United Electronic Industries

The High-Performance Alternative

DNx-CSDB-509

—

User Manual

8-port CSDB or RS-232/RS-422/485 serial communications interface for the PowerDNA Cube or RACK series chassis

August 2017

PN Man-DNx-CSDB-509

CE

Contacting United Electronic Industries

Mailing Address:

27 Renmar Avenue
Walpole, MA 02081
U.S.A.

For a list of our distributors and partners in the US and around the world, please contact our support team:

Support:

Telephone: (508) 921-4600
Fax: (508) 668-2350

Also see the FAQs and online “Live Help” feature on our web site.

Internet Support:

Support: support@ueidaq.com
Website: www.ueidaq.com
FTP Site: ftp://ftp.ueidaq.com

Product Disclaimer:

WARNING!

DO NOT USE PRODUCTS SOLD BY UNITED ELECTRONIC INDUSTRIES, INC. AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS.

Products sold by United Electronic Industries, Inc. are not authorized for use as critical components in life support devices or systems. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness. Any attempt to purchase any United Electronic Industries, Inc. product for that purpose is null and void and United Electronic Industries Inc. accepts no liability whatsoever in contract, tort, or otherwise whether or not resulting from our or our employees' negligence or failure to detect an improper purchase.

Specifications in this document are subject to change without notice. Check with UEI for current status.

Table of Contents

Table of Figures

Chapter 1 Introduction

This document outlines the feature set and use of the DNx-CSDB-509 serial communication boards.

CSDB-509 boards are 8-port serial communication interfaces for the Cube and RACK chassis. The CSDB-509 provides fully isolated software-configurable Commercial Standard Digital Bus (CSDB), RS-232, or RS-485 interfaces.

This chapter contains the following sections:

- Organization of this Manual (Section 1.1)
- CSDB-509 Board Overview (Section 1.2)
- Features (Section 1.3)
- Indicators (Section 1.4)
- Specification (Section 1.5)
- Overview of CSDB (Section 1.6)
- Overview of RS-232, RS-485, & RS-422 Serial Interfaces (Section 1.7)
- Device Architecture (Section 1.8)
- Module Capabilities (Section 1.9)
- Wiring & Connections (pinouts) (Section 1.10)
- Jumper Settings for CSDB-509 Boards for the Cube (Section 1.11)

1.1 Organization of this Manual

This DNx-CSDB-509 User Manual is organized as follows:

- **Introduction**
  Chapter 1 provides an overview of the DNx-CSDB-509 serial communication features, various models available, device architecture, connectivity and logic.
- **Programming with the High-Level API**
  Chapter 2 provides an overview of the how to create a session, configure the session, and format relevant data with the Framework API.
- **Programming with the Low-Level API**
  Chapter 3 describes low-level API commands for configuring and using the DNx-CSDB-509 series boards.
- **Appendix A - Accessories**
  Appendix A provides a list of accessories available for DNx-CSDB-509 board(s) including the DNA-STP-508 accessory screw terminal panel, which serves as a convenient connection interface between the 62-pin DB connector on the module and the individual cables for each of the eight serial lines.
- **Index**
  This is an alphabetical listing of the topics covered in this manual.

NOTE: A glossary of terms used with the PowerDNA Cube/RACK and I/O boards can be viewed or downloaded from www.ueidaq.com.

Manual Conventions

To help you get the most out of this manual and our products, please note that we use the following conventions:

Tips are designed to highlight quick ways to get the job done or to reveal good ideas you might not discover on your own.

NOTE: Notes alert you to important information.

CAUTION! *Caution advises you of precautions to take to avoid injury, data loss, and damage to your boards or a system crash.*

Text formatted in **bold** typeface generally represents text that should be entered verbatim. For instance, it can represent a command, as in the following example: “You can instruct users how to run setup using a command such as **setup.exe**.”

Bold typeface will also represent field or button names, as in “Click **Scan Network**.”

Text formatted in `fixed` typeface generally represents source code or other text that should be entered verbatim into the source code, initialization, or other file.

Examples of Manual Conventions

Before plugging any I/O connector into the Cube or RACKtangle, be sure to remove power from all field wiring. Failure to do so may cause severe damage to the equipment.

Usage of Terms

Throughout this manual, the term “Cube” refers to either a PowerDNA Cube product or to a PowerDNR RACKtangle™ rack mounted system, whichever is applicable. The term DNR is a specific reference to the RACKtangle, DNA to the PowerDNA I/O Cube, and DNx to refer to both.

1.2 CSDB-509 Board Overview

The DNx-CSDB-509 boards are 8-port serial communications interfaces for Cube and RACK I/O chassis. The boards offer fully isolated software-configurable Commercial Standard Digital Bus (CSDB), RS-232 or RS-485 interfaces. The board is an ideal interface to serial based avionics as well as general purpose serial I/O.

1.2.1 Baud Rates

DNx-CSDB-509 boards support both 12.5 kbaud and 50 kbaud data rates in CSDB mode as well as transfer rates up to 1 Mbaud in RS-485 mode or up to 256 kbaud in RS-232 mode. It also supports communications at 12, 12.5 and 50 kbaud with better than 0.1% data rate accuracy. Baud rates based on integer divisors of 4.125 MHz or 1.8415 MHz are also supported.

1.2.2 Interface Modes

Based upon an industry standard UART on each port, the board supports both half- and full-duplex modes for RS-422/485 operation.

The DNx-CSDB-509 boards are compatible with a wide variety of legacy avionics devices. The boards support RS-422 point to point or network applications when used in RS-485 mode and provide 200Ω software selectable TX and/or RX termination for RS-485 communications. In RS-232 mode, the board provides TX/RX as well as the standard RTS and CTS hardware hand shaking required by many external serial devices.

1.2.3 Accessories

All connections to the CSDB-509 are through a 62-pin dSub connector. However, the DNA-CBL-62 series cable and the DNA-STP-508 breakout board can optionally be used to bring out each port to easy-to-use 9-pin dSubs.

1.2.4 Software Support

The DNx-CSDB-509 product includes a custom UEI CSDB library that implements API for configuring one or more CSDB-509 serial ports as CSDB ports. The CSDB API sits on top of the PowerDNA low-level API.

The DNx-CSDB-509 is supported by all the popular operating systems and programming languages with a powerful, yet easy to use API. Window users may also take advantage of the UEIDAQ Framework which provides an extremely simple and complete software interface for programmers as well as supporting most data acquisition and control applications, (e.g. LabVIEW, MATLAB).

1.3 Features

The following is a summary of DNx-CSDB-509 features:

- Eight (8) independent ports
- Each port software-configurable as CSDB, RS-232, or RS-422/485
- Supports 12.5 kbit/s and 50 kbit/s baud rates in CSDB mode
- Configurable to a maximum speed of 256 kbit/s for RS-232 and 1 Mbit/s for RS-422/485
- Completely independent bit rate settings for every port
- 350 V isolation between ports, and between ports and circuitry; 15 kV ESD
- Compatible with RS-422 networks when used in RS-485 mode
- Half- and full-duplex support for RS-485

1.4 Indicators

The DNx-CSDB-509 indicators are described in **Table 1-1** and illustrated in **Figure 1-1**.

Table 1-1 CSDB-509 Indicators

LED Name	Description
RDY	Indicates board is powered up and operational
STS	Indicates which mode the board is running in: • **OFF**: Configuration mode, (e.g., configuring channels, running in point-by-point mode) • **ON**: Operation mode, (e.g., running in VMap mode)

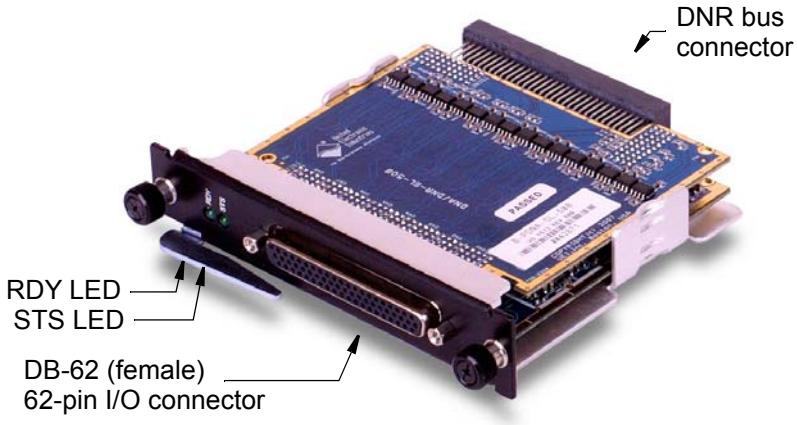


Figure 1-1 Photo of the DNR-CSDB-509 Board

1.5 Specification

The technical specification for the DNx-CSDB-509 is provided in the table below:

Table 1-2 . DNx-CSDB-509 Technical Specifications

General Specifications	
Number ports	8
Serial Interfaces	CSDB, RS-232, RS-422/485, per-port software-configurable
Supported baud rates: CSDB RS-232 RS-422/485	 12.5 kbaud or 50 kbaud Standard rates up to 250 kbaud Standard rates to 1 Mbaud
Supported Baud rates	Integer divisors of 4.124 MHz or 1.8415 Mhz
RS-422/485 Modes:	Half- and full- duplex
Hardware Transceiver	MAX3160E (w/ fail-safe RX termination)
UART Controller	16550C FPGA emulation
UART Base Clock	66 MHz or 24 MHz
FIFO Size	1024 (input and output)
Protection	350 V chan-to-chan isolation; 15 kV ESD protection. Does not meet CSDB 115VAC protection specification
Power Consumption	2-5W (CSDB mode with max current drive)
Operating Temperature	Tested -40 to +85 °C
Operating Humidity	95%, non-condensing
Vibration *IEC 60068-2-6* *IEC 60068-2-64*	5 g, 10-500 Hz, sinusoidal 5 g (rms), 10-500 Hz, broad-band random
Shock *IEC 60068-2-27*	50 g, 3 ms half sine, 18 shocks @ 6 orientations 30 g, 11 ms half sine, 18 shocks @ 6 orientations
MTBF	290,000 hours

1.6 Overview of CSDB

The Commercial Standard Digital Bus (CSDB) is a unidirectional asynchronous bus, formerly known as the Collins Standard Digital Bus.

Data is transmitted over an interconnect cable using devices compliant with the RS-422A standard. RS-422A dictates that only one device can transmit and up to ten devices can receive.

The CSDB standard conforms to the following specifications:

- Two bus speeds are supported: low bus speed 12,500 bps (+-0.1%) and high bus speed 50,000 bps (+-0.1%).
- Data is transmitted using the NRZ (non return to zero) format with a logic "1" positive and "0" negative. Start bits are logic "0" and stop bits logic "1".
- Data is sent in frames consisting of a synchronization block followed by a number of message blocks.
- A frame is defined from the start of a sync block to the start of the next sync block.
- A sync block consists of the sync character (0xA5) repeated N times.
- A message block consists of an address byte, followed by a status byte, followed by 6 or 8 data bytes.
- Bytes within the sync or message blocks are spaced in time by a fixed interbyte idle delay.
- Blocks are spaced in time by a fixed interblock idle delay.
- Frames are transmitted at a fixed rate.

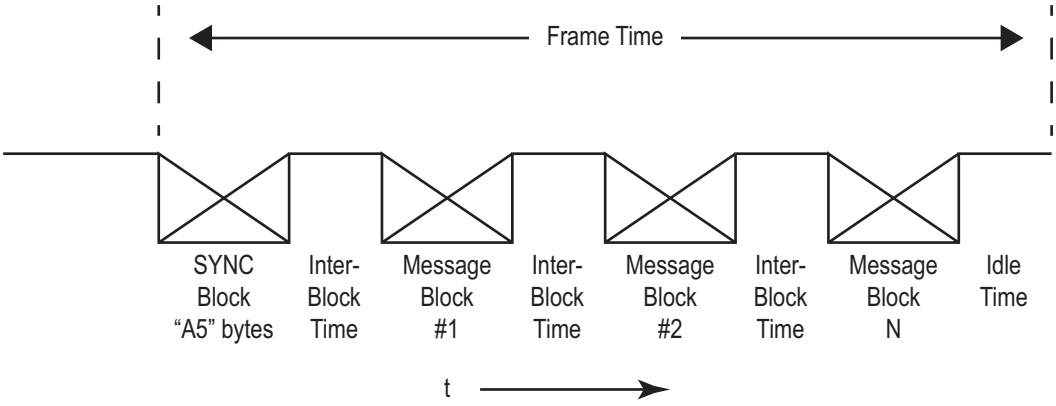


Figure 1-2 CSDB Message Protocol

1.6.1 CSDB-509 CSDB Support

The DNx-CSDB-509 is a modified version of the DNx-SL-508 that includes fine control of the interbyte and interblock delay.

The CSDB-509 supports RS-485 full duplex which is electrically equivalent to RS-422 but allows for one driver and up to 32 receivers (instead of one transmitter for 10 receivers on RS-422).

Refer to **Chapter 3** for more information regarding the CSDB library and low-level API implementation. Refer to **Chapter 2** for CSDB high-level Framework implementations.

1.7 Overview of RS-232, RS-485, & RS-422 Serial Interfaces

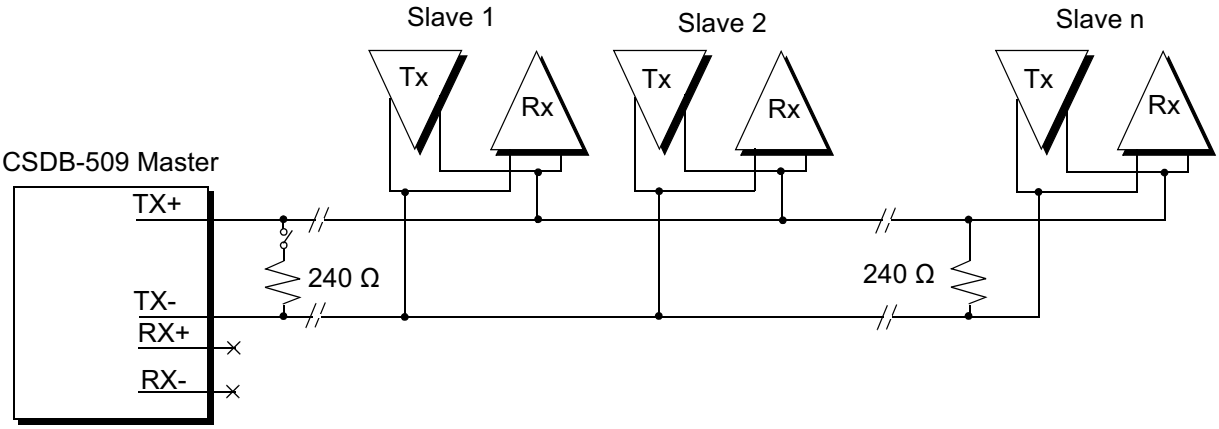
Serial ports send and receive data one bit at a time over a serial line (composed of a send, a receive, and one common ground wire).

RS-232 is a standard for serial binary data interconnection between a DTE (Data terminal equipment) and a DCE (Data communication equipment) and normally operates in a bipolar range of -10V to 10V.

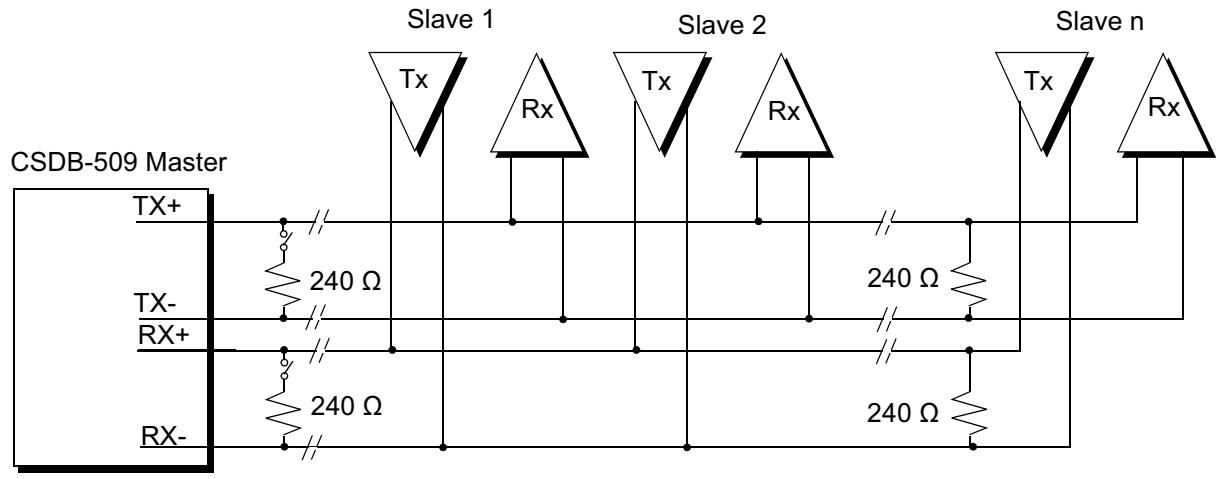
RS-485 (i.e., EIA-485) is a physical electrical specification of a two-wire, half-duplex, multipoint serial connection. A full duplex RS-485 system can be constructed by using two twisted-pair connections (transmit/receive pairs) together as shown in **Figure 1-3**.

UART data frames for RS-232 and RS-485 are shown in **Figure 1-4**.

RS- is an abbreviation for "Recommended Standard".



Two-wire Twisted-Pair Half Duplex Network with 240 Ω Terminating Resistors



Four-wire Twisted-Pair Full Duplex Network with 240 Ω Terminating Resistors

Figure 1-3 RS-485 Topologies

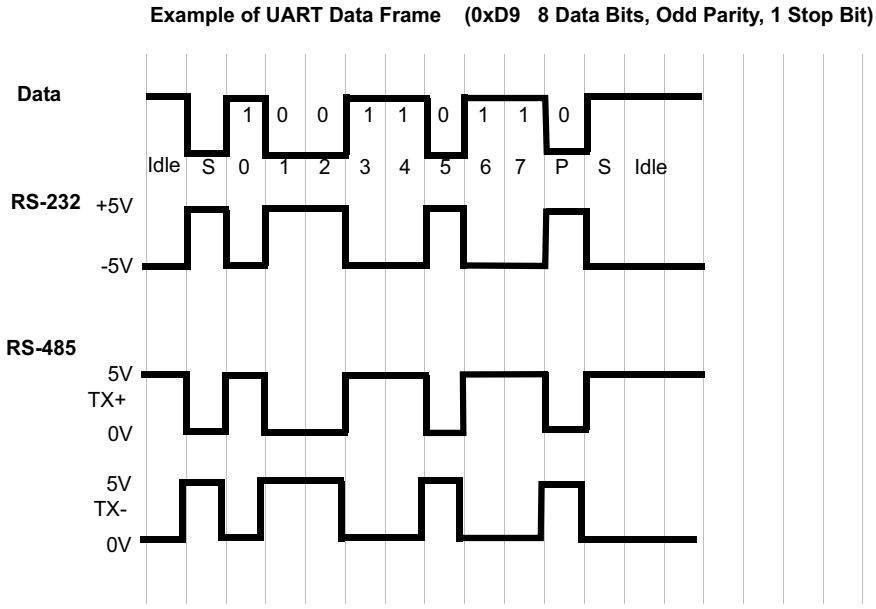


Figure 1-4 UART Data Frames for RS-232 and RS-485

1.8 Device Architecture

The architecture of the DNx-CSDB-509 is illustrated in the block diagram shown in **Figure 1-5**.

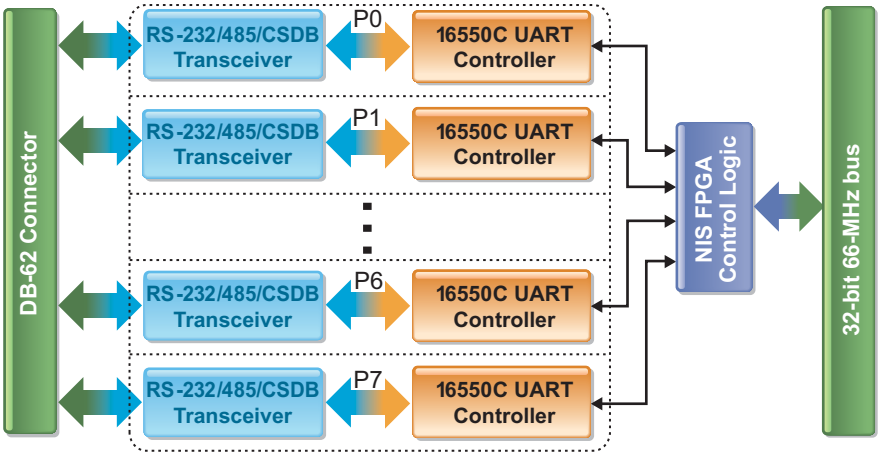


Figure 1-5 Logic Block Diagram: DNx-CSDB-509 Overview

Serial electrical impulses are received by and transmitted to a MAX3106E serial transceiver chip via the DB-62 connector. The MAX3106E is a multiprotocol transceiver that implements the RS-232/RS-485/RS-422 protocols by handling transmission/reception between the serial line and the UART16550.

The transceiver and controller are isolated from each other by a high-speed isolation integrated circuit (IC) capable of withstanding 350 V channel-to-channel or 15 kV ESD. There are eight MAX3106E » isolation » UART16550 structures, one per port; isolation is per-port.

The UART16550 is in turn controlled by a FPGA Control Chip, the board control chip. FPGA works in conjunction with the core module logic of the DNx chassis.

The CSDB-509 board consists of two PC boards, one of which is a 60x Base Board and another, which is the eight port CSDB-509-specific I/O board.The 509-specific board plugs into a bus connector on the base board.

1.9 Module Capabilities

The CSDB-509 transmits and receives data compliant with the RS-422A standard. When configured as a CSDB port, CSDB-509 supports two bus speeds: low bus speed 12,500 bps and high bus speed 50 kbit/s.

If the CSDB-509 port is used as a standard serial device using the RS-232 or RS-485 standard (non-CSDB mode), the controller is capable of communicating at speeds up to 256 kbit/s for RS-232 and 1 Mbit/s for RS-485. When in RS-485 mode, the CSDB-509 is compatible with RS-422 networks.

The UART16550 runs at a base-clock frequency of 66MHz, with a FIFO size of 1024 bytes.

Each port has independently programmable when used as a standard serial device:

- Baud/bit rate
- UART interrupt
- Timeout interrupt
- TX/RX FIFO interrupt
- Error interrupts (4 per port)

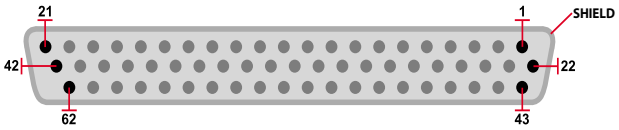
1.10 Wiring & Connections (pinouts)

The following signals are located at the DB-62 connector on the CSDB-509 board:

- GNDn - Isolated ground for the corresponding serial port
- TXDn/RXDn RS-232: Transmit/Receive
- RTSn/CTSn RS-232: Request to Send / Clear to Send
- TXn+/TXn- RS-422/485: Transmit pair
- RXn+/RXn- RS-422/485: Receive pair

The 62-pin female D-Sub connector on the CSDB-509 is divided into eight 9-pin serial ports as shown in the pinout of **Figure 1-6**.

A user can connect eight serial lines to this connector either through a custom made cable or by connecting to a DNA-STP-508 accessory panel, as described in “Preferred Configuration Using an Accessory Terminal Panel” on page 11.



Pin	232 signal	CSDB/ 422/485 signal	Pin	232 signal	CSDB/ 422/485 signal	Pin	232 signal	CSDB/ 422/485 signal
1	-	-	22	Gnd1	Gnd1	43	CTS1	RX1-
2	RTS1	TX1+	23	TX1	TX1-	44	RX1	RX1+
3	-	-	24	-	-	45	Gnd2	Gnd2
4	RX2	RX2+	25	CTS2	RX2-	46	-	-
5	RTS2	TX2+	26	TX2	TX2-	47	-	-
6	Gnd3	Gnd3	27	-	-	48	-	-
7	RX3	RX3+	28	CTS3	RX3-	49	-	-
8	RTS3	TX3+	29	TX3	TX3-	50	-	-
9	RX4	RX4+	30	CTS4	RX4-	51	Gnd4	Gnd4
10	RTS4	TX4+	31	TX4	TX4-	52	-	-
11	-	-	32	-	-	53	CTS5	RX5-
12	RTS5	TX5+	33	TX5	TX5-	54	RX5	RX5+
13	-	-	34	Gnd5	Gnd5	55	Gnd6	Gnd6
14	RX6	RX6+	35	CTS6	RX6-	56	-	-
15	RTS6	TX6+	36	TX6	TX6-	57	-	-
16	Gnd7	Gnd7	37	-	-	58	-	-
17	RX7	RX7+	38	CTS7	RX7-	59	-	-
18	RTS7	TX7+	39	TX7	TX7-	60	-	-
19	-	-	40	Gnd8	Gnd8	61	CTS8	RX8-
20	RTS8	TX8+	41	TX8	TX8-	62	RX8	RX8+
21	-	-	42	-	-			

Gndn	Isolated ground for the corresponding Serial Port “n”
TXn/**RX**n	RS-232 Transmit/Receive, Port n
RTSn/**CTS**n	RS-232: Request To Send / Clear To Send
TXn+/**TX**n-	RS-485: Transmit Pair, Port n
RXn+/**RX**n-	RS-485:Receive Pair, Port n
—	No Internal Connection

Figure 1-6 DNx-CSDB-509 Pinout Diagram

1.10.1 Preferred Configuration Using an Accessory Terminal Panel

Figure 1-7 shows a DNA-CBL-62 62-conductor cable, which is designed to connect the CSDB-509 board to a DNA-STP-508 accessory terminal panel.

The DNA-STP-508 accessory panel provides a convenient interface for connecting the eight serial cables for the serial ports to the CSDB-509 board. The panel accepts eight 9-pin DB-9 connectors and also has eight screw terminal blocks that may be used for individual wire connections, if preferred.

The eight DB-9 cable connectors are labeled JS1 to JS8 and the eight 5-terminal screw terminal blocks are labeled JT1 through JT8, as shown in **Figure 1-7**.

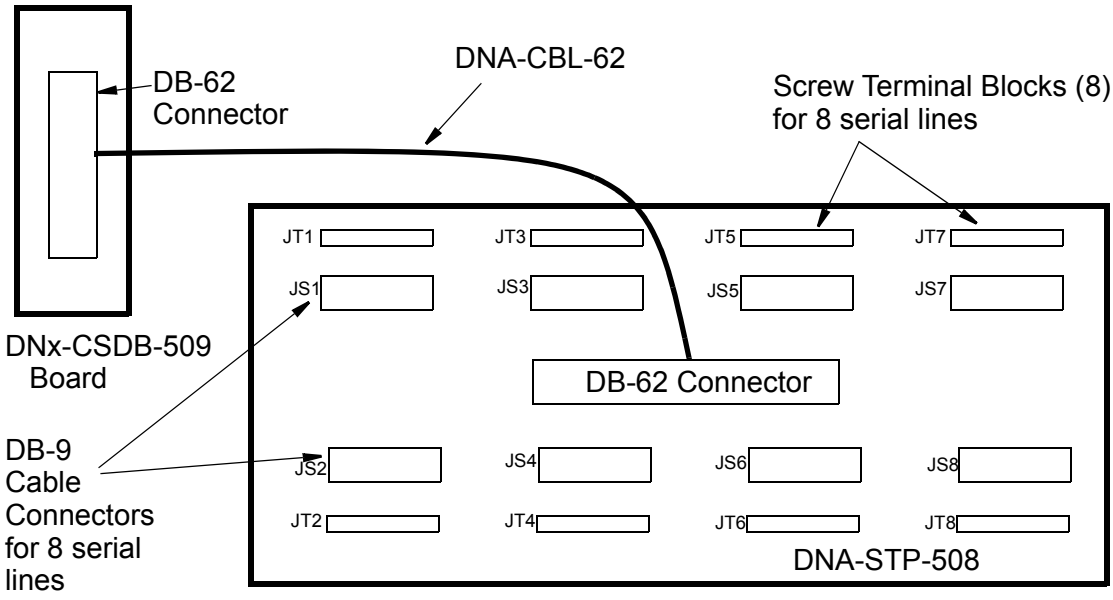


Figure 1-7 DNA-STP-508 Screw Terminal Panel Connections

1.10.2 Pinout of DNA-STP-508 Panel

The following diagram shows the pinout connections for the DNA-STP-508 panel.

DB-62 Pin No.	232 Signal	485 Signal	DB-9 Pin No.	DB-62 Pin No.	232 Signal	485 Signal	DB-9 Pin No.	DB-62 Pin No.	232 Signal	485 Signal	DB-9 Pin No.	DB-9 Connector
1	–	–		22	Gnd1	Gnd1	5	43	CTS1	RX1–	8	JS1
2	RTS1	TX1+	7	23	TX1	TX1–	3	44	RX1	RX1+	2	
3	–	–		24	–	–		45	Gnd2	Gnd2	5	
4	RX2	RX2+	2	25	CTS2	RX2–	8	46	–	–		JS2
5	RTS2	TX2+	7	26	TX2	TX2–	3	47	–	–		
6	Gnd3	Gnd3	5	27	–	–		48	–	–		
7	RX3	RX3+	2	28	CTS3	RX3–	8	49	–	–		JS3
8	RTS3	TX3+	7	29	TX3	TX3–	3	50	–	–		
9	RX4	RX4+	2	30	CTS4	RX4–	8	51	Gnd4	Gnd4	5	
10	RTS4	TX4+	7	31	TX4	TX4-	3	52	–	–		JS4
11	–	–		32	–	–						
12	RTS5	TX5+	7	33	TX5	TX5–	3	53	CTS5	RX5–	8	JS5
13	–	–		34	Gnd5	Gnd5	5	54	RX5	RX5+	2	
14	RX6	RX6+	2	35	CTS6	RX6–	8	55	Gnd6	Gnd6	5	JS6
15	RTS6	TX6+	7	36	TX6	TX6–	3	56				
								57	–	–		
16	Gnd7	Gnd7	5	37	–	–		58	–	–		JS7
17	RX7	RX7+	2	38	CTS7	RX7–	8	59	–	–		
18	RTS7	TX7+	7	39	TX7	TX7–	3	60	–	–		
19	–	–		40	Gnd8	Gnd8	5	61	CTS8	RX8–	8	
20	RTS8	TX8+	7	41	TX8	TX8–	3	62	RX8	RX8+	2	JS8
21	–	–		42	–	–						

1.11 Jumper Settings for CSDB-509 Boards for the Cube

The base board of a DNA-CSDB-509 has a jumper block that assigns the position of the module within a PowerDNA Cube. The jumpers must be set to match the physical position of an I/O board of CSDB-509 board in the Cube.

This function is not required with DNR/F (RACK) board versions.

NOTE: Since all CSDB-509s are assembled in Cubes before shipment to a customer, you should never have to change a jumper setting unless you change a CSDB-509 from one position to another in the field.

A diagram of the jumper block is shown in **Figure 1-8**. To set the CSDB-509 address, place jumpers as shown.

		Layer's Position as marked on the Faceplate*					
		I/O **1**	I/O **2**	I/O **3**	I/O **4**	I/O **5**	I/O **6**
Jx Pins	9-10	● ○	○ ○	● ○	○ ○	● ○	○ ○
	11-12	● ○	● ○	○ ○	○ ○	● ○	● ○
	13-14	● ○	● ○	● ○	● ○	○ ○	○ ○
	15-16	● ○	● ○	● ○	● ○	● ○	● ○

* *All I/O Layers are sequentially enumerated from top to the bottom of the Cube*

○ ○ – **Open** ● ○ – **Closed**

Figure 1-8 Diagram of DNA-CSDB-509 Layer Position Jumper Settings

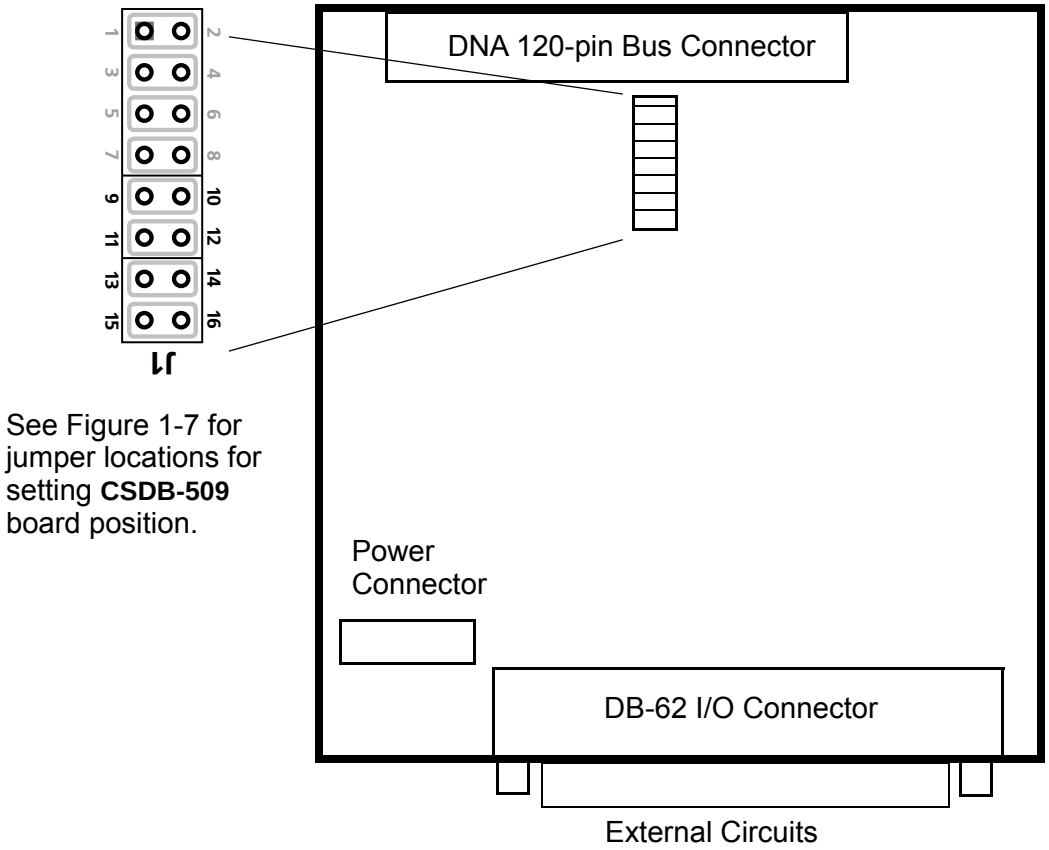


Figure 1-9 Physical Layout of DNA-CSDB-509 Base Board (60x)

Chapter 2 Programming with the High-level API

This chapter provides the following information about using the UeiDaq high-level Framework API to control the DNx-CSDB-509.

- About the High-level Framework (Section 2.1)
- Using High-level API to Program CSDB Ports (Section 2.2)
- Using High-level API to Program Standard Serial Ports (Section 2.3)

2.1 About the High-level Framework

UeiDaq Framework is object oriented and its objects can be manipulated in the same manner from different development environments, such as Visual C++, Visual Basic, or LabVIEW.

UeiDaq Framework is bundled with examples for supported programming languages. Examples are located under the UEI programs group in:

- *Start » Programs » UEI » Framework » Examples*

The following sections in this chapter focus on C++ API examples, but the concept is the same no matter what programming language you use.

Please refer to the *UeiDaq Framework User Manual* for more information on use of other programming languages.

2.2 Using High-level API to Program CSDB Ports

The following sections describe programming CSDB ports using the high-level Framework.

2.2.1 Creating a CSDB Session

The Session object controls all operations on your PowerDNA device. Therefore, the first task is to create a session object:

```
CUeiSession session;
```

2.2.2 Configuring the CSDB Ports

The Framework uses resource strings to select which device, subsystem, and channels to use within a session. The resource string syntax is similar to a web URL:

```
<device class>://<IP address>/<Device Id>/
<Subsystem><Channel list>
```

For PowerDNA, the device class is **pdna**.

For example, the following resource string selects serial ports 0,2,3 on device 1 at IP address 192.168.100.2:

```
pdna://192.168.100.2/Dev1/CSDB0,2,3
```

In addition to the resource, you will also configure:

- Bit rate (bits per second): 12500 or 50000
- Parity (odd or even): 0 for even, any value > 0 for odd
- The CSDB message block size: the number of bytes in a message block including the address and status bytes
- The number of CSDB message blocks per frame
- The inter-byte delay within a message block in microseconds
- The inter-block delay within a frame in microseconds
- The period at which the frame is transmitted in microseconds

```
mySession.CreateCSDBPort(
                  "pdna://192.168.100.3/Dev1/CSDB0,1",
                  bps,
                  parity,
                  blockSize,
                  numMessagesPerFrame,
                  interByteDelay,
                  interBlockDelay,
                  framePeriod);
```

2.2.3 Configuring the CSDB Timing

You need to configure the CSDB-509 to use the “messaging” timing mode.

When programming messaging timing, you typically need to pass a `bufferSize` value and a `refreshRate` value.

The message I/O parameters are ignored when setting up messaging for CSDB ports because these parameters are set by the protocol:

```
session.ConfigureTimingForMessagingIO(1, 0.0);
```

2.2.4 Reading CSDB Data

CSDB data is represented by the structure `tUeiCSDBMessage`:

```
typedef struct
{
    /// address byte of the message
   unsigned char address;

    /// status byte of the message
   unsigned char status;

    /// number of meaningful data bytes in payload
    ///    (block size – 2)
   unsigned char dataSize;

 /// payload bytes of the message.
   unsigned char data[10];

} tUeiCSDBMessage;
```

Reading data from the CSDB-509 is done using a reader object. As there is no multiplexing of data (contrary to what's being done with AI, DI, or CI sessions), you need to create one reader object per serial port to be able to read from each port in the port list.

The CSDB frame is constantly being transmitted by the transmitter node. The reader object returns data from the most recent CSDB frame.

The reader object can either read the entire CSDB frame as an array of `tUeiCSDBMessage`, or it can read a single message block specified by its index in the frame.

The following sample code shows how to create a reader object tied to port 1 and read the entire frame and a single message block.

```
// Create a reader and link it to the session's stream, port 1
reader = new CUeiCSDBReader(session.GetDataStream(), 1);

// Read up to 10 messages,
//   numMessagesRead contains the number of messages
//   actually received which can't be higher than
//   the number of messages per frame.
tUeiCSDBMessage frame[10];
reader->ReadFrame(10, frame, &numMessagesRead);

// Read message block at index 1
tUeiCSDBMessage block;
reader->ReadMessageByIndex(1, &block);
```

2.2.5 Writing CSDB Data

Writing data to the CSDB-509 is done using a writer object. As there is no multiplexing of data (contrary to what's being done with AI, DI, or CI sessions), you need to create one writer object per serial port to be able to write from each port in the port list.

The CSDB frame is constantly being transmitted by the transmitter node. The writer object updates data which will be transmitted on the next CSDB frame.

The writer object can either update the entire CSDB frame as an array of `tUeiCSDBMessage` or it can update a single message block specified by its index in the frame.

The following sample code shows how to create a writer object tied to port 0 and write the entire frame and a single message block.

```
// Create a writer and link it to the session's stream, port 0
writer = new CUeiCSDBWriter(session.GetDataStream(), 1);


// Write up to 10 messages,
//   numMessagesWritten contains the number of messages
//   actually written which can't be higher than the
//   number of messages per frame.
tUeiCSDBMessage frame[10];
writer->WriteFrame(10, frame, &numMessagesWritten);


// Update message block at index 1
tUeiCSDBMessage block;
writer->WriteMessageByIndex(1, &block);
```

2.2.6 Cleaning-up the CSDB Session

The session object cleans itself up when it goes out of scope or when it is destroyed. However, you can manually clean up the session (to reuse the object with a different set of channels or parameters).

```
session.CleanUp();
```

2.3 Using High-level API to Program Standard Serial Ports

The CSDB-509 can be used as a standard serial device, with the same functionality as the SL-508 board. The following sections provide details about programming the CSDB-509 using UEI's high-level Framework API; however, note that Section 2.3.1 thru Section 2.3.6 are not supported when using the CSDB protocol.

The CSDB API sits on top of the PowerDNA low-level API. The following examples only use the PowerDNA library, not the UEI CSDB library.

2.3.1 Creating a Standard Serial Session

The Session object controls all operations on your PowerDNA device. Therefore, the first task is to create a session object:

```
CUeiSession session;
```

2.3.2 Configuring Standard Serial Ports

Framework uses resource strings to select which device, subsystem, and channels to use within a session. The resource string syntax is similar to a web URL:

```
<device class>://<IP address>/<Device Id>/
<Subsystem><Channel list>
```

For PowerDNA, the device class is **pdna**.

For example, the following resource string selects serial ports 0,2,3 on device 1 at IP address 192.168.100.2:

```
pdna://192.168.100.2/Dev1/Com0,2,3
```

In addition to the resource, you will also configure:

- Port mode (RS-232, RS-485 half-duplex or RS-485 full duplex)
- Bit rate (bits per second)
- Number of data bits
- Parity
- Number of stop bits

```
// Configure Com ports 0, 2 and 3 on device 1
   session.CreateSerialPort(
                  “pdna://192.168.100.2/Dev1/Com0,2,3”,
                  UeiSerialModeRS232,
                  UeiSerialBitsPerSecond57600,
                  UeiSerialDataBits8,
                  UeiSerialParityNone,
                  UeiSerialStopBits1);
```

2.3.3 Configuring Standard Serial Timing

You need to configure the CSDB-509 to use the “messaging” timing mode. A message is represented by an array of bytes:

When the CSDB-509 is used as a standard serial device, it can be programmed to wait for a certain number of bytes to be received before notifying the session.

It is also possible to program the maximum amount of time to wait for the specified number of bytes before notifying the session.

The following sample shows how to configure the messaging I/O mode in standard serial mode to be notified when 10 bytes have been received or every second, whichever is less. (Note that if the serial port receives fewer than 10 bytes per second, it will return whatever number of bytes are available every second).

```
session.ConfigureTimingForMessagingIO(10, 1.0);
```

2.3.4 Reading Standard Serial Data

Reading data from the CSDB-509 is done using a reader object. As there is no multiplexing of data (contrary to what’s being done with AI, DI, or CI sessions), you need to create one reader object per serial port to be able to read from each port in the port list.

The following sample code shows how to create a reader object tied to port 1 and read up to 10 bytes from a standard serial port.

```
// Create a reader and link it to the session’s stream, port 1
   reader = new CUeiSerialReader(session.GetDataStream(), 1);
// read up to 10 bytes, numBytesRead contains the
// number of bytes actually received.
   Unsigned char bytes[10];
   reader->Read(10, bytes, &numBytesRead);
```

2.3.5 Writing Standard Serial Data

Writing data to the CSDB-509 is done using a writer object. As there is no multiplexing of data (contrary to what’s being done with AO, DO, or CO sessions), you need to create one writer object per serial port to be able to write to each port in the port list.

The following sample code shows how to create a writer object tied to port 2 and send one byte to a standard serial port.

```
// Create a writer and link it to the session’s stream, port 2
   writer = new CUeiSerialWriter(session.GetDataStream(), 2);

// Write 1 byte, numBytesWritten contains the
// number of bytes actually sent
   unsigned char bytes[2] = {0x23, 0};
   writer->Write(1, bytes, &numBytesWritten);
```

2.3.6 Cleaning-up the Standard Serial Session

The session object cleans itself up when it goes out of scope or when it is destroyed. However, you can manually clean up the session (to reuse the object with a different set of channels or parameters).

```
   session.CleanUp();
```

Chapter 3 Programming with the Low-level API

This chapter provides the following information about programming the DNx-CSDB-509 using the low-level API:

- About the Low-level API (Section 3.1)
- UEI CSDB Library (Section 3.2)
- CSDB Low-level Functions (Section 3.3)
- CSDB Low-level Programming Examples (Section 3.4)
- Standard Serial (non-CSDB) Programming Examples (Section 3.5)

3.1 About the Low-level API

UEI's low-level API provides direct access to the DAQBIOS protocol structure and registers in C. The CSDB-509 requires both the PowerDNx libraries and a custom CSDB library for CSDB implementations.

The CSDB API sits on top of the PowerDNA low-level API. User code must call the PowerDNA API to obtain a handle on the hardware and use that handle when calling the CSDB API.

3.2 UEI CSDB Library

The UEI CSDB library consists of custom API to configure one or more CSDB-509 serial ports as CSDB ports. The CSDB API periodically transmits CSDB frames and parses incoming frames.

The UEI CSDB library is implemented as a static library that will need to be included in your project:

For Windows systems, the UEI CSDB library is named "ueicsdb.lib":

- Add ueicsdb.lib to your Visual Studio project to call the CSDB API from your application.

For Linux systems, the UEI CSDB library is named "libueicsdb.a":

- Add "`-l lueicsdb`" to your compiler command line to call the CSDB API from your application.

3.3 CSDB Low-level Functions

Table 3-1 provides a summary of low-level CSDB-509-specific functions. CSDB functions are described in detail in the Section 3.3.1 thru Section 3.3.8 below.

Table 3-1 Summary of Low-level API Functions for DNx-CSDB-509

Function	Description
`UeiCSDBConfigureChannel`	Configures a CSDB port (device channel)
`UeiCSDBConfigureFrame`	Configures the CSDB frame transmitted on a given channel
`UeiCSDBGetStatus`	Gets last comm error that occurred on a given device
`UeiCSDBInitialize`	Initializes internal buffers and variables
`UeiCSDBReadRxFrame`	Reads an array containing the message blocks received in the most recent RX frame
`UeiCSDBSetTxFrame`	Sets the address, status and data bytes of a message block
`UeiCSDBStart`	Starts all configured CSDB channels
`UeiCSDBStop`	Stops all configured CSDB channels

3.3.1 UeiCSDBConfigureChannel()

```
int UeiCSDBConfigureChannel(int hd, int devn, int channel, int bps, int
parity)
```

Parameters:

`hd`	handle to the IOM where the serial device is located
`devn`	ID of the serial device (zero based index)
`channel`	ID of the channel to configure (zero based index)
`bps`	channel speed in bits per second: possible values are 12.5 kbps and 50 kbps
`parity`	0 for even, 1 for odd (see NOTE below)

Returns:

Status code: 0 upon success, negative upon error. **Table 3-2** on page 24 lists all CSDB-509-specific status codes.

Description:

Configures a serial port on a given device to work as a CSDB channel.

Note:

Parity is used for controlling the integrity of TX/RX data.

- **RX**: The CSDB-509 calculates the parity of received message characters based on the `parity` parameter setting for a specific channel. The integrity of the received data byte is determined by comparing the calculated parity with the parity bit read in the incoming message.
- **TX**: The CSDB-509 calculates the parity based on the data to be transmitted and based on the `parity` parameter for this channel. The calculated parity bit is transmitted with the data.

Parity is calculated as follows:

- For Odd parity, the parity bit will be set to a 1 or 0 to make the total number of ONES in the data byte odd.
- For Even parity, the parity bit is set to make the total number of ONES even.

3.3.2 UeiCSDBConfigureFrame()

```
int UeiCSDBConfigureFrame (int hd, int devn, int channel, int numMessages,
int blockSize, int interByteDelayUs, int interBlockDelayUs, int
framePeriodUs)
```

Parameters:

`hd`	handle to the IOM where the serial device is located
`devn`	ID of the serial device (zero based index)
`channel`	ID of the channel to configure (zero based index)
`numMessages`	number of message blocks to transmit
`blockSize`	number of bytes in message blocks (including address and status bytes): usually 6 or 8.
`interByteDelayUs`	idle time between each byte in micro seconds
`interBlockDelayUs`	idle time between each message block in micro seconds
`framePeriodUs`	period of the frame in microseconds, this value must be greater than the time taken to transmit the frame

Returns:

Status code: 0 upon success, negative upon error. **Table 3-2** on page 24 lists all CSDB-509-specific status codes.

Description:

Configures the CSDB frame transmitted on a given channel. A CSDB frame is composed of a SYNC block followed by one or more message blocks. Address, status and data bytes in each Message Block are set to 0 by default. The transmit frame is periodically transmitted. Refer to **Figure 3-1** for parameter relationships.

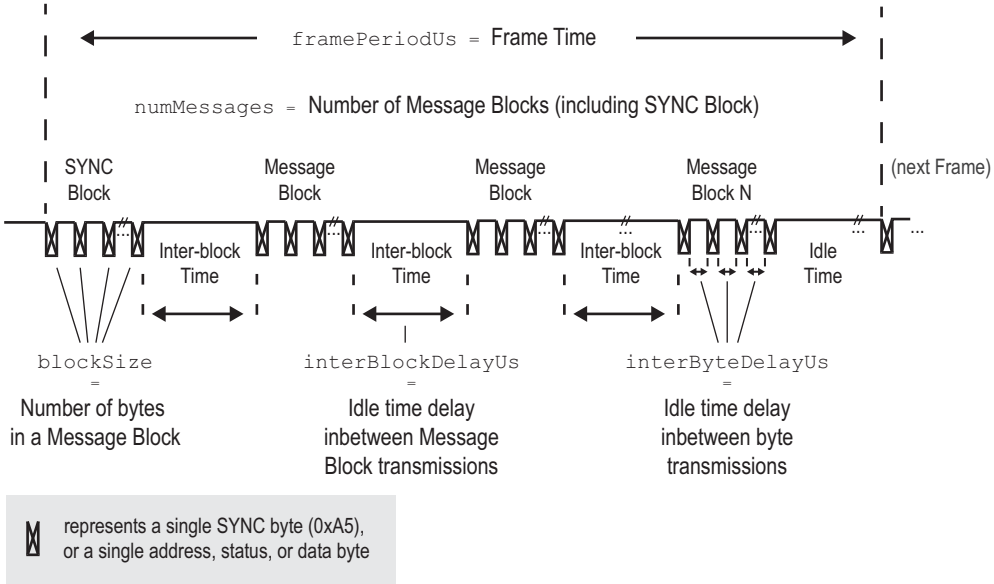


Figure 3-1 Frame Configuration Parameters

3.3.3 UeiCSDBGetStatus()

```
tUeiCSDBStatus UeiCSDBGetStatus (int hd, int devn)
```

Parameters:

`hd`	handle to the IOM where the serial device is located
`devn`	ID of the serial device (zero based index)

Returns:

Status code of type `tUeiCSDBStatus`: 1 is waiting for frame, 0 is success, negative numbers are last error. **Table 3-2** below lists all status codes that this function returns.:

1	`UeiCSDBNoFrameAvailable`	Did not receive a complete frame yet
0	`UeiCSDBSuccess`	Success
-1	`UeiCSDBInvalidId`	Handle or device parameter is incorrect
-2	`UeiCSDBChannelNotFound`	Can't find a channel with specified id
-3	`UeiCSDBDeviceNotFound`	Can't find a device with specified id
-4	`UeiCSDBTimeout`	Timeout error
-5	`UeiCSDBBadParameter`	Invalid parameter
-6	`UeiCSDBChannelNotConfigured`	Specified channel is not configured
-7	`UeiCSDBCommError`	Error occurred in the driver handling the serial device
-8	`UeiCSDBRXFifoOverflowError`	RX FIFO overflow error
-9	`UeiCSDBInvalidFrame`	Received invalid frame (no header detected, not enough message blocks etc...)
-10	`UeiCSDBInternalError`	Unrecoverable internal error, look at stderr output for details

Table 3-2 CSDB-509 Error / Status Codes Returned by API

Description:

Gets last comm error / status code that occurred on a given device.

3.3.4 UeiCSDBInitialize ()

```
void UeiCSDBInitialize ()
```

Parameters:

none

Returns:

none

Description:

Initializes internal buffers and variables

3.3.5 UeiCSDBReadRxFrame()

```
int UeiCSDBReadRxFrame (int hd, int devn, int channel, int numMessages,
tUeiCSDBMessage * messages)
```

Parameters:

`hd`	handle to the IOM where the serial device is located
`devn`	ID of the serial device (zero based index)
`channel`	ID of the channel to configure (zero based index)
`numMessages`	maximum number of message blocks that can be stored in the messages array
`messages`	array containing the received message blocks

Returns:

Status code: 0 upon success, negative upon error. **Table 3-2** on page 24 lists all CSDB-509-specific status codes.

Description:

Reads RX FIFO and transfers message block data received in the most recent RX frame to the `messages` array.

3.3.6 UeiCSDBSetTxFrame()

```
int UeiCSDBSetTxFrame (int hd, int devn, int channel, int messageIndex,
tUeiCSDBMessage * message)
```

Parameters:

`hd`	handle to the IOM where the serial device is located
`devn`	ID of the serial device (zero based index)
`channel`	ID of the channel to configure (zero based index)
`messageIndex`	zero based index of the message block to update
`message`	`tUeiCSDBMessage` structure containing the message content

Returns:

Status code: 0 upon success, negative upon error. **Table 3-2** on page 24 lists all CSDB-509-specific status codes.

Description:

Sets the address, status and data bytes of a message block. Messages are assembled in structures of type `tUeiCSDBMessage`:

```
typedef struct _UeiCSDBMessage
{
   unsigned char address; // address byte for the message
   unsigned char status;  // status byte for the message
   unsigned char data[10];//data bytes* for the message
                 // *Number of data bytes is block size - 2
} tUeiCSDBMessage;
```

3.3.7 UeiCSDBStart()

```
int UeiCSDBStart (int hd)
```

Parameters:

`hd`	handle to the IOM where the serial device is located

Returns:

Status code: 0 upon success, negative upon error. **Table 3-2** on page 24 lists all CSDB-509-specific status codes.

Description:

Starts all configured CSDB channels.

3.3.8 UeiCSDBStop()

```
int UeiCSDBStop (int hd)
```

Parameters:

hd	handle to the IOM where the serial device is located

Returns:

Status code: 0 upon success, negative upon error.**Table 3-2** on page 24 lists all CSDB-509-specific status codes.

Description:

Stops all configured CSDB channels.

3.3.9 PowerDNx Low-level Functions

Low-level functions that are not specific to the CSDB-509 are described in detail in the *PowerDNA API Reference Manual*.

For additional information regarding low-level programming, refer to the *PowerDNA API Reference Manual* located in:

- On Linux systems:
  <PowerDNA-x.y.z>/doc
- On Windows systems:
  Start » All Programs » UEI » PowerDNA » Documentation

NOTE: When the DNx-CSDB-509 is used as a standard serial device in non-CSDB mode, the CSDB-509 uses the same API functions as the DNx-SL-508 / SL-501 boards. CSDB-specific functions unique to the DNx-CSDB-509 are found only in the UEI CSDB library and described only in this manual.

3.4 CSDB Low-level Programming Examples

Application developers are encouraged to explore existing source code examples when first programming the CSDB-509. Sample code provided with the installation is self-documented and serves as a good starting point.

CSDB code examples are located in the following directories:

- For Linux: <UeiCSDB-x.y.z>/src/DAQLib_Samples
- For Windows: *Start » All Programs » CSDB » PowerDNA » Examples*

Section 3.4.1 thru Section 3.4.6 step through a programming example for the CSDB-509. The provided example relies on the following hardware conditions:

- A CSDB-509 device is installed in slot DEVN in an IOM
- CSDB-509 channels (ports) 1 and 0 are enabled
- Both channels are configured as 50 kbits/s baud rate with odd parity enabled
- Channel 0 TX pins are externally wired to feed Channel 1 RX pins
- Channel 1 is reading serial data transmitted from channel 0

3.4.1 Including Library Header Files

The first step is to include the PowerDNA low-level API and UEI CSDB library header files.

```
#include "PDNA.h"
#include "UeiCSDBLib.h"
```

3.4.2 Opening Communication between IOM and Host PC

A handle on the IOM must be created to identify where the CSDB-509 is located to open communication:

```
DqOpenIOM(IOM_IPADDR0, DQ_UDP_DAQ_PORT, TIMEOUT_DELAY,
&hd0, &DQRdCfg);
```

3.4.3 Initializing and Configuring Channels

The following initializes the CSDB library:

```
UeiCSDBInitialize();
```

The channel configuration API sets channel speed (50 kbit/second in this example) and parity (odd).

```
UeiCSDBConfigureChannel(hd0,    // Handle to IOM
                        DEVN,   // Device # in IOM
                        0,      // Channel #
                        50000,  // baud rate
                        1       // parity
                        );
UeiCSDBConfigureChannel(hd0,    // Handle to IOM
                        DEVN,   // Device # in IOM
                        1,      // Channel #
                        50000,  // baud rate
                        1       // parity
                        );
```

3.4.4 Configuring Timing and CSDB Frames

The following API configures the CSDB frame transmitted on a given channel, (e.g., block size, delays).

```
UeiCSDBConfigureFrame(hd0,     // Handle to IOM
                      DEVN,    // Device # in IOM
                      0,       // Channel #
                      NUM_MESSAGES, // # Messages in Frame
                      BLOCK_SIZE,   // # bytes in Message
                      100,    // Idle time between bytes
                      2000,   // Idle time between Messages
                      200000);// Frame (SYNC to SYNC time)
UeiCSDBConfigureFrame(hd0,     // Handle to IOM
                      DEVN,    // Device # in IOM
                      1,       // Channel #
                      NUM_MESSAGES, // # Messages in Frame
                      BLOCK_SIZE,   // # bytes in Message
                      100,    // Idle time between bytes
                      2000,   // Idle time between Messages
                      200000  // Frame (SYNC to SYNC time)
                      );
```

Frame configuration parameters are detailed in Figure 3-1 on page 23.

3.4.5 Writing and Reading Data

The following code initializes message block 0 in the frame; each message block can be initialized similarly:

```
message.address = 0x10;
message.status = 0x1A;
for (i = 0; i < 6; i++) {
   message.data[i] = message.address + i;
}
UeiCSDBSetTxFrame(hd0,  // Handle to IOM
                  DEVN, // Device # in IOM
                  0,    // Channel #
                  0,    // Message block index #
                  &message // address, status & data bytes
                  );       //   in Message block
```

The following starts periodic frame transmission, checks status, and updates message blocks periodically transmitted by port 0 and received on port 1:

```
UeiCSDBStart(hd0);
count = 0;
while (!stop &&
        (UeiCSDBSuccess == UeiCSDBGetStatus(hd0, DEVN))) {
   message.address = 0x10 + (count % 16);
   for (i = 0; i < 6; i++) {
      message.data[i] = message.address + i;
   }
   UeiCSDBSetTxFrame(hd0, DEVN, 0, 3, &message);


   UeiCSDBReadRxFrame(hd0, DEVN, 1, NUM_MESSAGES,
rxMessages);
   for (i = 0; i < NUM_MESSAGES; i++) {
      printf("Message #%d: address=0x%x, status=0x%x,
data=[ ", i, rxMessages[i].address, rxMessages[i].status);
      for (j = 0; j < BLOCK_SIZE - 2; j++) printf("0x%x ",
rxMessages[i].data[j]);
      printf("]\n");
   }
}
```

3.4.6 Stopping and Cleaning-up

The following API stops CSDB transmissions and stops the RX port from receiving data.

```
UeiCSDBStop(hd0);
```

Closing the IOM cleans up connections and frees up resources.

```
DqCloseIOM(hd0);
```

3.5 Standard Serial (non-CSDB) Programming Examples

To program the CSDB-509 as a standard serial device in non-CSDB mode, please refer to sample programs for the SL-501/8 provided with the PowerDNA installation. Sample code is self-documented and serves as a good starting point.

Code examples are located in the following directories:

- For Linux: <PowerDNA-x.y.z>/src/DAQLib_Samples
- For Windows: *Start » All Programs » UEI » CSDB » Examples*

When the DNx-CSDB-509 is used as a standard serial device, the CSDB-509 uses the same API as the DNx-SL-508 / SL-501 boards. See “PowerDNx Low-level Functions” on page 27 for PowerDNx API information.

Appendix A

A.1 Accessories

The following cables and STP boards are available for the CSDB-509 board.

DNA-CBL-62

This is a 62-conductor round shielded cable with 62-pin male D-sub connectors on both ends. It is made with round, heavy-shielded cable; 2.5 ft (75 cm) long, weight of 9.49 ounces or 269 grams; up to 10ft (305cm) and 20ft (610cm).

DNA-STP-62

The STP-62 is a Screw Terminal Panel with three 20-position terminal blocks (JT1, JT2, and JT3) plus one 3-position terminal block (J2). The dimensions of the STP-62 board are 4w x 3.8d x1.2h inch or 10.2 x 9.7 x 3 cm (with standoffs). The weight of the STP-62 board is 3.89 ounces or 110 grams.

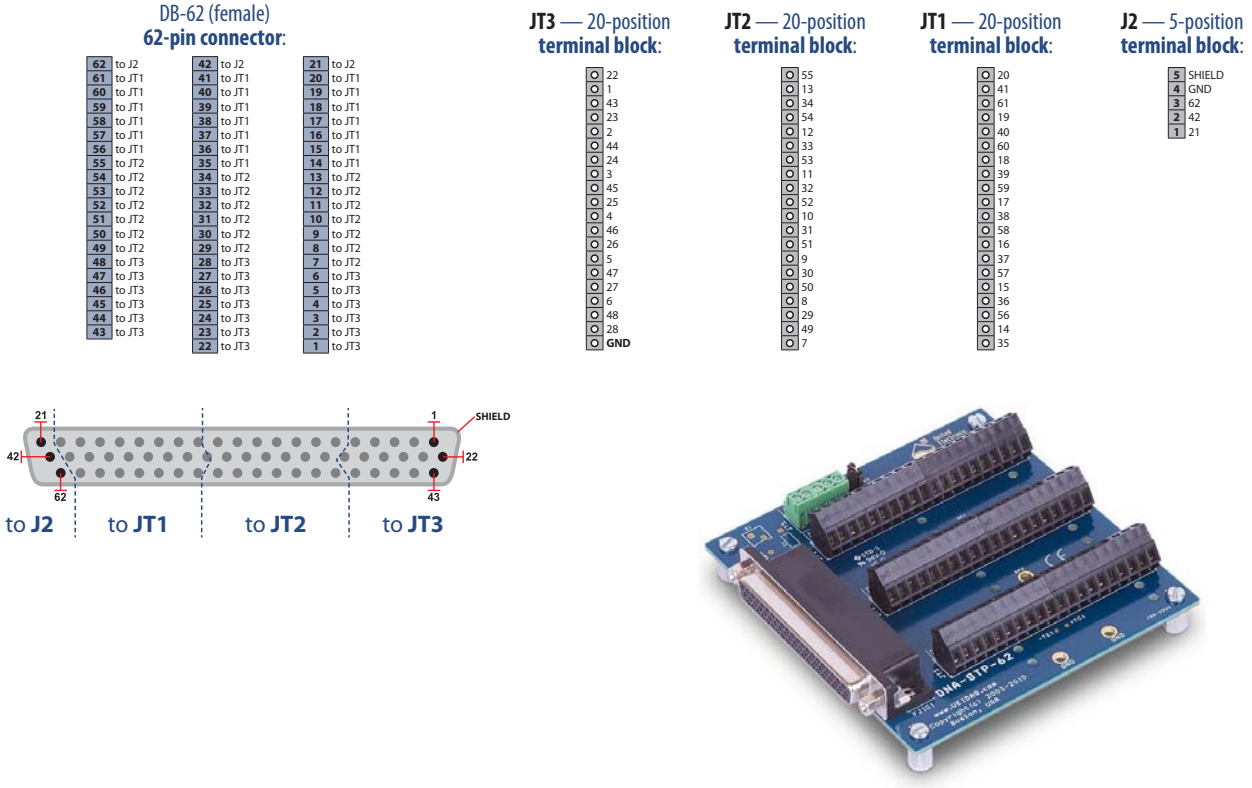


Figure A-1 Pinout and Photo of DNA-STP-62 Screw Terminal Panel

DNA-STP-508

A DNA-STP-508 screw terminal panel is an accessory that serves as a convenient wiring interface between a 62-pin connector that plugs into a mating connector on the CSDB-509 boards and either DB-9 serial cable connectors or eight 5-pin screw terminal blocks on the DNA-STP-508 accessory panel.

Index