TRIANGLE MICROWORKS, INC.

Source Code Library
User Manual

Version 3.30.0

May 15, 2022

Chapter 1 Introduction

Congratulations on your purchase of a Triangle MicroWorks, Inc. (TMW) communication protocol Source Code Library (SCL). This product is a member of a family of communication protocol libraries supported by Triangle MicroWorks, Inc. These libraries were designed to be flexible, easy to use, and easily ported to a wide variety of hardware and software platforms. For these reasons, the Triangle MicroWorks, Inc. Source Code Libraries will provide an excellent solution to your communication protocol requirements.

This document is the user manual for all of the TMW SCLs. The manual begins with generic information pertinent to all of the TMW SCLs and gets more specific in later chapters. In addition, a protocol specific document and a Configuration and Interoperability (C-I) guide are included for each purchased protocol.

Please Note

Your license agreement limits the installation of this source code library to specific products. Before installing this source code library in a new target application, check your license agreement to verify that it allows use on that target application. Contact Triangle MicroWorks, Inc. for information about extending the number of target applications that may use this source code library.

1.1 Supported Protocols

As mentioned above, TMW SCLs support a number of different protocols. The list of communication protocols currently supported can be found in Table 1. This table lists the protocol name, the protocol family, and the abbreviation used both within this manual and throughout the associated SCL for each protocol.

Table 1: Supported Protocols

Protocol Name	Protocol Family	Abbreviation(s)/Prefix(es)
IEC 60870-5-101	i870	m101, s101, ft12
IEC 60870-5-102	i870	m102, s102, ft12
IEC 60870-5-103	i870	m103, s103, ft12
IEC 60870-5-104	i870	m104, s104, i104
DNP3	dnp	mdnp, sdnp, dnp
MODBUS	modbus	mmb, smb, mb

Each of the protocols listed above is available as a Master (Controlling Station), Slave (Controlled Station), or Peer (combination Master and Slave) implementation. As shown in **Figure 1**, a typical substation topology consists of an off-site central station computer

that polls one or more Remote Terminal Units (RTU) in the substation. In turn, each RTU polls one or more Intelligent Electronic Devices (IED) that monitors and controls the substation. **Figure 1** illustrates how Master, Slave and Peer Source Code Libraries might be used in a typical substation.

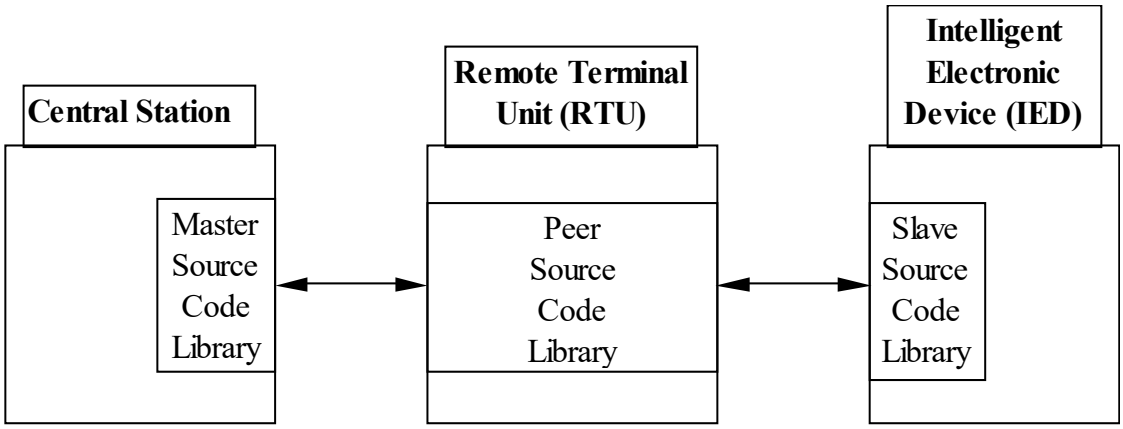


Figure 1: Typical Substation Communication Topology

All of the TMW SCLs share a common architecture, allowing them to be installed in any combination required. For example, it is straightforward to implement a device that supports DNP3 and IEC 60870-5-101. Combining protocols is discussed in detail in Section 5.1.

1.2 Manual Overview

This section describes the layout of this manual and is useful in determining which sections of the manual are pertinent to a particular situation. As mentioned above, all of the TMW SCLs share a common architecture. This manual documents all of the supported SCLs. The manual begins with generic information that applies to all of the SCLs. It then provides more specific information.

Chapter 1 provides an introduction to TMW SCLs and describes the layout of this manual and conventions used throughout this document.

Chapter 2 presents a Quick Start Guide that is intended to provide enough information to get a user with communication protocol experience and/or experience with a previous TMW SCL "up and running" quickly. It also provides a framework that less experienced users can use as a guide while progressing through the more detailed sections of the manual.

Chapter 3 provides an overview of the TMW SCLs. Separate subsections describe Master-, Slave-, and Peer-specific details.

Chapter 4 describes porting details, including instructions for installing the SCL and implementation considerations. Separate subsections describe implementation issues for Master, Slave, and Peer libraries.

Chapter 5 provides information on advanced topics. These topics are often library dependent.

Chapter 6 provides guidelines and suggestions on debugging and testing your implementation. These guidelines are techniques that have proven useful in previous porting and development efforts.

1.3 Manual Conventions and Abbreviations

The following conventions and abbreviations are used throughout this manual.

TMW – Triangle MicroWorks, Inc.

SCL – Source Code Library

<*scl*> - Replace with appropriate SCL prefix from Table 1.

<*family*> - Replace with appropriate SCL family from Table 1.

xxx – This abbreviation is used to represent a family of functions with similar names (e.g., a family of functions that exist for each DNP object or IEC 60870-5 ASDU type. Replace xxx with the appropriate object or type.

Chapter 2 Quick Start Guide

The purpose of this section is to provide users with previous experience with communication protocols, and/or the TMW architecture, a quick step-by-step guide to integrating the TMW SCL on their platform. First time users will also get a feel for the steps required to port a TMW SCL which they can use as a basis as they work through the more detailed information in the following chapters.

This section will highlight each step of the porting process and provide a very high level description of what is required. If more information is required, please refer to the detailed sections found later in the manual. Also, not all steps in this section apply to all of the SCLs. In some cases this will be mentioned, but for the sake of brevity not every situation can be explained in detail. Again, if there are any questions as to whether a specific step is appropriate to your SCL please refer to the detailed sections later in the manual.

2.1 Unpack and Install the SCL

Each TMW SCL is delivered as a self-extracting ZIP file. When delivered via email, the ZIP file is encrypted, and you will have to enter the password (usually delivered by Fax) in order to extract the SCL.

Extract the SCL by running the self-extracting ZIP file. The SCL files should be installed in the desired target directory (<*installdir*>). TMW recommends retaining the directory structure of the SCL as it is supplied.

2.2 Create Interoperability Guide or Device Profile

Your design documentation should include an Interoperability Document or Device Profile, as described in the IEC 60870-5 and DNP standards.

The Source Code Library includes an example Configuration/Interoperability Guide. For IEC 60870-5 protocols this guide is named <*scl*> *CI Guide.docx* and is located in the <*installdir*> directory. This document is a template with much of the information already supplied, indicating the features of the Source Code Library. You will need to edit the document to indicate the features your device will support and to describe methods of configuration.

For DNP the document is called a Device Profile. The DNP3 Specification Volume 8 Interoperability describes the content of the Device Profile Document that should be provided in XML format. A default Device Profile for the DNP Source Code library named *M/SDNP SCL Device Profile.xml* is provided in the <*installdir/DNP Device Profile*> directory. For more information see the protocol specific portion of the SDNP SCL user manual.

2.3 Build the SCL

The first step in porting the SCL should be to build it before making any modifications. Once installed, all SCL source code files reside in the subdirectories under <*installdir*>/*tmwscl*.

All of the source files in each of these subdirectories must be compiled. This step confirms proper installation of the SCL as well as highlighting any compiler-specific issues.

The source files can be compiled using the supplied makefiles or the Visual Studio 2017 or newer solution and project files (*.sln, *.vcxproj). The installed makefiles may require modifications to be compatible with your target platform.

If you are using Visual Studio, you can build the sample application file on Windows. The solution file for this application resides in <*installdir*> (e.g DNPSlave.sln). The source code for the sample application resides in a subdirectory under <*installdir*> (e.g., DNPSlave) . You should refer to this sample application as an example when completing the steps in the Quick Start Guide and building your own application.

The Source Code Libraries include source code for the target layer on Windows and on Linux. This source code may be used in your target application along with the rest of the Source Code Library.

See more information about Building the SCL in chapter 4.

2.4 Modify Basic Type Definitions

The file *'tmwscl/utils/tmwtypes.h'* includes definitions for data types that are used throughout the SCL.

These files should be reviewed to make sure the definitions are compatible with the target platform. Most target platforms will require few, if any, changes to this file. However, this file may need to be modified if the data types are of different sizes than defined in this file.

If any changes to this file are required, they will usually be limited to the `/* Type Definitions */` section.

It is important that TMWTYPES_ULONG and TMWTYPES_LONG are 32 bit values as the SCL was written to use that size data.

2.5 Modify SCL Configuration Files

Next, review the TMW SCL configuration files shown in **Table 2**.

Table 2. Configuration Files

Protocol	May Require Modifications	Requires Modifications
DNP Master	tmwtypes.h tmwcnfg.h dnpcnfg.h mdnpcnfg.h mdnpdata.h (supported objects only)	tmwtarg.c mdnpdata.c
DNP Slave	tmwtypes.h tmwcnfg.h dnpcnfg.h sdnpcnfg.h sdnpdata.h (supported objects only)	tmwtarg.c sdnpdata.c
IEC 60870-5-1-101 Master	tmwtypes.h tmwcnfg.h m14cnfg.h m14data.h (supported data types only)	tmwtarg.c m14data.c
IEC 60870-5-1-101 Slave	tmwtypes.h tmwcnfg.h s14cnfg.h s14data.h (supported data types only)	tmwtarg.c s14data.c
IEC 60870-5-102 Master	tmwtypes.h tmwcnfg.h m102cnfg.h m2data.h (supported data types only)	tmwtarg.c m2data.c
IEC 60870-5-102 Slave	tmwtypes.h tmwcnfg.h s102cnfg.h s2data.h (supported data types only)	tmwtarg.c s2data.c
IEC 60870-6-103 Master	tmwtypes.h tmwcnfg.h m13cnfg.h m13data.h (supported data types only)	tmwtarg.c m3data.c
IEC 60870-5-103 Slave	tmwtypes.h tmwcnfg.h	tmwtarg.c s3data.c

	s13cnfg.h s13data.h (supported data types only)	
IEC 60870-5-104 Master	tmwtypes.h tmwcnfg.h m14cnfg.h m14data.h (supported data types only)	tmwtarg.c s14data.c
IEC 60870-6-104 Slave	tmwtypes.h tmwcnfg.h s14cnfg.h s14data.h (supported data types only)	tmwtarg.c s14data.c
Modbus Master	tmwtypes.h mmbdata.h mmbcnfg.h tmwcnfg.h (supported function codes only)	tmwtarg.c mmbdata.c
Modbus Slave	tmwtypes.h tmwcnfg.h smbcnfg.h smbdata.h (supported function codes only)	tmwtarg.c smbdata.c

These files are used to specify support for generic TMW SCL features and define how memory will be allocated (static or dynamic), as well as define maximum sizes for various arrays used by the TMW SCL and the supported data types.

2.6 Add Calls to the SCL

Calls into the Source Code Library are limited to the following areas: timer initialization; application initialization; database initialization (for Master sessions) and managing channels, sessions, and (for IEC 60870-5 protocols) sectors.

Timer, application, and database initialization is described in the following sections. Managing channels, sessions, and sectors is described in Section 2.8.

Protocol specific information is provided in the protocol-specific document (e.g., DNP Slave.doc).

2.6.1 Timer Initialization

As described in Section 3.2.11, by default the Source Code Library uses a single timer. If this configuration is used, the *tmwtimer_initialize* function must be called once before any timers are started. Note that the polled timer implementation (in tmwpltmr.c) also uses this configuration.

2.6.2 Application Initialization

All protocols share a single application source file: '*tmwscl/utils/tmwappl.h/c'*. This file contains an initialization routine, *'tmwappl_initApplication'*, which must be called once to initialize an application context. It also contains a routine *'tmwappl_checkForInput'* that should be called periodically to see if received data is available on any of the communication channels.

As an alternative to periodically calling *'tmwappl_checkForInput'* to poll the input channels for data, the TMW SCL can be set up to run entirely event driven. For more details see the section on polled versus event driven implementations below.

2.6.3 Database Initialization

For Master applications, if the Asynchronous database is used (TMWCNFG_SUPPORT_ASYNCH_DB is TMWDEFS_TRUE), then the database queue must be initialized by calling the *tmwdb_init* function (see tmwdb.h).

2.7 Implement the SCL Target Interface

The interface between the SCL and the target platform is defined in the header file '*tmwscl/utils/tmwtarg.h'*. This interface consists of a set of functions that must be modified to properly interface to the target platform. The SCL is packaged with target platform support for Windows and Linux. There is also a Sample target layer which contains function templates that can be expanded to support additional target platforms.

The directories *'tmwscl/tmwtarg/WinIoTarg'* and *'tmwscl/tmwtarg/LinIoTarg'* are TMW Target implementation files that provide our mapping to the Windows and Linux target implementations. If your target is running either of these operating systems, these target layers can be used without modification.

The directory '*tmwscl/tmwtarg/SampleIoTarg'* contains target layer template files that can be expanded to support additional target(s). The file tmwtargcnfg.h is used to control features supported by the target layer. Features such as TCP, TLS, or UDP can be turned off to reduce the porting effort if they are not needed by your application. The remaining functions will need to be implemented as appropriate for the target operating system

The '*tmwscl/utils/tmwtargp.h'* header file specifies Target Private functions that are used by the sample applications but are NOT required by the protocol libraries themselves. The SCL is shipped with implementations of these functions for Windows and Linux in the tmwtargp.c files. Implementing these tmwtarp_xxx functions to port the protocol libraries to your target environment is not required.

The Modbus protocol has additional requirements regarding detection of the inter-character and inter-frame timeouts, which may require higher precision timers from the operating system. This functionality is accomplished in the target layer, rather than the SCL. (Since the SCL is generic, it cannot include target layer implementations). These Modbus-specific requirements are noted in the function header for tmwtarg_receive() and some of the configuration structures.

The DNP3 protocol also has additional requirements regarding UDP and Dual End Point functionality (both Client and Server). This is described in detail later in the DNP3 IP Networking section of this manual.

NOTE: Within the user modifiable files, there are compiler directives for internal use by Triangle MicroWorks, Inc.

The TMW_PRIVATE directive indicates that the following code is for internal TMW use. This code is used for debugging or internal use. TMW_PRIVATE should NOT be defined in your implementation.

Some default target source code implementations are provided, that can be used in your system if you are writing an application to run on Microsoft Windows or Linux. These source code implementations (in particular LinIoTarg) can also serve as examples if you need to implement your own target implementation. The following defines apply to these target implementations.

- TMW_WTK_TARGET is for the "windows toolkit" target; it is also used by the example programs as an indicator that they should use the WinIoTarg DLLS for the tmwtarg functions. The source code for this Windows target can be found in the '*tmwscl/tmwtarg/WinIoTarg'* directory.

- TMW_LINUX_TARGET is for the Linux target; it is also used by the example programs as an indicator that they should use the LinIoTarg source code for the tmwtarg functions. The source code for this Linux target can be found in *'tmwscl/tmwtarg/LinIoTarg'* directory.

2.8 Manage Channels, Sessions, and Sectors

Channels, sessions, and sectors (if required) are managed by routines found in '*tmwscl/<family>/<scl>chnl.h/c'*, '*tmwscl/<family>/<scl>sesn.h/c'*, and '*tmwscl/<family>/<scl>sctr.h/c'*.

These files provide methods for opening, modifying, and closing channels, sessions, or sectors. Each of these methods takes as an argument one or more data structures that contain configuration information. Each of these structures should be initialized using the corresponding *'xxx_initConfig'* subroutine before calling the corresponding open routine.

Note that the *'xxx_initConfig'* routines initialize the data structures to default values. In order to ensure that this function will always return reasonable default values (even if the structure is modified in a future release), you should not modify this function. Instead, after calling these functions, the target application should modify the structures with any specific values required by your application.

To support multiple channels/session/sectors, allocate a set of data structures for each channel/session/sector, and repeat the initialization steps for each channel/session/sector structure Then open each channel/session/sector as needed.

At this point it should be possible to test communications with a remote device using the simulated database shipped as part of the SCL.

2.9 Integrate the SCL with your Database

The interface between the SCL and the device database is a set of functions defined in *'tmwscl/<family>/<scl>data.h/c'*. When the SCL is shipped, these functions interface to a simple "simulated" database. This simulated database should not be used in your final implementation. They are simply there to allow you to get the Source Code Library compiled and running quickly. These simulated databases do not provide a documented interface to be used by your application. The database should be replaced by your own implementation.

The default implementations in *'tmwscl/<family>/<scl>data.c'* must be modified to access the target database instead of the simulated database. Modify the appropriate file listed in Table 3 to call your database functions.

Table 3: Database Integration Files

Protocol	Function Prototypes (No modifications to interface required)	Database File to Modify
MDNP	mdnpdata.h	mdnpdata.c
SDNP	sdnpdata.h	sdnpdata.c
IEC 60870-5-101 Master	m14data.h	m14data.c
IEC 60870-5-101 Slave	s14data.h	s14data.c
IEC 60870-5-102 Master	m2data.h	m2data.c
IEC 60870-5-102 Slave	s2data.h	s2data.c
IEC 60870-5-103 Master	s3data.h	s3data.c
IEC 60870-5-103 Slave	s3data.h	s3data.c
IEC 60870-5-104 Master	m14data.h	m14data.c
IEC 60870-5-104 Slave	s14data.h	s14data.c
Modbus Master	mmbdata.h	mmbdata.c
Modbus Slave	smbdata.h	smbdata.c

2.10 Issue Commands (Master Sessions)

For Master sessions, commands are issued by calling routines in the files *'tmwscl/<family>/<scl>brm.h/c'*. Each of the routines in these files returns a handle that can be used to identify the request.

When the request is completed, the SCL will call a user callback function. The callback function arguments include the request and pointers to the transmitted request and the received response.

Information on protocol-specific commands are given in the protocol-specific document.

More details on issuing requests can be found in the master specific section of this manual.

2.11 Add Support for Change Events (Slave Sessions)

For Slave sessions, the SCL can be configured to periodically scan for change events. Alternately, events can be generated directly by calling the appropriate *'xxx_addEvent'* function, where *'xxx'* specifies the type of event being generated.

Enabling SCL scanning for change events is a configuration option that is specified when opening the session or sector.

This topic will be described in more detail in the slave specific section of this manual.

2.12 Test your Implementation

To test your implementation, you will need a system or application to act as the 'opposite' end of the communication link. For example, if you are implementing a Slave device, you will need a corresponding Master.

Triangle MicroWorks provides a flexible, scriptable test harness that can be used for this purpose. The TMW Communication Protocol Test Harness is a Windows application that acts as a simple Master or Slave device and can be programmed with an automated test sequence through a scripting capability.

The protocol analyzer output from the Test Harness allows you to verify the proper operation of the device you are testing. It also allows you to generate a reference protocol analyzer output file that can be used for comparison to later tests to verify only intended modifications have occurred.

For more information please see the Triangle MicroWorks, Inc. web site at *'http://www.TriangleMicroWorks.com'*.

Chapter 3 Overview

This section provides an overview of the TMW SCL architecture. This information will be of interest to all TMW SCL users. Unless otherwise noted, the information in this section pertains to all of the TMW SCLs.

3.1 Coding Conventions

All TMW SCLs follow the same coding conventions. These conventions are used to specify file, function, and variable names, as well as other details. While a detailed knowledge of TMW coding conventions is not required, a few tips will help users more efficiently understand the TMW source code.

All filenames follow the standard '8.3' filename convention (i.e., the filename is 8 characters followed by a 3 character extension).

Most of the files that ship with the Source Code Library are for internal SCL use. These files should not be modified. The files that require modification are listed in Table 2.

All global functions and data begin with the filename followed by an underscore '_' followed by the function's name (e.g., tmwsesn_openSession). Local functions begin with a '_' followed by the function's name (e.g., _openSession).

3.2 Architecture

This section describes the overall architecture of a TMW SCL. All of the TMW SCLs share a common architecture and leverage common code. Not all of the information in this section is required to successfully integrate a TMW SCL. However, the information in this section is useful in understanding how the library behaves.

3.2.1 Layers

The TMW SCL architecture is broken into four layers: Physical, Link, Transport, and Application. Each of these layers provides different functionality and is designed to work completely independent of the layers above and below it in the protocol stack. While a thorough understanding of the different layers is not essential, a basic knowledge will help in implementing, tuning, and debugging a TMW SCL implementation.

It is important to note that although the TMW SCL architecture is designed to support four layers, not all protocols require or use all of the layers.

In a typical device, the user will not interface directly with the different layers. The Physical, Link, and Transport layers are all managed as a single channel, and the Application layer is managed by Sessions and Sectors.

The typical implementation does not require direct management of the Physical, Link, and Transport layers. Typically, the configuration parameters for the Physical, Link, and Transport layers are specified as part of the channel configuration.

Advanced implementations can leverage the layering to provide custom solutions. The interface between each layer is protocol and implementation independent, allowing different layer implementations to be interchanged as required. In addition, it provides the option of TMW or customer implemented custom layers that can be used in place of the standard layers. This possibility is discussed more in Section 3.3.

3.2.1.1 Physical Layer

The physical layer manages the target input/output (I/O) device. It accesses the actual physical device through the appropriate routines in *'tmwscl/utils/tmwtarg.h/c'*. The physical layer is responsible for:

1) Opening and closing the physical device
2) Transmitting and receiving data on the physical channel
3) Handling low level I/O errors when transmitting or receiving data

The physical layer manages the target input/output device by opening and closing the device as required, configuring the device, etc. The physical layer optionally manages parameters related to the transmission and reception of bytes, such as inter-character timeouts, transmission delays, etc.

When the Link layer needs to transmit data, it calls the Physical Layer transmit routine. The Physical layer determines if the channel is ready to transmit, and if so, writes the data to the channel using the appropriate target routines.

To receive bytes from an input I/O channel, the target application must call the physical layer receive routine, either directly or through the application layer *'tmwappl_checkForInput'* function. The physical layer first determines from the link layer how many bytes to read. It then reads the physical device through the appropriate target routines. If any data are read, the physical layer then passes it to the link layer to be parsed.

3.2.1.2 Link Layer

The link layer provides low-level framing, error detection, and retries. The link layer receives frames from the transport layer, adds the required link layer header and error detection information, and sends the result to the physical layer.

As the physical layer receives bytes, it passes them to the link layer, which validates the error detection, removes the link header, and sends received frames to the transport layer.

In general, the link layer pulls data from the layer above it when the link layer is available to transmit data. The application or transport layer calls a link layer method to tell the link layer that data is available. If the link layer is able to transmit, it will immediately ask the application/transport layer for the data. If the link layer is not currently available, it will ask the application/transport layer for the data as soon as the link layer becomes available.

The link layer receives bytes from the physical layer through a parsing routine. It then assembles frames as required by the protocol and passes the frames on to the transport or application layer.

3.2.1.3 Transport Layer

The transport layer, if used, breaks application layer messages into link layer frames. It also rebuilds application layer messages as they are received. The transport layer includes the sequencing required to guarantee proper transmission and reception of an entire application layer fragment.

As with the link layer, the transport layer pulls data from the layer above when the transport layer is free to transmit data.

The transport layer assembles received frames into application layer fragments and calls the application layer parsing routine to process the fragments.

3.2.1.4 Application Layer

The application layer provides application specific functionality and generates application messages to transmit. It also processes received application messages. This layer is responsible for generating and processing protocol specific requests.

3.2.2 Data Model

Each TMW SCL is managed through a set of hierarchical structures called channels, sessions, sectors, and points. Figure 2 below illustrates how each of these structures maps to a typical application. In addition to managing the SCL, these structures are used to access specific data points in the system.

In Figure 2, the master station can communicate using multiple communication channels (shown as COM1 and COM2 below), to multiple remote slave devices. Each remote slave device may subdivide its data into sectors, each of which may contain multiple data points. This model and these terms are further described in the sections below.

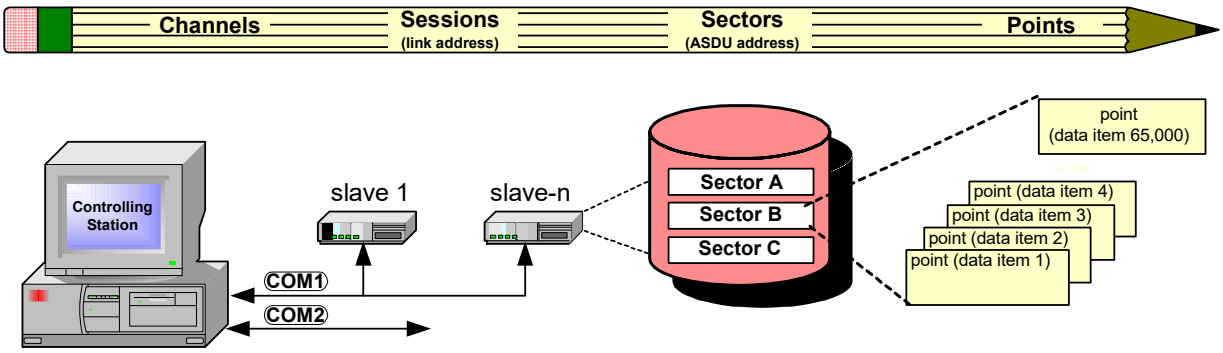


Figure 2: Triangle MicroWorks SCL Data Model

3.2.3 Channels

As mentioned above, the TMW SCLs encapsulate the physical, link, and transport layers into a single entity called a channel. Most implementations deal with channels rather than

dealing directly with a specific physical, link, or transport layer implementation. Figure 3 below illustrates communications channels in a typical implementation.

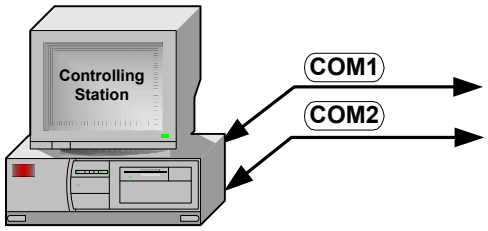


Figure 3: Illustration of Communication Channels

The SCL exchanges data with remote devices through one or more Communication Channels. The communication channels illustrated in Figure 3 are COM1 and COM2.

The communication channels may be physically discrete channels or logically discrete channels that share a physical connection. An example of a logical communication channel is a TCP/IP "serial pipe", which may share a physical 10BaseT interface with other TCP/IP serial pipes.

3.2.4 Sessions

Each channel can have one or more sessions. A session is a communication connection between the master device and one remote slave device. The total number of sessions is the total number of remote slave devices over all communication channels with which the master communicates. Figure 4 below illustrates multiple sessions in a multi-channel setup.

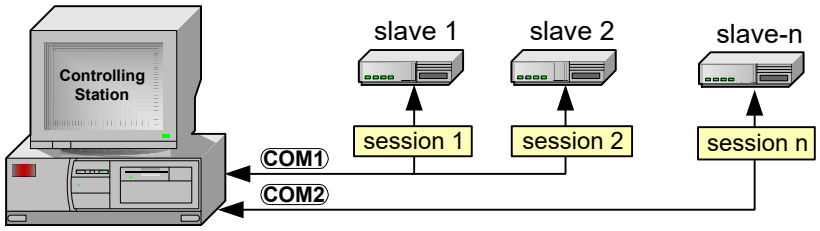


Figure 4: Illustration of Communication Sessions

When multiple sessions use the same communication channel (a multi-drop topology), the link address for the session must be unique among all sessions using that channel. In messages transmitted or received on a communication channel, the link address identifies the remote Slave device as either the destination of requests or the source of data.

Communication protocol specifications supported by the TMW SCLs use the ISO protocol standards for the Application (7), Data Link (2), and Physical (1) Layers. It should be noted that "session" in this product and document does *NOT* refer to the defined ISO Session(5) Layer.

3.2.5 Sectors

In IEC 60870-5 protocols, remote Slave devices may contain collections of data, called sectors. DNP3 does not support sectors. Figure 5 below illustrates a slave device that collects data into three sectors.

An example of a remote Slave device with multiple sectors is a data concentrator device that uses sectors to image downstream devices; i.e., each sector represents a single downstream device. The data within a sector represents data collected from the corresponding downstream device. Also, command requests sent by the SCL to a sector within the data concentrator may also result in commands being transmitted to the corresponding downstream device.

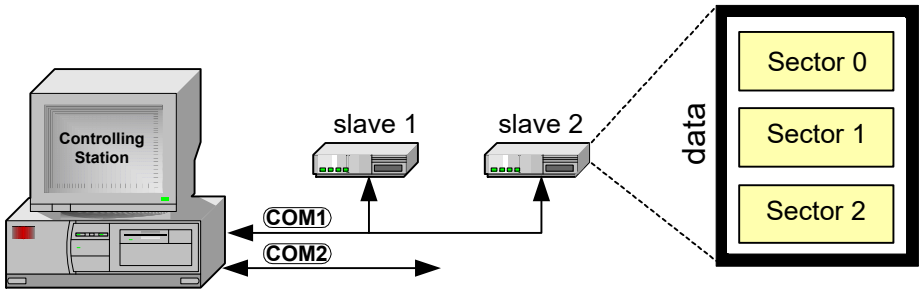


Figure 5: Illustration of Multiple Sectors

The Common Address of ASDU field may be either a single-octet or a two-octet field. It is configured for each session when the session is opened. Although it is possible for the size of the field to be different for each device (even on the same communication channel), the IEC 60870-5 protocol specifications dictate that the field size must be the same for all devices in the system (across all communication channels).

A Common Address of ASDU value of zero is not used. The value 255 (or 0xFF) for a single-octet address or 65535 (or 0xFFFF) for a two-octet address is used as a broadcast or global address in messages sent by a controlling station. A slave receiving such a message should process the message as if it were addressed to each sector (each different Common Address of ASDU) in the slave.

3.2.6 Data Points

A data point is an indivisible data object in the SCL and the level at which data is transferred to the target application database manager. In the IEC 60870-5 protocols, an Information Object Address is used to identify data points. The DNP3 protocol uses point numbers to identify data points.

3.2.6.1 Information Object Address

The Information Object Address field in IEC 60870-5-101 may be a one-, two- or three-octet field. The length of the IOA is configured when the session is opened. Although it is possible for size of the field to be different for each device (even on the same

communication channel), IEC 60870-5 protocol specifications dictate that the field size must be the same for all devices in the system (across all communication channels).

Information Object Address zero is not assigned to any data object. All other values are permitted. If a three-octet Information Object Address is used, the address values are a sparse mapping of only 65535 different values.

Information Object Addresses are not required to be sequential. They are typically organized in a "structured" format, where each data type occupies a range of values. For example, all binary inputs could be placed in the range 2000 to 2999, and all binary outputs in the range 3000 to 3999.

In the IEC 60870-5-103 protocol, the Information Number identifies a collection of related data, so an Element Index is required to uniquely identify each data point. In addition, a remote device can define the same information number for different protective relay "Function Types." Therefore, in the IEC 60870-5-103 protocol, three fields are required to identify each data point: Information Number, Element Index, and Function Type.

The IEC 60870-5 protocols categorize every data point in a device into a data type. For example, in IEC 60870-5-101, "single-point information" is a 1-bit binary value, and "Measured value, normalized value" is a 16-bit integer value. Monitored inputs and controlled outputs are considered to be separate types. Hence, if a data entity could be both set (controlled) and monitored, it must be assigned two data points: a control point and a monitor point, which must have different Information Object Address values.

Example Address Map

Consider a typical small slave device, having a fixed data object mapping, as shown in Table 4.

Table 4: Example IEC 60870-5 Address Map

Data Object	Object type	Information Object Address
CB Status	Double-Point Information	1
Bus Isolator	Double-Point Information	2
Feeder isolator	Double-Point Information	3
Bus Earth Switch	Single-Point Information	4
Feeder Earth Switch	Single-Point Information	5
Battery Fault	Single-Point Information	6
Control Supply	Single-Point Information	7
Auto Reclose In Progress	Single-Point Information	8
Bus A-Phase KV	Measured Value, Scaled	75
Bus B-Phase KV	Measured Value, Scaled	76
Bus C-Phase KV	Measured Value, Scaled	77
Feeder A-Phase KV	Measured Value, Scaled	78
Feeder B-Phase KV	Measured Value, Scaled	79
Feeder C-Phase KV	Measured Value, Scaled	80
A-Phase Amps	Measured Value, Scaled	81
B-Phase Amps	Measured Value, Scaled	82
C-Phase Amps	Measured Value, Scaled	83
CB Control	Double Command	128
Bus Isolator Control	Double Command	129
Feeder Isolator Control	Double Command	130
Bus Earth Switch Control	Single Command	171
Feeder Earth Switch Control	Single Command	172
Auto Reclose Enable	Single Command	173
Trip Current	Set Point Command, Scaled	190

Note from this example mapping:

- Each Information Object Address value is unique. The values may be assigned in any order, and do not need to form a contiguous numeric sequence.
- In IEC 60870-5 protocols, message efficiency is maximized if objects with the same object type are assigned consecutive Information Object Address values.
- If an association is to be established between a controllable object and a monitored object, they must have corresponding object types (e.g., Single Command corresponds to Single Information; Set Point Command, Scaled Value corresponds to Measured Value, Scaled Value; etc.).
- For instance, in the example mapping, the CB Status (Double-Point Information, Information Object Address 1) and CB Control (Double Command, Information Object Address 128) could be associated. When the

Master issues a command to Information Object Address 128, it would expect to see a corresponding change in the value of Information Object Address 1. If the CB Control had been defined as a single command, this association would not be possible because the control object type would not match the monitored object type.

- For a master to operate with this slave device, both master and slave must be configured to use the same Data Link Address and Common Address of ASDU, and the data object types and Information object Address values of the slave device must be configured in the Master. The master is also configured to associate any controlled objects and their corresponding indications, as appropriate.
- If two identical devices of this type were to be used in a system, they must be assigned different Common Address of ASDU values, to conform to the requirement that the combination of Common Address of ASDU and Information Object Address is unique across the whole system.

3.2.6.2 DNP3 Point Numbers

DNP3 uses a point number to identify a specific data point. Point numbers start with 0 for each data type and can go up to 65535. Unlike the IEC 60870-5 protocols, point numbers are associated with the data type, so a given number can be used in multiple data types.

The DNP Technical Committee strongly recommends using contiguous point numbers, starting at 0 for each data type because some DNP3 Master implementations allocate contiguous memory, from point 0 to the last point number for each data type.

While we do not recommend this approach in new DNP3 Master designs, we suggest that a DNP3 Slave device be designed with a contiguous memory map to avoid wasting memory in some DNP3 Master implementations.

In some implementations, it may be desirable to have small gaps in point numbers. Such gaps are easily accommodated by the SCL, as described in the header comments in sdnpdata.h.

3.2.7 Target Interface

The target interface provides the interface between the TMW SCL and the target hardware and software. This interface is a combination of macros and subroutines that must be modified as required by the target platform. The functionality includes timers, memory allocation, physical input/output, etc. Each of these will be discussed in more detail below.

3.2.8 Database Interface

The database interface provides the interface between the TMW SCL and the target systems data. For a slave device this will typically be tied directly to the actual data points in hardware. For a master device this will typically interface with a database of some sort.

The database interface consists of several subroutines that must be modified to interface to the actual target database. In general there will be several routines for each supported data type. Routines for data types that will not be supported on the target platform do not need to be implemented.

For each data type, there is a single routine that is used to get a handle to the desired data point. This routine returns a pointer to a void type that can contain any structure appropriate for the target database. For simple databases this could simply be a pointer to the memory location that contains the data value. Once the point's handle is obtained, are one or more routines provide access to information about the data point and the current value of the point.

3.2.9 SCL Application Interface

The SCL Application Interface consists of subroutines called by the target application to instruct the TMW SCL to perform specific operations. For a slave device this is limited to opening and configuring channels, sessions, and sectors, and possibly generating events. For a master device this includes opening and configuring channels, sessions, and sectors, and issuing requests to be sent to the remote devices.

In addition to calling into the SCL the application interface includes a number of callback functions that can be used to invoke functions within the target application when events take place in the SCL. These callback functions are called when a request completes, an application layer fragment is received, a statistically significant event occurs, etc. Each of these will be discussed in more detail below.

3.2.9.1 Channel Callback Function

The SCL supports a channel callback function (`TMWTARG_CHANNEL_CALLBACK_FUNC`) defined in tmwtarg.h to allow the target layer to tell the SCL that the connection has opened or closed. Support for this function is optional for most protocols, but is required for IEC 60870-5-104 so that the SCL can properly manage sequence numbers.

If the target layer returns `TMWDEFS_FALSE` from tmwtarg_openChannel(), then the SCL will continue to periodically retry to open the connection. However, the channel callback function can be used to notify the SCL immediately when the channel becomes open.

For example, this function can be used to notify the SCL when a TCP/IP connection has become established. The initial call to tmwtarg_openChannel can start the listener (or connector) and return `TMWDEFS_FALSE`. When the connection is actually established, the target can notify the SCL via the channel callback function.

The channel callback function can also be used for modem channels. The target can return `TMWDEFS_FALSE` from the call to tmwtarg_openChannel and begin the dialing process. When the modem successfully dials and establishes a connection with the remote station, the target layer can call the channel callback function to let the SCL know that the channel is open.

Similarly, if a connection is successfully opened but later closes before the SCL calls tmwtarg_closeChannel, the target layer can call the channel callback function to notify the SCL.

A pointer to the channel callback function is passed to the target in the target configuration structure (TMWTARG_CONFIG) parameter in the call to tmwtarg_initChannel. This pointer must be maintained by the target if it is going to utilize the channel callback function.

One approach is to copy these parameters from the tmwtarg_config structure into the target-specific channel context in tmwtarg_initChannel. Note that the tmwtarg_initChannel function returns a pointer to the channel context. This context is then passed to all subsequent tmwtarg functions. By copying the pointer to the channel callback function and the channel callback parameter into the channel context, all subsequent tmwtarg routines are given access to these values.

For example, the code in tmwtarg_initChannel might look something like:

```
.
.
.
myChannelContext->pChannelCallback = pTmwConfig->pChannelCallback;
myChannelContext->pCallbackParam = pTmwConfig->pCallbackParam;
return(myChannelContext);
```

Then, when the target detects that the connection has opened, it calls the channel callback function:

```
pContext->pChannelCallback(pContext->pCallbackParam,
TMWDEFS_TRUE, TMWDEFS_TARG_OC_SUCCESS);
```

Similarly, the target can also call the channel callback function if it detects that the connection has closed:

pContext->pChannelCallback(pContext->pCallbackParam,
TMWDEFS_FALSE, TMWDEFS_TARG_OC_SUCCESS);

3.2.9.2 Command Response Callback Function

The Source Code Library provides for a user-defined callback function to indicate when a response to a command has been received or the command has completed.

For a DNP Master this function (DNPCHNL_CALLBACK_FUNC) is called when a DNP3 any responses are received for this command or when this command completes. The parameter (DNPCHNL_RESPONSE_INFO) provides information about the response (e.g., the IIN bit settings, the return status, etc.)

For IEC or Modbus Masters this function (I870CHNL_CALLBACK_FUNC or MBCHNL_CALLBACK_FUNC) is called when the command completes.

3.2.10 Event Driven vs. Polled Data

As mentioned above, an TMW SCL implementation can use either event driven or polled mode to determine if data has been received on a channel. The polled implementation is the easiest and probably widest used in slave devices. Master devices, which typically have more stringent requirements on processor utilization, may implement an event driven receive data model.

The target layers for Windows and Linux that are included with the SCL include support for both modes. If you are porting the SCL to another operating system, the effort can be simplified by selecting a single mode of operation and only supporting one. The default mode of operation is polled.

For either model, the user application first calls the application layer initialization routine tmwappl_initialization found in *'tmwsclutils/tmwappl.h/c'*.

- To implement a polled design the target application periodically calls the 'tmwappl_checkForInput' routine to see if any of the communications channels have received data. The exact rate at which the 'tmwappl_checkForInput' routine must be called is somewhat system dependent but is generally on the order of 50 ms.

- To implement an event driven system the application must set the configuration parameter targTCP.polledMode to TMWDEFS_FALSE prior to calling xxx_openChannel. The target layer will then call into the SCL with any data as it is received. This is typically done using a receive thread in the target layer using some form of select function to determine when data is available on a channel.

  A callback function (pReceiveCallbackFunc) and parameter (pCallbackParam) are passed to the target layer by the tmwtarg_openChannel function. In the event driven mode, the target should call pReceiveCallbackFunc with a parameter of pCallbackParam when it has received data available on the channel. This will cause the SCL to call tmwtarg_receive and process the received data.

3.2.11 Timers

The TMW SCL, in its default configuration, requires only a single event timer. This timer must be able to invoke a specified callback function after a specified number of milliseconds. The timer must also support a cancel function.

The entire SCL will potentially run in the context of this timer callback function. This could call a user registered callback function or target function such as transmit and receive. Code for any channel could be run in this callback context. This timer callback function should be called from a context where the SCL is allowed to run. If for example, an OS event timer is used and it is not desirable to run the SCL in this context, the implementation should instead call the SCL timer callback from a context that is acceptable.

If an event timer is not available a simple polled timer has been implemented in *'tmwscl/utils/tmwpltmr.h/c'.* If this polled timer module is used, the user must call the *tmwpltmr_checkTimer* routine periodically to determine if the timer has expired.

The TMW SCL can also be configured to allow a separate timer queue per channel or thread. This configuration requires a separate timer per channel (the default polled timer implementation cannot be used). Defining TMWCNFG_MULTIPLE_TIMER_QS allows only expired timers and SCL code (including user callback and target functions) for the specified channel to be executed during the callback context.

To associate a timer with a thread, call the pCallback function from the appropriate (channel) thread. In other words, when a timer expires, notify the appropriate thread to call pCallback. Depending on the OS, the target code may be able to set an event on the timer, and that event would wake up the thread, which would call pCallback.

This can be desirable in some multithreaded implementations. See the section on Threads for more details.

It is important to specifically verify proper timer functionality. Since timers are mostly used for error detection and recovery, an otherwise functioning system may appear to run properly even though timers are not functioning at all or are running with the incorrect resolution.

If the default SCL implementation of polled timers is being used, tmwtimer_initialize() must be called once at startup, and tmwpltmr_checkTimer() must be called periodically.

_timerCallback in tmwtimer.c is called by the SCL when the polled timer or the single system timer that was requested by the SCL has expired.

3.2.12 Memory Allocation

All TMW Source Code Libraries can be implemented using either static or dynamic memory allocation. The advantage of dynamic memory allocation is that you do not have to provide predefined maximum limits for data structures (i.e. channel, session, sector control structures, message buffers, event buffers, etc.) and the amount of memory used at any given time is exactly the amount required at that time (if you close all the channels, sessions and sectors, all associated memory is freed). The advantage of static memory is that the maximum memory size is known at startup and will never change.

It should be noted that the static memory option has exactly the same issues as dynamic memory in that if you attempt to open more channels, sessions, and/or sectors, or store more events, than will fit in the statically allocated arrays the operation will fail for 'lack of memory'. Hence the exact same checks and error conditions are used in the static versus dynamic cases. Additionally, the dynamic memory allocation in the TMW SCLs has the option of specifying maximum limits for each structure that will be allocated hence allowing you to specify a maximum memory size a priori, but only using the actual amount of memory required at any given time.

The following list defines when memory is allocated and freed in the TMW SCL.

Master and Slave SCLs:

- Whenever channels, sessions, and sectors are opened and closed

Slave SCLs:

- When the slave sends a response to the master a message buffer is allocated
- In 104 a message buffer is allocated for each transmit since there can be multiple outstanding messages
- Whenever a new event is generated

Master SCLs:

- Whenever the user issues a new request

3.2.12.1 Allocation Only at Startup

There is an additional memory allocation option related to dynamic memory allocation that allocates all memory needed by the SCL at startup time. This allows the decision of how much memory to allocate for each memory pool to be done at run time. The SCL will allocate a single block of memory for each pool and then manage allocations and deallocations from those pools. Many embedded systems will find this a useful mechanism to set aside a specified amount of memory at startup, avoiding fragmentation of memory, but not requiring a compile time decision of memory needs.

3.2.13 Diagnostics

The TMW SCLs support the generation of diagnostic information that can be used to analyze the protocol message traffic. All TMW SCL diagnostic information is passed to the target device through a single routine called 'tmwtarg_putDiagString'. This routine takes a pointer to a string that contains the diagnostic message and a pointer to a structure that identifies the source of the message. This message source can be used to filter or redirect the diagnostic information.

Diagnostic support can be removed entirely by setting the TMWCNFG_SUPPORT_DIAG macro to TMWDEFS_FALSE in *'tmwscl/utils/tmwcnfg.h'*. This will reduce the memory footprint of the SCL significantly since it removes all of the static string declarations required to support the diagnostics.

3.2.14 Condition Notification and Statistics

The TMW SCLs generate events that can be monitored by the user. These events are generated whenever something significant happens in the source code library. This includes any error, transmission or reception of bytes, frames, and fragments, as well as others. These events can be used to maintain statistics, invoke additional processing, or whatever is required by the user's implementation.

This feature is exposed through user callbacks that can be added to the channel and/or session. Details can be found in the associated header files *'tmwscl/utils/tmwchnl.h'* and *'tmwscl/utils/tmwsesn.h'*. As an example, the code to capture and process channel events would look something like the following:

```
/* Channel Event Callback */
static void _chnlEventCallback(
  void *pCallbackParam,
  TMWCHNL_STAT_EVENT eventType,
```

```
  void *pEventData)
{
  switch(eventType)
  {
  case TMWCHNL_STAT_FRAME_SENT:
    _framesSent += 1;
    break;

  case TMWCHNL_STAT_FRAME_RECEIVED:
    _framesReceived += 1;
    break;
  }
}

/* In channel initialization code */
tmwchnl_setStatCallback(pChannel,
   _chnlEventCallback, myCallbackData);
```

Statistics support can be removed entirely by setting the TMWCNFG_SUPPORT_STATS macro to TMWDEFS_FALSE in *'tmwscl/utils/tmwcnfg.h'*.

3.2.15 Threads

All TMW SCLs support operation in a multi-threaded environment. There are several techniques that can be used to implement the TMW SCL in a multi-threaded environment. The first is simply to guarantee that all calls into the TMW SCL occur on the same thread. This includes all calls into the SCL application layer (including calls into the Build Request Module to generate new requests), as well as all callbacks from timers and other target routines.

A second technique would be to limit calls into the TMW SCL to at most one thread at a time. This can be accomplished by a single semaphore that must be locked before calling any SCL function and unlocked when the SCL function returns.

The above techniques do not rely on features of the TMW SCL to support multiple threads and hence all SCL processing is done sequentially. If desired the TMW SCL can support multiple threads of execution (i.e. each channel can be processed on a different thread).

In order to support execution on multiple threads, critical resources within the SCL are locked before use and unlocked after use. The TMW SCLs provide resource locking per channel. Hence different threads can be processing information from different channels simultaneously. Additionally, the database update queue is protected if Asynchronous Database Updates are being used as described below.

To support locking, TMWCNFG_SUPPORT_THREADS must be defined as TMWDEFS_TRUE in *tmwscl/utils/tmwcnfg.h*. In addition, the type for a resource lock must be defined in *tmwscl/utils/tmwtarg.h*, and the appropriate lock and unlock routines must be implemented in *tmwscl/utils/tmwtarg.c*.

The SCL does not really know or care about threads. If you configure TMWCNFG_SUPPORT_THREADS to TMWDEFS_TRUE, the SCL will call lockSection and unlockSection to protect shared resources. This allows you to call any SCL interface

function from multiple threads without concern for one thread interrupting another and modifying shared resources while inside of the SCL code. In other words, if you compile in thread support, it is safe to call the SCL functions from any thread.

The SCL may run in the context of multiple threads. One thread could call the timer callback function. A separate thread could call the SCL callback function to indicate that a channel has opened or closed. Another thread could call the tmwappl_checkForInput function to see if any data has been received, or to call the SCL callback function if data has been received. Event functions (on slaves) or brm request functions (on masters) can be called from yet another thread.

This means that if you want multiple threads, you must create multiple threads. You need to determine how many threads to create and what code should run in the context of each thread. The decision of what each thread does can be based completely upon your needs. However, understanding how the SCL functions may help you architect things to better take advantage of multiple threads.

The SCL provides locks around an entire channel, the entire memory pool and the timer queue(s). Locking around the entire channel means that if you are calling brm request functions (master), addEvent functions (slave), data received functions, and timer callback functions from separate threads, these threads may block each other. Instead, you might want to call the SCL functions for a specific channel from a single thread. For example, you could have your channel thread wait on a timer event, a data received event, and user request event. When one of those events wakes up the thread, it would then call the appropriate SCL function from within the context of that single thread.

Pay particular attention to timer functionality. The SCL can operate with only a single system timer for all channels or a separate system timer per channel. With a separate thread per channel it might also make sense to set TMWCNFG_MULTI_TIMER_QS to TMWDEFS_TRUE to provide for a separate timer queue per channel. In either case, remember that the SCL will run in the context of where the timer callback is called from. This means if the timer callback is called from a single timer thread – or even worse in the context of the Operating System callback – and you have multiple channels all using a single timer queue, the SCL code to process a timer for all channels will run in that same context.

It is also important when running in a multi-threaded environment to make sure callbacks from the TMW SCL into target application are properly protected. The SCL will make the callback on whichever thread is currently executing the corresponding SCL code. Hence appropriate locks will be needed in the target application.

When the function *'tmwappl_checkForInput'* is called, any channel which has been opened on that application context will be checked. Normally, for implementations that require a separate channel per thread, event driven input would be used. If however, the polled input mechanism is used each channel must be opened on a separate application context to guarantee that only that channel will be processed by the call to *'tmwappl_checkForInput'*.

The default SCL configuration provides a single timer queue for all channels. When the target timer expires and the SCL timer callback function is called, SCL code for any channel can be processed. If instead, a separate timer queue for each channel is required in order to limit processing to a specific channel, TMWCNFG_MULTIPLE_TIMER_QS should be defined. You may also want to take a look at the simple multi-threaded example found in a subdirectory under <*installdir*> (e.g., <scl>Slave or <scl>Master).

Please note that the Source Code Library was not designed to use binary semaphores; therefore, directly using binary semaphores could lead to a deadlock situation.

You may wish to write a target function that increments/decrements a counter and only calls your binary semaphore lock on the initial lock and final unlock. Note that TMWDEFS_RESOURCE_LOCK defines a structure for the lock. You could define this structure to contain your binary semaphore and an internal lock counter.

3.2.15.1 Gateway Functionality

When implementing gateway functionality between callback functions, databases, sessions or sectors on individual channels, avoid calling from the context of Channel A into Channel B at the same time Channel B is calling into Channel A. Because of the lock around each channel this would be a likely deadlock scenario. This is especially likely to be an issue when running channels in individual threads such as when polledMode=false. You must decouple processing between channels to prevent acquiring the locks for these two channels in a different order. When attempting to lock more than one critical section, locks must always be acquired in the same order to prevent deadlock.

3.2.16 Asynchronous Database Updates

Database updates on master devices frequently takes a considerable amount of time. This is typically the case when updating a relational, or otherwise complex, database. If the amount of time required to update the master database precludes returning to process input data and/or respond to requests in the SCL in a timely fashion the database updates must be handled asynchronously to the SCL processing.

The Source Code Library ships configured for asynchronous database accesses. In this mode, the Source Code Library stores data on a database queue. To store data in the database, the application must call tmwdb_storeEntry(). Typically, this call would be on a separate thread.

To disable asynchronous database accesses, set TMWCNFG_SUPPORT_ASYNCH_DB to TMWDEFS_FALSE. In this configuration, the Source Code Library will call the <scl>data_xxx routines to store data as each message from the remote device is processed.

3.3 Master Overview

This section describes details specific to TMW Master SCLs.

3.3.1 Architecture

This section describes architectural details specific to a master SCL. In addition to the basic features and functionality available in all TMW SCLs, each Master SCL performs the following functions:

1) Generate and send requests to remote devices as directed by the target application through the associated *'<scl>brm'* routines.
2) Monitor for messages from slave devices and process as follows:
    a. The message is first checked to see if it is in response to an outstanding request. If so, it processes the request, calling any user callback functions and either closes the request or proceeds to the next step of the request.
    b. If the message was an 'unsolicited' message it is processed and the appropriate action is taken.

3.3.1.1 Database Interface

The master database interface is designed to support the storage of values returned from slave devices. In some cases, the data points will be known by the master, in other cases the master database should support the dynamic creation of points returned from the slave device.

The interface is contained in the <scl>data.h/c files. A separate database may be maintained per session for DNP or per sector for the IEC 60870-5 protocols. The SCL calls a database initialization function to allow the target application to initialize the database. The target application returns a database handle from the database initialization function. The SCL provides this handle with all future accesses to the database.

The SCL provides an individual data store function for each type of data point object. The input parameters to these functions allow include the database handle, the Information Object Address that identifies the data point, and the data values that need to be stored.

A database close function is called by the SCL to that the target application can perform whatever actions are required by the database when the session or sector is closed.

3.3.1.2 Command Interface

The master command interface allows target application to generate and send requests to remote devices. The interface is provided in the <scl>brm.h/c files. There are specific individual request functions provided for each command type. All of these functions require a pointer to an SCL specific request structure to be passed as an argument. Some of the functions require additional arguments depending on their specific purpose.

For each request, the target application can specify a callback parameter and function to be called upon completion of that command. The target application can also specify a timeout value for each command request.

Each request function returns a pointer to a transmit data structure. The pointer to this structure should be used for future reference to this outstanding request. For example, this

pointer would be used to cancel this request. It would also be passed to the user callback function upon completion of this request to identify the request.

An interface is also provided to register to receive unsolicited or asynchronous messages from a remote device. This allows the target application to receive messages on a per session basis in addition to the responses to commands that were sent.

3.3.2 Supporting Redundant Communication Paths

There are several mechanisms for supporting redundant communications paths to an outstation device. The following paragraph summarizes the most common redundancy schemes.

1) The master simultaneously collects data over each channel. The same data are reported over each channel, and the Master merges these duplicated event reports into a single database update stream.

   This method can be implemented by creating a separate session (and channel) for each redundant path. When data are reported, the SCL calls the database interface routine for the current session. The target's database interface routine (or some higher level process) is responsible for processing duplicate events and merging the data into a single stream of database updates.

2) The master collects data from a single one of the possible paths to the slave, and switches to another path if the original channel fails. In this case the Master monitors the channels and switches channels as appropriate.

   One way to implement this method is to switch channels when a command fails. For example, the mdnpbrm functions that initiate a request message can be configured to call a function when the command completes (pUserCallbackParam in the MDNPBRM_REQ_DESC structure)

   Upon completion, the SCL calls the specified callback function, passing a status indication in the pUserCallback (DNPCHNL_RESPONSE_INFO field of DNPCHNL_CALLBACK_FUNC.) The target callback function can, upon receipt of a failure indication (e.g., DNPCHNL_RESP_STATUS_FAILURE or DNPCHNL_RESP_STATUS_TIMEOUT), reconfigure the device for a different channel, and resubmit the request message. In most systems, it is also desirable to notify the end user that this switch has taken place.

   With this method, it is often desirable to maintain a concept of primary and secondary channels, as the secondary channel is likely to be a slower and/or more expensive connection. Thus, it may be desirable to implement logic to switch back to the primary channel when it becomes available again.

Note that with either method, it is important to periodically verify the status of the redundant channel. With method 1), user notification can be made whenever either channel fails. With method 2), it is usually desirable to perform periodic link checks on the backup channel. The DNP3 link status message (configured via the linkStatusPeriod parameter of the MDNPSESN_CONFIG structure) can be used for this purpose.

3.3.2.1 IEC 60870-5-104 Redundancy

The IEC 60870-5-104 International Standard Edition 2 specifies a mechanism to be used for redundancy. This was previously known as ‘Norwegian Redundancy’. The M104 and S104 Source Code libraries include code to implement redundancy as specified in the standard. Details can be found in the protocol specific sections of this manual

3.4 Slave Overview

This section describes details specific to TMW Slave SCLs.

3.4.1 Architecture

This section describes architectural details specific to a slave SCL. In addition to the basic features and functionality available in all TMW SCLs each slave SCL performs the following functions:

1) Wait for and process requests from the remote master devices.
2) Optionally scan for change events.
3) Transmit events, either unsolicited or when requested.

3.4.1.1 Database Interface

The slave database interface is designed to support the retrieval of information about data points and their values to be sent to the master device. This will most likely interface with the data points themselves implemented in hardware and not with a conventional database.

The interface is contained in the <scl>data.h/c files. A separate database may be maintained per session for DNP or per sector for the IEC 60870-5 protocols. The SCL calls a database initialization function to allow the target application to initialize the database. The target application returns a database handle from the database initialization function. The SCL provides this handle with all future accesses to the database.

Functions are provided to retrieve information about individual data point object types. For each object type, the SCL provides a function to retrieve a handle for a particular point. This handle is then passed to the other data functions to determine if a value has changed, what group a point belongs to, or to retrieve the value of the data itself.

A close function is also provided to perform whatever actions are required when the session or sector is closed.

3.4.1.2 Event Generation

The SCL can be configured to periodically scan for changes in data points and automatically generate events, or user provided code can call the appropriate xxx_addEvent function when it determines that a change has occurred. If the periodic scan option is chosen, the user can specify how often to scan for changes on a per session or per sector basis depending on the protocol.

The statistics callback function is called if the event buffer overflows. For DNP3, the TMWSESN_STAT_CALLBACK function is called with the eventType parameter set to TMWSESN_STAT_EVENT_OVERFLOW. For IEC 60870-5 protocols, the TMWSCTR_STAT_CALLBACK function is called with the eventType parameter set to TMWSCTR_STAT_EVENT_OVERFLOW.

Note that the DNP3 specification requires setting of the IIN2.3 (Buffer Overflow) bit when the event buffer overflows. The SCL will set this bit according to the specifications; no special processing is required in the target software.

The IEC 60870-5 protocols do not include a buffer overflow indication. However, in Section 7.2.2.3 Buffer Overflow of IEC 60870-5-101 Edition 2, the specification states that a controlled station (i.e., a Slave or Outstation) may assign a single point information object to indicate a buffer overflow status. If this method is used, a status of <1> indicates an overflow condition; a value of <0> indicates no overflow condition. The action to be taken by a controlling station (i.e., a Master) when this bit is set is implementation specific.

If use of an overflow bit is desired in an IEC 60870-5 implementation, the target hardware must set the appropriate bit when the statistics callback function is called.

3.4.2 Supporting Redundant Communication Paths

There are several mechanisms for introducing redundancy in an outstation device. The following paragraph summarizes the most common redundancy schemes.

DNP3, IEC 60870-5, and other modern SCADA protocols employ an event reporting paradigm that ensures that field events that occur between data polls are reported to the master. To provide this feature, event buffers are created in the slave, and these buffers are reported to the master. If multiple redundant communication paths are provided between the master and the slave, the method of reporting events must be coordinated between these paths. This can be done in any of the following methods:

1) The master simultaneously collects data over each channel. The same data are reported over each channel, and the Master merges these duplicated event reports into a single database update stream.

   This method can be implemented by creating a separate session for each redundant path. When an event occurs, the target code should call xxx_addEvent for each session. Each session will then manage its own event queue to ensure that the event is passed to the Master.

2) The master collects data from a single one of the possible paths to the slave, and switches to another path if the original channel fails. In this case, Master monitors the channels switches channels as appropriate.

   One way to implement this method is to write tmwtarg_receive() to accept data from both a primary and secondary communication port and record the last port on which data are received. The tmwtarg_transmit() function sends data on the last port from which data were received.

   With this scheme, the SCL is only aware of one session on one channel; however, the target hardware is actually managing the two redundant serial ports transparently.

3.4.2.1 IEC 60870-5-104 Redundancy

The IEC 60870-5-104 International Standard Edition 2 specifies a mechanism to be used for redundancy. This was previously known as ‘Norwegian Redundancy’. The M104 and S104 Source Code libraries include code to implement redundancy as specified in the standard. Details can be found in the protocol specific sections of this manual.

3.5 Peer Overview

This section describes details specific to TMW Peer or dual mode SCLs. A Peer SCL is an SCL that supports all of the features of the master and slave SCLs described above Architecture. The Peer library contains both master and slave specific files. The Peer libraries include project files that include both Master and Slave specific files.

A peer channel can have both master and slave sessions opened on it. Otherwise, operation is exactly the same as described for a Master or Slave SCL.

See also Section 5.1Supporting Multiple Protocols.

Chapter 4 Porting Details

This section describes the steps required to port a TMW SCL in detail. This section discusses the steps required to port any TMW SCL. For details specific to a master, slave, or peer implementation see the appropriate chapter below and/or the protocol specific section.

4.1 Unpacking and Installing the SCL

Depending on the delivery mechanism the TMW SCL is delivered as either an encrypted Windows Zip file or a self-extracting Windows executable. This section describes in detail how to install each of these formats.

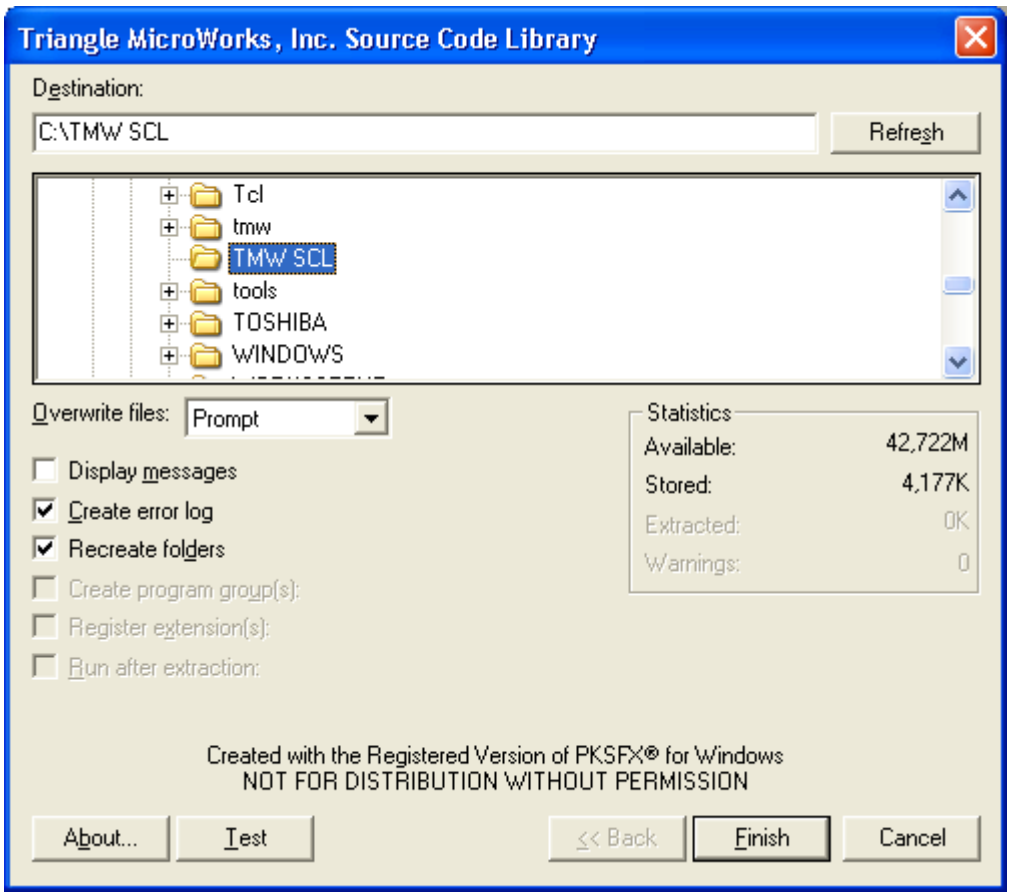
4.1.1 Encrypted Windows Zip File

If the source code library was delivered via email, it will be delivered as an encrypted Windows self-extracting Zip file named '<scl>vnmm.ex1' or an encrypted ZIP file named <scl>vnmm.zip, where *n* is the major version number and *mm* is the minor version number.

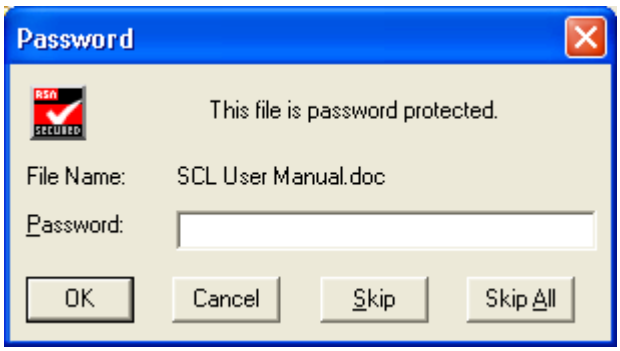
4.1.1.1 Encrypted Self-extracting Zip File

Because some email programs will automatically remove executable files from an email message, the self-extracting Zip file is renamed with a ".ex1" extension when it is delivered. To access the Source Code Library, rename it with the ".exe" extension and run it.

The extractor will present a screen similar to the following:



Use the Destination field or the graphical representation of the directory to select the location in which the SCL should be stored. Press the Finish button, and enter the password at the prompt:



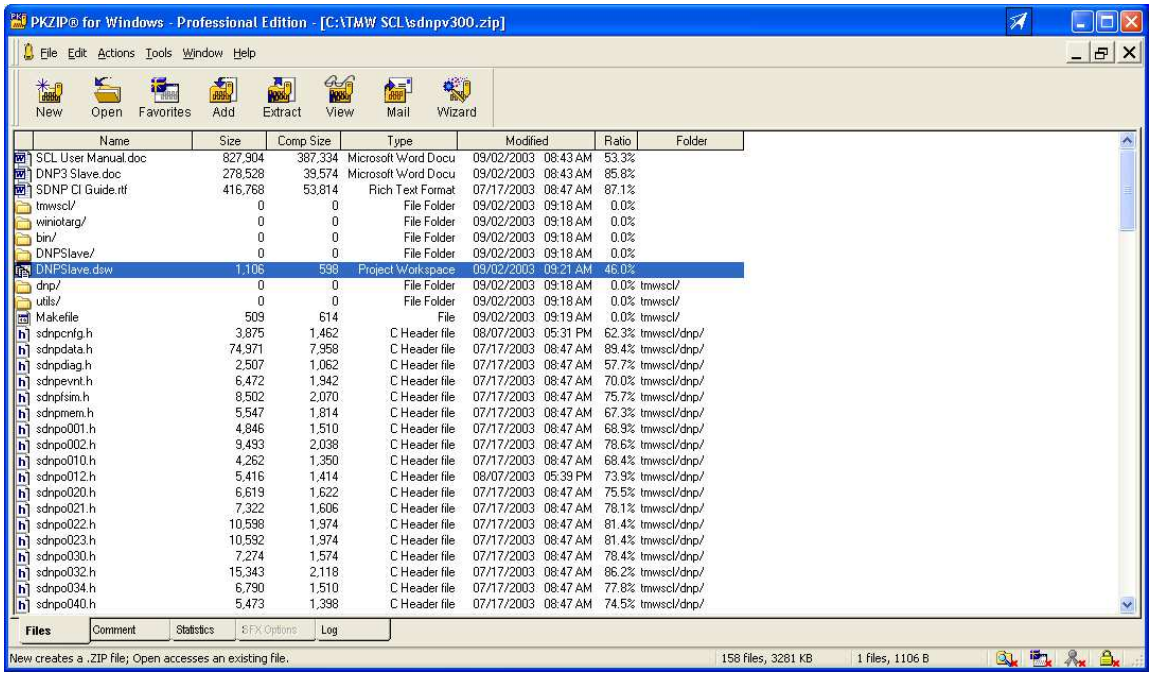
The SCL will then be extracted to the specified directory. Note that it is only necessary to enter the password once. If you are prompted to enter the password again for subsequent files, then you probably mistyped the password. Press Cancel, and repeat the above steps.

4.1.1.2 Encrypted Zip File

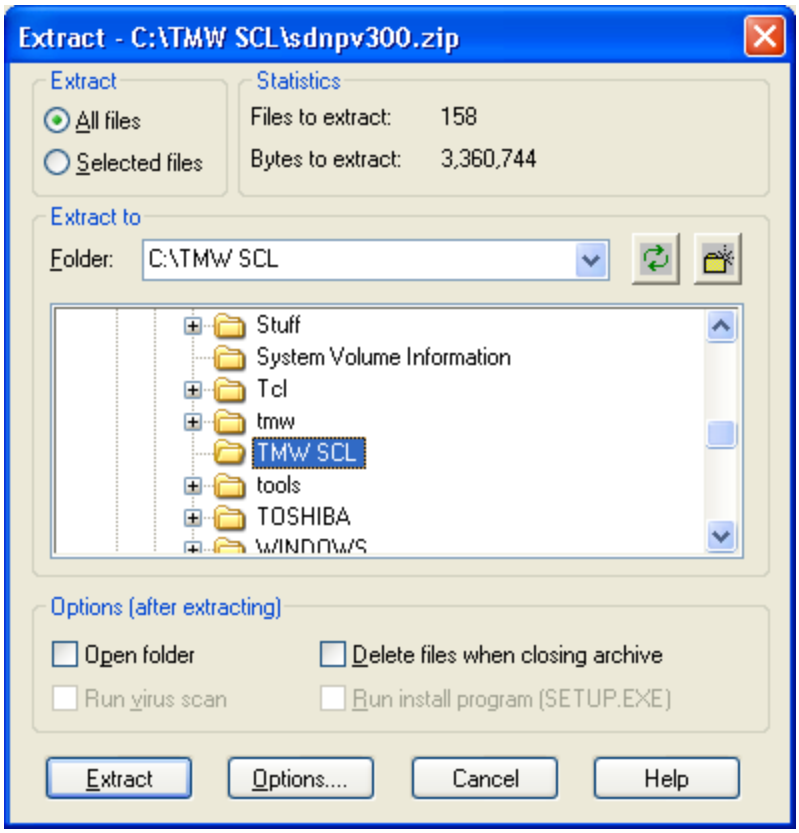
Some email programs will not deliver executable files, even if they do not have the .exe extension. In this case, Triangle MicroWorks, Inc. will, upon request, deliver the Source Code Library via email as an encrypted Zip file.

The encrypted Zip file uses strong encryption, so it can only be unzipped by a utility that supports it, such as PKZIP v6.0. A free unzip utility that supports strong encryption can be downloaded from http://www.pkzip.com/reader.

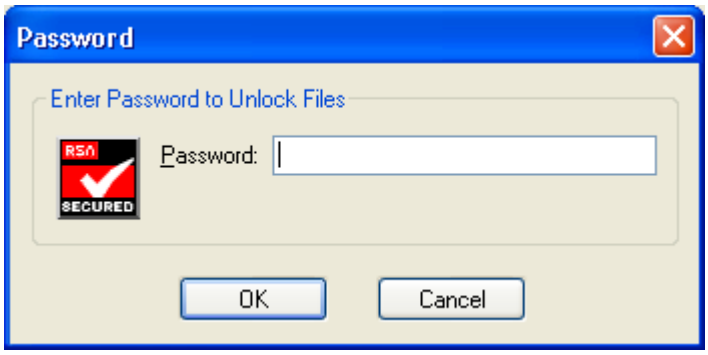
When you open the Zip file, a screen similar to the following is displayed:



Select Extract and navigate to the directory in which the files are to be extracted:



Then select Extract, and enter the password provided by Triangle MicroWorks, Inc.:



The SCL will then be extracted to the specified directory. Note that it is only necessary to enter the password once. If you are prompted to enter the password again for subsequent files, then you probably mistyped the password. Press Cancel, and repeat the above steps.

4.1.2 Self Extracting Windows Executable

If the SCL was delivered on a floppy disk, it will be delivered as a single self extracting Windows executable file named '<scl>vnmm.exe' where n is the major version number and mm is the minor version number. To install the SCL from this file simply execute the file, specify the desired target directory, and select 'OK'.

4.2 Building the SCL

Once the desired SCL(s) are installed, the first step in porting them to the target platform should be to build them as delivered using the target development environment. The TMW SCLs are all written using ANSI C, so this is typically a straightforward process. However, it is best to highlight any compiler issues before making modifications to the TMW delivered source code.

All of the TMW SCL files will be installed under the '*tmwscl'* directory in the installation directory specified during installation. The files will be installed in one or more subdirectories of the '*tmwscl'* directory. Each installation directory includes a Visual Studio 2017 project file as well as a GNU Makefile that can be used to compile the files in that directory.

4.2.1 Building on Windows

If your target platform is Windows and you are using Visual Studio 2017 or newer simply load the Visual Studio 2017 solution file found in the installation directory and perform a 'Build Solution'.

The Windows solution and project files as delivered can be used to build either a 32 bit or 64 bit executable by selecting Win32 or x64 on the Solution Platforms drop down box. The Solution Configurations drop down box will control whether the executable file will be put in the bin or bind directory depending by selecting Release or Debug.

4.2.2 Building on Linux

If your target platform is Linux, The GNU Makefiles provided will build the sample application and the source code library, simply type **make** to perform the build. Optionally, you can import the sample application and source code library into KDevelop Eclipse, or another IDE. Create a project that uses the existing Makefiles and then build the entire project.

4.2.3 Building for other operating systems.

If your target platform is not running either Windows or Linux, a target layer will have to be written for it. It will need to provide the same functionality that is provided in the same Windows and Linux target layers. The Linux sample target layer is more easily ported to other operating systems and TMW recommends using it as a basis. The GNU Makefiles provided can also be modified to support cross compilation and your target operating system.

4.2.4 Premake and lua files

The Makefile, solution and project files for the C examples were generated using premake 5.0.0-alpha9. https://premake.github.io/ Although not required, the SCL includes lua scripts used to regenereate these files so they can easily be updated if necessary, for example for an earlier version of Visual Studio.

To generate the Windows solution and project files for the DNP Slave example:

premake5 vs2017 --file=DNPSlave.lua

To generate the GNU Makefiles for the 104 Master example:

premake5 gmake --file=104Master.lua

4.2.5 Secure Authentication

If you have purchased the Secure Authentication feature for the SCL you should read the Advanced Topics sections on OpenSSL and Secure Authentication for more details about configuring and building that functionality.

4.3 Modify Basic Definitions

General definitions used throughout the TMW SCL can be found in *'tmwscl/utils/tmwdefs.h'*. These definitions include various optional compiler directives, basic data type definitions, simple macros used throughout the SCL, and SCL wide data types.

Additional data type definitions are defined in *'tmwscl/utils/tmwtypes.h'*.

Most of these definitions will not require changes based on the target platform. The only exceptions to this might be the optional compiler directives at the top of the file and the definitions of the basic data types. Each of these is documented in detail in the header files.

4.4 Modify SCL Configuration Files

This file *'tmwscl/utils/tmwcnfg.h'* is used to specify support for generic TMW SCL features and define how memory will be allocated (static or dynamic) as well as maximum sizes for various arrays used by the TMW SCL.

If TMW_USE_PRIVATE_DEFINES is defined, a private configuration file tmwprvt.h will be included. This include file may override some of the default library configuration and functionality to support internal Triangle MicroWorks applications as well as the .NET Component Protocol 21 day demo installed executables.

ANSI C (and beginning in version 3.29.0) .NET SCL customers should NOT define TMW_USE_PRIVATE_DEFINES or attempt to include tmwprvt.h. They should instead modify tmwcnfg.h and the other configuration files including *data.h to configure the library as desired. You should NOT simply use the default functionality configured as the SCLs are shipped. After careful consideration of your requirements, this file and other protocol specific files must be modified to enable only the functionality needed.

Disabling data types and functionality that are not required reduces memory consumption, the number of database functions that must be implemented, and the resulting testing effort. Removing unnecessary or unintended functionality is considered a good development practice as it limits the attack surface reducing the exposure to potential vulnerabilities. The capabilities and configuration of your application should then be documented in the CI Guide or Device Profile as required for each protocol.

4.5 Implementing the SCL Target Interface

The TMW SCL target interface, found in *'tmwscl/utils/tmwtarg.h/c'* can be broken into a number of categories. This section discusses each of these in detail. Sourtce Code implementations of the target layer for Windows and Linux Operating Systems are included with all SCLs. If you are implementing on these target environments the code for these are contained in the *'tmwscl/tmwtarg/WinIoTarg'* and *'tmwscl/tmwtarg/LinIoTtarg'* directories.

4.5.1 Compiler-specific definitions

The TMWTARG_UNUSED_PARAM macro is provided to eliminate "unused parameter" warnings that are generated by some compilers. The default definition works with most compilers.

The default definition may cause some compilers to generate unwanted side effects, such as creating enough space on the stack to store the parameter. In this case, the macro may be modified as needed to eliminate these side effects.

4.5.2 Dynamic Memory Allocation

If the target device supports dynamic memory allocation, the 'tmwtarg_alloc' and 'tmwtarg_free' routines should be modified to call the required target functions. Generally, this simply involves calling 'malloc' and 'free'. If the target device does not support dynamic memory, see Section 4.7 Memory Allocation.

4.5.3 Diagnostic Support

The Source Code Library supports a protocol analyzer log when diagnostic support is enabled. In fact, the Communication Protocol Analyzer uses the SCL's built-in protocol analyzer display. The sample applications (<scl>Slave.cpp and <scl>Master.cpp) demonstrate the capabilities of the protocol analyzer display. Note that the examples do not implement filtering, although that capability is supported by the SCL.

To support the protocol analyzer display, enable diagnostics support and modify the 'tmwtarg_putDiagString' routine to output a string to the desired 'display'. Depending on the display and output capabilities of the target device, the output string could be sent to an external display, auxiliary serial port, a text window, or a (circular) buffer in memory that can be displayed using a logic analyzer or it could be retrieved later.

4.5.4 Byte Ordering (big-endian vs little-endian)

All of the protocols currently supported by TMW SCLs require that the message byte order be least significant byte (LSB) first. If the target device uses an LSB order, then multi-byte data can be copied directly into the message buffer. Intel processors are typically LSB (little-endian), while Motorola and PowerPC are typically MSB (big-endian). The tmwtarg_getXX and tmwtarg_storeXX example routines in tmwtarg.c have been rewritten to allow them to run on both MSB and LSB processors. However, because of differences in the way 64 bit double precision floating point values are stored in memory, it may still be necessary to modify tmwtarg_get64 and tmwtarg_put64 used

only by DNP3, not IEC60870-5 or Modbus. No changes should be required to these functions if your system adheres to the IEEE 754 Standard for Floating Point. To verify these two functions, you should use the Communication Protocol Test Harness or another working DNP device and read DNP Analog Inputs using object group 30 variation 6 (double precision floating point). If the non-zero values on the outstation are properly received by the DNP master then these routines are working properly.

4.5.5 System Date and Time

The target must provide a means to read the current date and time.

4.5.6 Timers

The TMW SCL, in its default configuration, requires only a single event timer. This timer must be able to invoke a specified callback function after a specified number of milliseconds. If an event timer that provides this functionality is available on the target platform it can be used directly to provide this functionality by modifying 'tmwtarg_startTimer' and 'tmwtarg_cancelTimer'.

If the target platform does not support event timers, a polled timer implementation is provided in *'tmwscl/utils/tmwpltmr.h/c'*. The default implementation uses this polled timer. If the polled timer is used the main application will need to periodically call '*tmwpltmr_checkTimer'* to see if the enough time has elapsed for the requested timer to have expired. This will typically be done in the same loop that calls the *'tmwappl_checkForInput'* function.

The TMW SCL can also be configured to allow a separate timer queue per channel or thread. This configuration requires a separate event timer per channel (the default polled timer implementation cannot be used). Defining TMWCNFG_MULTI_TIMER_QS allows only expired timers and SCL code (including user callback and target functions) for the specified channel to be executed during the callback context. This can be desirable in some multithreaded implementations.

When TMWCNFG_MULTI_TIMER_QS is defined, the follow functions must be implemented in the target layer:

- twmtarg_initMultiTimer() -Initializes the timer for the channel.
- tmwtarg_setMultiTimer() – Starts a timer that will call the multitimer's callback function when the specified timeout expires.
- tmwtarg_deleteMultitimer – Deletes the channel's timer and frees any resources allocated by the tmwtarg_initMultiTimer function.

4.5.7 Physical Input/Output

The remainder of tmwtarg.h/c contains routines to access the target device's physical input/output channels. Open channels by calling the appropriate openChannel function (see Table 5). Input parameters to these functions provide data structures that hold configuration information for the physical, link, and transport (if required) layers, along with a void pointer to a user defined data structure. This void pointer is maintained, but

not used, by the TMW SCL and is passed to the tmwtarg_initChannel routine. This pointer reference is also passed to the target I/O routines and can be used to identify and/or configure the target channel. If the device only supports a single channel, this pointer can be TMWDEFS_NULL.

Table 5. openChannel Functions

Protocol	openChannel Function
dnp	dnpchnl_openChannel
IEC 60870-5-101	ft12chnl_openChannel
IEC 60870-5-102	ft12chnl_openChannel
IEC 60870-5-103	ft12chnl_openChannel
IEC 60870-5-104	i104chnl_openChannel

A callback function is also provided by the SCL to the target layer through a call to tmwtarg_openChannel. This callback function should be called by the target implementation if it receives a close notification from the operating system.

The channel initialization routine returns a pointer to the new context. The target application will use this context pointer to identify this channel when calling the remaining SCL I/O routines.

The SCL will call tmwtarg_receive to get received characters. This interface specifies the maximum number of characters the SCL is willing to retrieve on this call. For Modbus RTU this function should not return any bytes to the SCL until the entire frame has been received or the inter-character timeout has expired as specified by the protocol.

Note that some small Modbus slave devices with a limited number of TCP/IP connections can break the TCP/IP connection after a timeout to allow communication with multiple master devices. The target implementation may need to take this into account.

4.5.8 Version 3.22.000 Target Layer Changes

The Linux and Windows target libraries have been updated in this release to provide more uniform API's and improve portability. The directory structure was also changed to move the provided Windows and Linux target layers under *tmwscl/tmwtarg*. Target specific tmwtarg.c implementations are provided in those directories. The base *tmwscl/utils/tmwtarg.c* should be modified if you are not using the target layers provided.

Applications migrating to SCL 3.22 may need to make the following changes:

- Linux –
  - Update the application's Makefiles that include SCL headers to add tmwscl/tmwtarg/LinIoTarg to the default include path.
  - Replace calls to liniotarg_initConfig() with calls to tmwtargio_initConfig().

- Replace calls to Sleep(millisecs * 100) with calls to tmwtarg_sleep(millisecs);
- Replace invocations of the macro TMW_CreateThread() with TMW_ThreadCreate().
- The linTCP data structure defined in LINIO_CONFIG has been renamed targTCP so the initialization of this structure will have to be updated to use the updated name.
- The SCL has migrated to use of OS independent enum types for TCP_MODE, TCP_ROLE, and IO_TYPE. As a result, LINTCP_MODE enums should be renamed TMWTARGTCP_MODE, LINTCP_ROLE enums should be renamed TMWTARGTCP_MODE, and LINIO_TYPE enums should be renamed TMWTARGIO_TYPE.
- To ease the migration, in tmwscl/tmwtarg/LinIoTarg/tmwtargos.h LINIOTARG_SUPPORT_LEGACY_CONFIG can be defined. If defined, it maps the legacy definitions to their current implementation.

- Windows –
  - Update the application's project files that include SCL headers to add tmwscl/tmwtarg/WinIoTarg to the default include path.
  - Replace calls to WinIoTarg_initConfig() with calls to tmwtargio_initConfig().
  - Replace calls to tmwtargp_Sleep(millisecs) with calls to tmwtarg_sleep(millisecs);
  - Replace invocations of the macro TMW_CreateThread() with TMW_ThreadCreate().
  - The winTCP data structure defined in WINIO_CONFIG has been renamed targTCP so the initialization of this structure will have to be updated to use the updated name.
  - The SCL has migrated to use of OS independent enum types for TCP_MODE, TCP_ROLE, and IO_TYPE. As a result, WINTCP_MODE enums should be renamed TMWTARGTCP_MODE, WINTCP_ROLE enums should be renamed TMWTARGTCP_MODE, and WINIO_TYPE enums should be renamed TMWTARGIO_TYPE.
  - To ease the migration, in tmwscl/tmwtarg/WinIoTarg/tmwtargos.h WINIOTARG_SUPPORT_LEGACY_CONFIG can be defined. If defined, it maps the legacy definitions to their current implementation.

4.6 Managing Channels, Sessions, and Sectors

Channels, sessions, and sectors are managed using routines defined in <scl>chnl.h/c, <scl>sesn.h/c, and <scl>sctr.h/c. Depending on the requirements of the target device channels, sessions, and sectors can be initialized once at startup and never changed or dynamically opened and closed throughout the operation of the device.

Refer to the sample application for an example of the code required to initialize and open channels, sessions, and sectors.

The open session and sector commands allow for customization of processing. For example, the mdnpsesn_openSession command includes a pointer to the

MDNPSESN_CONFIG structure. The autoRequestMask field of this structure can be used to enable or disable automatic processing of requests (such as time synchronization) based on the Internal Indication (IIN) bits received from a Slave. By turning off the appropriate bit in the autoRequestMask and adding processing in the mdnpdata_processIIN() function, the application can customize how it handles these requests.

Similar configuration options are supported for the IEC 60870-5 protocols. For example, the meinaActionMask and onlineActionMask fields of the M101SCTR_CONFIG structure define actions that can be performed automatically by the Source Code Library when it receives an MEINA end of initialization message from a remote device and/or determines that a remote device has come on line. These fields are described in more detail in m101sctr.h.

4.7 Memory Allocation

TMW SCL memory allocation is controlled by the following files:

- tmwscl/utils/tmwcnfg.h
- tmwscl/utils/tmwmem.c/h
- tmwscl/<family>/<scl>mem.c/h

The SCL ships configured to use dynamic memory allocation with no predefined limits. To add limits to the number of structures that can be allocated, in tmwcnfg.h change the TMWCNFG_MAX_XXX macro definitions from TMWDEFS_NO_LIMIT to the desired maximum.

By default, the individual <scl>mem.h files are all dependent on the definitions in tmwcnfg.h. If a finer level of control is required, the individual definitions in the tmwscl/<family>/<scl>mem.h files can be changed.

To disable dynamic memory allocation in tmwcnfg.h set the TMWCNFG_USE_DYNAMIC_MEMORY macro to TMWDEFS_FALSE. Since it is not possible to have static compile time memory allocation and no limit on individual memory pools, it is also necessary to set the individual TMWCNFG_MAX_XXX macro definitions to the desired values. The maximum number of sessions, sectors and channels should be set to reflect the number of each that will be open at any one time. The number of messages and events supported per session for DNP or sector for IEC must also be set.

Other macros, such as the number of simulated databases, database points and strings in the database, should be set to the number you require.

Note that the above parameters set the total number of event buffers per type for a device. The maximum number of events in the queue on any session/sector is configured in the session/sector initConfig structure, which is passed to the SCL when the session/sector is opened.

4.7.1 Memory allocation only at startup

The SCL can be configured to make only one call to tmwtarg_alloc at startup time for each memory pool. The SCL would then manage allocations and deallocations from these memory pools internally. To enable this mode, set both

TMWCNFG_USE_DYNAMIC_MEMORY and TMWCNFG_ALLOC_ONLY_AT_STARTUP to TMWDEFS _TRUE. Embedded systems may find this a useful mechanism, to set aside a specified amount of memory in fixed size memory blocks at startup, avoiding memory fragmentation and the mixing of SCL memory with other system memory allocations. Since it is not possible to have the memory allocated at startup and no limit on individual memory pools, it is also necessary to set the individual TMWCNFG_MAX_XXX macro definitions to the desired values. The maximum number of sessions, sectors and channels should be set to reflect the number of each that will be open at any one time. The number of messages and events supported per session for DNP or sector for IEC must also be set.

4.7.2 Run time sizing of memory constraints

When TMWCNFG_USE_DYNAMIC_MEMORY is set to TMWDEFS_TRUE it is possible to change the maximum amount of memory at run time that is available to the SCL to allocate. Whether TMWCNFG_ALLOC_ONLY_AT_STARTUP is set to TMWDEFS_FALSE or TMWDEFS_TRUE a function may be called at startup to modify the compile time values.

Here are two examples of modifying the maximum quantity of buffers in some of the memory pools at run time. Note that separate functions exist for each protocol and whether master or slave functionality is supported.

```
/* Get the default quantities for the memory pools used by SDNP
 * SCL. Modify three of the quantities. Then initialize all of the
 * memory pools used by the SDNP SCL code.
 * NOTE: this must be done before calling
 *  tmwappl_initApplication()
 */
TMWMEM_CONFIG  tmwMemConfig;
DNPMEM_CONFIG  dnpMemConfig;
SDNPMEM_CONFIG sdnpMemConfig;

sdnpmem_initConfig(&sdnpMemConfig, &dnpMemConfig, &tmwMemConfig);
tmwMemConfig.numAppls = 1;
dnpMemConfig.numChannels  = 4;
sdnpMemConfig.numSessions = 8;

sdnpmem_initMemory(&sdnpMemConfig, &dnpMemConfig, &tmwMemConfig);

/* Change just the number of buffers in the memory pool used
 * for MSP events. Then initialize all of the memory pools used
 * by the S101 SCL code.
 * NOTE: this must be done before calling
 *  tmwappl_initApplication()
 */
TMWMEM_CONFIG   tmwMemConfig;
I870MEM_CONFIG  i870MemConfig;
I870MEM1_CONFIG i870Mem1Config;
S14MEM_CONFIG   s14MemConfig;
S101MEM_CONFIG  s101MemConfig;

s101mem_initConfig(&s101MemConfig, &s14MemConfig,
  &i870Mem1Config, &i870MemConfig, &tmwMemConfig);
s14MemConfig.numMSPEvents = 100;
```

```
s101mem_initMemory((&s101MemConfig, &s14MemConfig,
  &i870Mem1Config, &i870MemConfig, &tmwMemConfig);
```

4.8 Master Libraries

This section covers porting details specific to a TMW SCL Master.

4.8.1 Integrating the SCL with your Database

The master database is required to store data values that are received from the remote slave devices. The SCL provides a simple simulated database implementation contained in the <scl>sim.h/c files. The <scl>data.h/c files as shipped use this simulated database implementation. While this database implementation should be sufficient for the initial porting and testing activity, it must be replaced with a database implementation of your choice. To do so, the<scl>data.c files should be modified to use your database rather than the simulated database, and the <scl>sim.h/c files should be removed from your build process.

The SCL provides an <scl>data_init function. This function allows the target application to do whatever is necessary to prepare the database for access. This function returns a handle that is used for all other accesses to that database. The <scl>data_init function is called for each master DNP session or IEC sector that is opened.

The individual <scl>data_xxxStore functions use parameters closely aligned in data type and name with the relevant protocol standard. Any conversion to a form convenient to your database implementation should be performed in the <scl>data.c file. No changes to the interface should be made to the <scl>data.h files.

There is also a <scl>data_close function that is called when a session or sector is closed. It should perform whatever is specifically required for your database implementation.

The <scl>data.h files also contain <SCL>DATA_SUPPORT_XXX macros which allow you to tailor your implementation to the specific functionality that you wish to support. If certain data types are not required, the associated macro can be changed to TMWDEFS_FALSE and the associated <scl>data_xxxStore routines can be stubbed out. This technique can also be used in the porting process to provide incremental functionality.

4.8.2 Issuing Commands

Target application code generates command requests using the <scl>brm_xxx function calls found in <scl>brm.h/c. All of these request functions take a pointer to an SCL specific request structure. This request structure should be allocated by the target application and initialized by calling the <scl>brm_initReqDesc function. Any variations from the default values in the structure should then be completed. The required fields are filled in by the init function and the other fields are initialized appropriately. These fields allow the user to provide a callback parameter and function to be called when the command completes. An additional field allows for a timeout to be set by the caller.

In addition to the request structure, some request functions require command specific parameters. These are necessary to provide command specific information that is not required in general. These calls each return a pointer to a transmit data structure that will

be used to reference this outstanding request in future calls as well as in callback routines. When the command response is received, the user specified callback is invoked passing the original transmitted data, the received data, and the status of the request.

To register to receive notification of unsolicited messages on a particular session, the target application can call the <scl>sesn_setUnsolUserCallback function, providing a callback parameter and function to be called when a message is received. This process allows the user to be notified of intermediate or asynchronous messages from the slave.

The database storage functions in <scl>data.c/h are called regardless of whether a callback function is provided.

4.9 Slave Libraries

This section covers porting details specific to a TMW SCL slave.

4.9.1 Integrating the SCL with your Database

The slave database interface is designed to support the retrieval of information about data points and their values to be sent to the master device. This will most likely interface with the data points themselves implemented in hardware and not with a conventional database.

The SCL provides a simple simulated database implementation contained in the <scl>sim.h/c files. The <scl>data.h/c files as shipped use this simulated database implementation. While this database implementation should be sufficient for the initial porting and testing activity, it must be replaced with an implementation that provides access to the target device data points. To do so, the<scl>data.c files should be modified to access that data rather than the simulated database, and the <scl>sim.h/c files should be removed from your build process.

The SCL provides an <scl>data_init function. This function allows the target application to do whatever is necessary to prepare the database for access. This function returns a handle that is used for all other accesses to that database. The <scl>data_init function is called for each master DNP session or IEC sector that is opened.

The individual <scl>data_xxxGetPoint functions should return a handle that can then be used in the other data point access functions. Those access functions should be implemented to provide information about the individual data points. The <scl>data_xxxChanged function must be implemented if periodic scanning for changes to generate events is to be performed by the SCL. If the value has changed it should return TMWDEFS_TRUE as well as the new values themselves. Any conversion of data types to a form convenient to your hardware implementation should be performed in the <scl>data.c file. No changes to the interface should be made to the <scl>data.h files.

There is also a <scl>data_close function that is called when a session or sector is closed. It should perform whatever specifically is required for your hardware implementation.

The <scl>data.h files also contain <SCL>DATA_SUPPORT_XXX macros which allow you to tailor your implementation to the specific functionality that you wish to support.

The SCL is shipped configured to support most data types and functionality by default. If certain data types are not required, the associated macro should be changed to TMWDEFS_FALSE and the associated <scl>data_xxx routines can be stubbed out. This technique can also be used in the porting process to provide incremental functionality

Note that several objects are defined with values like TMWCNFG_SUPPORT_DOUBLE or TMWCNFG_SUPPORT_FLOAT. This is done so that if the application does not support the specified data type, support for objects requiring these data types will be automatically turned off.

To force these objects to not be supported, you must set <SCL>DATA_SUPPORT_XXX to TMWDEFS_FALSE. If you later re-enable support, Triangle MicroWorks, Inc. recommends that you set them back to their original value in order to keep the association to the required data type.

4.9.2 Supporting Change Events

The TMW SCL can scan for change events at a user specified period or the user can create events explicitly. Explicit event generation is generally preferred if the target device has a mechanism for signaling the software that a value has changed.

To enable periodic scanning for events by the SCL, simply configure the xxxScanPeriod to the desired (non-zero) value in the XXX_CONFIG structure when opening SDNP sessions or IEC sectors. The SCL will then call the xxx_Changed functions at the specified scan period. The TMW SCL is shipped with a default scan period of 0, which disables periodic scanning for events.

If event scanning is disabled, the target application should call <scl>xxx_addEvent defined in <scl>xxx.h/c when a change is detected. The parameters provided to each addEvent function are specific to each individual data point type.

The <scl>xxx_addEvent function parameters include a pointer to a timestamp. Calling <scl>util_getDateTime() to get the time stamp will allow the Source Code Library to set the time stamp properly, including extra fields such as the “invalid” field.

4.9.3 Detecting Master Going Offline

While the callback function (TMWSESN_STAT_CALLBACK pStatCallback) provided to sdnpsesn_openSession() will notify the target application of an offline status condition (TMWSESN_STAT_OFFLINE) based on some internal conditions, the SCL does not monitor for continued reception of requests from a master.

If this functionality is required it can be achieved in the slave target application by restarting a timer and setting the device online each time the callback function presents a status value of TMWSESN_STAT_ASDU_RECEIVED. If the timer expires the application may consider the session to be offline. This will allow the target application to determine if the master is continuing to communicate with it or if it has gone “offline”.

4.10 Peer Libraries

This section covers porting details specific to a TMW SCL peer library.

Porting of a Peer SCL is generally accomplished by porting either the master or slave portion as described in the sections above and then porting the 'opposite' portion by following the steps in the master or slave specific sections above. Note that the master and slave share a common target interface in a peer implementation so all of the master/slave common port is only done once.

Basic program flow for setting up a peer device might look like the following:

```
/*
 * Initialize the Channel Structures
 * Call function to initialize with default values,
 * then overwrite values as needed.
 */
<scl>chnl_initConfig(&prtConfig, &lnkConfig, &physConfig);

/*
 * Open the Channel which supports both master and slave sessions
 */
pChannel = <scl>chnl_openChannel(&linkConfig, &physConfig,
    &ioConfig);
if(pChannel == TMWDEFS_NULL)
    return;

/*
 * Initialize and open a master session on that channel
 * Call function to initialize with default
 * values, then overwrite values as needed.
 */
m<scl>sesn_initConfig(&sesnConfig);
pSession = m<scl>sesn_openSession(&sesnConfig);
if(pSession == TMWDEFS_NULL)
    return;

/*
 * Initialize and open a slave session on that channel
 * Call function to initialize with default
 * values, then overwrite values as needed.
 */
s<scl>sesn_initConfig(&sesnConfig);
pSession = m<scl>sesn_openSession(&sesnConfig);
if(pSession == TMWDEFS_NULL)
    return;


/*
 * For IEC 60870-5 protocols, initialize and open a master sector
 * Call function to initialize with default
 * values, then overwrite values as needed.
 */
m<scl>sctr_initConfig(&sctrConfig);
pSector = m<scl>sctr_openSector(&sctrConfig);
if(pSector == TMWDEFS_NULL)
```

```
    return;
/*
 * For IEC 60870-5 protocols, initialize and open a slave sector
 * Call function to initialize with default
 * values, then overwrite values as needed.
 */
s<scl>sctr_initConfig(&sctrConfig);
pSector = s<scl>sctr_openSector(&sctrConfig);
if(pSector == TMWDEFS_NULL)
    return;
```

4.11 Testing your Implementation

For details on how to best test the port see Chapter 7.

Chapter 5 Advanced Topics

This section discusses how the TMW SCL architecture can be leveraged to provide a wide range of custom solutions.

5.1 Supporting Multiple Protocols

Any of the TMW SCLs can be combined to provide support for multiple protocols in a single device. Since the different SCLs share many files it is imperative that they be the exact same version. Hence the first step in supporting multiple protocols in the same device is to acquire the same version of each of the required SCLs.

After confirming that all required libraries are the exact same version, extract the first SCL into the target directory. Extract the next SCL into the same directory, choosing either to overwrite or skip redundant files (these files will be identical). Repeat this last step for each desired protocol.

Now generate a makefile, or solution file, as required by your development environment that will compile all of the source files in the tmwscl/xxx directories created by the installation. You may want to merge the xxx.lua files shipped with the SCLs and download and run Premake to generate the Makefiles or solution files. For example, to merge SDNP and S104 SCLs into a single build, copy the missing i870 specific lines from 104Slave.lua to DNPSlave.lua and run Premake on DNPSlave.lua as directed at the top of that file.

A single target implementation in *'tmwscl/utils/tmwtarg.h/c'* will be shared by all the SCLs. A database interface must be provided for each SCL independently in *'tmwscl/<family>/<scl>data.h/c'*.

The remainder of the integration should be performed exactly as if you were integrating to the two SCLs independently. Specifically, open channels, sessions, and sectors exactly as you would if you were integrating each SCL independent of the others.

Note: You can create a Peer implementation by combining a Master and Slave of the same protocol. However, you will need to rebuild the project files to include both libraries. The Peer implementation ships with project files already configured for both Master and Slave libraries.

5.2 Distributed Architectures

In some situations, the physical, link, and possibly transport layer processing is performed on one or more dedicated front-end processors or coprocessors. The application layer runs on a single main processor. This architecture is depicted in the figure below with three front-end processors.

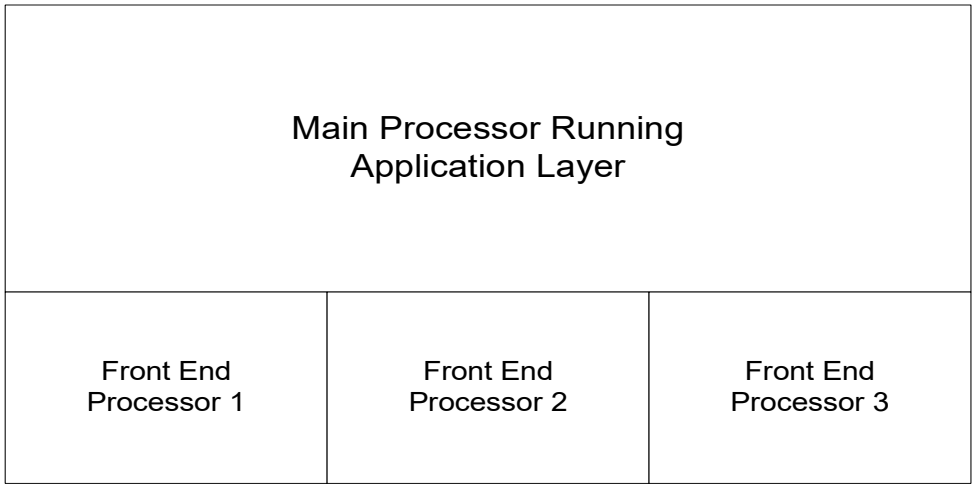


Figure 6: Example of a Distributed Architecture

The TMW SCL does not directly support this architecture, but it is easily supported by adding a simple pseudo transport layer that interfaces with the application layer. Data from the application layer is forwarded to the appropriate front end processor using the appropriate inter process communication. It is then passed to the real transport, link, and physical layers running on the front-end processor. Data received from the various input channels is forwarded through the physical, link, and transport layers as it would be in a typical implementation. It is then forwarded to the pseudo transport layer and passed to the application layer for processing.

The ability to implement the above solution is a result of the fact that each of the layers in a TMW SCL is independent of the layer above and/or below it in the protocol stack. This feature can be used to implement a wide variety of custom solutions.

5.3 60870-5-104 Redundancy

Another example of an extension to the standard TMW SCL architecture is an extension to the IEC 60870-5-104 protocol, known as ‘Norwegian Redundancy’. This has also been included in IEC 60870-5-104 Edition 2. This extension supports multiple redundant TCP/IP connections contained in a single IEC 60870-5-104 redundancy group. All user communications are performed on the current ‘active’ channel but each channel is monitored to make sure it is still operational. If the ‘active’ channel goes down communications are automatically switched to the next operational channel. The system then attempts to reestablish the channel that failed in addition to monitoring the other channels. The software architecture is depicted in the figure below.

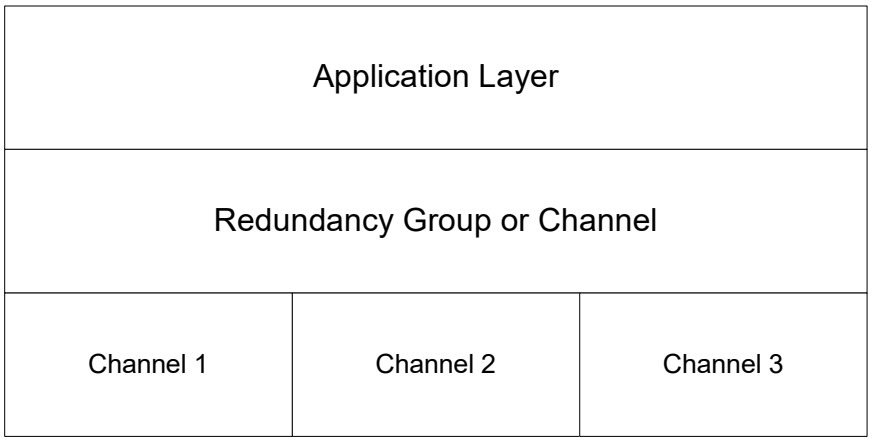


Figure 7: 60870-5-104 Redundancy Software Architecture

In this situation the physical and link layers still execute on the same processor as the application layer, but there are multiple physical/link layers or channels. The solution is similar to the one above in that a redundancy group or channel is implemented that manages multiple channels. When a request is received from the application layer it is dispatched to the current 'active' link layer. When information is received on the active link layer it is passed through the redundancy group and forwarded to the application layer.

Basic program flow for setting up a redundancy group and two redundant connections on a master device might look like the following:

```
TMWCHNL *pRdcyGroup;
TMWCHNL *pRdntChnl1;
TMWCHNL *pRdntChnl2;

/*
 * Call function to initialize the redundancy config structure, then
 * overwrite values as needed.
 */
LNK4RDCY_CONFIG rdcyConfig;
lnk4rdcy_initConfig(&rdcyConfig);

/*
 * On master set isControlling to true to handle determining which
 * channel is active and sending STARTDT.
 * On slave set isControlling to false.
 */
rdcyConfig.isControlling = TMWDEFS_TRUE;

/*
 * Open the Redundancy Group channel that will contain the redundant
 * connections (channels).
 */
pRdcyGroup = lnk4rdcy_initRdcyGroup(&rdcyConfig);
if(pRdcyGroup == TMWDEFS_NULL)
    return;
```

```
/*
 * Call function to initialize configuration structures, then
 * overwrite values as needed. On master set isControlling to true
 * to handle sending STARTDT.
 */
TMWPHYS_CONFIG physConfig;
TMWTARG_CONFIG targConfig;
I870LNK4_CONFIG linkConfig;
I870CHNL_CONFIG chnlConfig;
tmwtarg_initConfig(&targConfig);
i104chnl_initConfig(&chnlConfig, &linkConfig, &physConfig);

linkConfig.isControlling = TMWDEFS_TRUE;

/*
 * Open two redundant connections or channels as part of the
 * redundancy group.
 */
pRdntChnl1 = lnk4rdcy_openRedundantChannel(pApplContext, pRdcyGroup,
    &chnlConfig, &linkConfig, &physConfig, pIOConfig, &targConfig);
if(pRdntChnl1 == TMWDEFS_NULL)
    return;

pRdntChnl2 = lnk4rdcy_openRedundantChannel(pApplContext, pRdcyGroup,
    &chnlConfig, &linkConfig, &physConfig, pIOConfig, &targConfig);
if(pRdntChnl2 == TMWDEFS_NULL)
    return;

/*
 * Initialize and open a session on the channel
 * Call function to initialize with default
 * values, then overwrite values as needed.
 * Open a session on the redundancy group channel.
 */
m104sesn_initConfig(&sesnConfig);

pSession = m104sesn_openSession(pRdcyGroup, &sesnConfig);
if(pSession == TMWDEFS_NULL)
    return;

/*
 * Call function to initialize config structure, then overwrite
 * values as needed.
 * Open a sector on the session.
 */
m104sctr_initConfig(&sctrConfig);

pSector = m104sctr_openSector(pSession, &sctrConfig, pUserHandle);
if(pSector == TMWDEFS_NULL)
    return;
```

5.4 DNP3 IP Networking

This section describes the functionality that must be provided by the target layer to support DNP3 Specification IP Networking Version 2.2. If your target environment is on

Windows or Linux, source code is provided in the directories winiotarg and liniotarg that implements this.

The following is intended as a high level example to illustrate the interface with the SCL and the expected behavior of the target layer. It is still recommended to read the DNP3 Specification IP Networking Spec and to make sure that your target implementation performs all of the desired functionality as specified by that document.

The details of your design will be dependent on your target OS and hardware.

NOTE: There are other interface functions specified in tmwtarg.h that are required for diagnostics etc.

```
tmwtarg_initConfig()
   Set all fields of structure TMWTARG_CONFIG to default values


tmwtarg_initChannel()
   Save configuration data from TMWTARG_CONFIG

   Save configuration data from target specific config structure

   Set up pChannelCallback function to be used for open/close
   (connect/disconnect) indications


tmwtarg_deleteChannel()
   Deallocate any resources for this channel


tmwtarg_openChannel()
   If UDP support required
     Open local UDP port for sending/receiving datagrams on.
     Start a UDP reader thread to get received bytes from UDP socket
       (see details for example UDPReaderThread below)

   If server or dual end point system
     Start a listener thread to listen for incoming connect request
       (see details for example listenThread below)

   else if client
     Send active connect request to server
     Optionally start a connectorThread to periodically attempt
       active
       connect waiting without delaying the SCL.
       (see details for example connectThread below)


   /* The SCL will not try to transmit data until either this function
    * returns true or the pChannelCallback function is called with a
    * value of true indicating a channel is "open" to the remote device.
    * For a UDP only device this function should return true since no TCP
    *  connection is required.
    * For a dualEndPoint device this function should return true so that
    *  the SCL will attempt to send data. At that time this target layer
    *  should attempt to make the TCP connection.
```

```
    */
   if UDP only
     return true

   else if dualEndPoint system
     return true

   else if TCP connection
     return true

   else no TCP connection
     return false


tmwtarg_closeChannel()
   if listening
     stop listening

   if trying to connect
     stop trying to connect

   close any open connections


tmwtarg_receive()
   if UDP data has been received
     return up to max number of characters specified for this read

   if TCP data has been received
     Return up to max number of characters specified for this read

   if failure because TCP connection closed
     call pChannelCallback function indicating "closed"

   return 0


tmwtarg_getTransmitReady()
   If no connection AND DualEndPoint system
     Send active connect request to server
     Start a connectorThread to process connection setup
       (see details for example connectorThread below)

   If not ready to transmit
     return non zero value indicating how long SCL should wait
      before retrying

   Else if ready
     return 0


tmwtarg_transmit()
   Send specified bytes over TCP connection

   if failure because no connection
     call pChannelCallback function indicating "closed"
```

```
tmwtarg_transmitUDP()
   Send specified bytes in a UDP Datagram to remote port number
   indicated


The following are examples of internal target functionality called
from the above target interface functions

  UDPReaderThread()
    NOTE: When reading UDP bytes from a socket it may be necessary
    to read the bytes from an entire datagram, copying the bytes to
    a target layer buffer so that they can be given to the SCL as it
    asks for them.

    If activity on UDP socket
      recvfrom()
      If data received
        If configured to validate
           validate datagram source address
        Copy received bytes into target layer buffer to be read by
           SCL


  connectThread()
    Periodically call WinIoTarg_openChannel() (or internal
      equivalent) to connect to server
    If connection is successful
      Call pChannelCallback function indicating "open"
      exit thread


  listenThread()
    Open listen socket

    Bind port number 20000 to socket

    Wait for incoming connection request

      If configured
        Validate client IP address

      If already a TCP connection
        if this is dualEndPoint slave AND the existing connection
           was originated by this slave
          Disconnect existing connection
          Accept new connection

        Else
          Reject new connection

      Else if not already a TCP connection
        Accept new connection
        Call pChannelCallback function indicating "open"

      Continue listening for more connect indications
```

To fully implement the Networking Specification the following configuration or its equivalent may be necessary.

```
  // On client - the IP address to set up TCP connection to
  // On server - the IP address to accept TCP connection from
  //   May be *.*.*.* indicating accept connection from any client
  _TCHAR *ipAddress;


  // On client - the port to connect to
  // On server - the port to listen on
  // On Dual End Point Device - the port to listen on
  TMWTYPES_USHORT ipPort;

  // Indicate CLIENT, SERVER, DUAL END POINT, or UDP only
  // (DUAL END POINT provides both CLIENT and SERVER functionality
  //  but with only one connection at a time)
  WINTCP_MODE mode;


  // Indicate master or outstation (slave) role in dnp networking
  // as specified by DNP3 Specification IP Networking
  WINTCP_ROLE role;

  // If Dual End Point is supported a listen will be done on the
  // above ipPort and a connection request will be sent to this port
  // number when needed.
  // This should match ipPort on remote device.
  // Normal state is listen, connection will be made when there
  // is data to send.
  TMWTYPES_USHORT dualEndPointIpPort;

  // Local port for sending and rcving UDP datagrams on.
  // If this is set to WINTCP_UDP_PORT_NONE, UDP will not be
  // enabled.
  // For DNP networking UDP should be supported.
  // It is not needed for any of the current IEC or modbus
  // protocols.
  // On Master - If this is set to WINTCP_UDP_PORT_ANY, an
  //             unspecified available port will be used.
  // On Slave  - This should be chosen to match the UDP port that
  //             the master uses to send Datagram messages to.
  //             This must not be WINTCP_UDP_PORT_ANY or
  //             WINTCP_UDP_PORT_SRC.
  TMWTYPES_USHORT localUDPPort;

  // On Master - if TCP and UDP is configured this specifies the
  //              destination UDP/IP port to send broadcast requests
  //              in UDP datagrams to.
  //             if UDP ONLY is configured this specifies the
  //              destination UDP/IP port to send all requests in
  //              UDP datagrams to.
  //             This must match the "localUDPPort" on the slave.
  // On Slave  - if TCP and UDP this is not used.
  //             if UDP ONLY is configured this specifies the
```

```
//              destination UDP/IP port to send responses to.
//             Can be WINTCP_UDP_PORT_SRC indicating use the src
//              port from a UDP request received from master.
TMWTYPES_USHORT destUDPPort;

// On master - Not used.
// On Slave  - if TCP and UDP not used.
//             if UDP ONLY is configured this specifies the
//              destination UDP/IP port to send the initial
//              Unsolicited Null response to. After receiving a
//              UDP request from master, destUDPPort (which)may
//              indicate use src port) will be used for all
//              responses.
//              This must not be WINTCP_UDP_PORT_NONE,
//              WINTCP_UDP_PORT_ANY, or WINTCP_UDP_PORT_SRC
//              for a slave that supports UDP.
TMWTYPES_USHORT initUnsolUDPPort;

// Whether or not to validate source address of received UDP
// datagram.
TMWTYPES_BOOL   validateUDPAddress;
```

5.5 Adding support for Custom ASDUs for IEC60870-5 Protocols

The TMW SCLs provide support for the ASDUs defined in the relevant standards. If support for additional ASDU Type Identifiers is required, an SCL customer may integrate that functionality with the SCL in such a way that new releases of the SCL will not overwrite that functionality.

Support may be added for building non standard request ASDUs in a master 101/103/104 SCL to facilitate sending that request, provide error and timeout handling, and processing response messages. When a master session is opened, a target application callback function may be specified which will be called when a response containing an unknown Type Id is received.

When a slave session is opened two target application callback functions may be specified to allow processing of non standard requests and generating the responses to those requests.

The following examples will demonstrate how to add support for an additional ASDU Type Id to a 103 master and 101 slave implementation. The other IEC60870-5 master and slave implementations would be implemented in the same way.

103 Master example:

```
/* Example code to build a non standard 103 master request */
/* Code is similar to code found in m103brm.c              */
…
I870CHNL_TX_DATA *pTxData;
TMWSESN *pSession = pSector->pSession;

pTxData = (I870CHNL_TX_DATA *)i870chnl_newTxData(
  pSession->pChannel, pSession, pSector, sizeRequiredForRequest);

i870util_buildMessageHeader(pSession,
```

```
  (TMWSESN_TX_DATA *)pTxData, typeId, cot,
  0, ((M103SCTR *)pSector)->i870.asduAddress);

/* Set VSQ */
pTxData->tmw.pMsgBuf[1] = 0x81;

i870util_writeInfoObjectId((TMWSESN_TX_DATA *)pTxData,
  I3DEF_FUNC_GLOBAL, informationNumber);

/* Store one 16bit field in request message */
tmwtarg_store16(&var1,
  &pTxData->tmw.pMsgBuf[pTxData->tmw.msgLength]);

pTxData->tmw.msgLength += 2;

/* Store single byte field in requrest message */
pTxData->tmw.pMsgBuf[pTxData->tmw.msgLength++] = var2;

/* Send request */
if(!i870chnl_sendMessage((TMWSESN_TX_DATA *)pTxData))
{
  /* Failure */
}

…


/* function: _userResponseCallback
 * purpose: Example user registered callback for ASDU Type xxx
 * arguments:
 *  pResponseParam - user provided callback parameter
 *  pSector - pointer to sector structure returned by
 *   s103sctr_openSector()
 *  pMsgHeader - pointer to message structure containing response
 * returns:
 *  TMWDEFS_TRUE if this response is processed
 *  TMWDEFS_FALSE otherwise
 */
static TMWTYPES_BOOL _userResponseCallback(
  void *pResponseParam,
  TMWSCTR *pSector,
  I870UTIL_MESSAGE *pMsg)
{

  if(pMsg->typeId == xxx)
  {
    TMWTYPES_UCHAR  functionType;
    TMWTYPES_UCHAR  informationNumber;
    TMWTYPES_UCHAR  var2;
    TMWTYPES_USHORT var1;

    functionType = pMsg->pRxData->pMsgBuf[pMsg->offset++];

    informationNumber = pMsg->pRxData->pMsgBuf[pMsg->offset++];

    tmwtarg_get16(&pMsg->pRxData->pMsgBuf[pMsg->offset], &var1);
    pMsg->offset +=2;
```

```
    var2 = pMsg->pRxData->pMsgBuf[pMsg->offset++];

    /* Other processing here */
    …

    return(TMWDEFS_TRUE);
  }
  return(TMWDEFS_FALSE);
}

…
/* Example code to open session, registering callback function */
M103SESN_CONFIG configData;
m103sesn_initConfig(&configData);

configData.pProcessResponseFunc  = _userResponseCallback;
configData.pProcessResponseParam = TMWDEFS_NULL;

pSession = m103sesn_openSession(pChannel, &configData);
…
```

101 Slave example:

```
/* function: _userProcessRequestCallback
 * purpose: Example user registered callback for ASDU Type xxx
 * arguments:
 *  pProcessRequestParam - user provided callback parameter
 *  pSector - pointer to sector structure returned by
 *   s101sctr_openSector()
 *  pMsg - pointer to message structure containing request
 * returns:
 *  TMWDEFS_TRUE if this request was processed
 *  TMWDEFS_FALSE otherwise
 */
static TMWTYPES_BOOL _userProcessRequestCallback(
  void *pProcessRequestParam,
  TMWSCTR *pSector,
  I870UTIL_MESSAGE *pMsg)
{
  TMWTYPES_ULONG  ioa;
  TMWTYPES_USHORT var1;
  TMWTYPES_ULONG  var2;
  TMWTYPES_UCHAR  var3;

  if(pMsg->typeId == xxxx)
  {
    /* Parse Information Object Address */
    i870util_readInfoObjAddr(pSector->pSession, pMsg, &ioa);

    /* Read 16 bit variable from request message */
    tmwtarg_get16(&pMsg->pRxData->pMsgBuf[pMsg->offset], &var1);
    pMsg->offset += 2;

    /* Read 24 bit variable from request message */
    var2 = 0;
    tmwtarg_get24(&pMsg->pRxData->pMsgBuf[pMsg->offset], &var2);
```

```
      pMsg->offset += 3;

      /* Read 1 byte variable */
      var3 = pMsg->pRxData->pMsgBuf[pMsg->offset++];

      /* Other processing here */
      …

      return(TMWDEFS_TRUE);
    }
  return(TMWDEFS_FALSE);
}

/* function: _userBuildResponseCallback
 * purpose: Example user registered callback for ASDU Type Id not
 *  supported by SCL. This function should build a response if one
 *  is needed.
 * arguments:
 *  pBuildResponseParam - user provided callback parameter
 *  pSector - pointer to sector structure returned by
 *    s101sctr_openSector()
 *  buildResponse -
 *   if TMWDEFS_TRUE this function should build the response
 *   if there is one
 *   if TMWDEFS_FALSE this function should just report whether
 *    it needs to build and send a response message
 * returns:
 *  TMWDEFS_TRUE if this sent or needs to send a response message
 *  TMWDEFS_FALSE if no response is needed
 */
static TMWTYPES_BOOL _userBuildResponseCallback(
  void *pBuildResponseParam,
  TMWSCTR *pSector,
  TMWTYPES_BOOL buildResponse)
{
  /* If a user provided response needs to be sent,
   * probably as a result of the request processed in
   * userProcessRequestCallback().
  if(responseToSend)
  {
    if(!buildResponse)
    {
      return(TMWDEFS_TRUE);
    }

    /* build response */
    if((pTxData = i870chnl_newTxData(pSector->pSession->pChannel,
      pSector->pSession, pSector, pS14Session->maxASDUSize)) ==
      TMWDEFS_NULL)
    {
      return(TMWDEFS_FALSE);
    }

    /* Build response */
    i870util_buildMessageHeader(pSector->pSession,
      pTxData, typeId, cot,
      0, p14Sector->i870.asduAddress);
```

```
    /* Store IOA */
    i870util_storeInfoObjAddr(pSector->pSession, pTxData, ioa);

    /* 16 bit variable */
    tmwtarg_store16(&var1,
      &pTxData->pMsgBuf[pTxData->msgLength]);
    pTxData->msgLength += 2;

    /* 24 bit variable */
    tmwtarg_store24(&var2, &pTxData->pMsgBuf[pTxData->msgLength]);
    pTxData->msgLength += 3;

    /* 1 byte variable */
    pTxData->pMsgBuf[pTxData->msgLength++] = var3;

    /* Send the response */
    i870chnl_sendMessage(pTxData);

    return(TMWDEFS_TRUE);
  }
  return(TMWDEFS_FALSE);
}

…
/* Example code to open session, registering callback functions */
S101SESN_CONFIG configData;
s101sesn_initConfig(&configData);

configData.pBuildResponseCallback = _userBuildResponseCallback;
configData.pBuildResponseParam = TMWDEFS_NULL;
configData.pProcessRequestCallback = _userProcessRequestCallback;
configData.pProcessRequestParam = TMWDEFS_NULL;
pSession = s101sesn_openSession(pChannel, &configData);
…
```

5.6 OpenSSL Integration

The DNP3, IEC 60870-5-101, and IEC 60870-5-104 libraries have the option to support Secure Authentication as specified in IEC 62351. Secure Authentication support requires cryptography. A common cryptography interface, tmwcrypto.c/h has been added to the library and can be used with all the TMW libraries.

The file tmwcrypto.c is shipped as an example implementation that uses the readily available Open Source toolkit, OpenSSL, to provide that functionality. See http://openssl.org/ for more information about that project. To use this example implementation, the file tmwcnfg.h must be configured to do so by setting the following:
#define TMWCNFG_USE_OPENSSL TMWDEFS_TRUE

OpenSSL can also be used to provide TLS over TCP connections. TLS functionality is not accessed through the tmwcrypto interface. The interface to use TLS is implemented

in the provided target layer implementations WinIoTarg or LinIoTarg and can be implemented similarly if you are providing your own target layer. Configuration specific to the chosen target layer may be passed across the tmwtarg interface to enable or disable TLS and provide configuration settings. For more information on TLS see the next section on TLS.

The SCL can be configured to use OpenSSL Version 1.1.x or Version 1.0.2. However, this older OpenSSL version is no longer being updated by the OpenSSL community and TMW will deprecate this option in a future release. To configure the SCL to use the obsolete 1.0.x version of OpenSSL, set the following:
#define TMWCNFG_USE_OPENSSL_1_0_2 TMWDEFS_TRUE

Alternatively, you can replace this example implementation with your own cryptography code if OpenSSL is not a viable option. You will then need to modify tmwcrypto.c to call into your cryptography code where it indicates “Put target code here”.

NOTE: If you use the common cryptography interface, but do not use OpenSSL the library will build for testing purposes, but it does NOT provide any cryptography support and cannot be used for Secure Authentication implementations.

By default, the following defines are set to TMWDEFS_TRUE.
#define TMWCNFG_SUPPORT_SIMULATED_CRYPTO TMWDEFS_TRUE
#define TMWCNFG_SUPPORT_SIMULATED_CRYPTO_DB TMWDEFS_TRUE

If you replace OpenSSL with your own cryptography code
TMWCNFG_SUPPORT_SIMULATED_CRYPTO Must be set to TMWDEFS_FALSE.

Even if you use OpenSSL it is required that you implement your own cryptography database code. The simulated code shipped will only store a limited amount of Authentication User Key data for a small number of sessions or sectors as well as only a single set of DNP3 Authority and Outstation Asymmetric keys for all associations and should not be used for a real device.

The function tmwcrypto_configSimKey() is provided for the sample code to configure the keys in the simulated crypto database. tmwcrypto_setKeyData() should only be called by the internal DNP3 library code itself and should not be used to configure the simulated database.

TMWCNFG_SUPPORT_SIMULATED_CRYPTO_DB should be set to TMWDEFS_FALSE when you have implemented your database code. Search for TMWCNFG_SUPPORT_SIMULATED_CRYPTO_DB in tmwcrypto.c to see what functions must be implemented.

If the optional MAC algorithm AES-GMAC is required set the following in tmwcnfg.h:
#define TMWCNFG_SUPPORT_CRYPTO_AESGMAC TMWDEFS_TRUE

To build the TMW Protocol Libraries with OpenSSL, whether to support Secure Authentication and/or TLS, OpenSSL must be installed on your host build machine. TMW recommends using the most up to date approved version of OpenSSL to ensure that you have the latest updates and security patches. You should consult the OpenSSL web site for the most up to date installation procedures.

The following procedures were used by Triangle MicroWorks to install OpenSSL in our test environments for Windows and Linux. you may need to adjust this for your environment.

Install perl and NASM if you do not already have them on your PC. Both perl and NASM must be in your PATH.

Windows:

OpenSSL 1.1.x

cd yourpath\openssl (both here and below replace "yourpath" with the correct path.

The Source Code Library build files assume that OpenSSL will be installed in <*installdir*>/thirdPartyCode/openssl.)

32 bit support:

Open a Visual Studio 20xx x86 Native Tools Command Prompt and enter the following commands:

```
cd yourpath\openssl
perl Configure VC-WIN32
nmake
```

Copy the following files to the bin directory:

- libcrypto.lib
- libcrypto-1_1.dll
- libssl.lib
- libssl-1_1.dll
- openssl.exe

Create an inc32 directory under the openssl directory and copy the files from yourpath\openssl \include\openssl to yourpath\openssl\inc32\openssl.

To build a debug version, replace the perl line above with:
perl Configure debug-VC-WIN32 and copy the same files to the bind directory.

64 bit support:

Open a Visual Studio 20xx x64 Native Tools Command Prompt and enter the following commands:

```
cd yourpath\openssl
perl Configure VC-WIN64A
nmake
```

Copy the following files to the bin_x64 directory:

- libcrypto.lib
- libcrypto-1_1x64.dll
- libssl.lib
- libssl-1_1x64.dll
- openssl.exe

Create an inc64 directory under the openssl directory and copy the files from yourpath\openssl\include\openssl to yourpath\openssl\inc64\openssl.

To build a debug version, replace the perl line above with:
perl Configure debug-VC-WIN64A and copy the same files to the bind_x64 directory.

The project files shipped with the sample programs assume that OpenSSL has been installed in the <*installdir*>/thirdPartyCode/openssl directory. If OpenSSL is installed at different location, the project following modifications are required in the Visual Studio GUI once the project's solution file has been loaded:

utils properties
C/C++, General, Additional Include Directories
pathToOpenssl
pathToOpenssl /inc32 (or inc64)

winiotarg project properties
C/C++, General, Additional Include Directories
pathToOpenssl
pathToOpenssl /inc32 (or inc64)

Linker, General, Additional Library Directories
bin, bin_x64, bind, or bind_x64

OpenSSL 1.0.x (obsolete)

cd yourpath\openssl (both here and below replace "yourpath" with the correct path.

The Source Code Library build files assume that OpenSSL will be installed in <*installdir*>/thirdPartyCode/openssl.)

On a Windows machine we did the following:
"C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\vcvarsall.bat" x86
perl Configure VC-WIN32 no-asm --prefix=vc
ms\do_ms
nmake -f ms\ntdll.mak

The above steps should create a directory out32dll that contains the required DLLs libeay32.dll and ssleay32.dll.

To build a debug version in out32dll.dbg, replace the perl line above with:
perl Configure debug-VC-WIN32 no-asm --prefix=vc

You should copy the libeay32.dll, libeay32.lib, ssleay32.dll, and ssleay32.lib to the bin, bind, bin_x64 and bind_x64. The project files as shipped will look for the lib files there when building and the proper dll files must be found when running the application.

The project files shipped with the sample programs assume that OpenSSL has been installed in the <*installdir*>/thirdPartyCode/openssl directory. If OpenSSL is installed at different location, the project following modifications are required in the Visual Studio GUI once the project's solution file has been loaded:

utils properties
C/C++, General, Additional Include Directories
pathToOpenssl
pathToOpenssl /inc32 (or inc64)

winiotarg project properties
C/C++, General, Additional Include Directories
pathToOpenssl
pathToOpenssl /inc32 (or inc64)

Linker, General, Additional Library Directories
bin, bin_x64, bind, or bind_x64

64 bit support:

When building OpenSSL for 64 bit applications the output files and directories created by the standard OpenSSL build continue to use 32 in the names instead of 64. To allow switching between 32 and 64 bit builds the project files shipped with the SCL will look for 64 bit libraries in either the bin_x64, or bind_x64 directories. This requires either copying the 64 bit OpenSSL libeay32.lib and ssleay32.lib files to the bin_x64 or bind_x64 directories or modifying the WinIoTarg project file to look elsewhere.

If the OpenSSL dll's are not in the default execution path, copy the files libeay32.dll and ssleay32.dll into bin_x64 and bind_x64 directories with the executable file 10xMaster.exe, DNPMaster.exe, DNPSlave.exe etc .

Linux:

For debian based distributions, openssl can be installed using the package handling utility:
sudo apt-get install openssl libssl-dev

Alternatively, openssl can be built from source
./config –prefix=/usr/local –openssldir=/usr/local/openssl
make
make test
make install

The Linux Makefile, <installdir>/tmwscl/utils/Makefile shipped with the library is configured to search for the OpenSSL header files in the /usr/include/openssl directory. If OpenSSL not installed in this default location, the include path in this Makefile will need to be modified to include the directory of your OpenSSL installation.

In tmwcrypto.c if TMWCRYPTO_TESTING is defined to 1 (1/on by default) and TMWCRYPTO_ASYMTESTING is defined to 1 (0/off by default) multiple cryptography tests will be performed using known plain and encrypted data to verify the cryptography algorithms are working properly. After verifying your implementation, you may want to change these defines so that these tests are not run every time.

Common errors when building applications with secure authentication and OpenSSL:

Windows:

If you get one of the following errors, you may not have copied OpenSSL to the thirdPartyCode/openssl directory where it is expected. Or have not copied the include files to the inc32/openssl or inc64/openssl include directories.

tmwcrypto.c(72): fatal error C1083: Cannot open include file: 'openssl/e_os2.h': No such file or directory

wintcpchannel.h(50): fatal error C1083: Cannot open include file: 'openssl/ssl.h': No such file or directory

If you get the following error, the OpenSSL lib files have not been copied to bin, bind, bin_x64. Or bind_x64 as expected.

LINK : fatal error LNK1104: cannot open file 'libcrypto.lib'

or

LINK : fatal error LNK1104: cannot open file 'libeay32.lib'

If you get the following error, the you may be using the debug dlls instead of the release dlls or vice versa.

Unhandled exception at 0x….. (udcrtbase.dll) in xxxx.exe: An invalid parameter was passed to a function that considers invalid parameters fatal.

Linux:

If you get the following error, the include path has not been set correctly:

tmwcrypto.c:65:19: fatal error: e_os2.h: No such file or directory #include "e_os2.h"

Edit the file <*installdir*>/tmwscl/utils/Makefile and update the Makefile variable INCLUDES to the OpenSSL installation include directory.

If you get a large number of link errors such as:

```
../tmwscl/utils.lib(tmwcrypto.o): In function `tmwcrypto_init':
/projects/dnpMasterSA/tmwscl/utils/tmwcrypto.c:177: undefined reference to
`CRYPTO_set_mem_functions'
/projects/dnpMasterSA/tmwscl/utils/tmwcrypto.c:178: undefined reference to
`ERR_load_crypto_strings'
/projects/dnpMasterSA/tmwscl/utils/tmwcrypto.c:179: undefined reference to
`OPENSSL_add_all_algorithms_noconf'
/projects/dnpMasterSA/tmwscl/utils/tmwcrypto.c:180: undefined reference to
`ENGINE_load_builtin_engines'
/projects/dnpMasterSA/tmwscl/utils/tmwcrypto.c:181: undefined reference to `RAND_seed'
../tmwscl/utils.lib(tmwcrypto.o): In function `tmwcrypto_MACValue':
/projects/dnpMasterSA/tmwscl/utils/tmwcrypto.c:361: undefined reference to `EVP_sha1'
/projects/dnpMasterSA/tmwscl/utils/tmwcrypto.c:361: undefined reference to `HMAC'
/projects/dnpMasterSA/tmwscl/utils/tmwcrypto.c:377: undefined reference to `EVP_sha256'
/projects/dnpMasterSA/tmwscl/utils/tmwcrypto.c:377: undefined reference to `HMAC'
```

The openssl libraries are not being included in the application. The file, <*installdir*>/*application*/Makefile will need to be updated to include the openssl libraries (-lssl and -lcrypto) in the Makefile variable LIBS. A library path may also need to be added to the Makefile, if the libraries are not installed in the default location.

Once the application builds successfully, run the application.

Windows:

If you get the following popup error you probably don't have the OpenSSL dlls in the same directory as the executable.

The code execution cannot proceed because libcrypto-1_1.dll was not found.
or
The program can't start because LIBEAY32.dll is missing from your computer.

If you get the following popup error you are probably are using OpenSSL dlls that have been built with a different c library (Visual Studio Version) than your application is being built with.

Program:DNPMaster.exe
File …setmode.cpp
Expression: (_osfile(fh) & FOPEN)
…

If you get the following errors in the Windows Command Prompt window, this indicates that all encryption tests are failing. This may mean you have not modified tmwcnfg.h to use the OpenSSL implementation for the tmwcrypto interface.

**** Error: tmwcrypto: tmwcrypto_encryptData not implemented ****
**** Error: tmwcrypto: encryption failed. ****
**** Error: tmwcrypto: failed crypto test 1 ****
**** Error: tmwcrypto: failed crypto test 2 ****
Etc.

If you get the following errors in the Windows Command Prompt window, this indicates you have defined TMWCRYPTO_ASYMTESTING in tmwcrypto.c and that the cryptography testing failed for the asymmetric tests. This normally indicates that DNPMaster.exe cannot find the key files TMWTestxxxKey.pem. These files are installed in the DNPMaster directory, but if you are running from another directory, for example bin, they will not be found. You should move these files to the directory you are running from.

**** Error: tmwcrypto: encryption failed. ****
**** Error: tmwcrypto: failed crypto asymmetric test 7 ****
**** Error: tmwcrypto: failed crypto asymmetric test 8 ****

If you get the following error, it means you have not implemented a database to store important information related to Secure Authentication and you will have limited SA functionality.

**** Error: tmwcrypto: Using simulated cryptography
TMWCNFG_USE_SIMULATED_CRYPTO_DB. Must implement a database. ****

If you get the following popup error you may be trying to use 32 bit OpenSSL dlls with a 64 bit application or vice versa.

The application was unable to start correctly (0xc000007b).

5.7 TLS

TLS functionality is implemented in the provided target implementations WinIoTarg and LinIoTarg and may be implemented in your target layer if you are not using one of those. The target layer interfaces with OpenSSL to provide the TLS functionality. This functionality is separate from the Secure Authentication code implemented in some of the Protocol Libraries. The SCLs do not use the Secure Authentication tmwcrypto interface to access TLS. TMWCNFG_USE_OPENSSL must be defined to TMWDEFS_TRUE to compile in the TLS functionality. The previous section on OpenSSL Integration provides details about building and using OpenSSL with the Source Code Libraries.

The sample applications for protocols that can connect over TCP include example code for configuring and enabling TLS. By default, that example code is compiled or commented out. You should modify the application code you are using to enable TLS if desired. Certificate or Asymmetric Key files are required for TLS. Example files are included in the TMWCertificates directory or you may choose to use those provided by the Test Harness installed in a common public directory. You should determine what Certificate files you are using and configure compatible files for both ends of the TLS connection.

If you get the following error, it means the Key/Certificate files are not where they are expected to be. You can modify the configuration in the sample applications to point to the directory where your files are located.

09:40:48.604: ### DNP Master - 127.0.0.1:20000 - TLS, Can't read certificate from file

09:40:48.620: ### DNP Master - 127.0.0.1:20000 - TLS, Can't read DSA or RSA key files

5.8 File Transfer

Both DNP3 and IEC 60870-5-101 and 104 provide file transfer functionality within the protocol. The file transfer mechanisms specified between DNP3 and 101/104 are very different. DNP file transfer follows a familiar Open File, Read Data, Write Data, Close File paradigm. The 101 and 104 protocols specify a very different procedure that will be described below. With the DNP, 101, and 104 SCLs an interface for accessing files is provided with a complete implementation for both Windows and Linux. For other target environments you should use these implementations as examples to implement the interface for your target system.

The TMW Protocol Test Harness is a good tool for testing File Transfer in these protocols. It provides the ability to send, receive, and interpret all of the file transfer requests and responses as well as to provide access to Windows files in the PC it is running on. See the Protocol Test Harness User Manual for more information.

5.8.1 DNP File Transfer

DNP file transfer works in a familiar manner. The Master can send Open File, Close File, Read, Write, Delete File commands to the Outstation to perform file transfer in both directions. There is also File "Authentication" containing a User Name and clear text Password which is no longer recommended because it is not secure. Responses which are delayed can be sent as events when they are ready.

The target file access required by the Master and Outstation libraries is significantly different. Two separate interfaces mdnptarg.h and sdnptarg.h are provided to facilitate that. Fully functional code is provided in mdnptarg.c and sdnptarg.c for both Windows and Linux and can be found in the WinIoTarg and LinIoTarg directories. Calls are made to these interfaces from mdnpdata.c and sdndata.c, or mdnpfsim.c and sdnpfsim.c depending on compilation defines `TMWTARG_SUPPORT_DNPFILEIO` and `TMWCNFG_USE_SIMULATED_DB`.

5.8.2 IEC 101 and 104 File Transfer

File transfer for IEC 101 and 104 follows a different paradigm. You should refer to IEC 60870-5-101Ed 2 for more detail. Each file, like other data points, is identified by a unique combination of Common Address of ASDU(sector address) and IOA. Each file also has a "Name of File" which is a 16 bit integer actually indicating whether it is a "transparent file" or contains disturbance data, sequences of events, sequences of recorded analog value, or indicates reserved or private file types. Files consist of 1 or more sections of up to 64000 octets each with each section transmitted as individual segments with a maximum length of 240 octets each.

To "read" a file the M14 device does not send an Open File request, but rather sends Select File, Call File, Call Section x, Ack Section x, Ack File requests. The S14 device in response to those requests sends File Ready, Section Ready, Segment, Segment, …, Last Segment, …Last Section responses. The actual data is transferred in the Segment

responses which are sent one after the other until the data from the entire section has been sent.

To “write” a file an equivalent but mirrored sequence of requests and responses are exchanged with segments of data being sent in the control direction. The M14 device sends File Ready, Section Ready, Segment, Segment, …, Last Segment, …, Last Section Requests. The S14 device in response to those requests sends Call File, Call Section, Ack Section, …, Ack File responses. There are 8 different TypeId messages defined for File Transfer consisting of FFRNA, FSRNA, FSCNA, FLSNA, FAFNA, FSGNA, FDRTA, and FSCNB.

The target file access required by the M101/104 and S101/104 libraries is similar. A single interface i14targ.h is provided to facilitate that. Fully functional code is provided in i14targ.c for both Windows and Linux and can be found in the WinIoTarg and LinIoTarg directories. In addition to the target layer interface there is file database code in the SCLs, m14filedata.c and s14filedata.c, that calls the functions defined by that interface. Calls are made to m/s14filedata.c from m14data.c and s14data.c or m14sim.c and s14sim.c depending on compilation defines `TMWTARG_SUPPORT_DNPFILEIO` and `TMWCNFG_USE_SIMULATED_DB`.

5.8.2.1 IEC 101 and 104 File Names in Windows and Linux

The sample file database implementations provided in m14filedata.c and s14filedata.c converts numeric IOA, Name of File, and Sector Name to a filename string to be used by the target layer. Note this convention is not required an implementation even if it runs on Windows or Linux. You can choose to map the protocol specific numeric values to file data as you choose. This will be a local decision and is not visible to the remote device.

The provided implementation looks in the specified directory for files with the following naming convention (spaces or lack of them in the name is required as shown).

File system file names must be formatted as follows:

"File IOA1800 1 1" where IOA=1800, FileName=1, SectionName=1
"File IOA1800 1 2" where IOA=1800, FileName=1, SectionName=2
"File IOA1900 4 1" where IOA=1900, FileName=4, SectionName=1

File system directory names (only exist on the s14 device) must be formatted like

"File IOA1804 1810" where IOA=1804, FileName=1810, No SectionName
"File IOA1805 1820" where IOA=1805, FileName=1820, No SectionName

In 101/104 the Name of File in for a Directory is used to provide a mapping to the IOAs of the files in that directory. For the default implementation the FileName indicates what range of file IOAs are contained in that directory. For example, directory File IOA1804 1810 can only contain files of the range IOA 1810-1819. File IOA1805 1820 will contain files with an IOA 1820 or higher with no limit because there are no other directories identified.

By default, the directories for the files will be :

../m14files/
../s14files/

If you are using multiple sectors, you may want to specify separate directories for the sectors. This can be set by calling m/s14filedata_setFileDirectory(…);

If you are using TMWCNFG_USE_SIMULATED_DB some default files will be created in the s14files directory. If you are not using the simulated database you may want to create some files with names that comply with the above convention.

Chapter 6 Debugging and Testing your Implementation

This chapter will discuss techniques and tools which can be used to test and debug the TMW SCL on the target device. The information in this chapter consists of techniques that have proven useful in previous situations. Each device and implementation is different and it is up to the user whether these techniques are appropriate or not.

It is recommended that testing begins as soon as the device is able to communicate and continues incrementally until all the desired features are complete. This allows problems to be found before they are built upon and potentially made more complex. It also allows the user to see progress from the very early stages of the port.

The examples in this chapter make use of the Triangle MicroWorks Inc. Protocol Test Harness. The Protocol Test Harness is a Windows application that supports the testing of master or slave devices for all the protocols supported by the TMW SCLs. Use of the TMW Protocol Test Harness is not required, but a system with similar features is highly desirable. A 21 day evaluation version of the TMW Protocol Test Harness can be downloaded from the Triangle MicroWorks, Inc. web site at http://www.TriangleMicroWorks.com/downloads.htm

6.1 Testing Basic Communications

Basic communications can be tested as soon as the TMW SCL target interface has been completed and the ability to establish a channel, session, and sector (if required) provided. The recommended approach to testing basic communications is to open a single channel, session, and sector at each end of the communication channel and test this using application level commands. It is entirely possible to test the physical, link, and transport layers independently using the low level TMW SCL APIs and the TMW Protocol Test Harness but this is generally not required.

TMW master and slave SCLs support a simulated database that will provide reasonable execution for most application layer commands. This simulated database should be adequate for this initial testing.

The Triangle MicroWorks, Inc. Communication Protocol Test Harness can be used to facilitate communications testing. The Test Harness is a Windows application that acts as a simple Master or Slave device and can be programmed with an automated test sequence through a scripting capability.

The protocol analyzer output from the Communication Protocol Test Harness allows you to verify the proper operation of the device you are testing. It also allows you to generate a reference protocol analyzer output file that can be used for comparison to later tests to verify only intended modifications have occurred.

The Communication Protocol Test Harness currently supports the following protocol components: IEC 60870-5-101, IEC 60870-5-102, IEC 60870-5-103, IEC 60870-5-104, DNP3 and Modbus.

You may download a full 21-day evaluation version of the Communication Protocol Test Harness (including documentation) from the Triangle MicroWorks, Inc. website at: http://www.TriangleMicroworks.com/downloads.htm.

6.2 Testing Multiple Channels, Sessions, and Sectors

If the target device is designed to support multiple communication channels, sessions, and/or sectors this should be tested once the basic communications are complete. Once again the simulated database should be sufficient for this purpose.

6.3 Testing the Database Interface

Depending on the particular implementation, it is generally recommended to implement and test each data type independently. This is easily accomplished by implementing the required database access routines for the desired data type and issuing commands to exercise these routines.

6.4 Testing Event Generation

Event generation can be tested at the same time as the basic database interface or it can be delayed until the database interface is complete.

6.5 Testing Commands

The generation/processing of commands should be tested next. While the execution of some commands will obviously be done in the preceding steps, it is now time to thoroughly test all of the supported commands.

6.6 Regression Testing

Once the device has been thoroughly tested it is highly recommend that all the test scripts be assembled into a complete procedure that can be executed periodically to guarantee continued functionality.

Triangle MicroWorks, Inc. sells Conformance Test scripts that run on the Communication Protocol Test Harness. These scripts perform the conformance tests outlined by the technical committees of each supported protocols.

The scripts can be run in “attended” or “unattended” mode. The attended mode is required to verify complete device performance. The unattended mode skips steps that require specific configuration or values from the device. This mode is useful in automated regression tests.

1 DNP3 Master

This section provides detailed information specific to a DNP3 Master implementation. It includes a list of all SCL functions that are called by the target application and two lists of target application functions that may be called by the SCL. These lists are followed by an example DNP3 Master application that shows how to interface an application to the SCL. Finally, header file listings are provided for:

- Configuration files that may need to be modified
- Calls from the target application into the SCL
- Calls from the SCL into the target application
  - Target layer interface
  - Target database functions.

The interface between the Source Code Library (SCL) and Target Application (TA) can be viewed as a three-sided, or "triangle" interface. Two sides represent calls back into the TA from the SCL Interface, while the third side represents calls into the SCL from the TA. Each of these sides is organized into individual, well-documented modules or header files. These files are the only recommended customer-editable (or platform-specific) files. All other files are for internal use by the SCL and should not be called or modified by the customer.

1.1 Target Application Calls into the Source Code Library

1.1.1 General calls

Table 1 lists the function calls into the SCL from the target application. The definitions for these functions are in the corresponding header files, as shown in the table. These header files are listed in Section 0.

Refer to the Table of Contents of this section for assistance in locating the code listings for these header files.

Table 1. Target Application calls into Source Code Library (MDNP)

	Function Name	Header File
PT	tmwtimer_initialize	tmwtimer.h
	tmwappl_initApplication	tmwappl.h
	tmwtarg_initConfig	tmwtarg.h
	dnpchnl_initConfig	dnpchnl.h
	dnpchnl_openChannel	dnpchnl.h
	mdnpsesn_initConfig	mdnpsesn.h
	mdnpsesn_openSession	mdnpsesn.h
PT	tmwpltmr_checkTimer	tmwpltmr.h
PI	tmwappl_checkForInput	tmwappl.h
A	tmwdb_init	tmwdb.h
A	tmwdb_storeEntry	tmwdb.h
	mdnpbrm functions (see Table 2)	mdnpbrm.h

KEY:

PT These functions are only used if the optional Polled Timer implementation is used. It is not required if the low-level target interface supplies a system timer (e.g., OS timer) to driver the timer functions. See Section 3.2.11 in the SCL User Manual for more information about Polled vs. Event timers.

PI This function is only used if the polled input implementation is used. It is not required if event-driven input is used. See Section 3.2.11 in the SCL User Manual

A These functions are only used if the optional Asynchronous database option is used.

1.1.2 Command Functions

The Master Station can initiate requests to the remote. These requests are initiated through the mdnpbrm_xxx functions. The definitions for these functions are in the mdnpbrm.h header file, as shown in **Table 2**. These header files are listed in Section 0.

Refer to the Table of Contents of this section for assistance in locating the code listings for these header files.

Table 2. Master Command Functions (DNP3)

Function Name	Header File
mdnpbrm_initBroadcastDesc	mdnpbrm.h
mdnpbrm_initReqDesc	mdnpbrm.h
mdnpbrm_buildRequestHeader	mdnpbrm.h
mdnpbrm_addObjectHeader	mdnpbrm.h
mdnpbrm_addObjectData	mdnpbrm.h
mdnpbrm_readClass	mdnpbrm.h
mdnpbrm_integrityPoll	mdnpbrm.h
mdnpbrm_eventPoll	mdnpbrm.h
mdnpbrm_*xxx*FeedbackPoll	mdnpbrm.h
mdnpbrm_clearRestart	mdnpbrm.h

mdnpbrm_coldRestart	mdnpbrm.h
mdnpbrm_warmRestart	mdnpbrm.h
mdnpbrm_recordCurrentTime	mdnpbrm.h
mdnpbrm_writeRecordedTime	mdnpbrm.h
mdnpbrm_unsolEnable	mdnpbrm.h
mdnpbrm_unsolDisable	mdnpbrm.h
mdnpbrm_assignClass	mdnpbrm.h
mdnpbrm_readGroup	mdnpbrm.h
mdnpbrm_readPoints	mdnpbrm.h
mdnpbrm_binaryCommand	mdnpbrm.h
mdnpbrm_sendPatternMask	mdnpbrm.h
mdnpbrm_analogCommand	mdnpbrm.h
mdnpbrm_writeDeadband	mdnpbrm.h
mdnpbrm_freezeCounters	mdnpbrm.h
mdnpbrm_writeString	mdnpbrm.h
mdnpbrm_writeVirtualTerminal	mdnpbrm.h
mdnpbrm_fileInfo	mdnpbrm.h
mdnpbrm_fileAuthentication	mdnpbrm.h
mdnpbrm_fileOpen	mdnpbrm.h
mdnpbrm_fileClose	mdnpbrm.h
mdnpbrm_fileDelete	mdnpbrm.h
mdnpbrm_fileAbort	mdnpbrm.h
mdnpbrm_fileRead	mdnpbrm.h
mdnpbrm_fileWrite	mdnpbrm.h
mdnpbrm_copyLocalFileToRemote	mdnpbrm.h
mdnpbrm_copyRemoteFileToLocal	mdnpbrm.h

KEY:
xxx Represents one or more of: binOut, anlgOut, frznCntr

1.1.3 Callback Functions

In addition to the above functions, there are several opportunities for the target application to call back into the Source Code Library via a Callback Function. Calling a Callback Function simply consists of calling a function whose address was passed into the target application from the Source Code Library. When calling a callback function, the Callback Parameter passed in the structure with the function must be passed back as a parameter.

In most cases, calling callback functions implements optional functionality. For more information on Callback Functions, please refer to Section 3.2.9 Application Interface in the SCL User Manual.

Table 3 summarizes the callback functions that allow the target implementation to call into the Source Code Library.

Table 3. Target Application calls into Source Code Library

	Callback Function	Function Name	Header File	Notes
PE	Channel Callback	TMWTARG_CHANNEL_CALLBACK_FUNC	tmwtarg.h	Passed to target in tmwtarg_initChannel (). Allows target to indicate status of channel (open/connect or closed/disconnect)
PE	Data Received	TMWTARG_CHANNEL_READY_CBK_FUNC	tmwtarg.h	Passed to target in tmwtarg_initChannel ().Allows target to indicate that data has been received and should be read by SCL
T	Timer Callback	TMWTYPES_CALLBACK_FUNC	tmwtypes.h	Should be called by target after the specified number of milliseconds. Passed to target as parameter to tmwtarg_startTimer (for single timer implementations) or tmwtarg_startMultiTimer (for multiple timer implementations).

KEY:

T This function is only used if the optional Polled Timer implementation is not used. It is required if the low-level target interface supplies a system timer (e.g., OS timer) to driver the timer functions. See Section 3.2.11 in the SCL User Manual for more information about Polled vs. Event timers.

PE These functions are only used if the event-driven input implementation is used. They are not required if the polled nput is used. See Section 3.2.11 in the SCL User Manual

1.2 Calls from the Source Code Library into the Target Application

1.2.1 Low-level Target Interface

Table 4 lists the low-level target interface functions. These functions are calls into the target application from the Source Code Library, and must be written for the target application.

As shown in the table, these functions are defined in tmwtarg.h. The implementation should be added in tmwtarg.c in the blocks labeled

```
/* Put your code here */
```

For the definitions of these functions, please refer to Section 1.2.1.

Table 4. Low-level Target Interface Functions

	Function Name	Header File
	tmwtarg_initconfig	tmwtarg.h
D	tmwtarg_alloc	tmwtarg.h
D	tmwtarg_free	tmwtarg.h
	tmwtarg_snprintf	tmwtarg.h
P	tmwtarg_putdiagstring	tmwtarg.h
X	tmwtarg_appendstring	tmwtarg.h
	tmwtarg_get8	tmwtarg.h
	tmwtarg_store8	tmwtarg.h
	tmwtarg_get16	tmwtarg.h
	tmwtarg_store16	tmwtarg.h
	tmwtarg_get24	tmwtarg.h
	tmwtarg_store24	tmwtarg.h
	tmwtarg_get32	tmwtarg.h
	tmwtarg_store32	tmwtarg.h
	tmwtarg_get64	tmwtarg.h
	tmwtarg_store64	tmwtarg.h
	tmwtarg_getSFloat	tmwtarg.h
	tmwtarg_putSFloat	tmwtarg.h
	tmwtarg_getmstime	tmwtarg.h
	tmwtarg_getdatetime	tmwtarg.h
	tmwtarg_setdatetime	tmwtarg.h
	tmwtarg_starttimer	tmwtarg.h
	tmwtarg_canceltimer	tmwtarg.h
M	tmwtarg_startmultitimer	tmwtarg.h
M	tmwtarg_cancelmultitimer	tmwtarg.h
	tmwtarg_initchannel	tmwtarg.h
	tmwtarg_deletechannel	tmwtarg.h
P	tmwtarg_getchannelname	tmwtarg.h
	tmwtarg_getchannelinfo	tmwtarg.h
	tmwtarg_openchannel	tmwtarg.h
	tmwtarg_closechannel	tmwtarg.h
P	tmwtarg_getsessionname	tmwtarg.h
P	tmwtarg_getsectorname	tmwtarg.h
	tmwtarg_gettransmitready	tmwtarg.h
	tmwtarg_receive	tmwtarg.h
	tmwtarg_transmit	tmwtarg.h
N	tmwtarg_transmitudp	tmwtarg.h
M	tmwtarg_lockinit	tmwtarg.h
M	tmwtarg_locksection	tmwtarg.h
M	tmwtarg_unlocksection	tmwtarg.h
M	tmwtarg_lockdelete	tmwtarg.h
M	tmwtarg_lockshare	tmwtarg.h

KEY:
D These functions are only used if Dynamic Memory is used
M These functions are used only if multi-threading is used
N This function is used only for support of the DNP3 Specification for IP Networking; it is not used by other protocols
P These functions are only used for diagnostic support, such as the Protocol Analyzer
X These functions are only required to support the generation of an XML document from the target database

1.2.2 Database Interface

Table 5 lists the database interface functions for DNP3 Master implementations. These functions are calls into the target application from the Source Code Library, and must be written for the target application.

Database functions are all defined in mdnpdata.h. The implementation should be added in mdnpdata.c in the blocks labeled

```
/* Put target code here */
```

For the definitions of these functions, refer to Section 7.

Table 5. Database Interface Functions (MDNP)

	Function Name	Header File
	mdnpdata_processIIN	mdnpdata.h
	mdnpdata_storeIIN	mdnpdata.h
	mdnpdata_init	mdnpdata.h
	mdnpdata_close	mdnpdata.h
	mdnpdata_store*xxx*	mdnpdata.h
F	mdnpdata_store*fff*	mdnpdata.h
F	mdnpdata_getFileAuthKey	mdnpdata.h
F	mdnpdata_openLocalFile	mdnpdata.h
F	mdnpdata_closeLocalFile	mdnpdata.h
F	mdnpdata_readLocalFile	mdnpdata.h

KEY:

F These functions are only required if File Transfer is supported

xxx Represents one or more of: ReadTime, BinaryInput, DoubleInput, BinaryOutput, BinaryCounter, FrozenCounter, AnalogInput, AnalogInputDeadband,AnalogOutput, RestartTime, String, VirtualTerminal

fff Represents one or more of: AuthKey, FileStatus, FileData, FileDataStatus, FileInfo

1.2.3 Callback Functions

The Source Code Library can call callback functions that can be used to invoke functions within the target application when events take place in the SCL. These callback functions are called when a request completes, an application layer fragment is received, a statistically significant event occurs, etc. **Table 6** summarizes these callback functions.

Table 6. Callback Functions into Target Applications

	Callback Function	Function Name	Header File	Notes
O	Command Response	DNPCHNL_CALLBACK_FUNC	dnpchnl.h	Called for each response to the request, including intermediate responses
O	Idle Callback	TMWCHNL_IDLE_CALLBACK	tmwchnl.h	Callback function and parameter may be specified in fields in DNPCHNL_CONFIG structure when opening a channel
O	Unsolicited Callback	mdnpsesn_setUnsolUserCallback()	mdnpsesn.h	Called when an unsolicited response is received
O	Auto Open Callback	TMWCHNL_AUTO_OPEN_FUNC	tmwchnl.h	Callback function and parameter may be specified in DNPCHNL_CONFIG structure when opening Channel
O	Channel Statistics	TMWCHNL_STAT_CALLBACK	tmwchnl.h	Callback function and parameter is specified in DNPCHNL_CONFIG structure when opening Channel
O	Session Statistics	TMWSESN_STAT_CALLBACK	tmwsesn.h	Callback function and parameter is specified in MDNPSESN_CONFIG structure when opening channel

KEY:
O Implementation of these functions is optional; it is required only if the optional functionality is required.

2 DNP Master Example

'DNPMaster.cpp' contains an example of a simple DNP Master application showing use of the SCL interface. The source code for this application can be found in the directory *<installdir>/DNPMaster* .

```
/*****************************************************************************/
/* Triangle MicroWorks, Inc.                          Copyright (c) 1997-2022 */
/*****************************************************************************/
/*                                                                           */
/* This file is the property of:                                             */
/*                                                                           */
/*                       Triangle MicroWorks, Inc.                           */
/*                      Raleigh, North Carolina USA                          */
/*                       www.TriangleMicroWorks.com                          */
/*                          (919) 870-6615                                   */
/*                                                                           */
/* This Source Code and the associated Documentation contain proprietary     */
/* information of Triangle MicroWorks, Inc. and may not be copied or         */
/* distributed in any form without the written permission of Triangle        */
/* MicroWorks, Inc.  Copies of the source code may be made only for backup   */
/* purposes.                                                                 */
/*                                                                           */
/* Your License agreement may limit the installation of this source code to  */
/* specific products.  Before installing this source code on a new           */
/* application, check your license agreement to ensure it allows use on the  */
/* product in question.  Contact Triangle MicroWorks for information about   */
/* extending the number of products that may use this source code library or */
/* obtaining the newest revision.                                            */
/*                                                                           */
/*****************************************************************************/

// DNPMaster.cpp : Sample DNP Master console application.
//

#if defined(TMW_WTK_TARGET)
#include "StdAfx.h"
#endif

extern "C" {
#include "tmwscl/utils/tmwappl.h"
#include "tmwscl/utils/tmwdb.h"
#include "tmwscl/utils/tmwphys.h"
#include "tmwscl/utils/tmwpltmr.h"
#include "tmwscl/utils/tmwtarg.h"

#include "tmwscl/dnp/dnpchnl.h"
#include "tmwscl/dnp/dnplink.h"
#include "tmwscl/dnp/mdnpbrm.h"
#include "tmwscl/dnp/mdnpsesn.h"

#include "tmwtargio.h"
}

/* The USE_POLLED_MODE constant is used here to demonstrate how the library
 * can be used to configure the target layer to support polled mode vs.
 * event driven. The Linux and Windows TCP channels shipped with the SCL
 * support both. POLLED_MODE FALSE is recommended for Linux.
 * If writing your own target layer, only one configuration is required.
 */
#define USE_POLLED_MODE TMWDEFS_FALSE

#define USE_IPV6        TMWDEFS_FALSE
#define USE_TLS         TMWDEFS_FALSE

  /* To use TLS TMWCNFG_USE_OPENSSL must also be defined as TMWDEFS_TRUE */
#if USE_TLS && !TMWCNFG_USE_OPENSSL
#pragma message("To use TLS TMWCNFG_USE_OPENSSL must be defined")
#endif

/* Open a second channel and outstation session */
TMWTYPES_BOOL   openSecondConnection = TMWDEFS_FALSE;

#if DNPCNFG_SUPPORT_AUTHENTICATION
/* It is possible to start an SAV5 session with no users and have them added by
```

```
 70  * the Authority/Master using a remote key change mehthod for user number 1
"Common".
     */
    TMWTYPES_BOOL     noUserAtStartup = TMWDEFS_FALSE;

    /* Use SAv2 implementation if it is compiled in.
     */
    TMWTYPES_BOOL myUseSAv2;
    #endif


 80
    #if !TMWCNFG_MULTIPLE_TIMER_QS

    /* This can be changed to open multiple sessions per channel */
    #define NUM_SESSIONS 1

    typedef struct mySessionInfo
    {
      TMWTYPES_MILLISECONDS  integrityInterval;
      TMWTYPES_MILLISECONDS  lastIntegrityPoll;
 90   TMWTYPES_MILLISECONDS  eventInterval;
      TMWTYPES_MILLISECONDS  lastEventPoll;
      TMWTYPES_MILLISECONDS  binCmdInterval;
      TMWTYPES_MILLISECONDS  lastBinCmd;

      TMWTYPES_MILLISECONDS  actAsAuthorityInterval;
      TMWTYPES_MILLISECONDS  lastActAsAuthority;
    } MY_SESSION_INFO;


100 /* Config structures */
    TMWTARGIO_CONFIG IOCnfg;
    MDNPSESN_CONFIG sesnConfig;
    DNPCHNL_CONFIG chnlConfig;
    DNPTPRT_CONFIG tprtConfig;
    DNPLINK_CONFIG lnkConfig;
    TMWPHYS_CONFIG physConfig;
    TMWTARG_CONFIG targConfig;

    /* Context pointers */
110 TMWAPPL *pApplContext;
    TMWCHNL *pChannel;
    TMWCHNL *pChannel2;
    TMWSESN *pSessions[NUM_SESSIONS];
    MDNPBRM_REQ_DESC requests[NUM_SESSIONS];
    MDNPBRM_CROB_INFO CROBInfo[NUM_SESSIONS];

    TMWSESN *pSessions2[NUM_SESSIONS];
    MDNPBRM_REQ_DESC requests2[NUM_SESSIONS];

120 MDNPBRM_REQ_DESC SARequest;
    MDNPBRM_REQ_DESC SARequest2;


    /* The following code defines application specific functions
     * that are used in this example.
     *
     * These functions begin with "my" to distinguish them from
     * Source Code Library functions, and are defined at the end
     * of the example in order to emphasize the typical
130  * interface with the Source Code Library.
     */
    MY_SESSION_INFO * myInitMySession(void);
    TMWTYPES_BOOL     myTimeForIntegrity(MY_SESSION_INFO *pMySession);
    TMWTYPES_BOOL     myTimeForEvent(MY_SESSION_INFO *pMySession);
    TMWTYPES_BOOL     myTimeForBinCmd(MY_SESSION_INFO *pMySession);
    void              my_brmCallbackFcn(void *pUserCallbackParam,
                                        DNPCHNL_RESPONSE_INFO *pResponse);
    void              myPutDiagString(const TMWDIAG_ANLZ_ID *pAnlzId,
                                      const TMWTYPES_CHAR *pString);
140
    #if DNPCNFG_SUPPORT_AUTHENTICATION
    #include "tmwscl/dnp/mdnpsim.h"

    TMWTYPES_BOOL     myNowSendUpdateKey  = false;
    TMWTYPES_BOOL     myNowSendUpdateKey2 = false;

    void *            myUserNameDbHandle;

    void              myInitSecureAuthentication(MDNPSESN_CONFIG *pSesnConfig);
```

```
TMWTYPES_BOOL     myAddAuthUsers(MDNPSESN *pMDNPSession, TMWTYPES_BOOL
operateInV2Mode);

    TMWTYPES_BOOL     myTimeToActAsAuthority(MY_SESSION_INFO *pMySession);

    void              myActAsAuthority(MDNPBRM_REQ_DESC *pBRMRequest);
    void              mySendUpdateKey(MDNPBRM_REQ_DESC *pBRMRequest, int i);

    void my_brmSACallbackFcn(void *pUserCallbackParam, DNPCHNL_RESPONSE_INFO
*pResponse);


/* These are the default user keys the Test Harness uses for testing
 * DO NOT USE THESE IN A REAL DEVICE
 */
    static TMWTYPES_UCHAR defaultUserKey1[] = {
      0x49, 0xC8, 0x7D, 0x5D, 0x90, 0x21, 0x7A, 0xAF,
      0xEC, 0x80, 0x74, 0xeb, 0x71, 0x52, 0xfd, 0xb5
    };

    /* It is now recommended that SAV5 should be used with only a single user per
Association
     * This would be the default key that the Test Harness uses for other users.
     * static TMWTYPES_UCHAR defaultUserKeyOther[] = {
     *   0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
     *   0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff
    };
     */

    /* This one is used by the Authority and the Outstation, but
     * since this master is acting as the Authority it needs it.
     */
    static TMWTYPES_UCHAR  authoritySymCertKey[] = {
      0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
      0x09, 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06,
      0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
      0x09, 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06
    };
    #endif

    /* Main application entry point
     */
    int main(int argc, char* argv[])
    {
      size_t i;
      MY_SESSION_INFO  *pMySession;
      MY_SESSION_INFO  *pMySession2 = TMWDEFS_NULL;
      bool run = true;
      bool useSerial = false;

      TMWTARG_UNUSED_PARAM(argc);
      TMWTARG_UNUSED_PARAM(argv);

    #if TMWCNFG_SUPPORT_DIAG
      /* Register function to display diagnostic strings to console
       * This is only necessary if using the Windows or Linux target layer.
       * If implementing a new target layer, tmwtarg_putDiagString()
       * should be modified if diagnostic output is desired.
       */
      tmwtargp_registerPutDiagStringFunc(myPutDiagString);
    #endif

      /*  * Initialize the source code library.
       */
      tmwappl_initSCL();

      /* Initialize DNP SCL. This includes:
       *  - initialize polled timer
       *  - initialize the async database
       *  - initialize application context
       */
      tmwtimer_initialize();
      tmwdb_init(10000);
      pApplContext = tmwappl_initApplication();

      /* Initialize the Target config Structure
       * Call tmwtarg_initConfig to initialize with default values,
       * then overwrite values as needed.
       */
      tmwtarg_initConfig(&targConfig);
```

```
      /* Initialize the Channel config Structure
       * Call dnpchnl_initConfig to initialize with default values,
230    * then overwrite values as needed.
       */
      dnpchnl_initConfig(&chnlConfig, &tprtConfig, &lnkConfig, &physConfig);
      lnkConfig.networkType = DNPLINK_NETWORK_TCP_UDP;

      /* Initialize the common target IO configuration structure.
       * Call tmwtargio_initConfig to initialize default values, then overwrite
       * specific values as needed.
       */
      tmwtargio_initConfig(&IOCnfg);
240


      if(useSerial)
      {
        IOCnfg.type = TMWTARGIO_TYPE_232;

        /* Name displayed in analyzer window */
        strcpy(IOCnfg.targ232.chnlName, "DNP Master");

250     strcpy(IOCnfg.targ232.baudRate, "9600");
        IOCnfg.targ232.numDataBits = TMWTARG232_DATA_BITS_8;
        IOCnfg.targ232.numStopBits = TMWTARG232_STOP_BITS_1;
        IOCnfg.targ232.parity      = TMWTARG232_PARITY_NONE;
        IOCnfg.targ232.portMode    = TMWTARG232_MODE_NONE;
        IOCnfg.targ232.polledMode  = USE_POLLED_MODE;

    #if defined(TMW_WTK_TARGET)
        /* COM port to open */
        strcpy(IOCnfg.targ232.portName, "COM1");
260 #endif

    #if defined(TMW_LINUX_TARGET)
        /* port to open */
        strcpy(IOCnfg.targ232.portName, "/dev/ttyS0");
    #endif
      }
      else
      {
        /* Use TCP not RS232 */
270     IOCnfg.type = TMWTARGIO_TYPE_TCP;

        /* Name displayed in analyzer window */
        strcpy(IOCnfg.targTCP.chnlName, "DNP Master");

        /* TCP/IP address of remote device
         * 127.0.0.1 is the loopback address and will attempt to connect to an
outstation listener on this computer
         */
        strcpy(IOCnfg.targTCP.ipAddress, "127.0.0.1");

280     /* Connect to designated IP Address */
        //strcpy(IOCnfg.targTCP.ipAddress, "192.168.56.1");
        //strcpy(IOCnfg.targTCP.ipAddress, "192.168.1.100");

        /* You can also specify a NIC and/or local IP Address */
        //strcpy(IOCnfg.targTCP.nicName, "enp0s3");
        //strcpy(IOCnfg.targTCP.localIpAddress, "192.168.56.102");
        //strcpy(IOCnfg.targTCP.nicName, "enp0s8");
        //strcpy(IOCnfg.targTCP.localIpAddress, "192.168.1.117");

290     /* Port on outstation to connect to */
        IOCnfg.targTCP.ipPort = 20000;
        IOCnfg.targTCP.polledMode = USE_POLLED_MODE;

        /* Initiate the connection */
        IOCnfg.targTCP.mode = TMWTARGTCP_MODE_CLIENT;
        IOCnfg.targTCP.role = TMWTARGTCP_ROLE_MASTER;

        /* TLS Configuration */
        IOCnfg.targTCP.useTLS = USE_TLS;
300     /* You may want to change the TLS configuration from the following default
values:
         *  IOCnfg.targTCP.caCrlFileName
"../TMWCertificates/ca_public/tmw_sample_ca_certificate_revocation_list.pem"
         *  IOCnfg.targTCP.caFileName
"../TMWCertificates/ca_public/tmw_sample_ca_rsa_public_certificate.pem"
         *  IOCnfg.targTCP.dhFileName ""
         *  IOCnfg.targTCP.tlsCommonName "TLS"
```

```
         *  IOCnfg.targTCP.tlsRsaCertificateId
"../TMWCertificates/client_user/tmw_sample_tls_rsa_public_cert.pem"
         *  IOCnfg.targTCP.tlsRsaPrivateKeyFile
"../TMWCertificates/client_user/tmw_sample_tls_rsa_private_key.pem"
         *  IOCnfg.targTCP.tlsRsaPrivateKeyPassPhrase "triangle"
         */

#if TMWTARG_SUPPORT_UDP
        /* Only one channel can bind to the local UDP port */
        IOCnfg.targTCP.localUDPPort = 20001;

        /* Broadcast address to use for sending broadcast requests over UDP
         * This must be valid relative to the ipAddress you are connecting to
         */
        strcpy(IOCnfg.targTCP.udpBroadcastAddress, "192.168.1.255");

    #endif

    #if USE_IPV6
        /* If you want to use IPV6 instead */
        IOCnfg.targTCP.ipVersion = TMWTARG_IPV6;

        /* IPV6 Loopback address */
        strcpy(IOCnfg.targTCP.ipAddress, "::1");

        /* IPV6 specific address */
        //strcpy(IOCnfg.targTCP.ipAddress, "2600:1700:850:8690:64b5:ad53:9dbd:28ff");
        //strcpy(IOCnfg.targTCP.ipAddress, "2600:1700:850:8690:28f3:d72b:f609:c41a");
        //strcpy(IOCnfg.targTCP.ipAddress, "2600:1700:850:8690:dcb:c4b7:b101:9a48");

        /* IPV6 broadcast address */
        strcpy(IOCnfg.targTCP.udpBroadcastAddress, "FF02::1");

        /* You can also specify a NIC and/or local IP Address */
        //strcpy(IOCnfg.targTCP.nicName, "enp0s8");
        strcpy(IOCnfg.targTCP.localIpAddress, "::");
        //strcpy(IOCnfg.targTCP.localIpAddress,
"2600:1700:850:8690:64b5:ad53:9dbd:28ff");
#endif

      }

      bool dualEndPoint = false;
      bool udpOnly = false;
      /* This code will configure the channel for Dual End Point mode
       * Which is both a Client AND Server
       * It will both listen and try to connect to the remote device when
       * it has data to send. This mode should be used on both ends.
       */
      if(dualEndPoint)
      {
        IOCnfg.targTCP.mode = TMWTARGTCP_MODE_DUAL_ENDPOINT;
        IOCnfg.targTCP.dualEndPointIpPort = 20000;
      }
      else if(udpOnly)
      {
        /* This code will configure the channel for UDP only.
         */
        IOCnfg.targTCP.mode = TMWTARGTCP_MODE_UDP;
        lnkConfig.networkType = DNPLINK_NETWORK_UDP_ONLY;

    #if TMWTARG_SUPPORT_UDP
        /* Choose the local UDP port to send and receive on */
        IOCnfg.targTCP.localUDPPort = 20001;

        IOCnfg.targTCP.destUDPPort =  20000;

    #endif
      }

     /* Open the DNP Channel
      */
      pChannel = dnpchnl_openChannel(pApplContext, &chnlConfig, &tprtConfig,
        &lnkConfig, &physConfig, &IOCnfg, &targConfig);

      if(pChannel == TMWDEFS_NULL)
      {
        /* Failed to open channel */
        printf("Failed to open channel, exiting program \n");
```

```
        /* Sleep for 10 seconds before exiting */
        tmwtarg_sleep(10000);
        return (1);
      }

      /*   * Initialize and open the DNP Sessions
       */
390   mdnpsesn_initConfig(&sesnConfig);


      // If using TCP the DNP Spec requires keep alives to be configured in order to
detect disconnects.
      if(!useSerial && !udpOnly)
        sesnConfig.linkStatusPeriod = 30000;

    #if DNPCNFG_SUPPORT_AUTHENTICATION
      /* Set this to true to enable DNP3 Secure Authentication support */
      bool myUseAuthentication = true;
400
      /* If Secure Authentication is to be enabled */
      if(myUseAuthentication)
        myInitSecureAuthentication(&sesnConfig);
    #endif

      for (i = 0; i < NUM_SESSIONS; i++)
      {
        sesnConfig.destination = (unsigned short)(4 + i);
    #if DNPCNFG_SUPPORT_AUTHENTICATION
410     sesnConfig.authConfig.assocId = (TMWTYPES_USHORT)i;
    #endif
        /* Last argument is pUserHandle which can be used to uniquely identify session.
         * Sets pMDNPSession->dnp.pCryptoHandle for example
         */
        pSessions[i] = mdnpsesn_openSession(pChannel, &sesnConfig, (void*)(i+1));
        if(pSessions[i] == TMWDEFS_NULL)
        {
          /* Failed to open session */
          printf("Failed to open Session %zd, exiting program \n", i);
420
          /* Sleep for 10 seconds before exiting */
          tmwtarg_sleep(10000);
          return (1);
        }

    #if DNPCNFG_SUPPORT_AUTHENTICATION
        if (myUseAuthentication)
        {
          if (!myAddAuthUsers((MDNPSESN*)pSessions[i],
sesnConfig.authConfig.operateInV2Mode))
430       {
            /* Sleep for 10 seconds before exiting */
            tmwtarg_sleep(10000);
            return (1);
          }
        }
    #endif
      }

      /*   * Initialize Request Descriptors
440    */
      for (i = 0; i < NUM_SESSIONS; i++)
      {
        mdnpbrm_initReqDesc(&requests[i], pSessions[i]);
        requests[i].pUserCallback = my_brmCallbackFcn;
        requests[i].pUserCallbackParam = (void *)i;
      }

    #if MDNPCNFG_SUPPORT_SA_VERSION5 && DNPCNFG_SUPPORT_AUTHKEYUPDATE
      /* Only the first session on a channel will send the SA Authority Remote Update
requests in this example */
450   mdnpbrm_initReqDesc(&SARequest, pSessions[0]);
      SARequest.pUserCallbackParam = (void *)0;
    #endif

      pMySession = myInitMySession();

      /* This code will open a second channel and master session */
      if(openSecondConnection)
      {
        /* name displayed in analyzer window */
460     strcpy(IOCnfg.targTCP.chnlName, "Master2");
```

```
        IOCnfg.targTCP.ipPort = 20001;
        IOCnfg.targTCP.polledMode = USE_POLLED_MODE;

       /*     * Open the DNP Channel
        */
        pChannel2 = dnpchnl_openChannel(pApplContext, &chnlConfig, &tprtConfig,
          &lnkConfig, &physConfig, &IOCnfg, &targConfig);

        if(pChannel2 == TMWDEFS_NULL)
470     {
          /* Failed to open channel */
          printf("Failed to open second channel, exiting program \n");

          /* Sleep for 10 seconds before exiting */
          tmwtarg_sleep(10000);
          delete pMySession;
          return (1);
        }

480     /*     * Initialize and open the DNP Sessions
        */
        for(i = 0; i < NUM_SESSIONS; i++)
        {
          sesnConfig.destination = (unsigned short)(4 + i);
          pSessions2[i] = mdnpsesn_openSession(pChannel2, &sesnConfig, TMWDEFS_NULL);
          if(pSessions2[i] == TMWDEFS_NULL)
          {
            /* Failed to open session */
            printf("Failed to open Session %zd, exiting program \n", i);
490
            /* Sleep for 10 seconds before exiting */
            tmwtarg_sleep(10000);
            return (1);
          }

    #if DNPCNFG_SUPPORT_AUTHENTICATION
          if (myUseAuthentication)
          {
            if (!myAddAuthUsers((MDNPSESN*)pSessions2[i],
sesnConfig.authConfig.operateInV2Mode))
500         {
              /* Sleep for 10 seconds before exiting */
              tmwtarg_sleep(10000);
              return (1);
            }
          }
    #endif
        }
        /*     * Initialize Request Descriptors
        */
510     for(i = 0; i < NUM_SESSIONS; i++)
        {
          mdnpbrm_initReqDesc(&requests2[i], pSessions2[i]);
          requests2[i].pUserCallback = my_brmCallbackFcn;
          requests2[i].pUserCallbackParam = (void *)i;
        }

    #if MDNPCNFG_SUPPORT_SA_VERSION5 && DNPCNFG_SUPPORT_AUTHKEYUPDATE
        /* Only the first session on a channel will send the SA Authority Remote Update
requests in this example */
        mdnpbrm_initReqDesc(&SARequest2, pSessions2[0]);
520     SARequest2.pUserCallbackParam = (void *)1;
    #endif

        pMySession2 = myInitMySession();
    #if DNPCNFG_SUPPORT_AUTHENTICATION
        if (noUserAtStartup)
          pMySession2->actAsAuthorityInterval = 10000;  /* If SAv5, add first user over
DNP in 10 seconds  */
    #endif
      }

530
      /* Now that everything is set up, start a "main loop"
       * that sends and processes requests.
       */
      while (run == TMWDEFS_TRUE)
      {
      /* See if it's time to send a request.
       * This simple demo only issues the following requests:
       *  - Integrity Poll
```

```
        *  - Event Poll
        *  - Binary Output Command
        */
        if(myTimeForIntegrity(pMySession))
        {
          for (i = 0; i < NUM_SESSIONS; i++)
          {
            printf("Integrity Poll for Session %zd\n", i);
            mdnpbrm_integrityPoll(&requests[i]);
          }
        }

        if(myTimeForEvent(pMySession))
        {
          for (i = 0; i < NUM_SESSIONS; i++)
          {
            printf("Event Poll for Session %zd\n", i);
            mdnpbrm_eventPoll(&requests[i]);
          }
        }

        if(myTimeForBinCmd(pMySession))
        {
          for (i = 0; i < NUM_SESSIONS; i++)
          {
            printf("Issue Binary Output Command for Session %zd\n", i);

            CROBInfo[i].pointNumber = 0;
            CROBInfo[i].control = (TMWTYPES_UCHAR)(CROBInfo[i].control !=
DNPDEFS_CROB_CTRL_LATCH_ON
                ? DNPDEFS_CROB_CTRL_LATCH_ON : DNPDEFS_CROB_CTRL_LATCH_OFF);
            CROBInfo[i].onTime = 0;
            CROBInfo[i].offTime = 0;

    #if DNPCNFG_SUPPORT_AUTHENTICATION
            // If secure authentication is enabled, you can choose to send this
aggressive mode
            // to avoid the challenge/reply exchange
            if(sesnConfig.authenticationEnabled)
            {
              requests[i].authAggressiveMode = true;
              // You can either use the default user (1) or choose another one that is
configured
              requests[i].authUserNumber = DNPAUTH_DEFAULT_USERNUMBER;

              /* When using SAv5 use the user number that was sent in response to user
being added
               * dynamically over DNP. This SHOULD be the default user since only 1 per
association
               * is recommended.
               */
              if(!sesnConfig.authConfig.operateInV2Mode)
              {
                MDNPSESN *pMDNPSession = (MDNPSESN*)requests[i].pSession;
                TMWTYPES_USHORT userNumber = mdnpsim_authGetUserNumber(pMDNPSession-
>pDbHandle, myUserNameDbHandle);
                if(userNumber != 0)
                  requests[i].authUserNumber = userNumber;
              }
            }
    #endif
            mdnpbrm_binaryCommand(&requests[i], TMWDEFS_NULL, DNPDEFS_FC_SELECT,
              MDNPBRM_AUTO_MODE_OPERATE, 0, DNPDEFS_QUAL_16BIT_INDEX, 1, &CROBInfo[0]);

    #if DNPCNFG_SUPPORT_AUTHENTICATION
            // Since these request structures get reused, clear this flag.
            requests[i].authAggressiveMode = false;
    #endif

          }
        }


    #if MDNPCNFG_SUPPORT_SA_VERSION5 && DNPCNFG_SUPPORT_AUTHKEYUPDATE
        if (!myUseSAv2)
        {
          if (myTimeToActAsAuthority(pMySession))
          {
            myActAsAuthority(&SARequest);
          }
          if (myTimeToActAsAuthority(pMySession2))
```

```
        {
          myActAsAuthority(&SARequest2);
        }

        /* For symmetric update key change method, the encrypted Update Key data
         * from the Authority must be sent to complete the process
         */
        else if (myNowSendUpdateKey)
        {
          mySendUpdateKey(&SARequest, 0);
        }
        else if (myNowSendUpdateKey2)
        {
          mySendUpdateKey(&SARequest2, 1);
        }
      }
#endif


      ///* This shows an example of how to combine multiple objects in one request */
      //TMWTYPES_USHORT size = 512;
      //DNPCHNL_TX_DATA *pUserTxData  = (DNPCHNL_TX_DATA*)dnpchnl_newTxData(pChannel,
      //  pSessions[i], size, pSessions[i]->destAddress);
      //if(pUserTxData  != NULL)
      //{
      //  mdnpbrm_buildRequestHeader(pUserTxData, DNPDEFS_FC_ACTIVATE_CONFIG);

      //  TMWTYPES_UCHAR objectGroup = 70;
      //  TMWTYPES_CHAR *pValue = "firstString";
      //  mdnpbrm_activateConfig(&requests[i], pUserTxData, 0,
(TMWTYPES_UCHAR*)pValue,
      //    (TMWTYPES_USHORT)strlen(pValue), objectGroup);
      //
      //  pValue = "secondString";
      //  mdnpbrm_activateConfig(&requests[i], pUserTxData, 0,
(TMWTYPES_UCHAR*)pValue,
      //    (TMWTYPES_USHORT)strlen(pValue), objectGroup);

      //  objectGroup = 110;
      //  pValue = "firstFileName";
      //  mdnpbrm_activateConfig(&requests[i], pUserTxData, 0,
(TMWTYPES_UCHAR*)pValue,
      //    (TMWTYPES_USHORT)strlen(pValue), objectGroup);

      //  if(!dnpchnl_sendFragment((TMWSESN_TX_DATA*)pUserTxData))
      //  {
      //    /* Failed */
      //    dnpchnl_freeTxData((TMWSESN_TX_DATA*)pUserTxData);
      //  }
      //}

      /* If a second channel/session is open send event polls on it */
      if(openSecondConnection)
      {
        if (myTimeForEvent(pMySession2))
        {
          for (i = 0; i < NUM_SESSIONS; i++)
          {
            printf("Event Poll for Session2 %zd\n", i);
            mdnpbrm_eventPoll(&requests2[i]);
          }
        }
      }


      /* Process any data returned by the Outstation(slave).
       * Note that tmwappl_checkForInput will be called
       * many times in between calls to mdnpbrm_xxx()
       */
      /* If the target layer is configured to be event driven, there
       * is no need for the application to check for received data.
       */
      if (USE_POLLED_MODE)
      {
        tmwappl_checkForInput(pApplContext);
      }

      /* Store any data that is on the data base queue.
       * Typically, this call might be put on a separate
       * thread, or the Async database would be disabled
       * (by setting TMWCNFG_SUPPORT_ASYNCH_DB to TMWDEFS_FALSE).
```

```
         */
        while (tmwdb_storeEntry(TMWDEFS_TRUE))
          ;

        /* Check timers
         */
        tmwpltmr_checkTimer();

700     /* Sleep for 50 milliseconds */
        tmwtarg_sleep(50);
      }

     /* Of course, we'll never get here, since we're in an
      * infinite loop above, but just for clarity,
      * close all open sessions and channels
      */
      for (i = 0; i < NUM_SESSIONS; i++)
      {
710     mdnpsesn_closeSession(pSessions[i]);
      }

      dnpchnl_closeChannel(pChannel);
      delete pMySession;
      tmwappl_closeApplication(pApplContext, TMWDEFS_FALSE);

      return (0);
    }

720 #define ActAsAuthorityInterval 120000
    /* myInitMySession
     * Initialize session command interval values
     */
    MY_SESSION_INFO * myInitMySession(void)
    {
      mySessionInfo *p = new mySessionInfo;

      p->integrityInterval = 3600000;     /* Integrity poll once per hour */
      p->eventInterval = 10000;           /* Event poll once every 10 seconds */
730   p->binCmdInterval = 20000;          /* Binary output command once every 20 secs*/
      p->lastEventPoll = tmwtarg_getMSTime();
      p->lastIntegrityPoll = tmwtarg_getMSTime();
      p->lastBinCmd = tmwtarg_getMSTime();

    #if DNPCNFG_SUPPORT_AUTHENTICATION
      if (!noUserAtStartup)
        p->actAsAuthorityInterval = ActAsAuthorityInterval; /* If SAv5, add or edit a
user over DNP every xxx minutes */ /* Use the #define as this will be modified if there
are no users configured */
      else
    #endif
740     p->actAsAuthorityInterval = 5000;                   /* If SAv5, add first user
over DNP in 5 seconds  */
      p->lastActAsAuthority = tmwtarg_getMSTime();

      return p;
    }


    #if DNPCNFG_SUPPORT_AUTHENTICATION
    void myInitSecureAuthentication(MDNPSESN_CONFIG *pSesnConfig)
    {
750 #if MDNPCNFG_SUPPORT_SA_VERSION2

    #if !MDNPCNFG_SUPPORT_SA_VERSION5
      /* If SAv5 is not supported, use SAv2 */
      myUseSAv2 = TMWDEFS_TRUE;
    #endif
    #endif


      /* Enable DNP3 Secure Authentication support */
760   pSesnConfig->authenticationEnabled = TMWDEFS_TRUE;

      /* NOTE: Secure Authentication Version 2 (SAv2) will not function properly without
implementing
       * the functions mdnpdata_authxxx() in mdnpdata.c
       * Optional SAv5 also requires utils/tmwcrypto which uses OpenSSL by default.
       */

      /* Example configuration. Some of these may be the default values,
       * but are shown here as an example of what can be set.
```

```
       */
770    pSesnConfig->authConfig.extraDiags = TMWDEFS_TRUE;
       pSesnConfig->authConfig.aggressiveModeSupport = TMWDEFS_TRUE;

       pSesnConfig->authConfig.maxKeyChangeCount = 1000;

       // 60 seconds is very short for demonstration purposes only.
       pSesnConfig->authConfig.keyChangeInterval = 60000;

       /* For SAv2 configure the same user numbers and update keys for each user
        * number on both master and outstation devices
780     * SAv5 allows User Update Keys to be sent to the outstation over DNP.
        */
     #if MDNPCNFG_SUPPORT_SA_VERSION2
       if(myUseSAv2)
       {
         /* Use Secure Authentication Version 2 */
         pSesnConfig->authConfig.operateInV2Mode = TMWDEFS_TRUE;
         pSesnConfig->authConfig.maxErrorCount = 2;

         /* Configure user numbers */
790      /* Spec says default user number (1) provides a user number for the device or
"any" user */
         pSesnConfig->authConfig.authUsers[0].userNumber = DNPAUTH_DEFAULT_USERNUMBER;

         /* DNP3 Technical Bulletin TB2019-001 indicates that the next version of SA
          * will remove support for support multiple users per Master-Outstation
Association
          * and it is recommended that SAV5 should be used with only a single user per
Association.
          * While the MDNP SAv2 and SAv5 implementations support multiple users this
example
          * will not show that as it would violate that recommendation and not be allowed
in the future.
          */
       }
800  #endif
     }

     #if DNPCNFG_SUPPORT_AUTHENTICATION
     TMWTYPES_BOOL myAddAuthUsers(MDNPSESN *pMDNPSession, TMWTYPES_BOOL operateInV2Mode)
     {

     #if MDNPCNFG_SUPPORT_SA_VERSION2 && TMWCNFG_SUPPORT_CRYPTO
       /* If SAv2 and using optional TMWCRYPTO interface set the crypto database*/
       if (operateInV2Mode)
810    {
         TMWTYPES_USHORT userNumber = 1;

         /* If using simulated database in tmwcrypto, add the User Update Key for the
default user (1) to it.
          * This key should really should be in YOUR crypto database not the simulated
one in tmwcrypto.c.
          * You would not normally need to call tmwcrypto_configSimKey to add it to your
own database.
          */
         /* This used to use pMDNPSession->pDbHandle before v3.30, but mdnpdata.c now
uses the crypto handle as required */
         if (!tmwcrypto_configSimKey(pMDNPSession->dnp.pCryptoHandle,
TMWCRYPTO_USER_UPDATE_KEY, userNumber, defaultUserKey1, 16, TMWDEFS_NULL, 0))
         {
820        /* Failed to add key */
           printf("Failed to add key, exiting program \n");
           return (TMWDEFS_FALSE);
         }

         /* DNP3 Technical Bulletin TB2019-001 indicates that the next version of SA
          * will remove support for support multiple users per Master-Outstation
Association
          * and it is recommended that SAV5 should be used with only a single user per
Association.
          * While the MDNP SAv2 and SAv5 implementations support multiple users this
example
          * will not show that as it would violate that recommendation and not be allowed
in the future.
830       */

         return TMWDEFS_TRUE;
       }
     #endif
     #if MDNPCNFG_SUPPORT_SA_VERSION5
```

```
      /* In SAv5 we no longer use an array of user configuration, instead you can add
one user at a time.
       * The Authority can also tell the master to add a user using a globally unique
user name
       * and instruct the master to send the update key and role (permissions) for that
user over DNP to the outstation.
       *
840    *    See mdnpbrm_authUserStatusChange and mdnpbrm_authUserUpdateKeyChange
       */
      if (!operateInV2Mode)
      {
        TMWTYPES_USHORT userNumber = 1;

        if (!noUserAtStartup)
        {
          /* You are allowed to start with NO user configured on the master and
outstation and add the user at runtime.
           * In that case you don't have to configure a user here
850        */

           /* If using simulated database in tmwcrypto, add the User Update Key for the
default user (1) to it.
            * This key should really should be in YOUR crypto database not the simulated
one in tmwcrypto.c.
            * You would not normally call tmwcrypto_configSimKey to add it to your own
database.
            */
          if (!tmwcrypto_configSimKey(pMDNPSession->dnp.pCryptoHandle,
TMWCRYPTO_USER_UPDATE_KEY, userNumber, defaultUserKey1, 16, TMWDEFS_NULL, 0))
          {
            /* Failed to add key */
            printf("Failed to add key, exiting program \n");
860         return (TMWDEFS_FALSE);
          }

          /* For SAv5 add the user to the mdnp library */
          mdnpsesn_addAuthUser((TMWSESN*)pMDNPSession, userNumber);

          /* DNP3 Technical Bulletin TB2019-001 indicates that the next version of SA
           * will remove support for support multiple users per Master-Outstation
Association
           * and it is recommended that SAV5 should be used with only a single user per
Association.
           * While the MDNP SAv2 and SAv5 implementations support multiple users this
example
870        * will not show that as it would violate that recommendation and not be
allowed in the future.
            */
        }

        /* Configure other values in YOUR crypto database to allow remote user key and
role update from Master.*/

        /* This sample uses the same values that the Test Harness Outstation uses by
default */

        /* Outstation name must be configured in both Master and Outstation.
         * This is already set in the mdnpsim database by default.
880      * OutstationName = "SDNP Outstation";
         */

         /* If using the simulated database in tmwcrypto, add the Authority
Certification Symmetric Key to it.
          * This key really should be in YOUR crypto database not the simulated one in
tmwcrypto.c.
          * You would not normally call tmwcrypto_configSimKey to add it to your own
database.
          * This key is used by the Central Authority, not the master, but this sample
is acting as the Authority.
          */
        if (!tmwcrypto_configSimKey(pMDNPSession->dnp.pCryptoHandle,
TMWCRYPTO_AUTH_CERT_SYM_KEY, 0, (TMWTYPES_UCHAR *)&authoritySymCertKey, 32, TMWDEFS_NULL,
0))
        {
890       /* Failed to add key */
          printf("Failed to add key, exiting program \n");
          return (TMWDEFS_FALSE);
        }

        /* If using simulated database in tmwcrypto, configre Outstation Public Key when
Asymmetric Key Update is supported
```

```
          * This really should be in YOUR crypto database not the simulated one.
          * You would not normally call tmwcrypto_configSimKey to add it to your own
database.
          */
         const TMWTYPES_CHAR *pPublicKey = "TMWTestOSRsa2048PubKey.pem";
 900     TMWTYPES_USHORT keyLength = (TMWTYPES_USHORT)strlen(pPublicKey);
         if (!tmwcrypto_configSimKey(pMDNPSession->dnp.pCryptoHandle,
TMWCRYPTO_OS_ASYM_PUB_KEY, 0, (TMWTYPES_UCHAR *)pPublicKey, keyLength, TMWDEFS_NULL, 0))
         {
           /* Failed to add key */
           printf("Failed to add key, exiting program \n");
           return (TMWDEFS_FALSE);
         }

         return TMWDEFS_TRUE;
       }
 910 #endif
       return TMWDEFS_FALSE;
     }
     #endif


     #if MDNPCNFG_SUPPORT_SA_VERSION5 && DNPCNFG_SUPPORT_AUTHKEYUPDATE
     #define BUFFER_LENGTH 2048
     /* In SAv5 a trusted third party, known as the Authority will command the
      * master to add a user with a globally unique name and to send requests over the
DNP
 920  * protocol to add that user and his role on the outstation also.
      *
      * This function will act as that authority to demonstrate how your application
should
      * request the master SCl to take action.
      */
     TMWTYPES_BOOL     firstKeyUpdate = TMWDEFS_TRUE;
     TMWTYPES_ULONG    globalStatusChangeSequenceNumber = 0;
      void myActAsAuthority(MDNPBRM_REQ_DESC *pSARequest)
      {
        MDNPSESN         *pMDNPSession;
 930    const char       *pUserPublicKeyFile;
        const char       *pUserPrivateKeyFile;
        const char       *pAuthorityPrivateKeyFile;
        const char       *AsymPrvKeyPassword;
        const char       *pUserName;
        TMWTYPES_ULONG    sequence;
        TMWTYPES_USHORT   plainTextLength;
        TMWTYPES_USHORT   userNameLength;
        TMWTYPES_USHORT   certDataLength;
        TMWTYPES_USHORT   userRole;
 940    TMWTYPES_USHORT   userRoleExpiryInterval;
        TMWTYPES_BOOL     sendUpdateKey;
        TMWTYPES_UCHAR    algorithm;
        TMWTYPES_UCHAR    operation;
        TMWTYPES_UCHAR    keyChangeMethod;
        TMWTYPES_UCHAR    plainTextArray[BUFFER_LENGTH];
        TMWTYPES_UCHAR    certData[BUFFER_LENGTH];

        pMDNPSession = (MDNPSESN*)pSARequest->pSession;
        sequence = ++globalStatusChangeSequenceNumber;
 950    algorithm = 0;

        /* As an example, add and modify user common periodically */
        /* Multiple users per association is now being discouraged...*/
        pUserName = "Common";

        if(firstKeyUpdate)
        {
          operation = DNPAUTH_USER_STATUS_ADD;
          firstKeyUpdate = TMWDEFS_FALSE;
 960    }
        else
        {
          operation = DNPAUTH_USER_STATUS_CHANGE;
        }

        /* Certification data SHOULD be provided by the Central Authority,
         * but for this example act as the authority.
         */
        sendUpdateKey = TMWDEFS_TRUE;
 970

        keyChangeMethod = DNPAUTH_KEYCH_SYMAES256_SHA256;
```

```
        /* To use asymmetric key change method you need to configure the correct type and
size asymmetric keys
         * incuding the Authority, Outstation and User Keys.
         * Outstation must also be configured properly for this key change method.
         * keyChangeMethod = DNPAUTH_KEYCH_ASYM_RSA2048_SHA256;
         */

 980
        userRole = DNPAUTH_USER_ROLE_SINGLEUSER;
        userRoleExpiryInterval = 100;

        plainTextArray[0] = operation & 0xff;

        plainTextArray[1] = sequence & 0xff;
        plainTextArray[2] = (sequence>>8) & 0xff;
        plainTextArray[3] = (sequence>>16) & 0xff;
        plainTextArray[4] = (sequence>>24) & 0xff;
 990
        plainTextLength = 5;

        plainTextArray[plainTextLength++] = userRole & 0xff;
        plainTextArray[plainTextLength++] = (userRole>>8) & 0xff;

        plainTextArray[plainTextLength++] = userRoleExpiryInterval & 0xff;
        plainTextArray[plainTextLength++] = (userRoleExpiryInterval>>8) & 0xff;

        userNameLength = (TMWTYPES_USHORT)strlen((const char *)pUserName);
1000    plainTextArray[plainTextLength++] = userNameLength & 0xff;
        plainTextArray[plainTextLength++] = (userNameLength>>8) & 0xff;

        TMWCRYPTO_KEY userPubKey;
        if (keyChangeMethod < 64)
        {
          /* For symmetric methods the user name is next.  */
          memcpy(&plainTextArray[plainTextLength], pUserName, userNameLength);
          plainTextLength += userNameLength;
        }
1010    else
        {
          /* Asymmetric method */
          /* For asymmetric methods it would be User Public Key Length, User Name, and
User Public Key, */

          /* If using simulated database in tmwcrypto, configure the User Asymmetric keys
for this method
           * These keys should really should be in YOUR crypto database not the simulated
one in tmwcrypto.c.
           * You would not normally need to call tmwcrypto_configSimKey to add it to your
own database.
           */
          switch (keyChangeMethod)
1020      {
          case DNPAUTH_KEYCH_ASYM_RSA1024_SHA1:
            pAuthorityPrivateKeyFile = "TMWTestAuthorityDsa1024PrvKey.pem";
            pUserPublicKeyFile = "TMWTestUserDsa1024PubKey.pem";
            pUserPrivateKeyFile = "TMWTestUserDsa1024PrvKey.pem";
            AsymPrvKeyPassword = "triangle";
            algorithm = TMWCRYPTO_ALG_SIGN_DSA_1024_SHA1;
            break;

          case DNPAUTH_KEYCH_ASYM_RSA2048_SHA256:
1030        pAuthorityPrivateKeyFile = "TMWTestAuthorityDsa2048PrvKey.pem";
            pUserPublicKeyFile = "TMWTestUserDsa2048PubKey.pem";
            pUserPrivateKeyFile = "TMWTestUserDsa2048PrvKey.pem";
            AsymPrvKeyPassword = "triangle";
            algorithm = TMWCRYPTO_ALG_SIGN_DSA_2048_SHA256;
            break;

          case DNPAUTH_KEYCH_ASYM_RSA3072_SHA256:
            pAuthorityPrivateKeyFile = "TMWTestAuthorityDsa3072PrvKey.pem";
            pUserPublicKeyFile = "TMWTestUserDsa3072PubKey.pem";
1040        pUserPrivateKeyFile = "TMWTestUserDsa3072PrvKey.pem";
            AsymPrvKeyPassword = "triangle";
            algorithm = TMWCRYPTO_ALG_SIGN_DSA_3072_SHA256;
            break;

            // New key change methods from TB2016-002
          case DNPAUTH_KEYCH_ASYM_RSA1024_RSA_SHA1:
            pAuthorityPrivateKeyFile = "TMWTestAuthorityRsa1024PrvKey.pem";
            pUserPublicKeyFile = "TMWTestUserRsa1024PubKey.pem";
```

```
            pUserPrivateKeyFile = "TMWTestUserRsa1024PrvKey.pem";
1050        AsymPrvKeyPassword = "triangle";
            algorithm = TMWCRYPTO_ALG_SIGN_RSA_1024_SHA1;
            break;

          case DNPAUTH_KEYCH_ASYM_RSA2048_RSA_SHA256:
            pAuthorityPrivateKeyFile = "TMWTestAuthorityRsa2048PrvKey.pem";
            pUserPublicKeyFile = "TMWTestUserRsa2048PubKey.pem";
            pUserPrivateKeyFile = "TMWTestUserRsa2048PrvKey.pem";
            AsymPrvKeyPassword = "triangle";
            algorithm = TMWCRYPTO_ALG_SIGN_RSA_2048_SHA256;
1060        break;

          case DNPAUTH_KEYCH_ASYM_RSA3072_RSA_SHA256:
            pAuthorityPrivateKeyFile = "TMWTestAuthorityRsa3072PrvKey.pem";
            pUserPublicKeyFile = "TMWTestUserRsa3072PubKey.pem";
            pUserPrivateKeyFile = "TMWTestUserRsa3072PrvKey.pem";
            AsymPrvKeyPassword = "triangle";
            algorithm = TMWCRYPTO_ALG_SIGN_RSA_3072_SHA256;
            break;

1070      default:
            printf("FAILED, keys not configured for this key change method\n");
            return;
          }

          /* If using simulated database in tmwcrypto, configure Authority Private Key
known only by the Authority
           * This key is used by the Central Authority, not the master, but this sample
is acting as the Authority.
           * Since this master is simulating the central authority it needs to know this
key.
           * This really should be in YOUR crypto database not the simulated one.
           * You would not normally need to call tmwcrypto_configSimKey to add it to your
own database.
1080        */
          if (!tmwcrypto_configSimKey(pMDNPSession->dnp.pCryptoHandle,
TMWCRYPTO_AUTH_ASYM_PRV_KEY, 0, (TMWTYPES_UCHAR *)pAuthorityPrivateKeyFile,
(TMWTYPES_USHORT)strlen(pAuthorityPrivateKeyFile), (TMWTYPES_UCHAR *)"triangle", 8))
          {
            /* Failed to add key */
            printf("Failed to add key, exiting program \n");
            return;
          }

           /* Configure some user information including user public keys
            * This should really be updating your real database, not the simulated and
simulated crypto databases.
1090        */
           myUserNameDbHandle = mdnpsim_authConfigUser(pMDNPSession->pDbHandle,
(TMWTYPES_CHAR*)pUserName, userNameLength,
             sequence, keyChangeMethod, operation, userRole, userRoleExpiryInterval,
(TMWTYPES_UCHAR*)"No Update Key Yet", 1, (TMWTYPES_UCHAR*)"No Certification Data Yet",
1);

           tmwcrypto_configSimKey(pMDNPSession->dnp.pCryptoHandle,
TMWCRYPTO_USER_ASYM_PUB_KEY, 1, (TMWTYPES_UCHAR *)pUserPublicKeyFile,
(TMWTYPES_USHORT)strlen(pUserPublicKeyFile), TMWDEFS_NULL, 0);

           tmwcrypto_configSimKey(pMDNPSession->dnp.pCryptoHandle,
TMWCRYPTO_USER_ASYM_PRV_KEY, 1, (TMWTYPES_UCHAR *)pUserPrivateKeyFile,
(TMWTYPES_USHORT)strlen(pUserPrivateKeyFile), (TMWTYPES_UCHAR *)AsymPrvKeyPassword, 8);


          /* Get user public key from key file.
1100       * From Jan 6 2011 DNP3 Tech SA Teleconference TC11-01-06-SA Minutes.
           * The public key should be an octet by octet copy of the
           * SubjectPublicKeyInfo field from the X509 certificate (RFC 5280).
           *  SubjectPublicKeyInfo  ::=  SEQUENCE  {
           *    algorithm            AlgorithmIdentifier,
           *    subjectPublicKey     BIT STRING  }
           * There is an example in rfc5280 page 141
           * Get the encoded key data
          */
           TMWTYPES_USHORT userNumber = 1;
1110      if (tmwcrypto_getKey(pMDNPSession->dnp.pCryptoHandle,
TMWCRYPTO_USER_ASYM_PUB_KEY, (void *)userNumber, &userPubKey))
          {
            /* get index and leave 2 bytes for length of pub key data */
            TMWTYPES_USHORT lengthIndex = plainTextLength;
            plainTextLength += 2;
```

```
            // User name now
            memcpy(&plainTextArray[plainTextLength], pUserName, userNameLength);
            plainTextLength += userNameLength;

1120        /* Indicate how much room there is left in buffer before calling getKeyData
*/
            TMWTYPES_USHORT userPublicKeyLength = BUFFER_LENGTH - plainTextLength;
            if (!tmwcrypto_getKeyData(pMDNPSession->dnp.pCryptoHandle, &userPubKey, (void
*)userNumber,
              &plainTextArray[plainTextLength], &userPublicKeyLength))
            {
              printf("FAILED, get user public key data \n");
              return;
            }
            plainTextLength += userPublicKeyLength;

1130        plainTextArray[lengthIndex++] = userPublicKeyLength & 0xff;
            plainTextArray[lengthIndex] = (userPublicKeyLength >> 8) & 0xff;
          }
        }

        if (keyChangeMethod < 64)
        {
          /* Method is symmetric, so the the pre-shared (between AUTHORITY and
outstation), the
           * Authority Certification Key will be used to generate the MAC for this data.
           * Normally, master would NOT know this key.
1140       */
          certDataLength = 0;
          TMWCRYPTO_KEY certKey;
          tmwcrypto_getKey(pMDNPSession->dnp.pCryptoHandle, TMWCRYPTO_AUTH_CERT_SYM_KEY,
0, &certKey);
          if (!tmwcrypto_MACValue(pMDNPSession->dnp.pCryptoHandle,
TMWCRYPTO_ALG_MAC_SHA256, &certKey, 32, plainTextArray, plainTextLength, certData,
&certDataLength))
          {
            printf("FAILED, get Authority Cert Key \n");
            return;
          }
        }
1150    else
        {
          /* asymmetric */
          TMWCRYPTO_KEY authPrvKey;

           // Sign this with authority private key.
           tmwcrypto_getKey(pMDNPSession->dnp.pCryptoHandle, TMWCRYPTO_AUTH_ASYM_PRV_KEY,
0, &authPrvKey);
           memcpy(authPrvKey.password, "triangle", strlen("triangle"));
           authPrvKey.passwordLength = (TMWTYPES_USHORT)strlen("triangle");
           if (!tmwcrypto_genDigitalSignature(pMDNPSession->dnp.pCryptoHandle, algorithm,
&authPrvKey, plainTextArray, plainTextLength, certData, &certDataLength))
1160       {
            printf("FAILED, Generate Digital Signature\n");
            return;
          }
      }
       /*     * Call mdnpbrm_authUserStatusChange function to create or update an
existing user on the outstation.
         *
         * Your application should not use the simulated database. We will use it here so
this sample can run with the SCL as shipped.
         *
         * You should put the user data in Your database, and when
mdnpdata_authGetUserName(),
1170     * mdnpdata_authGetChangeUserData() etc are called you should use the
userNameDbHandle
         * you passed to mdnpbrm_authUserStatusChange to look up the user information.
         * When mdnpdata_authStoreUpdKeyChangeReply() is called the new user number
returned from the outstation
         * should be added to your database.
         *
         * If the keyChangeMethod is symmetric, the g120v12 will return the user number.
The master should then ask the
         * DNP Authority to provide the encrypted User Update Key data to be sent in a
g120v13/v15 request. This can then be sent
         * by calling mdnpbrm_authUserSendSymUpdateKey()
         * If the keyChangeMethod is asymmetric, the master does not have to ask the DNP
Authority for the data and the MDNP
         * library will authomatically send the g120v13/v14 request to send the User
```

```
Update Key
1180      */
          TMWCRYPTO_KEY key;
          TMWTYPES_USHORT requestedKeyLength = 32;

          if((keyChangeMethod == DNPAUTH_KEYCH_SYMAES128_SHA1) || (keyChangeMethod ==
DNPAUTH_KEYCH_ASYM_RSA1024_SHA1))
            requestedKeyLength = 16;

          tmwcrypto_generateNewKey(pMDNPSession->dnp.pCryptoHandle,
TMWCRYPTO_USER_UPDATE_KEY, requestedKeyLength, &key);

          /* Add the User Update Key and Certification data to the database.
1190       * This should really be updating your real database, not the simulated one
           */
          myUserNameDbHandle = mdnpsim_authConfigUser(pMDNPSession->pDbHandle,
(TMWTYPES_CHAR *)pUserName, userNameLength,
            sequence, keyChangeMethod, operation, userRole, userRoleExpiryInterval,
key.value, key.length, certData, certDataLength);

          if(keyChangeMethod < 67)
          {
            /* If Key Change Method is symmetric, register a callback function to complete
the remote update process */
            pSARequest->pUserCallback = my_brmSACallbackFcn;
          }
1200

          if(mdnpbrm_authUserStatusChange(pSARequest, myUserNameDbHandle, operation,
sendUpdateKey) == TMWDEFS_NULL)
          {
            printf("FAILED, mdnpbrm_authUserStatusChange() returned failure \n");
            return;
          }
        }

       /* This function will have the MDNP library send the encrypted User Update Key
1210    * that was received from the Authority and put in the database. In this sample
        * the MDNP library will call mdnpdata_authGetUpdateKeyData() to get this encrypted
data
        * but if it has not been implemented the MDNP library will simulate the Authority
        * and encrypt the data using a configured Authority Certification (Symmetric) Key
        */
       void mySendUpdateKey(MDNPBRM_REQ_DESC *pSARequest, int i)
       {
          pSARequest->pUserCallback = TMWDEFS_NULL;
          mdnpbrm_authUserSendSymUpdateKey(pSARequest, myUserNameDbHandle);
          if(i==0)
1220        myNowSendUpdateKey = false;
          else
            myNowSendUpdateKey2 = false;

       }

      /* myTimeToActAsAuthority
       * See if it is time to act as the Authority to tell the master to add or change a
user
       * to the master and outstation. If so, return TMWDEFS_TRUE and update the
       * "time of last poll" field to be the current time.
1230   */
      TMWTYPES_BOOL myTimeToActAsAuthority(MY_SESSION_INFO *pMySession)
      {
        TMWTYPES_MILLISECONDS myTime;
        TMWTYPES_BOOL returnVal;
        if (pMySession == NULL)
          return TMWDEFS_FALSE;

        myTime = tmwtarg_getMSTime();
        if (myTime >=(pMySession->lastActAsAuthority + pMySession-
>actAsAuthorityInterval))
1240    {
          pMySession->lastActAsAuthority = myTime;
          /* This permits a quick first time if no users were configured, then a longer
period */
          pMySession->actAsAuthorityInterval = ActAsAuthorityInterval;
          returnVal = TMWDEFS_TRUE;
        }
        else
        {
          returnVal = TMWDEFS_FALSE;
        }
```

```
  return (returnVal);
}
#endif
#endif

/* myTimeForIntegrity
 * See if it is time to conduct an Integrity Poll. If so, return
 * TMWDEFS_TRUE and update the "time of last poll" field to
 * be the current time.
 *
 * Note that this simplified example does not take into account
 * the fact that the lastIntegrityPoll field could roll over.
 */
TMWTYPES_BOOL myTimeForIntegrity(MY_SESSION_INFO *pMySession)
{
  TMWTYPES_MILLISECONDS myTime;
  TMWTYPES_BOOL returnVal;

  myTime = tmwtarg_getMSTime();
  if (myTime >=(pMySession->lastIntegrityPoll + pMySession->integrityInterval))
  {
    pMySession->lastIntegrityPoll = myTime;
    returnVal = TMWDEFS_TRUE;
  }
  else
  {
    returnVal = TMWDEFS_FALSE;
  }
  return (returnVal);
}

/* myTimeForEvent
 * See if it is time to conduct an Event Poll. If so, return
 * TMWDEFS_TRUE and update the "time of last poll" field to
 * be the current time.
 *
 * Note that this simplified example does not take into account
 * the fact that the lastEventPoll field could roll over.
 */
TMWTYPES_BOOL myTimeForEvent(MY_SESSION_INFO *pMySession)
{
  TMWTYPES_MILLISECONDS myTime;
  TMWTYPES_BOOL returnVal = TMWDEFS_FALSE;

  myTime = tmwtarg_getMSTime();
  if (pMySession != TMWDEFS_NULL)
  {
    if (myTime > (pMySession->lastEventPoll + pMySession->eventInterval))
    {
      pMySession->lastEventPoll = myTime;
      returnVal = TMWDEFS_TRUE;
    }
  }
  return (returnVal);
}

/* myTimeForBinCmd
 * See if it is time to do a binary output. If so, return
 * TMWDEFS_TRUE and update the "time of last poll" field to
 * be the current time.
 *
 * Note that this simplified example does not take into account
 * the fact that the lastBinCmd field could roll over.
 */
TMWTYPES_BOOL myTimeForBinCmd(MY_SESSION_INFO *pMySession)
{
  TMWTYPES_MILLISECONDS myTime;
  TMWTYPES_BOOL returnVal;

  myTime = tmwtarg_getMSTime();
  if (myTime > (pMySession->lastBinCmd + pMySession->binCmdInterval))
  {
    pMySession->lastBinCmd = myTime;
    returnVal = TMWDEFS_TRUE;
  }
  else
  {
    returnVal = TMWDEFS_FALSE;
  }
  return (returnVal);
}
```

```
     /* The following functions are called by the SCL */

     /* my_brmCallbackFcn
      * Example user request callback function
      * This function will be called for every response fragment rcvd for the request.
      * When the request completes pResponse->last will be TMWDEFS_TRUE.
      * pResponse->status will indicate if request was successful, timed out or
      * otherwise failed.
1340  */
     void my_brmCallbackFcn(void *pUserCallbackParam, DNPCHNL_RESPONSE_INFO *pResponse)
     {
       size_t index = (size_t)pUserCallbackParam;

       printf("in Callback Function, Session #%zd; status is %d\n", index, pResponse-
>status);
     }

     #if MDNPCNFG_SUPPORT_SA_VERSION5 && DNPCNFG_SUPPORT_AUTHKEYUPDATE
     /* my_brmSACallback
1350  * Example SA Auth User Status request callback function
      * This function will be called when the Auth User Status Change receives responses
      * including the g120v12 from the Outstation
      */
     void my_brmSACallbackFcn(void *pUserCallbackParam, DNPCHNL_RESPONSE_INFO *pResponse)
     {
       TMWTARG_UNUSED_PARAM(pUserCallbackParam);
       printf("in my_brmSACallback Callback Function, status is %d\n", pResponse-
>status);
       if (pResponse->last && (pResponse->status == DNPCHNL_RESP_STATUS_SUCCESS))
       {
1360     if(pUserCallbackParam == 0)
           myNowSendUpdateKey = true;
         else
           myNowSendUpdateKey2 = true;

       }
     }
     #endif

     #if TMWCNFG_SUPPORT_DIAG
1370 /* Simple diagnostic output function, registered with the Source Code Library */
     void myPutDiagString(const TMWDIAG_ANLZ_ID *pAnlzId,const TMWTYPES_CHAR *pString)
     {
       TMWDIAG_ID id = pAnlzId->sourceId;

       /* Commented out to turn off verbose diagnostics */
       /* Print all diagnostics */
       /* printf((char *)pString);
        * return;
        */
1380
       /* Display errors, application level, user level and secure authentication
diagnostics
        * Add other layers such as target as desired
        */
       if((TMWDIAG_ID_ERROR & id)
         ||(TMWDIAG_ID_APPL & id)
         ||(TMWDIAG_ID_USER & id)
         ||(TMWDIAG_ID_SECURITY_DATA & id)
         ||(TMWDIAG_ID_SECURITY_HDRS & id)
         /*||(TMWDIAG_ID_TARGET & id)*/
1390     )
       {
         printf( "%s", (char *)pString);

         return;
       }
     }
     #endif
     #endif

1400
```

3 Configuration Files

3.1 Default Functionality

The MDNP SCL as shipped has certain default functionality compiled in with additional functionality that can be enabled. This includes what Object Groups and Variations are supported, maximum sizes and other functionality and behavior. This default configuration and functionality is NOT intended to be used by all implementations. After careful consideration of your requirements, you should modify the configuration files to compile in only what is needed.

Disabling data types and functionality that are not required reduces memory consumption, the number of database functions that must be implemented, as well as the testing effort required. Removing unnecessary or unintended functionality is considered a good development practice as it limits the attack surface reducing the exposure to potential vulnerabilities. The capabilities and configuration of your application should then be documented in the Device Profile as required for DNP3.

In SCL Version 3.29.0 the default configuration has been reduced to mostly DNP Level 3 functionality. The previous capabilities are still present in the SCL, but may need to be compiled in by modifying certain #defines in the configuration files.

Here is a list of what was enabled in the SCL by default that is now compiled out.

MDNPDATA_CNFG_LEVEL_TMW Listed here	
File Transfer	Object Group 70
Activate Configuration response	Object Group 91
Octet String and Events	Object Groups 110 and 111
Virtual Terminal and Events	Object Groups 112 and 113

The .NET SCL had additional functionality compiled in by default beyond the ANSCI C SCL. The .NET SCL will now provide the same default functionality. Here is a list of things in addition to the above that was previously enabled in the .NET SCL but is no longer enabled by default.

Device Attributes	Object Group 0
Binary Output Events	Object Group 11
CROB Allow count to be specified	Object Group 12
Binary Output Command Events	Object Group 13
Frozen Analog Inputs and Events	Object Group 31 and 33
Analog Output Events	Object Group 42
Analog Output Command Events	Object Group 43
Indexed Abs. Time & Long Interval	Object Group 50 Var 4
Data Sets	Object Groups 85-88
Extended Strings and Events	Object Groups 114 and 115

The following was also disabled or reduced in size in the .NET SCL to match the C SCL default functionality.

DNPCNFG_MAX_TX_FRAGMENT_LENGTH	Was (8*1024) Now 2048
DNPCNFG_MAX_RX_FRAGMENT_LENGTH	Was (8*1024) Now 2048
DNPCNFG_MAX_DATASET_DESCR_ELEMS	Was 128 Now 32
DNPCNFG_MAX_DATASET_CTRLS	Was 16 Now 8
DNPCNFG_SUPPORT_BINCONFIG	Now FALSE
MDNPDATA_SUPPORT_OBJ120_V8	Now FALSE

The following subsections include the header files for configuration files that may need to be modified.

3.2 TMW Type Definitions

'tmwtypes.h' contains the definitions for data types that are used throughout the SCL. These should be reviewed to make sure the definitions are compatible with the target platform. These do not typically require modification.

```
/*****************************************************************************/
/* Triangle MicroWorks, Inc.                          Copyright (c) 1997-2022 */
/*****************************************************************************/
/*                                                                           */
/* This file is the property of:                                             */
/*                                                                           */
/*                        Triangle MicroWorks, Inc.                          */
/*                      Raleigh, North Carolina USA                          */
/*                       www.TriangleMicroWorks.com                          */
/*                          (919) 870-6615                                   */
/*                                                                           */
/* This Source Code and the associated Documentation contain proprietary     */
/* information of Triangle MicroWorks, Inc. and may not be copied or         */
/* distributed in any form without the written permission of Triangle        */
/* MicroWorks, Inc.  Copies of the source code may be made only for backup   */
/* purposes.                                                                 */
/*                                                                           */
/* Your License agreement may limit the installation of this source code to  */
/* specific products.  Before installing this source code on a new           */
/* application, check your license agreement to ensure it allows use on the  */
/* product in question.  Contact Triangle MicroWorks for information about   */
/* extending the number of products that may use this source code library or */
/* obtaining the newest revision.                                            */
/*                                                                           */
/*****************************************************************************/

/* file: tmwtypes.h
 * description: Triangle MicroWorks Source Code definitions.
 */
#ifndef TMWTYPES_DEFINED
#define TMWTYPES_DEFINED

#ifndef _WIN32
/* If this file with standard integer types is available use it */
#include <stdint.h>
#endif

#include "tmwscl/utils/tmwdefs.h"
#include "tmwscl/utils/tmwcnfg.h"

/* Type Definitions */

#ifdef __cplusplus
typedef bool           TMWTYPES_BOOL;   /*  false to  true                */
#else
typedef unsigned char  TMWTYPES_BOOL;   /*  TMWDEFS_FALSE to  TMWDEFS_TRUE */
#endif
```

```
typedef unsigned char  TMWTYPES_BYTE;   /*              0 to           255 */
typedef          char  TMWTYPES_CHAR;   /*           -128 to           127 */
typedef unsigned char  TMWTYPES_UCHAR;  /*              0 to           255 */

/* short must be 16 bits and long must be 32 bits.
 * Use stdint.h types if they are available.
 */
#ifdef _WIN32
typedef          short TMWTYPES_SHORT;  /*        -32,768 to        32,767 */
typedef unsigned short TMWTYPES_USHORT; /*              0 to        65,535 */
typedef          int TMWTYPES_LONG;    /* -2,147,483,648 to 2,147,483,647 */
typedef unsigned int TMWTYPES_ULONG;   /*              0 to 4,294,967,295 */

/* Note: The following 2 types are not used by the SCL and can be removed  */
/*       if there is no compiler support for 64-bit types.                 */
typedef      long long TMWTYPES_INT64;  /* -9,223,372,036,854,775,808 to    */
                                        /* 9,223,372,036,854,775,807        */
typedef unsigned long long
                       TMWTYPES_UINT64; /* 0 to 18,446,744,073,709,551,615 */
#else
typedef        int16_t TMWTYPES_SHORT;  /*        -32,768 to        32,767 */
typedef       uint16_t TMWTYPES_USHORT; /*              0 to        65,535 */
typedef        int32_t TMWTYPES_LONG;   /* -2,147,483,648 to 2,147,483,647 */
typedef       uint32_t TMWTYPES_ULONG;  /*              0 to 4,294,967,295 */

/* Note: The following 2 types are not used by the SCL and can be removed  */
/*       if there is no compiler support for 64-bit types.                 */
typedef        int64_t TMWTYPES_INT64;  /* -9,223,372,036,854,775,808 to    */
                                        /* 9,223,372,036,854,775,807        */
typedef       uint64_t TMWTYPES_UINT64; /* 0 to 18,446,744,073,709,551,615 */
#endif

typedef          int   TMWTYPES_INT;    /* machine dependant                */
typedef unsigned int   TMWTYPES_UINT;   /* machine dependant                */
typedef          float TMWTYPES_SFLOAT; /*  -3.4 * 10(38) to +3.4 * 10(38)  */
                                        /*     32-bit short floating point  */
                                        /*     number -- IEEE Standard 754  */
                                        /*       fraction = UI23[1..23]     */
                                        /*       exponent = UI8 [24..31]    */
                                        /*       sign     = BS1[32]         */
typedef         double TMWTYPES_DOUBLE; /* 2.2250738585072014 10(-308) to   */
                                        /*     1.7976931348623158 10(+308)  */
                                        /*  64-bit double precision number  */
                                        /*     number -- IEEE Standard 754  */
                                        /*       fraction = UI52[1..52]     */
                                        /*       exponent = UI11 [53..63]   */
                                        /*       sign     = BS1[64]         */

/* This enumeration type is used to identify the protocol and mode of
 * operation (master or slave) when multiple Triangle MicroWorks,
 * Inc. Source Code Libraries are combined.  A specific example of such
 * a combination is the Triangle MicroWorks, Inc. Gateway Source Code
 * Library and Executable.  This product uses multiple Source Code
 * Libraries to create a data concentrator and/or protocol translator.
 */
typedef enum TMWTYPES_PROTOCOL_ENUM
{
  TMWTYPES_PROTOCOL_101 = 0,
  TMWTYPES_PROTOCOL_102,
  TMWTYPES_PROTOCOL_103,
  TMWTYPES_PROTOCOL_104,
  TMWTYPES_PROTOCOL_DNP,
  TMWTYPES_PROTOCOL_MB,
  TMWTYPES_PROTOCOL_I61850,
  TMWTYPES_PROTOCOL_NUM_PROTOCOLS
} TMWTYPES_PROTOCOL;

/* Specify whether this is a master or slave
 */
typedef enum TMWTYPES_SESSION_TYPE_ENUM
{
  TMWTYPES_SESSION_TYPE_MASTER = 0,
  TMWTYPES_SESSION_TYPE_SLAVE,
  TMWTYPES_SESSION_TYPE_MONITOR,
  TMWTYPES_SESSION_TYPE_NUM_TYPES
} TMWTYPES_SESSION_TYPE;

/* The following definitions are used to provide easy representation of
 * time in units of milliseconds. Using TMWTYPES_ULONG to store milliseconds
 * allows a range of approximately 48 days.
```

```
 */
typedef TMWTYPES_ULONG TMWTYPES_MILLISECONDS;

/* This structure is used primarily by DNP Source Code Libraries to
 * specify the number of milliseconds since January 1, 1970.  Target
 * Application specific functions must convert between native date/time
 * structures and this structure.  Utility conversion routines are
 * provided in TMWdtime.c and DNPdtime.c.
 */
typedef struct TMWTypesMsSince70
{
  TMWTYPES_ULONG  mostSignificant;
  TMWTYPES_USHORT leastSignificant;
} TMWTYPES_MS_SINCE_70;


#if TMW_PRIVATE
#include "tmwscl/utils/tmwdtime.h"
#endif

/* Define data structure used to hold analog data point values
 * CURRENTLY ONLY USED BY DNP
 */
typedef enum TMWTYPES_ANALOG_TYPE_ENUM
{
  TMWTYPES_ANALOG_TYPE_SHORT = 0,
  TMWTYPES_ANALOG_TYPE_USHORT,
  TMWTYPES_ANALOG_TYPE_LONG,
  TMWTYPES_ANALOG_TYPE_ULONG,
  TMWTYPES_ANALOG_TYPE_CHAR,
  TMWTYPES_ANALOG_TYPE_UCHAR
#if TMWCNFG_SUPPORT_FLOAT
  ,TMWTYPES_ANALOG_TYPE_SFLOAT
  ,TMWTYPES_ANALOG_TYPE_SCALED
#endif
#if TMWCNFG_SUPPORT_DOUBLE
  ,TMWTYPES_ANALOG_TYPE_DOUBLE
  ,TMWTYPES_ANALOG_TYPE_DSCALED
#endif
#if TMW_PRIVATE
  ,TMWTYPES_ANALOG_TYPE_STRING
  ,TMWTYPES_ANALOG_TYPE_TIME
#endif
} TMWTYPES_ANALOG_TYPE;

/* This allows for a scaled integer representation of floating point value
 * as well as the floating point value itself. This also allows the database
 * to determine how the floating point value would be rounded if an integer
 * value is to be sent in a response. fval will be used if a floating point
 * value is required, lval will be used if an integer value is required.
 */
typedef struct TMWScaledFloat {
  TMWTYPES_SFLOAT fval;
  TMWTYPES_LONG   lval;
} TMWTYPES_SCALED_FLOAT;

/* This allows for a scaled integer representation of floating point value
 * as well as the floating point value itself. This also allows the database
 * to determine how the floating point value would be rounded if an integer
 * value is to be sent in a response. dval will be used if a floating point
 * value is required, lval will be used if an integer value is required.
 */
typedef struct TMWScaledDouble {
  TMWTYPES_DOUBLE dval;
  TMWTYPES_LONG   lval;
} TMWTYPES_SCALED_DOUBLE;

typedef struct TMWAnalogValue {
  TMWTYPES_ANALOG_TYPE type;
  union _valueUnion
  {
    TMWTYPES_SHORT  sval;
    TMWTYPES_USHORT usval;
    TMWTYPES_LONG   lval;
    TMWTYPES_ULONG  ulval;
    TMWTYPES_CHAR   cval;
    TMWTYPES_UCHAR  ucval;
#if TMWCNFG_SUPPORT_FLOAT
    TMWTYPES_SFLOAT fval;
    TMWTYPES_SCALED_FLOAT scaled;
#endif
```

```
#if TMWCNFG_SUPPORT_DOUBLE
    TMWTYPES_DOUBLE dval;
    TMWTYPES_SCALED_DOUBLE dscaled;
#endif
#if TMW_PRIVATE
    TMWTYPES_UCHAR *pString;
    TMWDTIME timeVal;
#endif
  } value;

#if TMW_PRIVATE
  /*   * The following members are used by an external application
   * to store overflow and underflow state for a type conversion
   */
  TMWTYPES_BOOL m_bOverFlow;
  TMWTYPES_BOOL m_bUnderFlow;
#endif /* TMW_PRIVATE */

} TMWTYPES_ANALOG_VALUE;

/* Define a generic callback function used throughout TMW code
 */
typedef void (*TMWTYPES_CALLBACK_FUNC)(void *pCallbackParam);

#endif /* TMWTYPES_DEFINED */
```

3.3 TMW Configuration

'tmwcnfg.h' contains configuration parameters that are used throughout the SCL. Each of these should be carefully considered. Normally a number of parameters in this file need to be changed from the default configuration.

```
/*****************************************************************************/
/* Triangle MicroWorks, Inc.                          Copyright (c) 1997-2022 */
/*****************************************************************************/
/*                                                                           */
/* This file is the property of:                                             */
/*                                                                           */
/*                       Triangle MicroWorks, Inc.                           */
/*                      Raleigh, North Carolina USA                          */
/*                       www.TriangleMicroWorks.com                          */
/*                          (919) 870-6615                                   */
/*                                                                           */
/* This Source Code and the associated Documentation contain proprietary     */
/* information of Triangle MicroWorks, Inc. and may not be copied or         */
/* distributed in any form without the written permission of Triangle        */
/* MicroWorks, Inc.  Copies of the source code may be made only for backup   */
/* purposes.                                                                 */
/*                                                                           */
/* Your License agreement may limit the installation of this source code to  */
/* specific products.  Before installing this source code on a new           */
/* application, check your license agreement to ensure it allows use on the  */
/* product in question.  Contact Triangle MicroWorks for information about   */
/* extending the number of products that may use this source code library or */
/* obtaining the newest revision.                                            */
/*                                                                           */
/*****************************************************************************/

/* file: tmwcnfg.h
 * description: This file specifies configuration parameters that apply to
 *  any TMW source code library.
 */
#ifndef TMWCNFG_DEFINED
#define TMWCNFG_DEFINED
#include "tmwscl/utils/tmwdefs.h"

#if TMW_USE_PRIVATE_DEFINES
/* Include private definitions for internal TMW applications.
 * SCL customers should not define TMW_USE_PRIVATE_DEFINES
 * or TMW_USE_GATEWAY_DEFINES.
 *
 * .NET SCL no longer defines TMW_USE_PRIVATE_DEFINES to select functionality
 * beyond the SCL defaults. tmwprvt.h is no longer included or modified to
 * select that extra functionality to match what is compiled in for the
 * 21 day install demo library.
 *
 * It is important for you to choose carefully what functionality you want
 * compiled in and provided by your application.
 *
 * Modify *cnfg.h and *data.h files as described in the xx xxx User Manual.pdf
 * to choose what functionality is supported.
 */
#include "tmwscl/config/tmwprvt.h"
#endif

#if TMW_USE_GATEWAY_DEFINES
 /* Include private definitions for TMW Gateway application. */

#include "tmwscl/config/tmwsdg.h"
#endif

/* The TMW_PRIVATE define is intended only for use by internal TMW applications
 * and is not required by customer applications.
 */

/* TMWTARG_HAS_VSNPRINTF specifies if the target C runtime library has a
 * vsnprintf function.  If this function exists length checking will be used
 * for tmwtarg_snprintf() function calls.  If not defined the length will not
 * be checked for tmwtarg_snprintf() function calls, note that this may cause
 * buffer overflows if the resulting length of a sprintf call exceeds the
```

```
   * length of the supplied buffer.
   */
#define TMWCNFG_HAS_VSNPRINTF         TMWDEFS_TRUE

/* TMWCNFG_SUPPORT_DIAG enables the generation of diagnostic information
 * which is passed to the target application by the tmwtarg_putDiagString
 * method defined below. Setting this parameter to TMWDEFS_FALSE will
 * disable the generation of diagnostic information.
 */
#define TMWCNFG_SUPPORT_DIAG          TMWDEFS_TRUE

/* TMWCNFG_SUPPORT_STATS enables the generation of statistical information
 * which is passed to the target application by the tmwtarg_updateStatistics
 * method defined below. Setting this parameter to TMWDEFS_FALSE line will
 * disable the generation of statistical information.
 */
#define TMWCNFG_SUPPORT_STATS         TMWDEFS_TRUE

/* Define whether or not single and/or double precision floating point
 * support should be included.
 *
 * Setting this parameter to TMWDEFS_FALSE will remove all references
 * to single precision floating point variables in the TMW SCL. This is
 * required for some data types in DNP and 101, 103 and 104. Setting this
 * to false will remove support for the data types that require single
 * precision floating point.
 */
#define TMWCNFG_SUPPORT_FLOAT         TMWDEFS_TRUE

/* Setting this parameter to TMWDEFS_FALSE will remove all references
 * to double precision floating point variables in the TMW SCL.
 * This is required for some data types in DNP. Setting this to false
 * will remove support for the data types that require double
 * precision floating point.
 */
#define TMWCNFG_SUPPORT_DOUBLE        TMWDEFS_TRUE

/* The simulated database requires double precision data types to provide
 * complete precision for both single precision floating point and large
 * long values. Even though both types require 32 bits of storage, single precision
 * only provides for 23 bits a precision with a larger range, while a long allows
 * for 32 bits of precision with a smaller range.
 * Setting this to TMWDEFS_FALSE will allow the simulated database to be
 * compiled without using double precision types, but at the expense of data
 * precision.  If short floats and longs are both required, your database may need
 * to provide better precision than the example simulated database.
 */
#define TMWCNFG_SIM_USE_DOUBLE        TMWDEFS_TRUE

/* The simulated database requires extended strings so support DNP Object group 114.
 * Setting this to TMWDEFS_FALSE will allow the simulated database to be
 * consume less memory, but will not be able to support DNP Object group 114.
 */
#define TMWCNFG_SIM_SUPPORT_EXT_STRINGS   TMWDEFS_TRUE

/* Define whether or not multiple threads are supported by the TMW SCL. If
 * this parameter is set to TMWDEFS_FALSE it is assumed that all the TMW SCL
 * code will run on a single thread (or different threads but not concurrently)
 * and all resource locking will be removed. If this parameter is set to
 * TMWDEFS_TRUE the TMW SCL will provide resource locking for each channel,
 * and optionally for the database processing queue as specified by the
 * TMWCNFG_SUPPORT_ASYNCH_DB parameter. This resource locking is managed
 * entirely within the SCL and no special support is required by the user.
 * For more information see the section on multi-threaded applications in
 * the TMW SCL user manual.
 */
#define TMWCNFG_SUPPORT_THREADS       TMWDEFS_TRUE

/* Define whether or not multiple timer queues, one for each channel are
 * supported. The default configuration for the SCL is to create a single
 * queue for all SCL timers and to require only a single system or polled timer.
 * For some multithreaded architectures it is desirable to create a separate
 * timer queue for each channel. This allows the timer callback functions to be
 * run in the context of the channel and can reduce resource contention. Setting
 * this to TMWDEFS_FALSE will not create separate timer queues. Setting this
 * to TMWDEFS_TRUE will create separate queues and require multiple system
 * timers through a call to tmwtarg_setMultiTimer().
 */
#define TMWCNFG_MULTIPLE_TIMER_QS     TMWDEFS_FALSE

/* Define whether database processing should be performed asynchronous
```

```
150  * to the rest of the TMW SCL processing in Master SCLs. This parameter is
     * ignored in Slave SCLs. If this parameter is set to TMWDEFS_TRUE database
     * updates will be queued during message parsing. The user can then process
     * the updates from the queue as time permits, possibly on a different
     * thread. This feature is useful if database updates take a significant
     * amount of time such that processing them while parsing the message would
     * delay further processing of the time dependent SCL code beyond acceptable
     * limits. Note that the use of asynchronous database updates currently
     * requires the use of dynamic memory, hence TMWCNFG_USE_DYNAMIC_MEMORY
     * must be set to TMWDEFS_TRUE.
160  */
    #ifndef TMWCNFG_SUPPORT_ASYNCH_DB
    #define TMWCNFG_SUPPORT_ASYNCH_DB     TMWDEFS_TRUE
    #endif

    /* If this parameter is set to TMWDEFS_TRUE the TMW SCL will use a simulated
     * database. Define this to TMWDEFS_FALSE to remove all references to the
     * simulated database.
     */
    #ifndef TMWCNFG_USE_SIMULATED_DB
170 #define TMWCNFG_USE_SIMULATED_DB       TMWDEFS_TRUE
    #endif

    /* TMWCNFG_SUPPORT_RXCALLBACKS controls whether certain optional callback
     * functionality provided for test or processing of received messages by
     * applications outside of the SCL is compiled in.
     * DNPLINK_CONTEXT pRxHeaderCallback
     * TMWLINK_CONTEXT pUserRxFrameCallback
     * TMWTPRT_CONTEXT pUserParseFunc
     * TMWPHYS_CONTEXT pUserParseFunc
180  * at .NET level
     * TMWChannel::rxLinkHdrCallbackEvent uses DNPLINK_CONTEXT pRxHeaderCallback
     *  only registered and implemented by DNP for now
     * TMWChannel::rxLinkCallbackEvent uses TMWLINK_CONTEXT pUserRxFrameCallback
     *  only registered and implemented by DNP for now
     * TMWChannel::rxTransportCallbackEvent uses TMWTPRT_CONTEXT pUserParseFunc
     *  only for DNP transport
     * TMWChannel::rxPhysCallbackEvent uses TMWPHYS_CONTEXT pUserParseFunc
     *  registered and implemented for all channels.
     */
190 #ifndef TMWCNFG_SUPPORT_RXCALLBACKS
    #define TMWCNFG_SUPPORT_RXCALLBACKS TMWDEFS_FALSE
    #endif

    /* Compile in common Cryptography interface tmwcrypto.h/c */
    #ifndef TMWCNFG_SUPPORT_CRYPTO
    #define TMWCNFG_SUPPORT_CRYPTO         TMWDEFS_FALSE
    #endif

    /* Compile in optional Asymmetric (public key) cryptography interface */
200 #ifndef TMWCNFG_SUPPORT_CRYPTO_ASYM
    #define TMWCNFG_SUPPORT_CRYPTO_ASYM    TMWDEFS_FALSE
    #endif

    /* Compile in optional AES-GMAC Hash Algorithm support
     * If using OpenSSL this requires a version that contains aes-gmac.c
     * such as openssl 1.1.1beta1.
     */
    #ifndef TMWCNFG_SUPPORT_CRYPTO_AESGMAC
    #define TMWCNFG_SUPPORT_CRYPTO_AESGMAC TMWDEFS_FALSE
210 #endif

    /* If you want to support Secure Authentication using the common
     * cryptography interface (TMWCNFG_SUPPORT_CRYPTO) and you do not use OpenSSL
     * you must modify tmwcrypto.c/h to use another cryptography library
     * To allow a test build as delivered simulated crypto code that builds but
     * does NOT actually perform cryptography is provided.
     * If you implement your own cryptography code, this define Must be set
     * to TMWDEFS_FALSE
     */
220 #define TMWCNFG_USE_SIMULATED_CRYPTO     TMWDEFS_TRUE

    /* If you want to support Secure Authentication using the common
     * cryptography interface (TMWCNFG_SUPPORT_CRYPTO) you must modify
     * tmwcrypto.c/h to provide a database to store users and keys.
     * To allow a test build as delivered simulated crypto database code that stores
     * a small amount of data in memory is provided.
     * When you implement database, this define Must be set to TMWDEFS_FALSE
     */
    #define TMWCNFG_USE_SIMULATED_CRYPTO_DB  TMWDEFS_TRUE
230
```

```
    /* Use OpenSSL implementation for tmwcrypto interface if Secure Authentication
     * is supported and/or for TLS if it is supported.
     */
    #ifndef TMWCNFG_USE_OPENSSL
    #define TMWCNFG_USE_OPENSSL              TMWDEFS_FALSE
    #endif
    /* Use OpenSSL version 1_0_2x instead of 1.1.1. OpenSSL library introduced a number
     * of API changes in 1.1.1 so this conditional was introduced to accomodate the
     * changes. If the application is using a 1.0.x OpenSSL library, this define
240  * must be set to TMWDEFS_TRUE.
     */
    #define TMWCNFG_USE_OPENSSL_1_0_2        TMWDEFS_FALSE

    /* Specify whether dynamic memory allocation is supported. Set this parameter
     * to TMWDEFS_FALSE and specify limits for each data type below to use
     * static compile time memory allocation. Set this to TMWDEFS_TRUE to support
     * dynamic memory allocation at runtime (tmwtarg_alloc() will be called).
     * Note that the maximum limits specified below for each allocated data
     * type apply when using any type of memory allocation. Hence it
250  * is possible to use dynamic memory allocation and still specify a
     * maximum limit to the amount of memory that will be used by the TMW SCL.
     */
    #define TMWCNFG_USE_DYNAMIC_MEMORY    TMWDEFS_TRUE

    /* To support allocation of all of the memory required for each memory pool
     * at startup, set both TMWCNFG_USE_DYNAMIC_MEMORY and
     * TMWCNFG_ALLOC_ONLY_AT_STARTUP to TMWDEFS_TRUE. It is also necessary to specify
     * the limit for each type of data below. The SCL will call tmwtarg_alloc()
     * once for each pool and then will manage the memory internally.
260  * To change the number of buffers in each pool at runtime, call protocol specific
     * xxxmem_initMemory functions such as sdnpmem_initMemory() or
     * m101mem_initMemory() to change the compiled in quantities.
     */
    #define TMWCNFG_ALLOC_ONLY_AT_STARTUP TMWDEFS_FALSE

    /* Specify if a processor requires long word (4byte) alignment or if the
     * compiler creates unpacked structures. Setting this to TMWDEFS_TRUE will pad
     * TMWMEM_HEADER to address memory alignment and sizof(TMWMEM_HEADER) issues.
     * If sizeof(TMWMEM_HEADER) == 2, but the data field in tmwmem.h starts at
270  * offset 4, (2 bytes were left empty), this should be set to TMWDEFS_TRUE to
     * pad structure TMWMEM_HEADER to 4 bytes.
     */
    #define  TMWCNFG_MEMORY_ALIGN_NEEDED  TMWDEFS_FALSE

    /* Specify maximum number of consecutive calls to tmwtarg_receive() to receive
     * data when tmwtarg_receive returns non zero. This can be used to prevent
     * looping in the SCL when a constant stream of data is being received. This
     * applies in both the poll for input or event driven input models.
     * NOTE: it may take multiple calls to tmwtarg_receive to receive a single frame.
280  * This is set to a large number by default, reduce it only as needed.
     */
    #define TMWCNFG_MAX_APPLRCVS          512

    /* Specify the maximum number of application contexts that can be in use
     * at once. Typically this value will be 1 per supported protocol unless
     * different threads are being used to manage multiple sessions.
     */
    #define TMWCNFG_MAX_APPLICATIONS      TMWDEFS_NO_LIMIT

290 /* Specify the maximum number of channels that can be open at once.
     */
    #define TMWCNFG_MAX_CHANNELS          TMWDEFS_NO_LIMIT

    /* Specify the maximum number of sessions that can be open at once.
     */
    #define TMWCNFG_MAX_SESSIONS          TMWDEFS_NO_LIMIT

    /* Specify the maximum number of sectors that can be open at once.
     * For protocols that do not support sectors this parameter is ignored.
300  */
    #define TMWCNFG_MAX_SECTORS           TMWDEFS_NO_LIMIT

    /* Specify the maximum number of application layer messages that can be
     * outstanding at one time (i.e. the total number of simultaneous commands
     * outstanding or the number of simultaneous msgs sent from a slave in 104)
     * This parameter can generally be set to 1 per session/sector for slave devices
     * other than IEC 60870-5-104. (SDNP configured for Identical Unsolicited Response
     * Retries requires 2). For 104 slaves and other master devices this should be set
to
     * N * the number of sessions or sectors where N is the maximum number of messages
310  * that will be queued at the application layer.
```

```
     */
    #define TMWCNFG_MAX_APPL_MSGS          TMWDEFS_NO_LIMIT

    /* Specify the maximum number of events that can be queued per data type.
     * Each data type maintains it's own queue of events. This parameter
     * specifies the default maximum length for each queue.
     */
    #define TMWCNFG_MAX_EVENTS             TMWDEFS_NO_LIMIT

320 /* Specify the maximum number of simulated databases at that can be open
     * at one time. Generally there will be one database for each IEC sector
     * or DNP session. This parameter is ignored if TMWCNFG_USE_SIMULATED_DB
     * is set to TMWDEFS_FALSE.
     */
    #define TMWCNFG_MAX_SIM_DATABASES      TMWDEFS_NO_LIMIT

    /* Specify the maximum number of simulated database points that can exist
     * at one time. This parameter is ignored if TMWCNFG_USE_SIMULATED_DB
     * is set to TMWDEFS_FALSE.
330  */
    #define TMWCNFG_MAX_POINTS             TMWDEFS_NO_LIMIT

    /* Specify the maximum number of simulated database strings that can exist
     * at one time. This parameter is ignored if TMWCNFG_USE_SIMULATED_DB
     * is set to TMWDEFS_FALSE.
     */
    #define TMWCNFG_MAX_STRINGS            TMWDEFS_NO_LIMIT
    #define TMWCNFG_MAX_EXT_STRINGS        TMWDEFS_NO_LIMIT

340 /* Specify the maximum number of bytes that can be buffered at the
     * physical layer. This parameter is only used if TMWCNFG_USE_DYNAMIC_MEMORY
     * is TMWDEFS_FALSE.
     */
    #define TMWCNFG_MAX_RX_BUFFER_LENGTH 265

    /* This should only be defined when a .NET Source Code Library has been purchased
     * in addition to the ANSI C SCL
     */
    /* define TMWCNFG_USE_MANAGED_SCL TMWDEFS_TRUE */
350
    /* check to make sure memory related defines do not conflict */
    #if !TMWCNFG_USE_DYNAMIC_MEMORY && TMWCNFG_ALLOC_ONLY_AT_STARTUP
    #pragma message("If dynamic memory is not supported, can't use alloc only at
startup")
    #endif

    #if !TMWCNFG_USE_DYNAMIC_MEMORY && TMWCNFG_SUPPORT_ASYNCH_DB
    #pragma message("If dynamic memory is not supported, can't use async database")
    #endif

360 #if TMWCNFG_ALLOC_ONLY_AT_STARTUP
    #if (TMWCNFG_MAX_APPLICATIONS == TMWDEFS_NO_LIMIT \
      || TMWCNFG_MAX_CHANNELS == TMWDEFS_NO_LIMIT \
      || TMWCNFG_MAX_SESSIONS == TMWDEFS_NO_LIMIT \
      || TMWCNFG_MAX_SECTORS == TMWDEFS_NO_LIMIT \
      || TMWCNFG_MAX_APPL_MSGS == TMWDEFS_NO_LIMIT \
      || TMWCNFG_MAX_EVENTS == TMWDEFS_NO_LIMIT)
    #pragma message("If TMWCNFG_ALLOC_ONLY_AT_STARTUP, TMWDEFS_NO_LIMIT is not allowed")
    #endif

370 #if TMWCNFG_USE_SIMULATED_DB
    #if (TMWCNFG_MAX_SIM_DATABASES == TMWDEFS_NO_LIMIT \
      || TMWCNFG_MAX_POINTS == TMWDEFS_NO_LIMIT \
      || TMWCNFG_MAX_STRINGS == TMWDEFS_NO_LIMIT)
    #pragma message("If TMWCNFG_ALLOC_ONLY_AT_STARTUP, TMWDEFS_NO_LIMIT for SIM_DATABASE
quantities is not allowed")
    #endif
    #endif /* TMWCNFG_USE_SIMULATED_DB */

    #endif /* TMWCNFG_ALLOC_ONLY_AT_STARTUP */

380 #ifndef TMWCNFG_UNUSED_PARAM
    /* Specify this to avoid 'unused parameter' warnings
     * Depending on your compiler, this definition may need to be redefined.
     */
    #define TMWCNFG_UNUSED_PARAM(x) (void)x
    #endif

    #endif /* TMWCNFG_DEFINED */
```

3.4 MDNP Configuration

'mdnpcnfg.h' contains Master DNP specific configuration parameters.

```
/*****************************************************************************/
/* Triangle MicroWorks, Inc.                          Copyright (c) 1997-2022 */
/*****************************************************************************/
/*                                                                           */
/* This file is the property of:                                             */
/*                                                                           */
/*                      Triangle MicroWorks, Inc.                            */
/*                     Raleigh, North Carolina USA                           */
/*                      www.TriangleMicroWorks.com                           */
/*                        (919) 870-6615                                     */
/*                                                                           */
/* This Source Code and the associated Documentation contain proprietary     */
/* information of Triangle MicroWorks, Inc. and may not be copied or         */
/* distributed in any form without the written permission of Triangle        */
/* MicroWorks, Inc.  Copies of the source code may be made only for backup   */
/* purposes.                                                                 */
/*                                                                           */
/* Your License agreement may limit the installation of this source code to  */
/* specific products.  Before installing this source code on a new           */
/* application, check your license agreement to ensure it allows use on the  */
/* product in question.  Contact Triangle MicroWorks for information about   */
/* extending the number of products that may use this source code library or */
/* obtaining the newest revision.                                            */
/*                                                                           */
/*****************************************************************************/

/* file: dnpcnfg.h
 * description:   DNP master configuration definitions
 */
#ifndef MDNPCNFG_DEFINED
#define MDNPCNFG_DEFINED

#include "tmwscl/utils/tmwcnfg.h"

/* All *_NUMALLOC_* defines are used only to limit the number of structures of that type
 * that can be allocated. A value of TMWDEFS_NO_LIMIT (the default) means there is no limit
 * To use static or only at startup memory configuration TMWDEFS_NO_LIMIT is not possible
 * This DO NOT have to depend on the TMWCNFG_xxx defines, but can be changed here to set
 * desired values.
 */

/* Specify the maximum number of Master DNP sessions that can be open */
#define MDNPCNFG_NUMALLOC_SESNS              TMWCNFG_MAX_SESSIONS

/* Specify how many operate requests can be in progress. Generally
 * you can only have one request pending per channel so setting this to
 * the maximum number of configured channel is usually correct.
 */
#define MDNPCNFG_NUMALLOC_OPERCBACKDATAS     TMWCNFG_MAX_CHANNELS

/* Specify how many file transfer requests can be in progress. Currently
 * you can only have one request pending per session so setting this to
 * the maximum number of configured sessions is usually correct.
 */
#define MDNPCNFG_NUMALLOC_FILECBACKDATAS     TMWCNFG_MAX_SESSIONS

/* Specify how many automatic data set exchanges can be in progress.
 * You can only have one request pending per session so setting this to
 * the maximum number of configured sessions is usually correct.
 */
#define MDNPCNFG_NUMALLOC_DATASET_CTXS       TMWCNFG_MAX_SESSIONS

/* Specify how much time between reading data set prototype from slave and writing
 * master defined data set prototypes and between reading data set descriptor from
 * slave and writing master defined data set descriptors when doing automatic
 * exchange of data set information when master or slave restarts. This delay is
 * necessary if async database is processing is implemented on the master to allow
 * the information to be stored in the database before being read by SCL.
 * This value is in milliseconds.
```

```
 */
#define MDNPCNFG_DATASET_DELAY                3000

/* Support DNP Secure Authentication Specification Version 2 */
#define MDNPCNFG_SUPPORT_SA_VERSION2          DNPCNFG_SUPPORT_SA_VERSION2

/* Support DNP Secure Authentication Specification Version 5
 * This is an optional component.
 * DNPCNFG_SUPPORT_AUTHENTICATION must be TMWDEFS_TRUE.
 * TMWCNFG_SUPPORT_CRYPTO must be TMWDEFS_TRUE
 */
#define MDNPCNFG_SUPPORT_SA_VERSION5          DNPCNFG_SUPPORT_SA_VERSION5

/* Specify the total number of master DNP Secure Authentication Version 5 User
structures
 * for all sessions in the entire system that can be allocated when static or
 * dynamic memory with limited quantities is used.
 */
#define MDNPCNFG_NUMALLOC_AUTH_USERS          (DNPCNFG_AUTH_MAX_NUMBER_USERS *
TMWCNFG_MAX_SESSIONS)

/* Specify number of SA Internal State Machine events that can be allocated
 * for all sessions in the entire system when static or
 * dynamic memory with limited quantities is used.
 * Generally 1 per user with at least 3 if only a single user is allowed
 */
#define MDNPCNFG_NUMALLOC_AUTH_EVENTS         (MDNPCNFG_NUMALLOC_AUTH_USERS + 2)


#if TMWCNFG_USE_SIMULATED_DB
/* Specify the number of master DNP simulated databases in use. The TMW SCL
 * will allocate a new simulated database for each mdnp session.
 * These are not needed once an actual database is implemented.
 */
#define MDNPCNFG_NUMALLOC_SIM_DATABASES       TMWCNFG_MAX_SIM_DATABASES

/* Specify the number of master DNP simulated database Data Set structures that
 * can be allocated. These are only needed to simulate Data Set points.
 * These are not needed once an actual database is implemented.
 */
#define MDNPCNFG_NUMALLOC_SIM_DATASETS        (TMWCNFG_MAX_SESSIONS*10)

#endif

/* Specify the maximum block size for file transfer, this is just a value
 * for mdnpsesn_initConfig. Values 16 to 18 bytes less than the tx and rx fragment
 * sizes can be used.
 */
#define MDNPCNFG_FILE_BLOCK_SIZE 1024

#endif /* MDNPCNFG_DEFINED */
```

4 Calls from the Target Application into the Source Code Library

4.1 TMW Application

'tmwappl.h' contains the definition of the interface to the application context routines to manage the DNP3 Master SCL. This interface includes the function tmwappl_initApplication().

```
/*****************************************************************************/
/* Triangle MicroWorks, Inc.                          Copyright (c) 1997-2022 */
/*****************************************************************************/
/*                                                                           */
/* This file is the property of:                                             */
/*                                                                           */
/*                       Triangle MicroWorks, Inc.                           */
/*                      Raleigh, North Carolina USA                          */
/*                       www.TriangleMicroWorks.com                          */
/*                          (919) 870-6615                                   */
/*                                                                           */
/* This Source Code and the associated Documentation contain proprietary     */
/* information of Triangle MicroWorks, Inc. and may not be copied or         */
/* distributed in any form without the written permission of Triangle        */
/* MicroWorks, Inc.  Copies of the source code may be made only for backup   */
/* purposes.                                                                 */
/*                                                                           */
/* Your License agreement may limit the installation of this source code to  */
/* specific products.  Before installing this source code on a new           */
/* application, check your license agreement to ensure it allows use on the  */
/* product in question.  Contact Triangle MicroWorks for information about   */
/* extending the number of products that may use this source code library or */
/* obtaining the newest revision.                                            */
/*                                                                           */
/*****************************************************************************/

/* file: tmwappl.h
 * description: Channel related structures and functions
 *  implementations.
 */
#ifndef TMWAPPL_DEFINED
#define TMWAPPL_DEFINED

#include "tmwscl/utils/tmwcnfg.h"
#include "tmwscl/utils/tmwdefs.h"
#include "tmwscl/utils/tmwdlist.h"
#include "tmwtargos.h"  /* TMWDEFS_RESOURCE_LOCK */

/* Used to determine if protocol specific initialization function
 * has been called
 */
typedef TMWTYPES_ULONG  TMWAPPL_INIT_COMPONENT;
#define TMWAPPL_INIT_DNP          0x00000001
#define TMWAPPL_INIT_MDNP         0x00000002
#define TMWAPPL_INIT_SDNP         0x00000004
#define TMWAPPL_INIT_MB           0x00000008
#define TMWAPPL_INIT_MMB          0x00000010
#define TMWAPPL_INIT_SMB          0x00000020
#define TMWAPPL_INIT_I870         0x00000040
#define TMWAPPL_INIT_FT12         0x00000080
#define TMWAPPL_INIT_101          0x00000100
#define TMWAPPL_INIT_M101         0x00000200
#define TMWAPPL_INIT_S101         0x00000400
#define TMWAPPL_INIT_102          0x00000800
#define TMWAPPL_INIT_M102         0x00001000
#define TMWAPPL_INIT_S102         0x00002000
#define TMWAPPL_INIT_103          0x00004000
#define TMWAPPL_INIT_M103         0x00008000
#define TMWAPPL_INIT_S103         0x00010000
#define TMWAPPL_INIT_104          0x00020000
#define TMWAPPL_INIT_M104         0x00040000
#define TMWAPPL_INIT_S104         0x00080000
#define TMWAPPL_INIT_61850        0x00100000
#define TMWAPPL_INIT_M61850       0x00200000
```

```
#define TMWAPPL_INIT_S61850       0x00400000
#define TMWAPPL_INIT_LAST         TMWAPPL_INIT_S61850

/* Application layer context
 */
typedef struct TMWApplicationStruct {

  /* List of channels managed by this application context */
  TMWDLIST channels;

#if TMWCNFG_SUPPORT_THREADS
  TMWDEFS_RESOURCE_LOCK lock;
#endif

} TMWAPPL;

#ifdef __cplusplus
extern "C" {
#endif

  /* function: tmwappl_init
   * purpose: Initialize SCL if initialized data is not provided.
   *  If your environment does not allow initialized data
   *  this must be called before calling tmwappl_initSCL and tmwapp_initApplication
   *  or any other functions from the SCL
   * arguments:
   *  void
   * returns:
   *  void
   */
  void TMWDEFS_GLOBAL tmwappl_init(void);

  /* function: tmwappl_internalInit
   * purpose: INTERNAL FUNCTION called from within SCL
   * arguments:
   *  void
   * returns:
   *  TMWDEFS_TRUE if this is the the first time this is called
   *  TMWDEFS_FALSE if this has already been called.
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL tmwappl_internalInit(void);

  /* function: tmwappl_initSCL
  * purpose: Initialize the SCL for a multithreaded environment when multiple
  *  application contexts are required.
  *  NOTE This function must be called prior to any calls to tmwappl_initApplication
  *  in a multi-threaded environment with multiple application contexts. The call
  *  is optional when TMWCNFG_SUPPORT_THREADS is not defined.
  * arguments:
  *  void
  * returns:
  *  TMWDEFS_TRUE if the SCL has initialized correctly
  *  TMWDEFS_FALSE The SCL initializaiton failed.
  */
  TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL tmwappl_initSCL(void);

  /* function: tmwappl_initApplication
   * purpose: Initialize a new application context
   *  NOTE to support multiple application contexts you need to support multiple
   *  timer queues. define TMWCNFG_MULTIPLE_TIMER_QS
   * arguments:
   *  void
   * returns:
   *  pointer to new application context or TMWDEFS_NULL
   */
  TMWDEFS_SCL_API TMWAPPL * TMWDEFS_GLOBAL tmwappl_initApplication(void);

  /* function: tmwappl_closeApplication
   * purpose: Close an application context
   * arguments:
   *  pApplContext - pointer to application context
   *  forceClose - TMWDEFS_TRUE to force context to close any open channels
   *   NOTE: This is not currently implemented
   * returns:
   *  TMWDEFS_TRUE if successful
   *  TMWDEFS_FALSE otherwise
   *   NOTE: close will fail if there are open channels for this appl
   *         context and forceClose if TMWDEFS_FALSE.
   */
  TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL tmwappl_closeApplication(
    TMWAPPL *pApplContext,
```

```
    TMWTYPES_BOOL forceClose);

  /* function: tmwappl_getInitialized
   * purpose: Determine if a specific protocol has been marked initialized. This
   *  will be called by the protocol specific portions of the SCL. It is not
   *  intended to be called by application code.
   * arguments:
   *   component - indicate protocol component
   * returns:
   *  TMWDEFS_TRUE if protocol/type has already been marked initialized
   *  TMWDEFS_FALSE if protocol/type had not been previously initialized
   */
  TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL tmwappl_getInitialized(
    TMWAPPL_INIT_COMPONENT component);

  /* function: tmwappl_setInitialized
   * purpose: Mark the specific protocol initialized.
   * arguments:
   *   component - indicate protocol component
   * returns:
   *   void
   */
  TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwappl_setInitialized(
    TMWAPPL_INIT_COMPONENT component);

  /* function: tmwappl_checkForInput
   * purpose: Check for input on any of the specified application context's
   *  input channels.
   * arguments:
   *  pApplContext - application context returned by tmwappl_initApplication
   * returns:
   *  void
   */
  TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwappl_checkForInput(
    TMWAPPL *pApplContext);

  /* function: tmwappl_findChannel
   * purpose: See if this channel is on the list of application channels. This
   *  can be used to check if a channel was deleted from a target layer callback
   *  function.
   * arguments:
   *  pApplContext - application context returned by tmwappl_initApplication
   *  pChannel - channel context to find
   * returns:
   * TMWDEFS_TRUE if channel is found on list
   */
  TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL tmwappl_findChannel(
    TMWAPPL *pApplContext,
    void *pChannel);

#ifdef __cplusplus
}
#endif
#endif /* TMWAPPL_DEFINED */
```

4.2 DNP Channels

'dnpchnl.h' contains the definition of the routines required to configure, open and close communication channels in the DNP3 Master SCL. In particular, the functions dnpchnl_initConfig() and dnpchnl_openChannel() are called from the target application. The DNPCHNL_CONFIG structure may optionally be modified after the return from the call to dnpchnl_initConfig().

```
/*****************************************************************************/
/* Triangle MicroWorks, Inc.                          Copyright (c) 1997-2022 */
/*****************************************************************************/
/*                                                                           */
/* This file is the property of:                                             */
/*                                                                           */
/*                      Triangle MicroWorks, Inc.                            */
/*                     Raleigh, North Carolina USA                           */
/*                      www.TriangleMicroWorks.com                           */
/*                         (919) 870-6615                                    */
/*                                                                           */
/* This Source Code and the associated Documentation contain proprietary     */
/* information of Triangle MicroWorks, Inc. and may not be copied or         */
/* distributed in any form without the written permission of Triangle        */
/* MicroWorks, Inc.  Copies of the source code may be made only for backup   */
/* purposes.                                                                 */
/*                                                                           */
/* Your License agreement may limit the installation of this source code to  */
/* specific products.  Before installing this source code on a new           */
/* application, check your license agreement to ensure it allows use on the  */
/* product in question.  Contact Triangle MicroWorks for information about   */
/* extending the number of products that may use this source code library or */
/* obtaining the newest revision.                                            */
/*                                                                           */
/*****************************************************************************/

/* file: dnpchnl.h
 * description: DNP3 Channel Implementation.
 */
#ifndef DNPCHNL_DEFINED
#define DNPCHNL_DEFINED

#include "tmwscl/utils/tmwappl.h"
#include "tmwscl/utils/tmwchnl.h"
#include "tmwscl/utils/tmwphys.h"

#include "tmwscl/dnp/dnplink.h"
#include "tmwscl/dnp/dnptprt.h"
#include "tmwscl/dnp/dnpbncfg.h"

/* Arbitrary magic number, to indicate if TMWSESN_TX_DATA structure has
 * been allocated or deallocated
 */
#define DNPCHNL_TXDATA_VALID 0x5555

 /* Default priorities for Tx Data */
#define DNPCHNL_DEFAULT_PRIORITY 128
#define DNPCHNL_DEFAULT_AUTH_PRIORITY 251

/* Define support for user response callbacks. The user can specify a user
 * callback when requests are generated by calling the various request
 * functions defined in mdnpbrm.h. This callback will be called to alert
 * the user to the the current status of the request. As described in
 * mdnpbrm.h most requests will consist of a single request/response in
 * which case the user callback will be called once with either a success
 * status or one of the failure status codes. Some requests require multiple
 * request/response cycles in which case the user callback will get called
 * one or more times with an intermediate response status. The request is
 * complete when the response status is success or a failure code.
 */
typedef enum DNPChannelRespStatus {

  /* This indicates the request has completed successfully.
   */
  DNPCHNL_RESP_STATUS_SUCCESS = 0,
```

```
    /* This indicates a response was received but the requested command is
     * not yet complete. This could mean the response is part of a multi-fragment
     * response and did not have the FINAL bit set. Or this could be a request such
     * as a select operate that requires multiple requests and responses.
     */
    DNPCHNL_RESP_STATUS_INTERMEDIATE,

    /* This indicates that the transmission of the request failed
     */
    DNPCHNL_RESP_STATUS_FAILURE,

    /* The response to a select or an execute did not echo the request
     */
    DNPCHNL_RESP_STATUS_MISMATCH,

    /* The response to a select or an execute echoed the request, except the
     * status code was different indicating a failure.
     */
    DNPCHNL_RESP_STATUS_STATUSCODE,

    /* The response to the request had IIN bits set indicating the command failed.
     *
     */
     DNPCHNL_RESP_STATUS_IIN,

    /* This indicates that the request has timed out. This could either be an
     * incremental timeout indicating we received no link layer frame from the
     * device in the specified time, or an application response timeout indicating
     * this particular request did not complete in the specified time.
     */
    DNPCHNL_RESP_STATUS_TIMEOUT,

    /* This indicates either that the user asked that the request be canceled by
     * calling dnpchnl_cancelFragment or that a second duplicate request has been
     * made and therefore this first one is canceled.
     */
    DNPCHNL_RESP_STATUS_CANCELED
  } DNPCHNL_RESP_STATUS;

  /* Structure which contains the information passed to the user code in the
   * user callback.
   *  last - TMWDEFS_TRUE if this is the last time the callback will be called
   *   for this request.
   *  pSession - session request was issued to
   *  iin - State of the Internal Indication Bits in the response
   *  status - response status
   *  requestStatus - request specific status information
   *  responseTime - time between when command was sent and response received
   *  pTxData - pointer to original request transmit data structure returned
   *   by the call to the mdnpbrm request function which generated the request.
   *  pRxData - pointer to received response data structure or TMWDEFS_NULL if
   *   no response.
   */
  typedef struct DNPChannelResponseStruct {
    TMWTYPES_BOOL last;
    TMWSESN *pSession;
    TMWTYPES_USHORT iin;
    DNPCHNL_RESP_STATUS status;
    TMWTYPES_UCHAR requestStatus;
    TMWTYPES_MILLISECONDS responseTime;
    TMWSESN_TX_DATA *pTxData;
    TMWSESN_RX_DATA *pRxData;
  } DNPCHNL_RESPONSE_INFO;

  /* Define the response callback function.
   */
  typedef void (*DNPCHNL_CALLBACK_FUNC)(
    void *pCallbackParam,
    DNPCHNL_RESPONSE_INFO *pResponse);

  /* DNP Specific tx flags */
  #define DNPCHNL_DNPTXFLAGS_AUTH_NOACKDELAY     0x0001

  typedef struct DNPChannelTransDataStruct {
    /* Generic TMW data structure, must be first entry */
    TMWSESN_TX_DATA tmw;

    /* DNP Specific tx flags */
    TMWTYPES_ULONG dnpTxFlags;

    /* Has this request been sent */
```

```
    TMWTYPES_BOOL sent;

    /* Priority of this request */
    TMWTYPES_UCHAR priority;

    TMWTYPES_UCHAR referenceCount;

    /* User number to use for secure authentication
     * if aggressive mode is selected or outstation challenges this request
     */
    TMWTYPES_USHORT authUserNumber;
    TMWTYPES_UCHAR  authAggrModeObjLength;
    TMWTYPES_BOOL   authAggressiveMode;

    /* Internal callback */
    void *pInternalCallbackParam;
    DNPCHNL_CALLBACK_FUNC pInternalCallback;

    /* User callback */
    void *pUserCallbackParam;
    DNPCHNL_CALLBACK_FUNC pUserCallback;

#if !TMWCNFG_USE_DYNAMIC_MEMORY || TMWCNFG_ALLOC_ONLY_AT_STARTUP
    /* Buffer to hold the message to be transmitted */
    TMWTYPES_UCHAR buffer[DNPCNFG_MAX_TX_FRAGMENT_LENGTH];
#endif

} DNPCHNL_TX_DATA;

/* DNP3 Channel configuration structure. This structure contains
 * configuration parameters that are specific to a DNP3 channel.
 */
typedef struct {

    /* Maximum receive and transmit application fragment sizes
     *  The DNP3 specification recommends a value of 2048.
     */
    TMWTYPES_USHORT        rxFragmentSize;
    TMWTYPES_USHORT        txFragmentSize;

    /* Removed rxFrameSize and txFrameSize.
     * To configure frame sizes set
     *  DNPLINK_CONFIG txFrameSize and rxFrameSize
     */

    /* Maximum number of request messages that will be queued on a DNP3 master
     * Setting this to 0 allows an unlimited number of requests to be queued.
     * Setting this to 1 allows only 1 outstanding request at a time,
     * Setting this to a number greater than 1 specifies that number of
     *   outstanding and queued requests
     * NOTE: MDNP session auto generation of requests requires more than
     *  1 request to be queued. If setting maxQueueSize to 1 you should
     *  set MDNP session autoRequestMask to 0.
     */
    TMWTYPES_USHORT        maxQueueSize;

    /* For a DNP master how long to wait for a response to a request that has
     * actually been transmitted.
     * This value can be made shorter than the
     * MDNPSESN_CONFIG defaultResponseTimeout
     * to quickly determine if a particular device is not responding, without
     * causing requests to other devices on the same channel to also time out.
     * NOTE: This is not used by a DNP slave.
     */
    TMWTYPES_MILLISECONDS  channelResponseTimeout;

    /* For a DNP master how long to delay sending a request to offline sessions.
     * This may allow other online sessions on that channel to communicate better.
     * When this time expires, a queued request will be attempted to an offline
     * session.
     * If that request times out the next request for an offline session may be
     * delayed again by this amount of time. The number of sessions on a channel
     * channelResponseTimeout should be considered when setting this value. For this
     * and the to have an affect it needs to be larger than channelResponseTimeout.
     * The longer this delay, the longer it might take to detect a session could
     * respond. Other responses such as link layer or unsolicited responses may
     * also bring the session online.
     */
    TMWTYPES_MILLISECONDS  channelOffLineDelay;

    /* routine to call when events occur on this channel,
```

```
     *   can be used to track statistics on this channel.
230   */
    TMWCHNL_STAT_CALLBACK  pStatCallback;

    /* user callback parameter to pass to pStatCallback
     */
    void                  *pStatCallbackParam;

    /* routine to call when channel becomes idle. That is there are
     * no commands queued to be sent out, none being transmitted,
     * and there are no responses or confirms expected back.
240  * This can be used to provide support for modem pools,
     * allowing the user to know when it is safe to disconnect
     * the line from the SCLs point of view.
     */
    TMWCHNL_IDLE_CALLBACK  pIdleCallback;

    /* user callback parameter to pass to pIdleCallback
     */
    void                  *pIdleCallbackParam;

250 /* routine to call when a message has been received for a session that is
     * not currently open. This user provided function can determine whether or
     * not to open the session and allow the received message to be processed.
     * For details on this see definition of TMWCHNL_AUTO_OPEN_FUNC in tmwchnl.h
     */
    TMWCHNL_AUTO_OPEN_FUNC  pAutoOpenCallback;

    /* user callback parameter to pass to pAutoOpenCallback
     */
    void                   *pAutoOpenCallbackParam;
260
    /* Mask for disabling generation of diagnostic strings */
    TMWTYPES_ULONG          chnlDiagMask;

  } DNPCHNL_CONFIG;

  /* Define context for a DNP3 specific channel. This structure is for internal
   * TMW SCL use only.
   */
  typedef struct {
270
    /* Define generic TMW channel, must be first entry */
    TMWCHNL tmw;

    /* To round robin between sessions */
    TMWSESN *pLastSessionRequest;
    TMWSESN *pLastOfflineSessionRequest;

    /* Maximum size for a transmit fragment */
    TMWTYPES_USHORT txFragmentSize;
280
    /* Maximum size for a receive fragment */
    TMWTYPES_USHORT rxFragmentSize;

    /* To delay sending requests to offline sessions if configured */
    TMWTYPES_MILLISECONDS  channelOffLineDelay;
    TMWTIMER channelOffLineDelayTimer;

  #if DNPCNFG_SUPPORT_AUTHENTICATION
    /* Used only on master with secure authentication,
290  * delay this long to see if a challenge is received
     * when sending non broadcast direct operate noack commands.
     */
    TMWTYPES_MILLISECONDS directNoAckDelayTime;

  #if !DNPCNFG_MULTI_SESSION_REQUESTS
    TMWTYPES_USHORT lastTxFragmentLength;
    TMWTYPES_USHORT lastUnsolTxFragmentLength;
    /* lastUnsolTxFragment is only required if there is a slave session on this
     * channel. This could be removed if only supporting master sessions.
300  */
    TMWTYPES_UCHAR  lastUnsolTxFragment[DNPCNFG_MAX_TX_FRAGMENT_LENGTH];
    TMWTYPES_UCHAR  lastTxFragment[DNPCNFG_MAX_TX_FRAGMENT_LENGTH];
  #endif

  #endif
  } DNPCHNL;

  #ifdef __cplusplus
  extern "C" {
```

```
#endif

/* function: dnpchnl_initConfig
 * purpose: Initialize DNP3 channel configuration data structures. The
 *  user should call this routing to initialize these data structures
 *  and then modify the desired fields before calling dnpchnl_openChannel
 *  to actually open the desired channel.
 * arguments:
 *  pDNPConfig - pointer to DNP channel configuration, note that some
 *   parameters in this structure will override values in the transport,
 *   link, and physical layer configurations.
 *  pTprtConfig - pointer to transport layer configuration
 *  pLinkConfig - pointer to link layer configuration
 *  pPhysConfig - pointer to physical layer configuration
 * returns:
 *  void
 */
TMWDEFS_SCL_API void TMWDEFS_GLOBAL dnpchnl_initConfig(
  DNPCHNL_CONFIG *pDNPConfig,
  DNPTPRT_CONFIG *pTprtConfig,
  DNPLINK_CONFIG *pLinkConfig,
  TMWPHYS_CONFIG *pPhysConfig);

/* function: dnpchnl_openChannel
 * purpose: Open a DNP3 channel
 * arguments:
 *  pApplContext - application context to add channel to
 *  pDNPConfig - pointer to DNP channel configuration
 *  pTprtConfig - pointer to transport layer configuration
 *  pLinkConfig - pointer to link layer configuration
 *  pPhysConfig - pointer to physical layer configuration
 *  pIOConfig - pointer to target I/O configuration data structure
 *   which is passed directly to the target routines implemented
 *   in tmwtarg.h/c
 *  pTmwTargConfig - TMW specific IO configuration information
 *   which will be passed to low level IO routines in tmwtarg.h.
 * returns:
 *  pointer to new channel, this pointer is used to reference this
 *  channel in other calls to the SCL.
 */
TMWDEFS_SCL_API TMWCHNL * TMWDEFS_GLOBAL dnpchnl_openChannel(
  TMWAPPL *pApplContext,
  DNPCHNL_CONFIG *pDNPConfig,
  DNPTPRT_CONFIG *pTprtConfig,
  DNPLINK_CONFIG *pLinkConfig,
  TMWPHYS_CONFIG *pPhysConfig,
  void *pIOConfig,
  struct TMWTargConfigStruct *pTmwTargConfig);

/* function: dnpchnl_getChannelConfig
 * purpose:  Get current configuration from an open channel
 * arguments:
 *  pSession - session to get configuration from
 *  pDNPConfig - dnp channel configuration data structure to be filled in
 *  pTprtConfig - transport configuration data structure to be filled in
 *  pLinkConfig - link configuration data structure to be filled in
 *  pPhysConfig - phys configuration data structure to be filled in
 * returns:
 *  TMWDEFS_TRUE if successful
 */
TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL dnpchnl_getChannelConfig(
  TMWCHNL *pChannel,
  DNPCHNL_CONFIG *pDNPConfig,
  DNPTPRT_CONFIG *pTprtConfig,
  DNPLINK_CONFIG *pLinkConfig,
  TMWPHYS_CONFIG *pPhysConfig);

/* function: dnpchnl_setChannelConfig
 * purpose: Modify a currently open channel
 *  NOTE: normally dnpchnl_getChannelConfig() will be called
 *   to get the current config, some values will be changed
 *   and this function will be called to set the values.
 * arguments:
 *  pChannel - channel to modify
 *  pDNPConfig - dnp channel configuration data structure
 *  pTprtConfig - transport layer configuration data structure
 *  pLinkConfig - link layer configuration data structure
 *  pPhysConfig - physical layer configuration data structure
 * returns:
 *  TMWDEFS_TRUE if successful
 */
```

```
TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL dnpchnl_setChannelConfig(
  TMWCHNL        *pChannel,
  DNPCHNL_CONFIG *pDNPConfig,
  DNPTPRT_CONFIG *pTprtConfig,
  DNPLINK_CONFIG *pLinkConfig,
  TMWPHYS_CONFIG *pPhysConfig);

/* function: dnpchnl_getBinFileChannelValues
 * purpose:
 * arguments:
 *  pDNPConfig - dnp channel configuration data structure
 *  pLinkConfig - link layer configuration data structure
 *  pPhysConfig - physical layer configuration data structure
 *  sessionIsOutstation - TMWDEFS_TRUE if session is outstation
 * returns:
 *  TMWDEFS_TRUE if successful
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL dnpchnl_getBinFileChannelValues(
  DNPCHNL_CONFIG *pDNPConfig,
  /*DNPTPRT_CONFIG *pTprtConfig,  */
  DNPLINK_CONFIG *pLinkConfig,
  /*TMWPHYS_CONFIG *pPhysConfig,  */
  DNPBNCFG_FILEVALUES *binFileValues,
  TMWTYPES_BOOL sessionIsOutstation);

/* function: dnpchnl_modifyPhys
 *  DEPRECATED FUNCTION, SHOULD USE dnpchnl_setChannelConfig()
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL dnpchnl_modifyPhys(
  TMWCHNL *pChannel,
  const TMWPHYS_CONFIG *pPhysConfig,
  TMWTYPES_ULONG configMask);

/* function: dnpchnl_modifyLink
 *  DEPRECATED FUNCTION, SHOULD USE dnpchnl_setChannelConfig()
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL dnpchnl_modifyLink(
  TMWCHNL *pChannel,
  const DNPLINK_CONFIG *pLinkConfig,
  TMWTYPES_ULONG configMask);

/* function: dnpchnl_modifyTprt
 *  DEPRECATED FUNCTION, SHOULD USE dnpchnl_setChannelConfig()
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL dnpchnl_modifyTprt(
  TMWCHNL *pChannel,
  const DNPTPRT_CONFIG *pTprtConfig,
  TMWTYPES_ULONG configMask);

/* function: dnpchnl_modifyChannel
 *  DEPRECATED FUNCTION, SHOULD USE dnpchnl_setChannelConfig()
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL dnpchnl_modifyChannel(
  TMWCHNL *pChannel,
  const DNPCHNL_CONFIG *pConfig,
  TMWTYPES_ULONG configMask);

/* function: dnpchnl_closeChannel
 * purpose: Close a previously opened channel
 * arguments:
 *  pChannel - channel to close
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL dnpchnl_closeChannel(
  TMWCHNL *pChannel);

/* function: dnpchnl_sendFragment
 * purpose: Sends a fragment to the designated channel or session
 * arguments:
 *  pTxData - fragment to send
 * returns:
 *  TMWDEFS_TRUE if fragment was successfully queued for transmission
 *  else TMWDEFS_FALSE
 * NOTE: if this function returns TMWDEFS_FALSE caller retains ownership of
 *  pTxData and should call dnpchnl_freeTxData to deallocate it.
 */
TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL dnpchnl_sendFragment(
  TMWSESN_TX_DATA *pTxData);

/* function: dnpchnl_cancelFragment
```

```
 * purpose: Cancel a previously sent fragment.
 * arguments:
 *  pTxData - fragment to cancel
 * returns:
 *  TMWDEFS_TRUE if fragment was cancelled, else TMWDEFS_FALSE
 */
TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL dnpchnl_cancelFragment(
  TMWSESN_TX_DATA *pTxData);

/* function: dnpchnl_deleteFragments
 * purpose: Remove any outstanding fragments on the channel for this session
 * arguments:
 *  pSession - pointer to session to remove fragments for
 * returns:
 *  void
 */
TMWDEFS_GLOBAL void dnpchnl_deleteFragments(
  TMWSESN *pSession);

/* function: dnpchnl_okToSend
 * purpose: Tell the dnp channel it is now OK to send requests
 * arguments:
 *  pChannel - pointer to channel
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL dnpchnl_okToSend(
  TMWCHNL *pChannel);

/* function: dnpchnl_checkForChannelIdle
 * purpose: Determine if channel is idle or if it currently
 *  has something to send or is expecting a response. If it is
 *  idle and there is an idle callback registered it will be called.
 * arguments:
 *  pChannel - channel to check for idle
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL dnpchnl_checkForChannelIdle(
  TMWCHNL *pChannel);

/* function: dnpchnl_userCallback
 * purpose: Call user callback function if one has been specified for
 *  this request
 * arguments:
 *  pChannel - channel data will be transmitted on
 *  pTxData - request
 *  pResponse - response information for this request
 * returns:
 *  void
 */
TMWDEFS_GLOBAL void dnpchnl_userCallback(
  TMWCHNL *pChannel,
  DNPCHNL_TX_DATA *pTxData,
  DNPCHNL_RESPONSE_INFO *pResponse);

/* function: dnpchnl_newTxData
 * purpose: Allocate a new transmit data structure to hold a fragment
 *  to be transmitted
 * arguments:
 *  pChannel - channel data will be transmitted on
 *  pSession - session data will be transmitted to or TMWDEFS_NULL if
 *   message is a broadcast message
 *  size - number of bytes in message
 *  destAddress - destination address
 * returns:
 *  allocated transmit data structure
 */
TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL dnpchnl_newTxData(
  TMWCHNL *pChannel,
  TMWSESN *pSession,
  TMWTYPES_USHORT size,
  TMWTYPES_USHORT destAddress);

/* function: dnpchnl_freeTxData
 * purpose: Free transmit data structure previously. This routine will
 *  generally be called by the SCL and should not be called by the user
 *  unless a previously allocated transmit data structure is canceled
 *  before it is passed to the SCL.
 * arguments:
 *  pTxData - transmit data structure to free
```

```
     * returns:
     *  void
     */
    TMWDEFS_SCL_API void TMWDEFS_GLOBAL dnpchnl_freeTxData(
      TMWSESN_TX_DATA *pTxData);

    /* function: dnpchnl_processFragment
560  * purpose: Process received fragment
     * arguments:
     *  pSession - pointer to session
     *  pRxFragment - pointer to received fragment
     * returns:
     *  void
     */
    TMWDEFS_SCL_API void TMWDEFS_CALLBACK dnpchnl_processFragment(
      TMWSESN *pSession,
      TMWSESN_RX_DATA *pRxFragment);

570
#ifdef __cplusplus
}
#endif
#endif /* DNPCHNL_DEFINED */
```

4.3 MDNP Sessions

'mdnpsesn.h' contains the definition of the routines required to configure, open and close sessions in the DNP3 Master SCL. In particular, the functions mdnpsesn_initConfig() and mdnpsesn_openSession() are called from the target application. The MDNPSESN_CONFIG structure may optionally be modified after the return from mdnpsesn_initConfig().

```
/*****************************************************************************/
/* Triangle MicroWorks, Inc.                          Copyright (c) 1997-2022 */
/*****************************************************************************/
/*                                                                           */
/* This file is the property of:                                             */
/*                                                                           */
/*                      Triangle MicroWorks, Inc.                            */
/*                     Raleigh, North Carolina USA                           */
/*                      www.TriangleMicroWorks.com                           */
/*                         (919) 870-6615                                    */
/*                                                                           */
/* This Source Code and the associated Documentation contain proprietary     */
/* information of Triangle MicroWorks, Inc. and may not be copied or         */
/* distributed in any form without the written permission of Triangle        */
/* MicroWorks, Inc.  Copies of the source code may be made only for backup   */
/* purposes.                                                                 */
/*                                                                           */
/* Your License agreement may limit the installation of this source code to  */
/* specific products.  Before installing this source code on a new           */
/* application, check your license agreement to ensure it allows use on the  */
/* product in question.  Contact Triangle MicroWorks for information about   */
/* extending the number of products that may use this source code library or */
/* obtaining the newest revision.                                            */
/*                                                                           */
/*****************************************************************************/

/* file: mdnpsesn.h
 * description: Master DNP Session Definitions
 */
#ifndef MDNPSESN_DEFINED
#define MDNPSESN_DEFINED

#include "tmwscl/utils/tmwsesn.h"
#include "tmwscl/dnp/mdnpdata.h"

#define MDNPSESN_TX_DATA_BUFFER_MAX 2048

/* Define MDNP Session Configuration Structure */
typedef TMWTYPES_USHORT MDNPSESN_AUTO_REQ;

/* The Application Layer Spec Volume 2 Version 2.10 December 2007
 * specifies some behavior the master should follow. The following allow the SCL
 * to be configured to automatically perform that behavior as well as the
 * the ability to tailor the behavior as needed. The default values will be set
 * to perform as spelled out in the application layer specification. Any changes
 * to the default behavior should be carefully considered
 */

/* Maintain current values, since some applications may save the actual values  */
#define MDNPSESN_AUTO_CLEAR_RESTART       0x0001 /* Clear IIN restart bit       */
#define MDNPSESN_AUTO_INTEGRITY_RESTART   0x0002 /* Issue integrity data poll   */
                                                 /* on restart                  */
#define MDNPSESN_AUTO_INTEGRITY_LOCAL     0x0004 /* Issue integrity data poll   */
                                                 /* after local IIN bit was set */
                                                 /* and cleared                 */
#define MDNPSESN_AUTO_INTEGRITY_TIMEOUT   0x0008 /* Issue integrity data poll   */
                                                 /* on timeout                  */
#define MDNPSESN_AUTO_INTEGRITY_OVERFLOW  0x0010 /* Issue integrity data poll   */
                                                 /* on buffer overflow          */
#define MDNPSESN_AUTO_DELAY_MEAS          0x0020 /* Use delay measurement in    */
                                                 /* time sync                   */
#define MDNPSESN_AUTO_TIME_SYNC_SERIAL    0x0040 /* Perform time sync on need   */
                                                 /* time using serial method    */
```

```
                                                    /* Define for LAN method below */
    #define MDNPSESN_AUTO_TIME_SYNC           MDNPSESN_AUTO_TIME_SYNC_SERIAL
                                                    /* For backward compatibility  */
                                                    /* keep this old define        */
    #define MDNPSESN_AUTO_EVENT_POLL          0x0080 /* Issue event data poll when  */
                                                    /* class 1, 2, or 3 IIN bit is */
 70                                                 /* set                         */
    #define MDNPSESN_AUTO_ENABLE_UNSOL        0x0100 /* automatically enable        */
                                                    /* unsolicited events upon     */
                                                    /* remote or master device     */
                                                    /* startup                     */
    #define MDNPSESN_AUTO_DISABLE_UNSOL       0x0200 /* automatically disable       */
                                                    /* unsolicited events upon     */
                                                    /* remote device startup, Not  */
                                                    /* necessary, outstation should*/
                                                    /* be disabled at startup      */
 80 #define MDNPSESN_AUTO_CONFIRM             0x0400 /* Enable/Disable automatic    */
                                                    /* generation of application   */
                                                    /* layer confirmations         */
    #define MDNPSESN_AUTO_TIME_SYNC_LAN       0x0800 /* Perform time sync on need   */
                                                    /* time using LAN method       */
    #define MDNPSESN_AUTO_DATASET_RESTART     0x1000 /* Exchange data set prototypes*/
                                                    /* and descriptors with slave  */
                                                    /* when IIN restart bit is set */
                                                    /* or when master restarts.    */
    #define MDNPSESN_AUTO_INTEGRITY_ONLINE    0x2000 /* Issue integrity data poll   */
 90                                                 /* when session becomes        */
                                                    /* "online", meaning connected */
    #define MDNPSESN_AUTO_UNSOL_STARTUP       0x4000 /* Master Unsolicited Startup  */
                          /* Behavior, see autoRequestMask comment below for details.*/


    /* DNP SECURE AUTHENTICATION Configuration
     * The following configuration structure may be ignored if Secure Authentication is
not supported
     */

100 #if DNPCNFG_SUPPORT_AUTHENTICATION
    #if MDNPCNFG_SUPPORT_SA_VERSION2
    /* MDNP Secure Authentication per user configuration structure */
    typedef struct MDNPAuthUserConfig {
      /* User Number */
      TMWTYPES_USHORT userNumber;
      void *          userHandle;
    } MDNPSESN_AUTH_USERCONFIG;
    #endif

110 /* MDNP Secure Authentication Configuration structure */
    typedef struct MDNPAuthConfig {

      /* MAC algorithm to sent in challenge requests sent by Master.
       * Outstation will use this algorithm to generate the MAC value in the challenge
       * reply. The Master will then use this same algorithm to generate a MAC to
       * authenticate the challenge reply. The MAC algorithm the Outstation is
configured
       * to send in a challenge does NOT have to match the MAC Algorithm the Master
       * sends in a challenge.
       *  1 DNPAUTH_HMAC_SHA1_4OCTET   (SHA1 truncated to    4 octets   Only for SA V2
and backward compatibility )
120    *  2 DNPAUTH_MAC_SHA1_10OCTET   (SHA1 truncated to   10 octets)
       *  3 DNPAUTH_MAC_SHA256_8OCTET  (SHA256 truncated to  8 octets)
       *  4 DNPAUTH_MAC_SHA256_16OCTET (SHA256 truncated to 16 octets)
       *  5 DNPAUTH_MAC_SHA1_8OCTET    (SHA1 truncated to    8 octets)
       *  6 DNPAUTH_MAC_AESGMAC_12OCTET
       *  6-127 reserved for future,
       *  128-255 reserved for vendor specific choices
       */
      TMWTYPES_UCHAR   MACAlgorithm;

130   /* Number of consecutive application layer timeouts before declaring a
       * Communications Failure Event. You may choose to set this to the same
       * value as maxErrorCount in SAv2 or Max Reply Timeout statistic in SAv5,
       * but this is separately configured and counted.
       */
      TMWTYPES_UCHAR  maxApplTimeoutCount;

      /* length of session keys to be generated and sent to the outstation */
      TMWTYPES_UCHAR   sessionKeyLength;

140   /* keyWrapAlgorithm to be used to encrypt and "wrap" session keys is received
       * from outstation in key status message and is not configurable on master
```

```
        */

       /* How long to wait for any authentication reply
        * default shall be 2 seconds
        */
       TMWTYPES_ULONG   responseTimeout;

       /* Session key interval in milliseconds.
150     * When time since last key change reaches this, session keys will be updated.
        * For systems that communicate infrequently,
        * this may be set to zero, using only the maxKeyChangeCount to determine
        * when to update keys.
        *  Spec says range should be up to 1 week
        *  default 15 minutes
        */
       TMWTYPES_MILLISECONDS keyChangeInterval;

       /* Session Authentication ASDU count since last key change, When this number
160     *  of authentication ASDUs is transmitted or rcved since last key change,
        *    Session keys will be updated.
        *  default shall be 1000 authentication ASDUs transmitted or received
        */
       TMWTYPES_USHORT  maxKeyChangeCount;

       /* Sent in error message.
        * More meaningful when outstation sends error message to master.
        * Not very useful when sent by master to outstation.
        */
170    TMWTYPES_USHORT assocId;

       /* Agressive mode is enabled */
       TMWTYPES_BOOL   aggressiveModeSupport;

       /* Bit mask to indicate which auto requests should be sent using Aggressive Mode
        * Depends on aggressiveModeSupport==TMWDEFS_TRUE and which autoRequestMask bits
        * are set. This mask uses the same bits that autoRequestMask uses.
        * For example if autoRequestMask has MDNPSESN_AUTO_TIME_SYNC_LAN bit set
        * and autoRequestAggrModeMask has the same bit MDNPSESN_AUTO_TIME_SYNC_LAN set,
180     * when a LAN time sync is automatically sent the library will use Aggressive Mode
        * if it can.
        * NOTE: This will NOT send auto integrity polls, auto application confirm or
        * auto data set read using Aggressive Mode even if those bits are set.
        * The following values (or OR'd combinations) are valid for this type:
        *  MDNPSESN_AUTO_CLEAR_RESTART = Clear IIN restart bit
        *  MDNPSESN_AUTO_INTEGRITY_RESTART = Not Supported
        *  MDNPSESN_AUTO_INTEGRITY_LOCAL = Not Supported
        *  MDNPSESN_AUTO_INTEGRITY_TIMEOUT = Not Supported
        *  MDNPSESN_AUTO_INTEGRITY_OVERFLOW = Not Supported
190     *  MDNPSESN_AUTO_DELAY_MEAS = Not Supported
        *  MDNPSESN_AUTO_TIME_SYNC_SERIAL = Perform time sync on need time using serial
method
        *  MDNPSESN_AUTO_EVENT_POLL = Automatic Event Poll when Class when class 1, 2, or
3 IIN bit is set
        *  MDNPSESN_AUTO_ENABLE_UNSOL = automatically enable unsolicited events upon
remote or master device startup
        *  MDNPSESN_AUTO_DISABLE_UNSOL = automatically disable unsolicited events upon
remote device startup, Not necessary, outstation should be disabled at startup
        *  MDNPSESN_AUTO_CONFIRM = Not Supported
        *  MDNPSESN_AUTO_TIME_SYNC_LAN = Perform time sync on need time using LAN method
        *  MDNPSESN_AUTO_DATASET_RESTART = Not Supported
        *  MDNPSESN_AUTO_INTEGRITY_ONLINE = Not Supported
        *  MDNPSESN_AUTO_UNSOL_STARTUP = Master Unsolicited Startup Behavior
200     */
       MDNPSESN_AUTO_REQ autoRequestAggrModeMask;

       /* Version 5 requires ability to disallow SHA1 */
       TMWTYPES_BOOL   disallowSHA1;

       /* Extra diagnostics including plain key data before it is encrypted or after it
is decrypted */
       TMWTYPES_BOOL   extraDiags;

       /* extra configuration to assist in testing. */
210    TMWTYPES_ULONG        testConfig;

       /* Secure Authentication Version 2 */
       TMWTYPES_BOOL   operateInV2Mode;

       /* SA Version 5 says master must either send aggressive mode or delay after
        * sending direct no ack in case the request is challenged. This is how long
        * the master will delay before sending next request.
```

```
       */
      TMWTYPES_MILLISECONDS directNoAckDelayTime;

    #if MDNPCNFG_SUPPORT_SA_VERSION2
      /* Number of errors messages to be sent before disabling error message
transmission,
       * default shall be 2
       * range 0-255
       */
      /* This has been replaced with a statistic in Secure Authentication version 4 */
      TMWTYPES_UCHAR  maxErrorCount;

      /* Configuration for each user
       * Specification says default user number is 1, configure it as first user in
array.
       * Add any other user numbers. For each user number the database must contain an
       * update key.
       */
      MDNPSESN_AUTH_USERCONFIG authUsers[DNPCNFG_AUTHV2_MAX_NUMBER_USERS];
    #endif

    #if MDNPCNFG_SUPPORT_SA_VERSION5

      /* Length of random challenge data to send in g120v1 challenge request.
       * TB2016-002 says Minimum length==4, Maximum length==64
       */
      TMWTYPES_USHORT  randomChallengeDataLength;

      /* These 5 maximum values are used to determine special actions when the related
statistics
       * counts exceed these values. Reporting thresholds which are used on the
outstation are not
       * applicable to the master.
       */
      TMWTYPES_USHORT  maxAuthenticationFailures;
      TMWTYPES_USHORT  maxReplyTimeouts;
      TMWTYPES_USHORT  maxAuthenticationRekeys;
      TMWTYPES_USHORT  maxErrorMessagesSent;
      TMWTYPES_USHORT  maxRekeysDueToRestarts;

      /* Master should treat all responses or unsoliciteds as critical and challenge
them */
      TMWTYPES_BOOL   authCriticalResponses;
      TMWTYPES_BOOL   authCriticalUnsoliciteds;
    #endif

    } MDNPSESN_AUTH_CONFIG;
    #endif

    /* MDNP Session Configuration Structure */
    typedef struct MDNPSessionConfigStruct {

      /* Source address (master address) for this session */
      TMWTYPES_USHORT source;

      /* Destination address (outstation address) for this session */
      TMWTYPES_USHORT destination;

      /* How often to send link status requests
       * if no DNP3 frames have been received on this session.
       * In DNP3 IP Networking spec this is called keep-alive interval
       * Enabling keep-alives is REQUIRED when using TCP
       * A value of zero will turn off keep alives.
       */
      TMWTYPES_MILLISECONDS linkStatusPeriod;

      /* Disconnect/reconnect a connection when link status request times out.
       * The spec says to do this when using TCP. However when configuring multiple
       * sessions over a single TCP channel to multiple serial devices you may not
       * want to disconnect when an individual session does not respond. You may still
       * want to determine whether individual Outstations are responding.
       */
      TMWTYPES_BOOL linkStatusTimeoutDisconnect;

      /* Determine whether the session is to be active or inactive. An inactive
       * session will not transmit or receive frames.
       */
      TMWTYPES_BOOL active;

      /* Mask used to enable/disable automatic processing of certain requests
       * based on internal events. If changing the default value of this field
```

```
     * explicitly set ALL of the bits that are desired, but only those bits.
     * For example:
     *   MDNPSESN_CONFIG config;
     *   mdnpsesn_initConfig(&config);
     *    Explicitly set the desired automatic behavior for this session
     *   config.autoRequestMask = MDNPSESN_AUTO_CLEAR_RESTART
300  *     | MDNPSESN_AUTO_DISABLE_UNSOL
     *     | MDNPSESN_AUTO_CONFIRM;
     *
     * Note: Setting both the
     *   MDNPSESN_AUTO_DISABLE_UNSOL and MDNPSESN_AUTO_ENABLE_UNSOL bits
     *   will cause just a DISABLE UNSOLICITED request to be sent followed
     *   by an ENABLE UNSOLICITED request. If  MDNPSESN_AUTO_CLEAR_RESTART
     *   and/or MDNPSESN_AUTO_INTEGRITY_RESTART are set, they will be sent
     *   after the DISABLE UNSOLICITED and before the ENABLE UNSOLICITED
     *   requests.
310  *
     *   Application Layer Spec Part 2 Volume 2 Version 2.10 December 2007
     *   Section 1.1.2 Master Startup and Part 3, Section 1.5 and Table 1.3
     *   Master Unsolicited Response State Table says:
     *    Send Disable Unsolicited Request,
     *    Send an integrity poll and discard any unsolicited
     *      responses until after the integrity poll completes.
     *    Send class assignments and analog deadbands if desired.
     *    Enable unsolicited responses
     *   If MDNPSESN_AUTO_UNSOL_STARTUP - An automatic disable unsolicited request
320  *    will be sent to the outstation. Unsolicited responses will be discarded
     *    and not confirmed until an integrity poll is completed. This is required
     *    by the application layer spec.
     *    This includes initial unsolicited NULL responses. To process unsolicited
     *    null responses even before the integrity poll is complete, set
     *    MDNPDATA_DISCARD_NULL_UNSOL define to TMWDEFS_FALSE at compile time.
     *   The following configuration also controls the automatic exchanges on master
     *    restart:
     *    If MDNPSESN_AUTO_DATASET_RESTART -An automatic exchange of data set
     *      prototypes and descriptors will be performed. Setting this causes the
330  *      exchange to take place before an integrity poll and before unsoliciteds
     *      are enabled. This helps assure that data set descriptors are present
     *      before events can be received.
     *    If MDNPSESN_AUTO_INTEGRITY_ONLINE -An integrity poll will be performed.
     *    If MDNPSESN_AUTO_ENABLE_UNSOL -An enable unsolicited will be issued using
     *       autoEnableUnsolClass1,2,3 to determine which classes to enable.
     *
     *  If the application needs to send class assignments, analog deadbands or other
     *    commands before the Integrity Poll or Enable Unsolicited, the application
     *    can clear MDNPSESN_AUTO_INTEGRITY_ONLINE and/or MDNPSESN_AUTO_ENABLE_UNSOL
340  *    and queue the desired sequence of commands in the order and with the
     *    priority desired.
     *
     */
    MDNPSESN_AUTO_REQ autoRequestMask;

    /* If MDNPSESN_AUTO_ENABLE_UNSOL is set, these three flags will indicate which
     * event classes should be enabled for unsolicited reporting
     */
    TMWTYPES_BOOL autoEnableUnsolClass1;
350 TMWTYPES_BOOL autoEnableUnsolClass2;
    TMWTYPES_BOOL autoEnableUnsolClass3;

    /* Default absolute response timeout. This is the default value for the
     * response timeout for any request generated on this session. This value
     * can be overridden on a per command basis by changing the responseTimeout
     * value in the MDNPBRM_REQ_DESC data structure. This value is the absolute
     * maximum amount of time this device will wait for the final response to
     * a request. This time starts as soon as the request is put into the
     * transmit queue.
360  */
    TMWTYPES_MILLISECONDS defaultResponseTimeout;

    /* Amount of time to delay after receiving a response to a select request
     * before sending any request other than an operate. If MDNPBRM_AUTO_MODE
     * operate is used, the mdnp library will not send a request before the
     * operate request. If however, a user application sends a select request
     * without using MDNPBRM_AUTO_MODE operate and autoRequestMask is non zero,
     * or Secure Authentication is enabled, the mdnp library could send a request
     * before the user application called the brm function to send the operate
request.
370  * Setting this to a nonzero value will delay the automatic sending of requests
     * by this master library for up to that many milliseconds.
     * A value of zero means no delay will be enforced by the library.
     * Call mdnpbrm_cancelSelOpDelayTimer() to cancel the delay timer early.
```

```
     */
    TMWTYPES_MILLISECONDS selOpDelayTime;

    /* Number of times a read request is allowed to timeout before the session
     * is considered offline. This is normally set to 0.
     * Setting this to a value larger than 0 can result in failure to recognize stale
     * values (i.e., data can be stale by n*polling interval)
     */
    TMWTYPES_UCHAR  readTimeoutsAllowed;

    /* Diagnostic mask */
    TMWTYPES_ULONG  sesnDiagMask;

  #if MDNPDATA_SUPPORT_OBJ91
    /* If multiple file objects are sent in Activate Config request combine
     * them to use just a single object header. Some outstations may not support
     * this even though allowed by the spec.
     */
    TMWTYPES_BOOL   combineActConfigData;
  #endif

  #if MDNPDATA_SUPPORT_OBJ70
    /* Maximimum block size to use during file transfer when
     * mdnpbrm_copyRemoteFileToLocal or mdnpbrm_copyLocalFileToRemote
     * functions are used. This must be less than max tx or rx fragment
     * size minus 16. Max size can be specified in mdnpbrm_fileOpen if
     * application is managing file transfer block by block.
     */
    TMWTYPES_USHORT maxFileBlockSize;
  #endif

    /* User registered statistics callback function and parameter */
    TMWSESN_STAT_CALLBACK pStatCallback;
    void *pStatCallbackParam;

  #if MDNPDATA_SUPPORT_OBJ120
    TMWTYPES_BOOL         authenticationEnabled;
    MDNPSESN_AUTH_CONFIG authConfig;
  #endif

  } MDNPSESN_CONFIG;

  /* DEPRECATED SHOULD USE mdnpsesn_getSessionConfig and
   *  mdnpsesn_setSessionConfig
   */
  #define MDNPSESN_CONFIG_SOURCE        0x00000001L
  #define MDNPSESN_CONFIG_DESTINATION   0x00000002L
  #define MDNPSESN_CONFIG_ACTIVE        0x00000004L
  #define MDNPSESN_CONFIG_AUTO          0x00000008L
  #define MDNPSESN_CONFIG_RESP_TIMEOUT  0x00000010L

  /* Define support for an unsolicited response callback. This callback
   *  function is specified per session and is called whenever the
   *  session receives an unsolicited response.
   */
  typedef struct MDNPSessionUnsolRespStruct {
    TMWSESN *pSession;
    TMWSESN_RX_DATA *pRxData;
    DNPCHNL_RESP_STATUS status;
    TMWTYPES_USHORT iin;
  } MDNPSESN_UNSOL_RESP_INFO;

  /* Define the unsolicited callback method */
  typedef void (*MDNPSESN_UNSOL_CALLBACK_FUNC)(
    void *pCallbackParam,
    MDNPSESN_UNSOL_RESP_INFO *pResponse);

  typedef struct MDNPSessionSARespStruct {
    TMWSESN *pSession;
    TMWSESN_RX_DATA *pRxData;
    TMWTYPES_UCHAR status;
    TMWTYPES_USHORT iin;
  } MDNPSESN_SA_RESP_INFO;

  /* Define the SA callback method */
  typedef void(*MDNPSESN_SA_CALLBACK_FUNC)(
    void *pCallbackParam,
    MDNPSESN_SA_RESP_INFO *pResponse);
```

```
    /* Include master DNP3 'private' structures and functions */
    #include "tmwscl/dnp/mdnpsesp.h"

    #ifdef __cplusplus
    extern "C" {
460 #endif

      /* function: mdnpsesn_initConfig
       * purpose: Initialize DNP3 master session configuration data structure.
       *  This routine should be called to initialize all the members of the
       *  data structure. Then the user should modify the data members they
       *  need in user code. Then pass the resulting structure to
       *  mdnpsesn_openSession.
       * arguments:
       *  pConfig - pointer to configuration data structure to initialize
470    * returns:
       *  void
       */
      TMWDEFS_SCL_API void TMWDEFS_GLOBAL mdnpsesn_initConfig(
        MDNPSESN_CONFIG *pConfig);


      /* function: mdnpsesn_applyBinaryFileValues
       purpose: Applies values found in a binary configuration file to DNP3 channel
       *  configuration data structures and session configuration data structures.
480    *  The structures must first be initialized to default values using
dnpchnl_initConfig()
       *  and mdnpsesn_initConfig().  After the binary file values are applied call
dnpchnl_openChannel
       *  to actually open the desired channel and mdnpsesn_openSession to open a
session
       *  on the channel.
       * arguments:
       *  pFileName - full path to the DNP3 binary configuration file
       *  pDNPConfig - pointer to DNP channel configuration, note that some
       *   parameters in this structure will override values in the transport,
       *   link, and physical layer configurations.
       *  pLinkConfig - pointer to link layer configuration
490    *  pIOConfig - pointer to target layer configuration
       *  pSesnConfig - pointer to configuration session configuration
       * returns:
       *  TMWDEFS_TRUE if successful
       */
      TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpsesn_applyBinaryFileValues (
        char * pFileName,
        DNPCHNL_CONFIG *pDNPConfig,
        DNPLINK_CONFIG *pLinkConfig,
        void *pIOConfig,
500     MDNPSESN_CONFIG *pSesnConfig);

      /* function: mdnpsesn_openSession
       * purpose: Open and DNP3 master session
       * arguments:
       *  pChannel - channel to open session on
       *  pConfig - DNP3 master configuration data structure
       *  pUserHandle - handle passed to session database initialization routine
       * returns:
       *  Pointer to new session or TMWDEFS_NULL.
510    */
      TMWDEFS_SCL_API TMWSESN * TMWDEFS_GLOBAL mdnpsesn_openSession(
        TMWCHNL *pChannel,
        const MDNPSESN_CONFIG *pConfig,
        void *pUserHandle);

      /* function: mdnpsesn_getSessionConfig
       * purpose:  Get current configuration from an open session
       * arguments:
       *  pSession - session to get configuration from
520    *  pConfig - dnp master configuration data structure to be filled in
       * returns:
       *  TMWDEFS_TRUE if successful
       */
      TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpsesn_getSessionConfig(
        TMWSESN *pSession,
        MDNPSESN_CONFIG *pConfig);

      /* function: mdnpsesn_setSessionConfig
       * purpose: Modify a currently open session
530    *  NOTE: normally mdnpsesn_getSessionConfig() will be called
       *   to get the current config, some values will be changed
       *   and this function will be called to set the values.
```

```
     * arguments:
     *  pSession - session to modify
     *  pConfig - dnp master configuration data structure
     * returns:
     *  TMWDEFS_TRUE if successful
     */
    TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpsesn_setSessionConfig(
      TMWSESN *pSession,
      const MDNPSESN_CONFIG *pConfig);

    /* function: mdnpsesn_modifySession
     *  DEPRECATED FUNCTION, SHOULD USE mdnpsesn_setSessionConfig()
     */
    TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpsesn_modifySession(
      TMWSESN *pSession,
      const MDNPSESN_CONFIG *pConfig,
      TMWTYPES_ULONG configMask);

    /* function: mdnpsesn_closeSession
     * purpose: Close a currently open session
     * arguments:
     *  pSession - session to close
     * returns:
     *  TMWDEFS_TRUE if successfull, else TMWDEFS_FALSE
     */
    TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpsesn_closeSession(
      TMWSESN *pSession);

    /* function: mdnpsesn_sendConfirmation
     * purpose: Send and application layer confirmation to the specified
     *  session. Note that the TMW SCL application layer generally performs
     *  this action automatically but this feature can be disabled by changing
     *  the session's autoRequestMask configuration parameter.
     * arguments:
     *  pSession - session to send confirmation to
     *  seqNumber - sequence number to put in message
     *  unsol - TMWDEFS_TRUE if this is a confirmation to an unsolicited
     *   response
     * returns:
     */
    TMWDEFS_SCL_API void TMWDEFS_GLOBAL mdnpsesn_sendConfirmation(
      TMWSESN *pSession,
      TMWTYPES_UCHAR seqNumber,
      TMWTYPES_BOOL unsol);

    /* function: mdnpsesn_getSequenceNumber
     * purpose: Retrieve the current response sequence number. In general
     *  the TMW SCL handles sequence number internally but this function is
     *  provided for the user to access the current application sequence
     *  nunmber.
     * arguments:
     *  pSession - session to get the sequence number from
     * returns:
     *  the current application layer sequence number
     */
    TMWDEFS_SCL_API TMWTYPES_UCHAR TMWDEFS_GLOBAL mdnpsesn_getSequenceNumber(
      TMWSESN *pSession);

    /* function: mdnpsesn_setSequenceNumber
     * purpose: Set the current response sequence number. In general the
     *  TMW SCL handles sequence number internally but this function is
     *  provided for the user to access the current application sequence
     *  nunmber.
     * arguments:
     *  pSession - session to set the sequence number on
     *  seqNumber - the new sequence number
     * returns:
     *  void
     */
    TMWDEFS_SCL_API void TMWDEFS_GLOBAL mdnpsesn_setSequenceNumber(
      TMWSESN *pSession,
      TMWTYPES_UCHAR seqNumber);

    /* function: mdnpsesn_setUnsolUserCallback
     * purpose: Specify a user callback to be invoked whenever an unsolicited
     *  response is received from the remote device.
     * arguments:
     *  pSession - session to add callback to
     *  pCallback - pointer to user callback function
     *  pCallbackParam - pointer to user supplied callback parameter
     * returns:
```

```
     *  void
     */
    TMWDEFS_SCL_API void TMWDEFS_GLOBAL mdnpsesn_setUnsolUserCallback(
      TMWSESN *pSession,
      MDNPSESN_UNSOL_CALLBACK_FUNC pCallback,
      void *pCallbackParam);

    /* function:mdnpsesn_cancelSelOpDelayTimer
     * purpose:
     *  If the application is going to be delayed sending the OPERATE command (for
     *  example waiting for human input) you can configure a nonzero mdnp session
     *  selOpDelayTime to delay the sending of other commands until the OPERATE is
sent
     *  or the timeout is reached. This function will cancel that timer if it is
running so
     *  that other commands can be sent.
     * arguments:
     *  pSession - pointer to a master DNP session
     * returns:
     *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
     */
    TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpsesn_cancelSelOpDelayTimer(
      TMWSESN *pSession);

    /* Secure Authentication code */

    /* Secure Authentication Version 5 code */

    /* function: mdnpsesn_addAuthUser
     * purpose: Add a Secure Authentication User
     * arguments:
     *  pSession - session
     *  userNumber - user number for this user
     * returns:
     *  TMWDEFS_TRUE if successfull, else TMWDEFS_FALSE
     */
    TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpsesn_addAuthUser(
      TMWSESN *pSession,
      TMWTYPES_USHORT userNumber);

    /* function: mdnpsesn_getAuthUser
     * purpose: Get the Secure Authentication User Number from SCL for specified
index.
     * arguments:
     *  pSession - session
     *  index - index of user to return user number for.
     * returns:
     *  userNumber or 0 if not found
     */
    TMWDEFS_SCL_API TMWTYPES_USHORT TMWDEFS_GLOBAL mdnpsesn_getAuthUser(
      TMWSESN *pSession,
      TMWTYPES_USHORT index);

    /* function: mdnpsesn_removeAuthUser
     * purpose: Remove a Secure Authentication User from SCL
     * arguments:
     *  pSession - session
     *  userNumber - user number to remove
     * returns:
     *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
     */
    TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpsesn_removeAuthUser(
      TMWSESN *pSession,
      TMWTYPES_USHORT userNumber);

    /* function: mdnpsesn_setSAUnsolCallback
     * purpose: Specify a user callback to be invoked whenever an SA
     *  response is received from the remote device.
     * arguments:
     *  pSession - session to add callback to
     *  pCallback - pointer to user callback function
     *  pCallbackParam - pointer to user supplied callback parameter
     * returns:
     *  void
     */
    TMWDEFS_SCL_API void TMWDEFS_GLOBAL mdnpsesn_setSAUserCallback(
      TMWSESN *pSession,
      MDNPSESN_SA_CALLBACK_FUNC pCallback,
      void *pCallbackParam);

    /* function: mdnpsesn_getAuthCSQ
```

```
        * purpose: Get the current SA Challenge Sequence number sent in a challenge
g120v1 request by the master.
        * It will be incremented before the next challenge is sent.
        * The TMW SCL manages the sequence number internally but this function is
        * provided for the user to access the current sequence number for test purposes.
        */
       TMWDEFS_SCL_API TMWTYPES_ULONG TMWDEFS_GLOBAL mdnpsesn_getAuthCSQ(
         TMWSESN *pSession);

700    /* function: mdnpsesn_setAuthCSQ
        * purpose:  Set the current SA Challenge Sequence number sent in a challenge
g120v1 request by the master.
        * It will be incremented before the next challenge is sent.
        * The TMW SCL manages the sequence number internally but this function is
        * provided for the user to access the current sequence number for test purposes.
        */
       TMWDEFS_SCL_API void TMWDEFS_GLOBAL mdnpsesn_setAuthCSQ(
         TMWSESN *pSession,
         TMWTYPES_ULONG csq);

710    /* function: mdnpsesn_getAuthAggrCSQ
        * purpose: Get the current SA Challenge Sequence Number sent in an aggressive
mode request by the master.
        * It will be incremented before the next aggressive mode is sent.
        * The TMW SCL manages the sequence number internally but this function is
        * provided for the user to access the current sequence number for test purposes.
       */
       TMWDEFS_SCL_API TMWTYPES_ULONG TMWDEFS_GLOBAL mdnpsesn_getAuthAggrCSQ(
         TMWSESN *pSession);

       /* function: mdnpsesn_setAuthAggrCSQ
720     * purpose:  Set the current SA Challenge Sequence Number sent in an aggressive
mode request by the master.
        * It will be incremented before the next aggressive mode request is sent.
        * The TMW SCL manages the sequence number internally but this function is
        * provided for the user to access the current sequence number for test purposes.
       */
       TMWDEFS_SCL_API void TMWDEFS_GLOBAL mdnpsesn_setAuthAggrCSQ(
         TMWSESN *pSession,
         TMWTYPES_ULONG csq);


730 #ifdef __cplusplus
    }
    #endif
    #endif /* MDNPSESN_DEFINED */
```

4.4 Timer

'tmwtimer.h' contains the definition of the routines required to manage timers in all TMW SCLs. In particular, the functions tmwtimer_initialize() should be called from the target application if the default implementation (using a single timer queue) is used. Note that this is the case if the polled timer (defined in tmwpltmr.h) is used.

```
/*****************************************************************************/
/* Triangle MicroWorks, Inc.                          Copyright (c) 1997-2022 */
/*****************************************************************************/
/*                                                                           */
/* This file is the property of:                                             */
/*                                                                           */
/*                       Triangle MicroWorks, Inc.                           */
/*                      Raleigh, North Carolina USA                          */
/*                       www.TriangleMicroWorks.com                          */
/*                          (919) 870-6615                                   */
/*                                                                           */
/* This Source Code and the associated Documentation contain proprietary     */
/* information of Triangle MicroWorks, Inc. and may not be copied or         */
/* distributed in any form without the written permission of Triangle        */
/* MicroWorks, Inc.  Copies of the source code may be made only for backup   */
/* purposes.                                                                 */
/*                                                                           */
/* Your License agreement may limit the installation of this source code to  */
/* specific products.  Before installing this source code on a new           */
/* application, check your license agreement to ensure it allows use on the  */
/* product in question.  Contact Triangle MicroWorks for information about   */
/* extending the number of products that may use this source code library or */
/* obtaining the newest revision.                                            */
/*                                                                           */
/*****************************************************************************/

/* file: tmwtimer.h
 * description:
 */
#ifndef TMWTIMER_DEFINED
#define TMWTIMER_DEFINED

#include "tmwscl/utils/tmwcnfg.h"
#include "tmwscl/utils/tmwdefs.h"
#include "tmwscl/utils/tmwdlist.h"
#include "tmwscl/utils/tmwappl.h"

/* The following allows timeouts to extend to one month (31 days).
 * The millisecond timer actually rolls over every 49 days, but
 * 31 is the maximum because of the math calculations in this file.
 */
#define TMWTIMER_MAX_SOFT_TIMER_DELAY  TMWDEFS_DAYS(31UL)

/* forward declaration */
struct TMWChannelStruct;

/* Timer structure */
typedef struct TMWTimer
{
  /* List Member, must be first entry */
  TMWDLIST_MEMBER listMember;

  /* If TRUE this timer is running */
  TMWTYPES_BOOL active;

  /* Time that this timer will expire */
  TMWTYPES_MILLISECONDS timeout;

  /* Channel that owns this timer, for multithreaded locking purposes */
  struct TMWChannelStruct *pChannel;

  /* Function to be called when this timer expires.
   * If TMWDEFS_NULL, no function will be called back
   */
  TMWTYPES_CALLBACK_FUNC pCallback;

  /* Parameter to be passed to timer expiration callback function */
  void *pCallbackParam;
```

```
} TMWTIMER;

#if TMWCNFG_MULTIPLE_TIMER_QS

/* Multiple timer queues exist. This is useful if a multithreaded architecture
 * exists with a single channel per thread and the ability to process timers
 * for a specific channel only when the timer callback function is called is
 * required. This will require a separate system timer per thread (channel).
 * In this case the standard polled timer implementation cannot be used since
 * it only supports a single timer.
 */
typedef struct TMWTimerQueue
{

  /* List of SCL timers that are running for this channel */
  TMWDLIST              list;

  /* No need for a separate lock, the channel lock will be sufficient */

  /* If true the tmwtarg timer is running and will call _timerCallback */
  TMWTYPES_BOOL         timerRunning;
} TMWTIMER_QUEUE;
#endif


#ifdef __cplusplus
extern "C"
{
#endif

#if !TMWCNFG_MULTIPLE_TIMER_QS
  /* function: tmwtimer_initialize
   * purpose: initialize the timer code. This should be called once by
   *  by customer target application, before any timers are started.
   *  This function is used if the default implementation using a
   *  single timer queue for all channels is used. This is the case
   *  if the polled timer tmwpltmr.c is used.
   * arguments:
   *   void
   * returns:
   *  void
   */
  TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtimer_initialize(void);
  /* function: tmwtimer_close
   * purpose: closes the timer. This should be called once by
   * customer target application at shutdown
  */
  TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtimer_close(void);


  /* Internal function called by tmwappl code */
  void TMWDEFS_GLOBAL tmwtimer_applInit(TMWAPPL *pApplContext);

  /* Get the maximum number of timer elements that were ever in the timer queue.
   * For performance analysis
   */
  TMWDEFS_SCL_API TMWTYPES_UINT TMWDEFS_GLOBAL tmwtimer_getHighWater(void);

#else

  /* Internal function, called by SCL */
  /* function: tmwtimer_initMultiTimer
   * purpose: Initialize the per channel timer queue for the specified channel.
   *  This will be called by the SCL when a channel is opened if
   *  TMWCNFG_MULTIPLE_TIMER_QS which creates a separate timer queue per channel
   *  is set to TMWDEFS_TRUE.
   * arguments:
   *   pChannel - pointer to the channel.
   * returns:
   *   void
   */
  void TMWDEFS_GLOBAL tmwtimer_initMultiTimer(
    struct TMWChannelStruct *pChannel);

  /* Internal function called by tmwappl code */
  void TMWDEFS_GLOBAL tmwtimer_applInit(TMWAPPL *pApplContext);

#endif

  /* Internal function, called by SCL */
```

```
150   /* function: tmwtimer_init
       * purpose: initialize a timer structure
       *   This will set the timer to not active. This must be
       *   called once on each TMWTIMER structure before it is started.
       *   for the first time.
       * arguments:
       *  pTimer - pointer to caller provided timer structure
       * returns:
       *  void
       */
160   TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtimer_init(
        TMWTIMER *pTimer);

      /* Internal function, called by SCL */
      /* function: tmwtimer_isActive
       * purpose: check to see if this timer is active (running)
       * arguments:
       *  pTimer - pointer to caller provided timer structure
       * returns:
       *  void
170    */
      TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL tmwtimer_isActive(
        TMWTIMER *pTimer);

      /* Internal function, called by SCL */
      /* function: tmwtimer_start
       * purpose: Start a timer.
       * arguments:
       *  pTimer - pointer to caller provided timer structure
       *  msTimeout - how many milliseconds before timeout
180    *   NOTE: maximum of TMWDEFS_DAYS(31UL) (2678400000) (0x9fa52400),
       *   exceeding this may cause timer to expire immediately.
       *  pChannel - channel that this timer applies to (for multithreading support)
       *  pCallback - callback function to call when this timer expire
       *  pCallbackParam -  parameter for callback function
       * returns:
       *  void
       */
      TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtimer_start(
        TMWTIMER *pTimer,
190     TMWTYPES_MILLISECONDS msTimeout,
        struct TMWChannelStruct *pChannel,
        TMWTYPES_CALLBACK_FUNC pCallback,
        void *pCallbackParam);

      /* Internal function, called by SCL */
      /* function: tmwtimer_cancel
       * purpose: cancel a timer.
       * arguments:
       *  pTimer - pointer to caller provided timer structure
200    * returns:
       *  void
       */
      TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtimer_cancel(
        TMWTIMER *pTimer);


    #ifdef __cplusplus
    }
    #endif
210 #endif /* TMWTIMER_DEFINED */
```

4.5 Polled Timer

'tmwpltmr.h' contains the definition of routines required for the polled timer implementation. This code is required only if the platform does not support event timer. If the polled timer is used, the target application must call tmwpltmr_startTimer() and tmwpltmr_cancelTimer() from the tmwtarg_startTimer() and tmwtarg_cancelTimer() functions. Also, the target application must periodically call tmwpltmr_checkTimer() to determine if any timers have expired.

```
/*****************************************************************************/
/* Triangle MicroWorks, Inc.                          Copyright (c) 1997-2022 */
/*****************************************************************************/
/*                                                                           */
/* This file is the property of:                                             */
/*                                                                           */
/*                        Triangle MicroWorks, Inc.                          */
/*                      Raleigh, North Carolina USA                          */
/*                       www.TriangleMicroWorks.com                          */
/*                          (919) 870-6615                                   */
/*                                                                           */
/* This Source Code and the associated Documentation contain proprietary     */
/* information of Triangle MicroWorks, Inc. and may not be copied or         */
/* distributed in any form without the written permission of Triangle        */
/* MicroWorks, Inc.  Copies of the source code may be made only for backup   */
/* purposes.                                                                 */
/*                                                                           */
/* Your License agreement may limit the installation of this source code to  */
/* specific products.  Before installing this source code on a new           */
/* application, check your license agreement to ensure it allows use on the  */
/* product in question.  Contact Triangle MicroWorks for information about   */
/* extending the number of products that may use this source code library or */
/* obtaining the newest revision.                                            */
/*                                                                           */
/*****************************************************************************/

/* file: tmwpltmr.h
 * description: Implement a polled timer. This code is only required
 *  if the platform does not support event timers. To use this code
 *  call the tmwpltmr_startTimer and tmwpltmr_cancelTimer functions
 *  from the tmwtarg_startTimer and tmwtarg_cancelTimer implementation.
 *  Then periodically call the tmwpltmr_checkTimer routine to determine
 *  if the timer has expired.
 */
#ifndef TMWPLTMR_DEFINED
#define TMWPLTMR_DEFINED

#include "tmwscl/utils/tmwcnfg.h"
#include "tmwscl/utils/tmwdefs.h"
#include "tmwscl/utils/tmwtypes.h"
#include "tmwscl/utils/tmwchnl.h"

/* The following allows timeouts to extend to one month (31 days). Because
 * the millisecond timer rolls over every 48 days, the software timer must
 * checked (to determine if it has elapsed) at least as often as once every
 * 17 days.
 */
#define TMWPLTMR_MAX_SOFT_TIMER_DELAY  TMWDEFS_DAYS(31UL)

#ifdef __cplusplus
extern "C"
{
#endif

  /* function: tmwpltmr_checkTimer
   * purpose: Check to see if polled timer has expired. This function should
   *  be called periodically if the tmwtarg timer implementation uses this polled
   *  timer functionality.
   * arguments:
   *  void
   * returns:
   *  void
   */
  TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwpltmr_checkTimer(void);
```

```
      /* Internal function, called by default tmwtarg_startTimer() implementation */
      TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwpltmr_startTimer(
        TMWTYPES_MILLISECONDS msTimeout,
        TMWTYPES_CALLBACK_FUNC pCallback,
70      void *pCallbackParam);

      /* Internal function, called by default tmwtarg_cancelTimer() implementation */
      TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwpltmr_cancelTimer(void);

      /* function: tmwpltmr_checkMultiTimer
       * purpose: Called by application code if MULTI TIMER support but using polled
timers.
       *  Check to see if polled timer queue on a channel has expired. This function
should
       *  be called periodically if using this functionality.
       * arguments:
80     *  pChannel - pointer to channel to check.
       * returns:
       *  void
       */
      TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwpltmr_checkMultiTimer(TMWCHNL *pChannel);

    #ifdef __cplusplus
    }
    #endif

90  #endif /* TMWPLTMR_DEFINED */
```

5 MDNP Build Request Message

'mdnpbrm.h' contains the definition of the routines used to issue requests from the DNP3 Master SCL.

```
/*****************************************************************************/
/* Triangle MicroWorks, Inc.                          Copyright (c) 1997-2022 */
/*****************************************************************************/
/*                                                                           */
/* This file is the property of:                                             */
/*                                                                           */
/*                        Triangle MicroWorks, Inc.                          */
/*                       Raleigh, North Carolina USA                         */
/*                        www.TriangleMicroWorks.com                         */
/*                           (919) 870-6615                                  */
/*                                                                           */
/* This Source Code and the associated Documentation contain proprietary     */
/* information of Triangle MicroWorks, Inc. and may not be copied or         */
/* distributed in any form without the written permission of Triangle        */
/* MicroWorks, Inc.  Copies of the source code may be made only for backup   */
/* purposes.                                                                 */
/*                                                                           */
/* Your License agreement may limit the installation of this source code to  */
/* specific products.  Before installing this source code on a new           */
/* application, check your license agreement to ensure it allows use on the  */
/* product in question.  Contact Triangle MicroWorks for information about   */
/* extending the number of products that may use this source code library or */
/* obtaining the newest revision.                                            */
/*                                                                           */
/*****************************************************************************/

/* file: mdnpbrm.h
 * description: Define methods used to create and send DNP3 requests.
 *
 * The routines in this file are used to generate and send DNP3 requests. In
 * general the user's code will call one of these for each request they want
 * to send to a remote device. The following general notes apply to one or
 * more of the routines below.
 *
 * Each of the request functions defined below returns a pointer to a transmit
 * data structure. In general the contents of this structure should not be
 * used, but the pointer itself should be used to reference this request in
 * the future. Specifically this request pointer can be used to cancel the
 * request via dnpchnl_cancelFragment or in the request's user callback to
 * indentify the specific request that the callback pertains to.
 *
 * The first argument to most of the request functions is a request descriptor
 * of type MDNPBRM_REQ_DESC. This descriptor holds generic information
 * pertinent to the request. This structure should be initialized by a call
 * to mdnpbrm_initReqDesc and then modified as required for the current
 * request before being passed to the request function.
 *
 * Several of the functions accept a pointer to a user supplied transmit data
 * structure, of type DNPCHNL_TX_DATA, called pUserTxData. In most cases this
 * parameter will be set to TMWDEFS_NULL. If pUserTxData is NULL the function
 * will allocate a local transmit data structure, fill it in, send it, and
 * return a pointer to the newly allocated structure. If pUserTxData is not
 * NULL this routine will simply add the appropriate object header and data
 * to the user specified transmit data structure and return without sending
 * it. This allows the user to build a request with multiple objects. In
 * general pUserTxData should be allocated by a call to dnpchnl_newTxData
 * and initialized by calling mdnpbrm_buildRequestHeader before being passed
 * to a request function multiple times. After that dnpchnl_sendFragment should
 * be called to send it.
 * DNPCHNL_TX_DATA *pUserTxData  = (DNPCHNL_TX_DATA*)dnpchnl_newTxData(pChannel,
pSession, size, pSession->destAddress);
 * if(pUserTxData  != NULL)
 * {
 *   mdnpbrm_buildRequestHeader(pUserTxData, DNPDEFS_FC_ACTIVATE_CONFIG);
 *   mdnpbrm_activateConfig(&request, pUserTxData, 0, pFileName1, fileName1Length,
70);
 *   mdnpbrm_activateConfig(&request, pUserTxData, 0, pFileName1, fileName2Length,
70);
 *   mdnpbrm_activateConfig(&request, pUserTxData, 0, pString, stringLength, 110);
 *   if(!dnpchnl_sendFragment((TMWSESN_TX_DATA*)pUserTxData))
 *      /. failed ./
```

```
 *      dnpchnl_freeTxData((TMWSESN_TX_DATA*)pUserTxData);
 * }
 */

#ifndef MDNPBRM_DEFINED
#define MDNPBRM_DEFINED

#include "tmwscl/utils/tmwdefs.h"
#include "tmwscl/dnp/mdnpsesn.h"
#include "tmwscl/dnp/dnpchnl.h"

#define MDNPBRM_AUTO_OPERATE_PRIORITY  254
#define MDNPBRM_AUTHENTICATE_PRIORITY  251
#define MDNPBRM_DISABLE_UNSOL_PRIORITY 250
#define MDNPBRM_CLEAR_RESTART_PRIORITY 249
#define MDNPBRM_TIMESYNC_PRIORITY      248
#define MDNPBRM_DATASET_XCHNG_PRIORITY 246
#define MDNPBRM_INTEGRITY_PRIORITY     244
#define MDNPBRM_ENABLE_UNSOL_PRIORITY  242


/* Allows adding more DOHs to an assign class request without adding another COH
 * No COH is added at the beginning of a request to remove from class 0,
 * but for more clarity an equivalent define is being created.
 */
#define MDNPBRM_ADD_NO_NEW_COH TMWDEFS_CLASS_MASK_NOTCLASS0

/* Structure to hold parameters common to several requests. This structure
 * should be initialized by a call to mdnpbrm_initReqDesc. Then the user
 * should overwrite the desired fields before calling the desired mdnpbrm
 * request function.
 */
typedef struct MDNPBrmReqDescStruct{
  /* Define where this request should be sent. Only one of pChannel or
   * pSession should be set. The other should be TMWDEFS_NULL. If pChannel
   * is not TMWDEFS_NULL this request will be broadcast to all sessions
   * on that channel. If pSession is not TMWDEFS_NULL this request will
   * be sent to the specified session. Most requests will set pSession.
   */
  TMWCHNL *pChannel;
  TMWSESN *pSession;

  /* If this is a broadcast request specify the destination address to
   * use. This should be one of the following:
   *     DNPDEFS_BROADCAST_ADDR_NOCON,
   *     DNPDEFS_BROADCAST_ADDR_CON
   *     DNPDEFS_BROADCAST_ADDR_ORIG
   */
  TMWTYPES_USHORT broadcastAddr;

  /* Request description, note that a copy is NOT made of this request so
   * it should be stored in a buffer that remains valid for the life of
   * the request
   */
  TMWTYPES_CHAR *pMsgDescription;

  /* Priority for this request. Priorities above 200 are normally reserved for
   * internal SCL use. An exception to this is when SELECT/OPERATE is performed
   * using separate BRM commands and automatic event processing is enabled for
   * the master session. See Note about this in mdnpbrm_binaryCommand
   * mdnpbrm_patternMask and mdnpbrm_analogCommand below.
   */
  TMWTYPES_UCHAR priority;

  /* Absolute response timeout, maximum amount of time, in milliseconds,
   * that the SCL will wait for a response to this request before terminating
   * the request. This period starts as soon as the request is submitted.
   */
  TMWTYPES_MILLISECONDS responseTimeout;

  /* User callback to be called when the request, or one step in a multi step
   * request, is complete. Most of the requests generated below consist of a
   * single request/response. In this case the user callback will be called
   * with status DNPCHNL_RESP_STATUS_SUCCESS or one of the failure status
   * codes when the response is received or the request times out. Some of
   * the requests require multiple request/response cycles. In this case the
   * user callback will be called for each response with a status of
   * DNPCHNL_RESP_STATUS_INTERMEDIATE.
   */
  void *pUserCallbackParam;
  DNPCHNL_CALLBACK_FUNC pUserCallback;
```

```
150
    #if MDNPDATA_SUPPORT_OBJ120
      /* Secure authentication */
      /* Send objectGroup 120 variation 3 and variation 9 agressive mode objects in
       * request
       */
      TMWTYPES_BOOL  authAggressiveMode;

      /* User number if aggressive mode is selected or if this request is challenged */
      TMWTYPES_USHORT authUserNumber;
160 #endif

    } MDNPBRM_REQ_DESC;

    /* Define 'automatic' processing to be done by the SCL for the various
     * 'command' requests. If the corresponding bit is set in an 'autoMode'
     * parameter to one of the control routines then the specified operation
     * will be performed as required. Specifically, if MDNPBRM_AUTO_MODE_OPERATE
     * is set than an operate request will be performed when a successful
     * select completes. If MDNPBRM_AUTO_MODE_FEEDBACK is set an operate
170  * feedback poll will be performed when an operate completes.
     */
    typedef TMWTYPES_UCHAR MDNPBRM_AUTO_MODE;
    #define MDNPBRM_AUTO_MODE_NONE      0x00
    #define MDNPBRM_AUTO_MODE_OPERATE   0x01
    #define MDNPBRM_AUTO_MODE_FEEDBACK  0x02

    /* Define structure to hold information for a single Control Relay
     * Output Block. An array of these structures will be passed to
     * mdnpbrm_binaryCommand to generate a single request with multiple
180  * CROB objects in it.
     */
    typedef struct MDNPBrmCROBInfo {
      TMWTYPES_USHORT pointNumber;

      /*     * Control Relay Output Block.
       *  control - control code
       *    low 4 bits is Op Type
       *      DNPDEFS_CROB_CTRL_NUL, DNPDEFS_CROB_CTRL_PULSE_ON,
DNPDEFS_CROB_CTRL_PULSE_OFF,
       *      DNPDEFS_CROB_CTRL_LATCH_ON, or DNPDEFS_CROB_CTRL_LATCH_OFF
190    *    Then the Queue Field which is obsolete
       *    Then the Clear Field DNPDEFS_CROB_CTRL_CLEAR
       *    Then the two bit Trip-Close field  0, or DNPDEFS_CROB_CTRL_PAIRED_CLOSE or
DNPDEFS_CROB_CTRL_PAIRED_TRIP
       *  count - number of times outstation shall execute the operation
       */
      DNPDEFS_CROB_CTRL control;
      TMWTYPES_ULONG onTime;
      TMWTYPES_ULONG offTime;

    /* MDNPDATA_SUPPORT_OBJ12_COUNT must be set to TMWDEFS_TRUE to support the count
200  * field
     */
    #if MDNPDATA_SUPPORT_OBJ12_COUNT
       /*  count - number of times outstation shall execute the operation */
      TMWTYPES_UCHAR count;
    #endif

    } MDNPBRM_CROB_INFO;

    /* Define structure to hold information for a single Analog
210  * Output Block. An array of these structures will be passed to
     * mdnpbrm_analogCommand to generate a single request with multiple
     * analog output objects in it.
     */
    typedef struct MDNPBrmAnalogInfo {
      TMWTYPES_USHORT pointNumber;
      TMWTYPES_ANALOG_VALUE value;
    } MDNPBRM_ANALOG_INFO;

    /* Define structure to hold information for a single indexed absolute time
220  * and long interval object. An array of these structures will be passed to
     * mdnpbrm_writeIndexedTime to generate a single request with multiple
     * indexed time objects in it.
     */
    typedef struct MDNPBrmIndexedTimeInfo {
      TMWTYPES_USHORT pointNumber;
      TMWDTIME indexedTime; /* UTC Time */
      TMWTYPES_ULONG intervalCount;
      TMWTYPES_BYTE intervalUnits;
```

```
} MDNPBRM_INDEXEDTIME_INFO;

/* Define structure to hold information for a single string
 * Output Block. An array of these structures will be passed to
 * mdnpbrm_writeStrings to generate a single request with multiple
 * string objects in it.
 */
typedef struct MDNPBrmStringInfo {
  TMWTYPES_USHORT pointNumber;
  TMWTYPES_CHAR  value[DNPDEFS_MAX_STRING_LENGTH];
} MDNPBRM_STRING_INFO;

/* Define structure to hold information for a single extended string
 * Output Block. An array of these structures will be passed to
 * mdnpbrm_writeExtStrings to generate a single request with multiple
 * extended string objects in it.
 */
typedef struct MDNPBrmExtStringInfo {
  TMWTYPES_USHORT pointNumber;
  TMWTYPES_CHAR  value[DNPCNFG_MAX_EXT_STRING_LENGTH];
} MDNPBRM_EXT_STRING_INFO;

/* Internal structure used to hold callback information for a select,
 * operate, feedback sequence of requests.
 */
typedef struct {
  TMWSESN *pSession;

  /* original tx data, to be reused for followup requests */
  DNPCHNL_TX_DATA *pTxData;

  TMWTYPES_BOOL operateRequired;
  TMWTYPES_BOOL binFeedbackRequired; /* binary output or counter feedback */
  TMWTYPES_BOOL anlgFeedbackRequired;
  TMWTYPES_MILLISECONDS feedbackDelay;
  TMWTIMER feedbackTimer;

} OperateCallbackData;

/* Data types for exchanging data set prototype and descriptor objects
 * between master and slave (outstation)
 */
typedef enum  {
  MDNPBRM_DATASET_READ_PROTO,
  MDNPBRM_DATASET_DELAY_PROTO,
  MDNPBRM_DATASET_WRITE_PROTO,
  MDNPBRM_DATASET_READ_DESCR,
  MDNPBRM_DATASET_DELAY_DESCR,
  MDNPBRM_DATASET_WRITE_DESCR,
  MDNPBRM_DATASET_OPERATE,
  MDNPBRM_DATASET_COMPLETE
} MDNPBRM_DATASET_OPER;

typedef struct _mdnpDataSetXferContext {
  MDNPBRM_DATASET_OPER  nextOperation;
  DNPCHNL_TX_DATA      *pTxData;
  TMWTYPES_USHORT       numberProtosDefinedOnSlave;
  TMWTYPES_USHORT       numberDescrsDefinedOnSlave;
  TMWTIMER              delayTimer;
} MDNPBRM_DATSET_XFER_CONTEXT;

#ifdef __cplusplus
extern "C" {
#endif

  /* function: mdnpbrm_initBroadcastDesc
   * purpose: Initialize a request descriptor for a broadcast
   *  request.
   * arguments:
   *  pReqDesc - request descriptor to initialize
   *  pChannel - channel to broadcast this request on
   *  destAddr - destination address to use in request.
   *    Must be one of:
   *     DNPDEFS_BROADCAST_ADDR_NOCON,
   *     DNPDEFS_BROADCAST_ADDR_CON
   *     DNPDEFS_BROADCAST_ADDR_ORIG
   * returns:
   *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
   */
  TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpbrm_initBroadcastDesc(
    MDNPBRM_REQ_DESC *pReqDesc,
```

```
    TMWCHNL *pChannel,
    TMWTYPES_USHORT destAddr);

  /* function: mdnpbrm_initReqDesc
   * purpose: Initialize a request descriptor
   * arguments:
   *  pReqDesc - request descriptor to initialize
   *  pSession - session this request will be sent to
   * returns:
   *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
   */
  TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpbrm_initReqDesc(
    MDNPBRM_REQ_DESC *pReqDesc,
    TMWSESN *pSession);

  /* function: mdnpbrm_buildRequestHeader
   * purpose: Initialize request application header
   * arguments:
   *  pTxData - tranmsit data structure to initialize
   *  funcCode - function code for request
   * returns:
   *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
   */
  TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpbrm_buildRequestHeader(
    DNPCHNL_TX_DATA *pTxData,
    TMWTYPES_UCHAR funcCode);

  /* function: mdnpbrm_addObjectHeader
   * purpose: Add an object header to the specified message
   * arguments:
   *  pTxData - transmit data structure to add object header to
   *  group - object group
   *  variation - object variation
   *  qualifier - address qualifier
   *  start - start address if qualifier is 8 or 16 bit start/stop
   *  stopOrQty - stop address if qualifier is 8 or 16 bit start/stop
   *   or quantity if qualifier is 8 or 16 bit limited or indexed.
   * returns:
   *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
   */
  TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpbrm_addObjectHeader(
    DNPCHNL_TX_DATA *pTxData,
    TMWTYPES_UCHAR group,
    TMWTYPES_UCHAR variation,
    TMWTYPES_UCHAR qualifier,
    TMWTYPES_USHORT start,
    TMWTYPES_USHORT stopOrQty);

  /* function: mdnpbrm_addObjectData
   * purpose: Add an object data to the specified message
   * arguments:
   *  pTxData - transmit data structure to add data to
   *  len - number of bytes to add
   *  pData - pointer to binary data to add to request
   * returns:
   *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
   */
  TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpbrm_addObjectData(
    DNPCHNL_TX_DATA *pTxData,
    TMWTYPES_USHORT len,
    TMWTYPES_UCHAR *pData);

  /* function: mdnpbrm_cancelSelOpDelayTimer
   * purpose:
   *  If the application is going to be delayed sending the OPERATE command (for
   *  example waiting for human input) you can configure a nonzero mdnp session
   *  selOpDelayTime to delay the sending of other commands until the OPERATE is
sent
   *  or the timeout is reached. This function will cancel that timer if it is
running.
   * arguments:
   *  pSession - pointer to a master DNP session
   * returns:
   *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
   */
  TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpbrm_cancelSelOpDelayTimer(
    TMWSESN *pSession);

  /* function: mdnpbrm_readClass
   * purpose: Issue an read class request to the specified session
   * arguments:
```

```
 *  pReqDesc - request descriptor containing generic parameters for
 *    this request
 *  pUserTxData - pointer to user transmit data structure see description
 *    above
 *  qualifier - address qualifier to use, must be all or 8/16 bit
 *    limited quantity.
 *  maxQuantity - quantity to request if qualifier is 8/16 bit limited
 *    quantity.
 *  class0 - TMWDEFS_TRUE to request class 0 (static data)
 *  class1 - TMWDEFS_TRUE to request class 1 event data
 *  class2 - TMWDEFS_TRUE to request class 2 event data
 *  class3 - TMWDEFS_TRUE to request class 3 event data
 * returns:
 *  Pointer to request transmit data structure or TMWDEFS_NULL
 */
TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_readClass(
  MDNPBRM_REQ_DESC *pReqDesc,
  DNPCHNL_TX_DATA *pUserTxData,
  TMWTYPES_UCHAR qualifier,
  TMWTYPES_USHORT maxQuantity,
  TMWTYPES_BOOL class0,
  TMWTYPES_BOOL class1,
  TMWTYPES_BOOL class2,
  TMWTYPES_BOOL class3);

/* function: mdnpbrm_integrityPoll
 * purpose: Issue an integrity data poll to the specified session
 * arguments:
 *  pReqDesc - request descriptor containing generic parameters for
 *    this request
 * returns:
 *  Pointer to request transmit data structure or TMWDEFS_NULL
 */
TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_integrityPoll(
  MDNPBRM_REQ_DESC *pReqDesc);

/* function: mdnpbrm_eventPoll
 * purpose: Issue an event data poll to the specified session
 * arguments:
 *  pReqDesc - request descriptor containing generic parameters for
 *    this request
 * returns:
 *  Pointer to request transmit data structure or TMWDEFS_NULL
 */
TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_eventPoll(
  MDNPBRM_REQ_DESC *pReqDesc);

/* function: mdnpbrm_binOutFeedbackPoll
 * purpose: Issue an operate feedback request to the specified session.
 *  This will send a read for class 1 2 and 3 events and
 *   binary input status values.
 * arguments:
 *  pReqDesc - request descriptor containing generic parameters for
 *    this request
 * returns:
 *  Pointer to request transmit data structure or TMWDEFS_NULL
 */
TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_binOutFeedbackPoll(
  MDNPBRM_REQ_DESC *pReqDesc);

/* function: mdnpbrm_anlgOutFeedbackPoll
 * purpose: Issue an operate feedback request to the specified session.
 *  This will send a read for class 1 2 and 3 events and
 *  analog input status values.
 * arguments:
 *  pReqDesc - request descriptor containing generic parameters for
 *    this request
 * returns:
 *  Pointer to request transmit data structure or TMWDEFS_NULL
 */
TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_anlgOutFeedbackPoll(
  MDNPBRM_REQ_DESC *pReqDesc);

/* function: mdnpbrm_frznCntrFeedbackPoll
 * purpose: Issue an operate feedback request to the specified session.
 *  This will send a read for class 1 2 and 3 events,
 *  and frozen counter values
 * arguments:
 *  pReqDesc - request descriptor containing generic parameters for
 *    this request
 * returns:
```

```
470     *  Pointer to request transmit data structure or TMWDEFS_NULL
        */
       TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_frznCntrFeedbackPoll(
         MDNPBRM_REQ_DESC *pReqDesc);

       /* function: mdnpbrm_frznAnlgInFeedbackPoll
        * purpose: Issue an operate feedback request to the specified session.
        *  This will send a read for class 1 2 and 3 events,
        *  and frozen analog input values
        * arguments:
480     *  pReqDesc - request descriptor containing generic parameters for
        *    this request
        * returns:
        *  Pointer to request transmit data structure or TMWDEFS_NULL
        */
       TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_frznAnlgInFeedbackPoll(
         MDNPBRM_REQ_DESC *pReqDesc);

       /* function: mdnpbrm_clearRestart
        * purpose: Issue a clear restart request to the specified session
490     * arguments:
        *  pReqDesc - request descriptor containing generic parameters for
        *    this request
        * returns:
        *  Pointer to request transmit data structure or TMWDEFS_NULL
        */
       TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_clearRestart(
         MDNPBRM_REQ_DESC *pReqDesc);

       /* function: mdnpbrm_clearNeedTime
500     * purpose: Issue a clear Need Time IIN bit request to the specified session
        * arguments:
        *  pReqDesc - request descriptor containing generic parameters for
        *    this request
        * returns:
        *  Pointer to request transmit data structure or TMWDEFS_NULL
        */
       TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_clearNeedTime(
         MDNPBRM_REQ_DESC *pReqDesc);

510    /* function: mdnpbrm_clearIINBit
        * purpose: Issue a clear specified IIN bit write request
        * arguments:
        *  pReqDesc - request descriptor containing generic parameters for
        *    this request
        *  index - index of the IIN bit to clear.
        *    DNPDEFS_IIN_NEEDTIME_INDEX  4
        *    DNPDEFS_IIN_RESTART_INDEX   7
        * returns:
        *  Pointer to request transmit data structure or TMWDEFS_NULL
520     */
       TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_clearIINBit(
         MDNPBRM_REQ_DESC *pReqDesc,
         TMWTYPES_UCHAR index);

       /* function: mdnpbrm_coldRestart
        * purpose: Issue a cold restart request to the specified session
        * arguments:
        *  pReqDesc - request descriptor containing generic parameters for
        *    this request
530     * returns:
        *  Pointer to request transmit data structure or TMWDEFS_NULL
        */
       TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_coldRestart(
         MDNPBRM_REQ_DESC *pReqDesc);

       /* function: mdnpbrm_warmRestart
        * purpose: Issue a warm restart request to the specified session
        * arguments:
        *  pReqDesc - request descriptor containing generic parameters for
540     *    this request
        * returns:
        *  Pointer to request transmit data structure or TMWDEFS_NULL
        */
       TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_warmRestart(
         MDNPBRM_REQ_DESC *pReqDesc);

   #if MDNPDATA_SUPPORT_OBJ50_V1
       /* function: mdnpbrm_delayMeasurement
        * purpose: Issue an delay measurement request to the specified session.
550     *  This is an optional step in a time synchronization, typically only
```

```
     *  used for serial channels.
     * arguments:
     *  pReqDesc - request descriptor containing generic parameters for
     *   this request
     * returns:
     *  Pointer to request transmit data structure or TMWDEFS_NULL
     */
    TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_delayMeasurement(
      MDNPBRM_REQ_DESC *pReqDesc);

    /* function: mdnpbrm_readTime
     * purpose: Issue an read time request to the specified session
     * arguments:
     *  pReqDesc - request descriptor containing generic parameters for
     *   this request
     * returns:
     *  Pointer to request transmit data structure or TMWDEFS_NULL
     */
    TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_readTime(
      MDNPBRM_REQ_DESC *pReqDesc);

    /* function: mdnpbrm_writeTime
     * purpose: Issue an write time request to the specified session
     *  This sets the time on the remote device, this request is typically
     *  only used for serial channels.
     * arguments:
     *  pReqDesc - request descriptor containing generic parameters for
     *   this request
     * returns:
     *  Pointer to request transmit data structure or TMWDEFS_NULL
     */
    TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_writeTime(
      MDNPBRM_REQ_DESC *pReqDesc);
  #endif

  #if MDNPDATA_SUPPORT_OBJ50_V3
    /* function: mdnpbrm_recordCurrentTime
     * purpose: Issue an record current time request to the specified session.
     *  This is generally the first step in a time synchronization on a LAN
     *  channel.
     * arguments:
     *  pReqDesc - request descriptor containing generic parameters for
     *   this request
     * returns:
     *  Pointer to request transmit data structure or TMWDEFS_NULL
     */
    TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_recordCurrentTime(
      MDNPBRM_REQ_DESC *pReqDesc);

    /* function: mdnpbrm_writeRecordedTime
     * purpose: Issue an write time request to the specified session
     *  This is generally the second step in a time synchronization on a LAN
     *  channel.
     * arguments:
     *  pReqDesc - request descriptor containing generic parameters for
     *   this request
     * returns:
     *  Pointer to request transmit data structure or TMWDEFS_NULL
     */
    TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_writeRecordedTime(
      MDNPBRM_REQ_DESC *pReqDesc);
  #endif

  #if MDNPDATA_SUPPORT_OBJ50_V1 || MDNPDATA_SUPPORT_OBJ50_V3
    /* Define enum which specifies the type of time sync required */
    typedef enum {
      MDNPBRM_SYNC_TYPE_SERIAL,
      MDNPBRM_SYNC_TYPE_LAN
    } MDNPBRM_SYNC_TYPE;

    /* function: mdnpbrm_timeSync
     * purpose: Synchronize time on a remote DNP3 device. DNP3 supports two
     *  mechanisms for time synchronization, one for standard serial channels
     *  and another for LAN applications. Each of these mechanisms may require
     *  multiple request/response operations. This routine will perform all
     *  the operations required to perform a time synchronization as specified.
     * arguments:
     *  pReqDesc - request descriptor containing generic parameters for
     *   this request.
     *  type - type of time synchronization to perform, serial or lan.
     *  measureDelay - TMWDEFS_TRUE to perform a delay measurement before
```

```
     *   the write time request.
     * returns:
     *  Pointer to request transmit data structure or TMWDEFS_NULL
     */
    TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_timeSync(
      MDNPBRM_REQ_DESC *pReqDesc,
      MDNPBRM_SYNC_TYPE type,
      TMWTYPES_BOOL measureDelay);
#endif

#if MDNPDATA_SUPPORT_OBJ50_V4
  /* function: mdnpbrm_writeIndexedTime
   * purpose: Issue an write indexed absoluted time request to the specified session
   *  It is used to specify the initial time of an action and the time between
   *  subsequent, regular repetitions of the actions or activities.
   * arguments:
   *  pReqDesc - request descriptor containing generic parameters for
   *   this request
   * returns:
   *  Pointer to request transmit data structure or TMWDEFS_NULL
   */
  TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_writeIndexedTime(
    MDNPBRM_REQ_DESC *pReqDesc,
    DNPCHNL_TX_DATA *pUserTxData,
    TMWTYPES_UCHAR qualifier,
    TMWTYPES_UCHAR numIndexedTime,
    MDNPBRM_INDEXEDTIME_INFO *pINDEXEDTIMEInfo);
#endif

  /* function: mdnpbrm_unsolEnable
   * purpose: Issue an unsolicited enable request to the specified
   *  session
   * arguments:
   *  pReqDesc - request descriptor containing generic parameters for
   *   this request
   *  class1 - TMWDEFS_TRUE to enable unsolicited responses for class 1 events
   *  class2 - TMWDEFS_TRUE to enable unsolicited responses for class 2 events
   *  class3 - TMWDEFS_TRUE to enable unsolicited responses for class 3 events
   * returns:
   *  Pointer to request transmit data structure or TMWDEFS_NULL
   */
  TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_unsolEnable(
    MDNPBRM_REQ_DESC *pReqDesc,
    TMWTYPES_BOOL class1,
    TMWTYPES_BOOL class2,
    TMWTYPES_BOOL class3);

  /* function: mdnpbrm_unsolDisable
   * purpose: Issue an unsolicited disable request to the specified
   *  session
   * arguments:
   *  pReqDesc - request descriptor containing generic parameters for
   *   this request
   *  class1 - TMWDEFS_TRUE to disable unsolicited responses for class 1 events
   *  class2 - TMWDEFS_TRUE to disable unsolicited responses for class 2 events
   *  class3 - TMWDEFS_TRUE to disable unsolicited responses for class 3 events
   * returns:
   *  Pointer to request transmit data structure or TMWDEFS_NULL
   */
  TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_unsolDisable(
    MDNPBRM_REQ_DESC *pReqDesc,
    TMWTYPES_BOOL class1,
    TMWTYPES_BOOL class2,
    TMWTYPES_BOOL class3);

  /* function: mdnpbrm_assignClass
   * purpose: Issue an assign class request to the specified session
   * arguments:
   *  pReqDesc - request descriptor containing generic parameters for
   *   this request
   *  pUserTxData - pointer to user transmit data structure see description
   *   above
   *  group - object group to assign class
   *  classMask - class to assign group to. This puts a Class Object Header (COH)
   *   in a request and all Data Object Headers (DOH) immediately following that
   *   are assigned to that class.
   *  TMWDEFS_CLASS_MASK_ONE   - assign to event class 1
   *  TMWDEFS_CLASS_MASK_TWO   - assign to event class 2
   *  TMWDEFS_CLASS_MASK_THREE - assign to event class 3
   *  TMWDEFS_CLASS_MASK_NONE  - assign to no event class
   *  TMWDEFS_CLASS_MASK_NOTCLASS0 - remove from static class 0. This adds a "NULL"
```

```
         *   COH which indicates NOTCLASS0 and if used must be the first in the request.
         *  MDNPBRM_ADD_NO_NEW_COH   - To add more DOHs/groups to this request using the
existing
         *   last COH to specify the class for this group.
         *  qualifier - address qualfier, must be all or 8/16 bit start/stop
         *   or 8/16 bit limited quantity or 8/16 bit indexed
         *  start - start if qualifier is 8/16 bit start/stop
         *  stopOrQty - stop if qualifier is 8/16 bit start/stop or quantity
720      *   if qualifier is 8/16 bit limited quantity
         *  pPoints - pointer to array of point numbers to read if 8/16 bit indexed
         * returns:
         *  Pointer to request transmit data structure or TMWDEFS_NULL
         */
        TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_assignClass(
          MDNPBRM_REQ_DESC *pReqDesc,
          DNPCHNL_TX_DATA *pUserTxData,
          TMWDEFS_CLASS_MASK classMask,
          TMWTYPES_UCHAR group,
730       TMWTYPES_UCHAR qualifier,
          TMWTYPES_USHORT start,
          TMWTYPES_USHORT stopOrQty,
          TMWTYPES_USHORT *pPoints);

        /* function: mdnpbrm_readGroup
         * purpose: Issue a read group request to the specified session
         * arguments:
         *  pReqDesc - request descriptor containing generic parameters for
         *   this request
740      *  pUserTxData - pointer to user transmit data structure see description
         *   above
         *  group - object group for read request
         *  variation - object variation to read. The type specified by this is
         *   specific to the object group being read.
         *  qualifier - address qualifier, must be all, 8/16 bit limited quantity,
         *   or 8/16 bit start/stop
         *  start - start point if qualifier is start/stop
         *  stopOrQty - stop point if qualifier is start/stop or quantity if
         *   qualifier is limited points.
750      * returns:
         *  Pointer to request transmit data structure or TMWDEFS_NULL
         */
        TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_readGroup(
          MDNPBRM_REQ_DESC *pReqDesc,
          DNPCHNL_TX_DATA *pUserTxData,
          TMWTYPES_UCHAR group,
          TMWTYPES_UCHAR variation,
          TMWTYPES_UCHAR qualifier,
          TMWTYPES_USHORT start,
760       TMWTYPES_USHORT stopOrQty);

        /* function: mdnpbrm_readPoints
         * purpose: Issue an read one or more specific data points request to the
         *  specified session. Note that this request uses indexed addressing which
         *  is not required by any of the DNP3 subset level definitions and is not
         *  supported by a significant percentage of DNP3 slave devices.
         * arguments:
         *  pReqDesc - request descriptor containing generic parameters for
         *   this request
770      *  pUserTxData - pointer to user transmit data structure see description
         *   above
         *  group - object group for read request
         *  variation - object variation to read
         *  qualifier - address qualifier, must be 8/16 bit indexed
         *  DNPDEFS_QUAL_8BIT_INDEX or DNPDEFS_QUAL_16BIT_INDEX
         *  pPoints - pointer to array of point numbers to read
         *  numPoints - number of points to read
         * returns:
         *  Pointer to request transmit data structure or TMWDEFS_NULL
780      */
        TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_readPoints(
          MDNPBRM_REQ_DESC *pReqDesc,
          DNPCHNL_TX_DATA *pUserTxData,
          TMWTYPES_UCHAR group,
          TMWTYPES_UCHAR variation,
          TMWTYPES_UCHAR qualifier,
          TMWTYPES_USHORT *pPoints,
          TMWTYPES_USHORT numPoints);

790     /* function: mdnpbrm_binaryOutWrite
         * purpose: Issue a binary output write command to the specified session
         * arguments:
```

```
     *  pReqDesc - request descriptor containing generic parameters for
     *   this request
     *  pUserTxData - pointer to user transmit data structure see description
     *   above
     *  qualifier - address qualifier, must be 8 or 16 bit startstop
     *    DNPDEFS_QUAL_8BIT_START_STOP or DNPDEFS_QUAL_16BIT_START_STOP
     *  start - starting point index
800  *  stop - ending point index
     *  pValues - pointer to array of values, 0 for off, 1 for on
     *   Status bits are not sent in a write request.
     * returns:
     *  Pointer to request transmit data structure or TMWDEFS_NULL
     */
    TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_binaryOutWrite(
      MDNPBRM_REQ_DESC *pReqDesc,
      DNPCHNL_TX_DATA *pUserTxData,
      TMWTYPES_UCHAR qualifier,
810   TMWTYPES_USHORT start,
      TMWTYPES_USHORT stop,
      TMWTYPES_UCHAR *pValues);

    /* function: mdnpbrm_binaryCommand
     * purpose: Issue a binary command to the specified session
     * arguments:
     *  pReqDesc - request descriptor containing generic parameters for
     *   this request
     *  pUserTxData - pointer to user transmit data structure see description
820  *   above
     *  funcCode - function code, must be DNPDEFS_FC_SELECT, DNPDEFS_FC_OPERATE,
     *   DNPDEFS_FC_DIRECT_OP, or DNPDEFS_FC_DIRECT_OP_NOACK
     *  autoMode - bit mask to specify which actions to perform internally within
     *   the SCL. See definition of MDNPBRM_AUTO_MODE above.
     *  opFeedbackDelay - specify a delay, in milliseconds, after receiving
     *   the response to an operate or direct operate request before issueing
     *   an operate feedback poll using the autoMode parameter.
     *  qualifier - address qualifier, must be 8 or 16 bit indexed
     *   DNPDEFS_QUAL_8BIT_INDEX or DNPDEFS_QUAL_16BIT_INDEX
830  *  numCROBs - number of Control Relay Output Blocks in this request
     *  pCROBInfo - data for each Control Relay Output Block
     * returns:
     *  Pointer to request transmit data structure or TMWDEFS_NULL
     *
     * NOTE: If you are doing a SELECT/OPERATE but choose not to use the autoMode
     *  mask to do this, but instead are going to call this brm function to send
     *  the SELECT and then call this function again to send the OPERATE, you will
     *  need to increase the pReqDesc->priority of the OPERATE to 255 to
     *  guarantee it gets sent before any automatic processing of events is performed.
840  *  (Automatic processing of events is enabled by default when the mdnp session
     *  was opened). For example, if in the response to the SELECT the restart IIN bit
     *  is set and the session was set up to automatically send a clear restart
     *  command, you will want to make sure the OPERATE gets sent before the clear
     *  restart, otherwise the operate will fail.
     */
    TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_binaryCommand(
      MDNPBRM_REQ_DESC *pReqDesc,
      DNPCHNL_TX_DATA *pUserTxData,
      TMWTYPES_UCHAR funcCode,
850   MDNPBRM_AUTO_MODE autoMode,
      TMWTYPES_MILLISECONDS opFeedbackDelay,
      TMWTYPES_UCHAR qualifier,
      TMWTYPES_UCHAR numCROBs,
      MDNPBRM_CROB_INFO *pCROBInfo);

    /* function: mdnpbrm_patternMask
     * purpose: Send a pattern mask to remote device
     * arguments:
     *  pReqDesc - request descriptor containing generic parameters for
860  *   this request
     *  funcCode - function code, DNPDEFS_FC_SELECT, DNPDEFS_FC_OPERATE,
     *   DNPDEFS_FC_DIRECT_OP, or DNPDEFS_FC_DIRECT_OP_NOACK
     *  autoMode - bit mask to specify which actions to perform internally within
     *   the SCL. See definition of MDNPBRM_AUTO_MODE above.
     *  opFeedbackDelay - specify a delay, in milliseconds, after receiving
     *   the response to an operate or direct operate request before issueing
     *   an operate feedback poll using the autoMode parameter.
     *  qualifier - address qualifier, must be 8 or 16 bit start/stop
     *    DNPDEFS_QUAL_8BIT_START_STOP or DNPDEFS_QUAL_16BIT_START_STOP
870  *  start - first point number for mask
     *  stop - last point number for mask
     *  control - control code
     *    low 4 bits is Op Type
```

```
         *      DNPDEFS_CROB_CTRL_NUL, DNPDEFS_CROB_CTRL_PULSE_ON,
DNPDEFS_CROB_CTRL_PULSE_OFF,
         *      DNPDEFS_CROB_CTRL_LATCH_ON, or DNPDEFS_CROB_CTRL_LATCH_OFF
         *    Then the Queue Field which is obsolete
         *    Then the Clear Field DNPDEFS_CROB_CTRL_CLEAR
         *    Then the two bit Trip-Close field  0, or DNPDEFS_CROB_CTRL_PAIRED_CLOSE or
DNPDEFS_CROB_CTRL_PAIRED_TRIP
         *  count - number of times outstation shall execute the operation
880      *  activationPeriod - pulse activation time
         *  deactivationPeriod - pulse deactivation time
         *  pMask - bitmask indicating which points in the range specified by start->stop
are to be controlled.
         *   bit value of 1 means that point should be controlled.
         *   Example: funcCode=DNPDEFS_FC_DIRECT_OP, start==2, stop==17,
         *              control=DNPDEFS_CROB_CTRL_LATCH_ON, *pMask=0x81,0x10;
         *            sends a latch on to points 2, 9 and 14
         * returns:
         *  Pointer to request transmit data structure or TMWDEFS_NULL
         *
890      * NOTE: If you are doing a SELECT/OPERATE but choose not to use the autoMode
         *  mask to do this, but instead are going to call this brm function to send
         *  the SELECT and then call this function again to send the OPERATE, you will
         *  need to increase the pReqDesc->priority of the OPERATE to 255 to
         *  guarantee it gets sent before any automatic processing of events is performed.
         *  (Automatic processing of events is enabled by default when the mdnp session
         *  was opened). For example, if in the response to the SELECT the restart IIN bit
         *  is set and the session was set up to automatically send a clear restart
         *  command, you will want to make sure the OPERATE gets sent before the clear
         *  restart, otherwise the operate will fail.
900      */
       TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_patternMask(
         MDNPBRM_REQ_DESC *pReqDesc,
         TMWTYPES_UCHAR funcCode,
         MDNPBRM_AUTO_MODE autoMode,
         TMWTYPES_MILLISECONDS opFeedbackDelay,
         TMWTYPES_UCHAR qualifier,
         TMWTYPES_USHORT start,
         TMWTYPES_USHORT stop,
         TMWTYPES_UCHAR control,
910      TMWTYPES_UCHAR count,
         TMWTYPES_ULONG activationPeriod,
         TMWTYPES_ULONG deactivationPeriod,
         TMWTYPES_UCHAR *pMask);

       /* function: mdnpbrm_analogCommand
        * purpose: Issue an analog command containing object group 41 to the
        *   specified session
        * arguments:
        *  pReqDesc - request descriptor containing generic parameters for
920     *   this request
        *  pUserTxData - pointer to user transmit data structure see description
        *   above
        *  funcCode - function code, must be DNPDEFS_FC_SELECT, DNPDEFS_FC_OPERATE,
        *   DNPDEFS_FC_DIRECT_OP, or DNPDEFS_FC_DIRECT_OP_NOACK
        *  autoMode - bit mask to specify which actions to perform internally within
        *   the SCL. See definition of MDNPBRM_AUTO_MODE above.
        *  opFeedbackDelay - specify a delay, in milliseconds, after receiving
        *   the response to an operate or direct operate request before issueing
        *   an operate feedback poll using the autoMode parameter.
930     *  qualifier - address qualifier, must be 8 or 16 bit indexed
        *   DNPDEFS_QUAL_8BIT_INDEX or DNPDEFS_QUAL_16BIT_INDEX
        *  variation - object variation to use in request, must be 1, 2, 3 or 4
        *   This will indicate what type value will be sent to the outstation.
        *   1 - 32 bit                          (this was added in subset level 3)
        *   2 - 16 bit                          (this is in subset level 1)
        *   3 - Single Precision Floating Point  (this was added in subset level 4)
        *   4 - Double Precision Floating Point  (this is beyond subset level 4 )
        *  numObjects - number of Analog Output Blocks in this request
        *  pAnlgInfo - data for each Analog Output Block
940     * returns:
        *  Pointer to request transmit data structure or TMWDEFS_NULL
        *
        * NOTE: If you are doing a SELECT/OPERATE but choose not to use the autoMode
        *  mask to do this, but instead are going to call this brm function to send
        *  the SELECT and then call this function again to send the OPERATE, you will
        *  need to increase the pReqDesc->priority of the OPERATE to 255 to
        *  guarantee it gets sent before any automatic processing of events is performed.
        *  (Automatic processing of events is enabled by default when the mdnp session
        *  was opened). For example, if in the response to the SELECT the restart IIN bit
950     *  is set and the session was set up to automatically send a clear restart
        *  command, you will want to make sure the OPERATE gets sent before the clear
```

```
 *  restart, otherwise the operate will fail.
 */
TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_analogCommand(
  MDNPBRM_REQ_DESC *pReqDesc,
  DNPCHNL_TX_DATA *pUserTxData,
  TMWTYPES_UCHAR funcCode,
  MDNPBRM_AUTO_MODE autoMode,
  TMWTYPES_MILLISECONDS opFeedbackDelay,
  TMWTYPES_UCHAR qualifier,
  TMWTYPES_UCHAR variation,
  TMWTYPES_UCHAR numObjects,
  MDNPBRM_ANALOG_INFO *pAnlgInfo);

/* function: mdnpbrm_writeDeadband
 * purpose: Issue a write request to Object 34 (Analog Input Deadbands)
 * arguments:
 *  pReqDesc - request descriptor containing generic parameters for
 *   this request
 *  pUserTxData - pointer to user transmit data structure see description
 *   above
 *  qualifier - address qualifier, must be 8/16 bit indexed or 8/16 bit start/stop
 *  variation - object variation to use in request
 *   1 - 16 bit                          (this was added in subset level 4)
 *   2 - 32 bit                          (this was added in subset level 4)
 *   3 - Single Precision Floating Point  (this was added in subset level 4)
 *  numObjects - number of deadbands in this request
 *  pAnlgInfo - data for each deadband
 * returns:
 *  Pointer to request transmit data structure or TMWDEFS_NULL
 */
TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_writeDeadband(
  MDNPBRM_REQ_DESC *pReqDesc,
  DNPCHNL_TX_DATA *pUserTxData,
  TMWTYPES_UCHAR qualifier,
  TMWTYPES_UCHAR variation,
  TMWTYPES_UCHAR numObjects,
  MDNPBRM_ANALOG_INFO *pAnlgInfo);

/* function: mdnpbrm_freezeCounters
 * purpose: Issue an freeze counters request to the specified session
 * arguments:
 *  pReqDesc - request descriptor containing generic parameters for
 *   this request
 *  pUserTxData - pointer to user transmit data structure see description
 *   above
 *  clear - clear counters after freeze
 *  noAck - don't ask for application layer response
 *  qualifier - address qualifier, must be all,
 *   or 8/16 bit start/stop
 *  start - start point if qualifier is 8/16 bit start/stop
 *  stopOrQty - stop point if qualifier is 8/16 bit start/top or quantity
 *   if qualifier is 8/16 bit limited quantity
 *  feedbackRequested - if true automatically read frozen counters when
 *   freeze command is complete.
 *   NOTE: if device supports frozen counter events this should be set to
 *   false to prevent reading all of the frozen counters in addition to
 *   receiving the frozen counter events.
 * returns:
 *  Pointer to request transmit data structure or TMWDEFS_NULL
 */
TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_freezeCounters(
  MDNPBRM_REQ_DESC *pReqDesc,
  DNPCHNL_TX_DATA *pUserTxData,
  TMWTYPES_BOOL clear,
  TMWTYPES_BOOL noAck,
  TMWTYPES_UCHAR qualifier,
  TMWTYPES_USHORT start,
  TMWTYPES_USHORT stopOrQty,
  TMWTYPES_BOOL feedbackRequested);

 /* function: mdnpbrm_freezeAnalogInputs
 * purpose: Issue an freeze analog inputs request to the specified session
 * arguments:
 *  pReqDesc - request descriptor containing generic parameters for
 *   this request
 *  pUserTxData - pointer to user transmit data structure see description
 *   above
 *  clear - clear analog inputs after freeze
 *  noAck - don't ask for application layer response
 *  qualifier - address qualifier, must be all,
 *   or 8/16 bit start/stop
```

```
         *  start - start point if qualifier is 8/16 bit start/stop
         *  stopOrQty - stop point if qualifier is 8/16 bit start/top or quantity
         *   if qualifier is 8/16 bit limited quantity
         *  feedbackRequested - if true automatically read frozen analog inputs when
         *   freeze command is complete.
         *   NOTE: if device supports frozen analog input events this should be set to
         *   false to prevent reading all of the frozen analog inputs in addition to
1040     *   receiving the frozen analog input events.
         * returns:
         *  Pointer to request transmit data structure or TMWDEFS_NULL
         */
        TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_freezeAnalogInputs(
          MDNPBRM_REQ_DESC *pReqDesc,
          DNPCHNL_TX_DATA *pUserTxData,
          TMWTYPES_BOOL clear,
          TMWTYPES_BOOL noAck,
          TMWTYPES_UCHAR qualifier,
1050      TMWTYPES_USHORT start,
          TMWTYPES_USHORT stopOrQty,
          TMWTYPES_BOOL feedbackRequested);

       /* function: mdnpbrm_freezeAtTime
        * purpose: Issue an freeze at time request for the object group to the specified
session
        * arguments:
        *  pReqDesc - request descriptor containing generic parameters for
        *   this request
        *  pUserTxData - pointer to user transmit data structure see description
1060    *   above
        *  noAck - don't ask for application layer response
        *  objectGroup (20 counters or 30 analog inputs)
        *  qualifier - address qualifier, must be all,
        *   or 8/16 bit start/stop
        *  timeDateEnum - time-date field schedule interpretation
        *  pFreezeTime - time in UTC to perform freeze
        *  freezeInterval - time interval to perfrom periodic freezes (milliseconds)
        *  start - start point if qualifier is 8/16 bit start/stop
        *  stopOrQty - stop point if qualifier is 8/16 bit start/top or quantity
1070    *   if qualifier is 8/16 bit limited quantity
        *  feedbackRequested - if true automatically read the object group when
        *   freeze command is complete.
        *   NOTE: if device supports events this should be set to
        *   false to prevent reading all of the object group's data in addition to
        *   receiving its events.
        * returns:
        *  Pointer to request transmit data structure or TMWDEFS_NULL
        */
        TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_freezeAtTime(
1080      MDNPBRM_REQ_DESC *pReqDesc,
          DNPCHNL_TX_DATA *pUserTxData,
          TMWTYPES_BOOL noAck,
          DNPDEFS_OBJ_GROUP_ID objectGroup,
          TMWTYPES_UCHAR qualifier,
          DNPDATA_FREEZE_TIME_DATE_FIELD timeDateEnum,
          TMWDTIME *pFreezeTime,
          TMWTYPES_ULONG freezeInterval,
          TMWTYPES_USHORT start,
          TMWTYPES_USHORT stopOrQty,
1090      TMWTYPES_BOOL feedbackRequested);

       /* function: mdnpbrm_writeString
        * purpose: Write a string object to the specified point
        * arguments:
        *  pReqDesc - request descriptor containing generic parameters for
        *   this request
        *  pUserTxData - pointer to user transmit data structure see description
        *   above
        *  point - point number of string to write
1100    *  pValue - pointer to string data to write
        *  length - length of string to write
        * returns:
        *  Pointer to request transmit data structure or TMWDEFS_NULL
        */
        TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_writeString(
          MDNPBRM_REQ_DESC *pReqDesc,
          DNPCHNL_TX_DATA *pUserTxData,
          TMWTYPES_USHORT point,
          TMWTYPES_UCHAR *pValue,
1110      TMWTYPES_UCHAR length);

       /* function: mdnpbrm_writeStrings
```

```
 * purpose: Write a string objects
 * arguments:
 *  pReqDesc - request descriptor containing generic parameters for
 *   this request
 *  pUserTxData - pointer to user transmit data structure see description
 *   above
 *  qualifier - address qualifier, must be 8 or 16 bit indexed
 *  numObjects - number of strings in this request
 *  pStringInfo - data for each string
 * returns:
 *  Pointer to request transmit data structure or TMWDEFS_NULL
 */
TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_writeStrings(
  MDNPBRM_REQ_DESC *pReqDesc,
  DNPCHNL_TX_DATA *pUserTxData,
  TMWTYPES_UCHAR qualifier,
  TMWTYPES_UCHAR numObjects,
  MDNPBRM_STRING_INFO *pStringInfo);

/* function: mdnpbrm_activateConfig
 * purpose: Send an Activate Configuration request
 * arguments:
 *  pReqDesc - request descriptor containing generic parameters for
 *   this request
 *  pUserTxData - pointer to user transmit data structure see description
 *   above. If multiple names are being sent using object group 70, they will
 *   be combined using a single object header if the first object is object 70
 *   and all objects added after that if they are object 70, until an object 110
 *   string object is added. After that each object will have a separate object
 *   header. If the outstation does not support multiple file names with 1 object
 *   header, you can set combineActConfigData = TMWDEFS_FALSE.
 *  point - point number of string to write
 *  pValue - pointer to string data to write
 *  length - length of string to write
 *    if objectGroup 110 is specified this has a max value of 255
 *    if objectGroup 70 is specified this max would be limited by
 *      tx fragment size.
 *  objectGroup - 70 to use file object in request
 *                110 to use string object in request
 * returns:
 *  Pointer to request transmit data structure or TMWDEFS_NULL
 */
TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_activateConfig(
  MDNPBRM_REQ_DESC *pReqDesc,
  DNPCHNL_TX_DATA *pUserTxData,
  TMWTYPES_USHORT point,
  TMWTYPES_UCHAR *pValue,
  TMWTYPES_USHORT length,
  DNPDEFS_OBJ_GROUP_ID objectGroup);

/* function: mdnpbrm_writeVirtualTerminal
 * purpose: Write a string object to the specified point
 * arguments:
 *  pReqDesc - request descriptor containing generic parameters for
 *   this request
 *  pUserTxData - pointer to user transmit data structure see description
 *   above
 *  point - point number of virtual terminal to write to
 *  pValue - pointer to string data to write
 *  length - length of string to write
 * returns:
 *  Pointer to request transmit data structure or TMWDEFS_NULL
 */
TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_writeVirtualTerminal(
  MDNPBRM_REQ_DESC *pReqDesc,
  DNPCHNL_TX_DATA *pUserTxData,
  TMWTYPES_USHORT point,
  TMWTYPES_UCHAR *pValue,
  TMWTYPES_UCHAR length);

/* function: mdnpbrm_writeExtString
 * purpose: Write an extended string object to the specified point
 * arguments:
 *  pReqDesc - request descriptor containing generic parameters for
 *   this request
 *  pUserTxData - pointer to user transmit data structure see description
 *   above
 *  point - point number of string to write
 *  pValue - pointer to string data to write
 *  length - length of string to write
 * returns:
```

```
     *  Pointer to request transmit data structure or TMWDEFS_NULL
     */
    TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_writeExtString(
      MDNPBRM_REQ_DESC *pReqDesc,
      DNPCHNL_TX_DATA *pUserTxData,
      TMWTYPES_USHORT point,
      TMWTYPES_UCHAR *pValue,
      TMWTYPES_USHORT length);

     /* function: mdnpbrm_writeExtStrings
     * purpose: Write extended string objects
     * arguments:
     *  pReqDesc - request descriptor containing generic parameters for
     *   this request
     *  pUserTxData - pointer to user transmit data structure see description
     *   above
     *  qualifier - address qualifier, must be 8 or 16 bit indexed
     *  variation - object variation to use in request, must be 1 or 2
     *  numObjects - number of strings in this request
     *  pStringInfo - data for each string
     * returns:
     *  Pointer to request transmit data structure or TMWDEFS_NULL
     */
    TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_writeExtStrings(
      MDNPBRM_REQ_DESC *pReqDesc,
      DNPCHNL_TX_DATA *pUserTxData,
      TMWTYPES_UCHAR qualifier,
      TMWTYPES_UCHAR variation,
      TMWTYPES_UCHAR numObjects,
      MDNPBRM_EXT_STRING_INFO *pStringInfo);

    /* function: mdnpbrm_fileInfo
     * purpose: - Issue a file information request to remote device
     * arguments:
     *  pReqDesc - request descriptor containing generic parameters for
     *   this request
     *  requestId - request id to be sent in request
     *  pFilename - pointer to name of file to retrieve information about.
     * returns:
     *  Pointer to request transmit data structure or TMWDEFS_NULL
     */
    TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_fileInfo(
      MDNPBRM_REQ_DESC *pReqDesc,
      TMWTYPES_USHORT requestId,
      const TMWTYPES_CHAR *pFilename);

    /* function: mdnpbrm_fileAuthentication
     * purpose: Issue a file authentication request to a remote device
     *  asking for an authentication key.
     * arguments:
     *  pReqDesc - request descriptor containing generic parameters for
     *   this request
     *  pUserName - pointer to Null Terminated user name to be sent
     *  pPassword - pointer to Null Terminated password to be sent
     * returns:
     *  Pointer to request transmit data structure or TMWDEFS_NULL
     */
    TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_fileAuthentication(
      MDNPBRM_REQ_DESC *pReqDesc,
      TMWTYPES_CHAR *pUsername,
      TMWTYPES_CHAR *pPassword);

    /* function: mdnpbrm_fileOpen
     * purpose: Send a request to open a file on the remote device. If file
     *  authentication is required the already acquired file authentication
     *  key may be passed to this function.
     * arguments:
     *  pReqDesc - request descriptor containing generic parameters for
     *   this request
     *  requestId - request id to be sent in request
     *  mode - mode for file open indicating open for reading, writing, or appending
     *  permissions - permissions value to send to outstation, only used if mode ==
DNPDEFS_FILE_MODE_WRITE
     *                these bits may be OR'ed together
     *                DNPDEFS_WORLD_EXECUTE_ALLOWED     0x0001
     *                DNPDEFS_WORLD_WRITE_ALLOWED       0x0002
     *                DNPDEFS_WORLD_READ_ALLOWED        0x0004
     *                DNPDEFS_GROUP_EXECUTE_ALLOWED     0x0008
     *                DNPDEFS_GROUP_WRITE_ALLOWED       0x0010
     *                DNPDEFS_GROUP_READ_ALLOWED        0x0020
     *                DNPDEFS_OWNER_EXECUTE_ALLOWED     0x0040
```

```
         *                DNPDEFS_OWNER_WRITE_ALLOWED       0x0080
         *                DNPDEFS_OWNER_READ_ALLOWED        0x0100
         *  fileType  DNPDEFS_FILE_TYPE_DIRECTORY if opening a directory for reading
         *            DNPDEFS_FILE_TYPE_SIMPLE if opening a file for reading or writing
         *  authentication key - key to be sent in open request. Should be 0 or have
         *   been acquired by mdnpbrm_fileAuthentication
1280     *  maxBlockSize - maximum block size to be read or written for this file
         *  pFileName - pointer to file or directory to open
         * returns:
         *  Pointer to request transmit data structure or TMWDEFS_NULL
         */
        TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_fileOpen(
          MDNPBRM_REQ_DESC *pReqDesc,
          TMWTYPES_USHORT requestId,
          DNPDEFS_FILE_MODE mode,
          TMWTYPES_USHORT permissions,
1290      DNPDEFS_FILE_TYPE fileType,
          TMWTYPES_ULONG authenticationKey,
          TMWTYPES_USHORT maxBlockSize,
          const TMWTYPES_CHAR *pFilename);

        /* function: mdnpbrm_fileOpenForWrite
         * purpose: Send a request to open a file for writing or appending on the remote
device. If file
         *  authentication is required the already acquired file authentication
         *  key may be passed to this function. This function was added for writing and
appending because
         *  Time of Creation and File Size should be sent with meaningful values in those
cases. This function can
1300     *  be used for opening for reading, in which case Time of Creation should be Jan
1 1970 and file size should
         *  be zero
         * arguments:
         *  pReqDesc - request descriptor containing generic parameters for
         *   this request
         *  requestId - request id to be sent in request
         *  mode - mode for file open indicating open for reading, writing, or appending
         *  permissions - permissions value to send to outstation, only used if mode ==
DNPDEFS_FILE_MODE_WRITE
         *                these bits may be OR'ed together
         *                DNPDEFS_WORLD_EXECUTE_ALLOWED     0x0001
1310     *                DNPDEFS_WORLD_WRITE_ALLOWED       0x0002
         *                DNPDEFS_WORLD_READ_ALLOWED        0x0004
         *                DNPDEFS_GROUP_EXECUTE_ALLOWED     0x0008
         *                DNPDEFS_GROUP_WRITE_ALLOWED       0x0010
         *                DNPDEFS_GROUP_READ_ALLOWED        0x0020
         *                DNPDEFS_OWNER_EXECUTE_ALLOWED     0x0040
         *                DNPDEFS_OWNER_WRITE_ALLOWED       0x0080
         *                DNPDEFS_OWNER_READ_ALLOWED        0x0100
         *  fileType  DNPDEFS_FILE_TYPE_DIRECTORY if opening a directory for reading
         *            DNPDEFS_FILE_TYPE_SIMPLE if opening a file for reading or writing
1320     *  authentication key - key to be sent in open request. Should be 0 or have
         *   been acquired by mdnpbrm_fileAuthentication
         *  maxBlockSize - maximum block size to be read or written for this file
         *  pFileName - pointer to file or directory to open
         *  timeOfCreation - time of creation for local file on master.
         *  fileSize - size of local file on master if writing or appending, use
0xffffffff if not known, and use 0 if reading
         * returns:
         *  Pointer to request transmit data structure or TMWDEFS_NULL
         */
        TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_fileOpenForWrite(
1330      MDNPBRM_REQ_DESC *pReqDesc,
          TMWTYPES_USHORT requestId,
          DNPDEFS_FILE_MODE mode,
          TMWTYPES_USHORT permissions,
          DNPDEFS_FILE_TYPE fileType,
          TMWTYPES_ULONG authenticationKey,
          TMWTYPES_USHORT maxBlockSize,
          const TMWTYPES_CHAR *pFilename,
          TMWDTIME *pTimeOfCreation,
          TMWTYPES_ULONG fileSize);
1340
        /* function: mdnpbrm_fileClose
         * purpose: Send a request to close a file on the remote device
         * arguments:
         *  pReqDesc - request descriptor containing generic parameters for
         *   this request
         *  requestId - request id to be sent in request.
         *  handle - handle that was returned from remote device when file was opened
         * returns:
```

```
     *  Pointer to request transmit data structure or TMWDEFS_NULL
1350 */
    TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_fileClose(
      MDNPBRM_REQ_DESC *pReqDesc,
      TMWTYPES_USHORT requestId,
      TMWTYPES_ULONG handle);

    /* function: mdnpbrm_fileDelete
     * purpose: send a request to delete a file on the remote device
     * arguments:
     *  pReqDesc - request descriptor containing generic parameters for
1360 *   this request
     *  requestId - request id to be sent in request.
     *  authenticationKey - authentication key to be sent if pUserName is
     *   TMWDEFS_NULL. Could have been acquired by mdnpbrm_fileAuthentication
     *   or could be zero if not required.
     *  pUserName - pointer to Null Terminated user name to be sent to get
     *   authentication key. This should be set to TMWDEFS_NULL if authentication
     *   request is not required to be performed by this function.
     *  pPassword - pointer to Null Terminated password to be sent to get
     *   authentication key
1370 *  pFileName - name of file to be deleted on remote device
     * returns:
     *  Pointer to request transmit data structure or TMWDEFS_NULL
     */
    TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_fileDelete(
      MDNPBRM_REQ_DESC *pReqDesc,
      TMWTYPES_USHORT requestId,
      TMWTYPES_ULONG authenticationKey,
      const TMWTYPES_CHAR *pUserName,
      const TMWTYPES_CHAR *pPassword,
1380  const TMWTYPES_CHAR *pFilename);

    /* function: mdnpbrm_fileAbort
     * purpose:
     * arguments:
     *  pReqDesc - request descriptor containing generic parameters for
     *   this request
     * returns:
     *  Pointer to request transmit data structure or TMWDEFS_NULL
     */
1390 TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_fileAbort(
      MDNPBRM_REQ_DESC *pReqDesc,
      TMWTYPES_USHORT requestId,
      TMWTYPES_ULONG handle);

    /* function: mdnpbrm_fileRead
     * purpose: Send a single file read block request to the remote device.
     *  The remote file should have been previously opened for writing.
     *  When a successful read response is received from the device the
     *  mdnpdata_storeFileData function will be called.
1400 * arguments:
     *  pReqDesc - request descriptor containing generic parameters for
     *   this request
     *  block - block number, should be sequential on each call to this
     *   function beginning with zero after file is opened.
     *  handle - handle that was returned from remote device when file was opened
     * returns:
     *  Pointer to request transmit data structure or TMWDEFS_NULL
     */
    TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_fileRead(
1410    MDNPBRM_REQ_DESC *pReqDesc,
      TMWTYPES_ULONG block,
      TMWTYPES_ULONG handle);

    /* function: mdnpbrm_fileWrite
     * purpose: Send a single file write block request to the remote device.
     *  The remote file should have previously been opened for writing.
     * arguments:
     *  pReqDesc - request descriptor containing generic parameters for
     *   this request
1420 *  block - block number, should be sequential on each call to this
     *   function beginning with zero after file is opened.
     *  last - TMWDEFS_FALSE unless this is the final write to this file
     *   then it should be TMWDEFS_TRUE
     *  handle - handle that was returned from remote device when file was opened
     *  pData - pointer to data to be written to file
     *  length - length of data to be written to file
     * returns:
     *  Pointer to request transmit data structure or TMWDEFS_NULL
     */
```

```
TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_fileWrite(
  MDNPBRM_REQ_DESC *pReqDesc,
  TMWTYPES_ULONG block,
  TMWTYPES_BOOL last,
  TMWTYPES_ULONG handle,
  const TMWTYPES_UCHAR *pData,
  TMWTYPES_USHORT length);

/* function: mdnpbrm_copyLocalFileToRemote
 * purpose: copy a file from local to remote device
 *  This will perform the authentication request if required, open
 *  the local file specified, send the file open request to the remote
 *  device, send the sequential block read requests and then close the
 *  file on the remote device as well as the local file.
 * arguments:
 *  pReqDesc - request descriptor containing generic parameters for
 *   this request
 *  pLocalFileName - name of the local file
 *  pRemoteFileName - name of the remote file
 *  pUserName - the user name for file authentication request.
 *   TMWDEFS_NULL if no authentication to be performed
 *  pPassword - the password for file authentication request.
 *   TMWDEFS_NULL if not required for authentication request.
 *  permissions - permissions value to send to outstation
 *                these bits may be OR'ed together
 *                DNPDEFS_WORLD_EXECUTE_ALLOWED     0x0001
 *                DNPDEFS_WORLD_WRITE_ALLOWED       0x0002
 *                DNPDEFS_WORLD_READ_ALLOWED        0x0004
 *                DNPDEFS_GROUP_EXECUTE_ALLOWED     0x0008
 *                DNPDEFS_GROUP_WRITE_ALLOWED       0x0010
 *                DNPDEFS_GROUP_READ_ALLOWED        0x0020
 *                DNPDEFS_OWNER_EXECUTE_ALLOWED     0x0040
 *                DNPDEFS_OWNER_WRITE_ALLOWED       0x0080
 *                DNPDEFS_OWNER_READ_ALLOWED        0x0100
 * returns:
 *  Pointer to request transmit data structure or TMWDEFS_NULL
 */
TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_copyLocalFileToRemote(
  MDNPBRM_REQ_DESC *pReqDesc,
  const TMWTYPES_CHAR *pLocalFileName,
  const TMWTYPES_CHAR *pRemoteFileName,
  const TMWTYPES_CHAR *pUserName,
  const TMWTYPES_CHAR *pPassword,
  const TMWTYPES_USHORT permissions);

/* function: mdnpbrm_copyRemoteFileToLocal
 * purpose: copy a file from remote to local device.
 *  This will perform the authentication request if required, open
 *  the local file specified, send the file open request to the remote
 *  device, send the sequential block write requests and then close the
 *  file on the remote device as well as the local file.
 * arguments:
 *  pReqDesc - request descriptor containing generic parameters for
 *   this request
 *  pLocalFileName - pointer to NULL terminated name of the local file
 *  pRemoteFileName - pointer to NULL terminated name of the remote file
 *  pUserName - pointer to the NULL terminated user name for the file
 *   authentication request.
 *   TMWDEFS_NULL if no authentication to be performed
 *  pPassword - pointer to the NULL terminated password for the file
 *   authentication request.
 *   TMWDEFS_NULL if not required for authentication request.
 * returns:
 *  Pointer to request transmit data structure or TMWDEFS_NULL
 */
TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_copyRemoteFileToLocal(
  MDNPBRM_REQ_DESC *pReqDesc,
  const TMWTYPES_CHAR *pLocalFileName,
  const TMWTYPES_CHAR *pRemoteFileName,
  const TMWTYPES_CHAR *pUserName,
  const TMWTYPES_CHAR *pPassword);

/* function: mdnpbrm_readRemoteDirectory
 * purpose: Read a directory from a remote device
 * arguments:
 *  pReqDesc - request descriptor containing generic parameters for
 *   this request
 *  pRemoteDirName - pointer to NULL terminated name of the remote directory
 *  pUserName - pointer to the NULL terminated user name for the file
 *   authentication request.
 *   TMWDEFS_NULL if no authentication to be performed
```

```
     *  pPassword - pointer to the NULL terminated password for the file
     *   authentication request.
     *   TMWDEFS_NULL if not required for authentication request.
     * returns:
     *  Pointer to request transmit data structure or TMWDEFS_NULL
     */
    TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_readRemoteDirectory(
      MDNPBRM_REQ_DESC *pReqDesc,
      const TMWTYPES_CHAR *pRemoteDirName,
      const TMWTYPES_CHAR *pUserName,
      const TMWTYPES_CHAR *pPassword);

    /* function: mdnpbrm_writeDeviceAttribute
     * purpose: Issue a device attribute write request using Object 0 variation as
     *  specified.
     * arguments:
     *  pReqDesc - request descriptor containing generic parameters for this request
     *  pUserTxData - pointer to user transmit data structure, see description above
     *  point - point index of device attribute to write to. Use 0 for standard
     *   attribute set. Use indexes other than 0 for user-specific attribute sets.
     *  variation - variation of device attribute to write to
     *   NOTE: device attributes use point number and variation to specify a specific
     *   attribute.   Defines DNPDEFS_OBJ0xxx in dnpdefs.h can be used to identify
     *   the standard attribute variations for Index 0.
     *  pData - pointer to DNPDATA_ATTRIBUTE_VALUE structure specifying value to be
     *   written.
     * returns:
     *  Pointer to request transmit data structure or TMWDEFS_NULL
     */
    TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_writeDeviceAttribute(
      MDNPBRM_REQ_DESC *pReqDesc,
      DNPCHNL_TX_DATA *pUserTxData,
      TMWTYPES_USHORT   point,
      TMWTYPES_UCHAR    variation,
      DNPDATA_ATTRIBUTE_VALUE *pData);

    /* function: mdnpbrm_writeDatasetProto
     * purpose: Issue a data set prototype write request using Object 85 variation 1
     * arguments:
     *  pReqDesc - request descriptor containing generic parameters for this request
     *  pUserTxData - pointer to user transmit data structure, see description above
     *  numObjects - number of prototypes to write
     *  pPointNumbers - if TMWDEFS_NULL, start with first prototype defined on
     *    master and send numObjects prototypes. If not TMWDEFS_NULL this points
     *    to an array of point numbers for prototypes to be written.
     *  NOTE: contents of prototype(s) will be retrieved by
     *    mdnpdata_datasetProtoGet() and not passed as a parameter to this function.
     * returns:
     *  Pointer to request transmit data structure or TMWDEFS_NULL
     */
    TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_writeDatasetProto(
      MDNPBRM_REQ_DESC *pReqDesc,
      DNPCHNL_TX_DATA  *pUserTxData,
      TMWTYPES_UCHAR    numObjects,
      TMWTYPES_USHORT  *pPointNumbers);

    /* function: mdnpbrm_writeDatasetDescr
     * purpose: Issue a write data set descriptor request using Object 86
     *   variation as specified in parameter
     * arguments:
     *  pReqDesc - request descriptor containing generic parameters for this request
     *  pUserTxData - pointer to user transmit data structure, see description above
     *  variation - descriptor variation to use when writing descriptors
     *   (1 Contents or 3 Point Index Attributes)
     *  numObjects - number of descriptors to write
     *  pPointNumbers - if TMWDEFS_NULL, start with first descriptor defined on
     *    master and send numObjects descriptors. If not TMWDEFS_NULL this points
     *    to an array of point numbers for descriptors to be written.
     *  NOTE: contents of descriptor(s) will be retrieved by
     *    mdnpdata_datasetDescrGetCont() or mdnpdata_datasetDescrGetIndex() and not
     *    passed as a parameter to this function.
     * returns:
     *  Pointer to request transmit data structure or TMWDEFS_NULL
     */
    TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_writeDatasetDescr(
      MDNPBRM_REQ_DESC *pReqDesc,
      DNPCHNL_TX_DATA  *pUserTxData,
      TMWTYPES_UCHAR    variation,
      TMWTYPES_UCHAR    numObjects,
      TMWTYPES_USHORT  *pPointNumbers);
```

```
/* function: mdnpbrm_writeDataset
 * purpose: Issue a write data set request using Object 87 variation 1
 * arguments:
 *  pReqDesc - request descriptor containing generic parameters for this request
 *  pUserTxData - pointer to user transmit data structure, see description above
 *  numObjects - number of data sets to write
 *  pPointNumbers - if TMWDEFS_NULL, start with data set id (index) 0
 *    and send numObjects data sets. If not TMWDEFS_NULL this points to an array
 *    of point numbers for data sets to be written.
 *  NOTE: contents of data set(s) will be retrieved by mdnpdata_datasetGet() and not
 *    passed as a parameter to this function.
 * returns:
 *  Pointer to request transmit data structure or TMWDEFS_NULL
 */
TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_writeDataset(
  MDNPBRM_REQ_DESC *pReqDesc,
  DNPCHNL_TX_DATA  *pUserTxData,
  TMWTYPES_UCHAR    numObjects,
  TMWTYPES_USHORT  *pPointNumbers);

/* function: mdnpbrm_controlDataset
 * purpose: Issue a control data set request using Object 87 variation 1
 * arguments:
 *  pReqDesc - request descriptor containing generic parameters for this request
 *  funcCode - function code, must be DNPDEFS_FC_SELECT, DNPDEFS_FC_OPERATE, or
 *   DNPDEFS_FC_DIRECT_OP, or DNPDEFS_FC_DIRECT_OP_NOACK
 *  autoMode - bit mask to specify which actions to perform internally within
 *   the SCL. MDNPBRM_AUTO_MODE_NONE or MDNPBRM_AUTO_MODE_OPERATE.
 *  numObjects - number of data sets to send (currently only 1 supported)
 *  pPointNumbers - if TMWDEFS_NULL, start with data set id (index) 0
 *    and send numObjects data sets. If not TMWDEFS_NULL this points to an array
 *    of point numbers for data sets to be written.
 *  NOTE: contents of data set(s) will be retrieved by mdnpdata_datasetGet() and not
 *    passed as a parameter to this function.
 * returns:
 *  Pointer to request transmit data structure or TMWDEFS_NULL
 * NOTE: If you are doing a SELECT/OPERATE but choose not to use the autoMode
 *  mask to do this, but instead are going to call this brm function to send
 *  the SELECT and then call this function again to send the OPERATE, you will
 *  need to increase the pReqDesc->priority of the OPERATE to 255 to
 *  guarantee it gets sent before any automatic processing of events is performed.
 *  (Automatic processing of events is enabled by default when the mdnp session
 *  was opened). For example, if in the response to the SELECT the restart IIN bit
 *  is set and the session was set up to automatically send a clear restart
 *  command, you will want to make sure the OPERATE gets sent before the clear
 *  restart, otherwise the operate will fail.
 */
TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_controlDataset(
  MDNPBRM_REQ_DESC *pReqDesc,
  TMWTYPES_UCHAR    funcCode,
  MDNPBRM_AUTO_MODE autoMode,
  TMWTYPES_UCHAR    numObjects,
  TMWTYPES_USHORT  *pPointNumbers);

/* function: mdnpbrm_addControlDataset
 * purpose: Issue a control data set request using Object 87 variation 1
 *  NOTE: This function is the equivalent to mdnpbrm_controlDataset except
 *  the parameter pUserData has been added to allow multiple control objects
 *  to be sent in the same request.
 * arguments:
 *  pReqDesc - request descriptor containing generic parameters for this request
 *  pUserTxData - pointer to user transmit data structure, see description above
 *  funcCode - function code, must be DNPDEFS_FC_SELECT, DNPDEFS_FC_OPERATE, or
 *   DNPDEFS_FC_DIRECT_OP
 *  autoMode - bit mask to specify which actions to perform internally within
 *   the SCL. MDNPBRM_AUTO_MODE_NONE or MDNPBRM_AUTO_MODE_OPERATE.
 *  numObjects - number of data sets to send (currently only 1 supported)
 *  pPointNumbers - if TMWDEFS_NULL, start with data set id (index) 0
 *    and send numObjects data sets. If not TMWDEFS_NULL this points to an array
 *    of point numbers for data sets to be written.
 *  NOTE: contents of data set(s) will be retrieved by mdnpdata_datasetGet() and not
 *    passed as a parameter to this function.
 * returns:
 *  Pointer to request transmit data structure or TMWDEFS_NULL
 * NOTE: If you are doing a SELECT/OPERATE but choose not to use the autoMode
 *  mask to do this, but instead are going to call this brm function to send
 *  the SELECT and then call this function again to send the OPERATE, you will
 *  need to increase the pReqDesc->priority of the OPERATE to 255 to
```

```
1670     *  guarantee it gets sent before any automatic processing of events is performed.
         *  (Automatic processing of events is enabled by default when the mdnp session
         *  was opened). For example, if in the response to the SELECT the restart IIN bit
         *  is set and the session was set up to automatically send a clear restart
         *  command, you will want to make sure the OPERATE gets sent before the clear
         *  restart, otherwise the operate will fail.
         */
        TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_addControlDataset(
          MDNPBRM_REQ_DESC *pReqDesc,
          DNPCHNL_TX_DATA  *pUserTxData,
1680      TMWTYPES_UCHAR    funcCode,
          MDNPBRM_AUTO_MODE autoMode,
          TMWTYPES_UCHAR    numObjects,
          TMWTYPES_USHORT  *pPointNumbers);

        /* function: mdnpbrm_datasetExchange
         * purpose: Begin exchange of data set prototype and descriptor objects with
         *   outstation(slave). This consists of a read of two device attributes
         *   indicating the number of prototypes and dataset descriptors defined
         *   on the outstation. The master will then send a request to read the
1690     *   prototypes defined on the slave, followed by a write of the prototypes
         *   defined on the master. The master will then send a request to read the
         *   descriptors defined on the slave, followed by a write of the descriptors
         *   defined on the master.
         * arguments:
         *  pReqDesc - request descriptor containing generic parameters for this request
         * returns:
         *  Pointer to request transmit data structure or TMWDEFS_NULL
         */
        TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_datasetExchange(
1700      MDNPBRM_REQ_DESC *pReqDesc);

        /* function: mdnpbrm_authManualSendSessionKey
        * purpose: Initiate a Secure Authentication Session Key Change sequence
        * arguments:
        *  pReqDesc - request descriptor containing generic parameters for this request
        *  userNumber - User on which to perform the Session Key Change operation.
        *     A Session Key Change is done automatically when needed. This can be used
        *     for testing purposes. You may want to set MDNPSESN_AUTH_CONFIG.testConfig
        *     to prevent automatic sending.
1710    * returns:
        *  Pointer to request transmit data structure or TMWDEFS_NULL
        */
        TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_authManualSendSessionKey(
          MDNPBRM_REQ_DESC *pReqDesc,
          TMWTYPES_USHORT userNumber);

        /* function: mdnpbrm_authUserCertificate
         * purpose: Initiate Secure Authentication Version 5 User Cerificate (instead of
         *  g120v10 Status Change) Sequence, starting by sending a g120v8 User Certificate
1720     *  request to the outstation. This uses a IEC/TS 62351-8 Certificate instead of
         *  a DNP specific set of data.
         * arguments:
         *  pReqDesc - request descriptor containing generic parameters for this request
         *  userNameDbHandle - handle for looking up user data received from the Authority
         *   that was stored in the database and will be retrieved by the SCL.
         *   This includes a IEC/TS 62351-8 Certificate which contains the user's public
key
         *   or an Attribute Certificate which does not.
         *   This data will be accessed by mdnpdata_authGetUserName,
mdnpdata_authGetChangeUserData
         *    tmwcrypto_getCertificate using this handle.
1730     *   This handle will also be passed to mdnpdata_authStoreUpdKeyChangeReply
         *  operation is in certificate
         *  sendUpdateKey -
         *   TMWDEFS_TRUE if User Status Change(g120v8) should be followed by
         *     Update Key Change Request (g120v11) and rest of key change sequence
         *   TMWDEFS_FALSE if Only User Certificate should be sent to Outstation.
         * returns:
         *  Pointer to request transmit data structure or TMWDEFS_NULL
         */
        TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_authUserCertificate(
1740      MDNPBRM_REQ_DESC *pReqDesc,
          void*             userNameDbHandle);

        /* function: mdnpbrm_authUserStatusChange
         * purpose: Initiate Secure Authentication Version 5 User Status Change Sequence
         *  starting by sending a g120v10 User Status Change request to the outstation.
         * arguments:
         *  pReqDesc - request descriptor containing generic parameters for this request
         *  userNameDbHandle - handle for looking up user data received from the Authority
```

```
         *   that was stored in the database and will be retrieved by the SCL. This
includes
1750     *   Status Change Sequence (SCS) number between Authority and Outstation,
         *   A globally unique identifier (name) representing the user,
         *   Interval, Role and Users public key.
         *   Also this same information digitally signed or encrypted depending on whether
         *   symmetric or asymmetric change method is being used. This handle
         *   will be passed to mdnpdata_authGetUserName, mdnpdata_authGetChangeUserData
         *   tmwcrypto_getKey(USER_ASYM_PUB_KEY), mdnpdata_authGetCertData, and
         *   mdnpdata_authStoreUpdKeyChangeReply
         *  operation - operation to perform
         *   DNPAUTH_USER_STATUS_ADD
1760     *   DNPAUTH_USER_STATUS_DELETE
         *   DNPAUTH_USER_STATUS_CHANGE
         *  sendUpdateKey -
         *   TMWDEFS_TRUE if User Status Change(g120v10) should be followed by
         *     Update Key Change Request (g120v11) and rest of key change sequence
         *   TMWDEFS_FALSE if Only User Status Change should be sent to Outstation.
         * returns:
         *  Pointer to request transmit data structure or TMWDEFS_NULL
         */
        TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_authUserStatusChange(
1770      MDNPBRM_REQ_DESC *pReqDesc,
          void             *userNameDbHandle,
          TMWTYPES_UCHAR    operation,
          TMWTYPES_BOOL     sendUpdateKey);

        /* function: mdnpbrm_authUserUpdateKeyChange
         * purpose: Initiate Secure Authentication Version 5 User Update Key Change
         *  or Authentication/Confirm Sequence to verify the existing User Update Key,
         *  starting by sending a g120v11 Update Key Change Request to the outstation.
         * arguments:
1780     *  pReqDesc - request descriptor containing generic parameters for this request
         *  userNameDbHandle - handle for looking up user data received from the Authority
         *   see description above.
         *  sendUpdateKey -
         *   TMWDEFS_TRUE if a new User Update Key should be sent to the Outstation
         *   TMWDEFS_FALSE if the existing User Update Key should be verified on the
Outstation.
         * returns:
         *  Pointer to request transmit data structure or TMWDEFS_NULL
         */
        TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_authUserUpdateKeyChange(
1790      MDNPBRM_REQ_DESC *pReqDesc,
          void *            userNameDbHandle,
          TMWTYPES_BOOL     sendUpdateKey);

        /* function: mdnpbrm_authUserSendSymUpdateKey
         * purpose: Send Secure Authentication Version 5 Symmetric User Update Key
         *  encrypted by the Authority (along with other data)
         *  in a g120v13 Update Key Change to the outstation.
         * arguments:
         *  pReqDesc - request descriptor containing generic parameters for this request
1800     *  userNameDbHandle - handle for looking up ecnrypted user data received from the
         *  Authority
         * returns:
         *  Pointer to request transmit data structure or TMWDEFS_NULL
         */
        TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_authUserSendSymUpdateKey(
          MDNPBRM_REQ_DESC *pReqDesc,
          void *            userNameDbHandle);

        /* function: mdnpbrm_authUserSendg120v6Test
1810     * purpose: For Test Purposes Only. Send just a Secure Authentication g120v6
         *  without first sending a g120v4 as specified by 1815-2012
         * arguments:
         *  pReqDesc - request descriptor containing generic parameters for this request
         *  userNumber - User on which to send the g120v6 Session Key Change request.
         * returns:
         *  Pointer to request transmit data structure or TMWDEFS_NULL
         */
        TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_authUserSendg120v6Test(
          MDNPBRM_REQ_DESC *pReqDesc,
1820      TMWTYPES_USHORT userNumber);

        /* function: mdnpbrm_authUserSendg120v15Test
         * purpose: For Test Purposes Only. Send just a Secure Authentication
         *  User Update Key Confirm a g120v15 to the Outstation. NOTE: This MUST
         *  only be sent after receiving a g120v12 from the Outstation indicating
         *  a User Number. The g120v15 request contains NO User Number so OS must
         *  know what User Number this relates to.
```

```
     * arguments:
     *  pReqDesc - request descriptor containing generic parameters for this request
     *  userNameDbHandle - handle for looking up ecnrypted user data received from the
     *  Authority
     * returns:
     *  Pointer to request transmit data structure or TMWDEFS_NULL
     */
    TMWDEFS_SCL_API TMWSESN_TX_DATA * TMWDEFS_GLOBAL mdnpbrm_authUserSendg120v15Test(
      MDNPBRM_REQ_DESC *pReqDesc,
      void *            userNameDbHandle);

#ifdef __cplusplus
}
#endif
#endif /* MDNPBRM_DEFINED */
```

6 TMW Target Layer Interface

'tmwtarg.h' contains the definition of the interface to the target layer that must be provided for your specific platform. Changes should not be made to this interface, but should be made to the corresponding tmwtarg.c file.

```
/*****************************************************************************/
/* Triangle MicroWorks, Inc.                          Copyright (c) 1997-2022 */
/*****************************************************************************/
/*                                                                           */
/* This file is the property of:                                             */
/*                                                                           */
/*                       Triangle MicroWorks, Inc.                           */
/*                      Raleigh, North Carolina USA                          */
/*                       www.TriangleMicroWorks.com                          */
/*                          (919) 870-6615                                   */
/*                                                                           */
/* This Source Code and the associated Documentation contain proprietary     */
/* information of Triangle MicroWorks, Inc. and may not be copied or         */
/* distributed in any form without the written permission of Triangle        */
/* MicroWorks, Inc.  Copies of the source code may be made only for backup   */
/* purposes.                                                                 */
/*                                                                           */
/* Your License agreement may limit the installation of this source code to  */
/* specific products.  Before installing this source code on a new           */
/* application, check your license agreement to ensure it allows use on the  */
/* product in question.  Contact Triangle MicroWorks for information about   */
/* extending the number of products that may use this source code library or */
/* obtaining the newest revision.                                            */
/*                                                                           */
/*****************************************************************************/

/* file: tmwtarg.h
 * description: This file defines the interface between all Triangle
 *   MicroWorks, Inc. (TMW) source code libraries and the target hardware
 *   and software. This file contains a number of function declarations
 *   (which are implemented in tmwtarg.c) that provide access to required
 *   system resources. The first step in porting a TMW source code library
 *   to your device is to implement each of these functions for your target
 *   platform
 *
 *  It should not be necessary for a target implementor to change anything
 *  in this file. Changes should be made to tmwtarg.c
 */
#ifndef TMWTARG_DEFINED
#define TMWTARG_DEFINED

/* Include target specific header files as required */
#if !defined(_lint)
#if defined(_MSC_VER)
#ifndef _BIND_TO_CURRENT_VCLIBS_VERSION
#define _BIND_TO_CURRENT_VCLIBS_VERSION 1
#endif
#include <stdio.h>
#include <string.h>
#include <memory.h>
#else
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#endif
#endif

/* Triangle MicroWorks, Inc. Header Files */
#include "tmwscl/utils/tmwcnfg.h"
#include "tmwscl/utils/tmwdefs.h"
#include "tmwscl/utils/tmwtypes.h"
#include "tmwscl/utils/tmwdiag.h"
#include "tmwscl/utils/tmwdtime.h"
#include "tmwscl/utils/tmwtargp.h"
#include "tmwtargcnfg.h"
#include "tmwtargos.h"

/* Used to avoid 'unused parameter' warnings
```

```
 */
#ifdef DONT_USE_TMWTARG_UNUSED_PARAM
  #define TMWTARG_UNUSED_PARAM(x)
#else
  #define TMWTARG_UNUSED_PARAM(x) TMWCNFG_UNUSED_PARAM(x)
#endif

/* Indicate what remote UDP port to send the datagram to
 * NOTE: UDP is only supported for DNP
 */
/* Don't use UDP */
#define TMWTARG_UDP_NONE        0

/* Send to the remote port to be used for requests or responses */
#define TMWTARG_UDP_SEND        1

/* Send to the remote port to be used for unsolicited responses
 * Once the outstation has received a request from master this will
 * use the same dest port to be used for all responses.
 */
#define TMWTARG_UDP_SEND_UNSOL  2

/* Send to broadcast address when UDP ONLY is configured */
#define TMWTARG_UDPONLY_BROADCAST 3


/* The following defines may be used when configuring UDP ports.
 *  (These may be redefined to any three ports that are not going
 *  to be used as real UDP port numbers).
 */

/* Don't open a socket for UDP */
#define TMWTARG_UDP_PORT_NONE 0

/* Let the UDP/IP stack determine what port number to use (master) */
#define TMWTARG_UDP_PORT_ANY  1

/* When sending responses use the source port number from the request (slave) */
#define TMWTARG_UDP_PORT_SRC  2

/* Length of strings contained in structures  */
#define TMWTARG_STR_LEN     256

typedef enum TmwTargIpVersionEnum {
  TMWTARG_IPV4,
  TMWTARG_IPV6
} TMWTARG_IP_VERSION;

#define TMWTARG_ADDR_IPV4_LOOPBACK  "127.0.0.1"
#define TMWTARG_ADDR_IPV4_ANY       "0.0.0.0"
#define TMWTARG_ADDR_IPV4_UDP_BCAST "192.168.1.255"

#define TMWTARG_ADDR_IPV6_LOOPBACK  "::1"
#define TMWTARG_ADDR_IPV6_ANY       "::"
#define TMWTARG_ADDR_IPV6_UDP_BCAST "FF02::1"

/* Maximum length for the name of a network interface */
#define TMWTARG_IF_NAME_LENGTH  32

/* Maximum length of common name to expect on incoming TLS certs */
#define TMWTARG_CRYPTO_TLS_NAME_LEN  128

#if !WINIOTARG_SUPPORT_LEGACY_CONFIG
/** parity for RS232 serial communications channel */
typedef enum TmwTarg232Parity
{
  TMWTARG232_PARITY_NONE,   /* no parity */
  TMWTARG232_PARITY_EVEN,   /* even  parity*/
  TMWTARG232_PARITY_ODD     /* odd  parity*/
} TMWTARG232_PARITY;

/** flow control for serial communications */
typedef enum TmwTarg232PortMode
{
  TMWTARG232_MODE_NONE,       /* no flow control */
  TMWTARG232_MODE_HARDWARE,   /* hardware flow control */
  TMWTARG232_MODE_WINDOWS     /* windows flow control */
} TMWTARG232_PORT_MODE;

/** DTR mode for serial communications port when in TMWTARG232_MODE_WINDOWS */
typedef enum TmwTarg232DtrMode
```

```
    {
      TMWTARG232_DTR_DISABLE = 0,     /* Disables the DTR line when the device is opened
                                       * and leaves it disabled. */
      TMWTARG232_DTR_ENABLE = 1,      /* Enables the DTR line when the device is opened
                                       * and leaves it on. */
      TMWTARG232_DTR_HANDSHAKE = 2    /* Enables DTR handshaking. If handshaking is
                                       * enabled, it is an error for the application to
                                       * adjust the line by using the EscapeCommFunction
                                       * function. */
160 } TMWTARG232_DTR_MODE;

    /** RTS mode for serial communications port when in TMWTARG232_MODE_WINDOWS */
    typedef enum TmwTarg232RtsMode
    {
      TMWTARG232_RTS_DISABLE = 0,     /* Disables the RTS line when the device is opened
                                       * and leaves it disabled. */
      TMWTARG232_RTS_ENABLE = 1,      /* Enables the RTS line when the device is opened
                                       * and leaves it on. */
      TMWTARG232_RTS_HANDSHAKE = 2,   /* Enables RTS handshaking. The driver raises the
170                                    * RTS line when the "type-ahead" (input) buffer
                                       * is less than one-half full and lowers the RTS
line
                                       * when the buffer is more than three-quarters
full.
                                       * If handshaking is enabled, it is an error for
the
                                       * application to adjust the line by using the
                                       * EscapeCommFunction function. */
      TMWTARG232_RTS_TOGGLE = 3       /* Specifies that the RTS line will be high if
bytes
                                       * are available for transmission. After all
buffered
                                       * bytes have been sent, the RTS line will be low.
*/
    } TMWTARG232_RTS_MODE;
180 #endif


    /* Maximum length of passphrase for cryptography */
    #define TMWTARG_CRYPTO_PASSPHRASE_LEN    128

    #ifdef _WIN32
    /* Maximum length of IP address strings (IPV6 addresses can be up to 45 bytes) */
    #define TMWTARG_IP_ADDR_LENGTH   128

190 /* Maximum length of path and filenames for cryptography */
    #define TMWTARG_CRYPTO_ID_LEN    512

    #else

    /* Maximum length of IP address strings (IPV6 addresses can be up to 45 bytes) */
    #define TMWTARG_IP_ADDR_LENGTH   64

    /* Maximum length of path and filenames for cryptography */
    #define TMWTARG_CRYPTO_ID_LEN    256  /* Reduce channel memory requirements for
200                                         * non-Windows applications. */

    typedef enum TmwTargTCPRoleEnum {
      /* Master, this only matters for DNP Dual End Point */
      TMWTARGTCP_ROLE_MASTER,

      /* Outstation, this only matters for DNP Dual End Point */
      TMWTARGTCP_ROLE_OUTSTATION
    } TMWTARGTCP_ROLE;

210 /* Ideally there would be a single enumerated type for all target layers.
    * The Windows specific mapping is done to ensure backward compatibility.
    * New target layers should use the non-windows mapping.
    */

     /* Define data types used to interface to the TCP target library. */
    typedef enum TmwTargTCPModeEnum {
      /* listen for connection */
      TMWTARGTCP_MODE_SERVER,

220   /* attempt to make a connection */
      TMWTARGTCP_MODE_CLIENT,

      /* UDP only, no TCP connection */
      TMWTARGTCP_MODE_UDP,
```

```
      /* both client and server functionality */
      TMWTARGTCP_MODE_DUAL_ENDPOINT,

    } TMWTARGTCP_MODE;

    // line up with _WINIO_TYPE_ENUM
    typedef enum TmwTargIOTypeEnum {
      TMWTARGIO_TYPE_232 = 0,
      TMWTARGIO_TYPE_TCP = 4,
      TMWTARGIO_TYPE_UDP_TCP = 5,
      TMWTARGIO_TYPE_NONE = 8
    } TMWTARGIO_TYPE_ENUM;

    typedef enum TmwTarg232StopBitsEnum {
      TMWTARG232_STOP_BITS_1,
      TMWTARG232_STOP_BITS_2
    } TMWTARG232_STOP_BITS;

    typedef enum TmwTarg232DataBitsEnum {
      TMWTARG232_DATA_BITS_7,
      TMWTARG232_DATA_BITS_8
    } TMWTARG232_DATA_BITS;

    #endif

    typedef enum TmwTargChannelState
    {
      TMWTARG_CHANNEL_INITIALIZED = 1,
      TMWTARG_CHANNEL_CLOSED,
      TMWTARG_CHANNEL_OPENED,
    } TMWTARG_CHANNEL_STATE;

    /**
      Data type used to configure the the RS232 interface.
     */
    typedef struct TmwTarg232ConfigStruct {
      char                 chnlName[TMWTARG_STR_LEN];  /* User specified channel name */
      char                 portName[TMWTARG_STR_LEN];  /* "COM1", "COM2", etc. */
      TMWTARG232_PORT_MODE portMode;                   /* hardware, software, windows */
      TMWTARG_TYPE_BAUDRATE baudRate;                  /* in string form; example:
"9600" */
      TMWTARG232_PARITY    parity;                     /* parity */
      TMWTARG232_DATA_BITS numDataBits;                /* 7 or 8 */
      TMWTARG232_STOP_BITS numStopBits;                /* 1 or 2 */
      TMWTYPES_BOOL        bModbusRTU;
      TMWTARG232_DTR_MODE  dtrMode;
      TMWTARG232_RTS_MODE  rtsMode;
      TMWTYPES_BOOL        disabled;
      TMWTYPES_BOOL        polledMode;                 /* Polled or Event Driven Receive
data mode */
    } TMWTARG232_CONFIG;

    /* NOTE: Common TCP config structure, which is convenient for test applications */
    typedef struct TmwTargTCPConfigStruct {
      /* User specified channel name */
      TMWTYPES_CHAR   chnlName[TMWTARG_STR_LEN];

      /* On client -
       *      this is the IP address to set up TCP connection to
       * On server and Dual End Point Device -
       *      this is the IP address to accept TCP connection from
       *      May be *.*.*.* indicating accept connection from any client
       */
      TMWTYPES_CHAR   ipAddress[TMWTARG_IP_ADDR_LENGTH];

      /* Allows binding to a specific address   */
      TMWTYPES_CHAR   localIpAddress[TMWTARG_IP_ADDR_LENGTH];

      /* On client -
      *      this is the port to connect to
      * On server and Dual End Point Device -
      *      this is the port to listen on
      */
      TMWTYPES_USHORT ipPort;

      /* Number of milliseconds to wait for TCP connect to succeed or fail */
      TMWTYPES_ULONG  ipConnectTimeout;

      /* Indicate CLIENT, SERVER, DUAL END POINT, or UDP only
      *  (DUAL END POINT provides both CLIENT and SERVER functionality but
      *  with only one connection at a time)
```

```
     */
    TMWTARGTCP_MODE mode;

    /* If TRUE, when a new connect indication comes in and this channel is
     * already connected, it will be marked for disconnect. This will allow a
     * new connection to come in next time. This handles not receiving notification
     * of disconnect from the remote end, but remote end trying to reconnect.
     * If you want to allow multiple simultaneous connections to multiple channels
     * from any IP address to a particular port number, this parameter should be
     * set to FALSE.
     *
     * For DNP this should be set to TMWDEFS_FALSE according to DNP3 Specification
     *  IP Networking. Keep alive will detect that original connection has
     * failed, which would then allow a new connection to be rcvd.
     */
    TMWTYPES_BOOL   disconnectOnNewSyn;

    /* NOTE: The following configuration parameters are required to support
     *   DNP3 Specification IP Networking. These are not required for
     *   the IEC or Modbus protocols.

     *  Indicate master or outstation (slave) role in dnp networking
     *  as specified by DNP3 Specification IP Networking
     */
    TMWTARGTCP_ROLE role;

    /* If Dual End Point is supported a listen will be done on the above ipPort
     *  and a connection request will be sent to this port number when needed.
     *  This should match ipPort on remote device.
     *  Normal state is listen, connection will be made when there is data to send.
     */
    TMWTYPES_USHORT dualEndPointIpPort;
  #if TMWTARG_SUPPORT_UDP
    /* Destination IP address for UDP broadcast requests.
     * This is only used by a DNP Master when TCP and UDP are supported.
     * If UDP ONLY is configured, ipAddress will be used as destination
     * for all requests.
     */
    TMWTYPES_CHAR     udpBroadcastAddress[TMWTARG_IP_ADDR_LENGTH];

    /* Local port for sending and receiving UDP datagrams on.
     * If this is set to TMWTARG_UDP_PORT_NONE, UDP will not be enabled.
     * For DNP networking UDP should be supported.
     * It is not needed for any of the current IEC or modbus protocols.
     * On Master - If this is set to TMWTARG_UDP_PORT_ANY, an unspecified available
     *             port will be used.
     * On Slave  - This should be chosen to match the UDP port that the master uses
     *             to send Datagram messages to.
     *             This must not be TMWTARGP_UDP_PORT_ANY or TMWTARG_UDP_PORT_SRC.
     */
    TMWTYPES_USHORT localUDPPort;

    /* On Master - if TCP and UDP is configured this specifies the destination UDP/IP
     *              port to send broadcast requests in UDP datagrams to.
     *             if UDP ONLY is configured this specifies the destination UDP/IP
     *              port to send all requests in UDP datagrams to.
     *             This must match the "localUDPPort" on the slave.
     * On Slave  - if TCP and UDP this is not used.
     *             if UDP ONLY is configured this specifies the destination UDP/IP
     *              port to send responses to.
     *              Can be TMWTARG_UDP_PORT_SRC indicating use the src port from a
     *              UDP request received from master.
     */
    TMWTYPES_USHORT destUDPPort;

    /* On master - Not used.
     * On Slave  - if TCP and UDP not used.
     *             if UDP ONLY is configured this specifies the destination UDP/IP
     *              port to send the initial Unsolicited Null response to.
     *              After receiving a UDP request from master, destUDPPort (which)
     *              may indicate use src port) will be used for all responses.
     *              This must not be TMWTARG_UDP_PORT_NONE, TMWTARG_UDP_PORT_ANY, or
     *              TMWTARG_UDP_PORT_SRC for a slave that supports UDP.
     */
    TMWTYPES_USHORT initUnsolUDPPort;

    /* Whether or not to validate source address of received UDP datagram. */
    TMWTYPES_BOOL   validateUDPAddress;
  #endif

    /* Use TLS Transport Layer Security for this channel */
```

```
  TMWTYPES_BOOL   useTLS;

  /* Polled or Event Driven Receive data mode */
  TMWTYPES_BOOL   polledMode;

  /* TLS configuration */
#if TMWTARG_SUPPORT_TLS
  /* File containing the private key for RSA TLS ciphers */
  char tlsRsaPrivateKeyFile[TMWTARG_CRYPTO_ID_LEN];

  /* PassPhrase for decrypting the private key for RSA TLS ciphers */
  char tlsRsaPrivateKeyPassPhrase[TMWTARG_CRYPTO_PASSPHRASE_LEN];

  /* File containing the certificate for key for RSA TLS ciphers */
  char tlsRsaCertificateId[TMWTARG_CRYPTO_ID_LEN];


  /* File containing the private key for DSA TLS ciphers */
  char tlsDsaPrivateKeyFile[TMWTARG_CRYPTO_ID_LEN];

  /* PassPhrase for decrypting the private key for DSA TLS ciphers */
  char tlsDsaPrivateKeyPassPhrase[TMWTARG_CRYPTO_PASSPHRASE_LEN];

  /* File containing the certificate for key for DSA TLS ciphers */
  char tlsDsaCertificateId[TMWTARG_CRYPTO_ID_LEN];


  /* Common name to expect on incoming TLS certs (empty string disables) */
  char tlsCommonName[TMWTARG_CRYPTO_TLS_NAME_LEN];

  /* File containing Certificate Authority Certificates */
  char caFileName[TMWTARG_CRYPTO_ID_LEN];

  /* Path to Directory of Certificate Authority Certificates (instead of caFileName)
*/
  char caPathName[TMWTARG_CRYPTO_ID_LEN];

  /* File containing Certificate Revocation List */
  char caCrlFileName[TMWTARG_CRYPTO_ID_LEN];

  /* Depth of certificate chaining verification */
  TMWTYPES_UCHAR  nCaVerifyDepth;

  /* Max time (seconds) before forcing cipher renegotiation */
  int  nTlsRenegotiationSeconds;

  /* Max PDUs before forcing cipher renegotiation */
  int  nTlsRenegotiationCount;

  /* Max time to wait for client to respond to renegotiation request
   * Not currently used.
   */
  int  nTlsRenegotiationMsTimeout;

  /* Max time in milliseconds to wait for TLS connect handshake to complete */
  int  tlsHandshakeMsTimeout;

  /* File containing DH parameters for TLS cipher suites */
  char dhFileName[TMWTARG_CRYPTO_ID_LEN];
#endif

  /* Allows binding to a specific interface */
  TMWTYPES_CHAR   nicName[TMWTARG_IF_NAME_LENGTH];
  TMWTARG_IP_VERSION ipVersion;
} TMWTARGTCP_CONFIG;

/* The following macros will use the safe string functions
 * available when compiling for windows
 * Modify these as necessary for your target
 */
#if defined(_MSC_VER) && (_MSC_VER >= 1400)
#define STRCPY(dest, size, source) \
  strcpy_s(dest,size,source)

#define STRNCPY(dest, size, source, length) \
  strncpy_s(dest, size, source, length)

#define STRCAT(dest, size, source) \
    strcat_s(dest, size, source);

#else
```

```
    /* If not using windows safe functions */
    #define STRCPY(dest, size, source) \
      strncpy(dest, source, size)

470 #define STRNCPY(dest, size, source, length) \
      strncpy(dest, source, length)

    #define STRCAT(dest, size, source) \
        strcat(dest, source);
    #endif



    /* Define a callback used by the target layer to tell the
480  * source code library (SCL) that the channel connection has been
     * asynchronously opened or closed.
     *  NOTE: This function should only be called after the SCL calls
     *   tmwtarg_openChannel and before it calls tmwtarg_closeChannel.
     *
     * arguments:
     *  pCallbackParam - parameter passed by the SCL to tmwtarg_initChannel
     *   in the target configuration data structure (TMWTARG_CONFIG).
     *  openOrClose -
     *   TMWDEFS_TRUE indicates that the connection has asynchronously opened
490  *    - ie the connection has opened after the previous call to
     *    tmwtarg_openChannel returned TMWDEFS_FALSE
     *   TMWDEFS_FALSE indicates that the connection has asynchronously closed
     *    - ie the connection has closed after the previous call to
     *    tmwtarg_openChannel returned TMWDEFS_TRUE
     *  reason - used to determine diagnostic message when openOrClose==TMWDEFS_FALSE
     */
    typedef void (*TMWTARG_CHANNEL_CALLBACK_FUNC)(
      void *pCallbackParam,
      TMWTYPES_BOOL openOrClose,
500   TMWDEFS_TARG_OC_REASON reason);

    /* Define a callback used by the target channel I/O to tell the
     * source code library that the channel is ready to transmit data.
     * This should be called by the target layer after the call to
     * tmwtarg_getTransmitReady returned a non zero value to the SCL
     */
    typedef void (*TMWTARG_CHANNEL_READY_CBK_FUNC)(
      void *pCallbackParam);

510 /* Define a callback used by the target channel I/O to tell the
     * source code library that the channel has received data
     */
    typedef void (*TMWTARG_CHANNEL_RECEIVE_CBK_FUNC)(
      void *pCallbackParam);

    /* Channel Operation Function Delcarations */
    struct TmwtargIOChannel;
    typedef struct TmwtargIOChannel TMWTARG_IO_CHANNEL;

520 typedef TMWTYPES_BOOL(*TMWTARG_CHANNEL_OPEN_FUNC)(
      TMWTARG_IO_CHANNEL *pTargIoChannel);

    typedef TMWTYPES_MILLISECONDS(*TMWTARG_CHANNEL_XMIT_READY_FUNC)(
      TMWTARG_IO_CHANNEL *pTargIoChannel);

    typedef TMWTYPES_BOOL(*TMWTARG_CHANNEL_XMIT_FUNC)(
      TMWTARG_IO_CHANNEL *pTargIoChannel,
      TMWTYPES_UCHAR *pBuff,
      TMWTYPES_USHORT numBytes);
530
    typedef TMWTYPES_BOOL(*TMWTARG_CHANNEL_XMIT_UDP_FUNC)(
      TMWTARG_IO_CHANNEL *pTargIoChannel,
      TMWTYPES_UCHAR UDPPort,
      TMWTYPES_UCHAR *pBuff,
      TMWTYPES_USHORT numBytes);

    typedef TMWTYPES_USHORT(*TMWTARG_CHANNEL_RECV_FUNC)(
      TMWTARG_IO_CHANNEL *pTargIoChannel,
      TMWTYPES_UCHAR *pBuff,
540   TMWTYPES_USHORT maxBytes,
      TMWTYPES_MILLISECONDS maxTimeout,
      TMWTYPES_BOOL *pInterCharTimeoutOccurred);

    typedef void(*TMWTARG_CHANNEL_CHECK_INPUT_FUNC)(
      TMWTARG_IO_CHANNEL *pTargIoChannel,
      TMWTYPES_MILLISECONDS timeout);
```

```
typedef void(*TMWTARG_CHANNEL_CLOSE_FUNC)(
  TMWTARG_IO_CHANNEL *pTargIoChannel);

/* * The incoming message is for this channel
 *  TMWPHYS_ADDRESS_MATCH_SUCCESS=0,
 *
 * The incoming message may be for this channel,
 * so far the bytes match this protocol,
 * but more bytes are needed to tell if the address matches.
 *  TMWPHYS_ADDRESS_MATCH_MAYBE,
 *
 * The incoming message is not for this channel
 *  TMWPHYS_ADDRESS_MATCH_FAILED
 */
typedef TMWPHYS_ADDRESS_MATCH_TYPE TMWTARG_ADDRESS_MATCH_TYPE;

/* Define a callback used by the target channel I/O to ask if
 * a received message is meant for this channel. This can be used
 * by modem pools to determine which channel to connect an incoming
 * call to.
 */
typedef TMWTARG_ADDRESS_MATCH_TYPE (*TMWTARG_CHECK_ADDRESS_FUNC)(
  void *pCallbackParam,
  TMWTYPES_UCHAR *buf,
  TMWTYPES_USHORT numBytes,
  TMWTYPES_MILLISECONDS firstByteTime);

typedef struct TMWTargConfigStruct {
  /*   * Specifies the amount of time (in character times) to use to
   * determine that a frame has been completed.  For modbus RTU this
   * value is 3.5 (i.e. 4 will be used)
   */
  TMWTYPES_USHORT numCharTimesBetweenFrames;

  /*   * Specifies the amount of time to use to
   * determine that an inter character timeout has occurred.
   * For modbus RTU this value is 1.5 character times (i.e. 2 would be used)
   */
  TMWTYPES_USHORT interCharTimeout;

  /* The following 4 callback parameters are set by the SCL to allow the
   *  target layer to callback to the SCL. These should NOT be set by
   *  the user.
   */

  /*   *  pChannelCallback - function that should be called if the channel
   *   is asynchronously opened or closed (from outside the source code
   *   library). Support for this parameter is optional for most protocols
   *   but recommended for target devices that support asynchronous notification.
   *   For IEC 60870-5-104 this support for this function is required so the SCL
   *   can maintain proper sequence numbers.
   *   The callback can also be called if a low level read or write fails
   *   in the target code as a result of a port being indirectly closed.
   *   This will force the SCL to close the channel and immediately start
   *   trying to reopen it.
   */
  TMWTARG_CHANNEL_CALLBACK_FUNC pChannelCallback;
  /*   *  pCallbackParam - parameter to be passed to channel callback
   */
  void *pCallbackParam;

  /*   *  pChannelReadyCallback - function that may be called to tell the
   *   source code library that the channel is ready to transmit data.
   *   This callback function may be called by the target layer after the call
   *   to tmwtarg_getTransmitReady returns a non zero value to the SCL.
   *   If the target layer does not call this callback function the SCL
   *   will retry after the amount of time indicated by tmwtarg_getTransmitReady
   */
  TMWTARG_CHANNEL_READY_CBK_FUNC pChannelReadyCallback;
  /*   * pChannelReadyCbkParam - parameter to be passed to channel ready callback.
   */
  void *pChannelReadyCbkParam;

  /* This is a pointer to the source code library channel */
  TMWCHNL *pChannel;

  /* Number of Milliseconds to wait before retrying to establish the connection.*/
  TMWTYPES_MILLISECONDS connectRetry;
} TMWTARG_CONFIG;
```

```
/* * Holds values related to the target that have been read from a binary
 * configuration file. (Only used with DNP3).
 */
typedef struct TMWBinFileTargValuesStruct {

  TMWTYPES_BOOL sessionIsOutstation;
  TMWTYPES_BOOL binFileIsOutstation;

  /*1.1.13*/
  TMWTYPES_BOOL useSupportedComm1_1_13;
  TMWTYPES_BOOL supportsSerialConn;
  TMWTYPES_BOOL supportsTCPConn;

  /*1.2.1*/
  TMWTYPES_BOOL useSerialPortName1_2_1;
  TMWTYPES_CHAR serialPortName[TMWTARG_STR_LEN];

  /*1.2.3*/
  TMWTYPES_BOOL useBuadRate1_2_3;
  TMWTYPES_CHAR baudRate[TMWTARG_STR_LEN];

  /*1.2.4 - flow control*/


  /*1.2.6*/
  /*TMWTYPES_BOOL supportsCollisionAvoidance; */
  /*TMWTYPES_MILLISECONDS minBackOffTime; */
  /*TMWTYPES_MILLISECONDS maxRandBackOffTime; */

  /*1.2.7*/
  /*TMWTYPES_BOOL checksRxInterCharGap; */
  /*TMWTYPES_BOOL interRxCharGapAllowed; */
  /*TMWTYPES_MILLISECONDS receiverInterCharTimeout; */

  /*1.2.8*/
  /*TMWTYPES_BOOL txInterCharGaps; */

  /*1.3.1*/
  TMWTYPES_BOOL useIpPortName1_3_1;
  TMWTYPES_CHAR ipPortName[TMWTARG_STR_LEN];

  /*1.3.2*/
  TMWTYPES_BOOL useEndpoint1_3_2;
  TMWTYPES_BOOL endpointIsTcpInitiating;
  TMWTYPES_BOOL endpointIsTcpListening;
  TMWTYPES_BOOL endpointIsTcpDual;
  TMWTYPES_BOOL endpointIsUDPDatagram;

  /*1.3.3*/
  /* IP Address of this device */
  TMWTYPES_BOOL useIpAddress1_3_3;
  TMWTYPES_CHAR ipAddress[TMWTARG_STR_LEN];

  /*1.3.4*/
  TMWTYPES_BOOL useSubnetMask1_3_4;
  TMWTYPES_CHAR subnetMask[TMWTARG_STR_LEN];

  /*1.3.5*/
  TMWTYPES_BOOL useGateWayIp1_3_5;
  TMWTYPES_CHAR gatewayIp[TMWTARG_STR_LEN];

  /*1.3.7*/
  /* Accepts TCP Connections or UDP Datagrams from IPs in the list
   * IPs separated by ";".  Could be *.*.*.*
   * Schema calls this IP Address of Remote Device
   * For master or dual end point this is the address to connect to.
   */
  TMWTYPES_BOOL useAllowedConnIpList1_3_7;
  TMWTYPES_CHAR allowedConnIpList[TMWTARG_STR_LEN];

  /*1.3.8*/
  TMWTYPES_BOOL useTcpListenPort1_3_8;
  TMWTYPES_USHORT tcpListenPort;

  /*1.3.9*/
  TMWTYPES_BOOL useTcpListenPortOfRemote1_3_9;
  TMWTYPES_USHORT tcpListenPortOfRemote;

  /*1.3.11*/
  TMWTYPES_BOOL useLocalUdpPort1_3_11;
```

```
    TMWTYPES_USHORT localUdpPort;

    /*1.3.12*/
    /* used by masters only*/
    TMWTYPES_BOOL useDestUpdPort1_3_12;
    TMWTYPES_USHORT destUdpPort;

    /*1.3.13*/
    /* used by outstations only*/
    TMWTYPES_BOOL useDestUdpPortForUnsol1_3_13;
    TMWTYPES_USHORT destUdpPortForUnsol;

    /*1.3.14*/
    /* used by outstations only*/
    TMWTYPES_BOOL useDestUdpPortForResponses1_3_14;
    TMWTYPES_USHORT destUdpPortForResponses;
    TMWTYPES_BOOL useSourcePortNumberForResponses;

} TMWTARG_BINFILE_VALS;

typedef enum TmwTargThreadState
{
    TMWTARG_THREAD_IDLE,
    TMWTARG_THREAD_EXITED,
    TMWTARG_THREAD_RUNNING,
    TMWTARG_THREAD_EXITING,
} TMWTARG_THREAD_STATE;

#ifdef __cplusplus
extern "C" {
#endif

#if TMWCNFG_SUPPORT_THREADS

  /* If multiple threads executing the SCL is to be supported, support for
   * locking critical resources must be provided. This lock function will be
   * called by the SCL before accessing a common resource. The SCL requires
   * the ability to prevent other threads from acquiring the same lock, but
   * may make nested calls to lock the same resource. If a binary semaphore
   * is the only native mechanism available this may have to be enhanced to
   * provide a counting semaphore.
   */
  TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtarg__lockInit(TMWDEFS_RESOURCE_LOCK
*pLock);
  TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtarg__lockSection(TMWDEFS_RESOURCE_LOCK
*pLock);
  TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtarg__unlockSection(TMWDEFS_RESOURCE_LOCK
*pLock);
  TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtarg__lockDelete(TMWDEFS_RESOURCE_LOCK
*pLock);
  TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtarg__lockShare(TMWDEFS_RESOURCE_LOCK
*pLock,
                                                         TMWDEFS_RESOURCE_LOCK
*pLock1);

  #define TMWTARG_LOCK_INIT(lock)          tmwtarg__lockInit(lock)
  #define TMWTARG_LOCK_SECTION(lock)       tmwtarg__lockSection(lock)
  #define TMWTARG_UNLOCK_SECTION(lock)     tmwtarg__unlockSection(lock)
  #define TMWTARG_LOCK_DELETE(lock)        tmwtarg__lockDelete(lock)

  /* This function is only required for 104 redundancy with a multi-threaded
   * architecture. It will allow the use of a single lock for the redundancy
   * group as well as the redundant connection channels.
   * NOTE: It does not need to be implemented if 104 redundancy with a
   * multi-threaded architecture is not being used.
   */
  #define TMWTARG_LOCK_SHARE(lock, lock1) tmwtarg__lockShare(lock, lock1)

#else

  #define TMWTARG_LOCK_INIT(lock)            ((void) 0)
  #define TMWTARG_LOCK_SECTION(lock)         ((void) 0)
  #define TMWTARG_UNLOCK_SECTION(lock)       ((void) 0)
  #define TMWTARG_LOCK_DELETE(lock)          ((void) 0)

  /* This function is only required for 104 redundancy with a multi-threaded
   * architecture. It will allow the use of a single lock for the redundancy
   * group as well as the redundant connection channels.
   * Copy the information for the lock into lock1 so that the same lock is used
   * for both structures. This cannot be the lock itself. It would typically
   * be a pointer or an index or a reference to the lock.
```

```
     * NOTE: This does not need to be implemented if 104 redundancy with a
     * multi-threaded architecture is not being used.
     */
    #define TMWTARG_LOCK_SHARE(lock, lock1)   ((void) 0)
  #endif

790
  typedef struct TmwtargIOChannel {
    TMWTARGIO_TYPE_ENUM               type;
    TMWTARG_CHANNEL_STATE             chanState;
    TMWCHNL                          *pChannel;
    void                             *pChannelInfo;
    TMWTYPES_CHAR                     chanInfoBuf[64];
    TMWTYPES_CHAR                    *pChannelName;

    /* Callback function for this channel */
800 TMWTARG_CHANNEL_CALLBACK_FUNC     pChannelCallback;      /* From TMWTARG_CONFIG */
    void                             *pChannelCallbackParam; /* From TMWTARG_CONFIG */
    TMWTARG_CHANNEL_READY_CBK_FUNC    pChannelReadyCallback; /* From TMWTARG_CONFIG */
    void                             *pChannelReadyCbkParam; /* From TMWTARG_CONFIG */
    TMWTARG_CHANNEL_RECEIVE_CBK_FUNC  pReceiveCallbackFunc;  /* From openChannel    */
    TMWTARG_CHECK_ADDRESS_FUNC        pCheckAddrCallbackFunc;/* From openChannel    */
    void                             *pCallbackParam;        /* From openChannel,
                                                                * used by both above  */

    /* Channel Operation Functions */
810 TMWTARG_CHANNEL_OPEN_FUNC         pOpenFunction;
    TMWTARG_CHANNEL_XMIT_READY_FUNC   pXmitReadyFunction;
    TMWTARG_CHANNEL_XMIT_FUNC         pXmitFunction;
    TMWTARG_CHANNEL_XMIT_UDP_FUNC     pXmitUdpFunction;
    TMWTARG_CHANNEL_RECV_FUNC         pRecvFunction;
    TMWTARG_CHANNEL_CHECK_INPUT_FUNC  pCheckInputFunction;
    TMWTARG_CHANNEL_CLOSE_FUNC        pCloseFunction;
    TMWTARG_CHANNEL_CLOSE_FUNC        pDeleteFunction;

    TMWTYPES_BOOL                     polledMode;
820 TMWTARG_THREAD_STATE              chanThreadState;

  #if TMWCNFG_SUPPORT_THREADS
    TMW_ThreadId                      chanThreadHandle;
  #endif
  } TMWTARG_IO_CHANNEL;

    /* function: tmwtarg_alloc
     * purpose:  Allocate memory. This function will only be called if
     * TMWCNFG_USE_DYNAMIC_MEMORY is TMWDEFS_TRUE.
830  * arguments:
     *  numBytes - number of bytes requested
     * returns:
     *  pointer to allocated memory if successful
     *  TMWDEFS_NULL if unsuccessful
     */
    TMWDEFS_SCL_API void *TMWDEFS_GLOBAL tmwtarg_alloc(TMWTYPES_UINT numBytes);

    /* function: tmwtarg_calloc
     * purpose:  Allocates storage space for an array of num elements, each of
840  *  length size bytes. Each element is initialized to 0. This function will
     *  only be called if TMWCNFG_USE_DYNAMIC_MEMORY is TMWDEFS_TRUE.
     * arguments:
     *  num - number of items to alloc
     *  size - size of each item
     * returns:
     *  pointer to allocated memory if successful
     *  TMWDEFS_NULL if unsuccessful
     */
    TMWDEFS_SCL_API void * TMWDEFS_GLOBAL tmwtarg_calloc(TMWTYPES_UINT num,
850                                                      TMWTYPES_UINT size);

    /* function: tmwtarg_free
     * purpose:  Free memory allocated by tmwtarg_alloc()
     * arguments:
     *  pBuf - pointer to buffer to be freed
     * returns:
     *  void
     */
    TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtarg_free(void *pBuf);
860
    /* function: tmwtarg_snprintf
     * purpose: Write formatted data to a string.
     * arguments:
     *  buf - Storage location for output
```

```
     *  count - Maximum number of characters that can be stored in buf
     *  format - Format-control string
     *  ... - Optional arguments
     * returns:
     *  TMWTYPES_INT - returns the number of bytes stored in buffer,
870  *  not counting the terminating null character
     */
    TMWDEFS_SCL_API TMWTYPES_INT TMWDEFS_GLOBAL tmwtarg_snprintf(
      TMWTYPES_CHAR *buf,
      TMWTYPES_UINT count,
      const TMWTYPES_CHAR *format,
      ...);

  #if TMWCNFG_SUPPORT_DIAG
    /* function: tmwtarg_putDiagString
880  * purpose: Display a string of characters. This routine is used
     *  to display diagnostic information from the source code library
     *  if desired.
     * arguments:
     *  pAnlzId - pointer to structure containing information about where
     *   and why this message originated.
     *  pString - pointer to null terminated character string to display
     * returns:
     *  void
     */
890  TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtarg_putDiagString(
      const TMWDIAG_ANLZ_ID *pAnlzId,
      const TMWTYPES_CHAR *pString);
  #endif

    /* function: tmwtarg_getMSTime
     * purpose: Return the current value of a continuously running
     *  millisecond timer.
     * arguments:
     *  none
900  * returns:
     *  Current value of millisecond clock.
     */
    TMWDEFS_SCL_API TMWTYPES_MILLISECONDS TMWDEFS_GLOBAL tmwtarg_getMSTime(void);

    /* function: tmwtarg_getDateTime
     * purpose: Return the current date and time. Some protocols (ie DNP3)
     *  are required to use UTC time or if devices span multiple time zones
     *  it may be recommended to use UTC time. This function should either return
     *  local or UTC time as desired.
910  * arguments:
     *  pDateTime - structure into which to store the current date and time
     *  pDateTime->pSession will point to a TMWSESN structure or TMWDEFS_NULL
     *    this allows target layer to return time on a per session basis.
     * returns:
     *  void
     */
    TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtarg_getDateTime(
      TMWDTIME *pDateTime);

920 /* function: tmwtarg_setDateTime
     * purpose: Set the current date and time. This function will only be
     *  called from a slave session as a result of a clock synchronization
     *  request.
     * arguments:
     *  pDateTime - pointer to structure containing new time
     * returns:
     *  TMWTYPES_BOOL - true if success
     */
    TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL tmwtarg_setDateTime(
930   TMWDTIME *pDateTime);

  #if !TMWCNFG_MULTIPLE_TIMER_QS
    /* function: tmwtarg_startTimer()
     * purpose: Start a timer that will call the specified  callback function
     *  in 'timeout' milliseconds. Only a single event timer is required by the
     *  source code library.
     * arguments:
     *  timeout - number of milliseconds to wait
     *    This value can be zero. In that case the timer should call the callback
940  *    function as soon as possible.
     *    NOTE: If this value is too large for the timer implementation, a timer
     *    with the largest supported value should be started. When the callback
     *    is called for the lesser value, the SCL will start another timer with
     *    the remaining time.
     *  pCallbackFunc - function to call when timer expires
```

```
     *  pCallbackParam - user specified callback parameter
     * returns:
     *  void
     *   NOTE: since it is not possible for this function to return failure it is
     *    important that a timer is started. If the timer cannot be started you
     *    should log this in some way or generate an exception since the SCL timers
     *    may not function after this failure. Calling the callback function sooner
     *    than asked for will cause the SCL to call this function again with the
     *    remaining time.
     */
    void TMWDEFS_GLOBAL tmwtarg_startTimer(
      TMWTYPES_MILLISECONDS timeout,
      TMWTYPES_CALLBACK_FUNC pCallbackFunc,
      void *pCallbackParam);

    /* function: tmwtarg_cancelTimer
     * purpose: Cancel current timer
     * arguments:
     *  none
     * returns:
     *  void
     */
    void TMWDEFS_GLOBAL tmwtarg_cancelTimer(void);

#else
    /* if TMWCNFG_MULTIPLE_TIMER_QS are supported */

    /* function: tmwtarg_initMultiTimer()
     * purpose: Create a timer for this channel. This will be a periodic
     *  timer that will be used to call the specified callback function
     *  in 'timeout' milliseconds. One timer will be required per channel.
     *  This is used when multiple timer queues (one per thread) are supported.
     * arguments:
     *  pChannel - pointer to channel to initialize the MultiTimer.
     * returns:
     *  TMWTYPES_INT - 0 if successful, error code on failure.
     */
    TMWTYPES_INT TMWDEFS_GLOBAL tmwtarg_initMultiTimer(
      TMWCHNL                *pChannel);

    /* function: tmwtarg_setMultiTimer()
     * purpose: Set the timeout period for this channel.
     * arguments:
     *  pChannel - pointer to channel to start the MultiTimer.
     *  timeout - number of milliseconds to wait.
     *    A value of zero will cancel the timer.
     *    NOTE: If this value is too large for the timer implementation, a timer
     *    with the largest supported value should be started. When the callback is
     *    called for the lesser value, the SCL will start another timer with the
     *    remaining time.
     * returns:
     *  TMWTYPES_INT - 0 if successful, error code on failure.
     *   NOTE: If the timer cannot be started you should log this in some way or
     *    generate an exception since the SCL timers may not function after this
     *    failure. Calling the callback function sooner than asked for will cause
     *    the SCL to call this function again with the remaining time.
     */
    TMWTYPES_INT TMWDEFS_GLOBAL tmwtarg_setMultiTimer(
      TMWCHNL                *pChannel,
      TMWTYPES_MILLISECONDS   timeout);

    /* function: tmwtarg_deleteMultiTimer
     * purpose: Delete the timer for this channel.
     * arguments:
     *  pChannel - pointer to channel to delete the MultiTimer
     * returns:
     *  TMWTYPES_INT - 0 if successful, error code on failure.
     */
    TMWTYPES_INT TMWDEFS_GLOBAL tmwtarg_deleteMultiTimer(
      TMWCHNL *pChannel);
#endif

    /* function: tmwtarg_exit
     * purpose: Application is notifying the target layer that it is exiting.
     *  Target layer MAY choose to deallocate memory or other resources.
     * arguments:
     *  none
     * returns:
     *  void
     */
    void tmwtarg_exit(void);
```

```
/* function: tmwtarg_initChannel
 * purpose: Initialize a communications channel. This routine creates
 *  a communications channel as specified in the pConfig argument. The
 *  channel does not need to be opened as this will be accomplished in
 *  the tmwtarg_openChannel function described below. This routine
 *  returns a user defined context which is passed to all successive
 *  calls for this channel. The contents of the context are not used
 *  by the TMW SCL and are defined as required by the target interface.
 * arguments:
 *  pUserConfig - Pointer to configuration data passed to the TMW
 *   physical layer code. This data is not used by the TMW code
 *   and should be used by the target routines to identify and
 *   configure the communications channel.
 *  pTmwConfig - TMW target configuration data structure
 *  pChannel - pointer to channel

 * returns:
 *  void * channel context
 *   The channel context is a target-defined context that
 *   will be passed to all of the remaining channel target functions.
 *   The source code library does not change or manipulate this
 *   pointer in any way. The pointer cannot be NULL since this
 *   is interpreted as a failure.
 */
void * TMWDEFS_GLOBAL tmwtarg_initChannel(
  const void *pUserConfig,
  TMWTARG_CONFIG *pTmwConfig,
  TMWCHNL *pChannel);

/* function: tmwtarg_stopThreads
 * purpose: Stop any threads running on this communications channel in
 *  anticipation of tmwtarg_deleteChannel being called. This allows any
 *  threads that might be looking for connections or received data from
 *  calling back into the library and contending for a channel critical
 *  section lock.
 * arguments:
 *  pContext - Context returned from call to tmwtarg_initChannel
 * returns:
 *  void
 */
TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtarg_stopThreads(
    void *pContext);

/* function: tmwtarg_deleteChannel
 * purpose: Delete a communications channel. This routine should
 *  delete a communications channel and free all associated memory
 *  and resources.
 * arguments:
 *  pContext - Context returned from call to tmwtarg_initChannel
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL tmwtarg_deleteChannel(
  void *pContext);

/* function: tmwtarg_getChannelName
 * purpose: Returns the name for this channel
 *  For Diagnostic Purposes only.
 * description: This method allows the target to return an appropriate
 *  name for this channel. Typically this would be something out of the
 *  configuration information passed to the tmwtarg_initChannel routine.
 * arguments:
 *  pContext - Channel context returned from call to tmwtarg_initChannel
 * returns: pointer to a null terminated string which contains the
 *  name.
 */
TMWDEFS_SCL_API const TMWTYPES_CHAR * TMWDEFS_GLOBAL tmwtarg_getChannelName(
  void *pContext);

/* function: tmwtarg_getChannelInfo
 * purpose: Return configuration information for this channel
 * description: This method allows the target to return a user defined
 *  information string to be displayed when the channel is opened.
 *  typically this would contain formatted information about the
 *  channel configuration and/or status.
 * arguments:
 *  pContext - Context returned from call to tmwtarg_initChannel
 * returns:
 *  Pointer to a null terminated string which contains the name.
 */
```

```
const TMWTYPES_CHAR * TMWDEFS_GLOBAL tmwtarg_getChannelInfo(
  void *pContext);

/* function: tmwtarg_openChannel
 * purpose: Open a communications channel.
 *  If this was over TCP/IP this function would attempt to listen
 *  or make the connection. This function should return TMWDEFS_TRUE when
 *  the connection was successfully set up. With an RS232 port this is
 *  probably when the port is opened. With a TCP Server where a listen would
 *  be performed, or a client where a connect request is sent out, this
 *  function should return TMWDEFS_FALSE until the connection is complete.
 *  If this function returns TMWDEFS_FALSE this function will be called
 *  periodically to try to connect. You can also call the pChannelCallback
 *  function, that was passed into tmwtarg_initChannel in the TMWTARG_CONFIG
 *  structure, to indicate the connection has been completed. This will cause
 *  the library to call tmwtarg_openChannel again to allow you to return
 *  TMWDEFS_TRUE. Until tmwtarg_openChannel gets TMWDEFS_TRUE as a return value
 *  the library will not consider the channel is open/connected.
 * arguments:
 *  pContext - Context returned from call to tmwtarg_initChannel
 *  pReceiveCallbackFunc - Function to be called when data is available
 *   to be read if using event driven rather than polled mode. Most
 *   implementations will not need to call this function. This is used for
 *   event driven I/O. NOTE: This callback should not be called until
 *   tmwtarg_openChannel has returned TMWDEFS_TRUE.
 *  pCheckAddrCallbackFunc - Function to be called to determine if this
 *   received data is intended for this channel. This is only supported for
 *   DNP and 101/103. It is intended to provide support for modem pool
 *   implementations for incoming unsolicited messages. Most implementations
 *   will not need to call this function. Note: this callback should not be
 *   called until tmwtarg_openChannel has returned TMWDEFS_TRUE.
 *  pCallbackParam - parameter to be passed to both the pReceivedCallbackFunc
 *   and pCheckAddrCallbackFunc.
 * returns:
 *  TMWDEFS_TRUE if connected (read description in purpose:),
 *  else TMWDEFS_FALSE if connection is not yet completed.
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL tmwtarg_openChannel(
  void *pContext,
  TMWTARG_CHANNEL_RECEIVE_CBK_FUNC pReceiveCallbackFunc,
  TMWTARG_CHECK_ADDRESS_FUNC pCheckAddrCallbackFunc,
  void *pCallbackParam);

/* function: tmwtarg_closeChannel
 * purpose: Close a communications channel
 * arguments:
 *  pContext - Context returned from call to tmwtarg_initChannel
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL tmwtarg_closeChannel(
  void *pContext);

/* function: tmwtarg_getSessionName
 * purpose: Returns the name for this session
 *  For Diagnostic Purposes only.  Registration function also provided.
 * description: This method allows the target to return an appropriate
 *  name for this session. This function could just return the name for the
 *  pSession->pChannel if names are not maintained per session.
 * arguments:
 *  pSession - pointer to session returned by xxxsesn_openSession()
 * returns: pointer to a null terminated string which contains the
 *  name.
 */
TMWDEFS_SCL_API const TMWTYPES_CHAR * TMWDEFS_GLOBAL tmwtarg_getSessionName(
  TMWSESN *pSession);

/* function: tmwtarg_getSectorName
 * purpose: Returns the name for this sector
 *  For Diagnostic Purposes only.  Registration function also provided.
 * description: This method allows the target to return an appropriate
 *  name for this sector. This function could just return the name for the
 *  pSector->pSession or pSector->pChannel if names are not maintained per sector.
 * arguments:
 *  pSession - pointer to session returned by xxxsesn_openSector()
 * returns: pointer to a null terminated string which contains the
 *  name.
 */
TMWDEFS_SCL_API const TMWTYPES_CHAR * TMWDEFS_GLOBAL tmwtarg_getSectorName(
  TMWSCTR *pSector);
```

```
1190  /* function: tmwtarg_isChannelOpen
       * purpose: Determine whether a channel is open/connected
       * arguments:
       *  pContext - Context returned from call to tmwtarg_initChannel
       * returns:
       *  true if channel is open/connected
       */
      TMWTYPES_BOOL tmwtarg_isChannelOpen(void *pContext);

      /* function: tmwtarg_getTransmitReady
1200   * purpose: Determine whether a channel is ready to transmit or not.
       *  This routine can be used to delay transmission until various
       *  target related dependencies have been satisfied. A common
       *  example is modem setup time.
       * arguments:
       *  pContext - Context returned from call to tmwtarg_initChannel
       * returns:
       *  0 if channel is ready to transmit,
       *  non-zero, if channel is not OK to transmit. This value will indicate
       *  the number of milliseconds the SCL should wait before calling this
1210   *  function again for this channel. If the SCL has registered a
       *  TMWTARG_CHANNEL_READY_CBK_FUNC callback function the target layer may
       *  call this callback function if the channel is ready sooner than
       *  this return value would indicate. If the callback function is not
       *  called the SCL will retry this channel in the number of milliseconds
       *  returned by this function.
       */
      TMWTYPES_MILLISECONDS TMWDEFS_GLOBAL tmwtarg_getTransmitReady(
        void *pContext);

1220  /* function: tmwtarg_receive
       * purpose: Receive bytes from the specified channel
       * arguments:
       *  pContext - Context returned from call to tmwtarg_initChannel
       *  pBuff - Buffer into which to store received bytes
       *  maxBytes - The maximum number of bytes to read
       *  maxTimeout - maximum time to wait in milliseconds for input
       *   from this channel.
       *  pInterCharTimeoutOccurred - TMWDEFS_TRUE if an intercharacter
       *   timeout occurred while receiving bytes. This is an optional
1230   *   timeout that can be implemented in the target to terminate
       *   a frame if too much time passes between receipt of bytes
       *   in a frame.
       *  pFirstByteTime - pointer to variable to be filled in indicating
       *   the time the first byte of message was received. If this is left
       *   unchanged the SCL will determine what time it saw the first byte.
       *   This can be set by calling tmwtarg_getMSTime() or its equivalent.
       * returns:
       *  The number of bytes actually read.
       * NOTES:
1240   *  - The Source Code Library will usually use a timeout value of 0;
       *    This indicates the call to tmwtarg_receive should be nonblocking
       *    (i.e., return 0 if no bytes are available.)
       *  - For Modbus RTU this function should not return any bytes
       *    until either the entire frame was received or an inter Character Timeout
       *    occurred. If you are implementing multiple protocols, one of which is
       *    Modbus RTU, then the pContext structure should include a flag that
       *    indicates whether full frames are required. The target implementation
       *    of tmwtarg_receive can use this indicator to ensure that it returns
       *    the entire frame for Modbus RTU. Other protocols can use this
1250   *    indicator to allow them to return any number of bytes actually
       *    read.
       */
      TMWTYPES_USHORT TMWDEFS_GLOBAL tmwtarg_receive(
        void *pContext,
        TMWTYPES_UCHAR *pBuff,
        TMWTYPES_USHORT maxBytes,
        TMWTYPES_MILLISECONDS maxTimeout,
        TMWTYPES_BOOL *pInterCharTimeoutOccurred,
        TMWTYPES_MILLISECONDS *pFirstByteTime);
1260
      /* function: tmwtarg_transmit
       * purpose: Transmit bytes on the specified channel
       * arguments:
       *  pContext - Context returned from call to tmwtarg_initChannel
       *  pBuff - Array of bytes to transmit
       *  numBytes - Number of bytes to transmit
       * returns:
       *  TMWDEFS_TRUE if all the bytes were successfully transmitted,
       *  else TMWDEFS_FALSE.
```

```
 */
TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL tmwtarg_transmit(
  void *pContext,
  TMWTYPES_UCHAR *pBuff,
  TMWTYPES_USHORT numBytes);

/* function: tmwtarg_transmitUDP
 * purpose: Transmit bytes using UDP on the specified channel
 * arguments:
 *  pContext - Context returned from call to tmwtarg_initChannel
 *  UDPPort - This is a define that indicates the remote UDP port to
 *   transmit to.
 *    TMWTARG_UDP_SEND       - Send to the remote port to be used for
 *                             requests or responses
 *    TMWTARG_UDP_SEND_UNSOL - Send to the remote port to be used for
 *                             unsolicited responses.  Once outstation has
 *                             received a request from master this would be
 *                             same port as all responses.
 *    TMWTARG_UDPONLY_BROADCAST Send to the broadcast address when UDP ONLY
 *                             is configured.
 *  pBuff - Array of bytes to transmit
 *  numBytes - Number of bytes to transmit
 * returns:
 *  TMWDEFS_TRUE if all the bytes were successfully transmitted,
 *  else TMWDEFS_FALSE.
 * NOTE: This only needs to be implemented for DNP to support
 *  the DNP3 Specification IP Networking. It is not required
 *  for IEC or modbus and will not be called by those protocols.
 *  If DNP3 UDP is not required, this function can simply return TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL tmwtarg_transmitUDP(
  void *pContext,
  TMWTYPES_UCHAR UDPPort,
  TMWTYPES_UCHAR *pBuff,
  TMWTYPES_USHORT numBytes);

/* Big Endian vs Little Endian
 * For all protocols currently supported by the Triangle MicroWorks
 * source code libraries the message byte order is least significant
 * byte first(LSB). The following get/store routines were rewritten
 * to allow them to work on either a LSB first (little-endian) or Most
 * Significant Byte first(MSB) processors. However, because of differences
 * in the way 64 bit floating point values are stored in memory, it may
 * be necessary to modify tmwtarg_get64 and tmwtarg_put64. (These functions
 * are currently only used by DNP for 64 bit floating point TMWTYPES_DOUBLE
 * and not by the IEC 60870-5 and modbus protocols).
 */

/* function: tmwtarg_get8
 * purpose: retrieve a 8 bit value from a message
 * arguments:
 *  pSource - pointer to location in message buffer to copy bytes from
 *  pDest - pointer to location in memory to copy bytes to.
 * returns:
 *  void
 */
TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtarg_get8(
  const TMWTYPES_UCHAR *pSource,
  TMWTYPES_UCHAR *pDest);

/* function: tmwtarg_store8
 * purpose: store a 8 bit value into a message
 * arguments:
 *  pSource - pointer to location in message buffer to copy bytes from
 *  pDest - pointer to location in memory to copy bytes to.
 * returns:
 *  void
 */
TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtarg_store8(
  const TMWTYPES_UCHAR *pSource,
  TMWTYPES_UCHAR *pDest);

/* function: tmwtarg_get16
 * purpose: retrieve a 16 bit value from a message compensating
 *  for byte order.
 * arguments:
 *  pSource - pointer to location in message buffer to copy bytes from
 *  pDest - pointer to location in memory to copy bytes to.
 * returns:
 *  void
 */
```

```
TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtarg_get16(
  const TMWTYPES_UCHAR *pSource,
  TMWTYPES_USHORT *pDest);

/* function: tmwtarg_store16
 * purpose: store a 16 bit value into a message compensating
 *  for byte order.
 * arguments:
 *  pSource - pointer to location in memory to copy bytes from
 *  pDest - pointer to location in message buffer to copy bytes to
 * returns:
 *  void
 */
TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtarg_store16(
  const TMWTYPES_USHORT *pSource,
  TMWTYPES_UCHAR *pDest);

/* function: tmwtarg_get24
 * purpose: retrieve a 24 bit value from a message compensating
 *  for byte order.
 * arguments:
 *  pSource - pointer to location in message buffer to copy bytes from
 *  pDest - pointer to location in memory to copy bytes to
 * returns:
 *  void
 */
TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtarg_get24(
  const TMWTYPES_UCHAR *pSource,
  TMWTYPES_ULONG *pDest);

/* function: tmwtarg_store24
 * purpose: store a 24 bit value into a message compensating
 *  for byte order.
 * arguments:
 *  pSource - pointer to location in memory to copy bytes from
 *  pDest - pointer to location in message buffer to copy bytes to
 * returns:
 *  void
 */
TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtarg_store24(
  const TMWTYPES_ULONG *pSource,
  TMWTYPES_UCHAR *pDest);

/* function: tmwtarg_get32
 * purpose: retrieve a 32 bit value from a message compensating
 *  for byte order.
 * arguments:
 *  pSource - pointer to location in message buffer to copy bytes from
 *  pDest - pointer to location in memory to copy bytes to
 * returns:
 *  void
 */
TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtarg_get32(
  const TMWTYPES_UCHAR *pSource,
  TMWTYPES_ULONG *pDest);

/* function: tmwtarg_store32
 * purpose: store a 32 bit value into a message compensating
 *  for byte order.
 * arguments:
 *  pSource - pointer to location in memory to copy bytes from
 *  pDest - pointer to location in message buffer to copy bytes to
 * returns:
 *  void
 */
TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtarg_store32(
  const TMWTYPES_ULONG *pSource,
  TMWTYPES_UCHAR *pDest);

/* function: tmwtarg_get64
 * purpose: retrieve a 64 bit value from a message compensating
 *  for byte order.
 * arguments:
 *  pSource - pointer to location in message buffer to copy bytes from
 *  pDest - pointer to location in memory to copy bytes to
 * returns:
 *  void
 */
TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtarg_get64(
  const TMWTYPES_UCHAR *pSource,
  TMWTYPES_DOUBLE *pDest);
```

```
/* function: tmwtarg_store64
 * purpose: store a 64 bit value into a message compensating
 *  for byte order.
 * arguments:
 *  pSource - pointer to location in memory to copy bytes from
 *  pDest - pointer to location in message buffer to copy bytes to
 * returns:
 *  void
 */
TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtarg_store64(
  const TMWTYPES_DOUBLE *pSource,
  TMWTYPES_UCHAR *pDest);

/* function: tmwtarg_getSFloat
 * purpose: retrieve a 32 bit single precision floating point value from a
 *  message compensating for byte order (and floating point format if native
 *  format is not IEEE-754 format as required by DNP).
 * arguments:
 *  pSource - pointer to location in message buffer to copy bytes from
 *  pDest - pointer to location in memory to copy bytes to
 * returns:
 *  void
 */
TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtarg_getSFloat(
  const TMWTYPES_UCHAR *pSource,
  TMWTYPES_SFLOAT *pDest);

/* function: tmwtarg_storeSFloat
 * purpose: store a 32 bit single precision floating point value into a
 *  message compensating for byte order (and floating point format if native
 *  format is not IEEE-754 format as required by DNP).
 * arguments:
 *  pSource - pointer to location in memory to copy bytes from
 *  pDest - pointer to location in message buffer to copy bytes to
 * returns:
 *  void
 */
TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtarg_storeSFloat(
  const TMWTYPES_SFLOAT *pSource,
  TMWTYPES_UCHAR *pDest);

/* function: tmwtarg_appendString
 * purpose: Append two string allocating new memory and freeing original
 *  string. This method is currently only required to support the generation
 *  of an XML document from the target database.
 * arguments:
 *  pStr1 - String to append to, call tmwtarg_free when done
 *  pStr2 - String to append to pStr1
 * returns:
 *  new string which contains str2 appended to str1
 */
TMWTYPES_CHAR *tmwtarg_appendString(
  TMWTYPES_CHAR *pStr1,
  TMWTYPES_CHAR *pStr2);

/* function: tmwtarg_initConfig
 * purpose: Initialize the TMW target layer.
 *  This routine should be called to initialize all the members of the
 *  data structure to the default values. The caller should then modify
 *  individual data fields as desired. The resulting structure will be
 *  passed as an argument to tmwtarg_initChannel
 * arguments:
 *  pConfig - pointer to target layer configuration data structure to
 *   be initialized
 * returns:
 *  void
 */
TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtarg_initConfig(
  TMWTARG_CONFIG *pConfig);

/* The following two functions are only required for if
 * DNPCNFG_SUPPORT_BINCONFIG is defined as TMWDEFS_TRUE
 */

/* function: tmwtarg_initBinFileValues
 * purpose: initialize a struct of target values
 * arguments:
 *  pBinTargFileValues - pointer to struct that will hold target values
 *   read from binary file
 * returns
```

```
     *  TRUE if successful
     */
    TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL tmwtarg_initBinFileValues(
      TMWTARG_BINFILE_VALS *pBinTargFileValues);


    /* function: tmwtarg_applyBinFileTargValues
     * purpose: copy target values from a struct of values read from a
     *  binary file to the target layer
     * arguments:
     *  pBinTargFileValues - pointer to struct holding target values read from binary
file
     * returns
     *  TRUE if successful
     */
    TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL tmwtarg_applyBinFileTargValues(
      void *pIoConfig,
      TMWTARG_BINFILE_VALS *pBinTargFileValues,
      TMWTYPES_BOOL *pIsChannelSerial);

    /* function: tmwtarg_sleep
     * purpose: suspends execution of the calling thread for specified number
     *  of milliseconds
     * arguments:
     *  milliseconds - specifies number of milliseconds to suspend the calling thread
     * returns
     *  void
     */
    TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtarg_sleep(
      TMWTYPES_MILLISECONDS milliseconds);

    /* macro: ASSERT( booleanExpression )
     * purpose: Evaluates its argument. If the result is 0, the macro
     *          prints a diagnostic message and aborts the program.
     *          If the condition is nonzero, it does nothing.  The
     *          diagnostic message has the form
     *          'assertion failed in file <name> in line <num>'
     *          where name is the name of the source file, and num
     *          is the line number of the assertion that failed in
     *          the source file.
     * Note:    This functionality is available only if
     *          TMWCNFG_INCLUDE_ASSERTS is defined and we are
     *          compiling on the microsoft compiler.
     * arguments:
     *   booleanExpression - Specifies an expression (including pointer values) that
     *   evaluates to nonzero or 0.
     *
     * returns:
     *  void
     */

  #if defined(TMW_PRIVATE) && defined(TMWCNFG_INCLUDE_ASSERTS) && defined(_MSC_VER)
    TMWDEFS_SCL_API void TMWAssertion(const char *expr, const char *file, int line);
    #ifdef ASSERT
      #undef ASSERT
    #endif
    #define ASSERT(expr) ((expr) ? ((void) 0) : TMWAssertion(#expr, __FILE__,
__LINE__))
  #else
    #ifdef ASSERT
      #undef ASSERT
    #endif
    #define ASSERT(expr) ((void) 0)
  #endif /* TMW_PRIVATE && TMWCNFG_INCLUDE_ASSERTS */

  #if defined(_MSC_VER)
  /* macro __LOC__
   * purpose: used in #pragma message to place file name and line number
   *          in message.
   * arguments:
   *   none
   * example:
   *   #pragma message(__LOC__ "Is this a reasonable test?");
   * returns:
   *  void
   */
  #ifndef __LOC__
  #define __STR2__(x) #x
  #define __STR1__(x) __STR2__(x)
  #define __LOC__ __FILE__ "(" __STR1__(__LINE__) ") : Note: "
  #endif
```

```
#endif


#ifdef __cplusplus
};
#endif

#endif /* TMWTARG_DEFINED */
```

7 MDNP Database Interface

'mdnpdata.h' contains configuration parameters controlling what Object Groups and Variations are supported, as well as defining the interface to the database routines that must be implemented. Modifying the configuration parameters at the top of the file can reduce functionality but also decrease memory requirements and reduce the amount of database interface code that must be written.

```
/*****************************************************************************/
/* Triangle MicroWorks, Inc.                          Copyright (c) 1997-2022 */
/*****************************************************************************/
/*                                                                           */
/* This file is the property of:                                             */
/*                                                                           */
/*                        Triangle MicroWorks, Inc.                          */
/*                      Raleigh, North Carolina USA                          */
/*                       www.TriangleMicroWorks.com                          */
/*                          (919) 870-6615                                   */
/*                                                                           */
/* This Source Code and the associated Documentation contain proprietary     */
/* information of Triangle MicroWorks, Inc. and may not be copied or         */
/* distributed in any form without the written permission of Triangle        */
/* MicroWorks, Inc.  Copies of the source code may be made only for backup   */
/* purposes.                                                                 */
/*                                                                           */
/* Your License agreement may limit the installation of this source code to  */
/* specific products.  Before installing this source code on a new           */
/* application, check your license agreement to ensure it allows use on the  */
/* product in question.  Contact Triangle MicroWorks for information about   */
/* extending the number of products that may use this source code library or */
/* obtaining the newest revision.                                            */
/*                                                                           */
/*****************************************************************************/

/* file: mdnpdata.h
 * description: This file defines the interface between the Triangle
 *  MicroWorks, Inc. DNP master source code library and the target database.
 */
#ifndef MDNPDATA_DEFINED
#define MDNPDATA_DEFINED

#include "tmwscl/utils/tmwdefs.h"
#include "tmwscl/utils/tmwdtime.h"
#include "tmwscl/utils/tmwcrypto.h"
#include "tmwscl/dnp/dnpcnfg.h"
#include "tmwscl/dnp/mdnpcnfg.h"
#include "tmwscl/dnp/dnputil.h"
#include "tmwscl/dnp/dnpdata.h"
#include "tmwscl/dnp/dnpauth.h"

/* * If you are a.NET Components customer there is no longer a separate
 * config / tmwprvt.h file included from tmwcnfg.h which added functionality
 * not compiled in by default for the ANSI C SCL.This simplifies choosing the
 * functionality a device should support.It is important to select the exact
 * set of features desired, test them thoroughly, and then document them in
 * your Device Profile.
 */

/* This define can be used to restore some functionality that
 * is now compiled out by default in the SCL starting in version 3.29.0.
 * ActivateConfig, Octet String, Virtual Terminal.
 */
#ifndef MDNPDATA_CNFG_LEVEL_TMW
#define MDNPDATA_CNFG_LEVEL_TMW  TMWDEFS_FALSE
#endif

/* Device Attributes */
#ifndef MDNPDATA_SUPPORT_OBJ0
#define MDNPDATA_SUPPORT_OBJ0    TMWDEFS_FALSE
#endif

/* Binary Inputs */
#define MDNPDATA_SUPPORT_OBJ1_V1 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ1_V2 TMWDEFS_TRUE
```

```
#define MDNPDATA_SUPPORT_OBJ1 \
  (MDNPDATA_SUPPORT_OBJ1_V1 | \
   MDNPDATA_SUPPORT_OBJ1_V2)

/* Binary Input Events */
#define MDNPDATA_SUPPORT_OBJ2_V1 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ2_V2 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ2_V3 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ2 \
  (MDNPDATA_SUPPORT_OBJ2_V1 | \
   MDNPDATA_SUPPORT_OBJ2_V2 | \
   MDNPDATA_SUPPORT_OBJ2_V3)

/* Double Bit Inputs */
#define MDNPDATA_SUPPORT_OBJ3_V1 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ3_V2 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ3 \
  (MDNPDATA_SUPPORT_OBJ3_V1 | \
   MDNPDATA_SUPPORT_OBJ3_V2)

/* Double Bit Input Events */
#define MDNPDATA_SUPPORT_OBJ4_V1 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ4_V2 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ4_V3 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ4 \
  (MDNPDATA_SUPPORT_OBJ4_V1 | \
   MDNPDATA_SUPPORT_OBJ4_V2 | \
   MDNPDATA_SUPPORT_OBJ4_V3)

/* Binary Output Status */
#define MDNPDATA_SUPPORT_OBJ10_V1 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ10_V2 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ10 \
  (MDNPDATA_SUPPORT_OBJ10_V1 | \
   MDNPDATA_SUPPORT_OBJ10_V2)

/* If Binary Output Event support is not defined on the command line
 * set the configuration here
 */
#ifndef MDNPDATA_SUPPORT_OBJ11
/* Binary Output Events */
#define MDNPDATA_SUPPORT_OBJ11_V1 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ11_V2 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ11 \
  (MDNPDATA_SUPPORT_OBJ11_V1 | \
   MDNPDATA_SUPPORT_OBJ11_V2)
#endif

/* Control Relay Output Block */
#define MDNPDATA_SUPPORT_OBJ12_V1 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ12_V2 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ12_V3 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ12 \
  (MDNPDATA_SUPPORT_OBJ12_V1 | \
   MDNPDATA_SUPPORT_OBJ12_V2 | \
   MDNPDATA_SUPPORT_OBJ12_V3)

/* Allow mdnpbrm_binaryCommand() to send CROB count value other than 1
 * When this function was first released it did not allow the caller to
 * specify count. In order to prevent unitialized values from being used
 * by these early implementations, this define must be changed to TMWDEFS_TRUE
 * and count in MDNPBRM_CROB_INFO set to the desired value.
 */
#ifndef MDNPDATA_SUPPORT_OBJ12_COUNT
#define MDNPDATA_SUPPORT_OBJ12_COUNT TMWDEFS_FALSE
#endif

/* If Binary Output Command Event support is not defined on the command line
 * set the configuration here
 */
#ifndef MDNPDATA_SUPPORT_OBJ13
/* Binary Output Command Events */
#define MDNPDATA_SUPPORT_OBJ13_V1 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ13_V2 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ13 \
  (MDNPDATA_SUPPORT_OBJ13_V1 | \
   MDNPDATA_SUPPORT_OBJ13_V2)
#endif

/* Binary Counters */
#define MDNPDATA_SUPPORT_OBJ20_V1 TMWDEFS_TRUE
```

```
#define MDNPDATA_SUPPORT_OBJ20_V2 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ20_V5 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ20_V6 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ20 \
  (MDNPDATA_SUPPORT_OBJ20_V1 | \
   MDNPDATA_SUPPORT_OBJ20_V2 | \
   MDNPDATA_SUPPORT_OBJ20_V5 | \
   MDNPDATA_SUPPORT_OBJ20_V6)

/* Frozen Counters */
#define MDNPDATA_SUPPORT_OBJ21_V1 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ21_V2 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ21_V5 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ21_V6 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ21_V9 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ21_V10 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ21 \
  (MDNPDATA_SUPPORT_OBJ21_V1 | \
   MDNPDATA_SUPPORT_OBJ21_V2 | \
   MDNPDATA_SUPPORT_OBJ21_V5 | \
   MDNPDATA_SUPPORT_OBJ21_V6 | \
   MDNPDATA_SUPPORT_OBJ21_V9 | \
   MDNPDATA_SUPPORT_OBJ21_V10)

/* Binary Counter Events */
#define MDNPDATA_SUPPORT_OBJ22_V1 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ22_V2 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ22_V5 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ22_V6 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ22 \
  (MDNPDATA_SUPPORT_OBJ22_V1 | \
   MDNPDATA_SUPPORT_OBJ22_V2 | \
   MDNPDATA_SUPPORT_OBJ22_V5 | \
   MDNPDATA_SUPPORT_OBJ22_V6)

/* Frozen Counter Events */
#define MDNPDATA_SUPPORT_OBJ23_V1 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ23_V2 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ23_V5 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ23_V6 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ23 \
  (MDNPDATA_SUPPORT_OBJ23_V1 | \
   MDNPDATA_SUPPORT_OBJ23_V2 | \
   MDNPDATA_SUPPORT_OBJ23_V5 | \
   MDNPDATA_SUPPORT_OBJ23_V6)

/* Analog Inputs */
#define MDNPDATA_SUPPORT_OBJ30_V1 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ30_V2 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ30_V3 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ30_V4 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ30_V5 TMWCNFG_SUPPORT_FLOAT
#define MDNPDATA_SUPPORT_OBJ30_V6 TMWCNFG_SUPPORT_DOUBLE
#define MDNPDATA_SUPPORT_OBJ30 \
  (MDNPDATA_SUPPORT_OBJ30_V1 | \
   MDNPDATA_SUPPORT_OBJ30_V2 | \
   MDNPDATA_SUPPORT_OBJ30_V3 | \
   MDNPDATA_SUPPORT_OBJ30_V4 | \
   MDNPDATA_SUPPORT_OBJ30_V5 | \
   MDNPDATA_SUPPORT_OBJ30_V6)

/* Frozen Analog Inputs */
#ifndef MDNPDATA_SUPPORT_OBJ31
#define MDNPDATA_SUPPORT_OBJ31_V1 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ31_V2 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ31_V3 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ31_V4 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ31_V5 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ31_V6 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ31_V7 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ31_V8 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ31 \
  (MDNPDATA_SUPPORT_OBJ31_V1 | \
   MDNPDATA_SUPPORT_OBJ31_V2 | \
   MDNPDATA_SUPPORT_OBJ31_V3 | \
   MDNPDATA_SUPPORT_OBJ31_V4 | \
   MDNPDATA_SUPPORT_OBJ31_V5 | \
   MDNPDATA_SUPPORT_OBJ31_V6 | \
   MDNPDATA_SUPPORT_OBJ31_V7 | \
   MDNPDATA_SUPPORT_OBJ31_V8)
#endif
```

```
/* Analog Input Events */
#define MDNPDATA_SUPPORT_OBJ32_V1 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ32_V2 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ32_V3 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ32_V4 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ32_V5 TMWCNFG_SUPPORT_FLOAT
#define MDNPDATA_SUPPORT_OBJ32_V6 TMWCNFG_SUPPORT_DOUBLE
#define MDNPDATA_SUPPORT_OBJ32_V7 TMWCNFG_SUPPORT_FLOAT
#define MDNPDATA_SUPPORT_OBJ32_V8 TMWCNFG_SUPPORT_DOUBLE
#define MDNPDATA_SUPPORT_OBJ32 \
  (MDNPDATA_SUPPORT_OBJ32_V1 | \
   MDNPDATA_SUPPORT_OBJ32_V2 | \
   MDNPDATA_SUPPORT_OBJ32_V3 | \
   MDNPDATA_SUPPORT_OBJ32_V4 | \
   MDNPDATA_SUPPORT_OBJ32_V5 | \
   MDNPDATA_SUPPORT_OBJ32_V6 | \
   MDNPDATA_SUPPORT_OBJ32_V7 | \
   MDNPDATA_SUPPORT_OBJ32_V8)

/* Frozen Analog Input Events */
#ifndef MDNPDATA_SUPPORT_OBJ33
#define MDNPDATA_SUPPORT_OBJ33_V1 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ33_V2 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ33_V3 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ33_V4 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ33_V5 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ33_V6 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ33_V7 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ33_V8 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ33 \
  (MDNPDATA_SUPPORT_OBJ33_V1 | \
   MDNPDATA_SUPPORT_OBJ33_V2 | \
   MDNPDATA_SUPPORT_OBJ33_V3 | \
   MDNPDATA_SUPPORT_OBJ33_V4 | \
   MDNPDATA_SUPPORT_OBJ33_V5 | \
   MDNPDATA_SUPPORT_OBJ33_V6 | \
   MDNPDATA_SUPPORT_OBJ33_V7 | \
   MDNPDATA_SUPPORT_OBJ33_V8)
#endif

/* Analog Input Deadbands */
#define MDNPDATA_SUPPORT_OBJ34_V1 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ34_V2 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ34_V3 TMWCNFG_SUPPORT_FLOAT
#define MDNPDATA_SUPPORT_OBJ34 \
  (MDNPDATA_SUPPORT_OBJ34_V1 | \
   MDNPDATA_SUPPORT_OBJ34_V2 | \
   MDNPDATA_SUPPORT_OBJ34_V3)

/* Analog Output Status */
#define MDNPDATA_SUPPORT_OBJ40_V1 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ40_V2 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ40_V3 TMWCNFG_SUPPORT_FLOAT
#define MDNPDATA_SUPPORT_OBJ40_V4 TMWCNFG_SUPPORT_DOUBLE
#define MDNPDATA_SUPPORT_OBJ40 \
  (MDNPDATA_SUPPORT_OBJ40_V1 | \
   MDNPDATA_SUPPORT_OBJ40_V2 | \
   MDNPDATA_SUPPORT_OBJ40_V3 | \
   MDNPDATA_SUPPORT_OBJ40_V4)

/* Analog Output Control Block */
#define MDNPDATA_SUPPORT_OBJ41_V1 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ41_V2 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ41_V3 TMWCNFG_SUPPORT_FLOAT
#define MDNPDATA_SUPPORT_OBJ41_V4 TMWCNFG_SUPPORT_DOUBLE
#define MDNPDATA_SUPPORT_OBJ41 \
  (MDNPDATA_SUPPORT_OBJ41_V1 | \
   MDNPDATA_SUPPORT_OBJ41_V2 | \
   MDNPDATA_SUPPORT_OBJ41_V3 | \
   MDNPDATA_SUPPORT_OBJ41_V4)

/* If Analog Output Event support is not defined on the command line
 * set the configuration here
 */
#ifndef MDNPDATA_SUPPORT_OBJ42
/* Analog Output Events */
#define MDNPDATA_SUPPORT_OBJ42_V1 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ42_V2 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ42_V3 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ42_V4 TMWDEFS_FALSE
```

```
#define MDNPDATA_SUPPORT_OBJ42_V5 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ42_V6 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ42_V7 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ42_V8 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ42 \
  (MDNPDATA_SUPPORT_OBJ42_V1 | \
   MDNPDATA_SUPPORT_OBJ42_V2 | \
   MDNPDATA_SUPPORT_OBJ42_V3 | \
   MDNPDATA_SUPPORT_OBJ42_V4 | \
   MDNPDATA_SUPPORT_OBJ42_V5 | \
   MDNPDATA_SUPPORT_OBJ42_V6 | \
   MDNPDATA_SUPPORT_OBJ42_V7 | \
   MDNPDATA_SUPPORT_OBJ42_V8)
#endif

/* If Analog Output Command Event support is not defined on the command line
 * set the configuration here
 */
#ifndef MDNPDATA_SUPPORT_OBJ43
/* Analog Output Command Events */
#define MDNPDATA_SUPPORT_OBJ43_V1 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ43_V2 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ43_V3 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ43_V4 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ43_V5 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ43_V6 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ43_V7 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ43_V8 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ43 \
  (MDNPDATA_SUPPORT_OBJ43_V1 | \
   MDNPDATA_SUPPORT_OBJ43_V2 | \
   MDNPDATA_SUPPORT_OBJ43_V3 | \
   MDNPDATA_SUPPORT_OBJ43_V4 | \
   MDNPDATA_SUPPORT_OBJ43_V5 | \
   MDNPDATA_SUPPORT_OBJ43_V6 | \
   MDNPDATA_SUPPORT_OBJ43_V7 | \
   MDNPDATA_SUPPORT_OBJ43_V8)
#endif

/* Time and Date */
#define MDNPDATA_SUPPORT_OBJ50_V1 TMWDEFS_TRUE
#define MDNPDATA_SUPPORT_OBJ50_V3 TMWDEFS_TRUE
#ifndef MDNPDATA_SUPPORT_OBJ50_V4
#define MDNPDATA_SUPPORT_OBJ50_V4 TMWDEFS_FALSE
#endif
#define MDNPDATA_SUPPORT_OBJ50 \
  (MDNPDATA_SUPPORT_OBJ50_V1 | \
   MDNPDATA_SUPPORT_OBJ50_V3 | \
   MDNPDATA_SUPPORT_OBJ50_V4)

/* File Transfer */
#ifndef MDNPDATA_SUPPORT_OBJ70
#define MDNPDATA_SUPPORT_OBJ70_V2 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ70_V3 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ70_V4 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ70_V5 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ70_V6 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ70_V7 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ70 \
  (MDNPDATA_SUPPORT_OBJ70_V2 | \
   MDNPDATA_SUPPORT_OBJ70_V3 | \
   MDNPDATA_SUPPORT_OBJ70_V4 | \
   MDNPDATA_SUPPORT_OBJ70_V5 | \
   MDNPDATA_SUPPORT_OBJ70_V6 | \
   MDNPDATA_SUPPORT_OBJ70_V7)
#endif

/* IIN bits */
#define MDNPDATA_SUPPORT_OBJ80    TMWDEFS_TRUE

/* If Data Set support is not defined on the command line
 * set the configuration here
 */
#ifndef MDNPDATA_SUPPORT_DATASETS

/* Data Set Prototypes */
#define MDNPDATA_SUPPORT_OBJ85    TMWDEFS_FALSE

/* Data Set Descriptors */
#define MDNPDATA_SUPPORT_OBJ86_V1 TMWDEFS_FALSE
#define MDNPDATA_SUPPORT_OBJ86_V2 TMWDEFS_FALSE
```

```
    #define MDNPDATA_SUPPORT_OBJ86_V3 TMWDEFS_FALSE
    #define MDNPDATA_SUPPORT_OBJ86 \
      (MDNPDATA_SUPPORT_OBJ86_V1 | \
       MDNPDATA_SUPPORT_OBJ86_V2 | \
       MDNPDATA_SUPPORT_OBJ86_V3)

    /* Data Set Present Value */
    #define MDNPDATA_SUPPORT_OBJ87    TMWDEFS_FALSE

400 /* Data Set Snapshot Events */
    #define MDNPDATA_SUPPORT_OBJ88    TMWDEFS_FALSE

    /* Data Sets in general */
    #define MDNPDATA_SUPPORT_DATASETS \
      (MDNPDATA_SUPPORT_OBJ85 | \
       MDNPDATA_SUPPORT_OBJ86 | \
       MDNPDATA_SUPPORT_OBJ87 | \
       MDNPDATA_SUPPORT_OBJ88)
    #endif
410
    /* Activate Configuration Response */
    #define MDNPDATA_SUPPORT_OBJ91  MDNPDATA_CNFG_LEVEL_TMW

    /* String Data */
    #define MDNPDATA_SUPPORT_OBJ110 MDNPDATA_CNFG_LEVEL_TMW

    /* String Events */
    #define MDNPDATA_SUPPORT_OBJ111 MDNPDATA_CNFG_LEVEL_TMW

420 /* Virtual Terminal Output */
    #define MDNPDATA_SUPPORT_OBJ112 MDNPDATA_CNFG_LEVEL_TMW

    /* Virtual Terminal Events */
    #define MDNPDATA_SUPPORT_OBJ113 MDNPDATA_CNFG_LEVEL_TMW

    /* Extended String Data */
    #ifndef MDNPDATA_SUPPORT_OBJ114
    #define MDNPDATA_SUPPORT_OBJ114_V1 TMWDEFS_FALSE
    #define MDNPDATA_SUPPORT_OBJ114_V2 TMWDEFS_FALSE
430 #define MDNPDATA_SUPPORT_OBJ114_V3 TMWDEFS_FALSE
    #define MDNPDATA_SUPPORT_OBJ114_V4 TMWDEFS_FALSE
    #define MDNPDATA_SUPPORT_OBJ114 \
      (MDNPDATA_SUPPORT_OBJ114_V1 | \
       MDNPDATA_SUPPORT_OBJ114_V2 | \
       MDNPDATA_SUPPORT_OBJ114_V3 | \
       MDNPDATA_SUPPORT_OBJ114_V4)
    #endif

    /* Extended String Events */
440 #ifndef MDNPDATA_SUPPORT_OBJ115
    #define MDNPDATA_SUPPORT_OBJ115_V1 TMWDEFS_FALSE
    #define MDNPDATA_SUPPORT_OBJ115_V2 TMWDEFS_FALSE
    #define MDNPDATA_SUPPORT_OBJ115_V3 TMWDEFS_FALSE
    #define MDNPDATA_SUPPORT_OBJ115_V4 TMWDEFS_FALSE
    #define MDNPDATA_SUPPORT_OBJ115 \
      (MDNPDATA_SUPPORT_OBJ115_V1 | \
       MDNPDATA_SUPPORT_OBJ115_V2 | \
       MDNPDATA_SUPPORT_OBJ115_V3 | \
       MDNPDATA_SUPPORT_OBJ115_V4)
450 #endif

    /* If Secure Authentication support is not defined on the command line
     * set the configuration here.
     * DNPCNFG_SUPPORT_AUTHENTICATION must also be defined appropriately
     * object group 121 and 122 are supported if object group 120 is supported
     */
    #ifndef MDNPDATA_SUPPORT_OBJ120
    #define MDNPDATA_SUPPORT_OBJ120 DNPCNFG_SUPPORT_AUTHENTICATION
    /* Secure Authentication Statistics Static and Event Objects are required if SA is
supported */
460 #endif

    /* Secure Authentication User Certificate for Asymmetric Remote Key Update */
    #ifndef MDNPDATA_SUPPORT_OBJ120_V8
    #define MDNPDATA_SUPPORT_OBJ120_V8 TMWDEFS_FALSE
    #endif


    /* If in Startup state, we should discard and not confirm
     * unsolicited responses until an integrity poll completes.
470  * App Layer Volume 2 Part 2 1.1.2 and Part 3 1.5.
```

```
     * This means even initial unsolicited NULL responses should be discarded.
     * The previous behavior of the SCL was to accept and confirm them. To allow
     * NULL Unsoliciteds to be accepted even before the integrity poll completes
     * set this define to TMWDEFS_FALSE.
     */
    #define MDNPDATA_DISCARD_NULL_UNSOL TMWDEFS_TRUE


    /* Secure Authentication Event that occurred */
480 typedef enum mdnpauth_event
    {
      /* Rcvd IIN 2-2 Function Code Not Implemented in response to Read Status g120v4
       * indicating outstation does not support Secure Authentication.
       */
      MDNPAUTH_EV_V4_BAD_FUNCTION,

      /* Rcvd IIN 2-2 Function Code Not Implemented in response to change user status
g120v10
       * indicating outstation does not support Secure Authentication Remote Key Update.
       */
490   MDNPAUTH_EV_V10_BAD_FUNCTION
    } MDNPAUTH_EVENT_ENUM;


    #ifdef __cplusplus
    extern "C" {
    #endif

      /* function: mdnpdata_processIIN
       * purpose: Process Internal Indication Bits received from remote device.
500    *  This routine gives the user the chance to process the IIN bits before
       *  the SCL processing. The user can clear any IIN bits they do not want
       *  the SCL to process.
       * arguments:
       *  pSession - pointer to session
       *  pIIN - pointer to IIN bits
       * returns:
       *  void
       */
      void TMWDEFS_GLOBAL mdnpdata_processIIN(
510     TMWSESN *pSession,
        TMWTYPES_USHORT *pIIN);

      /* function: mdnpdata_storeIIN
       * purpose: Store Internal Indication Bit received from remote device
       *  in response to read of object 80 variation 1. This would typically be
       *  used to read private IIN bits outside the range of 0-15, but could
       *  also used for the standard IIN bits.
       * arguments:
       *  pHandle - database handle returned from mdnpdata_init
520    *  pointNumber - index of IIN bit to be stored
       *  value - value of IIN bit to be stored.
       * returns:
       *  void
       */
      void TMWDEFS_GLOBAL mdnpdata_storeIIN(
        void *pHandle,
        TMWTYPES_USHORT pointNumber,
        TMWTYPES_BOOL value);

530   /* function: mdnpdata_init
       * purpose: Initialize DNP3 Master database on specified session
       * arguments:
       *  pSession - pointer to session on which to create database
       *  pUserHandle - user specified database handle passed to session open
       * returns:
       *  pointer to database handle for future database calls
       */
      void * TMWDEFS_GLOBAL mdnpdata_init(
        TMWSESN *pSession,
540     void *pUserHandle);

      /* function: mdnpdata_close
       * purpose: Close target database. After this call the database
       *  handle will be invalid.
       * arguments:
       *  pHandle - database handle returned from mdnpdata_init
       * returns:
       *  void
       */
550   void TMWDEFS_GLOBAL mdnpdata_close(
```

```
  void *pHandle);

/* function: mdnpdata_storeReadTime
 * purpose: Store time read from slave returned in Obj 50 Variation 1
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  pTimeStamp - pointer to time stamp returned in Obj 50 Variation 1
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL mdnpdata_storeReadTime(
 void *pHandle,
 TMWDTIME *pTimeStamp);

/* function: mdnpdata_storeBinaryInput
 * purpose: Store binary input value
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  pointNumber - point number of point to store
 *  flags - DNP3 flags (including value) to store. Flag values are OR'd
 *    combinations of the following:
 *      DNPDEFS_DBAS_FLAG_ON_LINE - the binary input point has been read
 *        successfully
 *      DNPDEFS_DBAS_FLAG_RESTART - the field device that originated the
 *        data object has been restarted. This device may be the deviced
 *        reporting this data object.
 *      DNPDEFS_DBAS_FLAG_COMM_LOST - the device reporting this data object
 *        has lost communication with the originator of the data object
 *      DNPDEFS_DBAS_FLAG_REMOTE_FORCED - the state of the binary object
 *        has been forced to its current state at the originating device
 *      DNPDEFS_DBAS_FLAG_LOCAL_FORCED - the state of the binary object
 *        has been forced to its current state at the device reporting
 *        this data object
 *      DNPDEFS_DBAS_FLAG_CHATTER - the binary input point has been filtered
 *        in order to remove unneeded transitions in the state of the input
 *      DNPDEFS_DBAS_FLAG_BINARY_ON  - the current state of the input (On)
 *  isEvent - TMWDEFS_TRUE if this update was caused by a change event
 *  pTimeStamp - pointer to time stamp reported with this change event
 *   or TMWDEFS_NULL if no time was reported. Will always be TMWDEFS_NULL
 *   if isEvent is TMWDEFS_FALSE.
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_storeBinaryInput(
  void *pHandle,
  TMWTYPES_USHORT pointNumber,
  TMWTYPES_UCHAR flags,
  TMWTYPES_BOOL isEvent,
  TMWDTIME *pTimeStamp);

/* function: mdnpdata_storeDoubleInput
 * purpose: Store double bit input value
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  pointNumber - point number of point to store
 *  flags - DNP3 flags (including value) to store. Flag values are OR'd
 *    combinations of the following:
 *      DNPDEFS_DBAS_FLAG_ON_LINE - the binary input point has been read
 *        successfully
 *      DNPDEFS_DBAS_FLAG_RESTART - the field device that originated the
 *        data object has been restarted. This device may be the deviced
 *        reporting this data object.
 *      DNPDEFS_DBAS_FLAG_COMM_LOST - the device reporting this data object
 *        has lost communication with the originator of the data object
 *      DNPDEFS_DBAS_FLAG_REMOTE_FORCED - the state of the binary object
 *        has been forced to its current state at the originating device
 *      DNPDEFS_DBAS_FLAG_LOCAL_FORCED - the state of the binary object
 *        has been forced to its current state at the device reporting
 *        this data object
 *      DNPDEFS_DBAS_FLAG_CHATTER - the binary input point has been filtered
 *        in order to remove unneeded transitions in the state of the input
 *      DNPDEFS_DBAS_FLAG_DOUBLE_INTER
 *        the current state of the input (Intermediate -transitioning condition)
 *      DNPDEFS_DBAS_FLAG_DOUBLE_OFF
 *        the current state of the input (Off)
 *      DNPDEFS_DBAS_FLAG_DOUBLE_ON
 *        the current state of the input (On)
 *      DNPDEFS_DBAS_FLAG_DOUBLE_INDET
 *        the current state of the input (Indeterminate -abnormal or custom
 *        condition)
 *  isEvent - TMWDEFS_TRUE if this update was caused by a change event
```

```
 *  pTimeStamp - pointer to time stamp reported with this change event
 *   or TMWDEFS_NULL if no time was reported. Will always be TMWDEFS_NULL
 *   if isEvent is TMWDEFS_FALSE.
 *  timeQualifier - specifies how the timestamp was determined if
 *   pTimeStamp is not TMWDEFS_NULL
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_storeDoubleInput(
  void *pHandle,
  TMWTYPES_USHORT pointNumber,
  TMWTYPES_UCHAR flags,
  TMWTYPES_BOOL isEvent,
  TMWDTIME *pTimeStamp);

/* function: mdnpdata_storeBinaryOutput
 * purpose: Store binary output value
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  pointNumber - point number of point to store
 *  flags - DNP3 flags (including value) to store. Flag values are OR'd
 *    combinations of the following:
 *      DNPDEFS_DBAS_FLAG_ON_LINE - the binary input point has been read
 *        successfully
 *      DNPDEFS_DBAS_FLAG_RESTART - the field device that originated the
 *        data object has been restarted. This device may be the deviced
 *        reporting this data object.
 *      DNPDEFS_DBAS_FLAG_COMM_LOST - the device reporting this data object
 *        has lost communication with the originator of the data object
 *      DNPDEFS_DBAS_FLAG_REMOTE_FORCED - the state of the binary object
 *        has been forced to its current state at the originating device
 *      DNPDEFS_DBAS_FLAG_LOCAL_FORCED - the state of the binary object
 *        has been forced to its current state at the device reporting
 *        this data object
 *      DNPDEFS_DBAS_FLAG_BINARY_ON  - the current state of the input (On)
 *  isEvent - TMWDEFS_TRUE if this update was caused by a change event
 *  pTimeStamp - pointer to time stamp reported with this change event
 *   or TMWDEFS_NULL if no time was reported. Will always be TMWDEFS_NULL
 *   if isEvent is TMWDEFS_FALSE.
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_storeBinaryOutput(
  void *pHandle,
  TMWTYPES_USHORT pointNumber,
  TMWTYPES_UCHAR flags,
  TMWTYPES_BOOL isEvent,
  TMWDTIME *pTimeStamp);

/* function: mdnpdata_storeBinaryCmdStatus
 * purpose: Store binary command status returned from slave
 *  NOTE: this will NOT be called in the case that an binary output request
 *   succeeded and all statuses are zero. Applications should register a
 *   brm request callback function to receive the status of the actual request.
 *   This function is intended to be called to indicate the point that had the
 *   failed status. This function would also be called if a g13 event was
 *   received from the outstation.
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  pointNumber - point number of point to store
 *  status - status returned from slave. Top bit(0x80) is Commanded State.
 *  isEvent - TMWDEFS_TRUE if this update was caused by a change event
 *  pTimeStamp - pointer to time stamp reported with this change event
 *   or TMWDEFS_NULL if no time was reported. Will always be TMWDEFS_NULL
 *   if isEvent is TMWDEFS_FALSE.
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_storeBinaryCmdStatus(
  void *pHandle,
  TMWTYPES_USHORT pointNumber,
  TMWTYPES_UCHAR status,
  TMWTYPES_BOOL isEvent,
  TMWDTIME *pTimeStamp);

/* function: mdnpdata_storeBinaryCounter
 * purpose: Store binary counter value
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  pointNumber - point number of point to store
 *  value - new counter value
```

```
 *  flags - DNP3 flags to store. Flag values are OR'd
 *    combinations of the following:
 *      DNPDEFS_DBAS_FLAG_ON_LINE - the binary input point has been read
 *        successfully
 *      DNPDEFS_DBAS_FLAG_RESTART - the field device that originated the
 *        data object has been restarted. This device may be the deviced
 *        reporting this data object.
720  *      DNPDEFS_DBAS_FLAG_COMM_LOST - the device reporting this data object
 *        has lost communication with the originator of the data object
 *      DNPDEFS_DBAS_FLAG_REMOTE_FORCED - the state of the binary object
 *        has been forced to its current state at the originating device
 *      DNPDEFS_DBAS_FLAG_LOCAL_FORCED - the state of the binary object
 *        has been forced to its current state at the device reporting
 *        this data object
 *      DNPDEFS_DBAS_FLAG_CNTR_ROLLOVER - the accumulated value has exceeded
 *        has exceeded its maximum and rolled over to zero.
 *        NOTE: This maximum value is not necessarily equal to (2^32-1) for
730  *        32 bit counters or (2^16-1) for 16 bit counters. It can be different
 *        for each counter instance. Technical Bulletin TB-2002-001 Counter
 *        Objects recommends "slave devices do not set the Rollover flag and
 *        that host(Master) devices ignore the Rollover flag".
 *  isEvent - TMWDEFS_TRUE if this update was caused by a change event
 *  pTimeStamp - pointer to time stamp reported with this change event
 *   or TMWDEFS_NULL if no time was reported. Will always be TMWDEFS_NULL
 *   if isEvent is TMWDEFS_FALSE.
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
740  */
TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_storeBinaryCounter(
  void *pHandle,
  TMWTYPES_USHORT pointNumber,
  TMWTYPES_ULONG value,
  TMWTYPES_UCHAR flags,
  TMWTYPES_BOOL isEvent,
  TMWDTIME *pTimeStamp);

/* function: mdnpdata_storeFrozenCounter
750  * purpose: Store frozen counter value
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  pointNumber - point number of point to store
 *  value - new counter value
 *  flags - DNP3 flags to store. Flag values are OR'd
 *    combinations of the following:
 *   The following values (or OR'd combinations) are valid for this type:
 *      DNPDEFS_DBAS_FLAG_ON_LINE - the binary input point has been read
 *        successfully
760  *      DNPDEFS_DBAS_FLAG_RESTART - the field device that originated the
 *        data object has been restarted. This device may be the deviced
 *        reporting this data object.
 *      DNPDEFS_DBAS_FLAG_COMM_LOST - the device reporting this data object
 *        has lost communication with the originator of the data object
 *      DNPDEFS_DBAS_FLAG_REMOTE_FORCED - the state of the binary object
 *        has been forced to its current state at the originating device
 *      DNPDEFS_DBAS_FLAG_LOCAL_FORCED - the state of the binary object
 *        has been forced to its current state at the device reporting
 *        this data object
770  *      DNPDEFS_DBAS_FLAG_CNTR_ROLLOVER - the accumulated value has exceeded
 *        has exceeded its maximum and rolled over to zero.
 *        NOTE: This maximum value is not necessarily equal to (2^32-1) for
 *        32 bit counters or (2^16-1) for 16 bit counters. It can be different
 *        for each counter instance. Technical Bulletin TB-2002-001 Counter
 *        Objects recommends "slave devices do not set the Rollover flag and
 *        that host(Master) devices ignore the Rollover flag".
 *  isEvent - TMWDEFS_TRUE if this update was caused by a change event
 *  pTimeStamp - pointer to time stamp reported with this message
 *   or TMWDEFS_NULL if no time was reported. Note that pTimeStamp
780  *   can be non null for frozen counters in which case the time will
 *   be the reported time of the last freeze for this counter.
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_storeFrozenCounter(
  void *pHandle,
  TMWTYPES_USHORT pointNumber,
  TMWTYPES_ULONG value,
  TMWTYPES_UCHAR flags,
790  TMWTYPES_BOOL isEvent,
  TMWDTIME *pTimeStamp);

/* function: mdnpdata_storeAnalogInput
```

```
 * purpose: Store analog input value
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  pointNumber - point number of point to store
 *  value - new value
 *  flags - DNP3 flags to store. Flag values are OR'd
 *    combinations of the following:
 *      DNPDEFS_DBAS_FLAG_ON_LINE - the binary input point has been read
 *        successfully
 *      DNPDEFS_DBAS_FLAG_RESTART - the field device that originated the
 *        data object has been restarted. This device may be the deviced
 *        reporting this data object.
 *      DNPDEFS_DBAS_FLAG_COMM_LOST - the device reporting this data object
 *        has lost communication with the originator of the data object
 *      DNPDEFS_DBAS_FLAG_REMOTE_FORCED - the state of the binary object
 *        has been forced to its current state at the originating device
 *      DNPDEFS_DBAS_FLAG_LOCAL_FORCED - the state of the binary object
 *        has been forced to its current state at the device reporting
 *        this data object
 *      DNPDEFS_DBAS_FLAG_OVER_RANGE - the digitized signal or calculation
 *        is greater than the type specified in TMWTYPES_ANALOG_VALUE. If the
 *        SCL determines that the value returned cannot fit in the type
 *        specified by the object variation read it will set this OVER_RANGE bit.
 *      DNPDEFS_DBAS_FLAG_REFERENCE_CHK - the reference signal used to
 *        digitize the signal is not stable, and the resulting digitized
 *        value may not be correct.
 *  isEvent - TMWDEFS_TRUE if this update was caused by a change event
 *  pTimeStamp - pointer to time stamp reported with this change event
 *   or TMWDEFS_NULL if no time was reported. Will always be TMWDEFS_NULL
 *   if isEvent is TMWDEFS_FALSE.
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_storeAnalogInput(
  void *pHandle,
  TMWTYPES_USHORT pointNumber,
  TMWTYPES_ANALOG_VALUE *pValue,
  TMWTYPES_UCHAR flags,
  TMWTYPES_BOOL isEvent,
  TMWDTIME *pTimeStamp);

 /* function: mdnpdata_storeFrozenAnalogInput
 * purpose: Store frozen analog input value
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  pointNumber - point number of point to store
 *  value - new value
 *  flags - DNP3 flags to store. Flag values are OR'd
 *    combinations of the following:
 *      DNPDEFS_DBAS_FLAG_ON_LINE - the binary input point has been read
 *        successfully
 *      DNPDEFS_DBAS_FLAG_RESTART - the field device that originated the
 *        data object has been restarted. This device may be the deviced
 *        reporting this data object.
 *      DNPDEFS_DBAS_FLAG_COMM_LOST - the device reporting this data object
 *        has lost communication with the originator of the data object
 *      DNPDEFS_DBAS_FLAG_REMOTE_FORCED - the state of the binary object
 *        has been forced to its current state at the originating device
 *      DNPDEFS_DBAS_FLAG_LOCAL_FORCED - the state of the binary object
 *        has been forced to its current state at the device reporting
 *        this data object
 *      DNPDEFS_DBAS_FLAG_OVER_RANGE - the digitized signal or calculation
 *        is greater than the type specified in TMWTYPES_ANALOG_VALUE. If the
 *        SCL determines that the value returned cannot fit in the type
 *        specified by the object variation read it will set this OVER_RANGE bit.
 *      DNPDEFS_DBAS_FLAG_REFERENCE_CHK - the reference signal used to
 *        digitize the signal is not stable, and the resulting digitized
 *        value may not be correct.
 *  isEvent - TMWDEFS_TRUE if this update was caused by a change event
 *  pTimeStamp - pointer to time stamp reported with this change event
 *   or TMWDEFS_NULL if no time was reported. Will always be TMWDEFS_NULL
 *   if isEvent is TMWDEFS_FALSE.
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_storeFrozenAnalogInput(
  void *pHandle,
  TMWTYPES_USHORT pointNumber,
  TMWTYPES_ANALOG_VALUE *pValue,
  TMWTYPES_UCHAR flags,
  TMWTYPES_BOOL isEvent,
```

```
  TMWDTIME *pTimeStamp);

/* function: mdnpdata_storeAnalogInputDeadband
 * purpose: Store analog input deadband
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  pointNumber - point number of point to store
 *  value - new value
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_storeAnalogInputDeadband(
  void *pHandle,
  TMWTYPES_USHORT pointNumber,
  TMWTYPES_ANALOG_VALUE *pValue);

/* function: mdnpdata_storeAnalogOutput
 * purpose: Store analog output value
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  pointNumber - point number of point to store
 *  value - new value
 *  flags - DNP3 flags to store. Flag values are OR'd
 *    combinations of the following:
 *      DNPDEFS_DBAS_FLAG_ON_LINE - the binary input point has been read
 *        successfully
 *      DNPDEFS_DBAS_FLAG_RESTART - the field device that originated the
 *        data object has been restarted. This device may be the deviced
 *        reporting this data object.
 *      DNPDEFS_DBAS_FLAG_COMM_LOST - the device reporting this data object
 *        has lost communication with the originator of the data object
 *      DNPDEFS_DBAS_FLAG_REMOTE_FORCED - the state of the binary object
 *        has been forced to its current state at the originating device
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_storeAnalogOutput(
  void *pHandle,
  TMWTYPES_USHORT pointNumber,
  TMWTYPES_ANALOG_VALUE *pValue,
  TMWTYPES_UCHAR flags,
  TMWTYPES_BOOL isEvent,
  TMWDTIME *pTimeStamp);

/* function: mdnpdata_storeAnalogCmdStatus
 * purpose: Store analog output command status returned from slave
 *  NOTE: this will NOT be called in the case that an analog output request
 *   succeeded and all statuses are zero. Applications should register a
 *   brm request callback function to receive the status of the actual request.
 *   This function is intended to be called to indicate the point that had the
 *   failed status. This function would also be called if a g43 event was
 *   received from the outstation.
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  pointNumber - point number of point to store
 *  pValue - value returned from slave
 *  status - status of command returned from slave
 *  isEvent - TMWDEFS_TRUE if this update was caused by a change event
 *  pTimeStamp - pointer to time stamp reported with this change event
 *   or TMWDEFS_NULL if no time was reported. Will always be TMWDEFS_NULL
 *   if isEvent is TMWDEFS_FALSE.
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_storeAnalogCmdStatus(
  void *pHandle,
  TMWTYPES_USHORT pointNumber,
  TMWTYPES_ANALOG_VALUE *pValue,
  TMWTYPES_UCHAR status,
  TMWTYPES_BOOL isEvent,
  TMWDTIME *pTimeStamp);

/* function: mdnpdata_storeRestartTime
 * purpose: Store time value from restart response containing
 *  object 52 var 1 or 2
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  time - time in milliseconds
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
```

```
TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_storeRestartTime(
  void *pHandle,
  TMWTYPES_ULONG time);

/* function: mdnpdata_storeBinaryOutput
 * purpose: Store binary output value
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  pointNumber - point number of point to store
 *  indexedTime - indexed absolute time
 *  intervalCount - interval count
 *  intervalUnit - interval unit
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_storeIndexedTime(
  void *pHandle,
  TMWTYPES_USHORT pointNumber,
  TMWDTIME indexedTime,
  TMWTYPES_ULONG intervalCount,
  TMWTYPES_BYTE intervalUnit
  );

/* function: mdnpdata_storeString
 * purpose: Store string value
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  pointNumber - point number of point to store
 *  pStrBuf - pointer to buffer with string value
 *  strLen - number of bytes in value
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_storeString(
  void *pHandle,
  TMWTYPES_USHORT pointNumber,
  TMWTYPES_UCHAR *pStrBuf,
  TMWTYPES_UCHAR strLen);

/* function: mdnpdata_storeExtString
 * purpose: Store extended string value
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  pointNumber - point number of point to store
 *  pStrBuf - pointer to buffer with string value
 *  strLen - number of bytes in value
 *  flags - DNP3 flags to store. Flag values are OR'd
 *    combinations of the following:
 *      DNPDEFS_DBAS_FLAG_ON_LINE - the binary input point has been read
 *        successfully
 *      DNPDEFS_DBAS_FLAG_RESTART - the field device that originated the
 *        data object has been restarted. This device may be the deviced
 *        reporting this data object.
 *      DNPDEFS_DBAS_FLAG_COMM_LOST - the device reporting this data object
 *        has lost communication with the originator of the data object
 *      DNPDEFS_DBAS_FLAG_REMOTE_FORCED - the state of the binary object
 *        has been forced to its current state at the originating device
 *  isEvent - TMWDEFS_TRUE if this update was caused by a change event
 *  pTimeStamp - pointer to time stamp reported with this change event
 *   or TMWDEFS_NULL if no time was reported. Will always be TMWDEFS_NULL
 *   if isEvent is TMWDEFS_FALSE.
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_storeExtString(
  void *pHandle,
  TMWTYPES_USHORT pointNumber,
  TMWTYPES_UCHAR *pStrBuf,
  TMWTYPES_USHORT strLen,
  TMWTYPES_UCHAR flags,
  TMWTYPES_BOOL isEvent,
  TMWDTIME *pTimeStamp);

/* function: mdnpdata_storeActiveConfig
 * purpose: Store Active Configuration Response
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  index - index since multiple statuses may be returned.
 *  timeDelay - how long to delay
 *  statusCode - status code
 *  pStrBuf - pointer to error string
```

```
 *  strLen - number of bytes in error string
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_storeActiveConfig(
  void *pHandle,
  TMWTYPES_UCHAR index,
  TMWTYPES_ULONG timeDelay,
  TMWTYPES_UCHAR statusCode,
  TMWTYPES_UCHAR *pStrBuf,
  TMWTYPES_UCHAR strLen);

/* function: mdnpdata_storeVirtualTerminal
 * purpose: Store virtual terminal value
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  pointNumber - point number of point to store
 *  pVtermBuf - pointer to buffer with virtual terminal value
 *  vtermLen - number of bytes in value
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_storeVirtualTerminal(
  void *pHandle,
  TMWTYPES_USHORT pointNumber,
  TMWTYPES_UCHAR *pVtermBuf,
  TMWTYPES_UCHAR vtermLen);

/* function: mdnpdata_storeAuthKey
 * purpose: Store file authentication key
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  authKey - authentication key returned from slave
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL mdnpdata_storeFileAuthKey(
  void *pHandle,
  TMWTYPES_ULONG authKey);

/* function: mdnpdata_storeFileStatus
 * purpose: Store file status
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  handle - file handle
 *  size - file size if open for read (NULL otherwise)
 *  maxBlockSize - maximum block size outstation will return
 *  requestId - id of request submitted
 *  status - status/error condition
 *  nOptionalChars - number of bytes in pOptionalChars
 *  pOptionalChars - Optional ASCII characters or TMWDEFS_NULL if none.
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL mdnpdata_storeFileStatus(
  void *pHandle,
  TMWTYPES_ULONG handle,
  TMWTYPES_ULONG fileSize,
  TMWTYPES_USHORT maxBlockSize,
  TMWTYPES_USHORT requestId,
  DNPDEFS_FILE_CMD_STAT status,
  TMWTYPES_USHORT nOptionalChars,
  const TMWTYPES_CHAR *pOptionalChars);

/* function: mdnpdata_storeFileData
 * purpose: Store file data
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  handle - file handle
 *  blockNumber - current file block number
 *  lastBlockFlag - TRUE if last block
 *  nBytesInBlockData - number of bytes in pBlockData
 *  pBlockData - the data for this block or TMWDEFS_NULL if none.
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL mdnpdata_storeFileData(
  void *pHandle,
  TMWTYPES_ULONG handle,
  TMWTYPES_ULONG blockNumber,
  TMWTYPES_BOOL lastBlockFlag,
```

```
  TMWTYPES_USHORT nBytesInBlockData,
  const TMWTYPES_UCHAR *pBlockData);

/* function: mdnpdata_storeFileDataStatus
 * purpose: Store file data status
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  handle - file handle
 *  blockNumber - current file block number
 *  lastBlockFlag - TRUE if last block
 *  status - status/error condition
 *  nOptionalChars - number of bytes in pOptionalChars
 *  pOptionalChars - Optional ASCII characters or TMWDEFS_NULL if none.
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL mdnpdata_storeFileDataStatus(
  void *pHandle,
  TMWTYPES_ULONG handle,
  TMWTYPES_ULONG blockNumber,
  TMWTYPES_BOOL lastBlockFlag,
  DNPDEFS_FILE_TFER_STAT status,
  TMWTYPES_USHORT nOptionalChars,
  const TMWTYPES_CHAR *pOptionalChars);

/* function: mdnpdata_storeFileInfo
 * purpose: Store file info for each file/directory returned
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  fileNameOffset - offset to file name in message. Probably no value.
 *  fileNameSize - length of pFileName
 *  fileType - simple file(1) or directory(0)
 *  fileSize - number of bytes in file
 *  pFileCreationTime - creation time of the file
 *  permissions - permissions of file
 *  pFileName - pointer to the file name or TMWDEFS_NULL if none.
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL mdnpdata_storeFileInfo(
  void *pHandle,
  TMWTYPES_USHORT fileNameOffset,
  TMWTYPES_USHORT fileNameSize,
  DNPDEFS_FILE_TYPE fileType,
  TMWTYPES_ULONG  fileSize,
  TMWDTIME *pFileCreationTime,
  DNPDEFS_FILE_PERMISSIONS permissions,
  const TMWTYPES_CHAR *pFileName);

/* function: mdnpdata_getFileAuthKey
 * purpose: Return the authentication key provided by the slave
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 * returns:
 *  TMWTYPES_ULONG the authentication key value
 */
TMWTYPES_ULONG mdnpdata_getFileAuthKey(
  void *pHandle);

/* function: mdnpdata_openLocalFile
 * purpose: Open a file on the local file system
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  pLocalFileName - name of local file to open, null terminated string.
 *  fileMode - simple file(1) or directory(0)
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL mdnpdata_openLocalFile(
  void *pHandle,
  const TMWTYPES_CHAR *pLocalFileName,
  DNPDEFS_FILE_MODE fileMode);

/* function: mdnpdata_closeLocalFile
 * purpose: Close a file on the local file system
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL mdnpdata_closeLocalFile(
```

```
  void *pHandle);

/* function: mdnpdata_getLocalFileInfo
 * purpose: Get size and time of creation for a file on the local file system
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  pLocalFileName - name of local file to get info for, null terminated string.
 *  pFileSize - size of file or 0xffffffff if not known
 *  pDateTime - time of creation for file or Jan 1 1970 if not known.
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL mdnpdata_getLocalFileInfo(
  void *pHandle,
  const TMWTYPES_CHAR *pLocalFileName,
  TMWTYPES_ULONG *pFileSize,
  TMWDTIME *pDateTime);

/* function: mdnpdata_readLocalFile
 * purpose: Read data from a local file and place it in a buffer
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  pLocalFileHandle - local file handle
 *  pBuf - buffer to fill
 *  bufSize - number of bytes to put in pBuf
 *  pLastBlock - fill this in to indicate last block
 * returns:
 *  TMWTYPES_USHORT - number of bytes read
 */
TMWTYPES_USHORT mdnpdata_readLocalFile(
  void *pHandle,
  TMWTYPES_UCHAR *pBuf,
  TMWTYPES_USHORT bufSize,
  TMWTYPES_BOOL *pLastBlock);

/* function: mdnpdata_storeDeviceAttribute
 * purpose:  Store particular device attribute value.
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  point - point index of device attribute to store
 *   0 for standard attributes, other values for user-specific attribute set
 *  variation - variation of attribute, used like an index to determine which
 *   attribute.
 *  pData - pointer to structure contaning attribute value to store
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL mdnpdata_storeDeviceAttribute(
  void *pHandle,
  TMWTYPES_USHORT point,
  TMWTYPES_UCHAR variation,
  DNPDATA_ATTRIBUTE_VALUE *pData);

/* function: mdnpdata_storeDeviceAttrProperty
 * purpose:  Store property value for a particular device attribute.
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  point - point index of device attribute property to store
 *   0 for standard attributes, other values for user-specific attribute set
 *  variation - variation of attribute, used like an index to determine which
 *   attribute.
 *  property - property of device attribute
 *   0x00 indicates attribute is NOT writable by master
 *   0x01 indicates attribute is writable by master
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL mdnpdata_storeDeviceAttrProperty(
  void *pDbHandle,
  TMWTYPES_USHORT point,
  TMWTYPES_UCHAR variation,
  TMWTYPES_UCHAR property);

/* function: mdnpdata_datasetProtoQuantity
 * purpose: Get quantity of Data Set Prototypes in the database.
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 * returns:
 *  Quantity of prototypes in database
 */
TMWTYPES_USHORT mdnpdata_datasetProtoQuantity(
```

```
1280      void *pHandle);

        /* function: mdnpdata_datasetProtoSlaveQty
         * purpose: Get quantity of Data Set Prototypes in the database that
         *  were read from the slave. Prototype ids must start with 0 and be
         *  contiguous, however the prototypes defined by the outstation or slave
         *  are numbered 0 to n-1. The ones defined by the master should be numbered
         *  n to x-1. Where n is the number of prototypes defined by the outstation,
         *  and x is the number of prototypes defined by the master and outstation
         *  combined.
1290     * arguments:
         *  pHandle - database handle returned from mdnpdata_init
         * returns:
         *  Quantity of prototypes read from the slave
         *  NOTE: this would be the number of different prototypes written to the
         *   database by mdnpdata_storeDatasetProtoUUID()
         */
        TMWTYPES_USHORT mdnpdata_datasetProtoSlaveQty(
          void *pHandle);

1300    /* function: mdnpdata_storeDatasetProtoUUID
         * purpose: Create an entry for a Data Set prototype if it does not
         *  exist and store this Universally Unique Identifier (UUID).
         * arguments:
         *  pHandle - database handle returned from mdnpdata_init
         *  pointNumber - point number or prototype id of Data Set prototype to store
         *  pUUID - pointer to 16 octet UUID.
         * returns:
         *  void
         */
1310    void TMWDEFS_GLOBAL mdnpdata_storeDatasetProtoUUID(
          void *pHandle,
          TMWTYPES_USHORT pointNumber,
          TMWTYPES_UCHAR *pUUID);

        /* function: mdnpdata_storeDatasetProto
         * purpose: Store Data Set prototype element data.
         * arguments:
         *  pHandle - database handle returned from mdnpdata_init
         *  pointNumber - point number or prototype id of prototype to store
1320     *  elemIndex - index of prototype element to store starting with zero
         *   for the first element after mandatory prototype ID and UUID. If a
         *   Namespace and Name are present these will be elemIndex 0 and 1.
         *  pDescr - pointer to descriptor element structure
         *   NOTE: if pDescr->ancillaryValue->type == DNPDATA_VALUE_STRPTR, this is just
         *    a pointer, the string itself must be copied somewhere.
         * returns:
         *  void
         */
        void TMWDEFS_GLOBAL mdnpdata_storeDatasetProto(
1330      void *pHandle,
          TMWTYPES_USHORT pointNumber,
          TMWTYPES_UCHAR  elemIndex,
          DNPDATA_DATASET_DESCR_ELEM *pDescr);

        /* function: mdnpdata_datasetProtoGet
         * purpose: Get a prototype from database for this prototype id.
         * arguments:
         *  pHandle - database handle returned from mdnpdata_init
         *  pointNumber - point number or prototype id of prototype to get
1340     *  pNumberElems - number of elements returned for specified Data Set Prototype
         *   not counting the mandatory prototype id and UUID, which should not
         *   be contained in the array of elements.
         *  pUUID - pointer to a 16 byte array to be filled in by this function with
         *   a UUID uniquely identifying this prototype.
         * returns:
         *  Pointer to array of Data Set prototype contents structures
         *   DNPDATA_DATASET_DESCR_ELEM.  This points to memory maintained
         *   by the database. It can be in ROM or in RAM. This pointer will need
         *   to be valid until mdnpdata_datasetProtoRelease() is called.
1350     *   NOTE: The array of elements should not contain the mandatory Prototype ID
         *   and UUID. Prototype ID is specified as a parameter and UUID will be
         *   returned by pUUID parameter.
         *   If a Namespace and Name element are present in the prototype they
         *   must be the first and second element in this array.  If either Namespace
         *   or Name is present the other must also be present.
         *  TMWDEFS_NULL if failed to get pointer to array.
         */
        DNPDATA_DATASET_DESCR_ELEM * TMWDEFS_GLOBAL mdnpdata_datasetProtoGet(
          void *pHandle,
1360      TMWTYPES_USHORT pointNumber,
```

```
  TMWTYPES_UCHAR *pNumberElems,
  TMWTYPES_UCHAR *pUUID);

/* function: mdnpdata_datasetProtoRelease
 * purpose: Release the pointer that was returned by in mdnpdata_datasetProtoGet
 *  The database is free to deallocate or reuse the memory that was pointed to.
 *  The SCL will not attempt to reference that pointer anymore.
 *  pHandle - database handle returned from mdnpdata_init
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL mdnpdata_datasetProtoRelease(
   void *pHandle);

/* function: mdnpdata_datasetProtoGetID
 * purpose: Get a prototype ID (point index) for a Data Set prototype
 *  with this UUID, if one exists in the database.
 * arguments:
 *  pHandle - handle to database returned from mdnpdata_init
 *  pUUID - pointer to 16 byte UUID string to be looked up
 *  pPointNumber - point number or prototype id to be filled in.
 * returns:
 *  TMWDEFS_TRUE if prototype was found
 *  TMWDEFS_FALSE otherwise
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_datasetProtoGetID(
  void *pHandle,
  TMWTYPES_UCHAR  *pUUID,
  TMWTYPES_USHORT *pPointNumber);

/* function: mdnpdata_datasetDescrQuantity
 * purpose: Get quantity of Data Set Descriptors in the database.
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 * returns:
 *  Quantity of Descriptors in database
 */
TMWTYPES_USHORT mdnpdata_datasetDescrQuantity(
  void *pHandle);

/* function: mdnpdata_datasetDescrSlaveQty
 * purpose: Get quantity of Data Set Descriptors in the database that
 *  were read from the slave. Descriptor ids must start with 0 and be
 *  contiguous, however the descriptors defined by the outstation or slave
 *  are numbered 0 to n-1. The ones defined by the master should be numbered
 *  n to x-1. Where n is the number of descriptors defined by the outstation,
 *  and x is the number of descriptors defined by the master and outstation
 *  combined.
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 * returns:
 *  Quantity of descriptors read from the slave
 *  NOTE: this would be the number of different descriptors written to the
 *   database by mdnpdata_storeDatasetDescrCont()
 */
TMWTYPES_USHORT mdnpdata_datasetDescrSlaveQty(
  void *pHandle);

/* function: mdnpdata_storeDatasetDescrCont
 * purpose: Store a Data Set descriptor contents structure to support
 *   read of object group 86 variation 1
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  pointNumber - point number of point to store
 *  elemIndex - index of element in Data Set descriptor to be stored starting
 *   with index 0. The mandatory Data Set descriptor ID will not be stored as
 *   a descriptor element. It will be passed in the pointNumber parameter.
 *  pDescr - pointer to structure containing data to be written
 *   NOTE: if pDescr->ancillaryValue->type == DNPDATA_VALUE_STRPTR, this is just
 *    a pointer, the string itself must be copied somewhere.
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL mdnpdata_storeDatasetDescrCont(
  void *pHandle,
  TMWTYPES_USHORT pointNumber,
  TMWTYPES_UCHAR  elemIndex,
  DNPDATA_DATASET_DESCR_ELEM *pDescr);

/* function: mdnpdata_storeDatasetDescrChars
 * purpose: Store a Data Set descriptor characteristics from a
```

```
 *   read of object group 86 variation 2
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  pointNumber - point number of Data Set descriptor to store
 *  value - characteristics value
 *   DNPDEFS_DATASET_CHAR_RD
 *   DNPDEFS_DATASET_CHAR_WR
 *   DNPDEFS_DATASET_CHAR_ST
 *   DNPDEFS_DATASET_CHAR_EV
 *   DNPDEFS_DATASET_CHAR_DF
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL mdnpdata_storeDatasetDescrChars(
  void *pHandle,
  TMWTYPES_USHORT pointNumber,
  TMWTYPES_UCHAR  value);

/* function: mdnpdata_storeDatasetDescrIndex
 * purpose: Store a Data Set descriptor index structure to support
 *   read of object group 86 variation 3
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  pointNumber - point number of point to store
 *  elemIndex - index of element in Data Set descriptor index to be stored
 *   starting with index 0. There will be one of these for each data and
 *   control value element in the descriptor and in any prototypes contained
 *   by it.
 *  pDescr - pointer to structure containing data to be written
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL mdnpdata_storeDatasetDescrIndex(
  void *pHandle,
  TMWTYPES_USHORT pointNumber,
  TMWTYPES_UCHAR  elemIndex,
  DNPDATA_DATASET_DESCR_INDEX *pDescr);

/* function: mdnpdata_datasetDescrGetCont
 * purpose: Get a pointer to array of Data Set descriptor contents
 *   structures to support write of object group 86 variation 1
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  pointNumber - descriptor id or point number of descriptor to get
 *  pNumberElems - pointer to number of elements returned in specified descriptor
 *   array.
 * returns:
 *  Pointer to array of data set descriptor contents structures
 *   DNPDATA_DATASET_DESCR_ELEM. This points to memory maintained
 *   by the database. It can be in ROM or in RAM. This pointer will need
 *   to be valid until mdnpdata_datasetDescrRelease() is called.
 *  NOTE: Data Set id is a mandatory element and is specified as a parameter of
 *  this function. It is not returned as part of the DNPDATA_DATASET_DESCR_ELEM
 *  array. Descriptor name is an optional element, but if present, must be the
 *  first element in the array returned.
 *  TMWDEFS_NULL if failed to get pointer to array.
 */
DNPDATA_DATASET_DESCR_ELEM * TMWDEFS_GLOBAL mdnpdata_datasetDescrGetCont(
  void *pHandle,
  TMWTYPES_USHORT pointNumber,
  TMWTYPES_UCHAR *pNumberElems);

/* The protocol does not support a write of object group 86 variation 2
 * characteristics, so there is no function to get the data from the database.
 */

/* function: mdnpdata_datasetDescrGetIndex
 * purpose: Get a pointer to array of Data Set descriptor index
 *   structures to support write of object group 86 variation 3
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  pointNumber - descriptor id or point number of descriptor to get
 *  pNumberElems - number of elements returned in specified Data Set array
 * returns:
 *  Pointer to array of Data Set descriptor index structures
 *   DNPDATA_DATASET_DESCR_INDEX. This points to memory maintained
 *   by the database. It can be in ROM or in RAM. This pointer will need
 *   to be valid until mdnpdata_datasetDescrRelease() is called.
 *   NOTE: there should be one element of this array for each data and control
 *    value element in the Data Set descriptor (including data and control
 *    value elements in the contained prototypes).
```

```
     *  TMWDEFS_NULL if failed to get pointer to array.
     */
    DNPDATA_DATASET_DESCR_INDEX * TMWDEFS_GLOBAL mdnpdata_datasetDescrGetIndex(
      void *pHandle,
      TMWTYPES_USHORT pointNumber,
      TMWTYPES_UCHAR *pNumberElems);

    /* function: mdnpdata_datasetDescrRelease
     * purpose: Release the pointer that was returned by mdnpdata_datasetDescrGetCont
     *  or mdnpdata_datasetDescrGetIndex
     *  The database is free to deallocate or reuse the memory that was pointed to.
    The
     *  SCL will not attempt to reference that pointer anymore.
     * arguments:
     *  pHandle - database handle returned from mdnpdata_init
     * returns:
     *  void
     */
    void TMWDEFS_GLOBAL mdnpdata_datasetDescrRelease(
       void *pHandle);

    /* function: mdnpdata_storeDatasetTime
     * purpose: Store a Data Set timestamp received in either Object Group 87
     *  Data Set present value, or Object Group 88 Data Set event response
     * arguments:
     *  pHandle - database handle returned from mdnpdata_init
     *  pointNumber - point number (Data Set ID) of Data Set to store
     *  pTimeStamp - pointer to timestamp received
     *  isEvent - TMWDEFS_TRUE if this update was caused by a change event
     * returns:
     *  void
     */
    void TMWDEFS_GLOBAL mdnpdata_storeDatasetTime(
      void *pHandle,
      TMWTYPES_USHORT pointNumber,
      TMWDTIME *pTimeStamp,
      TMWTYPES_BOOL isEvent);

    /* function: mdnpdata_storeDataset
     * purpose: Store Data Set value received in either Object Group 87
     *  Data Set present value, or Object Group 88 Data Set event response
     * arguments:
     *  pHandle - database handle returned from mdnpdata_init
     *  pointNumber - point number (datataset ID) of Data Set to store
     *  elemIndex - index of Data Set element to be stored starting
     *   with index 0 indicating the first element after the mandatory
     *   Data Set ID and timeStamp.
     *  pElem - pointer to structure containing data to be written
     *   NOTE: if pElem->type == DNPDATA_VALUE_STRPTR, this is just
     *    a pointer, the string itself must be copied somewhere.
     *  isEvent - TMWDEFS_TRUE if this update was caused by a change event
     * returns:
     *  void
     */
    void TMWDEFS_GLOBAL mdnpdata_storeDataset(
      void *pHandle,
      TMWTYPES_USHORT pointNumber,
      TMWTYPES_UCHAR  elemIndex,
      DNPDATA_DATASET_VALUE *pElem,
      TMWTYPES_BOOL isEvent);

    /* function: mdnpdata_datasetGet
     * purpose: Get a pointer to array of Data Set value
     *   structures to support write of object group 87 variation 1
     * arguments:
     *  pHandle - database handle returned from mdnpdata_init
     *  pointNumber - point number (datataset ID) of Data Set to get
     *  pNumberElems - number of elements returned in specified Data Set array
     *  pTimeStamp - pointer to mandatory timestamp
     * returns:
     *  Pointer to array of Data Set value structures
     *   DNPDATA_DATASET_VALUE. This points to memory maintained
     *   by the database. This pointer will need to be valid until
     *   mdnpdata_datasetDescrRelease() is called.
     *   NOTE: The array of elements should not contain the mandatory Data Set ID
     *   and timeStamp.
     *  TMWDEFS_NULL if failed to get pointer to array.
     */
    DNPDATA_DATASET_VALUE * TMWDEFS_GLOBAL mdnpdata_datasetGet(
      void *pHandle,
      TMWTYPES_USHORT pointNumber,
```

```
    TMWTYPES_UCHAR *pNumberElems,
    TMWDTIME *pTimeStamp);

  /* function: mdnpdata_datasetRelease
   * purpose: Release the pointer that was returned by mdnpdata_datasetGet
   *  The database is free to deallocate or reuse the memory that was pointed to.
   *  The SCL will not attempt to reference that pointer anymore.
   * arguments:
   *  pHandle - database handle returned from mdnpdata_init
   * returns:
   *  void
   */
  void TMWDEFS_GLOBAL mdnpdata_datasetRelease(
     void *pHandle);

  /* function: mdnpdata_authIsCriticalReq
   * purpose: Determine if this message should be considered critical.
   *  NOTE: The Specification lists requests that are required to be critical.
   *   Although allowed, no responses are required by the spec to be critical.
   *   It is acceptable and normal to always return TMWDEFS_FALSE from this function
   *   Only consider a response critical if it is necessary that you do so.
   * arguments:
   *  pHandle - database handle returned from mdnpdata_init
   *  pRxMsg - pointer to received message
   *  msgLength = length of received message
   *  pUserNumber - user number to be filled in if this is a critical message
   *   Authentication Spec says to use Default User Number (1) for challenging
   *   any response from outstation.
   * returns:
   *  TMWDEFS_TRUE if this is a critical message
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL mdnpdata_authIsCriticalReq(
    void            *pHandle,
    TMWTYPES_UCHAR  *pRxMsg,
    TMWTYPES_USHORT  msgLength,
    TMWTYPES_USHORT *pUserNumber);

#if MDNPCNFG_SUPPORT_SA_VERSION5

#if DNPCNFG_SUPPORT_AUTHKEYUPDATE
  /* function: mdnpdata_authGetOSName
   * purpose:  Get the name of the outstation this master is connected to.
   *  Spec says this must be preconfigured on both the master and outstation.
   *  This is used when Secure Authentication Version 5 or later Update Key Change
   *  symmetric and asymmetric methods are used.
   * arguments:
   *  pHandle - handle to database returned from mdnpdata_init
   *  pOSName - pointer to location where Outstation name should be copied
   *  pOSNameLength - when called this is the maximum length allowed for
   *    the name, on return this should be set to the length of the name.
   * returns:
   *  TMWDEFS_TRUE if successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_authGetOSName(
    void            *pHandle,
    TMWTYPES_UCHAR  *pOSName,
    TMWTYPES_USHORT *pOSNameLength);

  /* function: mdnpdata_authGetUserName
   * purpose:  Get globally unique user name provided by Authority from the database
   * arguments:
   *  pHandle - handle to database returned from mdnpdata_init
   *  userNameDbHandle - handle for looking up the data for a particular user name.
   *    This handle is meaningful to the database and is intended to be more
   *    convenient than the full user name. This handle was determined when the
   *    user name and other data was added to the database. The actual User
   *    Number will not be known on the master when remote Update Key
   *    distribution is used until the outstation assigns one and informs
   *    the master.
   *  pUserName - pointer to location where user name should be copied
   *  pUserNameLength - when called this is the maximum length allowed for
   *    the name, on return this should be set to the length of the name.
   * returns:
   *  TMWDEFS_TRUE if successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_authGetUserName(
    void            *pHandle,
    void            *userNameDbHandle,
```

```
  TMWTYPES_UCHAR  *pUserName,
  TMWTYPES_USHORT *pUserNameLength);

/* This is called when an x.509 user certificate is used */
/* function: mdnpdata_authGetKeyChangeMethod
 * purpose:  Determine what key change method to use. The specification says
 *  this shall be pre-configured at the master and only one method shall be active
 *  between a particular master and outstation. Therefore you may want to always
 *  return the same method and not allow different ones per request.
 * arguments:
 *  pHandle - handle to database returned from mdnpdata_init
 *  userNameDbHandle - handle for looking up the data for a particular user name.
 *    This handle is meaningful to the database and is intended to be more
 *    convenient than the full user name. This handle was determined when the
 *    user name and other data was added to the database. The actual User
 *    Number will not be known on the master when remote Update Key
 *    distribution is used until the outstation assigns one and informs
 *    the master.
 *  pKeyChangeMethod - pointer to key change method to be filled in
 * returns:
 *  TMWDEFS_TRUE if successful
 *  TMWDEFS_FALSE otherwise
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_authGetKeyChangeMethod(
  void            *pHandle,
  void            *userNameDbHandle,
  TMWTYPES_UCHAR  *pKeyChangeMethod);

/* function: mdnpdata_authGetChangeUserData
 * purpose:  Get g120v10 data to send to the outstation. ALL of this data was
 *  provided by the Authority and stored before this key change procedure was
 *  initiated.
 * arguments:
 *  pHandle - handle to database returned from mdnpdata_init
 *  userNameDbHandle - handle for looking up the data for a particular user name.
 *    This handle is meaningful to the database and is intended to be more
 *    convenient than the full user name. This handle was determined when the
 *    user name and other data was added to the database. The actual User
 *    Number will not be known on the master when remote Update Key
 *    distribution is used until the outstation assigns one and informs
 *    the master.
 *  pStatusChangeSequenceNumber - to be filled in
 *  pKeyChangeMethod - pointer to key change method to be filled in. The
specification says
 *    this shall be pre-configured at the master and only one method shall be
active
 *    between a particular master and outstation. Therefore you may want to always
 *    return the same method and not allow different ones per request.
 *  pUserRole - to be filled in.
 *   DNPAUTH_USER_ROLE_VIEWER
 *   DNPAUTH_USER_ROLE_OPERATOR
 *   DNPAUTH_USER_ROLE_ENGINEER
 *   DNPAUTH_USER_ROLE_INSTALLER
 *   DNPAUTH_USER_ROLE_SECADM
 *   DNPAUTH_USER_ROLE_SECAUD
 *   DNPAUTH_USER_ROLE_RBACMNT
 *   DNPAUTH_USER_ROLE_SINGLEUSER
 *  pUserExpiryInterval - to be filled in. Indicates number of days after
receiving the
 *    User Status Change object that the outstation shall consider the user role
to be expired
 * returns:
 *  TMWDEFS_TRUE if successful
 *  TMWDEFS_FALSE otherwise
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_authGetChangeUserData(
  void            *pHandle,
  void            *userNameDbHandle,
  TMWTYPES_ULONG  *pStatusChangeSequenceNumber,
  TMWTYPES_UCHAR  *pKeyChangeMethod,
  TMWTYPES_USHORT *pUserRole,
  TMWTYPES_USHORT *pUserRoleExpiryInterval);

/* function: mdnpdata_authGetCertData
 * purpose:  Get the certification data for the user specified by this userHandle
 *  to be sent in this g120v10 User Status Change message. This certification data
 *  will be provided by the AUTHORITY to certify that the other data in the
g120v10
 *   message is correct.
 *  If symmetric key method, the pre-shared (between AUTHORITY and outstation)
 *    Authority Certification Key will be used by the AUTHORITY to certify this
```

```
data.
1760     *  If asymmetric key method, the AUTHORITY will use the Authority Private Key to
sign
         *    this data.  Outstation will need to be configured with the Authority Public
Key.
         * arguments:
         *  pHandle - handle to database returned from mdnpdata_init
         *  userNameDbHandle - handle for looking up the data for a particular user name.
         *    This handle is meaningful to the database and is intended to be more
         *    convenient than the full user name. This handle was determined when the
         *    user name and other data was added to the database. The actual User
         *    Number will not be known on the master when remote Update Key
         *    distribution is used until the outstation assigns one and informs
1770     *    the master.
         * pCertData - pointer to buffer to be filled in with certification data.
         *   NOTE: This certification data will contain an operation,
         *   Status Change Sequence (SCS) number between the Authority and Outstation,
         *   Operation, User Role, Role Expiry Interval, A globally unique name
representing the user,
         *   and if asymmetric a Users public key.
         * pCertDataLength - pointer to the maximum certification length allowed and
should be
         *  filled in on return with the actual length of the certification data.
         */
        TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_authGetCertData(
1780      void            *pHandle,
          void            *userNameDbHandle,
          TMWTYPES_UCHAR  *pCertData,
          TMWTYPES_USHORT *pCertDataLength);

        /* function: mdnpdata_authStoreUpdKeyChangeReply
         * purpose:  Store the user number from the g120v12 Update Key Change Reply
message
         *   from the outstation.
         * arguments:
         *  pHandle - handle to database returned from mdnpdata_init
1790     *  userNameDbHandle - handle for looking up the data for a particular user name.
         *    This handle is meaningful to the database and is intended to be more
         *    convenient than the full user name. This handle was determined when the
         *    user name and other data was added to the database. The actual User
         *    Number will not be known on the master when remote Update Key
         *    distribution is used until the outstation assigns one and informs
         *    the master.
         *  userNumber - user number that has been selected by the outstation
         *   to be used for this user and it's update key.
         */
1800    void TMWDEFS_GLOBAL mdnpdata_authStoreUpdKeyChangeReply(
          void            *pHandle,
          void            *userNameDbHandle,
          TMWTYPES_USHORT  userNumber,
          TMWTYPES_UCHAR  *pOSData,
          TMWTYPES_USHORT  OSDataLength);

        /* function: mdnpdata_authGetUpdateKeyData
         * purpose:  Get the encrypted data generated by the Authority to be sent to the
outstation in a g120v13 message

1810     * arguments:
         *  pHandle - handle to database returned from mdnpdata_init
         *  userNameDbHandle - handle for looking up the data for a particular user name.
         *    This handle is meaningful to the database and is intended to be more
         *    convenient than the full user name. This handle was determined when the
         *    user name and other data was added to the database. The actual User
         *    Number will not be known on the master when remote Update Key
         *    distribution is used until the outstation assigns one and informs
         *    the master.
         */
1820    TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_authGetUpdateKeyData(
          void            *pHandle,
          void            *userNameDbHandle,
          TMWTYPES_UCHAR  *pOSData,
          TMWTYPES_USHORT *pOSDataLength);

        /* function: mdnpdata_authDeleteUser
         * purpose:  Delete this auth user from the database. A command has been sent to
         *  the outstation to delete this user and a response indicating success has been
         *  received back from the outstation.
1830     * arguments:
         *  pHandle - handle to database returned from mdnpdata_init
         *  userNameDbHandle - handle for looking up the data for a particular user name.
         *    This handle is meaningful to the database and is intended to be more
```

```
     *    convenient than the full user name. This handle was determined when the
     *    user name and other data was added to the database. The actual User
     *    Number will not be known on the master when remote Update Key
     *    distribution is used until the outstation assigns one and informs
     *    the master.
     *  userNumber - user number for this handle if one is known.
     *               zero otherwise.
     */
    TMWTYPES_USHORT TMWDEFS_GLOBAL mdnpdata_authDeleteUser(
      void            *pHandle,
      void            *userNameDbHandle);
  #endif

    /* function: mdnpdata_authStoreSecStat
     * purpose: Store master generated statistic value
     * arguments:
     *  pHandle - handle to database returned from mdnpdata_init
     *  index - index indicating which statistic
     *   for example DNPAUTH_UNEXPECTED_MSG_INDEX
     *  value - value of statistic
     */
    void TMWDEFS_GLOBAL mdnpdata_authStoreSecStat(
      void *pHandle,
      TMWTYPES_USHORT index,
      TMWTYPES_ULONG value);

    /* function: mdnpdata_authStoreOSSecStat
     * purpose:  Store outstation generated statistic value that was
     *  received from the outstation.
     * arguments:
     *  pHandle - handle to database returned from mdnpdata_init
     *  pointNumber - point Number recveived. Can be converted to index
     *   indicating which statistic, depending on which association this
     *   was received on.
     *  value - value of statistic
     *  flags - DNP3 flags to store. Flag value is OR'd
     *    combinations of the following:
     *      DNPDEFS_DBAS_FLAG_ON_LINE - the binary input point has been read
     *        successfully
     *      DNPDEFS_DBAS_FLAG_RESTART - the field device that originated the
     *        data object has been restarted. This device may be the deviced
     *        reporting this data object.
     *      DNPDEFS_DBAS_FLAG_COMM_LOST - the device reporting this data object
     *        has lost communication with the originator of the data object
     *      DNPDEFS_DBAS_FLAG_REMOTE_FORCED - the state of the binary object
     *        has been forced to its current state at the originating device
     *      DNPDEFS_DBAS_FLAG_LOCAL_FORCED - the state of the binary object
     *        has been forced to its current state at the device reporting
     *        this data object
     *      DNPDEFS_DBAS_FLAG_CNTR_ROLLOVER - the accumulated value has exceeded
     *        has exceeded its maximum and rolled over to zero.
     *      DNPDEFS_DBAS_FLAG_DISCONTINUITY - value cannot be compared against
     *        a prior value to obtain the correct count difference
     *  isEvent - TMWDEFS_TRUE if this update was caused by a change event
     *  pTimeStamp - pointer to time stamp reported with this change event
     *   or TMWDEFS_NULL if no time was reported. Will always be TMWDEFS_NULL
     *   if isEvent is TMWDEFS_FALSE.
     */
    TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_authStoreOSSecStat(
      void *pHandle,
      TMWTYPES_USHORT association,
      TMWTYPES_USHORT pointNumber,
      TMWTYPES_ULONG value,
      TMWTYPES_UCHAR flags,
      TMWTYPES_BOOL isEvent,
      TMWDTIME *pTimeStamp);

    /* function: mdnpdata_authLogMaxRekeyTCPClose
     * purpose:  log that Rekeys Due to Authentication Failure statistic
     *  has exceeded Max Authentication Rekeys and since this is TCP
     *  the connection will be closed.
     * arguments:
     *  pHandle - database handle returned from mdnpdata_init
     * returns:
     */
    void TMWDEFS_GLOBAL mdnpdata_authLogMaxRekeyTCPClose(
      void            *pHandle);

    /* function: mdnpdata_authLogUnexpectedMsg
     * purpose:  log that an unexpected message was received as indicated
     *  in SA Spec V4 Table 14
```

```
     * arguments:
     *  pHandle - database handle returned from mdnpdata_init
     *  state - current master state ie MDNPAUTH_STATE_XXX
     *  event - event being processes ie MDNPAUTH_EVT_XX
     *  pRxFragment - pointer to message that was received.
1920 * returns:
     */
    void TMWDEFS_GLOBAL mdnpdata_authLogUnexpectedMsg(
      void            *pHandle,
      TMWTYPES_UCHAR   state,
      TMWTYPES_ULONG   event,
      TMWSESN_RX_DATA *pRxFragment);

    /* function: mdnpdata_authLogFailedUpdateKey
     * purpose:  log that a Update Key Change failed as indicated in SA Spec V4
1930 *  Table 14
     * arguments:
     *  pHandle - database handle returned from mdnpdata_init
     *  state - current master state ie MDNPAUTH_STATE_XXX
     *  event - event being processes ie MDNPAUTH_EVT_XX
     * returns:
     */
    void TMWDEFS_GLOBAL mdnpdata_authLogFailedUpdateKey(
      void            *pHandle,
      TMWTYPES_UCHAR   state,
1940  TMWTYPES_ULONG   event);

  #endif

    /* function: mdnpdata_authLogTx
     * purpose:  log the Secure Authentication message that is sent to
     *  Outstation.
     *  mdnpdata_authLogErrorTx will be called for error messages (var 7)
     * arguments:
     *  pHandle - database handle returned from mdnpdata_init
1950 *  variation - variation of message sent (object group 120).
     *  userNumber - user number transmitted if applicable, 0 otherwise.
     *  sequenceNumber - sequence number sent if applicable, 0 otherwise.
     *  pMsgBuf - pointer to message being sent.
     *  msgLength - length of message being sent.
     * returns:
     *  void
     */
    void TMWDEFS_GLOBAL mdnpdata_authLogTx(
      void            *pHandle,
1960  TMWTYPES_UCHAR   variation,
      TMWTYPES_USHORT  userNumber,
      TMWTYPES_ULONG   sequenceNumber,
      TMWTYPES_UCHAR  *pMsgBuf,
      TMWTYPES_USHORT  msgLength);

    /* function: mdnpdata_authLogRx
     * purpose:  log the Secure Authentication message that was received
     *  from Outstation
     *  mdnpdata_authLogErrorRx will be called for error messages (var 7)
1970 * arguments:
     *  pHandle - database handle returned from mdnpdata_init
     *  variation - variation of message received (object group 120).
     *  userNumber - user number received
     *  sequenceNumber - sequence number received
     *  pMsgBuf - pointer to message received.
     *  msgLength - length of message received.
     * returns:
     *  void
     */
1980 void TMWDEFS_GLOBAL mdnpdata_authLogRx(
      void            *pHandle,
      TMWTYPES_UCHAR   variation,
      TMWTYPES_USHORT  userNumber,
      TMWTYPES_ULONG   sequenceNumber,
      TMWTYPES_UCHAR  *pMsgBuf,
      TMWTYPES_USHORT  msgLength);

    /* function: mdnpdata_authLogErrorTx
     * purpose:  log an error condition occurred for a particular user number and
1990 *  whether an error messsage is being sent to the outstation. It will not be
     *  sent if the max error count has been exceeded.
     *  mdnpdata_authLogTx will not be called for error messages.
     * arguments:
     *  pHandle - database handle returned from mdnpdata_init
     *  userNumber - user number transmitted
```

```
         *  assocId - Association Id identifying which master/outstation association
         *  sequenceNumber - challenge sequence number transmitted
         *  errorCode - error code from error message
         *  pTimeStamp - pointer to time stamp transmitted
2000     *  pMsgBuf - pointer to error message being sent
         *  msgLength - length of msg
         *  msgSent - indicates if an error message was sent to the outstation. When count
         *   exceeds max, error message will not be sent.
         * returns:
         *  void
         */
        void TMWDEFS_GLOBAL mdnpdata_authLogErrorTx(
          void           *pHandle,
          TMWTYPES_USHORT userNumber,
2010      TMWTYPES_USHORT assocId,
          TMWTYPES_ULONG  sequenceNumber,
          TMWTYPES_UCHAR  errorCode,
          TMWDTIME       *pTimeStamp,
          TMWTYPES_CHAR  *pMsgBuf,
          TMWTYPES_USHORT msgLength,
          TMWTYPES_BOOL   msgSent);

        /* function: mdnpdata_authLogErrorRx
         * purpose:  store g120v7 error data received from an outstation
2020     *  mdnpdata_authLogRx will not be called for error messages.
         * arguments:
         *  pHandle - handle to database returned from mdnpdata_init
         *  userNumber - user number from message
         *  assocId - association id indicating which master/outstation association
         *  sequenceNumber - Sequence number from error message
         *  errorCode - error identifier
         *  pTimeStamp - time stamp from message
         *  pMsgBuf - pointer to error message received
         *  msgLength - length of message being received
2030     * returns:
         */
        void TMWDEFS_GLOBAL mdnpdata_authLogErrorRx(
          void            *pHandle,
          TMWTYPES_USHORT  userNumber,
          TMWTYPES_USHORT  assocId,
          TMWTYPES_ULONG   sequenceNumber,
          TMWTYPES_UCHAR   errorCode,
          TMWDTIME        *pTimeStamp,
          TMWTYPES_UCHAR  *pMsgBuf,
2040      TMWTYPES_USHORT  msgLength);

        /* function: mdnpdata_authEvent
         * purpose: Indicates to database that a specific Secure Authentication
         *  event has occurred.
         * arguments:
         *  pHandle - handle to database returned from mdnpdata_init
         *  userNumber - user number that was involved if appropriate,
         *               0 if not user specific.
         *  event - event that occurred.
2050     *          MDNPAUTH_EV_V4_BAD_FUNCTION-master received IIN2-0 Function Code Not
         *            Implemented in response to Read Status g120v4 indicating
         *            outstation does not support Secure Authentication.
         *          MDNPAUTH_EV_V10_BAD_FUNCTION-master received IIN 2-2 Function Code Not
         *            Implemented in response to change user status g120v10 indicating
         *            outstation does not support Secure Authentication Remote Key Update.
         *
         * returns:
         *  void
         */
2060    void TMWDEFS_GLOBAL mdnpdata_authEvent(
          void                 *pHandle,
          TMWTYPES_USHORT       userNumber,
          MDNPAUTH_EVENT_ENUM   event);

    #if MDNPCNFG_SUPPORT_SA_VERSION2
        /* The following functions only need to be implemented for DNP3 Secure
Authentication Version 2
         * These function will NOT be called if ONLY SA V5 is supported
         * If support for BOTH Version 2 and Version 5 is required, the following
functions WILL be called.
         * See the SCL User Manual for details of how these relate to the Version 5
functions in tmwcrypto.h/c
2070     */

        /* function: mdnpdata_authLogKeyStatRqTx
         * purpose:  log that a key status request is being sent to the outstation
```

```
 *    NOTE: Only required for SA_VERSION2
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  userNumber - user number transmitted
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL mdnpdata_authLogKeyStatRqTx(
  void            *pHandle,
  TMWTYPES_USHORT  userNumber);

/* function: mdnpdata_authLogKeyChange
 * purpose:  log that a key change request is being sent to the outstation
 *    NOTE: Only required for SA_VERSION2
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  userNumber - user number transmitted
 *  sequenceNumber - key change sequence number transmitted
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL mdnpdata_authLogKeyChangeTx(
  void            *pHandle,
  TMWTYPES_USHORT  userNumber,
  TMWTYPES_ULONG   sequenceNumber);

/* function: mdnpdata_authLogAggrTx
 * purpose:  log that a aggressive mode request is being sent to the outstation
 *    NOTE: Only required for SA_VERSION2
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  userNumber - user number transmitted
 *  sequenceNumber - challenge sequence number transmitted
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL mdnpdata_authLogAggrTx(
  void            *pHandle,
  TMWTYPES_USHORT  userNumber,
  TMWTYPES_ULONG   sequenceNumber);

/* function: mdnpdata_authLogChallRplyTx
 * purpose:  log that a challenge reply is being sent to the outstation
 *    NOTE: Only required for SA_VERSION2
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  userNumber - user number transmitted
 *  sequenceNumber - challenge sequence number transmitted
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL mdnpdata_authLogChallRplyTx(
  void            *pHandle,
  TMWTYPES_USHORT  userNumber,
  TMWTYPES_ULONG   sequenceNumber);

/* function: mdnpdata_authLogKeyStatusRx
 * purpose:  log that a key status request has been received from an outstation
 *    NOTE: Only required for SA_VERSION2
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  userNumber - user number from message
 *  sequenceNumber -  Sequence number from message
 *  keyWrapAlgorithm - Key Wrap Algorithm from message
 *  keyStatus - Key Status from message
 *  macAlgorithm - MAC Algorithm number from message
 * returns:
 */
void TMWDEFS_GLOBAL mdnpdata_authLogKeyStatusRx(
  void            *pHandle,
  TMWTYPES_USHORT  userNumber,
  TMWTYPES_ULONG   sequenceNumber,
  TMWTYPES_UCHAR   keyWrapAlgorithm,
  TMWTYPES_UCHAR   keyStatus,
  TMWTYPES_UCHAR   macAlgorithm);

 /* function: mdnpdata_authLogChallRx
 * purpose:  log that a challenge request has been received from an outstation
 *    NOTE: Only required for SA_VERSION2
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
```

```
 *  userNumber - user number from message
 *  sequenceNumber -  Challenge Sequence Number from message
 *  macAlgorithm - MAC Algorithm number from message
 *  reasonForChallenge - reason for challenge. Authentication Spec currently
 *   only defines CRITICAL==1.
 * returns:
 */
void TMWDEFS_GLOBAL mdnpdata_authLogChallRx(
  void            *pHandle,
  TMWTYPES_USHORT  userNumber,
  TMWTYPES_ULONG   sequenceNumber,
  TMWTYPES_UCHAR   macAlgorithm,
  TMWTYPES_UCHAR   reasonForChallenge);

/* function: mdnpdata_authLogChallRplyRx
 * purpose:  log that a challenge reply has been received from an outstation
 *    NOTE: Only required for SA_VERSION2
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  userNumber - user number from message
 *  sequenceNumber -  Challenge Sequence Number from message
 *  status - TMWDEFS_TRUE if this message was authenticated properly
 *   TMWDEFS_FALSE otherwise
 * returns:
 */
void TMWDEFS_GLOBAL mdnpdata_authLogChallRplyRx(
  void            *pHandle,
  TMWTYPES_USHORT  userNumber,
  TMWTYPES_ULONG   sequenceNumber,
  TMWTYPES_BOOL    status);

/* function: mdnpdata_authLogAggrRx
 * purpose:  log that an aggressive mode response has been received from an
 *    NOTE: Only required for SA_VERSION2
 *  outstation
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  userNumber - user number from message
 *  sequenceNumber -  Challenge Sequence Number from message
 *  status - TMWDEFS_TRUE if this message was authenticated properly
 *   TMWDEFS_FALSE otherwise
 * returns:
 */
void TMWDEFS_GLOBAL mdnpdata_authLogAggrRx(
  void            *pHandle,
  TMWTYPES_USHORT  userNumber,
  TMWTYPES_ULONG   sequenceNumber,
  TMWTYPES_BOOL    status);

/* function: mdnpdata_authStoreErrorRx
 * purpose:  store g120v7 error data received from an outstation
 *    NOTE: Only required for SA_VERSION2
 * arguments:
 *  pHandle - handle to database returned from mdnpdata_init
 *  userNumber - user number from message
 *  assocId - association id indicating which master/outstation association
 *  sequenceNumber - Sequence number from error message
 *  errorCode - error identifier
 *  pTimeStamp - time stamp from message
 *  pErrorText - pointer to optional error text
 *  errorTextLength - length of optional error text.
 * returns:
 */
void TMWDEFS_GLOBAL mdnpdata_authStoreErrorRx(
  void            *pHandle,
  TMWTYPES_USHORT  userNumber,
  TMWTYPES_USHORT  assocId,
  TMWTYPES_ULONG   sequenceNumber,
  TMWTYPES_UCHAR   errorCode,
  TMWDTIME        *pTimeStamp,
  TMWTYPES_CHAR   *pErrorText,
  TMWTYPES_USHORT  errorTextLength);

/* function: mdnpdata_authGetNewSessionKeys
 * purpose:  This function should create new session keys to be used for
 *  authentication. These keys are sent to the outstation in a key change
 *  request. These keys will be changed frequently, every 15 minutes or
 *  so by default. These keys should random and generated, not configured.
 *    NOTE: Only required for SA_VERSION2, see utils/tmwcrypto.h for SA_VERSION5
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
```

```
 *  pControlSessionKey - pointer to control direction session key structure
 *    to be filled in
 *  pMonitorSessionKey - pointer to monitor direction session key structure
 *    to be filled in
 * returns:
 *  TMWDEFS_TRUE if successful
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL mdnpdata_authGetNewSessionKeys(
  void             *pHandle,
  DNPDATA_AUTH_KEY *pControlSessionKey,
  DNPDATA_AUTH_KEY *pMonitorSessionKey);

/* function: mdnpdata_authKeyWrapSupport
 * purpose:  This function should determine whether the key wrap algorithm
 *   requested by the outstation is supported
 *    NOTE: Only required for SA_VERSION2, see utils/tmwcrypto.h for SA_VERSION5
 * arguments:
 *  keyWrapAlgorithm - key wrap algorithm
 *   DNPAUTH_KEYWRAP_AES128 is the only one currently specified.
 * returns:
 *  TMWDEFS_TRUE if supported
 *  TMWDEFS_FALSE otherwise
 */
TMWTYPES_BOOL mdnpdata_authKeyWrapSupport(
  TMWTYPES_UCHAR  keyWrapAlgorithm);

/* function: mdnpdata_authEncryptKeyWrapData
 * purpose:  Encrypt the key data using the update key for the indicated
 *   session database. Data should be padded as required by encryption
 *   algorithm. Return the encrypted data in pValue setting *pLength to
 *   the length of the returned data
 *   Care should be take to protect the security of the Update Key.
 *   Encrypting it and/or limiting who can get or set this key is very
 *   important.
 *    NOTE: Only required for SA_VERSION2, see utils/tmwcrypto.h for SA_VERSION5
 * arguments:
 *  pHandle - database handle returned from mdnpdata_init
 *  userNumber - user number
 *  algorithm - Encryption algorithm to use
 *   DNPAUTH_KEYWRAP_AES128 is the only one currently specified.
 *   other values reserved for future use or vendor specific choices
 *   NOTE: The AES Key Wrap Algorithm specified in RFC3394 is not the same
 *    thing as AES encryption. The key wrap algorithm actually will call
 *    the AES encryption function multiple times depending on the length
 *    the key data to be encrypted.
 *  pPlainValue - pointer to data to be encrypted
 *  plainDataLength - length of data to be encrypted, this will NOT be padded
 *   out to 8 bytes. The AES128 key wrap algorithm expects multiples of 8 bytes.
 *    It does not matter what byte values are used for padding.
 *  pEncryptedData - where to copy the encrypted data
 *  pEncryptedLength - when called this is the maximum length allowed for
 *    ecnrypted data, on return this should be set to the length of the
 *    encrypted data.
 * returns:
 *  TMWDEFS_TRUE if successful
 *  TMWDEFS_FALSE otherwise
 */
TMWTYPES_BOOL mdnpdata_authEncryptKeyWrapData(
  void            *pHandle,
  TMWTYPES_USHORT  userNumber,
  TMWTYPES_UCHAR   keyWrapAlgorithm,
  TMWTYPES_UCHAR  *pPlainData,
  TMWTYPES_USHORT  plainDataLength,
  TMWTYPES_UCHAR  *pEncryptedData,
  TMWTYPES_USHORT *pEncryptedLength);

/* function: mdnpdata_authHMACSupport
 * purpose:  This function should determine whether the MAC algorithm
 *   requested by the outstation is supported and return the length of
 *   the MAC data requested by this algorithm
 *    NOTE: Only required for SA_VERSION2, see utils/tmwcrypto.h for SA_VERSION5
 * arguments:
 *  HHMACAlgorithm - HMAC algorithm
 *   DNPAUTH_HMAC_SHA1_4OCTET  Only for SA V2.
 *   DNPAUTH_HMAC_SHA1_8OCTET
 *   DNPAUTH_HMAC_SHA1_10OCTET
 *   DNPAUTH_HMAC_SHA256_8OCTET
 *   DNPAUTH_HMAC_SHA256_16OCTET
 *   other values reserved for future use or vendor specific choices
 * returns:
 *  length of data to be generated if algorithm is supported.
```

```
     *  0 if algorithm is not supported
     */
    TMWTYPES_CHAR mdnpdata_authHMACSupport(
      TMWTYPES_UCHAR HMACAlgorithm);

    /* function: mdnpdata_authHMACValue
     * purpose: Using the specified algorithm and key calculate the MAC value.
     * Copy up to the number of bytes allowed by *pMACValueLength into *pMACValue
     * and set *pHMACValueLength to the number of bytes copied.
     *    NOTE: Only required for SA_VERSION2, see utils/tmwcrypto.h for SA_VERSION5
     * arguments:
     *  algorithm - algorithm to use for creating hash value
     *   DNPAUTH_HMAC_SHA1_4OCTET  Only for SA V2.
     *   DNPAUTH_HMAC_SHA1_8OCTET
     *   DNPAUTH_HMAC_SHA1_10OCTET
     *   DNPAUTH_HMAC_SHA256_8OCTET
     *   DNPAUTH_HMAC_SHA256_16OCTET
     *   other values reserved for future use or vendor specific choices
     *  pKey - key to use
     *  pData - pointer to data to hash
     *  dataLength - length of data to hash
     *  pHMACValue - pointer where hashed data should be copied
     *  pHMACValueLength - when called this is the maximum length allowed for
     *   hashed data, on return this should be set to the length of the hashed data.
     * returns:
     *  TMWDEFS_TRUE if successful
     *  TMWDEFS_FALSE otherwise
     */
    TMWTYPES_BOOL mdnpdata_authHMACValue(
      TMWTYPES_UCHAR    algorithm,
      DNPDATA_AUTH_KEY *pKey,
      TMWTYPES_UCHAR   *pData,
      TMWTYPES_ULONG    dataLength,
      TMWTYPES_UCHAR   *pHMACValue,
      TMWTYPES_USHORT  *pHMACValueLength);

    /* function: mdnpdata_authRandomChallengeData
     * purpose:  generate pseudo-random data, at least 4 bytes long, using algorithm
     *   specified in FIPS 186-2 Digital Signal Standard
     *    NOTE: Only required for SA_VERSION2, see utils/tmwcrypto.h for SA_VERSION5
     * arguments:
     *  pBuf - pointer to where random data should be copied
     *  minLength - minimum length of data as required by spec
     *  pLength -  when called this is the maximum length allowed for the random data,
     *   on return this should be set to the length of the random data.
     * returns:
     *  TMWDEFS_TRUE of successful
     *  TMWDEFS_FALSE otherwise
     */
    TMWTYPES_BOOL TMWDEFS_LOCAL mdnpdata_authRandomChallengeData(
      TMWTYPES_UCHAR  *pBuf,
      TMWTYPES_USHORT  minLength,
      TMWTYPES_USHORT *pLength);

#endif

#ifdef __cplusplus
}
#endif
#endif /* MDNPDATA_DEFINED */
```

8 Advanced Topics

8.1 DNP3 Data Sets

Data Sets provide a method for transferring structured data within the DNP3 protocol. Full support for Data Sets is provided in the SDNP Source Code Library. For more information on Data Sets see the latest version of the DNP3 Application Layer and Object Library Specifications.

Data Sets consist of Data Set Prototype and Descriptor objects, each containing a sequence of elements describing the data contained in Data Set Present Value or Snapshot Event objects. Each descriptor or prototype element consists of a descriptor code, data type code, maximum length and ancillary value. Prototypes are identified by UUIDs and can be contained by more than one Descriptor object. The actual Data Set data objects (Object Group 87 or 88) contain only a sequence of elements containing length and value. The Descriptor and Prototypes (Object Group 85 and 86) must be present on a device in order to make sense of the received Data Set data.

Data Set Static Data Objects can be read and written by a DNP3 Master and Data Set Event Objects can be sent by an Outstation spontaneously or in response to a specific or class read request.

The MDNP SCL provides three request functions; mdnpbrm_writeDataset, mdnpbrm_writeDatasetProto and mdnpbrm_writeDatasetDescr, to allow writing the Data Set objects to the slave. For each of these, the data elements to be sent to the outstation are retrieved using mdnpdata interface functions. The mdnpdata database interface requires; mdnpdata_storeDatasetxxx, mdnpdata_datasetProtoxxx, mdnpdata_datasetDescrxxx and mdnpdata_datasetxxx functions to retrieve and store Data Set information from/to the database. The existing request functions; mdnpbrm_readGroup and mdnpbrm_readPoints and mdnpbrm_assignClass also apply to Data Sets.

Data Set Descriptor Object Group 86 supports 3 variations. The information transferred in these 3 variations is different. Variation 1 which can be read or written, describes the data contained in a Data Set. Variation 2, characteristics, which can be read but not written, indicates whether Data Set Values can be read or written, if outstation maintains static and or event Data Sets and if the Data Set was defined by the master or outstation.

A simple example implementation of the database for the master Data Set functionality can be found in mdnpsim.c. As always, the code in mdnpdata.c should be replaced with your own database implementation.

8.2 DNP3 Secure Authentication

In February 2007 the DNP3 Users group released the DNP3 Specification Volume 2, Supplement 1 Secure Authentication Version 1. The MDNP and SDNP Source Code Libraries have provided support for this Secure Authentication specification. During 2007 and 2008, after further review and discussion, changes were proposed and agreed

upon, that significantly changed some message formats and functionality. These changes resulted in Version 2.00 of the DNP3 Specification Secure Authentication being released at the end of July 2008.

Version 3.5.0 of the MDNP and SDNP SCLs implemented Version 2.00 of the DNP3 Specification Secure Authentication (SAv2).

DNP3 Secure AuthenticationVersion 5 (SAv5) was released in November 2011. This version of Secure Authentication is contained in IEEE 1815-2012. This version adds the ability to remotely distribute User Update Keys over the DNP3 protocol under the direction of a trusted third party known as an Authority. Other changes to SAv5 to address problems in SAv2 make it incompatible with SAv2. In order to interoperate with a device that supports SAv2 that association must use SAv2.

Starting in Version 3.17.0, MDNP and SDNP SCLs are shipped without Secure Authentication or with implementations of both SAv2 and SAv5. Either or both SAv2 and SAv5 can be selected at compile-time, and if both versions are compiled in, can be configured for use per association at run-time. Because of the issues in SAv2 it is recommended that all NEW secure DNP devices implement SAv5, though they may contain SAv2 if required for backward compatibility with older DNP3 SA devices. NEW devices should NOT be implemented with SAv2 only.

8.2.1 Basic Authentication Model

The Secure Authentication specification uses a Challenge/Response mechanism to provide authentication, but not encryption, of DNP3 commands and responses. This utilizes Symmetric User Update Keys which in SAv2 had to be preconfigured on both the master and outstation. Secure Authentication also provides for the concept of multiple users, each with their own set of Update Keys and role (permissions). The master is responsible for generating short time period Session keys, encrypting these Session keys and sending these to the outstation. These Session keys are then used in the authentication process.
When a message (request), which is considered critical, is received by the outstation, the outstation will send a challenge to the master. The master will use the Session Key to generate a Message Authentication Code (MAC) for the critical request and send a challenge reply containing this to the outstation. The outstation will then perform the same MAC function and verify that the request came from the proper master. Critical responses from the outstation to the master can be verified in a similar manner, with the master sending the challenge.
In SAv5 support for remotely distributing User Update Keys over the DNP protocol was added. A trusted third party or Authority may direct the master to Add a user to an association using a Globally Unique Name with a particular role or set of permissions. Similarly, the Authority can Modify a User’s role or Delete a User.

8.2.2 SCL Configuration

The SCLs that contain Secure Authentication as shipped are configured to compile in the Secure Authentication code for SAv5. Support for Secure Authentication may be changed by modifying the following #defines in the specified files. Support for OpenSSL is NOT

configured in by default. You should read the section in this manual on OpenSSL Integration for more details.

If the library purchased contains Secure Authentication support, SAv2 and/or SAv5 support can be modified by setting the following defines appropriately:

In utils/tmwcnfg.h

To support SAv5, common cryptography interface tmwcrypto.h/c is required. It can optionally be used for SAv2. NOTE: If you support CRYPTO but not OPENSSL you must implement the functions in tmwcrypto.c.
#define TMWCNFG_SUPPORT_CRYPTO TMWDEFS_TRUE

To support optional asymmetric methods for SAv5.
#define TMWCNFG_SUPPORT_CRYPTO_ASYM TMWDEFS_TRUE

To support optional AES-GMAC algorithm for SAv5.
#define TMWCNFG_SUPPORT_CRYPTO_AESGMAC TMWDEFS_TRUE

To support the example interface to OpenSSL.
(this also requires installing and building OpenSSL as described above)
#define TMWCNFG_USE_OPENSSL TMWDEFS_TRUE

To allow the SCL to build and run without OpenSSL or another cryptography library the following define is set by default. Simulated cryptography will NOT encrypt, decrypt or perform other cryptographic functionality so it does not actually implement Secure Authentication. If you do not use OpenSSL and replace it with your own crypto library this define should be set to TMWDEFS_FALSE.
#define TMWCNFG_USE_SIMULATED_CRYPTO TMWDEFS_TRUE

To allow the SCL to build and run without an actual cryptographic database to store Authentication User Keys the following define is set by default. The simulated database is NOT sufficient for Secure Authentication implementations and Must be replaced with more complete database functionality. Search for the following define in tmwcrypto.c to see what functions must be implemented.
#define TMWCNFG_USE_SIMULATED_CRYPTO_DB TMWDEFS_TRUE

In dnp/dnpcnfg.h

To Support Secure Authentication (if purchased).

#define DNPCNFG_SUPPORT_AUTHENTICATION TMWDEFS_TRUE

To Support Secure Authentication Version 2 SAv2.

#define DNPCNFG_SUPPORT_SA_VERSION2 TMWDEFS_TRUE

To Support Secure Authentication Version 5 SAv5.

#define DNPCNFG_SUPPORT_SA_VERSION5 TMWDEFS_TRUE

To Support Secure Authentication Version 5 SAv5 optional remote key update.

#define DNPCNFG_SUPPORT_AUTHKEYUPDATE TMWDEFS_TRUE

To Support Secure Authentication Version 5 SAv5 optional asymmetric methods

#define DNPCNFG_SUPPORT_AUTHKEYUPDASYM TMWDEFS_TRUE

In addition, the Secure Authentication database functions in mdnpdata.c must be implemented. A description of these functions, named mdnpdata_authxxx(), is contained in the file mdnpdata.h and further information is provided below.

If SAv5 is supported, functions in the common cryptography interface utils/tmwcrypto.c must also be implemented. A description of these functions, named tmwcrypto_xxx() is contained in the file tmwcrypto.h and further information is provided below.

For backward compatibility, SAv2 continues to use the cryptography functions that were provided through the mdnpdata interface. If both SAv2 and SAv5 are required, both cryptographic function interfaces mdnpdata and tmwcrypto must be implemented. While this common crypto interface is not required for an SAv2 implementation, there is example code in mdnpdata.c that maps the old database crypto interface to the new common crypto interface.

When opening a session that is intended to support Secure Authentication, the field authenticationEnabled in the MDNPSESN_CONFIG structure should be set to TMWDEFS_TRUE, and the other authentication configuration fields in the MDNPSESN_AUTH_CONFIG structure should be set to the desired values. These configuration parameters are described in mdnpsesn.h

Example code for configuring and using Secure Authentication has been added to the sample application DNPMaster.cpp. This code is surrounded by #if MDNPDATA_SUPPORT_OBJ120 #endif. In order for Secure Authentication to function properly in these examples, the functions mdnpdata_authxxx() and (for SAv5) tmwcrypto_xxx() must be implemented. For information about these functions please read the next two sections of this manual.

8.2.3 mdnpdata_authxxx() implementation

Detailed information about the specific interface for each function is provided in mnpdata.h. More general information about some of the Secure Authentication algorithms and functions that must be implemented is provided here.

In version 3.14.0 of the SCL, a new common cryptography interface was defined in utils/tmwcrypto.h. This common interface will be used for all new cryptography functionality, including key management, encryption and decryption, as well as signing data and verification of signatures. For backward compatibility purposes, the SAv2 implementation will continue to call the mdnpdata_authxxx() functions for *cryptography* related functions. Both SAv2 and SAv5 will call mdndata_authxxx() functions for *database* related functions that are used to support Secure Authentication. There is sample code in mdnpdata.c that maps the SAv2 database crypto functions to the new common crypto interface.

mdnpdata_authLogxxx() - The Secure Authentication specification recommends that implementations log all successful and unsuccessful authentications and key changes. The format of the log is not part of the specifications. These mdnpdata_authLogxxx functions will be called by the SCL for all significant authentication events.

8.2.3.1 SAv2 Database Implementation

mdnpdata_authGetNewSessionKeys() - Called periodically, for each user number that is active, to generate and return a set of session keys, which are used for authenticating critical messages.

mdnpdata_authEncryptKeyWrapData() - Encrypt the key wrap data (including the new session keys) sent from the Master to the Outstation using the specified algorithm. For SAv2 the AES Key Wrap algorithm specified in RFC3394 is the only required algorithm. It is important to note that this Key Wrap algorithm is not the same thing as AES Symmetric Key Encryption. The AES Encryption function will actually be called multiple times to perform the Key Wrap, depending on the length of the key data to be encrypted. There is a sample implementation of this Key Wrap Algorithm in mdnpdata.c

mdnpdata_authHMACValue() - Generates a Keyed-Hash Message Authentication Code (HMAC) value on the challenged message using the specified algorithm and session key. There are 2 algorithms (SHA1 and SHA256) and 2 data lengths for each algorithm specified.

mdnpdata_authRandomChallengeData() - Generates pseudo random data to be included in authentication messages. This function should use the algorithm specified in the FIPS 186-2 Digital Signal Standard.

8.2.3.2 SAv5 Database Implementation

Mandatory statistics functions

mdnpdata_authSecStatThreshold() - Retrieve the security statistic threshold value from the database.

mdnpdata_authStoreSecStat() – Store a security statistic that has been generated on the Master device.

mdnpdata_authStoreOSSecStat() - Store a security statistic that has been generated on the outstation device and received here on the Master. This could have been sent as an event or as a static object from the Outstation.

Optional Remote Key Update functions (`DNPCNFG_SUPPORT_AUTHKEYUPDATE`)

mdnpdata_authGetOSName () – Get the name of the Outstation this Master is connected to. This must be preconfigured to match on both the Master and the Outstation.

mdnpdata_authGetUserName() - Get the globally unique user name provided by the Authority from the database.

mdnpdata_authGetChangeUserData() - Get the data provided by the Authority required for a g120v10 User Status Change request to be sent to the outstation as part of the Remote Key Update procedure.

mdnpdata_authGetCertData() - Get the certification data provided by the Authority which is used to certify that the other data in the status request is good.

mdnpdata_authStoreUpdKeyChangeReply() – Store the user number returned from the Outstation to be used for the globally unique user that was added by the Authority.

8.2.4 tmwcrypto_xxx() implementation

Detailed information about the specific interface for each function is provided in tmwcrypto.h. General information about some of the cryptography functions that must be implemented is provided here. These functions come with code that interfaces with OpenSSL if you define TMWCNFG_USE_OPENSSL.

tmwcrypto_algorithmSupport() – Is the algorithm specified using one of the TMWCRYPT_ALG_xxx defines supported?

tmwcrypto_MACValue() – Using the specified key and algorithm calculate the MAC value for the specified data.

tmwcrypto_getRandomData() – Generate pseudo random data to be included in authentication messages. This should use the algorithm specified in the FIPS 186-2 Digital Signal Standard.

tmwcrypto_genDigitalSignature()- Master only Generate a digital signature for the specified data.

tmwcrypto_verifySignature()-Outstation only Verify the signature for the specified data.

tmwcrypto_encryptData()-Master only Encrypt the data using the algorithm and key specified.

tmwcrypto_decryptData()- Outstation only Decrypt the data using the algorithm and key specified.

tmwcrypto_generateNewKey()- Master only Generate a new key of the specified type.

tmwcrypto_getKey()-Get the specified key. See description of keys below.

tmwcrypto_getKeyData()- Master only Get the specified key in a form that can be sent to the outstation. This interface returns the actual key itself and not just the common key structure, since the key must be sent over DNP to the outstation.

tmwcrypto_setKeyData()- Outstation only Store the specified key that was sent by the Master. This interface includes the actual key itself and not just the common key structure since it was received over DNP from the Master.

tmwcrypto_commitKey()- Outstation only Indicate if the Key stored by tmwcrypto_setKeyData should be put into use, or discarded.

tmwcrypto_getCertificate()- Master only if optional User Certificate is supported Get the IEC 62351-8 (X.509) Certificate for the user specified.

tmwcrypto_verifyCertificate()- Outstation only if optional User Certificate is supported Verify the Certificate using the Authority Public key.

tmwcrypto_putCertificate()- Outstation only if optional User Certificate is supported Store the Certificate for the user specified.

8.2.4.1 tmwcrypto Key Management

A common key structure TMWCRYPTO_KEY has been defined in tmwcrypto.h. This will be used by the SCL to “retrieve” keys that are then used in other tmwcrypto functions such as tmwcrypto_encrypt() and tmwcrypto_decrypt(). The contents of this key structure are not used by the SCL. Your implementation may return a handle instead of the key value itself which would then be passed to the encrypt function. The example implementation copies the actual symmetric key into the structure, while using file names for asymmetric keys. Since you are responsible for the key structure, as well as the tmwcrypto_getKey and other cryptography functions that use the key, you can decide how keys should be handled in your system.

8.2.5 Key Types

Depending on whether the optional remote User Update Key Change and Asymmetric Key Change methods are supported a variety of keys are required for DNP Secure

Authentication. These keys are defined in tmwcrypto.h. Here is a description of each of these keys and where they are generated, configured and used.

TMWCRYPTO_USER_UPDATE_KEY - User Update Key, 1 per User. Used to encrypt/decrypt the Session Keys when sending them from the Master to the Outstation. These Keys can be preconfigured per User on both the Master and Outstation or dynamically added, changed, and sent over DNP to the Outstation under the direction of the Authority if SAv5 is supported.

TMWCRYPTO_USER_CONTROL_SESSION_KEY - User Control Direction Session Key, 1 per User. This Key is used for performing a MAC function on critical messages for Authentication purposes. This key is generated periodically by the Master and sent to the Outstation over DNP.

TMWCRYPTO_USER_MONITOR_SESSION_KEY - User Monitor Direction Session Key, 1 per User. This Key is used for performing a MAC function on critical messages for Authentication purposes. This key is generated periodically by the Master and sent to the Outstation over DNP.

TMWCRYPTO_USER_ASYM_PUB_KEY - User Public Key, 1 per user, when an Asymmetric Key Change method is used. The specification says the user should generate the Public/Private key pair and provide the Public Key to the Master. This key pair could be generated by the Authority and given to the User, or even generated by the Master on behalf of the User and given to the Authority. The Authority must certify this key and therefore both the Master and Authority must have this key.

TMWCRYPTO_USER_ASYM_PRV_KEY - User Private Key, 1 per user, when an Asymmetric Key Change Method is used. The specification says the user should generate the Public/Private key pair and provide the public and private key to the Master. This key pair could be generated by the Authority and given to the User, or even generated by the Master on behalf of the User. Only the Master will use the User Private Key to sign User Update Key data to be sent to the Outstation. The Outstation will use the User Public Key that the master has sent to the Outstation over DNP in a g120v10 object.

TMWCRYPTO_OS_ASYM_PUB_KEY - Outstation Public Key, 1 per Outstation, when an Asymmetric Key Change Method is used. The Outstation Public/Private key pair should be generated on the Outstation and the Public Key securely configured on the Master.

TMWCRYPTO_OS_ASYM_PRV_KEY- Outstation Private Key, 1 per Outstation, when an Asymmetric Key Change Method is used. The Outstation Public/Private key pair should be generated on the Outstation. The Private Key will only be known by the Outstation.

TMWCRYPTO_AUTH_CERT_SYM_KEY - Authority Certification Key, 1 per Outstation, when a Symmetric Key Change Method is used. The secret Certification Key will be generated on the Authority and must be securely configured on the Outstation. Both the Authority and the Outstation must know this key.

TMWCRYPTO_AUTH_ASYM_PUB_KEY- Authority Public Key, 1 per Authority, when an Asymmetric Key Change Method is used. The Public/Private key pair should be generated on the Authority and the Public Key must be securely configured on the Outstation.

TMWCRYPTO_AUTH_ASYM_PRV_KEY- Authority Private Key, 1 per Authority, when an Asymmetric Key Change Method is used. The Public/Private key pair should be generated on the Authority and the Private Key will only be known by the Authority.

8.2.6 User Update Key Management

The same User Update Keys that are used for sending session keys from the Master to the Outstation must be present on both the Master and Outstation. In SAv2 the mechanism for configuring the keys on the Master and Outstation was outside of the DNP3 Secure Authentication specification. SAv5 added optional distribution of Update Keys over the DNP3 protocol under the direction of a trusted third party known as the Authority. Each user number that is active on an association must have its own unique Update Key. These Update Keys will be used by the mdnpdata_authEncryptKeyWrapData() (for SAv2) or tmwcrypto_encryptData() functions. Depending on the target environment, access to these Update Keys should be protected by encryption, password protection or other means as deemed necessary.

8.2.7 Building Secure Authentication in DNPMaster.cpp Sample

The sample application DNPMaster.cpp contains code for configuring and using Secure Authentication. This code is surrounded by #if MDNPDATA_SUPPORT_OBJ120 #endif which by default is set to false. To compile in Secure Authentication in the MDNP SCL as well as this sample perform the following steps.

Set the defines appropriately in utils/tmwcnfg.h and dnp/dnpcnfg.h as described in the SCL Configuration section above.

In order for Secure Authentication to function properly in these examples, the functions mdnpdata_authxxx() and tmwcrypto_xxx() must first be implemented or the example tmwcrypto implementation linked against the OpenSSL library. In order to use OpenSSL to provide this functionality, enable the SCL's interface to OpenSSL by defining **TMWCNFG_USE_OPENSSL** in utils/tmwcnfg.h. Then refer to section 5.6 OpenSSL Integration, which describes how to integrate OpenSSL with this sample.

In the file DNPMaster.cpp
Set myUseAuthentication = true;
And set myUseSAv2 = true if you want to use SAv2 instead of SAv5

Build the DNPMaster application

8.2.8 Sample Asymmetric Keys Included With SCL

To facilitate testing SAv5 Remote Key Change some sample asymmetric private and public key files are included with the Source Code Library.

THESE SAMPLE KEYS SHOULD ONLY BE USED FOR TESTING!

The Key Change Method specifies the required type (RSA or DSA) and size of the asymmetric keys in bits. For each method. Asymmetric Key Change Method 67 requires 1024, 68 and 70 require 2048, and 69 and 71 require 3072. If an incorrect type or size key is specified for a particular method the Test Harness will indicate this and will fail to update the user.

TB2016-002 has added 5 new key change methods that use RSA keys for signing instead of DSA keys. Support for these was added in version 3.20.000 of the SCLs and Communication Protocol Test Harness.

Sample DNP Authority Private and Public Key pairs when the Test Harness or MDNP session is simulating the Authority. Configured in MDNP and SDNP sessions parameters AuthCryptoDb: AuthorityAsymPrvKey on MDNP Session, AuthCryptoDb: AuthorityAsymPubKey on SDNP session. In the MDNP and SDNP library, these keys are part of the cryptography database and not configured in the library itself.

TMWTestAuthorityDsa1024PrvKey.pem, …PubKey.pem, …Cert.pem
TMWTestAuthorityDsa2048PrvKey.pem, …PubKey.pem, …Cert.pem
TMWTestAuthorityDsa3072PrvKey.pem, …PubKey.pem, …Cert.pem
TMWTestAuthorityRsa1024PrvKey.pem, …PubKey.pem, …Cert.pem
TMWTestAuthorityRsa2048PrvKey.pem, …PubKey.pem, …Cert.pem
TMWTestAuthorityRsa3072PrvKey.pem, …PubKey.pem, …Cert.pem

Sample Outstation Private and Public Key pairs. Configured in MDNP and SDNP sessions. AuthCryptoDb: AuthOSAsymPubKey on MDNP Session, AuthCryptoDb: AuthOSAsymPrvKey on SDNP session.

TMWTestOSRsa1024PrvKey.pem, TMWTestOSRsa1024PubKey.pem
TMWTestOSRsa2048PrvKey.pem, TMWTestOSRsa2048PubKey.pem
TMWTestOSRsa3072PrvKey.pem, TMWTestOSRsa3072PubKey.pem

Sample User Private and Public Key pairs. Used in the Test Harness mdnpauthuserstatuschange command parameter UserPrvKey and UserPubKey when using an asymmetric method or when calling mdnpbrm_authUserStatusChange() function.

TMWTestUserDsa1024PrvKey.pem, …PubKey.pem, …Cert.pem
TMWTestUserDsa2048PrvKey.pem, …PubKey.pem, …Cert.pem
TMWTestUserDsa3072PrvKey.pem, …PubKey.pem, …Cert.pem

TMWTestUserRsa1024PrvKey.pem, …PubKey.pem, …Cert.pem
TMWTestUserRsa2048PrvKey.pem, …PubKey.pem, …Cert.pem
TMWTestUserRsa3072PrvKey.pem, …PubKey.pem, …Cert.pem

The included sample keys were generated using the following OpenSSL commands found in MakeCerts.bat . You should generate your own keys for testing and actual deployment.

Here is an example to Generate DSA 2048 private keys:
openssl dsaparam -out ca_dsaparam.pem 2048
openssl req -newkey dsa:ca_dsaparam.pem -sha256 -keyout TMWTestAuthorityDsa2048PrvKey.pem -out ca_dsa2048.csr -config ca.cnf -passout pass:triangle
openssl x509 -req -in ca_dsa2048.csr -sha256 -extfile ca.cnf -extensions certificate_extensions -signkey TMWTestAuthorityDsa2048PrvKey.pem -days 9999 -out TMWTestAuthorityDsa2048Cert.pem -passin pass:triangle
openssl dsa -in TMWTestAuthorityDsa2048PrvKey.pem -out TMWTestAuthorityDsa2048PubKey.pem -pubout -outform PEM -passin pass:triangle

Here is an example to Generate User DSA 2048 keys and X.509 Signed Certificates:
openssl dsaparam -out dsa_param.pem 2048
openssl req -newkey dsa:dsa_param.pem -sha256 -keyout TMWTestUserDsa2048PrvKey.pem -out dsa2048.csr -days 9999 -config user.cnf -subj "/CN=Common/C=US/ST=North Carolina/L=Raleigh/O=Triangle MicroWorks, Inc./" -passout pass:triangle
openssl ca -in dsa2048.csr -out TMWTestUserDsa2048Cert.pem -cert TMWTestAuthorityDsa2048Cert.pem -keyfile TMWTestAuthorityDsa2048PrvKey.pem -config ca.cnf -passin pass:triangle -outdir certs -batch -noemailDN
openssl dsa -in TMWTestUserDsa2048PrvKey.pem -out TMWTestUserDsa2048PubKey.pem -pubout -outform PEM -passin pass:triangle

Here is an example to Generate RSA 2048 private keys for Outstation:
openssl genrsa -out TMWTestOSRsa2048PrvKey.pem 2048
openssl rsa -in TMWTestOSRsa2048PrvKey.pem -out TMWTestOSRsa2048PubKey.pem -pubout -outform PEM

Here is an example to Generate the Authority RSA 2048 keys and the User RSA keys and Certificates and sign them for for DNP3 TB2016-002:
openssl genrsa -out TMWTestAuthorityRsa2048PrvKey.pem 2048 -config ca.cnf -passout pass:triangle
openssl rsa -in TMWTestAuthorityRsa2048PrvKey.pem -out TMWTestAuthorityRsa2048PubKey.pem -pubout -outform PEM
openssl x509 -req -in ca_dsa2048.csr -extfile ca.cnf -extensions certificate_extensions -signkey TMWTestAuthorityRsa2048PrvKey.pem -days 9999 -out TMWTestAuthorityRsa2048Cert.pem -passin pass:triangle
openssl genrsa -out TMWTestUserRsa2048PrvKey.pem 2048
openssl rsa -in TMWTestUserRsa2048PrvKey.pem -out TMWTestUserRsa2048PubKey.pem -pubout -outform PEM

openssl ca -in dsa2048.csr -out TMWTestUserRsa2048Cert.pem -cert TMWTestAuthorityRsa2048Cert.pem -keyfile TMWTestAuthorityRsa2048PrvKey.pem -config ca.cnf -passin pass:triangle -outdir certs -batch -noemailDN

8.2.9 Debugging

The Triangle MicroWorks Protocol Test Harness is a good way to test your implementation of Secure Authentication on a DNP3 Master. A slave (Outstation) session can be opened with Secure Authentication enabled. By configuring the same version Authentication Users and Update keys a number of tests can be performed. If the optional SAv5 component was purchased, Remote Key Distribution methods can also be tested. When an sdnp session is opened, AuthenticationEnabled should be set to true. Setting the session parameter AuthExtraDiags (under Advanced Settings) to true will cause additional diagnostic information to be displayed in the protocol analyzer window. AuthOperateInV2Mode can be used to control whether the session uses SAv2 or SAv5.

The protocol analyzer window can be used to look at the Secure Authentication message sequences. Communication with the Test Harness can be used to verify that the key generation, wrap, encryption, hashing functions and key change methods are properly implemented on the Master.

In the Tcl command window the command sdnpauthentication can be used for some Secure Authentication testing. You can cause solicited or unsolicited responses to be sent with an Aggressive Mode object. You can force a Session Key to timeout. You can cause responses containing events to issue a “preChallenge” of the expected application confirm. You can also force a Secure Authentication Error Message to be sent when the next request is received on the Test Harness slave. Enter sdnpauthentication in the Test Harness Tcl command window for more details.

8.3 DNP3 Device Profile as a Binary Configuration File

While the MDNP SCL cannot read an XML Device Profile directly, it does have the ability to read a file that contains most of the configuration values described in Section 1 of the DNP3 Device Profile. A complete DNP3 Device Profile contains much information that is not relevant to a master during its start up or operation. A DNP3 Binary Configuration file contains only the current values that could be used by the master to perform configuration tasks. The MDNP SCL contains a method that will read values from a DNP3 Binary Configuration file and apply the values to setting in the SCL to configure the master. The values are applied to setting that are both internal to the library and setting that are controlled in the target layer. The method applies all values found in the DNP3 Binary Configuration file for which the library or target layer has a corresponding setting. The method can also read a DNP3 Binary Configuration file that describes an outstation and use that file to configure a dnp channel and mdnp session to talk to the outstation.

DNP3 Binary Configuration files are created by the DNP3 Forge application. DNP3 Forge creates a Binary Configuration file in much the same way that it creates an XML Device Profile file. The user fills in values for applicable sections of the device profile and can then export those values as a Binary Configuration file. Only the current values from subsections with values are included in the file. The user can create a new device profile document from templates that are prepopulated with typical values that are used by the DNP3 SCL. The DNP3 Protocol Test Harness can also read Binary Configuration files to configure masters and outstations for testing purposes. The advantage of using a Binary Configuration file as opposed to an XML Device Profile file is that the binary file is considerably smaller and the SCL does not need the processing overhead of parsing an XML file that contains large amounts of unusable data.

In order to use DNP3 Binary Configuration file features in the DNP3 SCL, the DNPCNFG_SUPPORT_BINCONFIG flag must be set to TMWDEFS_TRUE in dnpcnfg.h to compile in the Binary Configuration file processing code. With this flag set, then the mdnpsesn_applyBinaryFileValues(…) can be called to read and process a Binary Configuration file. Parameters passed to this method include the path to the Binary Configuration file and the set of the channel (DNPCHNL_CONFIG), link (DNPLINK_CONFIG), session (SDNPSESN_CONFIG), and target configuration structures. These structures should be set to default values before calling the method. Values retrieved from the Binary Configuration are written to the structures if they are found in the file. If the operation completes successfully, the method returns a true value. If the operation fails, the configuration structures should be discarded and not used to create a channel.

The Binary Configuration file feature has been added as of version 3.18 of the DNP3 SCL. In the first version, only the Windows target layer is supported. Additional target layer support will be added in future releases. If not using winiotarg, other targets can be utilized by implementing tmwtarg_initBinFileValues(…) and tmwtarg_applyBinFileTargValues(…).

8.4 DNP3 File Transfer

DNP3 file transfer is performed using object group 70 variations 2,3,4,5, and 6 (variation 1 is no longer used). For more details about this see the DNP3 Specification Volume 2 Application Layer Part 1 and Part 2.

Optionally, the outstation may require authentication, which simply consists of the master sending a user name and password in the clear in a request containing g70v2. The outstation would return a key (0 if permission is denied) in a response containing g70v2. The master would use this one time key in the following open or delete request.

The master sends an open File Command request containing g70v3. The outstation responds with a File Command Status g70v4 indicating success or failure. If it was successful, the response contains a file handle which should be used for subsequent file read, write, close or abort requests.

The master reads data from the outstation using g70v5, containing the file handle and a sequentially increasing block number. The outstation responds with a g70v5 object containing the block number and block of data that has been read. When the last block of data is being sent a flag indicating it is the last block will be set in the response.

The master sends data to the outstation using g70v5 containing the file handle, block number and block of data. The outstation will respond with a g70v6 File Transport Status object indicating success or failure.

If the outstation cannot read or write the data immediately, it can respond with a null response and then send the g70v5 or g70v6 as an event.

After all of the data has been transferred, the master should send a File Command Status object g70v4 request to close the file.

8.4.1 SCL Functionality

The MDNP Source Code Library provides a number of functions to provide support for file transfer. There are high level functions that will attempt to read or write and entire file and there are lower level functions that allow the application to request authentication, open, read or write blocks of data, and then close the file.

mdnpbrm_copyLocalFileToRemote, mdnpbrm_copyRemoteFileToLocal, mdnpbrm_readRemoteDirectory are examples of the high level functionality. Mdnpbrm_fileAuthentication, mdnpbrm_fileOpen, mdnp_fileRead, mdnp_fileWrite, and mdnp_fileClose are examples of the low level functionality provided. The simulated database has sample code that opens, closes, reads and writes from windows files.

8.5 Multiple Master Sessions on a DNP Channel

Multiple MDNP Sessions may be opened on a DNP Channel to communicate with multiple Outstation devices. Typical uses of this are over a serial multidrop network to communicate with multiple individual Outstation devices, over TCP to connect to a device that contains multiple virtual Outstations or over TCP to connect to a converter that facilitates RS232 communication to multiple individual Outstations at the remote end.

The DNP Specification requires there is only a single request outstanding per Association at a time. Before version 3.25 of the MDNP SCL all sessions on a channel were treated as a belonging to a single association requiring requests to multiple sessions on that channel to be sequential. An outstanding request on that channel would need to complete (or timeout) before another request from a master session on that channel would be sent.

In normal operation this works well and in some cases is required to prevent collisions with multiple simultaneous requests and responses. In situations where an Outstation is not responding however, requests to other Outstations on that channel may be delayed by channelResponseTimeout which could have a default value of 10 seconds.

Beginning in version 3.25 of the DNP library code a feature has been added that allows managing requests on each association between a Master session on a channel and the

respective Outstation separately. This feature is compiled out by default. If compiled in, it will require more memory as a receive fragment buffer is required per session instead of per channel, and if Secure Authentication Version 5 is also compiled in additional memory per session rather than per channel is also required. To compile in this feature, modify dnpcnfg.h to #define DNPCNFG_MULTI_SESSION_REQUESTS TMWDEFS_TRUE. No changes to your application or other configuration changes are required to use this feature.