INDUSTRIAL CONTROL COMMUNICATIONS, INC.

DeviceNet Slave Driver Manual

May 1, 2016

TABLE OF CONTENTS

1 DeviceNet Slave

1.1 Overview

The gateway supports a DeviceNet slave driver on the DeviceNet port. Some notes of interest are:

- The driver acts as a Group 2 server and supports the Predefined Master/Slave Connection Set.
- COS, Cyclic, and Polled connections are supported with up to 256 bytes transferrable in each direction.
- The Bit Strobe connection is supported with up to 8 bytes of produced data.
- The driver is UCMM capable, and supports the Unconnected Explicit Message Request port, Group 3, Message ID=6.
- Up to 5 simultaneous explicit messaging connections are supported.
- The entire internal database is accessible through explicit messaging using vendor specific CIP objects called Database Access objects.

1.2 I/O Messaging

The driver supports up to 256 bytes each of produced and consumed data. The produced and consumed data array sizes and database mappings are configured using the Configuration Studio (refer to section 1.6.) Change-of-state (COS), cyclic and polled I/O may all be used with these sizes and mappings. The bit strobe connection, however, supports only up to 8 bytes of produced data and does not consume any data; the driver will only use the bit strobe input bit if no other I/O connections are active.

Note that because the driver supports only one common assembly object instance each for production and consumption, the configured I/O data mappings and sizes are used for all I/O connection types. This means that if both polled and cyclic I/O connections are used simultaneously, the data received as consumed data by the master will be the same for both connections. Conversely, the data produced by the master will be written to the same database addresses for both connections, causing data to be overwritten. For this reason, it is recommended that each I/O connection be used independently unless a monitor-only system is desired.

1.3 Explicit Messaging

The driver supports access to any address in the internal database using byte (8-bit), word (16-bit), or double word (32-bit) access via explicit messaging. This is accomplished by targeting vendor-specific CIP objects called “database access objects” (class code 0xA2). If a configuration tool (such as RSNetWorx for DeviceNet™) is used for explicit messaging, the Configuration Studio can be used to configure parameter objects and generate an EDS file containing access definitions for these objects. Once the EDS file is registered, these parameters can then be accessed by the tool.

Note that the driver does not actually support the CIP parameter object class. The parameter objects defined by the Configuration Studio are primarily intended to provide a configuration tool such as RSNetWorx a convenient method of explicit messaging access to the internal database. As defined in the EDS file, these parameter objects merely provide a path to the corresponding database access objects for the configuration tool to target. If direct access to the database via explicit messaging is to be implemented by a master that does not directly use the EDS file (such as via a MSG instruction executing directly on a PLC), then the master must be configured to directly target the database access objects (not parameter objects).

1.3.1 Database Access Object

This section defines the instance definitions, supported services and mapping conventions of the database access object class.

Object Description

Class Code: $A2_{HEX}$

Objects in this class provide access to any internal database address using one of the following three access types:

- Byte access (instances 1…4096) allows 1-byte access for any address in the database.
- Word access (instances 4097…8191) allows 2-byte access for any address in the database.
- Double word access (instances 8193…12285) allows 4-byte access for any address in the database.

Supported Services

Class Services

- Get Attribute Single

Instance Services

- Get Attribute Single
- Set Attribute Single

Class Attributes

Refer to Table 1.

Table 1: Database Access Object Class Attributes

#	Name	Access	Type	Value
1	Revision	Get	UINT	Object Revision (Current Value = 0001h)
2	Max Instance	Get	UINT	12290
3	Number of Instances	Get	UINT	12286

Instance Attributes

Refer to Table 2.

Table 2: Database Access Object Instance Attributes

#	Name	Access	Type	Value
1	Name	Get	SHORT_STRING	Parameter name[1] (including length)
2	Data Type	Get	USINT	4 – Byte 5 – Word 6 – DWord
3	Number Of Elements	Get	USINT	1
4	Descriptor	Get	USINT	Bit field describing the access rights for this instance: Bit ..... Meaning 0 ..... Get Access 1 ..... Set Access
5	Value	Get/Set	Determined by attribute #2	Instance value
6	Max Value	Get	Determined by attribute #2	The maximum permitted parameter value
7	Min Value	Get	Determined by attribute #2	The minimum permitted parameter value
8	Default Value	Get	Determined by attribute #2	The default parameter value

Note 1: The parameter name consists of the Data Type followed by the referenced database address.

Database Mapping

For any given address in the database, the database access object instance number can be determined by Equation 1, Equation 2, or Equation 3 (depending on the desired data size).

Byte Access (1 byte):

$$Instance = address + 1 \qquad \textbf{Equation 1}$$

Word Access (2 bytes):

$$Instance = address + 4097 \qquad \textbf{Equation 2}$$

Double Word Access (4 bytes):

$$Instance = address + 8193 \qquad \textbf{Equation 3}$$

For clarity, let's use the above equations in a calculation example. Say, for instance, we wish to access database address 2846. If we wish to access one byte starting at address 2846, then we must use Equation 1 to calculate the database access object instance associated with that address. We can determine that the instance is 2847, as 2846 + 1 = 2847. Now that we know

the instance, we can formulate an explicit request to read the byte at address 2846 with the following transaction arguments:

- Service Code = Get Attribute Single
- Class = $A2_{HEX}$ (Database Access Object)
- Instance = $B1F_{HEX}$ (2847_{10})
- Attribute = 5 (Value)

Using the same example, if we wish to access one word starting at database address 2846, then we must use Equation 2 to calculate the database access object instance. We can determine that the instance is 6943, as 2846 + 4097 = 6943. The explicit request to read the word at address 2846 will have the following transaction arguments:

- Service Code = Get Attribute Single
- Class = $A2_{HEX}$ (Database Access Object)
- Instance = $1B1F_{HEX}$ (6943_{10})
- Attribute = 5 (Value)

Again, using the same example, if we wish to access one double word starting at database address 2846, then we must use Equation 3 to calculate the database access object instance. The instance is calculated to be 11039, as 2846 + 8193 = 11039. The explicit request to read the double word at address 2846 will have the following transaction arguments:

- Service Code = Get Attribute Single
- Class = $A2_{HEX}$ (Database Access Object)
- Instance = $2B1F_{HEX}$ (11039_{10})
- Attribute = 5 (Value)

1.4 EDS File

When an error-free configuration has been completed, the Configuration Studio can create a corresponding Electronic Data Sheet (EDS) file for registration with a network configuration tool (such as RSNetWorx for DeviceNet™). To generate the EDS file, select the target gateway in the Project panel, and then navigate to **File…Generate EDS File** to specify the file name and location.

1.5 DeviceNet Settings

Baud Rate

Specifies the desired network baud rate (Auto, 125kbaud, 250kbaud or 500kbaud). Selecting Auto allows the driver to automatically detect the active network baud rate.

Address

Assigns the DeviceNet node address of the device (0…63).

1.6 I/O Messaging Settings

I/O messaging support is automatically added to the DeviceNet driver, and cannot be removed.

Produced Start Address

Specifies the starting address in the internal database for the produced I/O data array. The configuration studio will not allow entry of a starting database address that will cause the produced data array to run past the end of the database. The highest valid database address, therefore, will depend on the *Produced Data Size* setting.

Produced Data Size

Specifies the size in bytes of the produced data array (0…256).

Consumed Start Address

Specifies the starting address in the internal database for the consumed I/O data array. The configuration studio will not allow entry of a starting database address that will cause the consumed data array to run past the end of the database. The highest valid database address, therefore, will depend on the *Consumed Data Size* setting.

Consumed Data Size

Specifies the size in bytes of the consumed data array (0…256).

1.7 Parameter Object Settings

This section describes the configurable fields which define a parameter object. Note that this parameter object information is only used for creating an EDS file for a configuration tool (such as RSNetWorx for DeviceNet™): the driver itself does not directly use this information. For more information on parameter objects and explicit messaging, refer to section 1.3.

Object Name

The optional name of the parameter object. Enter a string of up to 16 characters in length.

Database Address

Defines the database address where the parameter object's value will reside. The configuration studio will not allow entry of a database address that will cause the object's value to run past the end of the database. The highest valid database address, therefore, depends on the designated Data Type of the parameter object.

Data Type

Specifies how many bytes are to be allocated for the parameter object's value data, as well as whether the value should be treated as signed or unsigned. Select the desired data type from this dropdown menu.

Note that each data type has its own range limitations: 8-bit can range from 0 to 255 (unsigned) or -128 to 127 (signed), 16-bit can range from 0 to 65535 (unsigned) or -32,768 to 32,767 (signed), and 32-bit can range from 0 to 4,294,967,295 (unsigned) or -2,147,483,648 to 2,147,483,647 (signed).

Automatic Min Value

Checking this checkbox automatically sets the *Min Value* to the smallest value allowed by the selected *Data Type*. Uncheck this checkbox to allow configuration of an alternate *Min Value*.

Min Value

Enabled only when Automatic Min Value is unchecked. Defines the minimum allowable value for the parameter object.

Automatic Max Value

Checking this checkbox automatically sets the *Max Value* to the largest value allowed by the selected *Data Type*. Uncheck this checkbox to allow configuration of an alternate *Max Value*.

Max Value

Enabled only when Automatic Max Value is unchecked. Defines the maximum allowable value for the parameter object.

Default Value

Defines the default value (*Min Value*…*Max Value*) for the parameter object.

Units

The optional engineering units of the parameter object. Enter a string of up to 4 characters in length.

Help String

The optional help string of the parameter object. Enter a string of up to 64 characters in length.

Multiplier, Divisor, Base, Offset

These fields are used in the scaling formulas provided in Equation 4 and Equation 5. "EngValue" is the engineering value that is displayed by the configuration tool with precision specified by the *Decimal Precision* field, and "ActualValue" is the raw value that is stored in the internal database.

$$EngValue = \frac{(ActualValue + Offset)\ x\ Multiplier\ x\ Base}{Divisor}$$ **Equation 4**

$$ActualValue = \frac{(EngValue \times Divisor)}{Multiplier \times Base} - Offset$$ **Equation 5**

Decimal Precision

Specifies the number of decimal places to use when displaying the engineering value in the configuration tool.

Access Rights

Specifies the access rights (Read-Write, Read-only or Write-only) of the parameter object.

ICC

INDUSTRIAL CONTROL COMMUNICATIONS, INC.

1600 Aspen Commons, Suite 210
Middleton, WI USA 53562-4720
Tel: [608] 831-1255 Fax: [608] 831-2045

http://www.iccdesigns.com

Printed in U.S.A