CDOAN-SC-DNP

DNP3 Outstation Source Code Library
Release 1.0
March 6, 2021

T P I L B

Contents

Overview 1
Introduction 1
Quick Start 1
Compiler 4
Integer Sizes 4
Not Linux 4
Feature List 1
DNP3 Features Supported 2
DNP3 Features Not Supported 3
How to Get This Running 5
Summary 5
Routines We Provide 5
Routines You May Need To Modify 5
General 5
Routines You Need to Call 6
Initialization 6
Details 6
Timeslice 7
Point Value/State Updates 7
Callback Routines You Need To Write 8
Control 8
Cold Restart 8
If You Are Not Running Under Linux 9
Network 9
int16_t InitNetwork() 9
Serial 9
DNP3 Protocol Notes 11
Subset Levels 11
Counters 11
Point Flags 11
Event Classes 12
Time Synchronization 12
Event Generation 13
File Operations 13

Configuration Options 15
Configuration Methods 15
Configuration Options Table 15
Default Variations 16
Event Class 16
Table of Configuration Options 17
Debug Aids 20
Test Point Data Base 20
Command Processor 20

Overview

Introduction

CDOAN-SC-DNP is a “C” source code library for implementation of DNP3 protocol on an outstation, primarily under Linux. The intent is to be able to provide a fast and simple way to implement DNP3 protocol with virtually no configuration of the protocol. Routine calls are kept to a minimum. The library performs as much work as possible. It is the intent that a basic DNP3 conformant implementation can be up and running within a day, with slightly more time required to connect all I/O points.

Those that are running under a non-Linux O/S can also use this library, with a little more work.

The library is provided free-of-charge and as-is. While we know of no problems with the library, we also assume no liability for its operation or for any missing feature.

Quick Start

The minimum required to see if the library will work for you:

- Compile all code using a compatible “C” compiler. We used Geany 1.33. There is a guide in the next section on how to correct compiler issues (if any).
- We also supply a routine called “main.c”. This will eventually be replaced by your main program, but you can try ours to start. It will create a data base with eight digital input, analog input, counter input, binary output, and analog output points, and provide a means for generating changes.
- Run. As configured with the defaults, the routine will accept a TCP/IP connection on port 20000, or operate serially over USB serial port “/dev/ttyUSB0”. Both use DNP3 address 1. It will respond to any master address.
- Any of these settings may be changed. See the section on Configuration Options.
- If you are using a serial port, the code will detect the baud rate, but it may take a couple of minutes to do so.

Once this works, you can make changes as described below to incorporate the code within your product.

Feature List

DNP3 Features Supported

<table>
<tr><td>Communication Types
TCP/IP with UDP broadcast
UDP only
Serial</td><td>Subset Levels
1
2
3</td></tr>
<tr><td>Data Link Layer
Responds to requests with Unconfirmed or Confirmed User Data
Sends data messages only with Unconfirmed User Data
Optionally supports self-address (0xFFFC)</td><td>Application Layer Function Codes
Confirm
Read
Write
Select, Operate, Direct, Direct NoAck
Freeze, Freeze and Clear
Cold Restart
Time Synchronization, LAN and serial
Enable/Disable Unsolicited
Assign Class (as expanded in IEEE 1815-202x)
File Operations</td></tr>
<tr><td>Point formats
1-bit and 2-bit binary
16-bit numeric
32-bit numeric
Floating point</td><td>Polls
Class data
Individual object groups
Qualifiers: All points, start/stop ranges, counts
Variations: All except double floating point</td></tr>
<tr><td>Input points
Binary
Double-bit binary
Binary output status
Counter (Running counter)
Frozen counter
Analog
Binary output status
Analog output status</td><td rowspan="2">Events
Binary and double binary: queue of multiple events per point. Queue size is configurable
Analog: one event per point based on value change and optional deadband setting
Counter: one event per point based on value change and optional deadband setting
Frozen Counter: one event per point on freeze
All points: Events generated on flag change</td></tr>
<tr><td>Output Points
Binary output (CROB)
Analog output</td></tr>
<tr><td>File Operation
Read
Write
Append
Delete
File Info
Directory read</td><td>Other
Unsolicited reporting
Download deadband
Device Attributes (Object group 0)</td></tr>
</table>

DNP3 Features Not Supported

- Data Sets
- Secure Authentication
- Dual Endpoint

Compiler

An attempt was made to conform to the widest set of "C" compiler rules and constructs. For example, there is no "Endian" dependent code; the library should work with both big and little-endian representations. But, even so, there are some possible compiler issues.

Integer Sizes

We used the following definitions, supported by our compiler, for 16, 32, and 64-bit integers.

- int16_t
- int32_t
- int64_t
- uint16_t
- uint32_t
- uint64_t

If your compiler does not define these types, you can create an appropriate set of "#define" statements. Insert these in "sc.h". Defines that work are dependent on the compiler you actually use. Following is an example of what is most likely to work.

- #define int16_t short
- #define int32_t long
- #define int64_t long long
- #define uint16_t unsigned short
- #define uint32_t unsigned long
- #define uint64_t unsigned long long

Not Linux

If you are not running under Linux, you may need to modify the network routine network.c and the serial I/O routine serial.c. These are both described in the section on Routines You May Need To Modify.

How to Get This Running

Summary

Here is all you need to do to start. Details are explained later in this section.

- Modify deviceprofile.h to set options for your system. To start, you can accept our defaults and make changes later.
- Create a main program:

```
StartCDOANInit(NULL);
FinishCDOANInit();
AddPoints(...);   // One call to define each input or output point
StartComm();
While (TRUE)
{
        Call your code;
        UpdatePointValues(...) as appropriate
        TimeSlice();            // Let the library do its work
}
```

- Create two callback routines

```
userControlRequest(...)  // To handle master control requests (binary and analog output)
Coldrestart()            // To handle master requests to restart the outstation
```

Routines We Provide

The library includes the following source code files:

C: comm.c, dnp.c, dnpdata.c, main.c, network.c, points.c, serial.c, timeslice.c, user.c, utilities.c
H: comm.h, deviceprofile.h, dnp.h, extern.h, points.h, sc.h, serial.h
MakeFile: makefile (Geany compatible)

Routines You May Need To Modify

Routines you likely need to modify are deviceprofile.h and main.c.

General

Deviceprofile.h

This module contains equates for all DNP options and may be changed to correlate to your device profile document. Definitions in this file are defaults. That is, they are parameters that will be used if not changed by real time code. This is discussed later in the section on Configuration Options.

Main.c

The main program should be written by the user of the library. However, a main.c was provided as a sample. You can use this routine as a “proof of concept” that the library will work in your environment.

Main.c provides examples for all routines described in the next section.

Routines You Need to Call

The library contains four classes of routines to interact with:

- ***Initialization*** routines are called once at startup
- ***Timeslice*** is called periodically and as often as possible. On each call, the library checks for and processes DNP requests, and formats and sends responses.
- ***Point value/state updates*** routines should be called whenever a point value or state changes. You only need to provide new point information. The library creates and maintains events.
- ***Callback*** routines are called by the library for you to process:
    - Binary and analog controls requests
    - Cold Restart requests

Initialization

On startup, you should call the following routines in the order shown:

- StartCDOANInit – To get things going
- SetOption – Optional: To change any DNP3 parameters before they are applied. If you do not want to change anything, you do not need to issue any calls
- FinishCDOANInit – To finish the initialization process
- AddPoints – To define your DNP3 point data base. One call per point
    - UpdatePointValue can also be called if you want to provide an initial value for any input point before communication starts. Points are assigned a RESTART status until an initial value is written
- StartComm – To start communication

Details

*StartCDOANInit(char *ConfigFileName)*

Initialize the DNP3 library with default options defined in the compile-time file deviceprofile.h. These compile-time options can be overridden by definitions contained in a text file specified by the *ConfigFileName* parameter (NULL if not used). If non-NULL, it is the full path of a text file of the same format as deviceProfile.h. The file is read, with option values overriding the compile-time values. That is, you can use one deviceprofile.h file to compile defaults, then modify those and apply the modified values at run-time.

*SetOption(char *OptionName, void *OptionValue)*

You can also override option values with information from your own configuration source. Each call to SetOption changes one option:

- OptionName is the name of the option to change, exactly as it appears in deviceprofile.h after a “#define” statement. OptionName is case insensitive.
- OptionValue is a pointer to the option value. For numeric options, it points to an *int32_t value (even if the value to be set is 8, 16, or 32 bits). For text options, it points to a *char 0-terminated string.

FinishCDOANInit ()

Once all options have been set, this call completes the initialization process.

AddPoints(char DNPStaticObject, int16_t index, int16_t deadband)

Define data base points, one point per call

- DNPStaticObject is the static object group defined in IEEE 1815 and “dnp.h”
  - DNPOBJ_BINARY_INPUT (1)
  - DNPOBJ_DOUBLE_BINARY_INPUT (3)
  - DNPOBJ_BINARY_OUTPUT_STATUS_INPUT (10) generates a binary output status point and the associated binary output point
  - DNPOBJ_RUNNING_COUNTER_INPUT (20) generates a running counter. A frozen counter at object group 21 is also generated if frozen counter points are allowed by the configured subset
  - DNPOBJ_ANALOG_INPUT (30)
  - DNPOBJ_ANALOG_OUTPUT_STATUS_INPUT (40) generates an analog output status point and the associated analog output point
- Index is the DNP object index. These should be (but not required) sequential starting at 0
- Deadband is the reporting deadband for analog and counter inputs. A value of “-1” instructs the code to use the default deadband. A value of 0 disables event generation based on a value change. Events are still generated based on flag changes.

StartComm()

Starts communication.

The simplest code to start the library and to use compiled defaults is:

- Init_CDOAN_Library((char *) 0);
- FinishInit();
- AddPoints(DNPOBJ_ANALOG_INPUT, 0, -1); // Create at least one point
- StartComm();

Timeslice

After initialization and as often as possible, call:

TimeSlice();

This passes control to the library to handle communication.

Point Value/State Updates

Point values and states can be updated by any of the following calls:

UpdatePointValue16(char DNPStaticObject, int16_t index, int16_t value, char flags)
UpdatePointValue32(char DNPStaticObject, int16_t index, int32_t value, char flags)
UpdatePointValueFloat(char DNPStaticObject, int16_t index, float value, char flags)

- DNPStaticObject is the DNP object used to define the point in the AddPoints call. Only static values need to be updated. The library generates and maintains events. Do not update values for analog output status, binary output status, or frozen counters. These values are maintained by the library.
- Index is the DNP3 point index
- Value is the point value in the format associated with the call. Either of the integer calls can be used for binary and double-binary inputs
- Flags is a flags octet defined by DNP3. Use DNP_POINT_FLAG_ONLINE (1) for points in the normal state.

Callback Routines You Need To Write

Control

This user routine is called whenever a control request is detected and has passed basic validation. A call is not made for a non-existent point, for an operate with no matching select, when an operate has timed out (too long after select), or for more simultaneous controls that can be supported.

Int16_t userControlRequest(char ApplicationFunctionCode, Control_t cRequest)

- **ApplicationFunctionCode** is:
  DNP_APPLICATION_FUNCTION_CODE_SELECT,
  DNP_APPLICATION_FUNCTION_CODE_OPERATE,
  DNP_APPLICATION_FUNCTION_CODE_DIRECT_OPERATE, or
  DNP_APPLICATION_FUNCTION_CODE_DIRECT_OPERATE_NO_ACK.

- The Control_t structure is defined in points.h and shown below

```
typedef struct
{
   Int32_t iValue;              // Integer value, valid only if ValueType is not PVAL_FLOAT
   float fValue;                // Floating point value, valid only if ValueType is PVAL_FLOAT
   int16_t DNPIndex;
   char ValueType;              // Sent from master: PVAL_32BIT, PVAL_16BIT, or PVAL_FLOAT
   char DNPObject;              // DNPOBJ_CROB or DNPOBJ_ANALOG_OUTPUT
   char Status;                 // Used by library only
   char SelectOperateValidated; // Used by library only
   char DNPQualifier;           // Used by library only
} Control_t;
```

  Only one of the value fields, iValue or fValue, will be valid, corresponding to ValueType.

- After performing whatever is needed to validate and perform the request, the routine returns a completion code. DNP_CONTROL_STATUS_CODE_SUCCESS is use for success. All other codes indicate a failure condition. A list of STATUS_CODE equates can be found in dnp.h

Cold Restart

A callback to:

ColdRestart()

is called to process a Cold Restart command, only if options indicate that a Cold Restart is supported. Before calling ColdRestart(), the library responds to the master with a delay time defined by the equate COLD_RESTART_TIME (in seconds). It is the responsibility of your routine to be up and running again within this time.

If You Are Not Running Under Linux

You may need to modify routines that handle network and serial I/O.

Network

All network I/O routines are contained in module "network.c". The ones you may need to rewrite (if not under Linux) are:

int16_t InitNetwork()

Initializes the TCP listen socket (if using TCP) and the UDP socket. Returns 0 if OK. Any non-zero return is using to report an error code

int16_t TCPConnected()

Called by the library to check on network communication capabilities. The return value indicates whether or not the system is in a state where it can send and receive network messages.

- TRUE if using TCP for normal communication and a TCP connection is active, or if using UDP for all communication
- FALSE if using TCP for normal communication and no TCP connection is active

void CheckForNetworkConnection()

Called by the library if the return from TCPConnected() was FALSE. This routine is expected to accept a network connection, if one is pending. Whether or not a network connection request is processed, it should return immediately.

*int16_t CheckForNetworkMessage(char *msg)*

Called by the library if the return from TCPConnected() was TRUE. Code should check for TCP or UDP input. Data read should be added to "char" array ***DNPMessage*** at index ***CompiledMessageSize***, and ***CompiledMessageSize*** incremented by the number of octets added.

The routine should return immediately if no data is available (never block).

If some data was received, then the following statement should be executed after data is added to ***DNPMessage***:

```
return (GetDNPMessage(msg));
```

GetDNPMessage extracts a valid DNP3 message from ***DNPMessage*** (if one exists), adjusting the buffer and size parameters accordingly.

*void NetworkSend(char *msg, int16_t length)*

Called by the library to send a message over:

- UDP if (***CommunicationMode*** & COMM_MODE_UDP) is true
- TCP if (***CommunicationMode*** & COMM_MODE_UDP) is false

Serial

All serial I/O routines are contained in module "serial.c". The ones you may need to rewrite (if not under Linux) are:

*static int16_t initSerial(char *portname, int32_t BaudRate)*

To initializes a serial port.

portname

The serial port name, valid if BaudRate >= 0

BaudRate

> 0 Defines an actual baud rate to use

0 States that the baud rate is unknown and the routine should search for the correct rate. The routine should configure the port for some possible baud rate and set:

SerialPort.flags |= SERIAL_FLAG_SEARCHING;

This flag allows the following call (with baudate == -1) to be made as needed.

-1 Indicates that the last baud rate setting did not work. Try another baud rate on the same port as the prior call (the portname parameter in this call will be NULL)

*static int16_t checkForSerialMessage(char *msg)*

Operates the same as ***CheckForNetworkMessage***, except on the serial port opened in ***initSerial()***. If the baud rate is still unknown (SERIAL_FLAG_SEARCHING is set in SerialPorts.Flags) then ***checkForSerialMessage()*** is also responsible for determining if the current baud rate is correct.

To determine if the current baud rate is correct, execute the following whenever at least ten octets have been received.

```
If (SerialPort.flags & SERIAL_FLAG_SEARCHING)
{
    DNPCrc(msg, 0, 8, crc);  // Calculates the DNP CRC on 8 octets starting at msg[0]
    If      (The message start with 0x05/0x64) and
            (the 9th and 10th octets match the two octets in char crc[2])
    {
            SerialPort.flags &= ~SERIAL_FLAG_SEARCHING; // Baud rate found
    }
}
```

Once some period of time has elapsed without detecting a valid message, try a new baud rate by calling **init_serial(NULL, -1)**

*static void SendSendMessage(char *msg, int16_t length)*

Transmit the message

DNP3 Protocol Notes

Subset Levels

The library supports subset levels 1, 2, and 3. There is not much difference between these levels from the point of view of what an outstation is allowed to do. Subsets constrain an outstation from sending a message outside the subset in response to a master request that is within the subset. An outstation is not constrained from sending a message outside the subset if the request was also outside the subset, as long as the request and response are both well-defined by DNP3.

For example, analog floating points variations are not part of any of the supported subset levels. A class 1/2/3/0 poll is part of the subsets. So, the response to a class 1/2/3/0 poll cannot return floating point analog values.

On the other hand, a master may issue an analog input read with an explicit request for a floating-point variation. Outstations are not required to parse such a request and may send a null error response. But they may also support the request and respond accordingly. Both the request and response happen to be outside the subset, and are both well-defined by DNP3.

The library attempts to conform to the widest possible implementation compatible with the subset. In this case, it will not use floating-point as the default variation, but will honor an explicit request.

Counters

The library maintains running and frozen counter values and events. While running counters are part of every subset level, frozen counters are not. Furthermore, while outstations may return both running and frozen counters in class data responses, they must be configurable to return only one or the other, but not both.

Accordingly, default class data responses are adjusted to conform as best as possible to the configured subset. The following default assignments are placed in effect at system startup. They may be overridden by Assign Class commands from the master, or by changing the default assignments.

Subset	Class Data Responses
1	Running counters
2	
3	Frozen counters

Static objects not returned in a class data poll can be obtained by reads of their specific object group. Events are not generated for any point not returned in a class 0 response.

Point Flags

Both point value and flag information are updated by the user via calls to **UpdatePointValue**. Events are always generated on a flag change, and may also be generated on a value change.

Most flags are set by the caller in calls to **UpdatePointValue.** Two are maintained by the library:

CONTINUITY
: For counter and frozen counter points

RESTART

This flag indicates the device has restarted and the point's value or state has not yet been set. The library sets RESTART when the point is defined and clears it when a value or state is written.

- If a point is defined and a value written before the point has been transmitted to the master, then an event is not generated on the RESTART flag change
- If a point is defined and sent to the master with the RESTART flag set, then an event is generated when a value is eventually written and the RESTART flag cleared

Event Classes

The library allows point to be placed in one of 5 events classes:

1, 2, 3: Points in classes 1, 2, or 3 return static data in class 0 and events in their assigned class

0: Points in class 0 return static data in a class 0 response. Events are not generated

-1: Points in class "-1" (no class) are not returned in class 0, 1, 2, or 3. Events are not generated. Static data can be obtained by explicit reads of the point's static object group.

IEEE 1815-2012 allows assignment of points to event classes 0, 1, 2, or 3 via the Assign Class command. A new IEEE 1815 (to be released in 2021 or 2022) enhances the Assign Class syntax to also allow assignment to "no class". This library supports both the old and expanded syntax.

Events are not generated for analog output status and binary output status points. These points can be only be assigned to class 0 or to "No Class".

Time Synchronization

The library can be configured to periodically require time synchronization commands from the master. An outstation that always obtains time from a local source, such as a GPS receiver, will not want to request or process DNP3 time synchronization requests.

If you have a local time source and do not want to support time synchronization over the protocol

Set NEED_TIME_FREQUENCY (defined in deviceprofile.h) to zero. In this case, the NEED_TIME internal indication bit is never set, the time is always considered valid, and time synchronization commands received from the master are responded to with a BAD_FUNCTION_CODE internal indication.

The library is configured to conform to DNP3 standards, which require use of UTC time in all generated time stamps. Users that want to use local time (which would be a protocol violation) can set the deviceprofile.h equate

```
#define USE_UTC_TIME        FALSE
```

If you want to support time synchronization over the protocol (no local time source)

Set NEED_TIME_FREQUENCY (defined in deviceprofile.h) to a positive value. This specifies a time (in seconds) for the library to set the NEED_TIME internal indication bit. NEED_TIME is set on system startup and whenever this specified time has elapsed without receiving a time

synchronization request from the master. NEED_TIME is cleared whenever a time synchronization process completes.

Outstation time is still considered valid for some period of time after the NEED_TIME flag is set. Time is considered invalid only when a time synchronization sequence has not been completed in a somewhat longer time, as specified by the equate TIME_INVALID_FREQUENCY. For example:

```
#define NEED_TIME_FRQUENCY        300
#define TIME_INVALID_FREQUENCY    360
```

causes the NEED_TIME internal indication to be set when no time synchronization sequence has occurred within 5 minutes, but allows outstation time to be valid until no time synchronization sequence has occurred within 6 minutes.

The library never uses the time in a time synchronization message to modify the system clock. The difference between time synchronized from the master and actual system time is remembered by the library. This difference is used to calculate time stamps.

The library does not check (nor can it) that time sent over the protocol is UTC or local time. The time sent is used for generated time stamped events.

Event Generation

The user is only required to update the point data base via an ***UpdatePointValue*** call as new values or states become available. A call should be issued as soon as possible after a new value or state is detected.

The library generates events and assigns time stamps as part of the update process. Time stamps are assigned the time quality of “synchronized” if the clock setting is determined to be valid when the point change is written, or “unsynchronized” otherwise. Binary change events recorded with unsynchronized time are reported to the master with the relative time variation and an unsynchronized common time of occurrence.

Binary (single and double bit) events are placed in an event queue. The same queue is used for both point types. Multiple events are maintained until the queue overflows. During an overflow condition, oldest events are deleted to make room for new events.

Analog and running counter events are declared on any change in a flag setting or when the difference between the current and last reported values meet or exceed the point’s deadband. Value change events are not processed if the point’s deadband is 0. Only one event per point is maintained for an analog or running counter point. The event is sent to the master using value and flag information in effect at the time of transmission.

Frozen counter events are generated on a counter freeze. One event is maintained per frozen counter point.

File Operations

The library supports file Open, Read, Write, Append, and Delete, and Get File Information. Read support include normal files and directories.

File support takes a significant amount of code. Devices with small memory footprints that do not require file operation support can remove the support code by adding (in sc.h) the statement:

```
#define REMOVE_FILE_OBJECT_CODE_SUPPORT
```

When file I/O is included, file operations are limited to a single base folder in the host system. This protects against a master deleting or overwriting critical files. The base folder is defined by the equate:

```
#define FILE_READ_WRITE_BASE_FOLDER "/home/pi/Public"
```

The defined folder should be writable and not require special access permissions. Any file path or file name sent from the master is prefixed by this folder. For example, a request to write file "test.txt", "\test.txt", or "/test.txt" will write file:

```
/home/pi/Public/test.txt
```

The library code supports writes using Linux and Windows file conventions. Any '\' character found in a file name in a message from the master is changed to '/', as required in Linux.

Configuration Options

Configuration Methods

The library has several options that are configurable in one of three ways:

- #define's located in "deviceprofile.h" provide initial definitions of all configurable options. For example,

  ```
  #define DEFAULT_SUBSET_LEVEL 2
  ```

  Configures the library to operate as a subset level 2 outstation.

- Any option can be overridden at run time by creating a file in the same format as deviceprofile.h and providing the name and path of that file as a parameter to **StartCDOANInit**. For example, if this file contains the entry:

  ```
  #define DEFAULT_SUBSET_LEVEL 3
  ```

  Then the library will operate as a subset level 3 outstation regardless of what was defined at compile-time.

- After calling **StartCDOANInit,** the user has one more chance to override any parameter, using information from its own configuration area, by calling:

  ```
  SetOption(char *OptionName, void *OptionValue)
  ```

  All **SetOption** calls must be made after StartCDOANInit() and before calling FinishCDOANInit().

  One call is made for each option to change.

  OptionName is a pointer to a string with the same name as the option name in deviceprofile.h. Option names in this context are case insensitive.

  OptionValue is a pointer to the option value. For numeric or Boolean options, it points to an int32_t location (even if the option is stored in an int16 or char variable). For string options, it points to a 0-termianted "C" string.

Configuration Options Table

This section defines all options that can be set in deviceprofile.h or in calls to SetOption.

Definitions

Default Variations

Default variations define the preferred variation to use in reporting an input point value when no specific variation was requested by the master. These are preferences only and subject to override by library code in its application of DNP3 rules. For example, the default variation for binary events may be "absolute time" (variation 2). If so, events will be reported with absolute time if the clock was synchronized when the event was recorded. If not, the library will report the event using the relative time variation and an unsynchronized common time of occurrence.

When a default variation is set to 0, the library determines the best variation to use based on various factors. It is recommended to set all default variations to 0.

Event Class

An event class setting can be -1, 0, 1, 2, or 3. -1 assigns points to "no class".

Table of Configuration Options

Area	Option Name	Meaning	Default
Communi-cation	NETWORK_ COMMUNICATION	Supports network communication	TRUE
	UDP_ONLY	If TRUE, supports UDP only (no TCP). If FALSE, UDP is used only to receive broadcast commands	FALSE
	NETWORK_PORT	For UDP and TCP	20000
	SERIAL_ COMMUNICATION	Support serial communication	TRUE
	SERIAL_PORT_NAME	Required only if serial communication is supported	/dev/tty USB0
	SERIAL_BAUD_RATE	Baud rate to use for serial communication, if known. A value of 0 causes the actual baud rate to be determined in real time	0
DNP Generic	SUBSET_LEVEL	Subset levels 1, 2, or 3	2
	OUTSTATION_ADDRESS	DNP3 Outstation address	1
	MASTER_ADDRESS	Messages are typically only processed from one known DNP3 address. A value of 65535 (not allowed if unsolicited reporting is enabled) allows messages from any master to be processed.	0
	UNSOLCITED_CAPABLE	Causes the null unsolicited message to be sent on startup. Processes ENABLE and DISABLE unsolicited messages.	FALSE
	MAX_POINTS	Maximum number of points that can be defined	5000
	USE_UTC_TIME	Use UTC for time-tagging events. Only applicable for outstations that maintain their own time source outside the protocol	TRUE
	SUPPORT_SELF_ADDRESS	Respond to requests sent to address 0xFFFC	TRUE
	SUPPORT_BROADCAST_ DIRECT_OPERATE_NOACK	Support a broadcast request with a direct operate No/Ack command	TRUE
	SUPPORT_BROADCAST_ CLEAR_RESTART	Support a broadcast command with to clear the restart internal indication bit	TRUE
Reporting variations	BINARY_ STATIC_VARIATION	Preferred variation to use to report static binary and double binary input values	0
	RUNNING_COUNTER_ STATIC_VARIATION	Preferred variation to use to report static running counter input values	0
	FROZEN_COUNTER_ STATIC_VARIATION	Preferred variation to use to report static frozen counter input values	0
	ANALOG_ STATIC_VARIATION	Preferred variation to use to report static analog input values	0
	ANALOG_OUTPUT_ STATUS_ STATIC_VARIATION	Preferred variation to use to report static analog output status values	0

Area	Option Name	Meaning	Default
	BINARY_ EVENT_VARIATION	Preferred variation to use to report binary and double binary input event values	0
	RUNNING_COUNTER_ EVENT_VARIATION	Preferred variation to use to report running counter input event values	0
	FROZEN_COUNTER_ EVENT_VARIATION	Preferred variation to use to report frozen counter input event values	0
	ANALOG_ EVENT_VARIATION	Preferred variation to use to report analog input event values	0
Input Points	ANALOG_DEADBAND	Analog event reporting deadband. 0 disables events based on a change in value	10
	RUNNING_ COUNTER_DEADBAND	Running counter event reporting deadband. 0 disables events based on a change in value	256
	BINARY_EVENT_QUEUE_ SIZE	Maximum number of combined binary and double binary events	256
	BINARY_ EVENT_CLASS	Event class for binary and double binary input points	1
	ANALOG_ EVENT_CLASS	Event class for analog input points	2
	COUNTER_ EVENT_CLASS	Event class for running and frozen counter input points	3
	BINARY_ OUTPUT_STATUS_ EVENT_CLASS	Event class for binary output status points	-1
	ANALOG_ OUTPUT_STATUS_ EVENT_CLASS	Event class for analog output status points	-1
	COLD_RESTART_TIME	Time to restart the outstation. Set to 0 if a COLD_RESTART command is not supported	60
Output Points	BINARY_AND_ANALOG_ OUT_IN_SAME_ MESSAGE	Allow a single control message to control both binary and analog outputs	TRUE
	MAX_CONTROLS_ PER_REQUEST	Maximum controls that can be processed in a single DNP3 message	7
File operations	ALLOW_FILE_READ	Allow file read	TRUE
	ALLOW_FILE_WRITE	Allow file write and append	TRUE
	ALLOW_FILE_DELETE	Allow file delete	TRUE
	FILE_READ_WRITE_ BASE_FOLDER	File base folder. I/O operation is limited to files contained within this folder. Access to the entire disk is not provided	/home/ pi/Public
Timing	APPLICATION_CONFIRM_ TIMEOUT	Maximum time (seconds) to wait for an application layer confirm	4
	NEED_TIME _FREQUENCY	How often the Need Time IIN is set, in seconds. This value should be set to 0 for outstations that synchronize their clocks through a method other than DNP protocol, such as through a local satellite time source	300

Area	Option Name	Meaning	Default
	TIME_ INVALID_FREQUENCY	How often, in seconds, the time must be set before being declared invalid. This should be longer than DEFAULT_NEED_TIME _FREQUENCY (ignored if that parameter is 0)	360
	DELAY_ MEASUREMENT_TIME	Value, in milliseconds, to use in delay measurement responses	10
	INITIAL_UNSOLCITED_ ACK_TIMEOUT	Time, in seconds, to wait for a confirmation to an unsolicited message	2
	MAXIMUM_ UNSOLCITED_ ACK_TIMEOUT	The unsolicited confirm timeout is increased by one second after every failure, but never above the maximum (in seconds). The confirm timeout is reset to the initial value after a successful confirmation	60
	MAX_TIME_BETWEEN_ SELECT_AND_OPERATE	Maximum time, in seconds, between select and operate control commands	2
Object Group 0. These values are returned in applicable Object Group 0 variations. They are not used for any other purpose	LOCAL_TIMING_ ACCURACY	Accuracy of time stamped events, in microseconds	10000
	SOFTWARE_VERSION	Local timing accuracy	10000 microseconds
	HARDWARE_VERSION	Software version	“1.0”
	INSTALLED_LOCATION	Hardware version	“To be supplied”
	USER_ID	Installed location	“Mars”
	DEVICE_NAME	User ID	“Unassigned”
	SERIAL_NUMBER	Device name	“CDOAN Library”
	PRODUCT_NAME	Serial number	“1”
	MANUFACTURERS_ NAME	Product name	“CDOAN Library”

Debug Aids

Main.c is not intended for use in production. The code in main.c should be replaced by user product code. Main.c can be used as a starting point and example.

Test Point Data Base

On startup, the code in main.c creates analog, binary, and counter input points, and binary and analog output points. The same number of points are created for each type as defined by equate

```
#define NUMBER_OF_POINTS_PER_TYPE_DEBUG          8
```

If you want to change the types of pointrs defined, search main.c for the string “AddPoints”. You should see a line commented out for double-bit binary inputs. Remove the comment to add these points.

The code also sets the value/state for each point to 0 by calling “UpdatePointValue16”.

Command Processor

Main.c contains a simple command processor. Each command consists of a keywork and a numeric value. For example:

```
aevents          5
```

generates an event at each of the first 5 analog input points. You can look at the structure “UserCommands” to get a feeling on how to add your own commands. Ones delivered in main.c are:

Command	Description	Value (n)
Aevents n	Generates an event for one or more analog inputs. Only one event per analog is maintained. For example, “aevents 1” generates an event for the first analog point. Calling “aevents 1” again will not generate a new event if the first event was not cleared (read by the master).	Generate events for point indices 0 to <n-1>
Bevents n	Generates multiple events for binary input index 0	Count of events to generate
Cevents n	Generates an event for one or more counter input points. Note same comments as for aevents.	Generate events for point indices 0 to <n-1>
ConfigCorrupt n	Changes the CONFIGURTION_CORRUPT internal indication flag	1 to set the flag, 0 to clear
Devents n	Generates multiple events for double binary input index 0	Count of events to generate
DeviceTrouble n	Changes the DEVICE_TROUBLE internal indication flag	1 to set the flag, 0 to clear
Localmode n	Changes the LOCAL_MODE internal indication flag	1 to set the flag, 0 to clear
Printqueue	Prints information from the circular event queue, used to maintain events for binary and double binary input points	Not used