to the default configuration, e.g. it is not possible to define a new OPC item called **`digitalInF`** since this name is used in the default configuration of the ELMB. In the following examples, the node **`ELMB_3F_1`** has been defined with the default configuration in the bus called **`MUON`**.

[2] NodeID = 0x3F has been used in the example.

[3] The type of these items depends upon the setFlag. If the setFlag is “counts” the type is VT_UI2, otherwise is it VT_UI4

a) To create a new OPC item to set the Debounce Timer for the digital inputs by SDO:

```
[CANBUS]
MUON = KVCANserver+ 0 125000

[ELMB]
ELMB_3F_1 = MUON 3F NG

[SDOItem]
debounce = ELMB_3F_1 2200 0 IO VT_UI1
```

b) To define two new (multiplexed)[4] OPC items for:

- 64 analog outputs: **ao**
- The state of each of the 64 analog input channels: **state**

```
[CANBUS]
MUON = KVCANserver+ 0 125000

[ELMB]
ELMB_3F_1 = MUON 3F NG

[PDO]
33F = MUON 404 OUT 0 64

[PDOItem]
ao = ELMB_3F_1 33F VT_UI4 ALL 2
state = ELMB_3F_1 2BF VT_UI1 ALL 1
```

Note that in the case of the OPC items called state, it is not necessary to define the TxPDO2 (COB-ID: 0x280 + 0x3F = 0x2BF), since this PDO forms part of the default configuration carried out by the OPC server (see Table 2)

[4] i.e. 64 items of each type will be created

A last feature of the OPC default configuration of the ELMB is that the user can create new OPC items, which duplicate the information contained in the address space of the OPC item, e.g.

```
[CANBUS]
MUON = KVCANserver+ 0 125000

[ELMB]
ELMB_3F_1 = MUON 3F NG

[PDOItem]
counts27 = ELMB_3F_1 2BF VT_UI2 27 2
```

In this case, the OPC address space would contain two items mapped to the value of the analog input channel number 27 of **ELMB_3F_1**, **MUON.ELMB_3F_1.ai_27** created in the default configuration and **MUON.ELMB_3F_1.counts27** as defined by the user.

The following example shows how the configuration file could look if the default configuration is applied to all ELMBs:

```
[CANBUS]
MDT_1 = KVCANserver+ 0 125000
MDT_2 = KVCANserver+ 1 125000
TILECAL = NICANserver+ can0 125000

[ELMB]
ELMB_3F_1 = MDT_1 3F NG
ELMB_3E_1 = MDT_1 3E
ELMB_3E_2 = MDT_2 3E NG
ELMB_TILE_1A = TILECAL 1A NG
```

This file defines three buses:

- The **MDT_1** bus is connected to port 0 of the Kvaser interface card and works at 125 kbits/s, which has two nodes: **ELMB_3F_1**, **ELMB_3E_1**. Node guarding is performed for the first of them.
- The **MDT_2** bus is connected to port 1 of the Kvaser interface card and works at 125 kbits/s with only one node: **ELMB_3E_2** with node guarding protocol
- The **TILECAL** bus connected to port 0 (can0) of the NI-CAN interface card and works at 125 kbits/s with only one node: **ELMB_TILE_1A** with node guarding protocol

Appendix B Manual Registration of OPC Server and components

The server is started automatically when a client connects to it. To allow the system to do this, it is necessary to register the server and card components. This should be completed during the setup process. If problems have occurred during the setup, this may need to be done manually. The following instructions give details on how to do this:

IMPORTANT: Please read the readme file (OPCServer_readme.txt, contained in 'Docs' subfolder of the installation directory) for *essential* information regarding the registration procedure. If you do not follow the instructions in this file, the OPC Server may not work.

Registration is performed by the following:

1. From the "Start" menu, select "Run…"
2. "Browse" the local hard drive(s) for the folder containing the OPC Server component (named "CanOpen25+.exe", which is copied into C:\CANopenOPC in the default installation)
3. Click the button labeled "Open"
4. Edit the text displayed to add a space and the text "/RegServer" e.g. if the OPC Server component was found in C:\CANopenOPC, the text in the "Run" dialog would be:
   C:\CANopenOPC\CanOpen25+.exe /RegServer
5. Click the button labeled "OK"
6. A message box should be displayed indicating that the server has been registered successfully
7. Click the button labeled "OK"

The OPC Server supports CAN cards from two manufacturers: National Instruments and Kvaser.

B.1 National Instruments

The cards supported are:

- NI PCI-CAN – Single port, high speed PCI to CAN interface
- NI PCI-CAN/2 – Dual port, high speed PCI to CAN interface
- NI PCMCIA-CAN – Single port, high speed PCMCIA to CAN interface
- NI PCMCIA-CAN/2 – Dual port, high speed PCMCIA to CAN interface

The COM component for the NI CAN cards is a Dynamic Link Library (DLL) file named "nicanbusplus.dll". To allow the OPC Server component to have access to this file, the component must be registered. This may be completed using the following steps.

1. From the "Start" menu, select "Run…"

2. "Browse" the local hard drive(s) for the folder containing the NI card dependant component (named "nicanbusplus.dll", which is copied into C:\CANopenOPC in the default installation)
3. Click the button labeled "Open"
4. Edit the text displayed to add the text "regsvr32 " BEFORE the name of the DLL (with a space between this text and the full DLL file path and name). e.g. if the NI card dependant component was found in C:\CANopenOPC, then the text in the "Run" dialog would be:
   regsvr32 C:\CANopenOPC\nicanbusplus.dll
5. Click the button labeled "OK"
6. A message box should be displayed indicating the DLL has been successfully registered
7. Click the button labeled "OK" on the message box

NOTE: **Version 2.0** of the NI CAN card driver is used. This driver and COM component can support several NI cards in the same PC at the same time. If you do not have this version of the driver, the OPC Server will not work, but no error will be given at registration time.

B.2 Kvaser

The cards supported are:

- PCICan-D – Dual port PCI to CAN interface
- PCICan-Q – Four port PCI to CAN interface
- LAPCan II – Dual port PCMCIA to CAN interface

The COM component for the Kvaser CAN cards is a Dynamic Link Library (DLL) file named "kvaser+.dll". To allow the OPC Server component to have an access to this file, the component must be registered. This may be completed using the following steps.

1. From the "Start" menu, select "Run…"
2. "Browse" the local hard drive(s) for the folder containing the Kvaser card dependant component (named "kvaser+.dll", which is copied into C:\CANopenOPC in the default installation)
3. Click the button labeled "Open"
4. Edit the text displayed to add the text "regsvr32" BEFORE the name of the DLL (with a space between this text and the full DLL file path and name. e.g. if the Kvaser card dependant component was found in C:\CANopenOPC, the text in the "Run" dialog would be:
   regsvr32 C:\CANopenOPC\kvaser+.dll
5. Click the button labeled "OK"
6. A message box should be displayed indicating the DLL has been successfully registered
7. Click the button labeled "OK" on the message box