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.

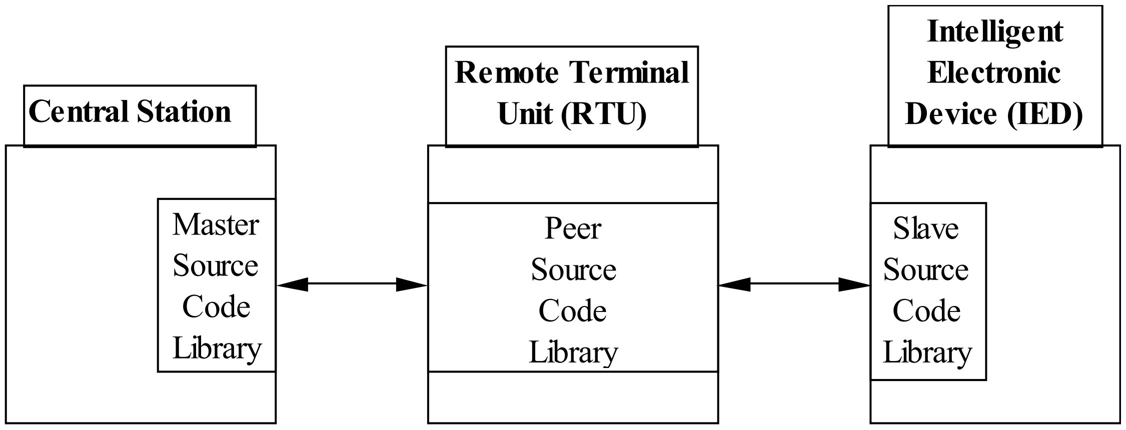

Central Station
Master Source Code Library
Remote Terminal Unit (RTU)
Peer Source Code Library
Intelligent Electronic Device (IED)
Slave Source Code Library

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.

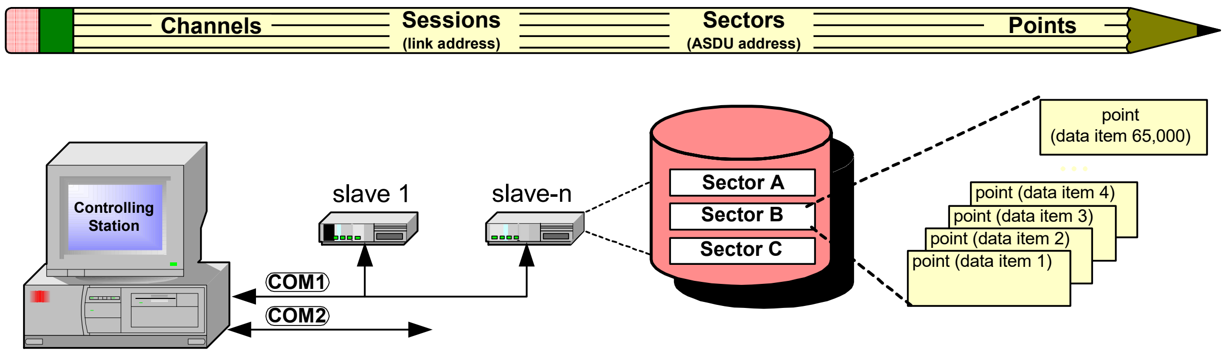


Figure 2: Triangle MicroWorks SCL Data Model

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.

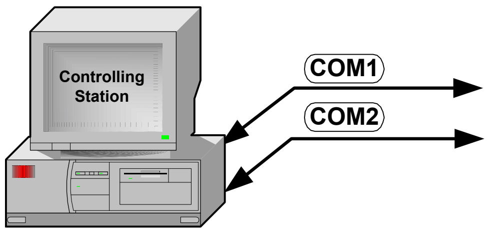


Figure 3: Illustration of Communication Channels

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.

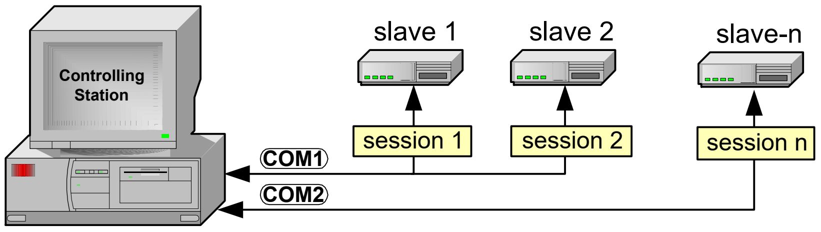


Figure 4: Illustration of Communication Sessions

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.

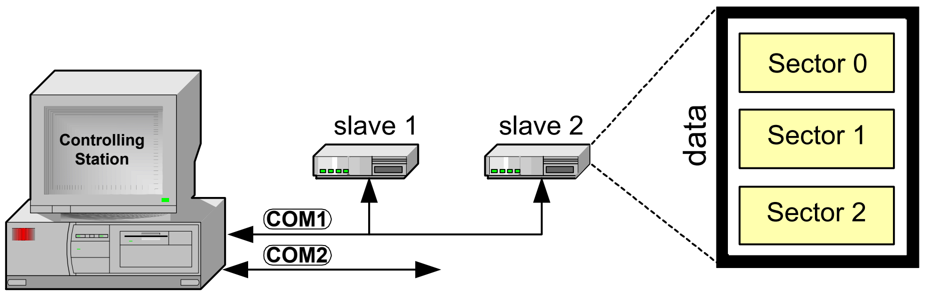


Figure 5: Illustration of Multiple Sectors

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.

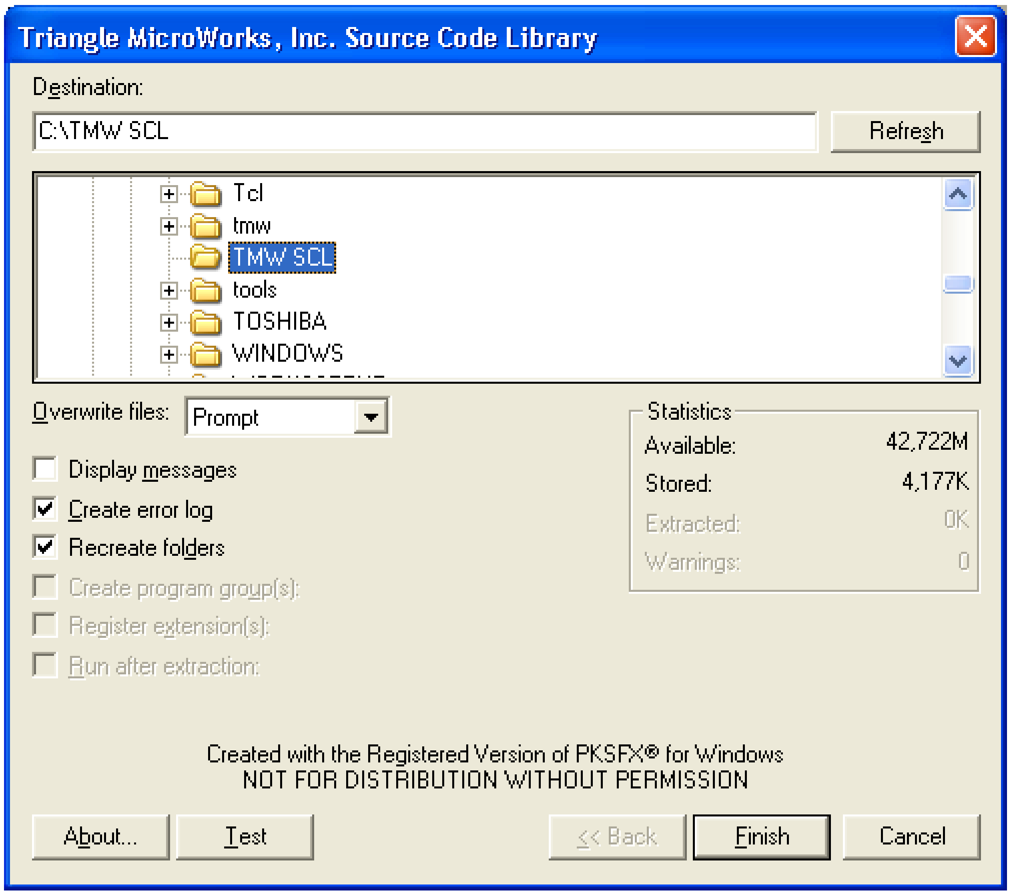
4.1.1 Encrypted Windows Zip File

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:


Triangle MicroWorks, Inc. Source Code Library

Destination: C:\TMW SCL

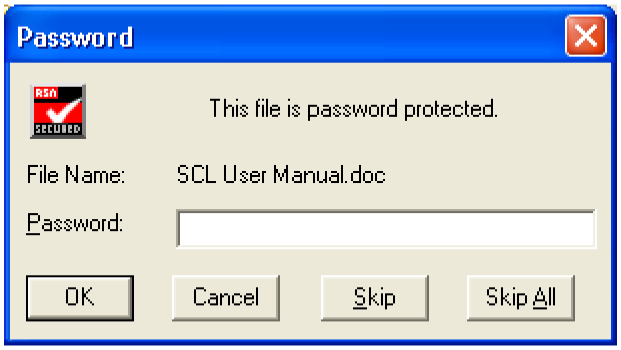
Overwrite files: Prompt

- [ ] Display messages
- [✓] Create error log
- [✓] Recreate folders
- [ ] Create program group(s):
- [ ] Register extension(s):
- [ ] Run after extraction:

Statistics	
Available:	42,722M
Stored:	4,177K
Extracted:	0K
Warnings:	0

Created with the Registered Version of PKSFX® for Windows
NOT FOR DISTRIBUTION WITHOUT PERMISSION

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:


Password

This file is password protected.

File Name: SCL User Manual.doc

Password:

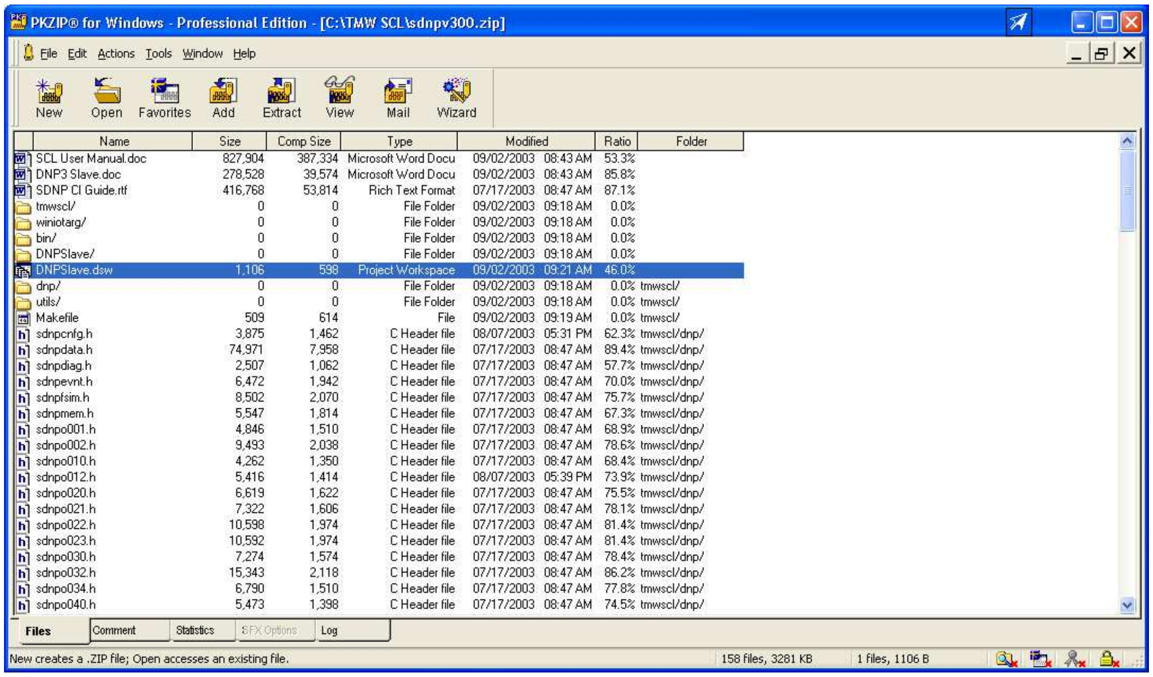
The SCL will then be extracted to the specified directory. Note that it is only necessary to enter the password once. If you are prompted to enter the password again for subsequent files, then you probably mistyped the password. Press Cancel, and repeat the above steps.

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.

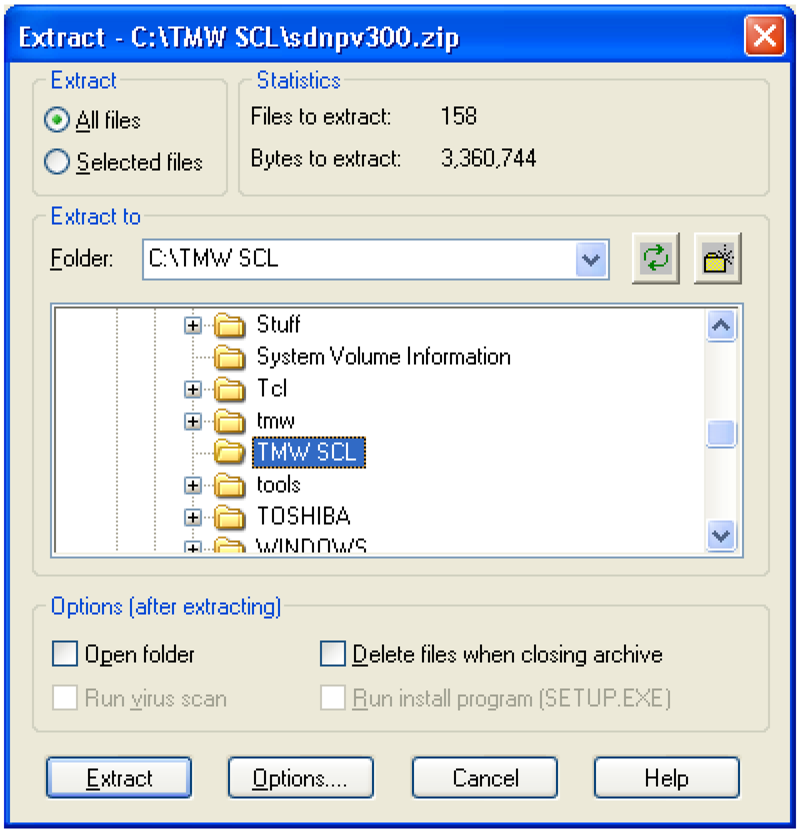
When you open the Zip file, a screen similar to the following is displayed:


PKZIP® for Windows - Professional Edition - [C:\TMW SCL\sdnpv300.zip]
File Edit Actions Tools Window Help
New Open Favorites Add Extract View Mail Wizard
Files Comment Statistics SFX Options Log

Name	Size	Comp Size	Type	Modified	Ratio	Folder
SCL User Manual.doc	827,904	387,334	Microsoft Word Docu	09/02/2003 08:43 AM	53.3%	
DNP3 Slave.doc	278,528	39,574	Microsoft Word Docu	09/02/2003 08:43 AM	85.8%	
SDNP CI Guide.rtf	416,768	53,814	Rich Text Format	07/17/2003 08:47 AM	87.1%	
tmwscl/	0	0	File Folder	09/02/2003 09:18 AM	0.0%	
winiotarg/	0	0	File Folder	09/02/2003 09:18 AM	0.0%	
bin/	0	0	File Folder	09/02/2003 09:18 AM	0.0%	
DNPSlave/	0	0	File Folder	09/02/2003 09:18 AM	0.0%	
DNPSlave.dsw	1,106	598	Project Workspace	09/02/2003 09:21 AM	46.0%	
dnp/	0	0	File Folder	09/02/2003 09:18 AM	0.0%	tmwscl/
utils/	0	0	File Folder	09/02/2003 09:18 AM	0.0%	tmwscl/
Makefile	509	614	File	09/02/2003 09:19 AM	0.0%	tmwscl/
sdnpcnfg.h	3,875	1,462	C Header file	08/07/2003 05:31 PM	62.3%	tmwscl/dnp/
sdnpdata.h	74,971	7,958	C Header file	07/17/2003 08:47 AM	89.4%	tmwscl/dnp/
sdnpdiag.h	2,507	1,062	C Header file	07/17/2003 08:47 AM	57.7%	tmwscl/dnp/
sdnpevnt.h	6,472	1,942	C Header file	07/17/2003 08:47 AM	70.0%	tmwscl/dnp/
sdnpfsim.h	8,502	2,070	C Header file	07/17/2003 08:47 AM	75.7%	tmwscl/dnp/
sdnpmem.h	5,547	1,814	C Header file	07/17/2003 08:47 AM	67.3%	tmwscl/dnp/
sdnpo001.h	4,846	1,510	C Header file	07/17/2003 08:47 AM	68.9%	tmwscl/dnp/
sdnpo002.h	9,493	2,038	C Header file	07/17/2003 08:47 AM	78.6%	tmwscl/dnp/
sdnpo010.h	4,262	1,350	C Header file	07/17/2003 08:47 AM	68.4%	tmwscl/dnp/
sdnpo012.h	5,416	1,414	C Header file	08/07/2003 05:39 PM	73.9%	tmwscl/dnp/
sdnpo020.h	6,619	1,622	C Header file	07/17/2003 08:47 AM	75.5%	tmwscl/dnp/
sdnpo021.h	7,322	1,606	C Header file	07/17/2003 08:47 AM	78.1%	tmwscl/dnp/
sdnpo022.h	10,598	1,974	C Header file	07/17/2003 08:47 AM	81.4%	tmwscl/dnp/
sdnpo023.h	10,592	1,974	C Header file	07/17/2003 08:47 AM	81.4%	tmwscl/dnp/
sdnpo030.h	7,274	1,574	C Header file	07/17/2003 08:47 AM	78.4%	tmwscl/dnp/
sdnpo032.h	15,343	2,118	C Header file	07/17/2003 08:47 AM	86.2%	tmwscl/dnp/
sdnpo034.h	6,790	1,510	C Header file	07/17/2003 08:47 AM	77.8%	tmwscl/dnp/
sdnpo040.h	5,473	1,398	C Header file	07/17/2003 08:47 AM	74.5%	tmwscl/dnp/

New creates a .ZIP file; Open accesses an existing file. 158 files, 3281 KB 1 files, 1106 B

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


Extract - C:\TMW SCL\sdnpv300.zip

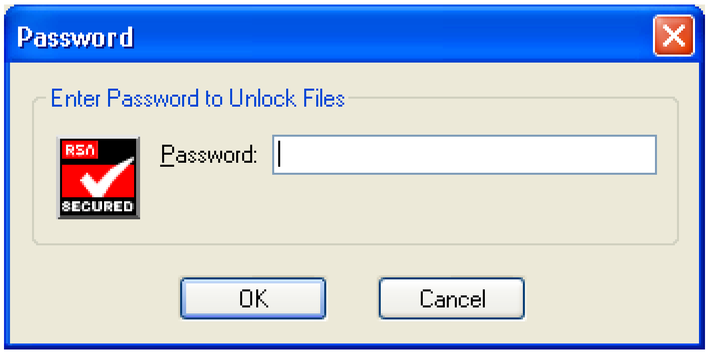
Extract: [✓ All files] Selected files

Statistics
Files to extract: 158
Bytes to extract: 3,360,744

Extract to
Folder: C:\TMW SCL

Options (after extracting)
Open folder ☐
Delete files when closing archive ☐
Run virus scan ☐
Run install program (SETUP.EXE) ☐

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


Password

Enter Password to Unlock Files

Password:

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.

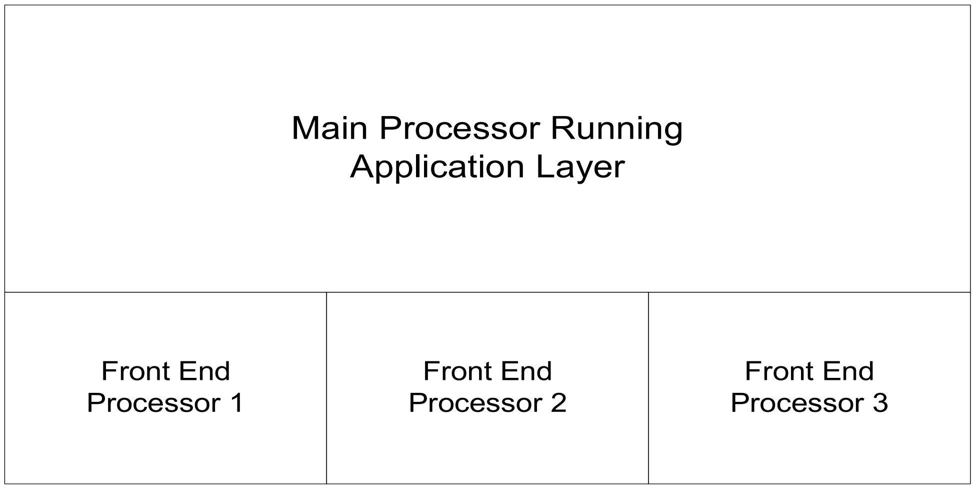


Figure 6: Example of a Distributed Architecture

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.

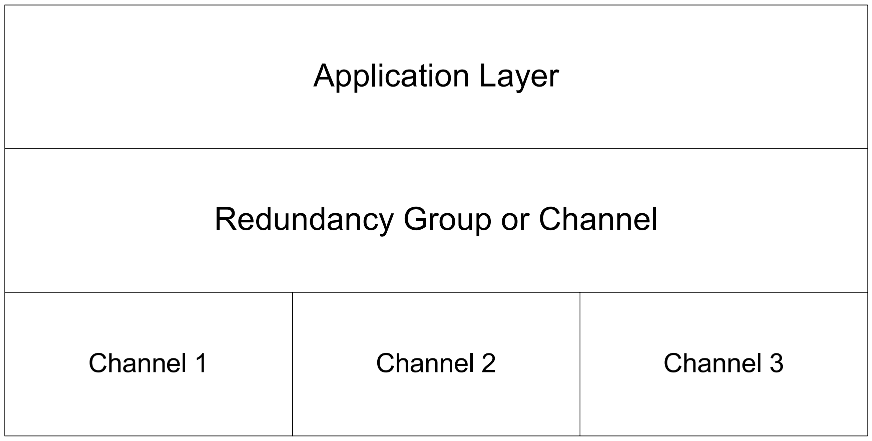


Figure 7: 60870-5-104 Redundancy Software Architecture

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.

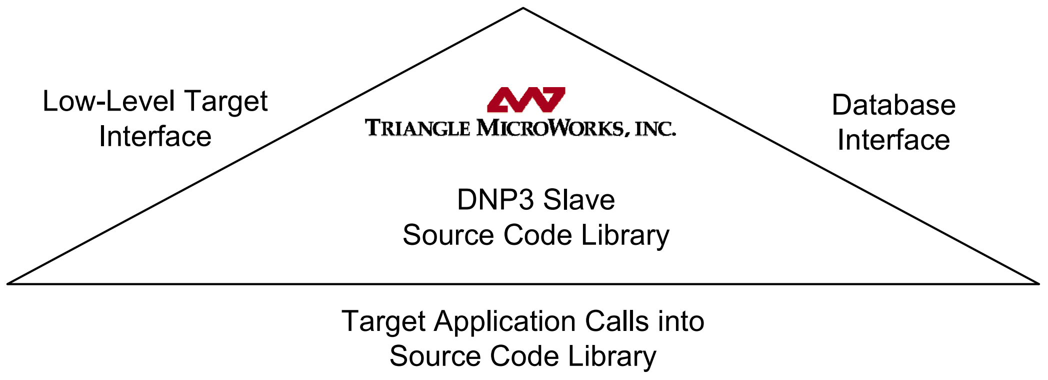
The scripts can be run in “attended” or “unattended” mode. The attended mode is required to verify complete device performance. The unattended mode skips steps that require specific configuration or values from the device. This mode is useful in automated regression tests.

1 DNP3 Slave

This section provides detailed information specific to a DNP3 Slave 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 Slave 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.


Low-Level Target Interface
TRIANGLE MICROWORKS, INC.
Database Interface
DNP3 Slave Source Code Library
Target Application Calls into Source Code Library

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, as well as the sdnpo*nnn*.h header files (where *nnn* represents a 3-digit DNP3 object group number).

The sdnpo*nnn*.h header files are not listed in this section. The xxx_addEvent function in each of these files can be called to insert events of that data type into the event buffer.

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

	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`
	`sdnpsesn_initConfig`	`sdnpsesn.h`
	`sdnpsesn_openSession`	`sdnpsesn.h`
PT	`tmwpltmr_checkTimer`	`tmwpltmr.h`
PI	`tmwappl_checkForInput`	`tmwappl.h`
E	sdnpo*nnn*_addEvent	sdnpo*nnn*.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
E	These functions are only used for an event-driven data implementation. Optionally, the SCL can poll the target for data changes, in which case these functions are not called by the target application. See Section 5.1.2 of the SCL User Manual for more information on handling events.
nnn	Represents a 3-digit Object number (e.g., sdnpo002_addEvent for binary input events)

1.1.2 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 2 summarizes the callback functions that allow the target implementation to call into the Source Code Library.

Table 2. Target Application calls into Source Code Library

	Callback Function	Function Name	Header File	Notes
E	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)
E	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:

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

1.2 Calls from the Source Code Library into the Target Application

1.2.1 Low-level Target Interface

Table 3 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 5.

Table 3. 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 SDNP Database Interface

Table 4 lists the database interface functions for DNP3 outstation 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 sdnpdata.h. The implementation should be added in sdnpdata.c in the blocks labeled

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

Table 4. Database Interface Functions

	Function Name	Header File
	sdnpdata_getIIN	sdnpdata.h
	sdnpdata_IINQuantity	sdnpdata.h
	sdnpdata_IINRead	sdnpdata.h
	sdnpdata_coldRestart	sdnpdata.h
	sdnpdata_warmRestart	sdnpdata.h
	sdnpdata_unsolEventMask	sdnpdata.h
	sdnpdata_eventAndStaticRead	sdnpdata.h
	sdnpdata_init	sdnpdata.h
	sdnpdata_close	sdnpdata.h
	sdnpdata_setTime	sdnpdata.h
X	sdnpdata_xxxGetDescription	sdnpdata.h
	sdnpdata_xxxQuantity	sdnpdata.h
	sdnpdata_xxxGetPoint	sdnpdata.h
V	sdnpdata_xxxDefVariation	sdnpdata.h
	sdnpdata_xxxEventClass	sdnpdata.h
V	sdnpdata_xxxEventDefVariation	sdnpdata.h
	sdnpdata_xxxAssignClass	sdnpdata.h
	sdnpdata_xxxRead	sdnpdata.h
P	sdnpdata_xxxChanged	sdnpdata.h
X	sdnpdata_xxxGetControlMask	sdnpdata.h
	sdnpdata_xxxSelect	sdnpdata.h
	sdnpdata_xxxOperate	sdnpdata.h
	sdnpdata_binOutSelPatternMask	sdnpdata.h
	sdnpdata_binOutOpPatternMask	sdnpdata.h
	sdnpdata_binCntrFreeze	sdnpdata.h
	sdnpdata_anlgInDbandWrite	sdnpdata.h
F	sdnpdata_getFileInfo	sdnpdata.h
F	sdnpdata_readFileInfo	sdnpdata.h
F	sdnpdata_getAuthentication	sdnpdata.h
F	sdnpdata_deleteFile	sdnpdata.h
F	sdnpdata_openFile	sdnpdata.h
F	sdnpdata_closeFile	sdnpdata.h
F	sdnpdata_readFile	sdnpdata.h
F	sdnpdata_writeFile	sdnpdata.h
U	sdnpdata_umEventAdd	sdnpdata.h
U	sdnpdata_umEventNotSentCount	sdnpdata.h
U	sdnpdata_umEventGet	sdnpdata.h
U	sdnpdata_umEventSent	sdnpdata.h
U	sdnpdata_umEventNotSent	sdnpdata.h
U	sdnpdata_umEventRemove	sdnpdata.h

KEY:

F	These functions are only required if
P	These functions are only used for a Polled Event implementation
U	These functions are only required if User-Managed Events are supported (e.g., to store events in non-volatile memory)
V	These functions are only used if configured for per-point default variation
X	These functions are only required to support the generation of an XML document from the target database
xxx	Represents one or more of: **binIn, binOut, binCntr, frznCntr, anlgIn, anlgInDBand,anlgOut, dblin, str, vterm, file**

	Function Name	Header File
	sdnpdeviceAttryyy	sdnpdata.h
D	sdnpdata_datasetyyy	sdnpdata.h
D	sdnpdata_datasetDescryyy	sdnpdata.h
D	sdnpdata_datasetProtoyyy	sdnpdata.h
	sdnpdata_activateConfig	sdnpdata.h
A	sdnpdata_authErrorEventClass	sdnpdata.h
A	sdnpdata_authIsCriticalReq	sdnpdata.h
A	sdnpdata_authDecryptKeyWrapData	sdnpdata.h
A	sdnpdata_authHMACSupport	sdnpdata.h
A	sdnpdata_authHMACValue	sdnpdata.h
A	sdnpdata_authRandomChallengeData	sdnpdata.h
A	sdnpdata_authLogyyy	Sdnpdata.h

KEY:
D These functions are only required if Data Sets are supported
A These functions are only required to if Secure Authentication is supported
xxx Represents one or more of: binIn, binOut, binCntr, frznCntr, anlgIn, anlgInDBand,anlgOut, dblin, str, vterm, file

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 5** summarizes these callback functions.

Table 5. Callback Functions into Target Applications

	Callback Function	Function Name	Header File	Notes
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	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 SDNPSESN_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 Slave Example

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

```
/*****************************************************************************/
/* 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.                                            */
/*                                                                           */
/*****************************************************************************/

// DNPSlave.cpp : Sample DNP Slave console application.
//

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

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

#include "tmwscl/dnp/dnpchnl.h"
#include "tmwscl/dnp/sdnpsesn.h"
#if TMWCNFG_USE_SIMULATED_DB
#include "tmwscl/dnp/sdnpsim.h"
#endif
#include "tmwscl/dnp/sdnpo032.h"
#include "tmwscl/dnp/sdnputil.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 the Authority/Master using a remote key change method for
 * user number 1 "Common".
 */
TMWTYPES_BOOL     noUserAtStartup = TMWDEFS_FALSE;
#endif



#if !TMWCNFG_MULTIPLE_TIMER_QS
/* forward references */
void myPutDiagString(const TMWDIAG_ANLZ_ID *pAnlzId, const TMWTYPES_CHAR *pString);

#if DNPCNFG_SUPPORT_AUTHENTICATION
void              myInitSecureAuthentication(SDNPSESN_CONFIG *pSesnConfig);
TMWTYPES_BOOL     myAddAuthUsers(SDNPSESN *pSDNPSession,
                              TMWTYPES_BOOL operateInV2Mode);

/* 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.
 */
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 entry point */
int main(int argc, char* argv[])
{
  TMWAPPL *pApplContext;
  TMWCHNL *pSclChannel;
  TMWSESN *pSclSession;
  DNPCHNL_CONFIG DNPConfig;
  DNPTPRT_CONFIG tprtConfig;
  TMWTARG_CONFIG targConfig;
  TMWPHYS_CONFIG physConfig;
  DNPLINK_CONFIG linkConfig;
  SDNPSESN_CONFIG sesnConfig;
  TMWTARGIO_CONFIG IOCnfg;
  bool useSerial = false;
  bool udpOnly = false;
  int addEventCtr = 0;
  int anlgInPointNum = 0;

  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 WinIOTarg 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 SDNP SCL. This includes:
   *  - initialize polled timer
   *  - initialize application context
   */
  tmwtimer_initialize();
  pApplContext = tmwappl_initApplication();

  /* Initialize channel configuration to defaults */
  tmwtarg_initConfig(&targConfig);
  dnpchnl_initConfig(&DNPConfig, &tprtConfig, &linkConfig, &physConfig);
  linkConfig.networkType = DNPLINK_NETWORK_TCP_UDP;

  /* Initialize IO Config Structure
   * Call tmwtargio_initConfig to initialize default values, then overwrite
   * specific values as needed.
   *
   * This example configures the Slave for a TCP/IP session using the
   * Loopback address and default Session values. This configuration
   * is compatible with the example scripts that ship with the
   * Communication Protocol Test Harness.
   */
  tmwtargio_initConfig(&IOCnfg);


  if(useSerial)
  {
    IOCnfg.type = TMWTARGIO_TYPE_232;

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

    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, "COM2");
#endif

#if defined(TMW_LINUX_TARGET)
    /* port to open */
    strcpy(IOCnfg.targ232.portName, "/dev/ttyS1");
#endif
  }
  else
  {
    IOCnfg.type = TMWTARGIO_TYPE_TCP;

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

    /* *.*.*.* allows any client to connect*/
    strcpy(IOCnfg.targTCP.ipAddress, "*.*.*.*");

    /* Allow IPV4 Loopback address only to connect */
    //strcpy(IOCnfg.targTCP.ipAddress, "127.0.0.1");

    /* Allow only client from this IP Address to connect */
    //strcpy(IOCnfg.targTCP.ipAddress, "192.168.56.1");
    //strcpy(IOCnfg.targTCP.ipAddress, "192.168.1.117");

    /* 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");

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

    /* *.*.*.* allows any client to connect */
    strcpy(IOCnfg.targTCP.ipAddress, "::");
    /* You can still use *.*.*.* to allow any even though not IPV6) */
    //strcpy(IOCnfg.targTCP.ipAddress, "*.*.*.*");

    /* Allow IPV6 Loopback address to connect */
    //strcpy(IOCnfg.targTCP.ipAddress, "::1");
    /* Allow only client from this IPV6 Address to connect */
    //strcpy(IOCnfg.targTCP.ipAddress, "2600:1700:850:8690:28f3:d72b:f609:c41a");
    //strcpy(IOCnfg.targTCP.ipAddress, "2600:1700:850:8690:b01a:c4fd:2cf0:45e6");
    //strcpy(IOCnfg.targTCP.ipAddress, "2600:1700:850:8690:dcb:c4b7:b101:9a48");

    /* You can also specify a NIC and/or local IPV6 Address */
    //strcpy(IOCnfg.targTCP.nicName, "enp0s8");
    //strcpy(IOCnfg.targTCP.localIpAddress,
    //       "2600:1700:850:8690:3035:bc19:378f:f1bd");
    //strcpy(IOCnfg.targTCP.localIpAddress,
    //       "2600:1700:850:8690:64b5:ad53:9dbd:28ff");
    strcpy(IOCnfg.targTCP.localIpAddress, "::");
#endif

    /* port to listen on */
    IOCnfg.targTCP.ipPort = 20000;
    IOCnfg.targTCP.polledMode = USE_POLLED_MODE;

    /* listen, do not initiate the connection */
    IOCnfg.targTCP.mode = TMWTARGTCP_MODE_SERVER;

    /* There are certain rules for an outstation in IP Networking */
    IOCnfg.targTCP.role = TMWTARGTCP_ROLE_OUTSTATION;
    IOCnfg.targTCP.disconnectOnNewSyn = TMWDEFS_FALSE;

#if TMWTARG_SUPPORT_UDP
    IOCnfg.targTCP.localUDPPort = 20000; /* TMWTARG_UDP_PORT_NONE */
    IOCnfg.targTCP.validateUDPAddress = TMWDEFS_TRUE;
#endif
    /* TLS Configuration */
    IOCnfg.targTCP.useTLS = USE_TLS;
#if TMWTARG_SUPPORT_TLS
    if (USE_TLS)
    {
      /* 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"
        */

      /* The default TMW configuration sets up the client's certificate files.
        * Because the file names are the same length but the values for
        * Linux and Windows differ, this sample application simply updates
        * the path to the server's certificates.
        */
      char *clientPtr;
      char serverString[] = "server";
      clientPtr = strstr(IOCnfg.targTCP.tlsRsaCertificateId, "client");
      if (clientPtr)
      {
        /* Copy without NULL terminator */
        memcpy(clientPtr, serverString, sizeof(serverString)-1);
      }
      clientPtr = strstr(IOCnfg.targTCP.tlsRsaPrivateKeyFile, "client");
      if (clientPtr)
      {
        /* Copy without NULL terminator */
        memcpy(clientPtr, serverString, sizeof(serverString)-1);
      }
    }
#endif

    /* Some other outstation configurations */
    bool dualEndPoint = false;
    if(dualEndPoint)
    {
      /* This code will configure the channel for Dual End Point mode
        * ie. Client AND Server
        * It will both listen and try to connect to the remote end when
        * it has data to send. This mode should be used on both ends.
        */
      IOCnfg.targTCP.mode = TMWTARGTCP_MODE_DUAL_ENDPOINT;
```

```c
      /* if dual end point, the ip address of the master must be specified */
      //strcpy(IOCnfg.targTCP.ipAddress, "127.0.0.1");
      strcpy(IOCnfg.targTCP.ipAddress, "192.168.56.1");
#if USE_IPV6
      strcpy(IOCnfg.targTCP.ipAddress, "2600:1700:850:8690:b01a:c4fd:2cf0:45e6");
#endif

      /* This is the port to send connect request to when
       * the outstation initiates the connection
       */
      IOCnfg.targTCP.dualEndPointIpPort = 20000;
    }
    else if(udpOnly)
    {
      /* This code will configure the channel for UDP only. */

      IOCnfg.targTCP.mode = TMWTARGTCP_MODE_UDP;
      linkConfig.networkType = DNPLINK_NETWORK_UDP_ONLY;

      /* if UDP ONLY, the IP address of the master must be specified */
      //strcpy(IOCnfg.targTCP.ipAddress, "127.0.0.1");
      strcpy(IOCnfg.targTCP.ipAddress, "192.168.56.1");
#if USE_IPV6
      strcpy(IOCnfg.targTCP.ipAddress, "2600:1700:850:8690:b01a:c4fd:2cf0:45e6");
#endif


#if TMWTARG_SUPPORT_UDP      /* Choose the local UDP port to send and receive on */
      IOCnfg.targTCP.localUDPPort = 20000;

      /* Send responses to the port the master sends from.
       * Normally this would be configured to use 20000.
       */
      IOCnfg.targTCP.destUDPPort =  TMWTARG_UDP_PORT_SRC; //20000;

      /* Send the initial Null unsolicited response to this port on the master */
#if defined(TMW_WTK_TARGET)
      IOCnfg.targTCP.initUnsolUDPPort = 20001;
#endif
#if defined(TMW_LINUX_TARGET)
      IOCnfg.targTCP.initUnsolUDPPort = 20000;
#endif
#endif
    }
  }

  /* Open DNP channel */
  pSclChannel = dnpchnl_openChannel(pApplContext, &DNPConfig, &tprtConfig,
    &linkConfig, &physConfig, &IOCnfg, &targConfig);

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

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

  /* Initialize and open DNP slave session */
  sdnpsesn_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;

  /* If Secure Authentication is to be enabled */
  if(myUseAuthentication)
    myInitSecureAuthentication(&sesnConfig);
#endif

  pSclSession = (TMWSESN *)sdnpsesn_openSession(pSclChannel, &sesnConfig,
    (void*)1);
  if(pSclSession == TMWDEFS_NULL)
  {
    /* Failed to open */
    printf("Failed to open session, exiting program \n");

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

  if(pSclSession == TMWDEFS_NULL)
  {
    /* Failed to open session */
    printf("Failed to open Session, exiting program \n");

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

#if DNPCNFG_SUPPORT_AUTHENTICATION
  if (myUseAuthentication)
  {
    if (!myAddAuthUsers((SDNPSESN*)pSclSession,
                        sesnConfig.authConfig.operateInV2Mode))
    {
      /* Sleep for 10 seconds before exiting */
      tmwtarg_sleep(10000);
      return (1);
    }
  }
#endif

  /* This code will open a second channel and outstation session */
  if(openSecondConnection)
  {
    TMWCHNL *pSclChannel2;
    TMWSESN *pSclSession2;
    strcpy(IOCnfg.targTCP.chnlName, "Second");
    IOCnfg.targTCP.ipPort = 20001;
#if TMWTARG_SUPPORT_UDP
    IOCnfg.targTCP.localUDPPort = 20001;
#endif
    IOCnfg.targTCP.polledMode = USE_POLLED_MODE;

    /* Open Second DNP channel */
    pSclChannel2 = dnpchnl_openChannel(pApplContext, &DNPConfig, &tprtConfig,
                                       &linkConfig, &physConfig, &IOCnfg,
                                       &targConfig);

    if(pSclChannel2 == TMWDEFS_NULL)
    {
      /* Failed to open */
      printf("Failed to open second channel, exiting program \n");

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

    pSclSession2 = (TMWSESN *)sdnpsesn_openSession(pSclChannel2, &sesnConfig,
      (void*)2);

    if(pSclSession2 == TMWDEFS_NULL)
    {
      /* Failed to open */
      printf("Failed to open second session, exiting program \n");

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

#if DNPCNFG_SUPPORT_AUTHENTICATION
    if (myUseAuthentication)
    {
      if (!myAddAuthUsers((SDNPSESN*)pSclSession2,
                          sesnConfig.authConfig.operateInV2Mode))
      {
        /* Sleep for 10 seconds before exiting */
        tmwtarg_sleep(10000);
        return (1);
      }
    }
#endif
  }


  /* Begin the main loop.
   * This example uses the Source Code Library in Polled mode.
   * In Polled mode, you must periodically:
   *  - check the timer
   *  - check for (and process) any received data
   */
  while(1)
  {
    tmwpltmr_checkTimer();

    /* 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);
    }

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

    /* Begin addEvent example.
     *  This code demonstrates how to use the xxx_addEvent function to
     *  inform the library that a data change has occurred instead of
     *  relying on SCL scanning.
     *  The period (once every 100 iterations) and the point count (10)
     *  are arbitrary, but demonstrate how the 1st 10 analog values are
     *  updated periodially and the slave will send unsolicited updates
     *  to the master if enabled to do so.
     * NOTE: Setting analogInputScanPeriod to a non-zero value will result
     *       in this value being updated by the SCL scan.
     */
    addEventCtr++;
    if ((addEventCtr % 100) == 0)
    {
      TMWTYPES_ANALOG_VALUE analogValue;
      TMWDTIME timeStamp;
      sdnputil_getDateTime(pSclSession, &timeStamp);
      analogValue.value.dval = rand();
      analogValue.type = TMWTYPES_ANALOG_TYPE_DOUBLE;

      sdnpo032_addEvent(pSclSession, anlgInPointNum,
        &analogValue, DNPDEFS_DBAS_FLAG_ON_LINE,
        &timeStamp);
      anlgInPointNum++;
      if (anlgInPointNum == 10)
      {
        anlgInPointNum = 0;
      }
    }
    /* End addEvent example. */
  }

  return 0;
}


#if DNPCNFG_SUPPORT_AUTHENTICATION
void myInitSecureAuthentication(SDNPSESN_CONFIG *pSesnConfig)
{
  /* set this to TMWDEFS_TRUE to use SAv2 implementation */
  TMWTYPES_BOOL myUseSAv2 = TMWDEFS_FALSE;

#if !SDNPCNFG_SUPPORT_SA_VERSION5
  /* If SAv5 is not supported, use SAv2 */
  myUseSAv2 = TMWDEFS_TRUE;
#endif

  printf( "Using SAv%c\n", myUseSAv2?'2':'5');

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

```
    /* NOTE: Secure Authentication Version 2 (SAv2) will not function properly without
     * implementing the functions sdnpdata_authxxx() in sdnpdata.c
     * SAv5 also requires utils/tmwcrypto which uses OpenSSL as a
     * sample implementation.
     */

    /* For SAv2 configure the same user numbers and update keys for each user
     * number on both master and outstation devices. These must be configured before
     * the session is opened.
     * SAv5 allows User Update Keys to also be sent to the outstation over DNP.
     */

  #if SDNPCNFG_SUPPORT_SA_VERSION2
    /* Use Secure Authentication Version 2 */
    if(myUseSAv2)
    {
      pSesnConfig->authConfig.operateInV2Mode = TMWDEFS_TRUE;
      pSesnConfig->authConfig.maxErrorCount = 2;

      /* Configure user numbers */
      /* Spec says default user number 1 provides a user number
       * for the device or "any" user
       */
      pSesnConfig->authConfig.authUsers[0].userNumber = DNPAUTH_DEFAULT_USERNUMBER;
    }
  #endif
    /* Example configuration. Some of these may be the default values,
     * but are shown here as an example of what can be set.
     */
    pSesnConfig->authConfig.extraDiags = TMWDEFS_TRUE;
    pSesnConfig->authConfig.aggressiveModeSupport = TMWDEFS_TRUE;
    pSesnConfig->authConfig.maxKeyChangeCount = 1500;
    // 120 seconds is very short for demonstration purposes only.
    // The DNP Master should be configured for 60 seconds if this is uncommented.
    //pSesnConfig->authConfig.keyChangeInterval = 120000;
    pSesnConfig->authConfig.assocId = 0;
  }

  TMWTYPES_BOOL myAddAuthUsers(SDNPSESN *pSDNPSession, TMWTYPES_BOOL operateInV2Mode)
  {

  #if SDNPCNFG_SUPPORT_SA_VERSION2 && TMWCNFG_SUPPORT_CRYPTO
    /* If SAv2 and using optional TMWCRYPTO interface set the crypto database*/
    if (operateInV2Mode)
    {
      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.
       * For SAv2 the handle used to look up keys in the sim database
       *  is the SDNP DB handle.
       */
      /* This used to use pSDNPSession->pDbHandle before v3.30, but sdnpdata.c now
uses the crypto handle as required */
      if (!tmwcrypto_configSimKey(pSDNPSession->dnp.pCryptoHandle,
TMWCRYPTO_USER_UPDATE_KEY, userNumber, defaultUserKey1, 16, TMWDEFS_NULL, 0))
      {
        /* 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.
       */
    }
  #endif
  #if SDNPCNFG_SUPPORT_SA_VERSION5
    /* In SAv5 we no longer use an array of user configuration, instead you can
     * add one user at a time. The DNP3 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.
       */
      if (!operateInV2Mode)
      {
        const TMWTYPES_CHAR *pKey;
        TMWTYPES_USHORT keyLength;
        TMWTYPES_USHORT userNumber = 1;

        /* It is possible to start an SAV5 session with no users and have them added
         * from the Authority/Master using a remote key change mehthod for
         * user number 1 "Common"
         */
        if (!noUserAtStartup)
        {
          /* 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.
           */
          if (!tmwcrypto_configSimKey(pSDNPSession->dnp.pCryptoHandle,
                                      TMWCRYPTO_USER_UPDATE_KEY, userNumber,
                                      defaultUserKey1, 16, TMWDEFS_NULL, 0))
          {
            /* Failed to add key */
            printf("Failed to add key, exiting program \n");
            return (TMWDEFS_FALSE);
          }

          /* For SAv5 add the user to the sdnp library */
          sdnpsesn_addAuthUser((TMWSESN*)pSDNPSession, userNumber);

          /* Also add this to the SDNP simulated database to support optional
           * remote key change methods
           */
          sdnpsim_authConfigUser(pSDNPSession->pDbHandle, (TMWTYPES_CHAR*)"Common", 6,
1, DNPAUTH_USER_ROLE_SINGLEUSER, 100);


          /* 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.
           */
        }

        /* 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 sdnpsim database
         * OutstationName = "SDNP Outstation";
         */

        /* If using simulated database in tmwcrypto, configure 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 need to call tmwcrypto_configSimKey
         * to add it to your own database. This key is used by the Central Authority
         * and configured on the Outstation.
         */
        if (!tmwcrypto_configSimKey(pSDNPSession->dnp.pCryptoHandle,
TMWCRYPTO_AUTH_CERT_SYM_KEY, 0, (TMWTYPES_UCHAR *)&authoritySymCertKey, 32, TMWDEFS_NULL,
0))
        {
          /* Failed to add key */
          printf("Failed to add key, exiting program \n");
          return (TMWDEFS_FALSE);
        }

        /* If using simulated database in tmwcrypto, configure Outstation Private Key
         * when Asymmetric Key Update is supported. 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.
         */
        pKey = "TMWTestOSRsa2048PrvKey.pem";
        keyLength = (TMWTYPES_USHORT)strlen(pKey);
        if (!tmwcrypto_configSimKey(pSDNPSession->dnp.pCryptoHandle,
TMWCRYPTO_OS_ASYM_PRV_KEY, 0, (TMWTYPES_UCHAR *)pKey, keyLength, (TMWTYPES_UCHAR
*)"triangle", 8))
        {
          /* Failed to add key */
          printf("Failed to add key, exiting program \n");
          return (TMWDEFS_FALSE);
        }

        /* If using simulated database in tmwcrypto, configure Authority Public Key
         * when Asymmetric Key Update is supported. 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.
         */
        /* One symmetric and one asymmetric key change algorithm is configured
         * to work at a time.
         */
        TMWTYPES_UCHAR keyChangeMethod = DNPAUTH_KEYCH_ASYM_RSA2048_SHA256;
        switch (keyChangeMethod)
         {
         case DNPAUTH_KEYCH_ASYM_RSA1024_SHA1:
           pKey = "TMWTestAuthorityDsa1024PubKey.pem";
           break;

         case DNPAUTH_KEYCH_ASYM_RSA2048_SHA256:
           pKey = "TMWTestAuthorityDsa2048PubKey.pem";
           break;

         case DNPAUTH_KEYCH_ASYM_RSA3072_SHA256:
           pKey = "TMWTestAuthorityDsa3072PubKey.pem";
           break;

           // New key change methods from TB2016-002
         case DNPAUTH_KEYCH_ASYM_RSA1024_RSA_SHA1:
           pKey = "TMWTestAuthorityRsa1024PubKey.pem";
           break;

         case DNPAUTH_KEYCH_ASYM_RSA2048_RSA_SHA256:
           pKey = "TMWTestAuthorityRsa2048PubKey.pem";
           break;

         case DNPAUTH_KEYCH_ASYM_RSA3072_RSA_SHA256:
           pKey = "TMWTestAuthorityRsa3072PubKey.pem";
           break;

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

        keyLength = (TMWTYPES_USHORT)strlen(pKey);
        if (!tmwcrypto_configSimKey(pSDNPSession->dnp.pCryptoHandle,
          TMWCRYPTO_AUTH_ASYM_PUB_KEY, 0, (TMWTYPES_UCHAR *)pKey, keyLength,
          TMWDEFS_NULL, 0))
        {
          /* Failed to add key */
          printf("Failed to add key, exiting program \n");
          return (TMWDEFS_FALSE);
        }

        /* add security statistics points to database */
        for (int i = 0; i < DNPAUTH_NUMBER_STATISTICS; i++)
        {
          /* spec says statistics SHALL be in an event class */
          sdnpsim_addAuthSecStat(pSDNPSession->pDbHandle, i, TMWDEFS_CLASS_MASK_THREE,
            0x01, 0);
        }
      }
    #endif
      return (TMWDEFS_TRUE);
    }

    #endif

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

    #if TMWCNFG_SUPPORT_DIAG
    /* 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;

  if((TMWDIAG_ID_ERROR & id)
    ||(TMWDIAG_ID_TARGET & id)
    ||(TMWDIAG_ID_APPL & id)
    ||(TMWDIAG_ID_USER & id)
    ||(TMWDIAG_ID_SECURITY_DATA & id)
    ||(TMWDIAG_ID_SECURITY_HDRS & id)
    )
  {
    printf( "%s", (char *)pString);
    return;
  }

  /* Comment this out to turn off verbose diagnostics */
  /* For now print everything */
  /* printf((char *)pString); */
}
#endif
#endif
```

3 Configuration Files

3.1 Default Functionality

The SDNP 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.

SDNPDATA_CNFG_LEVEL4	Listed here
SDNPDATA_CNFG_LEVEL_TMW	Listed here
SDNPDATA_SUPPORT_READ_XML	
Floating Point Variations	
Double Bit Inputs and Events	Object Groups 3 and 4
Binary Output Events	Object Group 11
Binary Output Pattern Mask	Object Group 12 Var 2 and 3
Binary Output Command Events	Object Group 13
Binary Counter Events with Time	Object Group 22 Var 5 and 6
Frozen Counter Events with Time	Object Group 23 Var 5 and 6
Analog Input Float	Object Group 30 Var 5 and 6
Analog Input Events Time & Float	Object Group 32 Var 3,4,5,6,7 and 8
Analog Input Deadband	Object Group 34
Analog Output Status Float	Object Group 40 Var 3 and 4
Analog Output Control Block Float	Object Group 41 Var 3 and 4
Analog Output Events	Object Group 42
Analog Output Command Events	Object Group 43
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 by default.

Device Attributes	Object Group 0
Frozen Analog Inputs and Events	Object Groups 31 and 33
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
SDNPDATA_SUPPORT_OBJ120_V8	Now FALSE
SDNPDATA_SUPPORT_SELECT_CANCEL	Now FALSE
SDNPDATA_SUPPORT_IDENT_UNSOL_RETRY	Now FALSE
SDNPCNFG_MAX_CONTROL_REQUESTS	Was 255 Now 10
SDNPSESN_SELECT_BUFFER_SIZE	Was (8*1024) Now (10*13+16)

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
```

```
 * 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.
 */
#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
#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
 * 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.
 */
#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 */
#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
#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
 */
#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

```

```
/* 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
 * 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
 * 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.
 * 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
 * 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.
 * 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

/* 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.
 */
#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
 * 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

/* 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.
 */
#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

/* 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 */

/* 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

#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

#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 */

#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 DNP Configuration

‘dnpcnfg.h’ contains DNP specific configuration parameters that would apply to both Master and Slave SCLs. Typically 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: dnpcnfg.h
 * description:   DNP configuration definitions
 */
#ifndef DNPCNFG_DEFINED
#define DNPCNFG_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. These DO NOT have to depend on the
 * TMWCNFG_xxx defines, but can be changed here to set desired values.
 */

/* Specify the maximum number of DNP channels
 * The TMW SCL needs one for each DNP serial or TCP connection
 */
#define DNPCNFG_NUMALLOC_CHANNELS             TMWCNFG_MAX_CHANNELS

/* Specify the number of link layer frames buffers that can be allocated.
 * The TMW SCL needs one per DNP channel if Diagnostics are compiled in.
 */
#define DNPCNFG_NUMALLOC_LINK_FRAMES          TMWCNFG_MAX_CHANNELS

/* Specify the number of link layer contexts that can be allocated.
 * The TMW SCL needs one per DNP channel.
 */
#define DNPCNFG_NUMALLOC_LINK_CONTEXTS        TMWCNFG_MAX_CHANNELS

/* Specify the number of transport layer contexts that can be allocated.
 * The TMW SCL needs one per DNP channel.
 */
#define DNPCNFG_NUMALLOC_TPRT_CONTEXTS        TMWCNFG_MAX_CHANNELS

/* Specify the number of link layer session contexts that can be allocated.
 * The TMW SCL needs one per master or slave session.
 */
#define DNPCNFG_NUMALLOC_LINK_SESSIONS        TMWCNFG_MAX_SESSIONS

/* Specify the number of transport sessions that can be allocated.
 * The TMW SCL needs one per master or slave session.
 */
#define DNPCNFG_NUMALLOC_TPRT_SESSIONS        TMWCNFG_MAX_SESSIONS
```

```
/* Specify the number of transmit data structures that can be allocated.
 * This is dependent on how may application layer fragments will be buffered
 * per session. etc.
 *  recommend at least 1 per session for slave
 *   (2 if Identical Unsolicited Response Retries is supported)
 *  recommend 2 or more per session for master
 *   requires more if multiple requests are to be queued on master.
 *   if requests are sent automatically based on IIN bits, 1 is needed
 *   for each simultaneous request.
 */
#define DNPCNFG_NUMALLOC_CHNL_TX_DATAS        TMWCNFG_MAX_APPL_MSGS

/* Define the maximum number of bytes transmitted or received in a
 * frame. This parameter specifies the actual number of bytes on the
 * wire including all DNP3 header and control bytes. The absolute
 * maximum value for this parameter is 292 bytes which includes 249
 * bytes of application data, 1 byte for the DNP3 transport header,
 * 8 bytes for the DNP3 link header, and 34 bytes of CRC data. A
 * theoretical minimum would be 14 which would only include 1 byte
 * of application data. A realistic minimum would be 28 which would
 * include 15 bytes of application data.
 *
 * If dynamic memory is supported this parameter is only used as a
 * default value for the appropriate transport and link layer config-
 * uration parameters. The actual value can be changed by the user to
 * a value larger or smaller than this value. If dynamic memory alloca-
 * tion is not supported this parameter is used to statically allocate
 * buffers and as the default for the configuration parameters. In this
 * case the user can change the configured values to smaller values but
 * not larger values.
 */
#define DNPCNFG_MAX_TX_FRAME_LENGTH 292
#define DNPCNFG_MAX_RX_FRAME_LENGTH 292

/* Define the maximum number of application layer data bytes in a
 * transport layer fragment. The DNP3 specification recommends a value
 * of 2048 and requires that devices that support a larger fragment
 * size be able to configure it down to no more than 2048 bytes.
 *
 * Typically slave devices can configure the received fragment size
 * much smaller than 2048 since they will only receive requests and
 * similarly the master can configure a transmit fragment size to
 * a smaller value since it will typically only transmit requests.
 *
 * If dynamic memory is supported this parameter is only used as a
 * default value for the appropriate transport and link layer config-
 * uration parameters. The actual value can be changed by the user to
 * a value larger or smaller than this value. If dynamic memory alloca-
 * tion is not supported this parameter is used to statically allocate
 * buffers and as the default for the configuration parameters. In this
 * case the user can change the configured values to smaller values but
 * not larger values.
 */
#ifndef DNPCNFG_MAX_TX_FRAGMENT_LENGTH
 /* Maximum allowed value of 65535 */
#define DNPCNFG_MAX_TX_FRAGMENT_LENGTH 2048
#endif
#ifndef DNPCNFG_MAX_RX_FRAGMENT_LENGTH
/* Maximum allowed value of 65535 */
#define DNPCNFG_MAX_RX_FRAGMENT_LENGTH 2048
#endif

/* Define maximum number of hex bytes to display in diagnostics output */
#define DNPCNFG_MAX_DIAG_HEX_BYTES 4000

/* Define maximum number of bytes in a filename
 * This is used for file transfer if Object 70 is supported
 */
#define DNPCNFG_MAX_FILENAME 256

/* Define maximum number of files that can be opened simultaniously
 * This is used for file transfer if Object 70 is supported
 */
#define DNPCNFG_MAX_FILES_OPEN 5

/* Define maximum number of bytes in an extended string
 * This is used if Object 114 is supported. Object 114 supports string
 * sizes between 0 and 65535. Since the maximum string length that can be
 * transmitted is limited by the fragment size, this parameter can be used
 * to save significant memory resources.
```

```
  */
#ifndef DNPCNFG_MAX_EXT_STRING_LENGTH
#define DNPCNFG_MAX_EXT_STRING_LENGTH 2048
#endif

/* Define maximum number of bytes displayed by the diagnostic functions for string
 * based objects. This includes object groups 110, 112, & 114. Setting this value
 * above 255 ensures that only the extended strings will be truncated.
 */
#define DNPCNFG_MAX_DIAG_STRING_LENGTH 255

/* Indicates whether or not functions used by Datasets and Device Attributes
 * should be compiled into the Master and Slave SCLs. Setting to TMWDEFS_FALSE
 * will save some code space.
 */
#define DNPCNFG_SUPPORT_DATASETS_ATTRS   TMWDEFS_TRUE

/* When the SCL parses a Data Set object 87 or 88 it will use a corresponding
 * Data Set descriptor and any contained prototypes to convert the data to its
 * proper type. The data types will be read onto the stack in the SCL. This
 * define controls how much memory is set aside to contain these. It requires
 * 1 byte for each of these
 */
#ifndef DNPCNFG_MAX_DATASET_DESCR_ELEMS
#define DNPCNFG_MAX_DATASET_DESCR_ELEMS 32
#endif

/* Define maximum number of control values in a dataset control group
 * (number of values per control status byte). This is used to determine how much
 * room to allow on the stack for accumulating values before sending control to
 * database. Each one is sizeof(DNPDATA_DATASET_CTRL_VALUE) which depends on other
 * defines, but by default is about 32 bytes.
 */
#ifndef DNPCNFG_MAX_DATASET_CTRLS
#define DNPCNFG_MAX_DATASET_CTRLS 8
#endif

/* This will determine how large the string array is in the DNPDATA_DATASET_VALUE
 * structure. Since value is a union, making this larger than TMWDTIME (16 bytes)
 * makes the union larger. If strings are too large to fit in the array they can
 * be passed using a pointer, but then the strings must be copied somewhere.
 */
#define DNPCNFG_MAX_STRING_ARRAY  16

/* Set this parameter to TMWDEFS_TRUE to include support for
 * configuring using binary version of device profile current values
 * Setting this to TMWDEFS_FALSE will save code space if
 * this feature is not being supported
 */
#ifndef DNPCNFG_SUPPORT_BINCONFIG
#define DNPCNFG_SUPPORT_BINCONFIG        TMWDEFS_FALSE
#endif

/* This will include code required for DNP3 Secure Authentication
 * Setting this to TMWDEFS_FALSE will save code and data space if secure
 * authentication is not being supported
 */
#ifndef DNPCNFG_SUPPORT_AUTHENTICATION
#define DNPCNFG_SUPPORT_AUTHENTICATION   TMWDEFS_FALSE
#endif

/* Specify maximum number of secure authentication user numbers supported
 * per SAv2 session.
 * SA Version 2 allocates an array of this many users per M/SDNP Session so
 * this is a hard limit per session.
 * The defines for SAv2 and SAv5 are allowed to be different (if both versions
 * are supported).
 */
#define DNPCNFG_AUTHV2_MAX_NUMBER_USERS  10

/* Specify maximum number of secure authentication user numbers supported
 * per SAv5 session.
 * Since SA Version 5 uses a linked list of allocated user structures this
 * is not actually a hard limit per session. The number of users is limited by
 * memory allocation, including whether static or limited size memory pools
 * are configured. This define IS by default used to limit the number of
 * SA State Machine events MDNPAUTH_MAX_EVENTS per MDNP session, generally
 * 1 per user, with at least 3 for a single user.
 * This define is also used to set the default values for the
 * M/SDNPCNFG_NUMALLOC_AUTH_USERS for which only apply if limited size memory
 * pools are configured.
 * The defines for SAv2 and SAv5 are allowed to be different (if both versions
 * are supported).
 */
#define DNPCNFG_AUTH_MAX_NUMBER_USERS    64

/* Support DNP Secure Authentication Specification Version 2
 * DNPCNFG_SUPPORT_AUTHENTICATION must be TMWDEFS_TRUE
 */
#ifndef DNPCNFG_SUPPORT_SA_VERSION2
#define DNPCNFG_SUPPORT_SA_VERSION2      TMWDEFS_FALSE
#endif

/* Support DNP Secure Authentication Specification Version 5
 * This is an optional component of the Source Code Library
 * DNPCNFG_SUPPORT_AUTHENTICATION must be TMWDEFS_TRUE.
 * TMWCNFG_SUPPORT_CRYPTO must be TMWDEFS_TRUE
 */
#ifndef DNPCNFG_SUPPORT_SA_VERSION5
#define DNPCNFG_SUPPORT_SA_VERSION5      TMWDEFS_FALSE
#endif

/* Secure Authentication Version 5 configuration */

/* This will include code for DNP3 Secure Authentication Remote changing of
 * User Update Keys over DNP. This optional feature was added in Version 5
 * of the DNP SA Specification
 */
#ifndef DNPCNFG_SUPPORT_AUTHKEYUPDATE
#define DNPCNFG_SUPPORT_AUTHKEYUPDATE    TMWDEFS_FALSE
#endif

/* This will include code for optional Asymmetric Key Change Methods.
 * If Authentication Remote Key Update is supported, support for
 * Symmetric Key Change Methods is required.
 */
#define DNPCNFG_SUPPORT_AUTHKEYUPDASYM   TMWCNFG_SUPPORT_CRYPTO_ASYM

/* Secure Authentication maximum name lengths */
#define DNPCNFG_AUTH_MAX_USERNAME_LENGTH 256
#define DNPCNFG_AUTH_MAX_OSNAME_LENGTH   256

/* Allow master to send requests to multiple sessions on a single channel
 * without waiting for responses from other sessions on that channel.
 * Only a single outstanding request per session (association) as required
 * by the DNP Specification.
 */
#define DNPCNFG_MULTI_SESSION_REQUESTS   TMWDEFS_FALSE

/* The following defines determine whether particular statistics are compiled in
 * All of these default to TMWDEFS_TRUE if statistics are compiled in
 * Change individual defines to TMWDEFS_FALSE to remove individual statistics
 */

/* Session Statistics */
/* support TMWSESN_STAT_REQUEST_TIMEOUT                                     */
#define DNPCNFG_SUPPORT_SSTAT_TIMEOUT          TMWCNFG_SUPPORT_STATS

/* support TMWSESN_STAT_REQUEST_FAILED                                      */
#define DNPCNFG_SUPPORT_SSTAT_FAILED           TMWCNFG_SUPPORT_STATS

/* support TMWSESN_STAT_ASDU_SENT and TMWSESN_STAT_ASDU_RECEIVED            */
#define DNPCNFG_SUPPORT_SSTAT_ASDUS            TMWCNFG_SUPPORT_STATS

/* support TMWSESN_STAT_DNPEVENT_SENT and TMWSESN_STAT_DNPEVENT_CONFIRM */
#define DNPCNFG_SUPPORT_SSTAT_EVENT            TMWCNFG_SUPPORT_STATS

/* support TMWSESN_STAT_DNPUNSOL_TIMER_START                                */
#define DNPCNFG_SUPPORT_SSTAT_UNSOL_TIMER      TMWCNFG_SUPPORT_STATS

/* support TMWSESN_STAT_EVENT_OVERFLOW                                      */
#define DNPCNFG_SUPPORT_SSTAT_OVERFLOW         TMWCNFG_SUPPORT_STATS

/* support TMWSESN_STAT_AUTHxxx                                             */
#define DNPCNFG_SUPPORT_SSTAT_AUTH             TMWCNFG_SUPPORT_STATS

/* Channel Statistics */
/* support TMWCHNL_STAT_ERROR                                               */
#define DNPCNFG_SUPPORT_CSTAT_ERRORS           TMWCNFG_SUPPORT_STATS

/* support TMWCHNL_STAT_FRAME_SENT and TMWCHNL_STAT_FRAME_RECEIVED          */
#define DNPCNFG_SUPPORT_CSTAT_FRAMES           TMWCNFG_SUPPORT_STATS
```

```
/* support TMWCHNL_STAT_FRAGMENT_SENT and TMWCHNL_STAT_FRAGMENT_RECEIVED*/
#define DNPCNFG_SUPPORT_CSTAT_FRAGS           TMWCNFG_SUPPORT_STATS


#if DNPCNFG_SUPPORT_AUTHENTICATION && !DNPCNFG_SUPPORT_SA_VERSION2 &&
!DNPCNFG_SUPPORT_SA_VERSION5
#pragma message("If DNPCNFG_SUPPORT_AUTHENTICATION is supported either
DNPCNFG_SUPPORT_SA_VERSION2 or VERSION5 must be selected")
#endif

#endif /* DNPCNFG_DEFINED */
```

3.5 SDNP Configuration

‘sdnpcnfg.h’ contains Slave DNP specific configuration parameters. Typically, 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: sdnpcnfg.h
 * description:   slave DNP configuration definitions
 */
#ifndef SDNPCNFG_DEFINED
#define SDNPCNFG_DEFINED

#include "tmwscl/utils/tmwcnfg.h"
#include "tmwscl/dnp/dnpcnfg.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. These DO NOT have to depend on the
 * TMWCNFG_xxx defines, but can be changed here to set desired values.
 */

/* Specify the maximum number of Slave DNP sessions that can be open at
 * a given time.
 */
#define SDNPCNFG_NUMALLOC_SESNS               TMWCNFG_MAX_SESSIONS

/* Specify the number of temporary ASDU buffers that can be allocated.
 * currently the TMW SCL needs one per SDNP session.
 */
#define SDNPCNFG_NUMALLOC_ASDU_BUFFERS        TMWCNFG_MAX_SESSIONS

/* For each event type specify the maximum number of events that can
 * be queued at once. This number specifies the maximum number of events
 * per data type for the entire system so it should generally be set to
 * the maximum number per session times the number of sessions.
 */
#define SDNPCNFG_NUMALLOC_OBJECT2_EVENTS      TMWCNFG_MAX_EVENTS
#define SDNPCNFG_NUMALLOC_OBJECT4_EVENTS      TMWCNFG_MAX_EVENTS
#define SDNPCNFG_NUMALLOC_OBJECT11_EVENTS     TMWCNFG_MAX_EVENTS
#define SDNPCNFG_NUMALLOC_OBJECT13_EVENTS     TMWCNFG_MAX_EVENTS
#define SDNPCNFG_NUMALLOC_OBJECT22_EVENTS     TMWCNFG_MAX_EVENTS
#define SDNPCNFG_NUMALLOC_OBJECT23_EVENTS     TMWCNFG_MAX_EVENTS
#define SDNPCNFG_NUMALLOC_OBJECT32_EVENTS     TMWCNFG_MAX_EVENTS
#define SDNPCNFG_NUMALLOC_OBJECT33_EVENTS     TMWCNFG_MAX_EVENTS
#define SDNPCNFG_NUMALLOC_OBJECT42_EVENTS     TMWCNFG_MAX_EVENTS
#define SDNPCNFG_NUMALLOC_OBJECT43_EVENTS     TMWCNFG_MAX_EVENTS
#define SDNPCNFG_NUMALLOC_OBJECT88_EVENTS     TMWCNFG_MAX_EVENTS
#define SDNPCNFG_NUMALLOC_OBJECT111_EVENTS    TMWCNFG_MAX_EVENTS
#define SDNPCNFG_NUMALLOC_OBJECT113_EVENTS    TMWCNFG_MAX_EVENTS
```

```
#define SDNPCNFG_NUMALLOC_OBJECT115_EVENTS   TMWCNFG_MAX_EVENTS
#define SDNPCNFG_NUMALLOC_OBJECT120_EVENTS   TMWCNFG_MAX_EVENTS
#define SDNPCNFG_NUMALLOC_OBJECT122_EVENTS   TMWCNFG_MAX_EVENTS

/* Maximum number of simultaneous file tranfers that can be in progress
 * which also determines the maximum number of file transfer events
 * that can be queued at any time.
 * Currently on one per session is supported.
 */
#define SDNPCNFG_NUMALLOC_OBJECT70_BLOCKS    TMWCNFG_MAX_SESSIONS

/* Determines the maximum number of controls allowed in a single request.
 * This applies to both Binary (CROB) and Analog Control outputs.
 * It is also possible to permit only a single CROB per request at runtime
 * by opening the SDNP Session with allowMultiCROBRequests set to TMWDEFS_FALSE
 */
#ifndef SDNPCNFG_MAX_CONTROL_REQUESTS
#define SDNPCNFG_MAX_CONTROL_REQUESTS          10
#endif


/* Set this parameter to the desired size of the memory buffer
 * used during generation of a device description XML file.
 * This buffer should be large enough to hold an indivisible block of
 * device description information.
 */
#define SDNPCNFG_XML_SAVE_BUF_SIZE             4096

/* Size of configuration array indicating what groups should be included
 * by a slave session in response to a read static data request.
 */
#define SDNPCNFG_MAX_NUMBER_STATIC_GROUPS      16

/* Sequential File Transfer Configuration using Object 70 Variation 2-7
 * Maximum block size for file transfer reads and writes. This will be
 * limited by maximum rx and tx fragment size but can be set smaller to
 * save memory.
 */
#define SDNPCNFG_MAX_BLOCK_SIZE                DNPCNFG_MAX_RX_FRAGMENT_LENGTH

/* Maximum number of directory entries that will be buffered before
 * sending response to a read directory request
 */
#define SDNPCNFG_OBJ70_DIR_SIZE                32

/* Number of milliseconds before retrying a sdnpdata_xxx call if that function
 * returned "ASYNC" indicating response should be sent later as an event.
 * NOTE: parameter below is used for configuring retry of directory read call.
 */
#define SDNPCNFG_OBJ70_RETRY_TIME              100

/* Number of milliseconds before retrying a sdnpdata_readFileInfo call if that
 * function returned "ASYNC" indicating response should be sent later as
 * an event. This parameter is used for read directory versus all other
 * sdnpdata_xxx calls for file transfer.
 */
#define SDNPCNFG_OBJ70_DIR_RETRY_TIME          100

/* Maximum length of Data Set Event buffer. Since it is required that a data set
 * fits inside of a Transmit Fragment this is normally set the same. However, if
 * you are limiting the size of Data Sets you could reduce this to reduce the size
 * of the data set event structure (SDNPEVNT_O088_EVENT), possibly saving memory
 * depending on your memory allocation scheme.
 */
#define SDNPCNFG_MAX_DSEVENT_LENGTH            DNPCNFG_MAX_TX_FRAGMENT_LENGTH

/* Set this to TMWDEFS_TRUE if you want to implement database code to perform
 * event queue management instead of having the SCL do this for you. This is
 * useful if events need to be kept in Non-volatile memory, or special queueing
 * algorithms are desired. If this is left at TMWDEFS_FALSE, the SCL will allocate
 * memory and completely manage the event queues.
 */
#define SDNPCNFG_USER_MANAGED_EVENTS           TMWDEFS_FALSE

/* Specify the number of slave DNP simulated databases that can be allocated.
 * The TMW SCL will allocate a new simulated database for each sdnp session.
 * These are not needed once an actual database is implemented.
 */
#define SDNPCNFG_NUMALLOC_SIM_DATABASES        TMWCNFG_MAX_SIM_DATABASES

/* Specify the number of slave DNP simulated database user managed events.
```

```
  * This is only necessary for testing the simulated database implementation of
  * database event queue management.
  */
#define SDNPCNFG_NUMALLOC_SIM_UMEVENTS        100

/* Specify the number of slave DNP simulated database Data Sets 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 SDNPCNFG_NUMALLOC_SIM_DATASETS        (TMWCNFG_MAX_SESSIONS*10)

/* Support DNP Secure Authentication Specification Version 2 */
#define SDNPCNFG_SUPPORT_SA_VERSION2          DNPCNFG_SUPPORT_SA_VERSION2

/* Support DNP Secure Authentication Specification Version 5
  * This is an optional component
  */
#define SDNPCNFG_SUPPORT_SA_VERSION5          DNPCNFG_SUPPORT_SA_VERSION5

/* Specify the total number of slave DNP Secure Authentication Version 5 Users
  * for all sessions in the entire system that can be allocated when static or
  * dynamic memory with limited quantities is used.
  */
#define SDNPCNFG_NUMALLOC_AUTH_USERS          (DNPCNFG_AUTH_MAX_NUMBER_USERS *
SDNPCNFG_NUMALLOC_SESNS)

/* Specify the number of buffers needed for generating a device profile in xml
  * format. This would never need to be more than 1 per slave session.
  * This would only need to be more than 1 if multiple sessions are supported
  * and they each need to be able to generate a device profile simultaneously.
  * Only applicable if SDNPDATA_SUPPORT_XML2 is defined.
  */
#define SDNPCNFG_NUMALLOC_DEVICE_PROFILES     TMWCNFG_MAX_SESSIONS

#endif /* SDNPCNFG_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 Slave 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 Slave 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.
   */
  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.
   * 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;

  /* 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;

  /* 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 {

  /* 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;

  /* 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,
   * 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.
   */
  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
  * 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);

#ifdef __cplusplus
}
#endif
#endif /* DNPCHNL_DEFINED */
```

4.3 SDNP Sessions

'sdnpsesn.h' contains the definition of the routines required to configure, open and close sessions in the DNP3 Slave SCL. In particular, the functions sdnpsesn_initConfig() and sdnpsesn_openSession() are called from the target application. The SDNPSESN_CONFIG structure may optionally be modified after the return from sdnpsesn_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: sdnpsesn.h
 * description: Defines structures and functions to support slave DNP3
 *  sessions.
 */
#ifndef SDNPSESN_DEFINED
#define SDNPSESN_DEFINED

#include "tmwscl/utils/tmwdefs.h"
#include "tmwscl/utils/tmwsesn.h"
#include "tmwscl/dnp/sdnpcnfg.h"
#include "tmwscl/dnp/sdnpdata.h"

#define UNSOL_NORETRY TMWDEFS_DAYS(31UL)

/* DNP SECURE AUTHENTICATION Configuration
 * The following configuration structure may be ignored if Secure Authentication is
 * not supported
 */

#if SDNPCNFG_SUPPORT_SA_VERSION2
/* Secure Authentication Per user configuration structure */
typedef struct SDNPAuthUserConfig {
  /* User Number */
  TMWTYPES_USHORT userNumber;
} SDNPSESN_AUTH_USERCONFIG;
#endif

/* Secure Authentication Configuration structure */
typedef struct SDNPAuthConfig {

  /* Association Id. Sent in error message to master. When error message is sent
   * to a different master, this identifies which association error occurred on.
   */
  TMWTYPES_USHORT assocId;

  /* MAC algorithm to sent in challenge requests sent by Outstation.
   * Master will use this algorithm to generate the MAC value in the challenge
   * reply. The Outstation will then use this same algorithm to generate a MAC to
   * authenticate the challenge reply. The MAC algorithm the Master is configured
   * to send in a challenge does NOT have to match the MAC Algorithm the Outstation
```

```
    * sends in a challenge.
    *  1 DNPAUTH_HMAC_SHA1_4OCTET   (SHA1 truncated to    4 octets   Only for SA V2)
    *  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
    */
  TMWTYPES_UCHAR   MACAlgorithm;

  /* Key wrap algorithm to use to decrypt and "unwrap" session keys
   * Deprecated on session, now per user.
   * TMWTYPES_UCHAR   keyWrapAlgorithm;
   */

  /* 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;

  /* How long to wait for any authentication reply
   * Default shall be 2 seconds
   */
  TMWTYPES_MILLISECONDS replyTimeout;

  /* Expected session key interval and count. When this amount of time elapses
   * or this quantity of Authentication messages are sent or received, the session
   * keys for this user will be invalidated.
   * Interval and count should be 2 times the master key change interval and count.
   * For systems that communicate infrequently, keyChangeInterval may be set to
   * zero, using only the maxKeyChangeCount to determine when keys should be
   * considered old and should be invalidated.
   *  default 30 minutes and 2000 messages.
   */
  TMWTYPES_MILLISECONDS keyChangeInterval;
  TMWTYPES_USHORT       maxKeyChangeCount;

  /* Agressive mode is enabled, allows aggressive mode requests from master */
  TMWTYPES_BOOL        aggressiveModeSupport;

  /* 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. */
  TMWTYPES_ULONG       testConfig;

  /* Secure Authentication Version 2 */
  TMWTYPES_BOOL        operateInV2Mode;

  /* These have been added as configuration in addition to modes being
   * set by functions
   */
  TMWTYPES_BOOL        authSendAggrResp;
  TMWTYPES_BOOL        authSendAggrUnsol;
  TMWTYPES_BOOL        preChallengeApplConf;

#if SDNPCNFG_SUPPORT_SA_VERSION2
  /* Number of errors messages to be sent before disabling error message
   * transmission.
   * default shall be 2
   * range 0-255
   */
  TMWTYPES_UCHAR       maxErrorCount;

  /* Additional 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.
   */
  SDNPSESN_AUTH_USERCONFIG authUsers[DNPCNFG_AUTHV2_MAX_NUMBER_USERS];
#endif

#if SDNPCNFG_SUPPORT_SA_VERSION5
  /* Length of random challenge data to send in g120v1 challenge and g120v5
```

```
     * key status.
     * TB2016-002 says Minimum length==4, Maximum length==64
     */
    TMWTYPES_USHORT  randomChallengeDataLength;

    /* SAv5 (1815-2012) 7.6.1.4.6 says that if this number of Session Key Status
     * Requests within the Expected Session Key Change Interval, the outstation
     * shall alert a human.
     * If a different DNP association is in use, the outstation shall send an Error
     * (g120v7) event object on that association with the code <12> Max Session Key
     * Status Requests Exceeded. This value shall be configurable up to a maximum of
     * 255 or down to 2. The default value shall be 5.
     */
    TMWTYPES_USHORT  maxSessionKeyStatusCount;

    /* These 4 maximum values are in addition to the thresholds for the individual
     * statistics. The fifth one, maxKeysDueToRestarts, is on master.
     * This is made clear in the Secure Authentication Test Procedures.
     * The outstation will take special actions when a statistic exceeds these Max
Values.
     */
    TMWTYPES_USHORT  maxAuthenticationFailures;
    TMWTYPES_USHORT  maxReplyTimeouts;
    TMWTYPES_USHORT  maxAuthenticationRekeys;
    TMWTYPES_USHORT  maxErrorMessagesSent;
  #endif

  } SDNPSESN_AUTH_CONFIG;

  /* Application layer Function code enabled for broadcast and device specific address
   * bit values. These can be or'ed together in enabledBroadcastFCs and enabledFCs to
   * indicate which FCs are enabled for broadcast and device specific addresses.
   */
#define SDNPSESN_ENABLE_FC_CONFIRM              (1<<DNPDEFS_FC_CONFIRM)
#define SDNPSESN_ENABLE_FC_READ                 (1<<DNPDEFS_FC_READ)
#define SDNPSESN_ENABLE_FC_WRITE                (1<<DNPDEFS_FC_WRITE)
#define SDNPSESN_ENABLE_FC_SELECT               (1<<DNPDEFS_FC_SELECT)
#define SDNPSESN_ENABLE_FC_OPERATE              (1<<DNPDEFS_FC_OPERATE)
#define SDNPSESN_ENABLE_FC_DIRECT_OP            (1<<DNPDEFS_FC_DIRECT_OP)
#define SDNPSESN_ENABLE_FC_DIRECT_OP_NOACK      (1<<DNPDEFS_FC_DIRECT_OP_NOACK)
#define SDNPSESN_ENABLE_FC_FRZ                  (1<<DNPDEFS_FC_FRZ)
#define SDNPSESN_ENABLE_FC_FRZ_NOACK            (1<<DNPDEFS_FC_FRZ_NOACK)
#define SDNPSESN_ENABLE_FC_FRZ_CLEAR            (1<<DNPDEFS_FC_FRZ_CLEAR)
#define SDNPSESN_ENABLE_FC_FRZ_CLEAR_NOACK      (1<<DNPDEFS_FC_FRZ_CLEAR_NOACK)
#define SDNPSESN_ENABLE_FC_FRZ_TIME             (1<<DNPDEFS_FC_FRZ_TIME)
#define SDNPSESN_ENABLE_FC_FRZ_TIME_NOACK       (1<<DNPDEFS_FC_FRZ_TIME_NOACK)
#define SDNPSESN_ENABLE_FC_COLD_RESTART         (1<<DNPDEFS_FC_COLD_RESTART)
#define SDNPSESN_ENABLE_FC_WARM_RESTART         (1<<DNPDEFS_FC_WARM_RESTART)
#define SDNPSESN_ENABLE_FC_INIT_DATA            (1<<DNPDEFS_FC_INIT_DATA_TO_DFLTS)
#define SDNPSESN_ENABLE_FC_INIT_APP             (1<<DNPDEFS_FC_INIT_APPLICATION)
#define SDNPSESN_ENABLE_FC_START_APP            (1<<DNPDEFS_FC_START_APPLICATION)
#define SDNPSESN_ENABLE_FC_STOP_APP             (1<<DNPDEFS_FC_STOP_APPLICATION)
#define SDNPSESN_ENABLE_FC_SAVE_CONFIG          (1<<DNPDEFS_FC_SAVE_CONFIGURATION)
#define SDNPSESN_ENABLE_FC_ENABLE_UNSOL         (1<<DNPDEFS_FC_ENABLE_UNSOL)
#define SDNPSESN_ENABLE_FC_DISABLE_UNSOL        (1<<DNPDEFS_FC_DISABLE_UNSOL)
#define SDNPSESN_ENABLE_FC_ASSIGN_CLASS         (1<<DNPDEFS_FC_ASSIGN_CLASS)
#define SDNPSESN_ENABLE_FC_DELAY_MEASURE        (1<<DNPDEFS_FC_DELAY_MEASURE)
#define SDNPSESN_ENABLE_FC_RECORD_CTIME         (1<<DNPDEFS_FC_RECORD_CURRENT_TIME)
#define SDNPSESN_ENABLE_FC_OPEN_FILE            (1<<DNPDEFS_FC_OPEN_FILE)
#define SDNPSESN_ENABLE_FC_CLOSE_FILE           (1<<DNPDEFS_FC_CLOSE_FILE)
#define SDNPSESN_ENABLE_FC_DELETE_FILE          (1<<DNPDEFS_FC_DELETE_FILE)
#define SDNPSESN_ENABLE_FC_GET_FILE_INFO        (1<<DNPDEFS_FC_GET_FILE_INFO)
#define SDNPSESN_ENABLE_FC_AUTHENTICATE         (1<<DNPDEFS_FC_AUTHENTICATE)
#define SDNPSESN_ENABLE_FC_ABORT                (1<<DNPDEFS_FC_ABORT)
#define SDNPSESN_ENABLE_FC_ACTIVATE_CONFIG      (1<<DNPDEFS_FC_ACTIVATE_CONFIG)

/* If FC Write is supported for broadcast, further specify which writes
 * as indicated in Device Profile for broadcast support
 */
#define SDNPSESN_ENABLE_WRITE_CLOCK             0x01
#define SDNPSESN_ENABLE_WRITE_LRTIME            0x02
#define SDNPSESN_ENABLE_WRITE_CRESTART          0x04
#define SDNPSESN_ENABLE_WRITE_OTHER             0x08

/* Slave DNP Session Configuration Info */
typedef struct SDNPSessionConfigStruct {

  /* Source address (outstation address)for this session */
  TMWTYPES_USHORT source;

  /* Destination address (master address) for this session
 * If validateSourceAddress is TMWDEFS_TRUE, this will be the
 * address to compare the masters source address to.
 * If validateSourceAddress is TMWDEFS_FALSE, this address
 * will be used as the address to send unsolicited responses.
 * All other responses will be sent to the source address received
 * from the master
 */
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 you may not want this behavior when a
 * single session times out. This is probably more likely when configuring
 * a master session.
 */
TMWTYPES_BOOL linkStatusTimeoutDisconnect;

/* Specify whether or not to validate source address in received
 * frames. DNP3 frames contain both a source address field and a
 * destination address field. If TMWDEFS_FALSE the TMW DNP3 SCL does not
 * validate the source address and frames whose destination address
 * matches a configured slave session will be accepted.
 * Setting this to TMWDEFS_TRUE requires both source and destination
 * addresses to match a local slave session before the frame is accepted.
 */
TMWTYPES_BOOL validateSourceAddress;

/* Specify whether or not to enable self address functionality on this
 * this slave device as specified by DNP Technical Bulletin 2003-003
 * Self-Address Reservation. Slave will respond to address 0xfffc as
 * though it received a request for its configured address. It will
 * respond with its own address so the master can automatically
 * discover the slave address
 */
TMWTYPES_BOOL enableSelfAddress;

/* Is this session 'active'. Inactive sessions will not transmit
 * frames or process received frames.
 */
TMWTYPES_BOOL active;

/* Specify whether application is allowed to send multi fragment responses.
 */
TMWTYPES_BOOL multiFragRespAllowed;

/* Specify whether application layer confirmations will be requested
 * for non-final fragments of a multi fragment response. Setting this to
 * TMWDEFS_FALSE would now be considered NON-COMPLIANT. Application layer
 * confirmations are always requested for responses that contain events.
 */
TMWTYPES_BOOL multiFragConfirm;

/* Parsing Table says send qualifier 5b for activate config response.
 * For backward compatibility with old implementations send qualifier 7 instead.
 */
TMWTYPES_BOOL sendObj91RespQual7;

/* Specifies whether this device will set the Need Time IIN bit
 * in response to this session at startup and after the clock valid
 * period has elapsed. If this bit is set the master will respond
 * with a time synchronization request. Typically this parameter
 * should be true for one session for each slave device. Set this
 * parameter to TMWDEFS_FALSE if report by exception is not supported
 * or there is no reason this device needs to be synchronized from
 * the master.
 */
TMWTYPES_BOOL respondNeedTime;

/* Specifies how long the local clock will remain valid after receiving
 * a time synchronization. (or after sdnpsesn_restartClockValidTime is called)
 */
TMWTYPES_MILLISECONDS clockValidPeriod;
```

```
/* Application confirm timeout specifies how long the slave DNP device will
 * wait for an application layer confirmation from the master for a
 * SOLICITED response.
 */
TMWTYPES_MILLISECONDS applConfirmTimeout;

/* Unsolicited confirm timeout specifies how long the slave DNP device will
 * wait for an application layer confirmation from the master for an
 * UNSOLICITED response. This in combination with unsolRetryDelay or
 * unsolOfflineRetryDelay will determine how frequently an unsolicited response
 * will be resent.
 */
TMWTYPES_MILLISECONDS unsolConfirmTimeout;

/* selectTimeout specifies the maximum amount of time that a select
 * will remain valid before the corresponding operate is received. If
 * an operate request is received after this period has elapsed since
 * the previous select the select will not be valid and the operate
 * request will fail.
 */
TMWTYPES_MILLISECONDS selectTimeout;

/* The time, in ms, a master must wait after receiving a response to a
 * warm restart request. This value is encoded in a time delay fine object
 * in the response which limits it to 65535 ms.
 */
TMWTYPES_USHORT warmRestartDelay;

/* The time, in ms, a master must wait after receiving a response to a
 * cold restart request. This value is encoded in a time delay fine object
 * in the response which limits it to 65535 ms.
 */
TMWTYPES_USHORT coldRestartDelay;

/* Determines whether multiple Control Relay Output Block objects are
 * allowed in a single request. The DNP3 specification allows this and
 * assumes that each operation will be performed in serial. Some devices
 * may want to limit CROB objects to one per request to simplify processing.
 *  NOTE: If this is set to TMWDEFS_FALSE, Analog Control Objects will not
 *  be allowed in the same request as the single CROB object.
 */
TMWTYPES_BOOL allowMultiCROBRequests;

/* Determines the maximum number of controls allowed in a single request.
 * This applies to both Binary (CROB) and Analog Control outputs. This must
 * be less than SDNPCNFG_MAX_CONTROL_REQUESTS. If allowMultiCROBRequests
 * is set to TMWDEFS_FALSE this won't matter for CROBs.
 */
TMWTYPES_UCHAR maxControlRequests;

/* Bit mask indicating which Writes are enabled for broadcast requests if
 * broadcast Write is enabled
 */
TMWTYPES_UCHAR enabledBroadcastWrites;

/* Bit mask indicating which Writes are enabled for device specific requests,
 * if device specific Write is enabled
 */
TMWTYPES_UCHAR enabledWrites;

/* Bit mask indicating which Function Codes are enabled for broadcast requests */
TMWTYPES_ULONG enabledBroadcastFCs;

/* Bit mask indicating which Function Codes are enabled for
 * device specific requests
 */
TMWTYPES_ULONG enabledFCs;

/* If set to TMWDEFS_FALSE, reject all file transfer requests
 * even if file transfer code is compiled in.
 */
TMWTYPES_BOOL fileTransferEnabled;

/* If a file transfer operation is in progress (i.e. a file is open) this
 * specifies the maximum amount of time, in ms, allowed between operations.
 * If no file transfer operations occur in this period the file will be
 * closed and future file transfer operations on this file will fail.
 */
TMWTYPES_MILLISECONDS fileTransferTimeout;

/* Determines whether unsolicited responses are allowed. If unsolAllowed
```

```
   * is set to TMWDEFS_FALSE no unsolicited responses will be generated and
   * requests to enable or disable unsolicited responses will fail.
   */
  TMWTYPES_BOOL unsolAllowed;

  /* Determines whether unsolicited null responses will be sent when session
   * comes "online" (meaning connected). Specs say send initial unsolicited
   * null response on restart.
   * Previous versions of SCL would also send unsolicited null response when
   * a session reconnected. Add this configuration to allow user to maintain
   * that behavior by setting this to TMWDEFS_TRUE if desired.
   */
  TMWTYPES_BOOL sendUnsolWhenOnline;

  /* Determines whether to try to send identical unsolicited retries.
   * If set to TMWDEFS_FALSE, regenerated unsolicited retries will be sent.
   * The DNP Specification allows for either identical retries with the
   * same events and sequence number or regenerated retries which may
   * contain additional new events, different IIN bits, etc, with a
   * different sequence number.
   * If set to TMWDEFS_TRUE, identical retries will be sent until confirmed by the
   * master or until the retry sequence is completed. Set unsolOfflineRetryDelay
   * to UNSOL_NORETRY to limit identical or regenerated retries to unsolMaxRetries
   */
  TMWTYPES_BOOL unsolSendIdenticalRetry;

  /* Specify the initial/new state of the unsolicited event mask. This mask
   * is used to determine which event class(es) will generate unsolicited
   * responses. According to the DNP specification, unsolicited responses
   * should be disabled until an 'Enable Unsolicited Response' request is
   * received from the master. Hence this value should generally be
   * TMWDEFS_CLASS_MASK_NONE, but some masters do not generate the 'Enable
   * Unsolicited Response' message, in which case they must be enabled here.
   */
  TMWDEFS_CLASS_MASK unsolClassMask;

  /* Determines whether an initial Null unsolicited response is sent.
   * According to the DNP Specification if unsolicited responses are supported
   * "unsolAllowed==true" then an initial null UR shall be sent until an
   * application confirm is received from the master. Some masters do not handle
   * the initial UR Null response but do support URs.
   * Setting this to true will disable the sending of the initial Null UR.
   */
  TMWTYPES_BOOL unsolDontSendInitialNull;

  /* If unsolicited responses are enabled, unsolClassXMaxEvents specifies
   * the maximum number of events in the corresponding class to be allowed
   * before an unsolicited response will be generated.
   */
  TMWTYPES_UCHAR unsolClass1MaxEvents;
  TMWTYPES_UCHAR unsolClass2MaxEvents;
  TMWTYPES_UCHAR unsolClass3MaxEvents;

  /* If unsolicited responses are enabled, unsolClassXMaxDelay specifies
   * the maximum amount of time in milliseconds after an event in the
   * corresponding class is received before an unsolicited response will
   * be generated.
   */
  TMWTYPES_MILLISECONDS unsolClass1MaxDelay;
  TMWTYPES_MILLISECONDS unsolClass2MaxDelay;
  TMWTYPES_MILLISECONDS unsolClass3MaxDelay;

  /* Specify the maximum number of unsolicited retries before changing to
   * the 'offline' retry period described below. This parameter allows you
   * to specify up to 65535 retries. If you want an infinite number of
   * retries at the same time period, set unsolOfflineRetryDelay to the
   * same value as unsolRetryDelay.
   */
  TMWTYPES_USHORT unsolMaxRetries;

  /* Specifies the time, in milliseconds, to delay after an unsolicited confirm
   * timeout before retrying the unsolicited response.
   */
  TMWTYPES_MILLISECONDS unsolRetryDelay;

  /* Specifies the time, in milliseconds, to delay after an unsolicited
   * timeout before retrying the unsolicited response after unsolMaxRetries
   * have been attempted. To disable retries after unsolMaxRetries set this
   * value to UNSOL_NORETRY or TMWDEFS_DAYS(31) which means this unsolicited
   * will not be retried. A new unsolicited response may be generated when
   * conditions warrant.
   */
  TMWTYPES_MILLISECONDS unsolOfflineRetryDelay;

  /* Specifies whether to continue using unsolOfflineRetryDelay value or to
   * use it for one time period and then revert back to unsolRetryDelay till
   * unsolMaxRetries is once again exceeded.
   */
  TMWTYPES_BOOL unsolOfflineDelayOnce;

  /* Default variations
   * Specifies the variation that will be used for unsolicited responses
   * and in response to an integrity poll or a read requesting variation 0.
   *
   * If any of these configuration variables is set to zero a function
   * sdnpdata_xxxDefVariation will be called for each point of that
   * object group to determine the default variation for that point.
   */

  /* If this is TMWDEFS_TRUE, only unsolicited events in the class whose timer
   * timer expires or count is exceeded will be sent. Otherwise unsolicited
   * events in all classes will be sent.
   */
  TMWTYPES_BOOL       unsolSendByClass;

  TMWTYPES_UCHAR obj01DefaultVariation;
  TMWTYPES_UCHAR obj02DefaultVariation;
  TMWTYPES_UCHAR obj03DefaultVariation;
  TMWTYPES_UCHAR obj04DefaultVariation;
  TMWTYPES_UCHAR obj10DefaultVariation;
  TMWTYPES_UCHAR obj11DefaultVariation;
  TMWTYPES_UCHAR obj13DefaultVariation;
  TMWTYPES_UCHAR obj20DefaultVariation;
  TMWTYPES_UCHAR obj21DefaultVariation;
  TMWTYPES_UCHAR obj22DefaultVariation;
  TMWTYPES_UCHAR obj23DefaultVariation;
  TMWTYPES_UCHAR obj30DefaultVariation;
  TMWTYPES_UCHAR obj31DefaultVariation;
  TMWTYPES_UCHAR obj32DefaultVariation;
  TMWTYPES_UCHAR obj33DefaultVariation;
  TMWTYPES_UCHAR obj34DefaultVariation;
  TMWTYPES_UCHAR obj40DefaultVariation;
  TMWTYPES_UCHAR obj42DefaultVariation;
  TMWTYPES_UCHAR obj43DefaultVariation;
  TMWTYPES_UCHAR obj114DefaultVariation;
  TMWTYPES_UCHAR obj115DefaultVariation;
  TMWTYPES_UCHAR obj122DefaultVariation;

  /* Object groups included in response to read static data request
   * "empty" entries should be set to 0. The entries do not have to
   * be in the first x entries, ie {1,0,21,40,0,0,0,0...} is valid.
   * and would indicate Binary Inputs, Frozen Counters and Analog Output
   * Statuses should be included in a response to a read static data request.
   */
  TMWTYPES_UCHAR staticGroups[SDNPCNFG_MAX_NUMBER_STATIC_GROUPS];

  /* If this is TMWDEFS_TRUE event queueing and retrieval will be handled outside
   * of the SCL through the sdnpdata_umEventxxx() functions. When this option is
   * used, the user provided code must implement max queue sizes, event modes,
   * whether to delete the oldest or newest event on overflow, etc.
   * SDNPCNFG_USER_MANAGED_EVENTS must be set to TMWDEFS_TRUE at compile time.
   */
  TMWTYPES_BOOL userManagedEvents;

  /* If this is TMWDEFS_TRUE events will be sorted by timestamp when
   * xxx_addEvent is called. This was the desired behavior in IEEE1815-2012
   * This is no longer the correct behavior. The new version of 1815-202x(1?)
   * Tables 5-4 and 5-5 clearly show the order events are queued is the order
   * they should be sent in responses, NOT by timestamp.
   */
  TMWTYPES_BOOL sortEventsByTime;

  /* If this is TMWDEFS_TRUE timestamps are provided for DNP3 events.
   * If this is set to TMWDEFS_FALSE event variations without time will be sent
   * for events even if with time variations are requested.
   */
  TMWTYPES_BOOL supportEventsWithTime;

  /* If this is TMWDEFS_TRUE the event with the earliest timeStamp will be
   * deleted when a new event is added to an event queue that is full.
   */
  TMWTYPES_BOOL deleteOldestEvent;
  /* Diagnostic mask */
  TMWTYPES_ULONG sesnDiagMask;

#if SDNPDATA_SUPPORT_OBJ2
  /* Binary Input Event Configuration */
  TMWTYPES_USHORT binaryInputMaxEvents;
  TMWDEFS_EVENT_MODE binaryInputEventMode;
  TMWTYPES_MILLISECONDS binaryInputScanPeriod;
#endif

#if SDNPDATA_SUPPORT_OBJ4
  /* Double Bit Input Event Configuration */
  TMWTYPES_USHORT doubleInputMaxEvents;
  TMWDEFS_EVENT_MODE doubleInputEventMode;
  TMWTYPES_MILLISECONDS doubleInputScanPeriod;
#endif

#if SDNPDATA_SUPPORT_OBJ11
  /* Binary Output Event Configuration */
  TMWTYPES_USHORT binaryOutputMaxEvents;
  TMWDEFS_EVENT_MODE binaryOutputEventMode;
  TMWTYPES_MILLISECONDS binaryOutputScanPeriod;
#endif

#if SDNPDATA_SUPPORT_OBJ13
  /* Binary Output Command Event Configuration */
  TMWTYPES_USHORT binaryOutCmdMaxEvents;
  TMWDEFS_EVENT_MODE binaryOutCmdEventMode;
  TMWTYPES_MILLISECONDS binaryOutCmdScanPeriod;
#endif

#if SDNPDATA_SUPPORT_OBJ32
  /* Analog Input Event Configuration */
  TMWTYPES_USHORT analogInputMaxEvents;
  TMWDEFS_EVENT_MODE analogInputEventMode;
  TMWTYPES_MILLISECONDS analogInputScanPeriod;
#endif

#if SDNPDATA_SUPPORT_OBJ42
  /* Analog Output Event Configuration */
  TMWTYPES_USHORT analogOutputMaxEvents;
  TMWDEFS_EVENT_MODE analogOutputEventMode;
  TMWTYPES_MILLISECONDS analogOutputScanPeriod;
#endif

#if SDNPDATA_SUPPORT_OBJ43
  /* Analog Output Command Event Configuration */
  TMWTYPES_USHORT analogOutCmdMaxEvents;
  TMWDEFS_EVENT_MODE analogOutCmdEventMode;
  TMWTYPES_MILLISECONDS analogOutCmdScanPeriod;
#endif

#if SDNPDATA_SUPPORT_OBJ22
  /* Binary Counter Event Configuration */
  TMWTYPES_USHORT binaryCounterMaxEvents;
  TMWDEFS_EVENT_MODE binaryCounterEventMode;
  TMWTYPES_MILLISECONDS binaryCounterScanPeriod;
#endif

#if SDNPDATA_SUPPORT_OBJ23
  /* Frozen Counter Event Configuration */
  TMWTYPES_USHORT frozenCounterMaxEvents;
  TMWDEFS_EVENT_MODE frozenCounterEventMode;
  TMWTYPES_MILLISECONDS frozenCounterScanPeriod;
#endif

#if SDNPDATA_SUPPORT_OBJ33
  /* Frozen Counter Event Configuration */
  TMWTYPES_USHORT frozenAnalogInMaxEvents;
  TMWDEFS_EVENT_MODE frozenAnalogInEventMode;
  TMWTYPES_MILLISECONDS frozenAnalogInScanPeriod;
#endif

#if SDNPDATA_SUPPORT_OBJ88
  /* Data Set Snapshot Event Configuration */
  TMWTYPES_USHORT datasetMaxEvents;
  TMWDEFS_EVENT_MODE datasetEventMode;
#endif

#if SDNPDATA_SUPPORT_OBJ110
```

```
  /* Set this to TRUE to truncate string to length specified by read variation,
   * which is the normally desired behavior. Setting this to false may send
   * a longer string than the variation specified. ie reading group 110
   * variation 10 would either truncate all strings to at most 10 bytes,
   * or if this parameter is FALSE would allow a longer string to be returned.
   * This behavior would be more like reading variation 0.
   */
  TMWTYPES_BOOL truncateStrings;
#if SDNPDATA_SUPPORT_OBJ111
  /* String Event Configuration */
  TMWTYPES_USHORT stringMaxEvents;

  TMWDEFS_EVENT_MODE stringEventMode;
  TMWTYPES_MILLISECONDS stringScanPeriod;
#endif
#endif

#if SDNPDATA_SUPPORT_OBJ113
  /* Virtual Terminal Event Configuration */
  TMWTYPES_USHORT virtualTerminalMaxEvents;

  /* See description of truncateStrings */
  TMWTYPES_BOOL truncateVirtualTerminalEvents;
  TMWDEFS_EVENT_MODE virtualTerminalEventMode;
  TMWTYPES_MILLISECONDS virtualTerminalScanPeriod;
#endif

#if SDNPDATA_SUPPORT_OBJ115
  /* Extended String Event Configuration */
  TMWTYPES_USHORT extStringMaxEvents;
  TMWDEFS_EVENT_MODE extStringEventMode;
  TMWTYPES_MILLISECONDS extStringScanPeriod;
#endif

  /* User registered statistics callback function and parameter */
  TMWSESN_STAT_CALLBACK pStatCallback;
  void *pStatCallbackParam;

#if SDNPDATA_SUPPORT_OBJ120
  TMWTYPES_BOOL         authenticationEnabled;
  TMWTYPES_USHORT       authSecStatMaxEvents;
  TMWDEFS_EVENT_MODE    authSecStatEventMode;
  SDNPSESN_AUTH_CONFIG  authConfig;
#endif

#if SDNPDATA_SUPPORT_XML2
  /*    * If this is TMWDEFS_TRUE, the per point value, quality and timestamp
   * called dnpData in the Device Profile schema will be generated, in addition
   * to the current configuration values. SDNPDATA_SUPPORT_XML2_DNPDATA must
   * be defined as TMWDEFS_TRUE also.
   */
  TMWTYPES_BOOL       xml2WriteDnpData;
#endif

#if SDNPCNFG_DISABLE_SELOP_SEQ_CHECK
  /* Disable checking for consecutive application sequence numbers required
   * between Select and Operate requests.
   * This will allow only read requests between the select and operate requests
   * NOTE: This SDNP device will fail the DNP3 Conformance Tests if configured
   * this way.
   */
  TMWTYPES_BOOL disableSelOpSeqCheck;
#endif

#if SDNPDATA_SUPPORT_STATIC_VAR_POINT || SDNPDATA_SUPPORT_EVENT_VAR_POINT
  /* If this is TMWDEFS_TRUE allow per point specification of precision of counter
   * and frozen counter points and events.
   * This allows the database to indicate a point value should be sent as a
   * 16 or 32 bit variation even when read as the other variation as allowed in
   * the new 1815-202x.
   * For example a read from the master of variation 1 32-bit could now result
   * in the response using variation 2 to indicate only 16 bit counter precision
   * (or the reverse) is to be used for that point. This functionality is an
   * extension to the ability to specify per point variation when variation zero
   * is requested from the master using the same database interface functionality.
   * obj20DefaultVariation, obj21DefaultVariation, obj22DefaultVariation, and/or
   * obj23DefaultVariation must each also be set to zero to enable this for running
   * counter, frozen counter and associated event groups.
   */
  TMWTYPES_BOOL counterSizeVarPerPoint;
  /* If this is TMWDEFS_TRUE allow per point specification of precision of analog
   * and frozen analog input points and events.
   * This allows the database to indicate a point value should be sent as a 16 or
   * 32 bit variation even when read as the other variation as allowed in the
   * new 1815-202x.
   * For example a read from the master of variation 1 32-bit could now result
   * in the response using variation 2 to indicate 16 bit analog precision
   * (or the reverse) is to be used for that point. This also applies to the
   * variations for Single and Double precision if supported. This functionality
   * is an extension to the ability to specify per point variation when variation
   * zero is requested from the master using the same database interface
functionality.
   * obj30DefaultVariation, obj31DefaultVariation, obj32DefaultVariation, and/or
   * obj33DefaultVariation must each also be set to zero to enable this for
   * analog input, frozen analog input and associated event groups.
   */
  TMWTYPES_BOOL analogInSizeVarPerPoint;

  /* If this is TMWDEFS_TRUE allow per point specification of precision of
   * analog output points and events.
   * This allows the database to indicate a point value should be sent as a 16
   * or 32 bit variation even when read as the other variation as allowed in
   * the new 1815-202x.
   * For example a read from the master of variation 1 32-bit could now result
   * in the response using variation 2 to indicate 16 bit analog precision
   * (or the reverse) is to be used for that point. This also applies to the
   * variations for Single and Double precision if supported. This functionality
   * is an extension to the ability to specify per point variation when variation
   * zero is requested from the master using the same database interface
functionality.
   * obj40DefaultVariation, obj42DefaultVariation, and/or obj43DefaultVariation must
   * each also be set to zero to enable this for analog output status and associated
   * event groups.
   */
  TMWTYPES_BOOL analogOutSizeVarPerPoint;
#endif

} SDNPSESN_CONFIG;

/* DEPRECATED SHOULD USE sdnpsesn_getSessionConfig and
 *  sdnpsesn_setSessionConfig
 */
#define SDNPSESN_CONFIG_SOURCE            0x00000001
#define SDNPSESN_CONFIG_DESTINATION       0x00000002
#define SDNPSESN_CONFIG_UNSOL_ALLOWED     0x00000004
#define SDNPSESN_CONFIG_UNSOL_MAX_EVENTS  0x00000008
#define SDNPSESN_CONFIG_SELECT_TIMEOUT    0x00000010
#define SDNPSESN_CONFIG_UNSOL_MAX_DELAY   0x00000020
#define SDNPSESN_CONFIG_ACTIVE            0x00000040
#define SDNPSESN_CONFIG_CNFM_TIMEOUT      0x00000080
#define SDNPSESN_CONFIG_MULTI_FRAG        0x00000100
#define SDNPSESN_CONFIG_UNSOL_MASK        0x00000200
#define SDNPSESN_CONFIG_UNSOL_RETRIES     0x00000400
#define SDNPSESN_CONFIG_UNSOL_DELAY       0x00000800
#define SDNPSESN_CONFIG_UNSOL_OFFDELAY    0x00001000
#define SDNPSESN_CONFIG_STATIC_GROUPS     0x00002000

/* Include slave DNP3 'private' structures and functions */
#include "tmwscl/dnp/sdnpsesp.h"

#ifdef __cplusplus
extern "C" {
#endif

  /* function: sdnpsesn_initConfig
   * purpose: Initialize DNP3 slaver 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
   *  sdnpsesn_openSession.
   * arguments:
   *  pConfig - pointer to configuration data structure to initialize
   * returns:
   *  void
   */
  TMWDEFS_SCL_API void TMWDEFS_GLOBAL sdnpsesn_initConfig(
    SDNPSESN_CONFIG *pConfig);

  /* function: sdnpsesn_openSession
   * purpose: Open and DNP3 slave session
   * arguments:
```

```
 *  pChannel - channel to open session on
 *  pConfig - DNP3 slave configuration data structure
 *  pUserHandle - handle passed to session database initialization routine
 * returns:
 *  Pointer to new session or TMWDEFS_NULL.
 */
TMWDEFS_SCL_API TMWSESN * TMWDEFS_GLOBAL sdnpsesn_openSession(
  TMWCHNL *pChannel,
  const SDNPSESN_CONFIG *pConfig,
  void *pUserHandle);

/* function: sdnpsesn_applyBinaryFileValues
 purpose: Initialize DNP3 channel configuration data structures and session
 *  configuration data structures using the values in a binary configuration
 *  file. The structures are first initialized to default values just like they
 *  are by dnpchnl_initConfig() and sdnpsesn_initConfig().  The values found in
the
 *  binary configuration file are then applied over the default values.
 *  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 and sdnpsesn_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
 *  pIOConfig - pointer to target layer configuration
 *  pSesnConfig - pointer to configuration session configuration
 * returns:
 *  TMWDEFS_TRUE if successful
 */
TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpsesn_applyBinaryFileValues (
  char * pFileName,
  DNPCHNL_CONFIG *pDNPConfig,
  DNPLINK_CONFIG *pLinkConfig,
  void *pIOConfig,
  SDNPSESN_CONFIG *pSesnConfig);

/* function: sdnpsesn_getSessionConfig
 * purpose:  Get current configuration from a currently open session
 * arguments:
 *  pSession - session to get configuration from
 *  pConfig - dnp slave configuration data structure to be filled in
 * returns:
 *  TMWDEFS_TRUE if successful
 */
TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpsesn_getSessionConfig(
  TMWSESN *pSession,
  SDNPSESN_CONFIG *pConfig);

/* function: sdnpsesn_setSessionConfig
  * purpose: Modify a currently open session
 *  NOTE: normally sdnpsesn_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 slave configuration data structure
 * returns:
 *  TMWDEFS_TRUE if successful
 */
TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpsesn_setSessionConfig(
  TMWSESN *pSession,
  const SDNPSESN_CONFIG *pConfig);

/* function: sdnpsesn_modifySession
 *  DEPRECATED FUNCTION, SHOULD USE sdnpsesn_setSessionConfig()
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpsesn_modifySession(
  TMWSESN *pSession,
  const SDNPSESN_CONFIG *pConfig,
  TMWTYPES_ULONG configMask);

/* function: sdnpsesn_closeSession
 * purpose: Close a currently open session
 * arguments:
 *  pSession - session to close
 * returns:
```

```
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWDEFS_SCL_API TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpsesn_closeSession(
  TMWSESN *pSession);

/* function: sdnpsesn_restartClockValidTime
 * purpose: Restart the clock valid time. An application could call this if it
 *   had set the time using an external mechanism and wanted the SCL to consider
 *   the time to be valid for the configured clockValidPeriod.
 * arguments:
 *  pSession - pointer to session returned by sdnpsesn_openSession
 * returns:
 *  void
 */
TMWDEFS_SCL_API void TMWDEFS_GLOBAL sdnpsesn_restartClockValidTime(
  TMWSESN *pSession);

/* function: sdnpsesn_addAuthUser
 * purpose: Add a Secure Authentication User
 * arguments:
 *  pSession - session
 *  userNumber - user number for this user
 * returns:
 *  TMWDEFS_TRUE if successfull, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpsesn_addAuthUser(
  TMWSESN *pSession,
  TMWTYPES_USHORT userNumber);

/* function: sdnpsesn_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
 */
TMWTYPES_USHORT TMWDEFS_GLOBAL sdnpsesn_getAuthUser(
  TMWSESN *pSession,
  TMWTYPES_USHORT index);

/* function: sdnpsesn_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
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpsesn_removeAuthUser(
  TMWSESN *pSession,
  TMWTYPES_USHORT userNumber);

/* function: sdnpsesn_authSendAggrResp
 * purpose: Specify whether SOLICITED responses sent to the master should have
 *  the aggressive mode objects added. Only the default user number (1) is
 *  used as required by the specification.
 * arguments:
 *  pSession - pointer to session structure returned by sdnpsesn_openSession()
 *  sendAggressive - TMWDEFS_TRUE turns on aggressive mode sending
 *                   TMWDEFS_FALSE turns off aggressive mode sending
 * returns:
 *  void
 */
TMWDEFS_SCL_API void TMWDEFS_GLOBAL sdnpsesn_authSendAggrResp(
  TMWSESN *pSession,
  TMWTYPES_BOOL sendAggressive);

/* function: sdnpsesn_authSendAggrUnsol
 * purpose: Specify whether UNSOLICITED responses sent to the master should have
 *  the aggressive mode objects added. Only the default user number (1) is
 *  used as required by the specification.
 * arguments:
 *  pSession - pointer to session structure returned by sdnpsesn_openSession()
 *  sendAggressive - TMWDEFS_TRUE turns on aggressive mode sending
 *                   TMWDEFS_FALSE turns off aggressive mode sending
 * returns:
 *  void
 */
TMWDEFS_SCL_API void TMWDEFS_GLOBAL sdnpsesn_authSendAggrUnsol(
    TMWSESN *pSession,
    TMWTYPES_BOOL sendAggressive);

  /* function: sdnpsesn_authPreChallApplConf
   * purpose: Send a "preChallenge" in responses requesting application confirms.
   *  This includes responses containing events and ALL cases where an application
   *  confirm is requested including nonfinal fragments of a multifragment response.
   *  If outstation is going to challenge an application confirm, there can be a
   *  problem if the master does not expect it. The master may send another
   *  request, or if there is
   *  a half duplex line, there may be a collision. Setting this to TMWDEFS_TRUE
   *  will cause a "preChallenge" G120V1 to be appended to the data the outstation
   *  sends with the APPL CON bit set. The master should then send the application
   *  confirm using aggressive mode. This means the outstation will not have to
   *  challenge the confirm to verify the master.
   *  This can be left set to TMWDEFS_TRUE, and ALL responses requesting confirms
   *  will be "pre-challenged" or this can be set to TMWDEFS_TRUE only when a
   *  pre-challenge is desired and then set to TMWDEFS_FALSE, so no more
   *  pre-challenges will be sent.
   * arguments:
   *  pSession - pointer to session structure returned by sdnpsesn_openSession()
   *  preChallenge - TMWDEFS_TRUE if g120v1 is to be appended to responses
   *                   containing request for application confirm
   *                 TMWDEFS_FALSE to not append g120v1 object to responses
   * returns:
   *  void
   */
  TMWDEFS_SCL_API void TMWDEFS_GLOBAL sdnpsesn_authPreChallApplConf(
    TMWSESN *pSession,
    TMWTYPES_BOOL preChallenge);

  /* function: sdnpsesn_requestApplConfirm
   * purpose: Function to do some common processing to request an application
   *  confirm and also check to see if prechallenge is needed in response.
   *  pSession - pointer to session structure returned by sdnpsesn_openSession()
   *  pResponse - pointer to response being built.
   * returns:
   *  void
   */
  TMWDEFS_SCL_API void TMWDEFS_GLOBAL sdnpsesn_requestApplConfirm(
    TMWSESN *pSession,
    TMWSESN_TX_DATA *pResponse);

  /* function: sdnpsesn_failedUnsolTx
   * purpose: Called if transmission of unsolicited response failed.
   *  This includes if the connection is not open.
   * arguments:
   *  pSession - pointer to session structure returned by sdnpsesn_openSession()
   * returns:
   *  void
   */
  void TMWDEFS_CALLBACK sdnpsesn_failedUnsolTx(
    TMWSESN *pSession);


#ifdef __cplusplus
}
#endif
#endif /* SDNPSESN_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 */
/* 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
 */
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
 */
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
 *   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,
  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
 * returns:
 *  void
 */
TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwtimer_cancel(
  TMWTIMER *pTimer);


#ifdef __cplusplus
}
#endif
#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,
    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:
   *  pChannel - pointer to channel to check.
   * returns:
   *  void
   */
  TMWDEFS_SCL_API void TMWDEFS_GLOBAL tmwpltmr_checkMultiTimer(TMWCHNL *pChannel);

#ifdef __cplusplus
}
#endif

#endif /* TMWPLTMR_DEFINED */
```

5 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. */
} 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
                                   * 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;
#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

/* 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
                                       * 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;

/* 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,

  /* 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)

#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
 * 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
 *    - 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,
  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);

/* 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;

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);

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,
  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

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 */
  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 */
  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;
  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.
   * 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
   *  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,
                                                       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);

  /* 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,
   *  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
   * 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
   */
  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
   * 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.
   * 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);

  /* 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(
    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
   *    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);
```

```
/* 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
 * 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
 *  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);

/* 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
 *   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:
 *  - 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
 *    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);

/* 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 */
```

6 SDNP Database Interface

'sdnpdata.h' contains configuration parameters controlling what Object Groups and Variations and functionality are supported, as well as defining the interface to the database routines that must be implemented. This default functionality is NOT intended to be used by all implementations. After careful consideration of your requirements, this file 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, 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 CI Guide or Device Profile as required for each protocol.

NOTE: Level 4 and beyond functionality including Object Group 70 File Transfer has been removed from the SDNP SCL functionality compiled in by default. The .NET SDNP SCL will now use most of the same default settings as the C SCL. To enable functionality you should modify the defines in sdnpdata.h. For a detailed list of functionality that is no longer compiled in by default see Configuration Files section Default Functionality.

```
/*****************************************************************************/
/* 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: sdnpdata.h
 * description: This file defines the interface between the TMW DNP3 slave
 *  SCL and the target database. The user should modify the corresponding
 *  implementations in sdnpdata.c as required to access the target database.
 *
 * The DNP3 specification states that point numbers for each object type
 *  should start at 0 and proceed sequentially to N-1 where N is the number
 *  of points of that type. Hence the TMW SCL assumes that the point index
 *  and point number are equivalent and will frequently loop from 0 to N-1
 *  when processing requests with the 'All Points' qualifier.
 *
 * Although not recommended by the DNP3 specification, it is sometimes
 *  desirable to 'disable' specific points in the DNP3 point map creating
 *  gaps in the DNP3 point map. This is supported by the TMW SCL by returning
 *  TMWDEFS_NULL from the appropriate sdnpdata_xxxGetPoint subroutine defined
 *  below.
 *
 * Routines are provided below to access data in a DNP3 slave device. A set
 *  of routines are provided for each data type based on the requirements of
 *  that data type. The following generic descriptions apply to corresponding
 *  data type specific subroutines.
 *
 * sdnpdata_xxxQuantity - This routine returns the number of data points of
 *  of the corresponding type in the specified database. This includes ALL
 *  of the data points in the slave DNP3 database whether they are currently
 *  enabled or disabled (i.e., the quantity is one more than the highest point
 *  number).
 *
 * sdnpdata_xxxGetPoint - Return a pointer to a handle that will be used to
 *  access information and values for the specified point. This routine should
 *  return TMWDEFS_NULL if the specified data point is currently disabled.
 *
 * sdnpdata_xxxRead - For data types that support reading their current value
 *  this subroutine is used to read the current value from the specified data
 *  point.
 *
 * sdnpdata_xxxWrite - For data types that support writing this subroutine
 *  is used to write a new value to the specified data point.
 *
 * sdnpdata_xxxChanged - For data types that support the generation of change
 *  events this subroutine is used to periodically scan for changes. These will
 *  be called if xxxScanPeriod is set to a nonzero value.
 *
 * Several of the routines below return an 8 bit flags value in the memory
 *  location pointed to by the pFlags argument. This value should be an or'd
 *  combination of the appropriate DNPDEFS_DBAS_FLAG_XXX masks defined in
 *  dnpdefs.h. Note that not all flags are valid for all objects/variations.
 *  Valid flags for each object/variation are listed in the header comments
 *  for each function.
 */
#ifndef SDNPDATA_DEFINED
#define SDNPDATA_DEFINED

#include "tmwscl/utils/tmwdefs.h"
#include "tmwscl/utils/tmwtypes.h"
#include "tmwscl/utils/tmwdtime.h"
#include "tmwscl/dnp/dnpcnfg.h"
#include "tmwscl/dnp/dnpdata.h"
#include "tmwscl/dnp/dnpauth.h"

/* Define constants */

/* Beginning in version 3.29.0 LEVEL_TMW and LEVEL_4 are set to
 * TMWDEFS_FALSE by default.
 */

/* These defines give an idea of what functionality is defined at each
 * DNP3 Level according to the specification. LEVEL_TMW indicates functionality
 * beyond Level 4 that WAS compiled in by default in the SDNP SCL before
 * version 3.29
 * There is additional beyond Level 4 functionality that was compiled out by
 * default.
 * You can set TMW to FALSE to disable some default functionality, or you can
 * set individual items below to FALSE. If you want to limit functionality to a
 * Level you should set that define and lower to TRUE and higher Level defines
 * to FALSE. For example to limit functionality to LEVEL2 you should leave
 * LEVEL1 and LEVEL2 TRUE and set LEVEL3, LEVEL4 and LEVEL_TMW to FALSE.
 * You may still choose to set individual items to TRUE or FALSE below.
 * Some functionality is dependent on other so you may have to experiment.
 * For example frozen counters depend on binary counters being supported.
 *
 * 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.
 */
#define SDNPDATA_CNFG_LEVEL1     TMWDEFS_TRUE
#define SDNPDATA_CNFG_LEVEL2     TMWDEFS_TRUE
#define SDNPDATA_CNFG_LEVEL3     TMWDEFS_TRUE
#ifndef SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_CNFG_LEVEL4     TMWDEFS_FALSE
#endif

/* Note in v3.29.0 File Transfer Object Group 70 was removed from LEVEL_TMW
 * to require this functionality to be intentionally incuded.
 */
#ifndef SDNPDATA_CNFG_LEVEL_TMW
#define SDNPDATA_CNFG_LEVEL_TMW  TMWDEFS_FALSE
#endif

#if SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_CNFG_SUPPORT_FLOAT  TMWCNFG_SUPPORT_FLOAT
#define SDNPDATA_CNFG_SUPPORT_DOUBLE TMWCNFG_SUPPORT_DOUBLE
#else
#define SDNPDATA_CNFG_SUPPORT_FLOAT  TMWDEFS_FALSE
#define SDNPDATA_CNFG_SUPPORT_DOUBLE TMWDEFS_FALSE
#endif

/* Device Attributes */
#ifndef SDNPDATA_SUPPORT_OBJ0
#define SDNPDATA_SUPPORT_OBJ0    TMWDEFS_FALSE
#endif

/* Binary Inputs */
/* If you support for Variation 1, you should also provide support
 * for variation 2. Otherwise if flags are not nominal it won't be
 * possible to send back the flags as required by Technical Bulletin
 * TB 2003-002 Object Flags
 */
#define SDNPDATA_SUPPORT_OBJ1_V1 SDNPDATA_CNFG_LEVEL1
#define SDNPDATA_SUPPORT_OBJ1_V2 SDNPDATA_CNFG_LEVEL1
#define SDNPDATA_SUPPORT_OBJ1 \
  (SDNPDATA_SUPPORT_OBJ1_V1 | \
   SDNPDATA_SUPPORT_OBJ1_V2)

/* Binary Input Events */
#define SDNPDATA_SUPPORT_OBJ2_V1 SDNPDATA_CNFG_LEVEL1
#define SDNPDATA_SUPPORT_OBJ2_V2 SDNPDATA_CNFG_LEVEL1
#define SDNPDATA_SUPPORT_OBJ2_V3 SDNPDATA_CNFG_LEVEL1
#define SDNPDATA_SUPPORT_OBJ2 \
  (SDNPDATA_SUPPORT_OBJ2_V1 | \
   SDNPDATA_SUPPORT_OBJ2_V2 | \
   SDNPDATA_SUPPORT_OBJ2_V3)

/* Double Bit Inputs */
#ifndef SDNPDATA_SUPPORT_OBJ3
#define SDNPDATA_SUPPORT_OBJ3_V1 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ3_V2 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ3 \
  (SDNPDATA_SUPPORT_OBJ3_V1 | \
   SDNPDATA_SUPPORT_OBJ3_V2)
#endif

/* Double bit Input Events */
#ifndef SDNPDATA_SUPPORT_OBJ4
#define SDNPDATA_SUPPORT_OBJ4_V1 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ4_V2 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ4_V3 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ4 \
  (SDNPDATA_SUPPORT_OBJ4_V1 | \
   SDNPDATA_SUPPORT_OBJ4_V2 | \
   SDNPDATA_SUPPORT_OBJ4_V3)
#endif

/* Binary Output Status */
/* If you support for Variation 1, you should also provide support
 * for variation 2. Otherwise if flags are not nominal it won't be
 * possible to send back the flags as required by Technical Bulletin
 * TB 2003-002 Object Flags
 */
#define SDNPDATA_SUPPORT_OBJ10_V1 SDNPDATA_CNFG_LEVEL1
#define SDNPDATA_SUPPORT_OBJ10_WRITE SDNPDATA_CNFG_LEVEL1
#define SDNPDATA_SUPPORT_OBJ10_V2 SDNPDATA_CNFG_LEVEL1
#define SDNPDATA_SUPPORT_OBJ10 \
  (SDNPDATA_SUPPORT_OBJ10_V1 | \
   SDNPDATA_SUPPORT_OBJ10_V2)

/* If Binary Output Event support is not defined on the command line
 * set the configuration here
 */
#ifndef SDNPDATA_SUPPORT_OBJ11
/* Binary Output Events */
#define SDNPDATA_SUPPORT_OBJ11_V1 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ11_V2 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ11 \
  (SDNPDATA_SUPPORT_OBJ11_V1 | \
   SDNPDATA_SUPPORT_OBJ11_V2)
#endif

/* Control Relay Output Block */
#define SDNPDATA_SUPPORT_OBJ12_V1 SDNPDATA_CNFG_LEVEL1
#define SDNPDATA_SUPPORT_OBJ12_V2 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ12_V3 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ12 \
  (SDNPDATA_SUPPORT_OBJ12_V1 | \
   SDNPDATA_SUPPORT_OBJ12_V2 | \
   SDNPDATA_SUPPORT_OBJ12_V3)

/* If Binary Output Command Event support is not defined on the command line
 * set the configuration here
 */
#ifndef SDNPDATA_SUPPORT_OBJ13
/* Binary Output Command Events */
#define SDNPDATA_SUPPORT_OBJ13_V1 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ13_V2 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ13 \
  (SDNPDATA_SUPPORT_OBJ13_V1 | \
   SDNPDATA_SUPPORT_OBJ13_V2)
#endif

/* Binary Counters */
/* If you support for Variation 5, you should also provide support
 * for variation 1. Otherwise if flags are not nominal it won't be
 * possible to send back the flags as required by Technical Bulletin
 * TB 2003-002 Object Flags.
 * For the same reason if Variation 6 is supported, Variation 2 should be
 * supported.
 */
#define SDNPDATA_SUPPORT_OBJ20_V1 SDNPDATA_CNFG_LEVEL1
#define SDNPDATA_SUPPORT_OBJ20_V2 SDNPDATA_CNFG_LEVEL1
#define SDNPDATA_SUPPORT_OBJ20_V5 SDNPDATA_CNFG_LEVEL1
#define SDNPDATA_SUPPORT_OBJ20_V6 SDNPDATA_CNFG_LEVEL1
#define SDNPDATA_SUPPORT_OBJ20 \
  (SDNPDATA_SUPPORT_OBJ20_V1 | \
   SDNPDATA_SUPPORT_OBJ20_V2 | \
   SDNPDATA_SUPPORT_OBJ20_V5 | \
   SDNPDATA_SUPPORT_OBJ20_V6)

/* Frozen Counters */
/* If you support for Variation 9, you should also provide support
 * for variation 1. Otherwise if flags are not nominal it won't be
 * possible to send back the flags as required by Technical Bulletin
 * TB 2003-002 Object Flags.
 * For the same reason if Variation 10 is supported, Variation 2 should be
 * supported.
 */
#define SDNPDATA_SUPPORT_OBJ21_V1 SDNPDATA_CNFG_LEVEL2
#define SDNPDATA_SUPPORT_OBJ21_V2 SDNPDATA_CNFG_LEVEL2
#define SDNPDATA_SUPPORT_OBJ21_V5 SDNPDATA_CNFG_LEVEL2
#define SDNPDATA_SUPPORT_OBJ21_V6 SDNPDATA_CNFG_LEVEL2
#define SDNPDATA_SUPPORT_OBJ21_V9 SDNPDATA_CNFG_LEVEL2
#define SDNPDATA_SUPPORT_OBJ21_V10 SDNPDATA_CNFG_LEVEL2
#define SDNPDATA_SUPPORT_OBJ21 \
  (SDNPDATA_SUPPORT_OBJ21_V1 | \
   SDNPDATA_SUPPORT_OBJ21_V2 | \
   SDNPDATA_SUPPORT_OBJ21_V5 | \
   SDNPDATA_SUPPORT_OBJ21_V6 | \
   SDNPDATA_SUPPORT_OBJ21_V9 | \
   SDNPDATA_SUPPORT_OBJ21_V10)

/* Binary Counter Events */
#define SDNPDATA_SUPPORT_OBJ22_V1 SDNPDATA_CNFG_LEVEL1
#define SDNPDATA_SUPPORT_OBJ22_V2 SDNPDATA_CNFG_LEVEL1
#define SDNPDATA_SUPPORT_OBJ22_V5 SDNPDATA_CNFG_LEVEL_TMW
#define SDNPDATA_SUPPORT_OBJ22_V6 SDNPDATA_CNFG_LEVEL_TMW
#define SDNPDATA_SUPPORT_OBJ22 \
  (SDNPDATA_SUPPORT_OBJ22_V1 | \
   SDNPDATA_SUPPORT_OBJ22_V2 | \
   SDNPDATA_SUPPORT_OBJ22_V5 | \
   SDNPDATA_SUPPORT_OBJ22_V6)

/* Frozen Counter Events */
#define SDNPDATA_SUPPORT_OBJ23_V1 SDNPDATA_CNFG_LEVEL3
#define SDNPDATA_SUPPORT_OBJ23_V2 SDNPDATA_CNFG_LEVEL3
#define SDNPDATA_SUPPORT_OBJ23_V5 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ23_V6 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ23 \
    (SDNPDATA_SUPPORT_OBJ23_V1 | \
     SDNPDATA_SUPPORT_OBJ23_V2 | \
     SDNPDATA_SUPPORT_OBJ23_V5 | \
     SDNPDATA_SUPPORT_OBJ23_V6)

/* Analog Inputs */
/* If you support for Variation 3, you should also provide support
 * for variation 1. Otherwise if flags are not nominal it won't be
 * possible to send back the flags as required by Technical Bulletin
 * TB 2003-002 Object Flags.
 * For the same reason if Variation 4 is supported, Variation 2 should be
 * supported.
 */
#define SDNPDATA_SUPPORT_OBJ30_V1 SDNPDATA_CNFG_LEVEL1
#define SDNPDATA_SUPPORT_OBJ30_V2 SDNPDATA_CNFG_LEVEL1
#define SDNPDATA_SUPPORT_OBJ30_V3 SDNPDATA_CNFG_LEVEL1
#define SDNPDATA_SUPPORT_OBJ30_V4 SDNPDATA_CNFG_LEVEL1
#define SDNPDATA_SUPPORT_OBJ30_V5 SDNPDATA_CNFG_SUPPORT_FLOAT
#define SDNPDATA_SUPPORT_OBJ30_V6 SDNPDATA_CNFG_SUPPORT_DOUBLE
#define SDNPDATA_SUPPORT_OBJ30 \
  (SDNPDATA_SUPPORT_OBJ30_V1 | \
   SDNPDATA_SUPPORT_OBJ30_V2 | \
   SDNPDATA_SUPPORT_OBJ30_V3 | \
   SDNPDATA_SUPPORT_OBJ30_V4 | \
   SDNPDATA_SUPPORT_OBJ30_V5 | \
   SDNPDATA_SUPPORT_OBJ30_V6)

/* Frozen Analog Inputs */
#ifndef SDNPDATA_SUPPORT_OBJ31
#define SDNPDATA_SUPPORT_OBJ31_V1 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ31_V2 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ31_V3 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ31_V4 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ31_V5 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ31_V6 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ31_V7 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ31_V8 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ31 \
  (SDNPDATA_SUPPORT_OBJ31_V1 | \
   SDNPDATA_SUPPORT_OBJ31_V2 | \
   SDNPDATA_SUPPORT_OBJ31_V3 | \
   SDNPDATA_SUPPORT_OBJ31_V4 | \
   SDNPDATA_SUPPORT_OBJ31_V5 | \
   SDNPDATA_SUPPORT_OBJ31_V6 | \
   SDNPDATA_SUPPORT_OBJ31_V7 | \
   SDNPDATA_SUPPORT_OBJ31_V8)
#endif

/* Analog Input Events */
#define SDNPDATA_SUPPORT_OBJ32_V1 SDNPDATA_CNFG_LEVEL1
#define SDNPDATA_SUPPORT_OBJ32_V2 SDNPDATA_CNFG_LEVEL1
#define SDNPDATA_SUPPORT_OBJ32_V3 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ32_V4 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ32_V5 SDNPDATA_CNFG_SUPPORT_FLOAT
#define SDNPDATA_SUPPORT_OBJ32_V6 SDNPDATA_CNFG_SUPPORT_DOUBLE
#define SDNPDATA_SUPPORT_OBJ32_V7 SDNPDATA_CNFG_SUPPORT_FLOAT
#define SDNPDATA_SUPPORT_OBJ32_V8 SDNPDATA_CNFG_SUPPORT_DOUBLE
#define SDNPDATA_SUPPORT_OBJ32 \
  (SDNPDATA_SUPPORT_OBJ32_V1 | \
   SDNPDATA_SUPPORT_OBJ32_V2 | \
   SDNPDATA_SUPPORT_OBJ32_V3 | \
   SDNPDATA_SUPPORT_OBJ32_V4 | \
   SDNPDATA_SUPPORT_OBJ32_V5 | \
   SDNPDATA_SUPPORT_OBJ32_V6 | \
   SDNPDATA_SUPPORT_OBJ32_V7 | \
   SDNPDATA_SUPPORT_OBJ32_V8)

/* Frozen Analog Input Events */
#ifndef SDNPDATA_SUPPORT_OBJ33
#define SDNPDATA_SUPPORT_OBJ33_V1 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ33_V2 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ33_V3 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ33_V4 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ33_V5 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ33_V6 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ33_V7 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ33_V8 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ33 \
  (SDNPDATA_SUPPORT_OBJ33_V1 | \
   SDNPDATA_SUPPORT_OBJ33_V2 | \
   SDNPDATA_SUPPORT_OBJ33_V3 | \
    SDNPDATA_SUPPORT_OBJ33_V4 | \
    SDNPDATA_SUPPORT_OBJ33_V5 | \
    SDNPDATA_SUPPORT_OBJ33_V6 | \
    SDNPDATA_SUPPORT_OBJ33_V7 | \
    SDNPDATA_SUPPORT_OBJ33_V8)
#endif

/* Analog Input Deadband */
#define SDNPDATA_SUPPORT_OBJ34_V1 SDNPDATA_CNFG_LEVEL_TMW
#define SDNPDATA_SUPPORT_OBJ34_V2 SDNPDATA_CNFG_LEVEL_TMW
#define SDNPDATA_SUPPORT_OBJ34_V3 SDNPDATA_CNFG_SUPPORT_FLOAT
#define SDNPDATA_SUPPORT_OBJ34 \
  (SDNPDATA_SUPPORT_OBJ34_V1 | \
   SDNPDATA_SUPPORT_OBJ34_V2 | \
   SDNPDATA_SUPPORT_OBJ34_V3)

/* Analog Output Status */
#define SDNPDATA_SUPPORT_OBJ40_V1 SDNPDATA_CNFG_LEVEL1
#define SDNPDATA_SUPPORT_OBJ40_V2 SDNPDATA_CNFG_LEVEL1
#define SDNPDATA_SUPPORT_OBJ40_V3 SDNPDATA_CNFG_SUPPORT_FLOAT
#define SDNPDATA_SUPPORT_OBJ40_V4 SDNPDATA_CNFG_SUPPORT_DOUBLE
#define SDNPDATA_SUPPORT_OBJ40 \
  (SDNPDATA_SUPPORT_OBJ40_V1 | \
   SDNPDATA_SUPPORT_OBJ40_V2 | \
   SDNPDATA_SUPPORT_OBJ40_V3 | \
   SDNPDATA_SUPPORT_OBJ40_V4)

/* Analog Output Control Block */
#define SDNPDATA_SUPPORT_OBJ41_V1 SDNPDATA_CNFG_LEVEL3
#define SDNPDATA_SUPPORT_OBJ41_V2 SDNPDATA_CNFG_LEVEL1
#define SDNPDATA_SUPPORT_OBJ41_V3 SDNPDATA_CNFG_SUPPORT_FLOAT
#define SDNPDATA_SUPPORT_OBJ41_V4 SDNPDATA_CNFG_SUPPORT_DOUBLE
#define SDNPDATA_SUPPORT_OBJ41 \
  (SDNPDATA_SUPPORT_OBJ41_V1 | \
   SDNPDATA_SUPPORT_OBJ41_V2 | \
   SDNPDATA_SUPPORT_OBJ41_V3 | \
   SDNPDATA_SUPPORT_OBJ41_V4)

/* If Analog Output Event support is not defined on the command line
 * set the configuration here
 */
#ifndef SDNPDATA_SUPPORT_OBJ42
/* Analog Output Events */
#define SDNPDATA_SUPPORT_OBJ42_V1 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ42_V2 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ42_V3 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ42_V4 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ42_V5 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ42_V6 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ42_V7 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ42_V8 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ42 \
  (SDNPDATA_SUPPORT_OBJ42_V1 | \
   SDNPDATA_SUPPORT_OBJ42_V2 | \
   SDNPDATA_SUPPORT_OBJ42_V3 | \
   SDNPDATA_SUPPORT_OBJ42_V4 | \
   SDNPDATA_SUPPORT_OBJ42_V5 | \
   SDNPDATA_SUPPORT_OBJ42_V6 | \
   SDNPDATA_SUPPORT_OBJ42_V7 | \
   SDNPDATA_SUPPORT_OBJ42_V8)
#endif

/* If Analog Output Command Event support is not defined on the command line
 * set the configuration here
 */
#ifndef SDNPDATA_SUPPORT_OBJ43
/* Analog Output Command Events */
#define SDNPDATA_SUPPORT_OBJ43_V1 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ43_V2 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ43_V3 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ43_V4 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ43_V5 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ43_V6 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ43_V7 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ43_V8 SDNPDATA_CNFG_LEVEL4
#define SDNPDATA_SUPPORT_OBJ43 \
  (SDNPDATA_SUPPORT_OBJ43_V1 | \
   SDNPDATA_SUPPORT_OBJ43_V2 | \
   SDNPDATA_SUPPORT_OBJ43_V3 | \
   SDNPDATA_SUPPORT_OBJ43_V4 | \
   SDNPDATA_SUPPORT_OBJ43_V5 | \
   SDNPDATA_SUPPORT_OBJ43_V6 | \
   SDNPDATA_SUPPORT_OBJ43_V7 | \
   SDNPDATA_SUPPORT_OBJ43_V8)
#endif

/* Time and Date */
#define SDNPDATA_SUPPORT_OBJ50_V1 SDNPDATA_CNFG_LEVEL1
/* Required for LAN Time Sync Procedure */
#define SDNPDATA_SUPPORT_OBJ50_V3 SDNPDATA_CNFG_LEVEL1

#ifndef SDNPDATA_SUPPORT_OBJ50_V4
/* Required for Indexed absolute time */
#define SDNPDATA_SUPPORT_OBJ50_V4 TMWDEFS_FALSE
#endif

#define SDNPDATA_SUPPORT_OBJ50 \
  (SDNPDATA_SUPPORT_OBJ50_V1 | \
   SDNPDATA_SUPPORT_OBJ50_V3 | \
   SDNPDATA_SUPPORT_OBJ50_V4)

/* File Transfer */
#ifndef SDNPDATA_SUPPORT_OBJ70
/* Note: Starting in version 3.29.0 this is no longer set to LEVEL_TMW by default.
 * You should only set this to TRUE if you intend to allow file access from an MDNP
 * device across a DNP Association.
 */
#define SDNPDATA_SUPPORT_OBJ70    TMWDEFS_FALSE
#endif

/* Read IIN bit support */
#define SDNPDATA_SUPPORT_OBJ80_READ SDNPDATA_CNFG_LEVEL3

/* If Data Set support is not defined on the command line
 * set the configuration here
 */
#ifndef SDNPDATA_SUPPORT_DATASETS

/* Data Set Prototype */
#define SDNPDATA_SUPPORT_OBJ85  TMWDEFS_FALSE

/* Data Set Descriptor */
#define SDNPDATA_SUPPORT_OBJ86_V1 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ86_V2 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ86_V3 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ86 \
  (SDNPDATA_SUPPORT_OBJ86_V1 | \
   SDNPDATA_SUPPORT_OBJ86_V2 | \
   SDNPDATA_SUPPORT_OBJ86_V3)

/* Data Set Present Value */
#define SDNPDATA_SUPPORT_OBJ87  TMWDEFS_FALSE

/* Data Set Snapshot Events */
#define SDNPDATA_SUPPORT_OBJ88  TMWDEFS_FALSE

/* Data Sets in general */
#define SDNPDATA_SUPPORT_DATASETS \
  (SDNPDATA_SUPPORT_OBJ85 | \
   SDNPDATA_SUPPORT_OBJ86 | \
   SDNPDATA_SUPPORT_OBJ87 | \
   SDNPDATA_SUPPORT_OBJ88)
#endif

/* FC 31 Activate Configuration response */
#ifndef SDNPDATA_SUPPORT_OBJ91
#define SDNPDATA_SUPPORT_OBJ91  SDNPDATA_CNFG_LEVEL_TMW
#endif

/* String Data */
#ifndef SDNPDATA_SUPPORT_OBJ110
#define SDNPDATA_SUPPORT_OBJ110 SDNPDATA_CNFG_LEVEL_TMW
#endif

/* String Events */
#ifndef SDNPDATA_SUPPORT_OBJ111
#define SDNPDATA_SUPPORT_OBJ111 SDNPDATA_CNFG_LEVEL_TMW
#endif

/* Virtual Terminal Output */
#ifndef SDNPDATA_SUPPORT_OBJ112
#define SDNPDATA_SUPPORT_OBJ112 SDNPDATA_CNFG_LEVEL_TMW
```

```
#endif

/* Virtual Terminal Events */
#ifndef SDNPDATA_SUPPORT_OBJ113
#define SDNPDATA_SUPPORT_OBJ113 SDNPDATA_CNFG_LEVEL_TMW
#endif

/* Extended String Data */
#ifndef SDNPDATA_SUPPORT_OBJ114
#define SDNPDATA_SUPPORT_OBJ114_V1 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ114_V2 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ114_V3 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ114_V4 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ114 \
  (SDNPDATA_SUPPORT_OBJ114_V1 | \
   SDNPDATA_SUPPORT_OBJ114_V2 | \
   SDNPDATA_SUPPORT_OBJ114_V3 | \
   SDNPDATA_SUPPORT_OBJ114_V4)
#endif

/* Set this to TMWDEFS_TRUE to reduce stack size requirements to
 * process extended strings. It requires the implementation of
 * sdnpdata_extStrRelease to inform the database that the pointer
 * returned by sdnpdata_extStrGetPtr is no longer in use.
 */
#define SDNPDATA_SUPPORT_OBJ114_MIN_STACK TMWDEFS_TRUE

/* Extended String Events */
#ifndef SDNPDATA_SUPPORT_OBJ115
#define SDNPDATA_SUPPORT_OBJ115_V1 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ115_V2 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ115_V3 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ115_V4 TMWDEFS_FALSE
#define SDNPDATA_SUPPORT_OBJ115 \
  (SDNPDATA_SUPPORT_OBJ115_V1 | \
   SDNPDATA_SUPPORT_OBJ115_V2 | \
   SDNPDATA_SUPPORT_OBJ115_V3 | \
   SDNPDATA_SUPPORT_OBJ115_V4)
#endif

/* If Secure Authentication support is not defined on the command line
 * set the configuration here
 * DNPCNFG_SUPPORT_AUTHENTICATION must also be defined appropriately
 * Security Statistics object groups 121 and 122 must be supported
 * if SAv5 is supported.
 */
#ifndef SDNPDATA_SUPPORT_OBJ120
#define SDNPDATA_SUPPORT_OBJ120 DNPCNFG_SUPPORT_AUTHENTICATION
#endif

/* Secure Authentication User Certificate for Asymmetric Remote Key Update */
#ifndef SDNPDATA_SUPPORT_OBJ120_V8
#define SDNPDATA_SUPPORT_OBJ120_V8 TMWDEFS_FALSE
#endif

/* Set this to TMWDEFS_FALSE to remove code used to support assign class
 * function code
 */
#define SDNPDATA_SUPPORT_ASSIGN TMWDEFS_TRUE

/* Set this to TMWDEFS_FALSE to remove code used to support scanning for events
 * If this is set to TMWDEFS_TRUE, the SCL can be configured to call the database
 * sdnpdata_xxxChanged functions periodically to see if values have changed and
 * if so to call the sdnpxxx_addEvent functions automatically.
 */
#define SDNPDATA_SUPPORT_EVENT_SCAN TMWDEFS_TRUE

/* Set this to TMWDEFS_FALSE to remove code for Identical Unsolicited Response
Retries.
 * If this is set to TMWDEFS_TRUE, the SCL can be configured to send Identical
 * Unsolicited Retries.
 * The DNP3 specification allows for either identical retries with the same events
 * and sequence number (identical in every octet including IIN bits), or a
 * regenerated retry which may contain new events,  different IIN bits and a
 * new sequence number. The master MUST be able to handle both.
 * If compiled in and configured the outstation will send identical retries until
 * the retries are exceeded or the retry series is cancelled by another read etc.
 */
#ifndef SDNPDATA_SUPPORT_IDENT_UNSOL_RETRY
#define SDNPDATA_SUPPORT_IDENT_UNSOL_RETRY TMWDEFS_FALSE
#endif
```

```
/* Set this to TMWDEFS_FALSE to remove code used to support
 * per point default static variation
 */
#define SDNPDATA_SUPPORT_STATIC_VAR_POINT TMWDEFS_TRUE

/* Set this to TMWDEFS_FALSE to remove code used to support
 * per point default event variation
 */
#define SDNPDATA_SUPPORT_EVENT_VAR_POINT TMWDEFS_TRUE

/* Set this to TMWDEFS_FALSE to remove code used to support
 * per point event mode (SOE or LAST)
 */
#ifndef SDNPDATA_SUPPORT_EVENT_MODE_POINT
#define SDNPDATA_SUPPORT_EVENT_MODE_POINT TMWDEFS_TRUE
#endif

/* Set this to TMWDEFS_FALSE to remove code used to support
 * per point membership in class0
 */
#define SDNPDATA_SUPPORT_CLASS0_POINT TMWDEFS_TRUE

/* Set this to TMWDEFS_FALSE to remove code used to call sdnpdata function when a
 * select is cancelled either because the timer expired or because another command
 * was received before the operate. Only set this to TMWDEFS_TRUE if your
 * implementation requires that the database is notified when the select is
cancelled.
 */
#ifndef SDNPDATA_SUPPORT_SELECT_CANCEL
#define SDNPDATA_SUPPORT_SELECT_CANCEL TMWDEFS_FALSE
#endif

/*  * Setting this to TMWDEFS_TRUE will cause the Source Code Library to handle
 * a master reread of the same file block.
 * To save memory and processing time, and to disable this resend of file
 * read response, set this to TMWDEFS_FALSE. The Database can still support
 * rereading of the same block.
 */
#define SDNPDATA_SUPPORT_FILE_REREAD TMWDEFS_FALSE

/* Application Layer Spec, Feb 2007, says to keep a copy of the last solicited
 * response that was sent. If a request with the same sequence number is
 * is received and all of the octets are identical, resend last response,
 * but do not process the duplicate request.
 * To save memory and processing time, and to disable this resend of response,
 * set this to TMWDEFS_FALSE. Even with this set to TMWDEFS_FALSE, duplicate
 * select and duplicate operate requests will be handled properly as required
 * by the conformance tests.
 */
#define SDNPDATA_KEEP_LAST_RESPONSE  TMWDEFS_FALSE

 /* Private XML format required for save/restore in Test Harness.
  * Can also be used for save/restore of SDNP .NET Library database.
  */
#ifndef SDNPDATA_SUPPORT_XML
#define SDNPDATA_SUPPORT_XML         TMWDEFS_FALSE
#endif

/* Set this parameter to TMWDEFS_TRUE to include support for generating
 * an XML document based on the current state of the DNP3 slave database.
 * This allows the generation of an XML document which conforms to the new
 * (version 2.11) DNP3 configuration schema.
 */
#ifndef SDNPDATA_SUPPORT_XML2
#define SDNPDATA_SUPPORT_XML2        TMWDEFS_FALSE
#endif

/* Set this parameter to TMWDEFS_TRUE to include support for per point data
 * called dnpData in the Device Profile schema. This includes values, quality,
 * and timestamps for the individual points in the database.
 */
#ifndef SDNPDATA_SUPPORT_XML2_DNPDATA
#define SDNPDATA_SUPPORT_XML2_DNPDATA TMWDEFS_FALSE
#endif

 /* Support file transfer read of the above gnerated XML "files" */
#ifndef SDNPDATA_SUPPORT_READ_XML
#define SDNPDATA_SUPPORT_READ_XML     TMWDEFS_FALSE
#endif
```

```
/* Do this down here so the support macros as defined in sdnpsesn.h */
#include "tmwscl/dnp/sdnpsesn.h"

/* Define types for functions below */
typedef TMWTYPES_USHORT (*SDNPDATA_QUANTITY_FUNC)(void *);
typedef void * (*SDNPDATA_GET_POINT_FUNC)(void *, TMWTYPES_USHORT);
typedef TMWTYPES_BOOL (*SDNPDATA_IS_IN_CLASS0_FUNC)(void *);

/* bit values for determining if particular control operation is supported
 * These are NOT the values that are sent in the DNP message
 */
typedef TMWTYPES_UCHAR SDNPDATA_CROB_CTRL;
#define SDNPDATA_CROB_CTRL_PULSE_ON      0x01
#define SDNPDATA_CROB_CTRL_PULSE_OFF     0x02
#define SDNPDATA_CROB_CTRL_LATCH_ON      0x04
#define SDNPDATA_CROB_CTRL_LATCH_OFF     0x08
#define SDNPDATA_CROB_CTRL_PAIRED_CLOSE  0x10
#define SDNPDATA_CROB_CTRL_PAIRED_TRIP   0x20

/* Data set modes, used in sdnpdata_datasetCreatePoint() function */
typedef enum {
  SDNPDATA_DATASET_MODE_WRITE,
  SDNPDATA_DATASET_MODE_SELECT,
  SDNPDATA_DATASET_MODE_OPERATE
} SDNPDATA_DATASET_MODE;

/* Data Set interface structure used for passing data set element
 * control values between SCL and target layer database.
 */
typedef struct {
  /* element index in data set for this control value     */
  TMWTYPES_UCHAR        elemIndex;
  /* control value received in select or operate request */
  DNPDATA_DATASET_VALUE data;
} DNPDATA_DATASET_CTRL_VALUE;

/* union of values used for adding events internally in the SCL and
 * to the database if user managed events is configured.
 */
typedef union {
  /* This field is used when adding a counter or frozen counter event
   * from database
   */
  TMWTYPES_ULONG           ulValue;

  /* This field is used when adding a analog input event to database */
  TMWTYPES_ANALOG_VALUE   *analogPtr;

  /* This field is used when adding a string or vterm event to database */
  struct {
    TMWTYPES_UCHAR         length;
    TMWTYPES_UCHAR        *pBuf;
  } stringPtr;

  /* This field is used when adding an extended string event to database */
  struct {
    TMWTYPES_USHORT        length;
    TMWTYPES_UCHAR        *pBuf;
  } extendedStringPtr;

  struct {
    TMWTYPES_UCHAR         numberElems;
    DNPDATA_DATASET_VALUE *pData;
  } dataset;

  struct {
    TMWTYPES_USHORT        assocId;
    TMWTYPES_ULONG         sequenceNumber;
    TMWTYPES_UCHAR         errorCode;
    TMWTYPES_CHAR         *pErrorText;
    TMWTYPES_USHORT        errorTextLength;
  } authError;

  struct {
    TMWTYPES_USHORT        assocId;
    TMWTYPES_ULONG         ulValue;
  } authSecStat;

} SDNPDATA_ADD_EVENT_VALUE;

#ifdef __cplusplus
```

```
extern "C" {
#endif

  /* function: sdnpdata_getIIN
   * purpose: Called before each response is sent to allow the user
   *  to specify the current device IIN bits. The SCL will OR in additional
   *  IIN bits maintained by the SCL but will never clear a bit set
   *  in this function.
   * The following bits should be set by the user if appropriate:
   *  DNPDEFS_IIN_TROUBLE     - should be set when an abnormal condition exists,
   *                            such as hardware problems. Only set this if
   *                            another IIN bit does not indicate this condition.
   *  DNPDEFS_IIN_LOCAL       - should be set if any output point is in the local
   *                            operation mode
   *  DNPDEFS_IIN_BAD_CONFIG  - should be set when a corrupt configuration is
   *                            detected. Setting this bit is optional.
   * The following bits are managed by the SCL and should generally not
   * be set by this routine:
   *  DNPDEFS_IIN_RESTART             see NOTE 1
   *  DNPDEFS_IIN_NEED_TIME
   *  DNPDEFS_IIN_CLASS_3
   *  DNPDEFS_IIN_CLASS_2
   *  DNPDEFS_IIN_CLASS_1
   *  DNPDEFS_IIN_ALL_STATIONS
   *  DNPDEFS_IIN_ALREADY_EXECUTING
   *  DNPDEFS_IIN_BUFFER_OVFL         set NOTE 1
   *  DNPDEFS_IIN_OUT_OF_RANGE
   *  DNPDEFS_IIN_OBJECT_UNKNOWN
   *  DNPDEFS_IIN_BAD_FUNCTION
   *  NOTE 1: user does not normally, but may set, IIN_RESTART and IIN_BUFFER_OVFL.
   *   The SCL will manage the clearing of these two bits when appropriate.
   *  NOTE 2: *pIIN is zero when this function is called. You are free to leave it
   *   as zero or set any bits you choose.
   * arguments:
   *  pSession - pointer to session
   *  pIIN - pointer in which to store IIN bits
   * returns:
   *  void
   */
  void TMWDEFS_GLOBAL sdnpdata_getIIN(
    TMWSESN *pSession,
    TMWTYPES_USHORT *pIIN);

  /* function: sdnpdata_IINQuantity
   * purpose: Determine how many IIN bits are supported on this session
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   * returns:
   *  16 would indicate no private IIN bits are supported
   *  any number > 16 would indicate how many total IIN bits,
   *   (standard(16) + private bits) are supported.
   */
  TMWTYPES_USHORT TMWDEFS_GLOBAL sdnpdata_IINQuantity(
    void *pHandle);

  /* function: sdnpdata_IINRead
   * purpose: Read private IIN bit because of read ObjectGroup 80 request
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  pointNumber - index of IIN bit to read, 0-15 are standard IIN bits
   * returns:
   *  TMWTYPES_BOOL - TMWTYPES_BOOL if bit is set.
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_IINRead(
    void *pHandle,
    TMWTYPES_USHORT pointNumber);

  /* function: sdnpdata_coldRestart
   * purpose: Perform cold restart. Note that this function can NOT alter the
   *  SCL context and return since it will be called from within the SCL and
   *  any changes will result in undefined behavior in the SCL. The options
   *  are to perform a hard reset (i.e. reset the program counter, stack,
   *  etc.) and never return or set a flag to be processed in the user code
   *  after returning from the SCL.
   * arguments:
   *  pSession - pointer to session on which cold restart request was received
   * returns:
   *  void
   */
  void TMWDEFS_GLOBAL sdnpdata_coldRestart(
    TMWSESN *pSession);
```

```
/* function: sdnpdata_warmRestart
 * purpose: Perform warm restart. Note that this function can NOT alter the
 *  SCL context and return since it will be called from within the SCL and
 *  any changes will result in undefined behavior in the SCL. The options
 *  are to perform a hard reset (i.e. reset the program counter, stack,
 *  etc.) and never return or set a flag to be processed in the user code
 *  after returning from the SCL.
 * arguments:
 *  pSession - pointer to session on which warm restart request was received
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL sdnpdata_warmRestart(
  TMWSESN *pSession);

/* function: sdnpdata_init
 * purpose: Initialize DNP3 slave database for specified session
 * arguments:
 *  pSession - pointer to session
 *  pUserHandle - User provided handle passed into sdnpsesn_openSession()
 *    This handle could be used to identify the database associated with
 *    a particular session. The SCL does not use this handle internally.
 * returns:
 *  pointer to NON NULL database handle. This handle will be passed into
 *   other sdnpdata_xxx functions and only needs to be meaningful to the
 *   those routines. If multiple sessions and therefore databases are opened.
 *   This handle could be used to determine which database (which session)
 *   this function is being called for. The SCL will not use this handle
 *  The SCL will not use this handle internally.
 *  TMWDEFS_NULL indicates failure.
 */
void * TMWDEFS_GLOBAL sdnpdata_init(
  TMWSESN *pSession,
  void *pUserHandle);

/* function: sdnpdata_close
 * purpose: Close DNP3 slave database
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL sdnpdata_close(
  void *pHandle);

/* function: sdnpdata_setTime
 * purpose: Set the time because a write time request has been
 *  received from the master.  Default behavior is to set the clock.
 *  Target implementations may only want 1 session on a device to set the time.
 *  Otherwise multiple masters might set the time differently.
 *  There is also a session configuration parameter "respondNeedTime" that
 *  controls whether a session will set the need time IIN bit, requesting a
 *  time sync from the master.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  pNewTime - pointer to a time structure containing the time sent by the master
 */
void sdnpdata_setTime(
  void *pHandle,
  TMWDTIME *pNewTime);

/* function: sdnpdata_unsolEventMask
 * purpose: Indicate slave has received a function code enable unsolicited
 *  or function code disable unsolicited request from the master.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  unsolEventMask - mask containing three bits indicating which classes are
 *   now enabled for unsolicited responses.
 *    TMWDEFS_CLASS_MASK_ONE
 *    TMWDEFS_CLASS_MASK_TWO
 *    TMWDEFS_CLASS_MASK_THREE
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL sdnpdata_unsolEventMask(
  void *pHandle,
  TMWDEFS_CLASS_MASK unsolEventMask);

/* function: sdnpdata_eventAndStaticRead
 * purpose: Indicate that a read of events and static data is
 *  beginning or ending. It may be desirable to prevent changes
 *  to the values of static data in the database while a read is
 *  in progress. This can be used to guarantee that events are reported
 *  before the new changed values in the static data. This is not
 *  an issue if events only are read, since another read will be required
 *  to get any remaining events in the queue before the static values are
 *  read. NOTE: Reading static data and events in separate reads is discouraged
 *  as it is even more likely that the static data would be read before the
 *  queued events.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  inProgress - TMWDEFS_TRUE if read is in progress (beginning)
 *               TMWDEFS_FALSE if read is ended
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL sdnpdata_eventAndStaticRead(
  void *pHandle,
  TMWTYPES_BOOL inProgress);

/* function: sdnpdata_funcCode
 * purpose: Indicate that a request has been received from the master and that
 *  the specified function code processing is being started or has completed.
 *  The database may want to accumulate the individual points and data from
 *  a request before acting on them.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  functionCode - function code being requested.
 *                 (DNPDEFS_FC_WRITE, DNPDEFS_FC_DIRECT_OP_NOACK etc)
 *  inProgress - TMWDEFS_TRUE if request is in progress (beginning)
 *               TMWDEFS_FALSE if request is ended
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL sdnpdata_funcCode(
  void *pHandle,
  char functionCode,
  TMWTYPES_BOOL inProgress);

/* Binary Inputs */

/* function: sdnpdata_binInGetDescription
 * purpose: Get description of this point
 *  NOTE: this is used only by sdnpxml/sdnpxml2.c if SDNPDATA_SUPPORT_XMLxx is
TRUE
 *  code to generate the configuration file or Device Profile for this device.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  pointer to string describing point
 *  TMWDEFS_NULL if failure
 */
TMWTYPES_CHAR * TMWDEFS_GLOBAL sdnpdata_binInGetDescription(
  void *pPoint);

/* function: sdnpdata_binInQuantity
 * purpose: Return the number of binary input data points in the
 *  specified database.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 * returns:
 *  The number of binary input data points (see note for
 *  sdnpdata_xxxQuantity at top of this file)
 */
TMWTYPES_USHORT TMWDEFS_CALLBACK sdnpdata_binInQuantity(
  void *pHandle);

/* function: sdnpdata_binInGetPoint
 * purpose: Return a handle to a specific data point. This handle is
 *  used by the routines below to read and write values and control
 *  information to this point.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  pointNum - point number to return
 * returns:
 *  Pointer to specified data point or TMWDEFS_NULL if this point
 *   is currently disabled.
 */
void * TMWDEFS_CALLBACK sdnpdata_binInGetPoint(
  void *pHandle,
  TMWTYPES_USHORT pointNum);
```

```
/* function: sdnpdata_binInDefVariation
 * purpose: Determine default static variation for this binary input
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  default variation for this point
 *  NOTE: this will only be called if the default variation for binary
 *    inputs obj01DefaultVariation is configured as zero
 */
TMWTYPES_UCHAR TMWDEFS_GLOBAL sdnpdata_binInDefVariation(
  void *pPoint);

/* function: sdnpdata_binInEventClass
 * purpose: Return the class in which events from this data point
 *  belong.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  Class in which these events will be returned
 */
TMWDEFS_CLASS_MASK TMWDEFS_GLOBAL sdnpdata_binInEventClass(
  void *pPoint);

/* function: sdnpdata_binInIsClass0
 * purpose: Should this point be reported in response to an object 60
 *   variation 1 read request. This allows individual points to be excluded
 *   from a class 0 response but still readable by a specific object group
 *   read request.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  TMWDEFS_TRUE if point should be reported.
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_binInIsClass0(
  void *pPoint);

/* function: sdnpdata_binInEventDefVariation
 * purpose: Determine default variation for this binary input
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  classMask - mask indicating what event class is being requested
 * returns:
 *  default variation for this point
 *  NOTE: this will only be called if the default variation for binary
 *    input change events obj02DefaultVariation is configured as zero
 */
TMWTYPES_UCHAR TMWDEFS_GLOBAL sdnpdata_binInEventDefVariation(
  void *pPoint,
  TMWDEFS_CLASS_MASK classMask);

/* function: sdnpdata_binInEventMode
 * purpose: Determine event mode for this point
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  event mode for this point, SOE or LAST
 *  NOTE: this will only be called if the event mode for binary inputs
 *    if binaryInputEventMode is configured as TMWDEFS_EVENT_MODE_PER_POINT
 */
TMWDEFS_EVENT_MODE TMWDEFS_GLOBAL sdnpdata_binInEventMode(
  void *pPoint);

/* function: sdnpdata_binInAssignClass
 * purpose: Assign the class in which events from this data point
 *  will belong.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  classMask - new class in which to generate events from this point
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_binInAssignClass(
  void *pPoint,
  TMWDEFS_CLASS_MASK classMask);

/* function: sdnpdata_binInRead
 * purpose: Return the current value of the specified point.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  pFlags - pointer to location to store current DNP3 flags and value.
 *   pFlags contains a status indication and the current state of the point.
 *   The following values (or OR'd combinations) are valid for this type:
 *      DNPDEFS_DBAS_FLAG_OFF_LINE - the point is off-line, and the returned
 *        state of this point may not be correct
 *      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)
 *      DNPDEFS_DBAS_FLAG_BINARY_OFF - the current state of the input (Off)
 * returns:
 *  void
 */
TMWDEFS_SCL_API void TMWDEFS_GLOBAL sdnpdata_binInRead(
  void *pPoint,
  TMWTYPES_UCHAR *pFlags);

/* function: sdnpdata_binInChanged
 * purpose: Determine if the specified point has changed and if so
 *  return the new value. This function is used to scan for events
 *  on each data point. It will be called if SDNPSESN_CONFIG
 *  binaryInputScanPeriod is nonzero.
 *  NOTE: this functionality is compiled out by defining
 *  SDNPDATA_SUPPORT_OBJ2 FALSE
 *  or SDNPDATA_SUPPORT_EVENT_SCAN FALSE
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  pFlags - pointer to location to store current DNP3 flags and value
 *   pFlags contains a status indication and the current state of the point.
 *   The following values (or OR'd combinations) are valid for this type:
 *      DNPDEFS_DBAS_FLAG_OFF_LINE - the point is off-line, and the returned
 *        state of this point may not be correct
 *      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)
 *      DNPDEFS_DBAS_FLAG_BINARY_OFF - the current state of the input (Off)
 * returns:
 *  TMWDEFS_TRUE if the point has changed, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_binInChanged(
  void *pPoint,
  TMWTYPES_UCHAR *pFlags);

/* Binary Outputs */

/* function: sdnpdata_binOutGetDescription
 * purpose: Get description of this point
 *  NOTE: this is used only by sdnpxml/sdnpxml2.c if SDNPDATA_SUPPORT_XMLxx is
TRUE
 *  code to generate the configuration file or Device Profile for this device.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  pointer to string describing point
 *  TMWDEFS_NULL if failure
 */
TMWTYPES_CHAR * TMWDEFS_GLOBAL sdnpdata_binOutGetDescription(
  void *pPoint);
```

```
/* function: sdnpdata_binOutQuantity
 * purpose: Return the number of binary output data points in the
 *  specified database.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 * returns:
 *  The number of binary output data points (see note for
 *  sdnpdata_xxxQuantity at top of this file)
 */
TMWTYPES_USHORT TMWDEFS_GLOBAL sdnpdata_binOutQuantity(
  void *pHandle);

/* function: sdnpdata_binOutGetPoint
 * purpose: Return a handle to a specific data point. This handle is
 *  used by the routines below to read and write values and control
 *  information to this point.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  pointNum - point number to return
 * returns:
 *  Pointer to specified data point or TMWDEFS_NULL if this point
 *   is currently disabled.
 */
void * TMWDEFS_GLOBAL sdnpdata_binOutGetPoint(
  void *pHandle,
  TMWTYPES_USHORT pointNum);

/* function: sdnpdata_binOutDefVariation
 * purpose: Determine default static variation for this binary output status
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  default variation for this point
 *  NOTE: this will only be called if the default variation for binary
 *   inputs obj10DefaultVariation is configured as zero
 */
TMWTYPES_UCHAR TMWDEFS_GLOBAL sdnpdata_binOutDefVariation(
  void *pPoint);

/* function: sdnpdata_binOutEventClass
 * purpose: Return the class in which binary output events from this
 *  data point belong.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  Class in which these events will be returned
 */
TMWDEFS_CLASS_MASK TMWDEFS_GLOBAL sdnpdata_binOutEventClass(
  void *pPoint);

/* function: sdnpdata_binOutIsClass0
 * purpose: Should this point be reported in response to an object 60
 *   variation 1 read request. This allows individual points to be excluded
 *   from a class 0 response but still readable by a specific object group
 *   read request.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  TMWDEFS_TRUE if point should be reported.
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_binOutIsClass0(
  void *pPoint);

/* function: sdnpdata_binOutEventDefVariation
 * purpose: Determine default event variation for this binary output point
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  classMask - mask indicating what event class is being requested
 * returns:
 *  default variation for this point
 *  NOTE: this will only be called if the default variation for binary
 *   output events obj11DefaultVariation is configured as zero
 */
TMWTYPES_UCHAR TMWDEFS_GLOBAL sdnpdata_binOutEventDefVariation(
  void *pPoint,
  TMWDEFS_CLASS_MASK classMask);

/* function: sdnpdata_binOutEventMode
 * purpose: Determine event mode for this point
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  event mode for this point, SOE or LAST
 *  NOTE: this will only be called if the event mode for binary outputs
 *   if binaryOutputEventMode is configured as TMWDEFS_EVENT_MODE_PER_POINT
 */
TMWDEFS_EVENT_MODE TMWDEFS_GLOBAL sdnpdata_binOutEventMode(
  void *pPoint);

/* function: sdnpdata_binOutAssignClass
 * purpose: Assign the class in which binary output events from this
 *  data point will belong.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  classMask - new class in which to generate events from this point
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_binOutAssignClass(
  void *pPoint,
  TMWDEFS_CLASS_MASK classMask);

/* function: sdnpdata_binOutRead
 * purpose: Return the current value of the specified point.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  pFlags - pointer to location to store current DNP3 flags and value
 *   pFlags contains a status indication and the current state of the point.
 *   The following values (or OR'd combinations) are valid for this type:
 *     DNPDEFS_DBAS_FLAG_OFF_LINE - the point is off-line, and the returned
 *       state of this point may not be correct
 *       NOTE: if the point or device is in local mode the point should be OFF_LINE
 *     DNPDEFS_DBAS_FLAG_ON_LINE - the binary output 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 device
 *       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
 *       NOTE: this bit is NOT related to the point being in LOCAL MODE.
 *     DNPDEFS_DBAS_FLAG_BINARY_ON  - the current state of the output (On)
 *     DNPDEFS_DBAS_FLAG_BINARY_OFF - the current state of the output (Off)
 * returns:
 *  void
 */
TMWDEFS_SCL_API void TMWDEFS_GLOBAL sdnpdata_binOutRead(
  void *pPoint,
  TMWTYPES_UCHAR *pFlags);

/* function: sdnpdata_binOutChanged
 * purpose: Determine if the specified point has changed and if so
 *  return the new value. This function is used to scan for events
 *  on each data point. It will be called if SDNPSESN_CONFIG
 *  binaryOutputScanPeriod is nonzero.
 *  NOTE: this functionality is compiled out by defining
 *  SDNPDATA_SUPPORT_OBJ11 FALSE  (default)
 *  or SDNPDATA_SUPPORT_EVENT_SCAN FALSE
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  pFlags - pointer to location to store current DNP3 flags and value
 *   pFlags contains a status indication and the current state of the point.
 *   The following values (or OR'd combinations) are valid for this type:
 *     DNPDEFS_DBAS_FLAG_OFF_LINE - the point is off-line, and the returned
 *       state of this point may not be correct
 *     DNPDEFS_DBAS_FLAG_ON_LINE - the binary output 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 device
 *       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 output (On)
 *      DNPDEFS_DBAS_FLAG_BINARY_OFF - the current state of the output (Off)
 * returns:
 *  TMWDEFS_TRUE if the point has changed, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_binOutChanged(
  void *pPoint,
  TMWTYPES_UCHAR *pFlags);

/* function: sdnpdata_binOutWrite
 * purpose: perform write operation on the specified binary output point and
 *   return the result. Write to object group 10 variation 1 was received.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  value - Value to be written. Does not contain status.
 *      DNPDEFS_DBAS_FLAG_BINARY_ON  - set the output to (On)
 *      DNPDEFS_DBAS_FLAG_BINARY_OFF - set the output to (Off)
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_binOutWrite(
  void *pPoint,
  TMWTYPES_UCHAR value);

/* function: sdnpdata_binOutGetControlMask
 * purpose: Determine what control operations this particular point supports.
 *  NOTE: this is used only by sdnpxml/sdnpxml2.c if SDNPDATA_SUPPORT_XMLxx is TRUE
 *  code to generate the configuration file or Device Profile for this device.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  bitmask indicating what controls are allowed.
 *   SDNPDATA_CROB_CTRL_PULSE_ON
 *   SDNPDATA_CROB_CTRL_PULSE_OFF
 *   SDNPDATA_CROB_CTRL_LATCH_ON
 *   SDNPDATA_CROB_CTRL_LATCH_OFF
 *   SDNPDATA_CROB_CTRL_PAIRED_CLOSE
 *   SDNPDATA_CROB_CTRL_PAIRED_TRIP
 */
SDNPDATA_CROB_CTRL TMWDEFS_GLOBAL sdnpdata_binOutGetControlMask(
  void *pPoint);

/* function: sdnpdata_binOutSelect
 * purpose: perform Select operation on the specified point and return
 *   the result.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  controlCode - the control function to perform. The Control Code
 *   may contained OR'd values of the following values:
 *     DNPDEFS_CROB_CTRL_PULSE_ON - The point(s) is turned on for the
 *      specified onTime (activationPeriod), turned off for the specified
 *       offTime, and leftin the off state.
 *     DNPDEFS_CROB_CTRL_PULSE_OFF - The point(s) is turned off for the
 *      specific offTime (deactivationPeriod), then turned on for the specified
 *      onTime, and left in the on state.
 *     DNPDEFS_CROB_CTRL_LATCH_ON - The point(s) is latched on
 *     DNPDEFS_CROB_CTRL_LATCH_OFF - The point(s) is latched off
 *     DNPDEFS_CROB_CTRL_QUEUE - place the operation back in the control
 *      queue when complete
 *     DNPDEFS_CROB_CTRL_CLEAR - cancel the currently running operation and
 *      remove queued operations on affected points immediately before
 *      activating this new operation
 *     DNPDEFS_CROB_CTRL_PAIRED_CLOSE - activate Close relay
 *     DNPDEFS_CROB_CTRL_PAIRED_TRIP - activate Trip relay
 *     DNPDEFS_CROB_CTRL_PAIRED_CLOSE and DNPDEFS_CROB_CTRL_PAIRED_TRIP are
 *      used in systems where a Trip and Close relay pair is used to energize
 *      and de-energize  the field points. Both of these bits can be 0 to
 *      activate the field point select relay only, without activating the
 *      Trip/Close relays. In a system without field point relays, if both
 *      of these bits are 0, then no control operation should be performed.
 *      In a system without Trip/Close relays, both of these bits should
 *      always be 0 to indicate normal digital control operations. It is
 *      invalid for DNPDEFS_CROB_CTRL_PAIRED_TRIP and
 *      DNPDEFS_CROB_CTRL_PAIRED_CLOSE to both be set.
 *  count - the number of times that the control operation should be
 *    performed in succession. If count is 0, do not execute the control.
 *  onTime - amount of time (in ms) the digital output is to be turned on
 *     (may not apply to all control types)
 *  offTime - amount of time (in ms) the digital output is to be turned off
 *     (may not apply to all control types)
 * returns:
 *  status of CROB operation. Valid values are:
```

```
 *    DNPDEFS_CROB_ST_SUCCESS - the command was performed successfully
 *    DNPDEFS_CROB_ST_FORMAT_ERROR - the request was not accepted due
 *      to formatting errors in the request
 *    DNPDEFS_CROB_ST_NOT_SUPPORTED - the request was not accepted because
 *      Control operations are not supported for this point
 *    DNPDEFS_CROB_ST_ALREADY_ACTIVE - the request was not accepted because
 *      the control queue is full or the point is already active
 *    DNPDEFS_CROB_ST_HARDWARE_ERROR - the request was not accepted due to
 *      control hardware problems
 *    DNPDEFS_CROB_ST_LOCAL - the request was not accepted because
 *      Local/Remote switch is in Local Position
 *    DNPDEFS_CROB_ST_NOT_AUTHORIZED - the request was not accepted because
 *      of insufficient authorization.
 *    DNPDEFS_CROB_ST_AUTO_INHIBIT - the request was not accepted because it
 *      was prevented or inhibited by a local automation process
 *    DNPDEFS_CROB_ST_PROC_LIMITED - the request was not accepted because the
 *     device cannot process any more activities than are presently in progress
 *    DNPDEFS_CROB_ST_OUT_OF_RANGE - the request was not accepted because the
 *     value is outside the acceptable range permitted for this point.
 *    DNPDEFS_CROB_ST_DOWNSTRM_LOCAL - the request was not accepted because the
 *     control is being forwarded to a downstream device that is reporting Local.
 *    DNPDEFS_CROB_ST_ALR_COMPLETE - the request was not accepted because the
 *     operation is already complete. For example if the request is to close a
 *     switch and the switch is already closed.
 *    DNPDEFS_CROB_ST_BLOCKED - the request was not accepted because it is
 *     specifically blocked at the outstation.
 *    DNPDEFS_CROB_ST_CANCELLED - the request was not accepted because the
 *     operation was cancelled.
 *    DNPDEFS_CROB_ST_BLOCKED_OM - the request was not accepted because another
 *     master has exclusive rights to operate this point.
 *    DNPDEFS_CROB_ST_DOWNSTRM_FAIL -  the request was not accepted because the
 *     control is being forwarded to a downstream device which cannot be reached
 *     or is otherwise incapable of performing the request.
 *    DNPDEFS_CROB_ST_UNDEFINED - the request not accepted because of some
 *     other undefined reason
 */
DNPDEFS_CROB_ST TMWDEFS_GLOBAL sdnpdata_binOutSelect(
  void *pPoint,
  TMWTYPES_UCHAR controlCode,
  TMWTYPES_UCHAR count,
  TMWTYPES_ULONG onTime,
  TMWTYPES_ULONG offTime);

/* function: sdnpdata_binOutCancelSelect
 * purpose: cancel outstanding Select on the specified point
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL sdnpdata_binOutCancelSelect(
  void *pPoint);

/* function: sdnpdata_binOutOperate
 * purpose: perform Operate operation on the specified point and return
 *   the result. If Object 13 Binary Output Command Events are desired
 *   you may call sdnpo013_addEvent() from this function.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  controlCode - the control function to perform. The Control Code
 *   may contained OR'd values of the following values:
 *     DNPDEFS_CROB_CTRL_PULSE_ON - The point(s) is turned on for the
 *      specified onTime, turned off for the specified offTime, and left
 *      in the off state.
 *     DNPDEFS_CROB_CTRL_PULSE_OFF - The point(s) is turned off for the
 *      specific offTime, then turned on for the specified onTime, and left
 *      in the on state.
 *     DNPDEFS_CROB_CTRL_LATCH_ON - The point(s) is latched on
 *     DNPDEFS_CROB_CTRL_LATCH_OFF - The point(s) is latched off
 *     DNPDEFS_CROB_CTRL_QUEUE - place the operation back in the control
 *      queue when complete. This bit is obsolete, outstations that receive
 *      this should return a status code DNPDEFS_CROB_ST_NOT_SUPPORTED.
 *     DNPDEFS_CROB_CTRL_CLEAR - cancel the currently running operation and
 *      remove queued operations on affected points immediately before
 *      activating this new operation
 *     DNPDEFS_CROB_CTRL_PAIRED_CLOSE - activate Close relay
 *     DNPDEFS_CROB_CTRL_PAIRED_TRIP - activate Trip relay
 *     DNPDEFS_CROB_CTRL_PAIRED_CLOSE and DNPDEFS_CROB_CTRL_PAIRED_TRIP are
 *      used in systems where a Trip and Close relay pair is used to energize
 *      and de-energize  the field points. Both of these bits can be 0 to
 *      activate the field point select relay only, without activating the
 *      Trip/Close relays. In a system without field point relays, if both
 *      of these bits are 0, then no control operation should be performed.
 *      In a system without Trip/Close relays, both of these bits should
 *      always be 0 to indicate normal digital control operations. It is
 *      invalid for DNPDEFS_CROB_CTRL_PAIRED_TRIP and
 *      DNPDEFS_CROB_CTRL_PAIRED_CLOSE to both be set.
 *  count - the number of times that the control operation should be
 *    performed in succession. If count is 0, do not execute the control.
 *  onTime - amount of time (in ms) the digital output is to be turned on
 *    (may not apply to all control types)
 *  offTime - amount of time (in ms) the digital output is to be turned off
 *    (may not apply to all control types)
 * returns:
 *  status of CROB operation. Valid values are:
 *    DNPDEFS_CROB_ST_SUCCESS - the command was performed successfully
 *    DNPDEFS_CROB_ST_TIMEOUT - the request was not accepted because the
 *      Operate message was received after the Arm timer timed out. The
 *      Arm timer is started when a Select operation for the same point
 *      was received.
 *    DNPDEFS_CROB_ST_NO_SELECT - No previously matching select message
 *      (i.e., an Operate message was sent to activate a control point
 *      that was not previously armed with a Select message.)
 *    DNPDEFS_CROB_ST_FORMAT_ERROR - the request was not accepted due
 *      to formatting errors in the request
 *    DNPDEFS_CROB_ST_NOT_SUPPORTED - the request was not accepted because
 *      Control operations are not supported for this point
 *    DNPDEFS_CROB_ST_ALREADY_ACTIVE - the request was not accepted because
 *      the control queue is full or the point is already active
 *    DNPDEFS_CROB_ST_HARDWARE_ERROR - the request was not accepted due to
 *      control hardware problems
 *    DNPDEFS_CROB_ST_LOCAL - the request was not accepted because
 *      Local/Remote switch is in Local Position
 *    DNPDEFS_CROB_ST_NOT_AUTHORIZED - the request was not accepted because
 *      of insufficient authorization.
 *    DNPDEFS_CROB_ST_AUTO_INHIBIT - the request was not accepted because it
 *      was prevented or inhibited by a local automation process
 *    DNPDEFS_CROB_ST_PROC_LIMITED - the request was not accepted because the
 *     device cannot process any more activities than are presently in progress
 *    DNPDEFS_CROB_ST_OUT_OF_RANGE - the request was not accepted because the
 *     value is outside the acceptable range permitted for this point.
 *    DNPDEFS_CROB_ST_DOWNSTRM_LOCAL - the request was not accepted because the
 *     control is being forwarded to a downstream device that is reporting Local.
 *    DNPDEFS_CROB_ST_ALR_COMPLETE - the request was not accepted because the
 *     operation is already complete. For example if the request is to close a
 *     switch and the switch is already closed.
 *    DNPDEFS_CROB_ST_BLOCKED - the request was not accepted because it is
 *     specifically blocked at the outstation.
 *    DNPDEFS_CROB_ST_CANCELLED - the request was not accepted because the
 *     operation was cancelled.
 *    DNPDEFS_CROB_ST_BLOCKED_OM - the request was not accepted because another
 *     master has exclusive rights to operate this point.
 *    DNPDEFS_CROB_ST_DOWNSTRM_FAIL -  the request was not accepted because the
 *     control is being forwarded to a downstream device which cannot be reached
 *     or is otherwise incapable of performing the request.
 *    DNPDEFS_CROB_ST_UNDEFINED - the request not accepted because of some
 *     other undefined reason
 */
DNPDEFS_CROB_ST TMWDEFS_GLOBAL sdnpdata_binOutOperate(
  void *pPoint,
  TMWTYPES_UCHAR controlCode,
  TMWTYPES_UCHAR count,
  TMWTYPES_ULONG onTime,
  TMWTYPES_ULONG offTime);

/* function: sdnpdata_binOutSelPatternMask
 * purpose: perform Select Pattern Mask operation on the specified point
 *  and return the result.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  controlCode - the control function to perform. The Control Code
 *   may contained OR'd values of the following values:
 *     DNPDEFS_CROB_CTRL_PULSE_ON - The point(s) is turned on for the
 *      specified onTime, turned off for the specified offTime, and left
 *      in the off state.
 *     DNPDEFS_CROB_CTRL_PULSE_OFF - The point(s) is turned off for the
 *      specific offTime, then turned on for the specified onTime, and left
 *      in the on state.
 *     DNPDEFS_CROB_CTRL_LATCH_ON - The point(s) is latched on
 *     DNPDEFS_CROB_CTRL_LATCH_OFF - The point(s) is latched off
 *     DNPDEFS_CROB_CTRL_CLEAR - cancel the currently running operation and
 *      remove queued operations on affected points immediately before
 *      activating this new operation
 *     DNPDEFS_CROB_CTRL_PAIRED_CLOSE - activate Close relay
 *     DNPDEFS_CROB_CTRL_PAIRED_TRIP - activate Trip relay
 *     DNPDEFS_CROB_CTRL_PAIRED_CLOSE and DNPDEFS_CROB_CTRL_PAIRED_TRIP are
 *      used in systems where a Trip and Close relay pair is used to energize
 *      and de-energize  the field points. Both of these bits can be 0 to
 *      activate the field point select relay only, without activating the
 *      Trip/Close relays. In a system without field point relays, if both
 *      of these bits are 0, then no control operation should be performed.
 *      In a system without Trip/Close relays, both of these bits should
 *      always be 0 to indicate normal digital control operations. It is
 *      invalid for DNPDEFS_CROB_CTRL_PAIRED_TRIP and
 *      DNPDEFS_CROB_CTRL_PAIRED_CLOSE to both be set.
 *  count - the number of times that the control operation should be
 *    performed in succession. If count is 0, do not execute the control.
 *  activationPeriod - amount of time the digital output is to be turned on
 *    (may not apply to all control types)
 *  deactivationPeriod - amount of time the digital output is to be turned off
 *    (may not apply to all control types)
 *  firstPointNumber - first point number represented in pattern mask (*pMask).
 *  lastPointNumber - last point number represented in the pattern mask (*pMask).
 *  pMask - pointer to an array of bytes containing individual bits indicating
 *    which points in the range should have the control applied. If the bit is
 *    set the control operation is applied to the corresponding point. The first
 *    point is represented by the bit in bit 0 position of the first octet. The
 *    second point by bit 1 position etc.
 * returns:
 *  status of CROB operation. Valid values are:
 *    DNPDEFS_CROB_ST_SUCCESS - the command was performed successfully
 *    DNPDEFS_CROB_ST_TIMEOUT - the request was not accepted because the
 *      Operate message was received after the Arm timer timed out. The
 *      Arm timer is started when a Select operation for the same point
 *      was received.
 *    DNPDEFS_CROB_ST_FORMAT_ERROR - the request was not accepted due
 *      to formatting errors in the request
 *    DNPDEFS_CROB_ST_NOT_SUPPORTED - the request was not accepted because
 *      Control operations are not supported for this point
 *    DNPDEFS_CROB_ST_ALREADY_ACTIVE - the request was not accepted because
 *      the control queue is full or the point is already active
 *    DNPDEFS_CROB_ST_HARDWARE_ERROR - the request was not accepted due to
 *      control hardware problems
 *    DNPDEFS_CROB_ST_LOCAL - the request was not accepted because
 *      Local/Remote switch is in Local Position
 *    DNPDEFS_CROB_ST_NOT_AUTHORIZED - the request was not accepted because
 *      of insufficient authorization.
 *    DNPDEFS_CROB_ST_AUTO_INHIBIT - the request was not accepted because it
 *      was prevented or inhibited by a local automation process
 *    DNPDEFS_CROB_ST_PROC_LIMITED - the request was not accepted because the
 *     device cannot process any more activities than are presently in progress
 *    DNPDEFS_CROB_ST_OUT_OF_RANGE - the request was not accepted because the
 *     value is outside the acceptable range permitted for this point.
 *    DNPDEFS_CROB_ST_DOWNSTRM_LOCAL - the request was not accepted because the
 *     control is being forwarded to a downstream device that is reporting Local.
 *    DNPDEFS_CROB_ST_ALR_COMPLETE - the request was not accepted because the
 *     operation is already complete. For example if the request is to close a
 *     switch and the switch is already closed.
 *    DNPDEFS_CROB_ST_BLOCKED - the request was not accepted because it is
 *     specifically blocked at the outstation.
 *    DNPDEFS_CROB_ST_CANCELLED - the request was not accepted because the
 *     operation was cancelled.
 *    DNPDEFS_CROB_ST_BLOCKED_OM - the request was not accepted because another
 *     master has exclusive rights to operate this point.
 *    DNPDEFS_CROB_ST_DOWNSTRM_FAIL -  the request was not accepted because the
 *     control is being forwarded to a downstream device which cannot be reached
 *     or is otherwise incapable of performing the request.
 *    DNPDEFS_CROB_ST_UNDEFINED - the request not accepted because of some
 *     other undefined reason
 */
DNPDEFS_CROB_ST sdnpdata_binOutSelPatternMask(
  void *pHandle,
  TMWTYPES_UCHAR control,
  TMWTYPES_UCHAR count,
  TMWTYPES_ULONG activationPeriod,
  TMWTYPES_ULONG deactivationPeriod,
  TMWTYPES_USHORT firstPointNumber,
  TMWTYPES_USHORT lastPointNumber,
  TMWTYPES_UCHAR *pMask);

/* function: sdnpdata_binOutOpPatternMask
 * purpose: perform Operate Pattern Mask operation on the specified point
 * and return the result. If Object 13 Binary Output Command Events are desired
 *   you may call sdnpo013_addEvent() from this function.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  controlCode - the control function to perform. The Control Code
 *   may contained OR'd values of the following values:
 *     DNPDEFS_CROB_CTRL_PULSE_ON - The point(s) is turned on for the
 *      specified onTime (activationPeriod), turned off for the specified
 *       offTime, and leftin the off state.
 *     DNPDEFS_CROB_CTRL_PULSE_OFF - The point(s) is turned off for the
 *      specific offTime (deactivationPeriod), then turned on for the specified
 *      onTime, and left in the on state.
 *     DNPDEFS_CROB_CTRL_LATCH_ON - The point(s) is latched on
 *     DNPDEFS_CROB_CTRL_LATCH_OFF - The point(s) is latched off
 *     DNPDEFS_CROB_CTRL_CLEAR - cancel the currently running operation and
 *      remove queued operations on affected points immediately before
 *      activating this new operation
 *     DNPDEFS_CROB_CTRL_PAIRED_CLOSE - activate Close relay
 *     DNPDEFS_CROB_CTRL_PAIRED_TRIP - activate Trip relay
 *     DNPDEFS_CROB_CTRL_PAIRED_CLOSE and DNPDEFS_CROB_CTRL_PAIRED_TRIP are
 *      used in systems where a Trip and Close relay pair is used to energize
 *      and de-energize  the field points. Both of these bits can be 0 to
 *      activate the field point select relay only, without activating the
 *      Trip/Close relays. In a system without field point relays, if both
 *      of these bits are 0, then no control operation should be performed.
 *      In a system without Trip/Close relays, both of these bits should
 *      always be 0 to indicate normal digital control operations. It is
 *      invalid for DNPDEFS_CROB_CTRL_PAIRED_TRIP and
 *      DNPDEFS_CROB_CTRL_PAIRED_CLOSE to both be set.
 *  count - the number of times that the control operation should be
 *    performed in succession. If count is 0, do not execute the control.
 *  activationPeriod - amount of time the digital output is to be turned on
 *    (may not apply to all control types)
 *  deactivationPeriod - amount of time the digital output is to be turned off
 *    (may not apply to all control types)
 *  firstPointNumber - first point number represented in the pattern mask(*pMask).
 *  lastPointNumber - last point number represented in the pattern mask (*pMask).
 *  pMask - pointer to an array of bytes containing individual bits indicating
 *    which points in the range should have the control applied. If the bit is
 *    set the control operation is applied to the corresponding point. The first
 *    point is represented by the bit in bit 0 position of the first octet. The
 *    second point by bit 1 position etc.
 * returns:
 *  status of CROB operation. Valid values are:
 *    DNPDEFS_CROB_ST_SUCCESS - the command was performed successfully
 *    DNPDEFS_CROB_ST_TIMEOUT - the request was not accepted because the
 *      Operate message was received after the Arm timer timed out. The
 *      Arm timer is started when a Select operation for the same point
 *      was received.
 *    DNPDEFS_CROB_ST_NO_SELECT - No previously matching select message
 *      (i.e., an Operate message was sent to activate a control point
 *      that was not previously armed with a Select message.)
 *    DNPDEFS_CROB_ST_FORMAT_ERROR - the request was not accepted due
 *      to formatting errors in the request
 *    DNPDEFS_CROB_ST_NOT_SUPPORTED - the request was not accepted because
 *      Control operations are not supported for this point
 *    DNPDEFS_CROB_ST_ALREADY_ACTIVE - the request was not accepted because
 *      the control queue is full or the point is already active
 *    DNPDEFS_CROB_ST_HARDWARE_ERROR - the request was not accepted due to
 *      control hardware problems
 *    DNPDEFS_CROB_ST_LOCAL - the request was not accepted because
 *      Local/Remote switch is in Local Position
 *    DNPDEFS_CROB_ST_NOT_AUTHORIZED - the request was not accepted because
 *      of insufficient authorization.
 *    DNPDEFS_CROB_ST_AUTO_INHIBIT - the request was not accepted because it
 *      was prevented or inhibited by a local automation process
 *    DNPDEFS_CROB_ST_PROC_LIMITED - the request was not accepted because the
 *     device cannot process any more activities than are presently in progress
 *    DNPDEFS_CROB_ST_OUT_OF_RANGE - the request was not accepted because the
 *     value is outside the acceptable range permitted for this point.
 *    DNPDEFS_CROB_ST_DOWNSTRM_LOCAL - the request was not accepted because the
 *     control is being forwarded to a downstream device that is reporting Local.
 *    DNPDEFS_CROB_ST_ALR_COMPLETE - the request was not accepted because the
 *     operation is already complete. For example if the request is to close a
 *     switch and the switch is already closed.
 *    DNPDEFS_CROB_ST_BLOCKED - the request was not accepted because it is
 *     specifically blocked at the outstation.
 *    DNPDEFS_CROB_ST_CANCELLED - the request was not accepted because the
 *     operation was cancelled.
 *    DNPDEFS_CROB_ST_BLOCKED_OM - the request was not accepted because another
 *     master has exclusive rights to operate this point.
 *    DNPDEFS_CROB_ST_DOWNSTRM_FAIL -  the request was not accepted because the
 *     control is being forwarded to a downstream device which cannot be reached
 *     or is otherwise incapable of performing the request.
 *    DNPDEFS_CROB_ST_UNDEFINED - the request not accepted because of some
 *     other undefined reason
 */
DNPDEFS_CROB_ST sdnpdata_binOutOpPatternMask(
  void *pHandle,
  TMWTYPES_UCHAR control,
  TMWTYPES_UCHAR count,
  TMWTYPES_ULONG activationPeriod,
  TMWTYPES_ULONG deactivationPeriod,
  TMWTYPES_USHORT firstPointNumber,
  TMWTYPES_USHORT lastPointNumber,
  TMWTYPES_UCHAR *pMask);

/* function: sdnpdata_binOutCancelPatMask */
void TMWDEFS_GLOBAL sdnpdata_binOutCancelPatMask(
  void *pHandle,
  TMWTYPES_USHORT firstPointNumber,
  TMWTYPES_USHORT lastPointNumber,
  TMWTYPES_UCHAR *pMask);

/* function: sdnpdata_binOutCmdEventDefVariation
 * purpose: Determine default event variation for this binary output command
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  classMask - mask indicating what event class is being requested
 * returns:
 *  default variation for this point
 *  NOTE: this will only be called if the default variation for binary
 *   output command obj13DefaultVariation is configured as zero
 */
TMWTYPES_UCHAR TMWDEFS_GLOBAL sdnpdata_binOutCmdEventDefVariation(
  void *pPoint,
  TMWDEFS_CLASS_MASK classMask);

/* function: sdnpdata_binOutCmdEventMode
 * purpose: Determine event mode for this point
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  event mode for this point, SOE or LAST
 *  NOTE: this will only be called if the event mode binary output command
 *    if binaryOutCmdEventMode is configured as TMWDEFS_EVENT_MODE_PER_POINT
 */
TMWDEFS_EVENT_MODE TMWDEFS_GLOBAL sdnpdata_binOutCmdEventMode(
  void *pPoint);

/* function: sdnpdata_binOutCmdEventClass
 * purpose: Return the class in which binary output command events from
 *  this data point belong.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  Class in which these events will be returned
 */
TMWDEFS_CLASS_MASK TMWDEFS_GLOBAL sdnpdata_binOutCmdEventClass(
  void *pPoint);

/* function: sdnpdata_binOutCmdChanged
 * purpose: Determine if the command status of the specified point has changed
 *  and if so return the new value. This function is used to scan for events
 *  on each data point. It will be called if SDNPSESN_CONFIG
 *  binaryOutCmdScanPeriod is nonzero.
 *  NOTE: this functionality is compiled out by defining
 *  SDNPDATA_SUPPORT_OBJ13 FALSE  (default)
 *  or SDNPDATA_SUPPORT_EVENT_SCAN FALSE
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  pStatus - pointer to command status to be filled in, representing the control
 *   requested for the output.
 *   DNPDEFS_DBAS_FLAG_BINARY_ON or DNPDEFS_DBAS_FLAG_BINARY_OFF
 *   or'ed with status of CROB operation. Valid values are:
 *    DNPDEFS_CROB_ST_SUCCESS - the command was performed successfully
 *    DNPDEFS_CROB_ST_TIMEOUT - the request was not accepted because the
 *      Operate message was received after the Arm timer timed out. The
 *      Arm timer is started when a Select operation for the same point
 *      was received.
 *    DNPDEFS_CROB_ST_NO_SELECT - No previously matching select message
 *      (i.e., an Operate message was sent to activate a control point
 *      that was not previously armed with a Select message.)
 *    DNPDEFS_CROB_ST_FORMAT_ERROR - the request was not accepted due
 *      to formatting errors in the request
 *    DNPDEFS_CROB_ST_NOT_SUPPORTED - the request was not accepted because
 *      Control operations are not supported for this point
 *    DNPDEFS_CROB_ST_ALREADY_ACTIVE - the request was not accepted because
 *      the control queue is full or the point is already active
 *    DNPDEFS_CROB_ST_HARDWARE_ERROR - the request was not accepted due to
 *      control hardware problems
 *    DNPDEFS_CROB_ST_LOCAL - the request was not accepted because
 *      Local/Remote switch is in Local Position
 *    DNPDEFS_CROB_ST_NOT_AUTHORIZED - the request was not accepted because
 *      of insufficient authorization.
 *    DNPDEFS_CROB_ST_AUTO_INHIBIT - the request was not accepted because it
 *      was prevented or inhibited by a local automation process
 *    DNPDEFS_CROB_ST_PROC_LIMITED - the request was not accepted because the
 *     device cannot process any more activities than are presently in progress
 *    DNPDEFS_CROB_ST_OUT_OF_RANGE - the request was not accepted because the
 *     value is outside the acceptable range permitted for this point.
 *    DNPDEFS_CROB_ST_DOWNSTRM_LOCAL - the request was not accepted because the
 *     control is being forwarded to a downstream device that is reporting Local.
 *    DNPDEFS_CROB_ST_ALR_COMPLETE - the request was not accepted because the
 *     operation is already complete. For example if the request is to close a
 *     switch and the switch is already closed.
 *    DNPDEFS_CROB_ST_BLOCKED - the request was not accepted because it is
 *     specifically blocked at the outstation.
 *    DNPDEFS_CROB_ST_CANCELLED - the request was not accepted because the
 *     operation was cancelled.
 *    DNPDEFS_CROB_ST_BLOCKED_OM - the request was not accepted because another
 *     master has exclusive rights to operate this point.
 *    DNPDEFS_CROB_ST_DOWNSTRM_FAIL -  the request was not accepted because the
 *     control is being forwarded to a downstream device which cannot be reached
 *     or is otherwise incapable of performing the request.
 *    DNPDEFS_CROB_ST_UNDEFINED - the request not accepted because of some
 *     other undefined reason
 * returns:
 *  TMWDEFS_TRUE if the command status of point has changed, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_binOutCmdChanged(
  void *pPoint,
  TMWTYPES_UCHAR *pStatus);

/* function: sdnpdata_binOutCmdAssignClass
 * purpose: Assign the class in which binary output command events from this
 *  data point will belong.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  classMask - new class in which to generate events from this point
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_binOutCmdAssignClass(
  void *pPoint,
  TMWDEFS_CLASS_MASK classMask);

/* Binary Counters */

/* function: sdnpdata_binCntrGetDescription
 * purpose: Get description of this point
 *  NOTE: this is used only by sdnpxml/sdnpxml2.c if SDNPDATA_SUPPORT_XMLxx is
TRUE
 *  code to generate the configuration file or Device Profile for this device.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  pointer to string describing point
 *  TMWDEFS_NULL if failure
 */
TMWTYPES_CHAR * TMWDEFS_GLOBAL sdnpdata_binCntrGetDescription(
  void *pPoint);

/* function: sdnpdata_binCntrQuantity
 * purpose: Return the number of binary counters in the
 *  specified database.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 * returns:
 *  The number of binary counters (see note for
 *  sdnpdata_xxxQuantity at top of this file)
 */
TMWTYPES_USHORT TMWDEFS_GLOBAL sdnpdata_binCntrQuantity(
  void *pHandle);
```

```
/* function: sdnpdata_binCntrGetPoint
 * purpose: Return a handle to a specific data point. This handle is
 *  used by the routines below to read values from this counter and to
 *  assign this counter to an event class.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  pointNum - point number to return
 * returns:
 *  Pointer to specified data point or TMWDEFS_NULL if this point
 *   is currently disabled for reading.
 */
void * TMWDEFS_GLOBAL sdnpdata_binCntrGetPoint(
  void *pHandle,
  TMWTYPES_USHORT pointNum);

/* function: sdnpdata_binCntrGetFrzPoint
 * purpose: Return a handle to a specific data point. This handle is
 *  used by the routines below to freeze this counter. This is separate
 *  from the binCntrGetPoint() function to allow a running counter to not be
 *  reported in response to a class 0 read, but still allow the counter to
 *  be frozen, as required by the Application Layer Spec.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  pointNum - point number to return handle for
 * returns:
 *  Pointer to specified data point or TMWDEFS_NULL if this point does not
 *  exist.
 */
void * TMWDEFS_GLOBAL sdnpdata_binCntrGetFrzPoint(
  void *pHandle,
  TMWTYPES_USHORT pointNum);

/* function: sdnpdata_binCntrIsClass0
 * purpose: Should this point be reported in response to an object 60
 *   variation 1 read request. This allows individual points to be excluded
 *   from a class 0 response but still readable by a specific object group
 *   read request.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  TMWDEFS_TRUE if point should be reported.
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_binCntrIsClass0(
  void *pPoint);

/* function: sdnpdata_binCntrAssignClass
 * purpose: Assign the class in which events from this data point
 *  will belong.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  classMask - new class in which to generate events from this point
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_binCntrAssignClass(
  void *pPoint,
  TMWDEFS_CLASS_MASK classMask);

/* function: sdnpdata_binCntrDefVariation
 * purpose: Determine default static variation for this binary counter
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  default variation for this point
 *  NOTE: this will only be called if the default variation for
 *  binary counter obj20DefaultVariation is configured as 0.
 */
TMWTYPES_UCHAR TMWDEFS_GLOBAL sdnpdata_binCntrDefVariation(
  void *pPoint);

#if SDNPDATA_SUPPORT_OBJ22
/* function: sdnpdata_binCntrEventDefVariation
 * purpose: Determine default variation for this counter
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  classMask - mask indicating what event class is being requested
 * returns:
 *  default variation for this point
 *  NOTE: this will only be called if the default variation for
 *  counter change events obj22DefaultVariation is configured as 0.
```

```
  */
 TMWTYPES_UCHAR TMWDEFS_GLOBAL sdnpdata_binCntrEventDefVariation(
   void *pPoint,
   TMWDEFS_CLASS_MASK classMask);

 /* function: sdnpdata_binCntrEventClass
* purpose: Return the class in which events from this data point
*  belong.
* arguments:
*  pPoint - handle to data point returned from 'getPoint' function.
* returns:
*  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
*/
 TMWDEFS_CLASS_MASK TMWDEFS_GLOBAL sdnpdata_binCntrEventClass(
   void *pPoint);

  /* function: sdnpdata_binCntrEventMode
  * purpose: Determine event mode for this point
  * arguments:
  *  pPoint - handle to data point returned from 'getPoint' function.
  * returns:
  *  event mode for this point, SOE or LAST
  *  NOTE: this will only be called if the event mode binary counters
  *    if binaryCounterEventMode is configured as TMWDEFS_EVENT_MODE_PER_POINT
  */
 TMWDEFS_EVENT_MODE TMWDEFS_GLOBAL sdnpdata_binCntrEventMode(
   void *pPoint);
#endif

 /* function: sdnpdata_binCntrRead
  * purpose: Return the current value of the specified point.
  * arguments:
  *  pPoint - handle to data point returned from 'getPoint' function.
  *  pValue - pointer to location to store current value
  *  pFlags - pointer to location to store current DNP3 flags
  *   pFlags contains a status indication and the current state of the point.
  *   The following values (or OR'd combinations) are valid for this type:
  *     DNPDEFS_DBAS_FLAG_OFF_LINE - the point is off-line, and the returned
  *       state of this point may not be correct
  *     DNPDEFS_DBAS_FLAG_ON_LINE - the binary counter 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 device
  *       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. The counter
  *       value should be set to 0 upon rollover, and counting is resumed as
  *       normal. The Rollover bit should be cleared when the counter value
  *       and roll-over state have been reported.
  *       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".
  *     DNPDEFS_DBAS_FLAG_DISCONTINUITY - value cannot be compared against
  *       a prior value to obtain the correct count difference
  * returns:
  *  void
  */
 void TMWDEFS_GLOBAL sdnpdata_binCntrRead(
   void *pPoint,
   TMWTYPES_ULONG *pValue,
   TMWTYPES_UCHAR *pFlags);

 /* function: sdnpdata_binCntrChanged
  * purpose: Determine if the specified point has changed and if so
  *  return the new value. This function is used to scan for events on
  *  each data point. It will be called if SDNPSESN_CONFIG
  *  binaryCounterScanPeriod is nonzero.
  *  NOTE: this functionality is compiled out by defining
  *  SDNPDATA_SUPPORT_OBJ22 FALSE
  *  or SDNPDATA_SUPPORT_EVENT_SCAN FALSE
  * arguments:
  *  pPoint - handle to data point returned from 'getPoint' function.
  *  pFlags - pointer to location to store current DNP3 flags.
  *   The following values (or OR'd combinations) are valid for this type:
  *      DNPDEFS_DBAS_FLAG_OFF_LINE - the point is off-line, and the returned
  *        state of this point may not be correct
  *      DNPDEFS_DBAS_FLAG_ON_LINE - the binary counter 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. The counter
  *        value should be set to 0 upon rollover, and counting is resumed as
  *        normal. The Rollover bit should be cleared when the counter value
  *        and roll-over state have been reported.
  *        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".
  *      DNPDEFS_DBAS_FLAG_DISCONTINUITY - value cannot be compared against
  *        a prior value to obtain the correct count difference
  * returns:
  *  TMWDEFS_TRUE if the point has changed, else TMWDEFS_FALSE
  */
 TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_binCntrChanged(
   void *pPoint,
   TMWTYPES_ULONG *pValue,
   TMWTYPES_UCHAR *pFlags);

 /* function: sdnpdata_binCntrFreeze
  * purpose:  Freeze the specified counter
  * arguments:
  *  pPoint - handle to data point returned from 'getPoint' function.
  *  clearAfterFreeze - whether or not counter should be cleared after freeze
  * returns:
  *  TMWDEFS_TRUE if successful
  *  TMWDEFS_FALSE otherwise
  */
 TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_binCntrFreeze(
   void *pPoint,
   TMWTYPES_BOOL clearAfterFreeze);

 /* function: sdnpdata_binCntrFreezeAtTime
  * purpose:  Freeze the specified counter at time
  * arguments:
  *  pPoint - handle to data point returned from 'getPoint' function.
  *  timeDateEnum - time-date field schedule interpretation
  *  pFreezeTime - time to perform freeze
  *  freezeInterval - time interval to perfrom periodic freezes (milliseconds)
  * returns:
  *  TMWDEFS_TRUE if successful
  *  TMWDEFS_FALSE otherwise
  */
 TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_binCntrFreezeAtTime(
   void *pPoint,
   DNPDATA_FREEZE_TIME_DATE_FIELD timeDateEnum,
   TMWDTIME *pFreezeTime,
   TMWTYPES_ULONG freezeInterval);

 /* Frozen Counters */

 /* function: sdnpdata_frznCntrGetDescription
  * purpose: Get description of this point
  *  NOTE: this is used only by sdnpxml/sdnpxml2.c if SDNPDATA_SUPPORT_XMLxx is
 TRUE
  *  code to generate the configuration file or Device Profile for this device.
  * arguments:
  *  pPoint - handle to data point returned from 'getPoint' function.
  * returns:
  *  pointer to string describing point
  *  TMWDEFS_NULL if failure
  */
 TMWTYPES_CHAR * TMWDEFS_GLOBAL sdnpdata_frznCntrGetDescription(
   void *pPoint);
```

```
/* function: sdnpdata_frznCntrQuantity
 * purpose: Return the number of frozen counters in the
 *  specified database.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 * returns:
 *  The number of frozen counters (see note for
 *  sdnpdata_xxxQuantity at top of this file)
 */
TMWTYPES_USHORT TMWDEFS_GLOBAL sdnpdata_frznCntrQuantity(
  void *pHandle);

/* function: sdnpdata_frznCntrGetPoint
 * purpose: Return a handle to a specific data point. This handle is
 *  used by the routines below to read and write values and control
 *  information to this point.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  pointNum - point number to return
 * returns:
 *  Pointer to specified data point or TMWDEFS_NULL if this point
 *   is currently disabled.
 */
void * TMWDEFS_GLOBAL sdnpdata_frznCntrGetPoint(
  void *pHandle,
  TMWTYPES_USHORT pointNum);

/* function: sdnpdata_frznCntrEventClass
 * purpose: Return the class in which events from this data point
 *  belong.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWDEFS_CLASS_MASK TMWDEFS_GLOBAL sdnpdata_frznCntrEventClass(
  void *pPoint);

/* function: sdnpdata_FrznCntrIsClass0
 * purpose: Should this point be reported in response to an object 60
 *   variation 1 read request. This allows individual points to be excluded
 *   from a class 0 response but still readable by a specific object group
 *   read request.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  TMWDEFS_TRUE if point should be reported.
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_frznCntrIsClass0(
  void *pPoint);

/* function: sdnpdata_frznCntrAssignClass
 * purpose: Assign the class in which events from this data point
 *  will belong.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  classMask - new class in which to generate events from this point
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_frznCntrAssignClass(
  void *pPoint,
  TMWDEFS_CLASS_MASK classMask);

/* function: sdnpdata_frznCntrDefVariation
 * purpose: Determine default static variation for this frozen counter
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  default variation for this point
 *  NOTE: this will only be called if the default variation for
 *  frozen counter obj21DefaultVariation is configured as 0.
 */
TMWTYPES_UCHAR TMWDEFS_GLOBAL sdnpdata_frznCntrDefVariation(
  void *pPoint);

/* function: sdnpdata_frznCntrEventDefVariation
 * purpose: Determine default variation for this frozen counter
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  classMask - mask indicating what event class is being requested
 * returns:
 *  default variation for this point
 *  NOTE: this will only be called if the default variation for frozen
 *  counter change events obj23DefaultVariation is configured as 0.
 */
TMWTYPES_UCHAR TMWDEFS_GLOBAL sdnpdata_frznCntrEventDefVariation(
  void *pPoint,
  TMWDEFS_CLASS_MASK classMask);

 /* function: sdnpdata_frznCntrEventMode
 * purpose: Determine event mode for this point
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  event mode for this point, SOE or LAST
 *  NOTE: this will only be called if the event mode frozen counters
 *    if frozenCounterEventMode is configured as TMWDEFS_EVENT_MODE_PER_POINT
 */
TMWDEFS_EVENT_MODE TMWDEFS_GLOBAL sdnpdata_frznCntrEventMode(
  void *pPoint);

/* function: sdnpdata_frznCntrRead
 * purpose: Return the current value of the specified point.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  pValue - pointer to location to store current value
 *  pFlags - pointer to location to store current DNP3 flags
 *   The following values (or OR'd combinations) are valid for this type:
 *      DNPDEFS_DBAS_FLAG_OFF_LINE - the point is off-line, and the returned
 *        state of this point may not be correct
 *      DNPDEFS_DBAS_FLAG_ON_LINE - the frozen counter 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. The counter
 *        value should be set to 0 upon rollover, and counting is resumed as
 *        normal. The Rollover bit should be cleared when the counter value
 *        and roll-over state have been reported.
 *        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".
 *      DNPDEFS_DBAS_FLAG_DISCONTINUITY - value cannot be compared against
 *        a prior value to obtain the correct count difference
 *  pTimeOfFreeze - pointer to location to store time of freeze if known
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL sdnpdata_frznCntrRead(
  void *pPoint,
  TMWTYPES_ULONG *pValue,
  TMWTYPES_UCHAR *pFlags,
  TMWDTIME *pTimeOfFreeze);

/* function: sdnpdata_frznCntrChanged
 * purpose: Determine if the specified point has changed and if so
 *  return the new value. This function is used to scan for events on
 *  each data point. It will be called if SDNPSESN_CONFIG
 *  frozenCounterScanPeriod is nonzero.
 *  NOTE: this functionality is compiled out by defining
 *  SDNPDATA_SUPPORT_OBJ23 FALSE
 *  or SDNPDATA_SUPPORT_EVENT_SCAN FALSE
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  pFlags - pointer to location to store current DNP3 flags
 *   The following values (or OR'd combinations) are valid for this type:
 *      DNPDEFS_DBAS_FLAG_OFF_LINE - the point is off-line, and the returned
 *        state of this point may not be correct
 *      DNPDEFS_DBAS_FLAG_ON_LINE - the frozen counter 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. The counter
 *        value should be set to 0 upon rollover, and counting is resumed as
 *        normal. The Rollover bit should be cleared when the counter value
 *        and roll-over state have been reported.
 *        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".
 *      DNPDEFS_DBAS_FLAG_DISCONTINUITY - value cannot be compared against
 *        a prior value to obtain the correct count difference
 * returns:
 *  TMWDEFS_TRUE if the point has changed, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_frznCntrChanged(
  void *pPoint,
  TMWTYPES_ULONG *pValue,
  TMWTYPES_UCHAR *pFlags);

/* Analog Inputs */

/* function: sdnpdata_anlgInGetDescription
 * purpose: Get description of this point
 *  NOTE: this is used only by sdnpxml/sdnpxml2.c if SDNPDATA_SUPPORT_XMLxx is
TRUE
 *  code to generate the configuration file or Device Profile for this device.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  pointer to string describing point
 *  TMWDEFS_NULL if failure
 */
TMWTYPES_CHAR * TMWDEFS_GLOBAL sdnpdata_anlgInGetDescription(
  void *pPoint);

/* function: sdnpdata_anlgInQuantity
 * purpose: Return the number of analog input data points in the
 *  specified database.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 * returns:
 *  The number of analog inputs (see note for
 *  sdnpdata_xxxQuantity at top of this file)
 */
TMWTYPES_USHORT TMWDEFS_CALLBACK sdnpdata_anlgInQuantity(
  void *pHandle);

/* function: sdnpdata_anlgInGetPoint
 * purpose: Return a handle to a specific data point. This handle is
 *  used by the routines below to read and write values and control
 *  information to this point.
 * NOTE: This is not the function that will be called for Freeze
 *  This allows the analog input to be disabled for reading, but still
 *  can be frozen as required by TB2018-004
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  pointNum - point number to return
 * returns:
 *  Pointer to specified data point or TMWDEFS_NULL if this point
 *   is currently disabled.
 */
void * TMWDEFS_CALLBACK sdnpdata_anlgInGetPoint(
  void *pHandle,
  TMWTYPES_USHORT pointNum);

/* function: sdnpdata_anlgInGetFreezePoint
 * purpose: Return a handle to a specific data point. This handle is
 *  used by the routines below to freeze the analog input point.
 *  This allows the analog input to be disabled for reading, but still
 *  can be frozen as required by TB2018-004
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  pointNum - point number to return
 * returns:
 *  Pointer to specified data point or TMWDEFS_NULL if this point
 *   is currently disabled.
 */
void * TMWDEFS_CALLBACK sdnpdata_anlgInGetFreezePoint(
  void *pHandle,
  TMWTYPES_USHORT pointNum);

/* function: sdnpdata_anlgInDefVariation
 * purpose: Determine default static variation for this analog input
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  default variation for this point
 *  NOTE: this will only be called if the default variation for
 *  analog input obj30DefaultVariation is configured as 0.
 */
TMWTYPES_UCHAR TMWDEFS_GLOBAL sdnpdata_anlgInDefVariation(
  void *pPoint);

/* function: sdnpdata_anlgInDbandDefVar
 * purpose: Determine default static variation for deadband for this analog
 *  input
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  default variation for this point
 *  NOTE: this will only be called if the default variation for
 *  analog input obj34DefaultVariation is configured as 0.
 */
TMWTYPES_UCHAR TMWDEFS_GLOBAL sdnpdata_anlgInDbandDefVar(
  void *pPoint);

/* function: sdnpdata_anlgInEventDefVariation
 * purpose: Determine default variation for this frozen counter
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  classMask - mask indicating what event class is being requested
 * returns:
 *  default variation for this point
 *  NOTE: this will only be called if the default variation for
 *   analog input change events obj32DefaultVariation is configured as zero
 */
TMWTYPES_UCHAR TMWDEFS_GLOBAL sdnpdata_anlgInEventDefVariation(
  void *pPoint,
  TMWDEFS_CLASS_MASK classMask);

/* function: sdnpdata_anlgInEventMode
 * purpose: Determine event mode for this point
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  event mode for this point, SOE, LAST, or CURRENT
 *  NOTE: this will only be called if the event mode for analog inputs
 *    if analogInputEventMode is configured as TMWDEFS_EVENT_MODE_PER_POINT
 */
TMWDEFS_EVENT_MODE TMWDEFS_GLOBAL sdnpdata_anlgInEventMode(
  void *pPoint);

/* function: sdnpdata_anlgInEventClass
 * purpose: Return the class in which events from this data point
 *  belong.
 *  NOTE: if Frozen Analog Inputs are supported, read the description of
 *   the proper behavior related to Class 0 and Event Class.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  Class in which these events will be returned
 */
TMWDEFS_CLASS_MASK TMWDEFS_GLOBAL sdnpdata_anlgInEventClass(
  void *pPoint);

/* function: sdnpdata_anlgInIsClass0
 * purpose: Should this point be reported in response to an object 60
 *   variation 1 read request. This allows individual points to be excluded
 *   from a class 0 response but still readable by a specific object group
 *   read request.
 *  NOTE: if Frozen Analog Inputs are supported, read the description of
 *   the proper behavior related to Class 0 and Event Class.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  TMWDEFS_TRUE if point should be reported.
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_anlgInIsClass0(
  void *pPoint);

/* function: sdnpdata_anlgInAssignClass
 * purpose: Assign the class in which events from this data point
 *  will belong.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  classMask - new class in which to generate events from this point
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_anlgInAssignClass(
  void *pPoint,
  TMWDEFS_CLASS_MASK classMask);

/* function: sdnpdata_anlgInRead
 * purpose: Return the current value of the specified point.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  pValue - pointer to location to store current value
 *   function should set pValue->type to indicate data type returned
 *     TMWTYPES_ANALOG_TYPE_DOUBLE
 *     TMWTYPES_ANALOG_TYPE_SFLOAT
 *     TMWTYPES_ANALOG_TYPE_LONG
 *     TMWTYPES_ANALOG_TYPE_SHORT
 *     TMWTYPES_ANALOG_TYPE_SCALED - allows database to return a single
 *      precision floating point value to support a read of sfloat, AND a
 *      long value to support a read of a scaled long representation of
 *      the sfloat. 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.
 *     TMWTYPES_ANALOG_TYPE_DSCALED - allows database to return a double
 *      precision floating point value to support a read of double, AND a
 *      long value to support a read of a scaled long representation of
 *      the double. 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.
 *   NOTE:
 *    If TMWCNFG_SUPPORT_DOUBLE == TRUE returning the value as
 *    TMWTYPES_ANALOG_TYPE_DOUBLE allows for proper reads of DOUBLE FLOAT
 *    SFLOAT, ULONG or USHORT.
 *    If TMWCNFG_SUPPORT_DOUBLE == FALSE and
 *       TMWCNFG_SUPPORT_SFLOAT == TRUE
 *    To provide for best precision and OVER_RANGE flag setting:
 *    return the value as TMWTYPES_ANALOG_TYPE_LONG if the value is between
 *    TMWDEFS_LONG_MIN and TMWDEFS_LONG_MAX.
 *    return the value as TMWTYPES_ANALOG_TYPE_SFLOAT if the value is outside
 *    of that range. While each type is 32 bits in size the long allows
 *    for 31 bits of precision. Returning a float instead allows greater values,
 *    but only 23 bits of precision.
 *    If both TMWCNFG_SUPPORT_DOUBLE and TMWCNFG_SUPPORT_SFLOAT == FALSE and
 *    the long value is over or under range, set DNPDEFS_DBAS_FLAG_OVER_RANGE
 *    bit in pFlags and set lval to TMWDEFS_LONG_MIN or TMWDEFS_LONG_MAX.
 *
 *  pFlags - pointer to location to store current DNP3 flags
 *   The following values (or OR'd combinations) are valid for this type:
 *      DNPDEFS_DBAS_FLAG_OFF_LINE - the point is off-line, and the returned
 *        state of this point may not be correct
 *      DNPDEFS_DBAS_FLAG_ON_LINE - the analog 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.
 *        NOTE: If both TMWCNFG_SUPPORT_DOUBLE and TMWCNFG_SUPPORT_SFLOAT ==
 *        FALSE and the long value is over or under range, set
 *        DNPDEFS_DBAS_FLAG_OVER_RANGE bit in pFlags and set lval to
 *        TMWDEFS_LONG_MIN or TMWDEFS_LONG_MAX.
 *      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.
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL sdnpdata_anlgInRead(
  void *pPoint,
  TMWTYPES_ANALOG_VALUE *pValue,
  TMWTYPES_UCHAR *pFlags);

/* function: sdnpdata_anlgInChanged
 * purpose: Determine if the specified point has changed and if so
 *  return the new value. This function is used to scan for events on
 *  each data point. It will be called if SDNPSESN_CONFIG
 *  analogInputScanPeriod is nonzero.
 *  NOTE: this functionality is compiled out by defining
 *  SDNPDATA_SUPPORT_OBJ32 FALSE
 *  or SDNPDATA_SUPPORT_EVENT_SCAN FALSE
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  pValue - pointer to location to store current value
 *   function should set pValue->type to indicate data type returned
 *     TMWTYPES_ANALOG_TYPE_DOUBLE
 *     TMWTYPES_ANALOG_TYPE_SFLOAT
 *     TMWTYPES_ANALOG_TYPE_LONG
 *     TMWTYPES_ANALOG_TYPE_SHORT
 *     TMWTYPES_ANALOG_TYPE_SCALED - allows database to return a single precision
 *      floating point value to support a read of sfloat, AND a long value
 *      to support a read of a scaled long representation of the sfloat.
 *      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.
 *     TMWTYPES_ANALOG_TYPE_DSCALED - allows database to return a double
 *      precision floating point value to support a read of double, AND a
 *      long value to support a read of a scaled long representation of
 *      the double. 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.
 *   NOTE:
 *    If TMWCNFG_SUPPORT_DOUBLE == TRUE returning the value as
 *    TMWTYPES_ANALOG_TYPE_DOUBLE allows for proper values for events of type
 *    DOUBLE FLOAT, SFLOAT, ULONG or USHORT.
 *
 *    If TMWCNFG_SUPPORT_DOUBLE == FALSE and
 *       TMWCNFG_SUPPORT_SFLOAT == TRUE
 *    To provide for best precision and OVER_RANGE flag setting:
 *    return the value as TMWTYPES_ANALOG_TYPE_LONG if the value is between
 *    TMWDEFS_LONG_MIN and TMWDEFS_LONG_MAX.
 *    return the value as TMWTYPES_ANALOG_TYPE_SFLOAT if the value is outside
 *    of that range. While each type is 32 bits in size the long allows
 *    for 31 bits of precision. Returning a float instead allows greater values,
 *    but only 23 bits of precision.
 *    If both TMWCNFG_SUPPORT_DOUBLE and TMWCNFG_SUPPORT_SFLOAT == FALSE and
 *    the long value is over or under range, set DNPDEFS_DBAS_FLAG_OVER_RANGE
 *    bit in pFlags and set lval to TMWDEFS_LONG_MIN or TMWDEFS_LONG_MAX.
 *
 *  pFlags - pointer to location to store current DNP3 flags
 *   The following values (or OR'd combinations) are valid for this type:
 *      DNPDEFS_DBAS_FLAG_OFF_LINE - the point is off-line, and the returned
 *        state of this point may not be correct
 *      DNPDEFS_DBAS_FLAG_ON_LINE - the analog 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.
 *        NOTE: If both TMWCNFG_SUPPORT_DOUBLE and TMWCNFG_SUPPORT_SFLOAT ==
 *        FALSE and the long value is over or under range, set
 *        DNPDEFS_DBAS_FLAG_OVER_RANGE bit in pFlags and set lval to
 *        TMWDEFS_LONG_MIN or TMWDEFS_LONG_MAX.
 *      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.
 * returns:
 *  TMWDEFS_TRUE if the point has changed, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_anlgInChanged(
  void *pPoint,
  TMWTYPES_ANALOG_VALUE *pValue,
  TMWTYPES_UCHAR *pFlags);

/* function: sdnpdata_anlgInFreeze
 * purpose:  Freeze the specified analog input
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  clearAfterFreeze - whether or not analog input should be cleared after freeze
 * returns:
 *  TMWDEFS_TRUE if successful
 *  TMWDEFS_FALSE otherwise
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_anlgInFreeze(
  void *pPoint,
  TMWTYPES_BOOL clearAfterFreeze);

/* function: sdnpdata_anlgInFreezeAtTime
 * purpose:  Freeze the specified analog input at time
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  timeDateEnum - time-date field schedule interpretation
 *  pFreezeTime - time to perform freeze
 *  freezeInterval - time interval to perfrom periodic freezes (milliseconds)
 * returns:
 *  TMWDEFS_TRUE if successful
 *  TMWDEFS_FALSE otherwise
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_anlgInFreezeAtTime(
  void *pPoint,
  DNPDATA_FREEZE_TIME_DATE_FIELD timeDateEnum,
  TMWDTIME *pFreezeTime,
  TMWTYPES_ULONG freezeInterval);
/* Frozen Analog Inputs */

/* function: sdnpdata_frznAnlgInGetDescription
 * purpose: Get description of this point
 *  NOTE: this is used only by sdnpxml/sdnpxml2.c if SDNPDATA_SUPPORT_XMLxx is
TRUE
 *  code to generate the configuration file or Device Profile for this device.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  pointer to string describing point
 *  TMWDEFS_NULL if failure
 */
TMWTYPES_CHAR * TMWDEFS_GLOBAL sdnpdata_frznAnlgInGetDescription(
  void *pPoint);

/* function: sdnpdata_frznAnlgInQuantity
 * purpose: Return the number of analog input data points in the
 *  specified database.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 * returns:
 *  The number of analog inputs (see note for
 *  sdnpdata_xxxQuantity at top of this file)
 */
TMWTYPES_USHORT TMWDEFS_CALLBACK sdnpdata_frznAnlgInQuantity(
  void *pHandle);

/* function: sdnpdata_frznAnlgInGetPoint
 * purpose: Return a handle to a specific data point. This handle is
 *  used by the routines below to read and write values and control
 *  information to this point.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  pointNum - point number to return
 * returns:
 *  Pointer to specified data point or TMWDEFS_NULL if this point
 *   is currently disabled.
 */
void * TMWDEFS_CALLBACK sdnpdata_frznAnlgInGetPoint(
  void *pHandle,
  TMWTYPES_USHORT pointNum);

/* function: sdnpdata_frznAnlgInDefVariation
 * purpose: Determine default static variation for this analog input
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  default variation for this point
 *  NOTE: this will only be called if the default variation for
 *  analog input obj30DefaultVariation is configured as 0.
 */
TMWTYPES_UCHAR TMWDEFS_GLOBAL sdnpdata_frznAnlgInDefVariation(
  void *pPoint);

/* function: sdnpdata_frznAnlgInEventDefVariation
 * purpose: Determine default variation for this frozen counter
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  classMask - mask indicating what event class is being requested
 * returns:
 *  default variation for this point
 *  NOTE: this will only be called if the default variation for
 *   analog input change events obj32DefaultVariation is configured as zero
 */
TMWTYPES_UCHAR TMWDEFS_GLOBAL sdnpdata_frznAnlgInEventDefVariation(
  void *pPoint,
  TMWDEFS_CLASS_MASK classMask);

/* function: sdnpdata_frznAnlgInEventMode
 * purpose: Determine event mode for this point
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  event mode for this point, SOE, LAST, or CURRENT
 *  NOTE: this will only be called if the event mode for analog inputs
 *    if analogInputEventMode is configured as TMWDEFS_EVENT_MODE_PER_POINT
 */
TMWDEFS_EVENT_MODE TMWDEFS_GLOBAL sdnpdata_frznAnlgInEventMode(
  void *pPoint);

/* function: sdnpdata_frznAnlgInEventClass
 * purpose: Return the class in which events from this data point
 *  belong.
 *  NOTE: Each AnlgIn pointNum x may generate either Obj32 or Obj33 event
 *   but not both, as required by TB2018-004. If this function returns an
 *   event class for point x then sdnpdata_anlgInEventClass should return
 *    TMWDEFS_CLASS_MASK_NONE.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  Class in which these events will be returned
 */
TMWDEFS_CLASS_MASK TMWDEFS_GLOBAL sdnpdata_frznAnlgInEventClass(
  void *pPoint);

/* function: sdnpdata_frznAnlgInIsClass0
 * purpose: Should this point be reported in response to an object 60
 *   variation 1 read request. This allows individual points to be excluded
 *   from a class 0 response but still readable by a specific object group
 *   read request.
 *  NOTE: Each AnlgIn pointNum x may send Obj30 or Obj31 in response to
 *   a Class0 poll, but not both, as required by TB2018-004. If this function
 *    returns TMWDEFS_TRUE for point x then sdnpdata_anlgInIsClass0 should
 *    return TMWDEFS_FALSE.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  TMWDEFS_TRUE if point should be reported.
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_frznAnlgInIsClass0(
  void *pPoint);

/* function: sdnpdata_frznAnlgInAssignClass
 * purpose: Assign the class in which events from this data point
 *  will belong.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  classMask - new class in which to generate events from this point
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_frznAnlgInAssignClass(
  void *pPoint,
  TMWDEFS_CLASS_MASK classMask);

/* function: sdnpdata_frznAnlgInRead
 * purpose: Return the current value of the specified point.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  pValue - pointer to location to store current value
 *   function should set pValue->type to indicate data type returned
 *     TMWTYPES_ANALOG_TYPE_DOUBLE
 *     TMWTYPES_ANALOG_TYPE_SFLOAT
 *     TMWTYPES_ANALOG_TYPE_LONG
 *     TMWTYPES_ANALOG_TYPE_SHORT
 *     TMWTYPES_ANALOG_TYPE_SCALED - allows database to return a single
 *      precision floating point value to support a read of sfloat, AND a
 *      long value to support a read of a scaled long representation of
 *      the sfloat. 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.
 *     TMWTYPES_ANALOG_TYPE_DSCALED - allows database to return a double
 *      precision floating point value to support a read of double, AND a
 *      long value to support a read of a scaled long representation of
 *      the double. 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.
 *   NOTE:
 *    If TMWCNFG_SUPPORT_DOUBLE == TRUE returning the value as
 *    TMWTYPES_ANALOG_TYPE_DOUBLE allows for proper reads of DOUBLE FLOAT
 *    SFLOAT, ULONG or USHORT.
 *    If TMWCNFG_SUPPORT_DOUBLE == FALSE and
 *       TMWCNFG_SUPPORT_SFLOAT == TRUE
 *    To provide for best precision and OVER_RANGE flag setting:
 *    return the value as TMWTYPES_ANALOG_TYPE_LONG if the value is between
 *    TMWDEFS_LONG_MIN and TMWDEFS_LONG_MAX.
 *    return the value as TMWTYPES_ANALOG_TYPE_SFLOAT if the value is outside
 *    of that range. While each type is 32 bits in size the long allows
 *    for 31 bits of precision. Returning a float instead allows greater values,
 *    but only 23 bits of precision.
 *    If both TMWCNFG_SUPPORT_DOUBLE and TMWCNFG_SUPPORT_SFLOAT == FALSE and
 *    the long value is over or under range, set DNPDEFS_DBAS_FLAG_OVER_RANGE
 *    bit in pFlags and set lval to TMWDEFS_LONG_MIN or TMWDEFS_LONG_MAX.
 *
 *  pFlags - pointer to location to store current DNP3 flags
 *   The following values (or OR'd combinations) are valid for this type:
 *      DNPDEFS_DBAS_FLAG_OFF_LINE - the point is off-line, and the returned
 *        state of this point may not be correct
 *      DNPDEFS_DBAS_FLAG_ON_LINE - the analog 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.
 *        NOTE: If both TMWCNFG_SUPPORT_DOUBLE and TMWCNFG_SUPPORT_SFLOAT ==
 *        FALSE and the long value is over or under range, set
 *        DNPDEFS_DBAS_FLAG_OVER_RANGE bit in pFlags and set lval to
 *        TMWDEFS_LONG_MIN or TMWDEFS_LONG_MAX.
 *      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.
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL sdnpdata_frznAnlgInRead(
  void *pPoint,
  TMWTYPES_ANALOG_VALUE *pValue,
  TMWTYPES_UCHAR *pFlags,
  TMWDTIME *pTimeOfFreeze);

/* function: sdnpdata_frznAnlgInChanged
 * purpose: Determine if the specified point has changed and if so
 *  return the new value. This function is used to scan for events on
 *  each data point. It will be called if SDNPSESN_CONFIG
 *  analogInputScanPeriod is nonzero.
 *  NOTE: this functionality is compiled out by defining
 *  SDNPDATA_SUPPORT_OBJ32 FALSE
 *  or SDNPDATA_SUPPORT_EVENT_SCAN FALSE
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  pValue - pointer to location to store current value
 *   function should set pValue->type to indicate data type returned
 *     TMWTYPES_ANALOG_TYPE_DOUBLE
 *     TMWTYPES_ANALOG_TYPE_SFLOAT
 *     TMWTYPES_ANALOG_TYPE_LONG
 *     TMWTYPES_ANALOG_TYPE_SHORT
 *     TMWTYPES_ANALOG_TYPE_SCALED - allows database to return a single precision
 *      floating point value to support a read of sfloat, AND a long value
 *      to support a read of a scaled long representation of the sfloat.
 *      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.
 *     TMWTYPES_ANALOG_TYPE_DSCALED - allows database to return a double
 *      precision floating point value to support a read of double, AND a
 *      long value to support a read of a scaled long representation of
 *      the double. 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.
 *   NOTE:
 *    If TMWCNFG_SUPPORT_DOUBLE == TRUE returning the value as
 *    TMWTYPES_ANALOG_TYPE_DOUBLE allows for proper values for events of type
 *    DOUBLE FLOAT, SFLOAT, ULONG or USHORT.
 *
 *    If TMWCNFG_SUPPORT_DOUBLE == FALSE and
 *       TMWCNFG_SUPPORT_SFLOAT == TRUE
 *    To provide for best precision and OVER_RANGE flag setting:
 *    return the value as TMWTYPES_ANALOG_TYPE_LONG if the value is between
 *    TMWDEFS_LONG_MIN and TMWDEFS_LONG_MAX.
 *    return the value as TMWTYPES_ANALOG_TYPE_SFLOAT if the value is outside
 *    of that range. While each type is 32 bits in size the long allows
 *    for 31 bits of precision. Returning a float instead allows greater values,
 *    but only 23 bits of precision.
 *    If both TMWCNFG_SUPPORT_DOUBLE and TMWCNFG_SUPPORT_SFLOAT == FALSE and
 *    the long value is over or under range, set DNPDEFS_DBAS_FLAG_OVER_RANGE
 *    bit in pFlags and set lval to TMWDEFS_LONG_MIN or TMWDEFS_LONG_MAX.
 *
 *  pFlags - pointer to location to store current DNP3 flags
 *   The following values (or OR'd combinations) are valid for this type:
 *      DNPDEFS_DBAS_FLAG_OFF_LINE - the point is off-line, and the returned
 *        state of this point may not be correct
 *      DNPDEFS_DBAS_FLAG_ON_LINE - the analog 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.
 *        NOTE: If both TMWCNFG_SUPPORT_DOUBLE and TMWCNFG_SUPPORT_SFLOAT ==
 *        FALSE and the long value is over or under range, set
 *        DNPDEFS_DBAS_FLAG_OVER_RANGE bit in pFlags and set lval to
 *        TMWDEFS_LONG_MIN or TMWDEFS_LONG_MAX.
 *      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.
 * returns:
 *  TMWDEFS_TRUE if the point has changed, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_frznAnlgInChanged(
  void *pPoint,
  TMWTYPES_ANALOG_VALUE *pValue,
    TMWTYPES_UCHAR *pFlags);

#if SDNPDATA_SUPPORT_OBJ34
  /* Analog Input Deadbands */

  /* function: sdnpdata_anlgInDBandGetDescription
   * purpose: Get description of this point
   *  NOTE: this is used only by sdnpxml/sdnpxml2.c if SDNPDATA_SUPPORT_XMLxx is
TRUE
   *  code to generate the configuration file or Device Profile for this device.
   * arguments:
   *  pPoint - handle to data point returned from 'getPoint' function.
   * returns:
   *  pointer to string describing point
   *  TMWDEFS_NULL if failure
   */
  TMWTYPES_CHAR * TMWDEFS_GLOBAL sdnpdata_anlgInDBandGetDescription(
    void *pPoint);

  /* function: sdnpdata_anlgInDBandQuantity
   * purpose: Return the number of analog input deadbands in the
   *  specified database.
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   * returns:
   *  The number of analog input deadbands (see note for
   *  sdnpdata_xxxQuantity at top of this file)
   */
  TMWTYPES_USHORT TMWDEFS_GLOBAL sdnpdata_anlgInDBandQuantity(
    void *pHandle);

  /* function: sdnpdata_anlgInDBandGetPoint
   * purpose: Return a handle to a specific data point. This handle is
   *  used by the routines below to read and write values and control
   *  information to this point.
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  pointNum - point number to return
   * returns:
   *  Pointer to specified data point or TMWDEFS_NULL if this point
   *   is currently disabled.
   */
  void * TMWDEFS_GLOBAL sdnpdata_anlgInDBandGetPoint(
    void *pHandle,
    TMWTYPES_USHORT pointNum);

  /* function: sdnpdata_anlgInDBandRead
   * purpose: Return the current value of the specified point.
   * arguments:
   *  pPoint - handle to data point returned from 'getPoint' function.
   *  pValue - pointer to location to hold current value
   *   set pValue->type to indicate data type returned.
   *   NOTE:
   *    If TMWCNFG_SUPPORT_DOUBLE == TRUE returning the value as
   *    pValue->type = TMWTYPES_ANALOG_TYPE_DOUBLE allows for reads of
   *    SFLOAT, ULONG or USHORT.
   *
   *    If TMWCNFG_SUPPORT_DOUBLE == FALSE and
   *       TMWCNFG_SUPPORT_SFLOAT == TRUE
   *    To provide for best precision:
   *    return the value as TMWTYPES_ANALOG_TYPE_ULONG if the value is between
   *    0 and TMWDEFS_ULONG_MAX.
   *    return the value as TMWTYPES_ANALOG_TYPE_SFLOAT if the value is outside
   *    of that range. While each is 32 bits in size the ulong allows
   *    for 32 bits of precision. Returning a float instead allows greater values,
   *    but only 23 bits of precision.
   *    If both TMWCNFG_SUPPORT_DOUBLE and TMWCNFG_SUPPORT_SFLOAT == FALSE and
   *    the long value exceeds TMWDEFS_ULONG_MAX, return TMWDEFS_ULONG_MAX.
   *   NOTE: Deadband values should be greater than zero. Negative numbers are not
   *    allowed by the specification.
   * returns:
   *  void
   */
  void TMWDEFS_GLOBAL sdnpdata_anlgInDBandRead(
    void *pPoint,
    TMWTYPES_ANALOG_VALUE *pValue);

  /* function: sdnpdata_anlgInDBandWrite
   * purpose: Write the specified deadband value
   * arguments:
   *  pPoint - handle to data point returned from 'getPoint' function.
   *  *pValue - pointer to new deadband value to be stored
   *   pValue->type indicates storage type
   *    TMWTYPES_ANALOG_TYPE_SFLOAT
   *    TMWTYPES_ANALOG_TYPE_ULONG
   *    TMWTYPES_ANALOG_TYPE_USHORT
   * returns:
   *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_anlgInDBandWrite(
    void *pPoint,
    TMWTYPES_ANALOG_VALUE *pValue);
 #endif

  /* Analog Outputs */

  /* function: sdnpdata_anlgOutGetDescription
   * purpose: Get description of this point
   *  NOTE: this is used only by sdnpxml/sdnpxml2.c if SDNPDATA_SUPPORT_XMLxx is
 TRUE
   *  code to generate the configuration file or Device Profile for this device.
   * arguments:
   *  pPoint - handle to data point returned from 'getPoint' function.
   * returns:
   *  pointer to string describing point
   *  TMWDEFS_NULL if failure
   */
  TMWTYPES_CHAR * TMWDEFS_GLOBAL sdnpdata_anlgOutGetDescription(
    void *pPoint);

  /* function: sdnpdata_anlgOutQuantity
   * purpose: Return the number of analog output data points in the
   *  specified database.
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   * returns:
   *  The number of analog outputs data points (see note for
   *  sdnpdata_xxxQuantity at top of this file)
   */
  TMWTYPES_USHORT TMWDEFS_GLOBAL sdnpdata_anlgOutQuantity(
    void *pHandle);

  /* function: sdnpdata_anlgOutGetPoint
   * purpose: Return a handle to a specific data point. This handle is
   *  used by the routines below to read and write values and control
   *  information to this point.
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  pointNum - point number to return
   * returns:
   *  Pointer to specified data point or TMWDEFS_NULL if this point
   *   is currently disabled.
   */
  void * TMWDEFS_GLOBAL sdnpdata_anlgOutGetPoint(
    void *pHandle,
    TMWTYPES_USHORT pointNum);

  /* function: sdnpdata_anlgOutDefVariation
   * purpose: Determine default static variation for this analog output
   * arguments:
   *  pPoint - handle to data point returned from 'getPoint' function.
   * returns:
   *  default variation for this point
   *  NOTE: this will only be called if the default variation for
   *  analog output obj40DefaultVariation is configured as 0.
   */
  TMWTYPES_UCHAR TMWDEFS_GLOBAL sdnpdata_anlgOutDefVariation(
    void *pPoint);

  /* function: sdnpdata_anlgOutEventDefVariation
   * purpose: Determine default event variation for this analog output
   * arguments:
   *  pPoint - handle to data point returned from 'getPoint' function.
   *  classMask - mask indicating what event class is being requested
   * returns:
   *  default variation for this point
   *  NOTE: this will only be called if the default variation for
   *   analog input change events obj42DefaultVariation is configured as zero
   */
  TMWTYPES_UCHAR TMWDEFS_GLOBAL sdnpdata_anlgOutEventDefVariation(
    void *pPoint,
    TMWDEFS_CLASS_MASK classMask);
```

```
/* function: sdnpdata_anlgOutEventMode
 * purpose: Determine event mode for this point
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  event mode for this point, SOE or LAST
 *  NOTE: this will only be called if the event mode for analog outputs
 *    if analogOutputEventMode is configured as TMWDEFS_EVENT_MODE_PER_POINT
 */
TMWDEFS_EVENT_MODE TMWDEFS_GLOBAL sdnpdata_anlgOutEventMode(
  void *pPoint);

/* function: sdnpdata_anlgOutEventClass
 * purpose: Return the class in which output events from this data point
 *  belong.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  Class in which these events will be returned
 */
TMWDEFS_CLASS_MASK TMWDEFS_GLOBAL sdnpdata_anlgOutEventClass(
  void *pPoint);

/* function: sdnpdata_anlgOutIsClass0
 * purpose: Should this point be reported in response to an object 60
 *   variation 1 read request. This allows individual points to be excluded
 *   from a class 0 response but still readable by a specific object group
 *   read request.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  TMWDEFS_TRUE if point should be reported.
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_anlgOutIsClass0(
  void *pPoint);

/* function: sdnpdata_anlgOutAssignClass
 * purpose: Assign the class in which output events from this data point
 *  will belong.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  classMask - new class in which to generate events from this point
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_anlgOutAssignClass(
  void *pPoint,
  TMWDEFS_CLASS_MASK classMask);

/* function: sdnpdata_anlgOutRead
 * purpose: Return the current value of the specified point.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  pValue - pointer to location to store current value
 *    set pValue->type to indicate data type returned.
 *   NOTE:
 *    If TMWCNFG_SUPPORT_DOUBLE == TRUE returning the value as
 *    TMWTYPES_ANALOG_TYPE_DOUBLE allows for proper events of type
 *    DOUBLE FLOAT SFLOAT, ULONG or USHORT.
 *
 *    If TMWCNFG_SUPPORT_DOUBLE == FALSE and
 *       TMWCNFG_SUPPORT_SFLOAT == TRUE
 *    To provide for best precision and OVER_RANGE flag setting:
 *    return the value as TMWTYPES_ANALOG_TYPE_LONG if the value is between
 *    TMWDEFS_LONG_MIN and TMWDEFS_LONG_MAX.
 *    return the value as TMWTYPES_ANALOG_TYPE_SFLOAT if the value is outside
 *    of that range. While each is 32 bits in size the long allows
 *    for 31 bits of precision. Returning a float instead allows greater values,
 *    but only 23 bits of precision.
 *    If both TMWCNFG_SUPPORT_DOUBLE and TMWCNFG_SUPPORT_SFLOAT == FALSE and
 *    the long value is over or under range, set DNPDEFS_DBAS_FLAG_OVER_RANGE
 *    bit in pFlags and set lval to TMWDEFS_LONG_MIN or TMWDEFS_LONG_MAX.
 *
 *  pFlags - pointer to location to store current DNP3 flags
 *   The following values (or OR'd combinations) are valid for this type:
 *      DNPDEFS_DBAS_FLAG_OFF_LINE - the point is off-line, and the returned
 *        state of this point may not be correct
 *      DNPDEFS_DBAS_FLAG_ON_LINE - the analog output 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 analog object
 *        has been forced to its current state at the originating device
 *      DNPDEFS_DBAS_FLAG_LOCAL_FORCED - the state of the analog object
 *        has been forced to its current state at the device reporting
 *      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.
 *        NOTE: If both TMWCNFG_SUPPORT_DOUBLE and TMWCNFG_SUPPORT_SFLOAT ==
 *        FALSE and the long value is over or under range, set
 *        DNPDEFS_DBAS_FLAG_OVER_RANGE bit in pFlags and set lval to
 *        TMWDEFS_LONG_MIN or TMWDEFS_LONG_MAX.
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL sdnpdata_anlgOutRead(
  void *pPoint,
  TMWTYPES_ANALOG_VALUE *pValue,
  TMWTYPES_UCHAR *pFlags);

/* function: sdnpdata_anlgOutChanged
 * purpose: Determine if the specified point has changed and if so
 *  return the new value. This function is used to scan for events on
 *  each data point. This function is called to determine whether an Object
 *  Group 42 Analog Output Event should be queued. It will be called if
 *  SDNPSESN_CONFIG analogOutputScanPeriod is nonzero.
 *  NOTE: this functionality is compiled out by defining
 *  SDNPDATA_SUPPORT_OBJ42 FALSE (default)
 *  or SDNPDATA_SUPPORT_EVENT_SCAN FALSE
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  pValue - pointer to location to store current value
 *    set pValue->type to indicate data type returned.
 *   NOTE:
 *    If TMWCNFG_SUPPORT_DOUBLE == TRUE returning the value as
 *    TMWTYPES_ANALOG_TYPE_DOUBLE allows for proper events of type
 *    DOUBLE FLOAT SFLOAT, ULONG or USHORT.
 *
 *    If TMWCNFG_SUPPORT_DOUBLE == FALSE and
 *       TMWCNFG_SUPPORT_SFLOAT == TRUE
 *    To provide for best precision and OVER_RANGE flag setting:
 *    return the value as TMWTYPES_ANALOG_TYPE_LONG if the value is between
 *    TMWDEFS_LONG_MIN and TMWDEFS_LONG_MAX.
 *    return the value as TMWTYPES_ANALOG_TYPE_SFLOAT if the value is outside
 *    of that range. While each is 32 bits in size the long allows
 *    for 31 bits of precision. Returning a float instead allows greater values,
 *    but only 23 bits of precision.
 *    If both TMWCNFG_SUPPORT_DOUBLE and TMWCNFG_SUPPORT_SFLOAT == FALSE and
 *    the long value is over or under range, set DNPDEFS_DBAS_FLAG_OVER_RANGE
 *    bit in pFlags and set lval to TMWDEFS_LONG_MIN or TMWDEFS_LONG_MAX.
 *
 *  pFlags - pointer to location to store current DNP3 flags
 *   The following values (or OR'd combinations) are valid for this type:
 *      DNPDEFS_DBAS_FLAG_OFF_LINE - the point is off-line, and the returned
 *        state of this point may not be correct
 *      DNPDEFS_DBAS_FLAG_ON_LINE - the analog output 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 device
 *        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 analog object
 *        has been forced to its current state at the originating device
 *      DNPDEFS_DBAS_FLAG_LOCAL_FORCED - the state of the analog object
 *        has been forced to its current state at the device reporting
 *      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.
 *        NOTE: If both TMWCNFG_SUPPORT_DOUBLE and TMWCNFG_SUPPORT_SFLOAT ==
 *        FALSE and the long value is over or under range, set
 *        DNPDEFS_DBAS_FLAG_OVER_RANGE bit in pFlags and set lval to
 *        TMWDEFS_LONG_MIN or TMWDEFS_LONG_MAX.
 * returns:
 *  TMWDEFS_TRUE if the point has changed, else TMWDEFS_FALSE
 */
```

```
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_anlgOutChanged(
  void *pPoint,
  TMWTYPES_ANALOG_VALUE *pValue,
  TMWTYPES_UCHAR *pFlags);

/* function: sdnpdata_anlgOutSelect
 * purpose: perform Select operation on the specified point and return
 *   the result.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  pValue - pointer to location to containing value
 *   pValue->type will indicate storage type
 *    TMWTYPES_ANALOG_TYPE_LONG
 *    TMWTYPES_ANALOG_TYPE_SHORT
 *    TMWTYPES_ANALOG_TYPE_SFLOAT
 *    TMWTYPES_ANALOG_TYPE_DOUBLE
 * returns:
 *  status of Analog Output operation. Valid values are:
 *    DNPDEFS_CTLSTAT_SUCCESS - the command was performed successfully
 *    DNPDEFS_CTLSTAT_FORMAT_ERROR - The request was not accepted due
 *      to formatting errors in the request. This value may also be use
 *      to indicate that the value in the request exceeds the permitted
 *      level (see Table 4-2 in DNP3 Specification Volume 2, Application
 *      Layer.)
 *    DNPDEFS_CTLSTAT_NOT_SUPPORTED - the request was not accepted because
 *      Control operations are not supported for this point
 *    DNPDEFS_CTLSTAT_ALREADY_ACTIVE - the request was not accepted because
 *      the control queue is full or the point is already active
 *    DNPDEFS_CTLSTAT_HARDWARE_ERROR - the request was not accepted due to
 *      control hardware problems
 *    DNPDEFS_CTLSTAT_LOCAL - the request was not accepted because
 *      Local/Remote switch is in Local Position
 *    DNPDEFS_CTLSTAT_NOT_AUTHORIZED - the request was not accepted because
 *      of insufficient authorization.
 *    DNPDEFS_CTLSTAT_AUTO_INHIBIT - the request was not accepted because it
 *      was prevented or inhibited by a local automation process
 *    DNPDEFS_CTLSTAT_PROC_LIMITED - the request was not accepted because the
 *     device cannot process any more activities than are presently in progress
 *    DNPDEFS_CTLSTAT_OUT_OF_RANGE - the request was not accepted because the
 *     value is outside the acceptable range permitted for this point.
 *    DNPDEFS_CTLSTAT_DOWNSTRM_LOCAL - the request was not accepted because the
 *     control is being forwarded to a downstream device that is reporting Local.
 *    DNPDEFS_CTLSTAT_ALR_COMPLETE - the request was not accepted because the
 *     operation is already complete. For example if the request is to close a
 *     switch and the switch is already closed.
 *    DNPDEFS_CTLSTAT_BLOCKED - the request was not accepted because it is
 *     specifically blocked at the outstation.
 *    DNPDEFS_CTLSTAT_CANCELLED - the request was not accepted because the
 *     operation was cancelled.
 *    DNPDEFS_CTLSTAT_BLOCKED_OM - the request was not accepted because another
 *     master has exclusive rights to operate this point.
 *    DNPDEFS_CTLSTAT_DOWNSTRM_FAIL -  the request was not accepted because the
 *     or control is being forwarded to a downstream device which cannot be
reached
 *     is otherwise incapable of performing the request.
 *    DNPDEFS_CTLSTAT_UNDEFINED - the request not accepted because of some
 *     other undefined reason
 *
 *   NOTE- if pValue->type TMWTYPES_ANALOG_TYPE_LONG is passed for a point that
 *     only supported 16 bit outputs, you could return
 *     DNPDEFS_CTLSTAT_OUT_OF_RANGE
 */
DNPDEFS_CTLSTAT TMWDEFS_GLOBAL sdnpdata_anlgOutSelect(
  void *pPoint,
  TMWTYPES_ANALOG_VALUE *pValue);

/* function: sdnpdata_anlgOutOperate
 * purpose: perform Operate operation on the specified point and return
 *   the result. If Object 43 Analog Output Command Events are desired
 *   you may call sdnpo043_addEvent() from this function.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  pValue - pointer to location to containing value
 *   pValue->type will indicate storage type
 *    TMWTYPES_ANALOG_TYPE_LONG
 *    TMWTYPES_ANALOG_TYPE_SHORT
 *    TMWTYPES_ANALOG_TYPE_SFLOAT
 *    TMWTYPES_ANALOG_TYPE_DOUBLE
 * returns:
 *  status of Analog Output operation. Valid values are:
 *    DNPDEFS_CTLSTAT_SUCCESS - the command was performed successfully
 *    DNPDEFS_CTLSTAT_FORMAT_ERROR - The request was not accepted due
 *      to formatting errors in the request. This value may also be use
 *      to indicate that the value in the request exceeds the permitted
 *      level (see Table 4-2 in DNP3 Specification Volume 2, Application
 *      Layer.)
 *    DNPDEFS_CTLSTAT_NOT_SUPPORTED - the request was not accepted because
 *      Control operations are not supported for this point
 *    DNPDEFS_CTLSTAT_ALREADY_ACTIVE - the request was not accepted because
 *      the control queue is full or the point is already active
 *    DNPDEFS_CTLSTAT_HARDWARE_ERROR - the request was not accepted due to
 *      control hardware problems
 *    DNPDEFS_CTLSTAT_LOCAL - the request was not accepted because
 *      Local/Remote switch is in Local Position
 *    DNPDEFS_CTLSTAT_NOT_AUTHORIZED - the request was not accepted because
 *      of insufficient authorization.
 *    DNPDEFS_CTLSTAT_AUTO_INHIBIT - the request was not accepted because it
 *      was prevented or inhibited by a local automation process
 *    DNPDEFS_CTLSTAT_PROC_LIMITED - the request was not accepted because the
 *     device cannot process any more activities than are presently in progress
 *    DNPDEFS_CTLSTAT_OUT_OF_RANGE - the request was not accepted because the
 *     value is outside the acceptable range permitted for this point.
 *    DNPDEFS_CTLSTAT_DOWNSTRM_LOCAL - the request was not accepted because the
 *     control is being forwarded to a downstream device that is reporting Local.
 *    DNPDEFS_CTLSTAT_ALR_COMPLETE - the request was not accepted because the
 *     operation is already complete. For example if the request is to close a
 *     switch and the switch is already closed.
 *    DNPDEFS_CTLSTAT_BLOCKED - the request was not accepted because it is
 *     specifically blocked at the outstation.
 *    DNPDEFS_CTLSTAT_CANCELLED - the request was not accepted because the
 *     operation was cancelled.
 *    DNPDEFS_CTLSTAT_BLOCKED_OM - the request was not accepted because another
 *     master has exclusive rights to operate this point.
 *    DNPDEFS_CTLSTAT_DOWNSTRM_FAIL -  the request was not accepted because the
 *     control is being forwarded to a downstream device which cannot be reached
 *     or is otherwise incapable of performing the request.
 *    DNPDEFS_CTLSTAT_UNDEFINED - the request not accepted because of some
 *     other undefined reason
 *
 *   NOTE- if pValue->type TMWTYPES_ANALOG_TYPE_LONG is passed for a point that
 *     only supported 16 bit outputs, you could return
 *     DNPDEFS_CTLSTAT_OUT_OF_RANGE
 */
DNPDEFS_CTLSTAT TMWDEFS_GLOBAL sdnpdata_anlgOutOperate(
  void *pPoint,
  TMWTYPES_ANALOG_VALUE *pValue);

/* function: sdnpdata_anlgOutCancelSelect
 * purpose: cancel outstanding Select on the specified point
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL sdnpdata_anlgOutCancelSelect(
  void *pPoint);

/* function: sdnpdata_anlgOutEventCmdDefVariation
 * purpose: Determine default variation for this frozen counter
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  classMask - mask indicating what event class is being requested
 * returns:
 *  default variation for this point
 *  NOTE: this will only be called if the default variation for
 *   analog input change events obj43DefaultVariation is configured as zero
 */
TMWTYPES_UCHAR TMWDEFS_GLOBAL sdnpdata_anlgOutCmdEventDefVariation(
  void *pPoint,
  TMWDEFS_CLASS_MASK classMask);

/* function: sdnpdata_anlgOutCmdEventMode
 * purpose: Determine event mode for this point
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  event mode for this point, SOE or LAST
 *  NOTE: this will only be called if the event mode for analog output commands
 *    if analogOutCmdEventMode is configured as TMWDEFS_EVENT_MODE_PER_POINT
 */
TMWDEFS_EVENT_MODE TMWDEFS_GLOBAL sdnpdata_anlgOutCmdEventMode(
  void *pPoint);
```

```
/* function: sdnpdata_anlgOutCmdEventClass
 * purpose: Return the class in which analog output command events for
 *  this data point belong.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  Class in which these events will be returned
 */
TMWDEFS_CLASS_MASK TMWDEFS_GLOBAL sdnpdata_anlgOutCmdEventClass(
  void *pPoint);

/* function: sdnpdata_anlgOutCmdAssignClass
 * purpose: Assign the class in which analog output command events for
 *  this data point will belong.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  classMask - new class in which to generate events from this point
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_anlgOutCmdAssignClass(
  void *pPoint,
  TMWDEFS_CLASS_MASK classMask);

/* function: sdnpdata_anlgOutCmdChanged
 * purpose: Determine if the command status for the specified point has
 *  changed and if so return the new value. This function is used to scan
 *  for events on each data point. This function is called to determine
 *  whether an Object Group 43 Analog Output Command Event should be queued.
 *  It will be called if SDNPSESN_CONFIG analogOutCmdScanPeriod is nonzero.
 *  NOTE: this functionality is compiled out by defining
 *  SDNPDATA_SUPPORT_OBJ43 FALSE (default)
 *  or SDNPDATA_SUPPORT_EVENT_SCAN FALSE
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  pValue - pointer to location to store current value
 *    set pValue->type to indicate data type returned.
 *   NOTE:
 *    If TMWCNFG_SUPPORT_DOUBLE == TRUE returning the value as
 *    TMWTYPES_ANALOG_TYPE_DOUBLE allows for proper events of type
 *    DOUBLE FLOAT SFLOAT, ULONG or USHORT.
 *
 *    If TMWCNFG_SUPPORT_DOUBLE == FALSE and
 *       TMWCNFG_SUPPORT_SFLOAT == TRUE
 *    To provide for best precision and OVER_RANGE flag setting:
 *    return the value as TMWTYPES_ANALOG_TYPE_LONG if the value is between
 *    TMWDEFS_LONG_MIN and TMWDEFS_LONG_MAX.
 *    return the value as TMWTYPES_ANALOG_TYPE_SFLOAT if the value is outside
 *    of that range. While each is 32 bits in size the long allows
 *    for 31 bits of precision. Returning a float instead allows greater values,
 *    but only 23 bits of precision.
 *    If both TMWCNFG_SUPPORT_DOUBLE and TMWCNFG_SUPPORT_SFLOAT == FALSE and
 *    the long value is over or under range, set DNPDEFS_DBAS_FLAG_OVER_RANGE
 *    bit in pFlags and set lval to TMWDEFS_LONG_MIN or TMWDEFS_LONG_MAX.
 *
 *  pStatus - pointer to location to store current command status
 * returns:
 *  TMWDEFS_TRUE if the point has changed, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_anlgOutCmdChanged(
  void *pPoint,
  TMWTYPES_ANALOG_VALUE *pValue,
  TMWTYPES_UCHAR *pStatus);


/* Double Bit Inputs */

/* function: sdnpdata_dblInGetDescription
 * purpose: Get description of this point
 *  NOTE: this is used only by sdnpxml/sdnpxml2.c if SDNPDATA_SUPPORT_XMLxx is
TRUE
 *  code to generate the configuration file or Device Profile for this device.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  pointer to string describing point
 *  TMWDEFS_NULL if failure
 */
TMWTYPES_CHAR * TMWDEFS_GLOBAL sdnpdata_dblInGetDescription(
  void *pPoint);
```

```
/* function: sdnpdata_dblInQuantity
 * purpose: Return the number of binary input data points in the
 *  specified database.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 * returns:
 *  The number of binary input data points (see note for
 *  sdnpdata_xxxQuantity at top of this file)
 */
TMWTYPES_USHORT TMWDEFS_CALLBACK sdnpdata_dblInQuantity(
  void *pHandle);

/* function: sdnpdata_dblInGetPoint
 * purpose: Return a handle to a specific data point. This handle is
 *  used by the routines below to read and write values and control
 *  information to this point.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  pointNum - point number to return
 * returns:
 *  Pointer to specified data point or TMWDEFS_NULL if this point
 *   is currently disabled.
 */
void * TMWDEFS_CALLBACK sdnpdata_dblInGetPoint(
  void *pHandle,
  TMWTYPES_USHORT pointNum);

/* function: sdnpdata_dblInDefVariation
 * purpose: Determine default static variation for this binary input
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  default variation for this point
 *  NOTE: this will only be called if the default variation for double bit
 *   inputs obj03DefaultVariation is configured as zero
 */
TMWTYPES_UCHAR TMWDEFS_GLOBAL sdnpdata_dblInDefVariation(
  void *pPoint);

/* function: sdnpdata_dblInEventMode
 * purpose: Determine event mode for this point
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  event mode for this point, SOE or LAST
 *  NOTE: this will only be called if the event mode double bit inputs
 *    if doubleInputEventMode is configured as TMWDEFS_EVENT_MODE_PER_POINT
 */
TMWDEFS_EVENT_MODE TMWDEFS_GLOBAL sdnpdata_dblInEventMode(
  void *pPoint);

/* function: sdnpdata_dblInEventClass
 * purpose: Return the class in which events from this data point
 *  belong.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  Class in which these events will be returned
 */
TMWDEFS_CLASS_MASK TMWDEFS_GLOBAL sdnpdata_dblInEventClass(
  void *pPoint);

 /* function: sdnpdata_dblInIsClass0
  * purpose: Should this point be reported in response to an object 60
  *   variation 1 read request. This allows individual points to be excluded
  *   from a class 0 response but still readable by a specific object group
  *   read request.
  * arguments:
  *  pPoint - handle to data point returned from 'getPoint' function.
  * returns:
  *  TMWDEFS_TRUE if point should be reported.
  */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_dblInIsClass0(
  void *pPoint);

/* function: sdnpdata_dblInEventDefVariation
 * purpose: Determine default variation for this frozen counter
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  classMask - mask indicating what event class is being requested
 * returns:
 *  default variation for this point
 *  NOTE: this will only be called if the default variation for binary
 *   input change events obj04DefaultVariation is configured as zero
 */
TMWTYPES_UCHAR TMWDEFS_GLOBAL sdnpdata_dblInEventDefVariation(
  void *pPoint,
  TMWDEFS_CLASS_MASK classMask);

/* function: sdnpdata_dblInAssignClass
 * purpose: Assign the class in which events from this data point
 *  will belong.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  classMask - new class in which to generate events from this point
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_dblInAssignClass(
  void *pPoint,
  TMWDEFS_CLASS_MASK classMask);

/* function: sdnpdata_dblInRead
 * purpose: Return the current value of the specified point.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  pFlags - pointer to location to store current DNP3 flags and value.
 *   pFlags contains a status indication and the current state of the point.
 *   The following values (or OR'd combinations) are valid for this type:
 *      DNPDEFS_DBAS_FLAG_OFF_LINE - the point is off-line, and the returned
 *        state of this point may not be correct
 *      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)
 * returns:
 *  void
 */
TMWDEFS_SCL_API void TMWDEFS_GLOBAL sdnpdata_dblInRead(
  void *pPoint,
  TMWTYPES_UCHAR *pFlags);

/* function: sdnpdata_dblInChanged
 * purpose: Determine if the specified point has changed and if so
 *  return the new value. This function is used to scan for events on
 *  each data point. It will be called if SDNPSESN_CONFIG
 *  doubleInputScanPeriod is nonzero.
 *  NOTE: this functionality is compiled out by defining
 *  SDNPDATA_SUPPORT_OBJ4 FALSE
 *  or SDNPDATA_SUPPORT_EVENT_SCAN FALSE
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  pFlags - pointer to location to store current DNP3 flags and value
 *   pFlags contains a status indication and the current state of the point.
 *   The following values (or OR'd combinations) are valid for this type:
 *      DNPDEFS_DBAS_FLAG_OFF_LINE - the point is off-line, and the returned
 *        state of this point may not be correct
 *      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)
 * returns:
 *  TMWDEFS_TRUE if the point has changed, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_dblInChanged(
  void *pPoint,
  TMWTYPES_UCHAR *pFlags);

/* Time and date - indexed abs time and long interval */

/* function: sdnpdata_indexedTimeGetDescription
 * purpose: Get description of this point
 *  NOTE: this is used only by sdnpxml/sdnpxml2.c if SDNPDATA_SUPPORT_XMLxx is
TRUE
 *  code to generate the configuration file or Device Profile for this device.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  pointer to string describing point
 *  TMWDEFS_NULL if failure
 */
TMWTYPES_CHAR * TMWDEFS_GLOBAL sdnpdata_indexedTimeGetDescription(
  void *pPoint);

/* function: sdnpdata_indexedTimeQuantity
 * purpose: Return the number of indexed abs time data points in the
 *  specified database.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 * returns:
 *  The number of indexed abs time data points (see note for
 *  sdnpdata_xxxQuantity at top of this file)
 */
TMWTYPES_USHORT TMWDEFS_GLOBAL sdnpdata_indexedTimeQuantity(
  void *pHandle);

/* function: sdnpdata_indexedTimeGetPoint
 * purpose: Return a handle to a specific data point. This handle is
 *  used by the routines below to read and write values and control
 *  information to this point.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  pointNum - point number to return
 * returns:
 *  Pointer to specified data point or TMWDEFS_NULL if this point
 *   is currently disabled.
 */
void * TMWDEFS_GLOBAL sdnpdata_indexedTimeGetPoint(
  void *pHandle,
  TMWTYPES_USHORT pointNum);

/* function: sdnpdata_indexedTimeIsClass0
* purpose: Should this point be reported in response to an object 50
*   variation 1 read request. This allows individual points to be excluded
*   from a class 0 response but still readable by a specific object group
*   read request.
* arguments:
*  pPoint - handle to data point returned from 'getPoint' function.
* returns:
*  TMWDEFS_TRUE if point should be reported.
*/
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_indexedTimeIsClass0(
  void *pPoint);
```

```
/* function: sdnpdata_indexedTimeRead
 * purpose: Return the current value of the specified point.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  pTimeStamp - pointer to the time at which the action shall initially occur.
 *  pIntervalCount - pointer to the interval after the initial occurrence.
 *  pIntervalUnit - pointer to the units of the interval between actions.
 * returns:
 *  void
 */
TMWDEFS_SCL_API void TMWDEFS_GLOBAL sdnpdata_indexedTimeRead(
  void *pPoint,
  TMWDTIME *pTimeStamp,
  TMWTYPES_ULONG *pIntervalCount,
  TMWTYPES_BYTE  *pIntervalUnit);

/* function: sdnpdata_indexedTimeWrite
 * purpose: perform write operation on the specified indexed abs time point and
 *   return the result. Write to object group 50 variation 4 was received.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  pTimeStamp - pointer to the time at which the action shall initially occur.
 *  pIntervalCount - pointer to the interval after the initial occurrence.
 *  pIntervalUnit - pointer to the units of the interval between actions.
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_indexedTimeWrite(
  void *pPoint,
  TMWDTIME pTimeStamp,
  TMWTYPES_ULONG pIntervalCount,
  TMWTYPES_BYTE  pIntervalUnit);

/* Strings */

/* function: sdnpdata_strGetDescription
 * purpose: Get description of this point
 *  NOTE: this is used only by sdnpxml/sdnpxml2.c if SDNPDATA_SUPPORT_XMLxx is TRUE
 *  code to generate the configuration file or Device Profile for this device.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  pointer to string describing point
 *  TMWDEFS_NULL if failure
 */
TMWTYPES_CHAR * TMWDEFS_GLOBAL sdnpdata_strGetDescription(
  void *pPoint);

/* function: sdnpdata_strQuantity
 * purpose: Return the number of string data points in the
 *  specified database.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 * returns:
 *  The number of string data points (see note for
 *  sdnpdata_xxxQuantity at top of this file)
 */
TMWTYPES_USHORT TMWDEFS_GLOBAL sdnpdata_strQuantity(
  void *pHandle);

/* function: sdnpdata_strGetPoint
 * purpose: Return a handle to a specific data point. This handle is
 *  used by the routines below to read and write values and control
 *  information to this point.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  pointNum - point number to return
 * returns:
 *  Pointer to specified data point or TMWDEFS_NULL if this point
 *   is currently disabled.
 */
void * TMWDEFS_GLOBAL sdnpdata_strGetPoint(
  void *pHandle,
  TMWTYPES_USHORT pointNum);

/* function: sdnpdata_strEventMode
 * purpose: Determine event mode for this point
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  event mode for this point, SOE or LAST
 *  NOTE: this will only be called if the event mode for octet strings
 *   if stringEventMode is configured as TMWDEFS_EVENT_MODE_PER_POINT
 */
TMWDEFS_EVENT_MODE TMWDEFS_GLOBAL sdnpdata_strEventMode(
  void *pPoint);

/* function: sdnpdata_strEventClass
 * purpose: Return the class in which events from this data point
 *  belong.
 * arguments:
 *  pHandle - handle to data point returned from 'getPoint' function.
 * returns:
 *  Class in which these events will be returned
 */
TMWDEFS_CLASS_MASK TMWDEFS_GLOBAL sdnpdata_strEventClass(
  void *pPoint);

 /* function: sdnpdata_strIsClass0
  * purpose: Should this point be reported in response to an object 60
  *   variation 1 read request. This allows individual points to be excluded
  *   from a class 0 response but still readable by a specific object group
  *   read request.
  * arguments:
  *  pPoint - handle to data point returned from 'getPoint' function.
  * returns:
  *  TMWDEFS_TRUE if point should be reported.
  */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_strIsClass0(
  void *pPoint);

/* function: sdnpdata_strAssignClass
 * purpose: Assign the class in which events from this data point
 *  will belong.
 * arguments:
 *  pHandle - handle to data point returned from 'getPoint' function.
 *  classMask - new class in which to generate events from this point
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_strAssignClass(
  void *pPoint,
  TMWDEFS_CLASS_MASK classMask);

/* function: sdnpdata_strRead
 * purpose: Return the current value of the specified point.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function
 *  maxLength - maximum length of buffer to read.
 *  pBuf - pointer to buffer into which to store string.
 *  pLength - pointer to location into which to store string length.
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL sdnpdata_strRead(
  void *pPoint,
  TMWTYPES_UCHAR maxLength,
  TMWTYPES_UCHAR *pBuf,
  TMWTYPES_UCHAR *pLength);

TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_strWrite(
  void *pPoint,
  TMWTYPES_UCHAR *pBuf,
  TMWTYPES_UCHAR bufLength);

/* function: sdnpdata_strChanged
 * purpose: Determine if the specified point has changed and if so
 *  return the new value. This function is used to scan for events on
 *  each data point. It will be called if SDNPSESN_CONFIG
 *  stringScanPeriod is nonzero.
 *  NOTE: this functionality is compiled out by defining
 *  SDNPDATA_SUPPORT_OBJ111 FALSE
 *  or SDNPDATA_SUPPORT_EVENT_SCAN FALSE
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  pFlags - pointer to location to store current DNP3 flags
 *   See description above.
 * returns:
 *  TMWDEFS_TRUE if the point has changed, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_strChanged(
  void *pPoint,
  TMWTYPES_UCHAR maxLength,
  TMWTYPES_UCHAR *pBuf,
  TMWTYPES_UCHAR *pLength);

/* Virtual Terminal */

/* function: sdnpdata_vtermGetDescription
 * purpose: Get description of this point
 *  NOTE: this is used only by sdnpxml/sdnpxml2.c if SDNPDATA_SUPPORT_XMLxx is
TRUE
 *  code to generate the configuration file or Device Profile for this device.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  pointer to string describing point
 *  TMWDEFS_NULL if failure
 */
TMWTYPES_CHAR * TMWDEFS_GLOBAL sdnpdata_vtermGetDescription(
  void *pPoint);

/* function: sdnpdata_vtermQuantity
 * purpose: Return the number of virtual terminals in the
 *  specified database.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 * returns:
 *  The number of virtual terminals (see note for
 *  sdnpdata_xxxQuantity at top of this file)
 */
TMWTYPES_USHORT TMWDEFS_GLOBAL sdnpdata_vtermQuantity(
  void *pHandle);

/* function: sdnpdata_vtermGetPoint
 * purpose: Return a handle to a specific data point. This handle is
 *  used by the routines below to read and write values and control
 *  information to this point.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  pointNum - point number to return
 * returns:
 *  Pointer to specified data point or TMWDEFS_NULL if this point
 *   is currently disabled.
 */
void * TMWDEFS_GLOBAL sdnpdata_vtermGetPoint(
  void *pHandle,
  TMWTYPES_USHORT pointNum);

/* function: sdnpdata_vtermEventMode
 * purpose: Determine event mode for this point
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  event mode for this point, SOE or LAST
 *  NOTE: this will only be called if the event mode for octet strings
 *   if vtermEventMode is configured as TMWDEFS_EVENT_MODE_PER_POINT
 */
TMWDEFS_EVENT_MODE TMWDEFS_GLOBAL sdnpdata_vtermEventMode(
  void *pPoint);

/* function: sdnpdata_vtermEventClass
 * purpose: Return the class in which events from this data point
 *  belong.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  Class in which these events will be returned
 */
TMWDEFS_CLASS_MASK TMWDEFS_GLOBAL sdnpdata_vtermEventClass(
  void *pPoint);

/* function: sdnpdata_vtermAssignClass
 * purpose: Assign the class in which events from this data point
 *  will belong.
 * arguments:
 *  pHandle - handle to data point returned from 'getPoint' function.
 *  classMask - new class in which to generate events from this point
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_vtermAssignClass(
  void *pPoint,
  TMWDEFS_CLASS_MASK classMask);
```

```
/* function: sdnpdata_vtermRead
 * purpose: Return the current value of the specified point.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function
 *  maxLength - maximum length of buffer to read.
 *  pBuf - pointer to buffer into which to store string.
 *  pLength - pointer to location into which to store string length.
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL sdnpdata_vtermRead(
  void *pPoint,
  TMWTYPES_UCHAR maxLength,
  TMWTYPES_UCHAR *pBuf,
  TMWTYPES_UCHAR *pLength);

TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_vtermWrite(
  void *pPoint,
  TMWTYPES_UCHAR *pBuf,
  TMWTYPES_UCHAR bufLength);

/* function: sdnpdata_vtermChanged
 * purpose: Determine if the specified point has changed and if so
 *  return the new value. This function is used to scan for events on
 *  each data point. It will be called if SDNPSESN_CONFIG
 *  virtualTerminalScanPeriod is nonzero.
 *  NOTE: this functionality is compiled out by defining
 *  SDNPDATA_SUPPORT_OBJ113 FALSE
 *  or SDNPDATA_SUPPORT_EVENT_SCAN FALSE
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  maxLength - maximum length of buffer to read.
 *  pBuf - pointer to buffer into which to store string.
 *  pLength - pointer to location into which to store string length.
 * returns:
 *  TMWDEFS_TRUE if the point has changed, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_vtermChanged(
  void *pPoint,
  TMWTYPES_UCHAR maxLength,
  TMWTYPES_UCHAR *pBuf,
  TMWTYPES_UCHAR *pLength);

/* Extended Strings */

/* function: sdnpdata_extStrGetDescription
 * purpose: Get description of this point
 *  NOTE: this is used only by sdnpxml/sdnpxml2.c if SDNPDATA_SUPPORT_XMLxx is
TRUE
 *  code to generate the configuration file or Device Profile for this device.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  pointer to string describing point
 *  TMWDEFS_NULL if failure
 */
TMWTYPES_CHAR * TMWDEFS_GLOBAL sdnpdata_extStrGetDescription(
  void *pPoint);

/* function: sdnpdata_extStrQuantity
 * purpose: Return the number of string data points in the
 *  specified database.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 * returns:
 *  The number of string data points (see note for
 *  sdnpdata_xxxQuantity at top of this file)
 */
TMWTYPES_USHORT TMWDEFS_GLOBAL sdnpdata_extStrQuantity(
  void *pHandle);

/* function: sdnpdata_extStrGetPoint
 * purpose: Return a handle to a specific data point. This handle is
 *  used by the routines below to read and write values and control
 *  information to this point.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  pointNum - point number to return
 * returns:
 *  Pointer to specified data point or TMWDEFS_NULL if this point
 *   is currently disabled.
 */
void * TMWDEFS_GLOBAL sdnpdata_extStrGetPoint(
  void *pHandle,
  TMWTYPES_USHORT pointNum);

/* function: sdnpdata_extStrDefVariation
 * purpose: Determine default static variation for this extended string
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  default variation for this point
 *  NOTE: this will only be called if the default variation for extended
 *   string obj114DefaultVariation is configured as zero
 */
TMWTYPES_UCHAR TMWDEFS_GLOBAL sdnpdata_extStrDefVariation(
  void *pPoint);

/* function: sdnpdata_extStrEventDefVariation
 * purpose: Determine default variation for this extended string
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  classMask - mask indicating what event class is being requested
 * returns:
 *  default variation for this point
 *  NOTE: this will only be called if the default variation for
 *   extended string change events obj115DefaultVariation is configured as zero
 */
TMWTYPES_UCHAR TMWDEFS_GLOBAL sdnpdata_extStrEventDefVariation(
  void *pPoint,
  TMWDEFS_CLASS_MASK classMask);

/* function: sdnpdata_extStrEventMode
 * purpose: Determine event mode for this point
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  event mode for this point, SOE or LAST
 *  NOTE: this will only be called if the event mode for octet strings
 *   if stringEventMode is configured as TMWDEFS_EVENT_MODE_PER_POINT
 */
TMWDEFS_EVENT_MODE TMWDEFS_GLOBAL sdnpdata_extStrEventMode(
  void *pPoint);

/* function: sdnpdata_extStrEventClass
 * purpose: Return the class in which events from this data point
 *  belong.
 * arguments:
 *  pHandle - handle to data point returned from 'getPoint' function.
 * returns:
 *  Class in which these events will be returned
 */
TMWDEFS_CLASS_MASK TMWDEFS_GLOBAL sdnpdata_extStrEventClass(
  void *pPoint);

 /* function: sdnpdata_extStrIsClass0
  * purpose: Should this point be reported in response to an object 60
  *   variation 1 read request. This allows individual points to be excluded
  *   from a class 0 response but still readable by a specific object group
  *   read request.
  * arguments:
  *  pPoint - handle to data point returned from 'getPoint' function.
  * returns:
  *  TMWDEFS_TRUE if point should be reported.
  */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_extStrIsClass0(
  void *pPoint);

/* function: sdnpdata_extStrAssignClass
 * purpose: Assign the class in which events from this data point
 *  will belong.
 * arguments:
 *  pHandle - handle to data point returned from 'getPoint' function.
 *  classMask - new class in which to generate events from this point
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_extStrAssignClass(
  void *pPoint,
  TMWDEFS_CLASS_MASK classMask);
 #if SDNPDATA_SUPPORT_OBJ114_MIN_STACK
  /* function: sdnpdata_extStrGetPtr
   * purpose: Returns a pointer to the extended string value of the specified point.
   * arguments:
   *  pPoint - handle to data point returned from 'getPoint' function
   *  pLength - pointer to location into which to store extended string's length.
   *  pFlags - pointer to location to store current DNP3 flags
   *   See description above.
   * returns:
   *  TMWTYPES_UCHAR * - pointer to the location of the extended string's value.
   */
  TMWTYPES_UCHAR * TMWDEFS_GLOBAL sdnpdata_extStrGetPtr(
    void *pPoint,
    TMWTYPES_USHORT *pLength,
    TMWTYPES_UCHAR *pFlags);

  /* function: sdnpdata_extStrRelease
   * purpose: Release the pointer that was returned by in sdnpdata_extStrGetPtr
   *  The database is free to update the extended string value. The
   *  SCL will not attempt to reference that pointer anymore.
   * arguments:
   *  pPoint - handle to data point returned from 'getPoint' function
   * returns:
   *  TMWDEFS_TRUE - if the database is correctly informed of the pointer's release.
   */
  void TMWDEFS_GLOBAL sdnpdata_extStrRelease(
    void *pPoint);
#else
    /* function: sdnpdata_extStrRead
   * purpose: Return the current value of the specified point.
   * arguments:
   *  pPoint - handle to data point returned from 'getPoint' function
   *  maxLength - maximum length of buffer to read.
   *  pBuf - pointer to buffer into which to store string.
   *  pLength - pointer to location into which to store string length.
   *  pFlags - pointer to location to store current DNP3 flags
   *   See description above.
   * returns:
   *  void
   */
    void TMWDEFS_GLOBAL sdnpdata_extStrRead(
    void *pPoint,
    TMWTYPES_USHORT maxLength,
    TMWTYPES_UCHAR *pBuf,
    TMWTYPES_USHORT *pLength,
    TMWTYPES_UCHAR *pFlags);
#endif

  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_extStrWrite(
    void *pPoint,
    TMWTYPES_UCHAR *pBuf,
    TMWTYPES_USHORT bufLength);

  /* function: sdnpdata_extStrChanged
   * purpose: Determine if the specified point has changed and if so
   *  return the new value. This function is used to scan for events on
   *  each data point. It will be called if SDNPSESN_CONFIG
   *  stringScanPeriod is nonzero.
   *  NOTE: this functionality is compiled out by defining
   *  SDNPDATA_SUPPORT_OBJ115 FALSE
   *  or SDNPDATA_SUPPORT_EVENT_SCAN FALSE
   * arguments:
   *  pPoint - handle to data point returned from 'getPoint' function.
   *  pFlags - pointer to location to store current DNP3 flags
   *   See description above.
   * returns:
   *  TMWDEFS_TRUE if the point has changed, else TMWDEFS_FALSE
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_extStrChanged(
    void *pPoint,
    TMWTYPES_USHORT maxLength,
    TMWTYPES_UCHAR *pBuf,
    TMWTYPES_USHORT *pLength,
    TMWTYPES_UCHAR *pFlags);

  /* function: sdnpdata_fileEventClass
   * purpose: Return the class in which file events belong.
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   * returns:
   *  Class in which these events will be returned
   */
  TMWDEFS_CLASS_MASK TMWDEFS_GLOBAL sdnpdata_fileEventClass(
    void *pHandle);

  /* function: sdnpdata_fileAssignClass
   * purpose: Assign the class in which file transfer response events
   *  will belong.
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  classMask - new class in which to generate events from this
   * returns:
   *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_fileAssignClass(
    void *pHandle,
    TMWDEFS_CLASS_MASK classMask);

  /* function: sdnpdata_getFileInfo
   * purpose: return information about file or directory
   *  This is used to build a response to a FC 28 Get File Info request.
   *   NOTE: you may want to limit file access to only certain files and/or
  directories
   *   based on the filename specified by returning DNPDEFS_FILE_CMD_STAT_DENIED or
   *   other error.
   * arguments:
   *  pSession - pointer to session request was received on
   *  pFilename - null terminated name of file to get info from
   *  pType - return type of file
   *    DNPDEFS_FILE_TYPE_SIMPLE  or
   *    DNPDEFS_FILE_TYPE_DIRECTORY
   *  pSize - return size of simple file in bytes
   *    If this was directory, size is the number of files contained in
   *    the directory excluding links to itself and parent, according to
   *    DNP3 Specification, Application Layer Draft H 6.7.4
   *  pTimeOfCreation - return file or directory time of creation
   *  pPermissions - return file or directory permissions
   *   see DNPDEFS_FILE_PERMISSIONS in dnpdefs.h
   * returns:
   *  DNPDEFS_FILE_CMD_STAT_SUCCESS if successful
   *  DNPDEFS_FILE_CMD_STAT_ASYNC if the SCL should send a NULL response
   *   and later send the response as an event. The target database can call
   *   sdnpo070_fileCmdComplete(); or the SCL will call sdnpdata_getFileInfo
  periodically
   *  DNPDEFS_FILE_CMD_STAT_NOT_FOUND If file does not exist
   *  Other Appropriate DNPDEFS_FILE_CMD_STAT values
   *   DNPDEFS_FILE_CMD_STAT_LOST_COMM
   *   DNPDEFS_FILE_CMD_STAT_MISC
   */
  DNPDEFS_FILE_CMD_STAT TMWDEFS_GLOBAL sdnpdata_getFileInfo(
    TMWSESN *pSession,
    TMWTYPES_CHAR *pFilename,
    DNPDEFS_FILE_TYPE *pType,
    TMWTYPES_ULONG *pSize,
    TMWDTIME *pTimeOfCreation,
    DNPDEFS_FILE_PERMISSIONS *pPermissions);

  /* function: sdnpdata_readFileInfo
   * purpose: return information about file or subdirectory
   *  This is used to build a response to a FC 1 read request on a directory.
   *  This function will be called multiple times. It should set pLast to
   *  TRUE when the last entry in a directory is being read.
   * arguments:
   *  pSession - pointer to session request was received on
   *  handle - DNP file handle returned from previous call to sdnpdata_openFile
   *  maxNameSize - max length of pFileName to be returned
   *    If name won't fit, return DNPDEFS_FILE_TFER_STAT_MISC and SCL
   *    will read this file info again on next request.
   *  pFilename - return null terminated name of file or subdirectory entry
   *  pLast - return TMWDEFS_TRUE if this is the last block in this file
   *  pType - return type of file
   *    DNPDEFS_FILE_TYPE_SIMPLE  or
   *    DNPDEFS_FILE_TYPE_DIRECTORY
   *  pSize - return size of simple file in bytes
   *    If this was directory, size is the number of files contained in
   *    the directory excluding links to itself and parent, according to
   *    DNP3 Specification, Application Layer Draft H 6.7.4
   *  pTimeOfCreation - return file time of creation
   *  pPermissions - return file permissions
   * returns:
   *  DNPDEFS_FILE_TFER_STAT_SUCCESS if successful
   *  DNPDEFS_FILE_TFER_STAT_ASYNC if the SCL should send a NULL response
```

```
     *   and later send the response as an event. The target database can call
     *   sdnpo070_fileCmdComplete(); or the SCL will call sdnpdata_readFileInfo
     *   periodically. When the SCL is reading an entire directory this will be called
     *   repeatedly. The response will not be sent until either all entries or
     *   enough to fill a response are read.
     *  Appropriate DNPDEFS_FILE_TFER_STAT values on error.
     *   DNPDEFS_FILE_TFER_STAT_INV_HANDLE
     *   DNPDEFS_FILE_TFER_STAT_LOST_COMM
     *   DNPDEFS_FILE_TFER_STAT_MISC
     */
    DNPDEFS_FILE_TFER_STAT TMWDEFS_GLOBAL sdnpdata_readFileInfo(
      TMWSESN *pSession,
      TMWTYPES_ULONG handle,
      TMWTYPES_USHORT maxNameSize,
      TMWTYPES_CHAR *pName,
      TMWTYPES_BOOL *pLast,
      DNPDEFS_FILE_TYPE *pType,
      TMWTYPES_ULONG *pSize,
      TMWDTIME *pTimeOfCreation,
      DNPDEFS_FILE_PERMISSIONS *pPermissions);

    /* function: sdnpdata_getAuthentication
     * purpose: return authentication key to be used for next file
     *  operation. If authentication is not supported, or  this routine
     *  should set pAuthKey to 0 and return success.
     * arguments:
     *  pSession - pointer to session request was received on
     *  pUsername - username of user requesting authentication
     *  pPassword - password of user requesting authentication
     *  pAuthKey - return authentication key to be used by master in
     *   open or delete request. If unacceptable make pAuthKey point
     *   to authentication key value of zero.
     * returns:
     *  TMWDEFS_TRUE if authentication key has be returned,
     *  TMWDEFS_FALSE otherwise
     *  NOTE: The DNP specs do not allow a file authentication response to be
     *   delayed and sent later as an event.
     */
    TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_getAuthentication(
      TMWSESN *pSession,
      TMWTYPES_CHAR *pUsername,
      TMWTYPES_CHAR *pPassword,
      TMWTYPES_ULONG *pAuthKey);

    /* function: sdnpdata_deleteFile
     * purpose: delete the requested file
     *   NOTE: you may want to limit delete access to only certain files and/or
directories
     *   based on the filename specified by returning DNPDEFS_FILE_CMD_STAT_DENIED or
     *   other error.
     * arguments:
     *  pSession - pointer to session request was received on
     *  pFilename - null terminated name of file to delete
     *  authKey - authentication key for this request
     * returns:
     *  DNPDEFS_FILE_CMD_STAT_SUCCESS if successful
     *   return DNPDEFS_FILE_CMD_STAT_LOCKED if the specified file is open,
     *    and do not delete the file.
     *  DNPDEFS_FILE_CMD_STAT_ASYNC if the SCL should send a NULL response
     *   and later send the response as an event. The target database can call
     *   sdnpo070_fileCmdComplete(); or the SCL will call sdnpdata_deleteFile
periodically
     *  Other appropriate DNPDEFS_FILE_CMD_STAT values.
     *   DNPDEFS_FILE_CMD_STAT_DENIED
     *   DNPDEFS_FILE_CMD_STAT_NOT_FOUND
     *   DNPDEFS_FILE_CMD_STAT_LOST_COMM
     *   DNPDEFS_FILE_CMD_STAT_MISC
     */
    DNPDEFS_FILE_CMD_STAT TMWDEFS_GLOBAL sdnpdata_deleteFile(
      TMWSESN *pSession,
      TMWTYPES_CHAR *pFilename,
      TMWTYPES_ULONG authKey);

    /* function: sdnpdata_openFile
     * purpose: open specified file
     *   NOTE: you may want to limit access to only certain files and/or directories
     *   based on the filename specified by returning DNPDEFS_FILE_CMD_STAT_DENIED or
     *   other error.
     * arguments:
     *  pSession - pointer to session request was received on
     *  pFilename - null terminated name of file to open. This is the entire
```

```
  *   name string as sent from the master.
  *  authKey - authentication key for this request
  *  mode - access mode for file
  *   DNPDEFS_FILE_MODE_READ
  *   DNPDEFS_FILE_MODE_WRITE
  *   DNPDEFS_FILE_MODE_APPEND
  *  pMaxBlockSize - pointer to proposed maximum block size for this file
  *   this function can modify this to return a reduced maxBlockSize
  *  pPermissions - permissions to give new file if created
  *    NOTE: this data is passed to the database from the SCL. It is NOT
  *          retrieved from the database, it is a pointer to a USHORT
  *          for backward compatibility reasons.
  *  pTimeOfCreation - return time of creation for file
  *  pHandle - return DNP file handle used in DNP g70/v4 and g70/v5 Messages,
  *   non-zero value if successful.
  *  pSize - return file size.
  *   If file was opened for reading, size should indicate the total number
  *    of octets in the file that was opened. If this is a directory, size
  *    would be the number of octets that would be transferred by the read
  *    requests. This is (the number of files contained in the directory * 20)
  *    + (sum of the lengths of the names of all of the files in the directory).
  *   If file was opened for writing size SCL will set size to zero.
  *  pType - return type of file
  *   DNPDEFS_FILE_TYPE_DIRECTORY
  *   DNPDEFS_FILE_TYPE_SIMPLE
  * returns:
  *  DNPDEFS_FILE_CMD_STAT_SUCCESS if successful
  *  DNPDEFS_FILE_CMD_STAT_ASYNC if the SCL should send a NULL response
  *   and later send the response as an event. The target database can call
  *   sdnpo070_fileCmdComplete(); or the SCL will call sdnpdata_openFile
periodically
  *  Other appropriate DNPDEFS_FILE_CMD_STAT values on error or if access is not
allowed.
  *   DNPDEFS_FILE_CMD_STAT_DENIED
  *   DNPDEFS_FILE_CMD_STAT_INV_MODE
  *   DNPDEFS_FILE_CMD_STAT_NOT_FOUND
  *   DNPDEFS_FILE_CMD_STAT_TOO_MANY
  *   DNPDEFS_FILE_CMD_STAT_LOST_COMM
  *   DNPDEFS_FILE_CMD_STAT_MISC
  */
DNPDEFS_FILE_CMD_STAT TMWDEFS_GLOBAL sdnpdata_openFile(
  TMWSESN *pSession,
  TMWTYPES_CHAR *pFilename,
  TMWTYPES_ULONG authKey,
  DNPDEFS_FILE_MODE mode,
  TMWTYPES_USHORT *pMaxBlockSize,
  DNPDEFS_FILE_PERMISSIONS *pPermissions,
  TMWDTIME *pTimeOfCreation,
  TMWTYPES_ULONG *pHandle,
  TMWTYPES_ULONG *pSize,
  DNPDEFS_FILE_TYPE *pType);

/* function: sdnpdata_closeFile
 * purpose: close existing file
 * arguments:
 *  pSession - pointer to session request was received on
 *  handle - DNP file handle returned from previous call to sdnpdata_openFile
 * returns:
 *  DNPDEFS_FILE_CMD_STAT_SUCCESS if successful
 *  DNPDEFS_FILE_CMD_STAT_ASYNC if the SCL should send a NULL response
 *   and later send the response as an event. The target database can call
 *   sdnpo070_fileCmdComplete(); or the SCL will call sdnpdata_closeFile
periodically
 *  Other appropriate DNPDEFS_FILE_CMD_STAT values on error.
 *   DNPDEFS_FILE_CMD_STAT_INV_HANDLE
 *   DNPDEFS_FILE_CMD_STAT_LOST_COMM
 *   DNPDEFS_FILE_CMD_STAT_MISC
 */
DNPDEFS_FILE_CMD_STAT TMWDEFS_GLOBAL sdnpdata_closeFile(
  TMWSESN *pSession,
  TMWTYPES_ULONG handle);

/* function: sdnpdata_readFile
 * purpose: read block of data from file.
 * arguments:
 *  pSession - pointer to session request was received on
 *  handle - DNP file handle returned from previous call to sdnpdata_openFile
 *  pLast - return TMWDEFS_TRUE if this is the last block in this file
 *  pBytesRead - return the actual number of bytes read, up to maxBlockSize
 *   specified in sdnpdata_openFile()
 *  pBuf - buffer into which to store data
```

```
   * returns:
   *  DNPDEFS_FILE_TFER_STAT_SUCCESS if successful
   *  DNPDEFS_FILE_TFER_STAT_ASYNC if the SCL should send a NULL response
   *   and later send the response as an event. The database code can call
   *   sdnpo070_fileCmdComplete(); or the SCL will call sdnpdata_readFile again
   *  Appropriate DNPDEFS_FILE_TFER_STAT values on error.
   *   DNPDEFS_FILE_TFER_STAT_INV_HANDLE
   *   DNPDEFS_FILE_TFER_STAT_LOST_COMM
   *   DNPDEFS_FILE_TFER_STAT_NOT_OPEN
   *   DNPDEFS_FILE_TFER_STAT_HANDLE_EXP
   *   DNPDEFS_FILE_TFER_STAT_BAD_FILE
   *   DNPDEFS_FILE_TFER_STAT_MISC
   */
  DNPDEFS_FILE_TFER_STAT TMWDEFS_GLOBAL sdnpdata_readFile(
    TMWSESN *pSession,
    TMWTYPES_ULONG handle,
    TMWTYPES_BOOL *pLast,
    TMWTYPES_USHORT *pBytesRead,
    TMWTYPES_UCHAR *pBuf);

  /* function: sdnpdata_confirmFileRead
   * purpose: The last file or directory read has been confirmed by the master.
   *  The SCL will handle the master rereading the last block that was sent.
   *  You could use this call to delete data records from a log file once the master
   *  has the data. This will be called once for each sdnpdata_readFile() call that
   *  returns SUCCESS, when the appl confirm for that data is received from the
   *  master.
   *  For directory reads  sdnpdata_readFileInfo() will be called multiple times for
   *  each response to the master depending on block size and amount of data in a
   *  file info. The call to sdnpdata_confirmFileRead will be called once for each
   *  read directory response. This call will acknowledge all
sdnpdata_readFileInfo()
   *  calls up to that time.
   * arguments:
   *  pSession - pointer to session request was received on
   *  handle - DNP file handle returned from previous call to sdnpdata_openFile
   * returns:
   *  void;
   */
  void TMWDEFS_GLOBAL sdnpdata_confirmFileRead(
    TMWSESN *pSession,
    TMWTYPES_ULONG handle);

  /* function: sdnpdata_writeFile
   * purpose: write block of data to file
   * arguments:
   *  pSession - pointer to session request was received on
   *  handle - DNP file handle returned from previous call to sdnpdata_openFile
   *  last - TMWDEFS_TRUE if this is the last block in this file
   *  numBytes - number of bytes to write. This will not exceed maxBlockSize
   *   specified in sdnpdata_openFile()
   *  pBuf - pointer to data buffer to write
   * returns:
   *  DNPDEFS_FILE_TFER_STAT_SUCCESS if successful
   *  DNPDEFS_FILE_TFER_STAT_ASYNC if the SCL should send a NULL response
   *   and later send the response as an event. The target database can call
   *   sdnpo070_fileCmdComplete(); or the SCL will call sdnpdata_writeFile
   *   periodically
   *  Appropriate DNPDEFS_FILE_TFER_STAT values on error.
   *   DNPDEFS_FILE_TFER_STAT_INV_HANDLE
   *   DNPDEFS_FILE_TFER_STAT_LOST_COMM
   *   DNPDEFS_FILE_TFER_STAT_NOT_OPEN
   *   DNPDEFS_FILE_TFER_STAT_HANDLE_EXP
   *   DNPDEFS_FILE_TFER_STAT_OVERRUN
   *   DNPDEFS_FILE_TFER_STAT_BAD_FILE
   *   DNPDEFS_FILE_TFER_STAT_MISC
   */
  DNPDEFS_FILE_TFER_STAT TMWDEFS_GLOBAL sdnpdata_writeFile(
    TMWSESN *pSession,
    TMWTYPES_ULONG handle,
    TMWTYPES_BOOL last,
    TMWTYPES_USHORT numBytes,
    TMWTYPES_UCHAR *pBuf);

#if SDNPCNFG_USER_MANAGED_EVENTS
  /* The following code is necessary only if User Managed Event
   * queues are implemented, instead of letting the SCL manage events.
   * This might be useful if events needed to be saved in nonVolatile
   * memory, or particular queuing algorithms are desired.
   * #define SDNPCNFG_USER_MANAGED_EVENTS TMWDEFS_TRUE when compiling
   * and set userManagedEvents = TMWDEFS_TRUE in SDNPSESN_CONFIG when the
   * session is opened.
   */
  /* Structure used to retrieve user managed events from database */
  typedef struct  {
    TMWTYPES_USHORT          point;
    TMWDEFS_CLASS_MASK       classMask;
    TMWTYPES_UCHAR           group;
    TMWTYPES_UCHAR           defaultVariation;
    TMWTYPES_UCHAR           flags;
    TMWDTIME                 timeStamp;
    union {
      /* This field is used when getting a counter or frozen counter event
       * from database
       */
      TMWTYPES_ULONG           ulValue;

      /* This field is used when getting an analog input event (objectGroup 32),
        * an analog output event (objectGroup 42)
        * or an analog command event (objectGroup 43)
        * from the database
        */
      TMWTYPES_ANALOG_VALUE    analogValue;

      /* This field is used when getting a string or vterm event from database */
      struct {
        TMWTYPES_UCHAR         length;
        TMWTYPES_UCHAR         buf[DNPDEFS_MAX_STRING_LENGTH+1];
      } stringValue;

#if SDNPDATA_SUPPORT_OBJ115
      /* This field is used when getting a an extended string event from database */
      struct {
        TMWTYPES_USHORT        length;
        TMWTYPES_UCHAR         buf[DNPCNFG_MAX_EXT_STRING_LENGTH+1];
      } extendedStringValue;
#endif

      /* This field is used when getting a Data Set event from database */
      struct {
        TMWTYPES_UCHAR         numberElems;
        DNPDATA_DATASET_VALUE *pDataSet;
      } datasetValue;

      /* This field is used when getting an Authentication Error Event
        * from the database. This event would be on an association other
        * than the association where the error occurred.
        */
      struct {
        TMWTYPES_USHORT        assocId;
        TMWTYPES_ULONG         sequenceNumber;
        TMWTYPES_UCHAR         errorCode;
        TMWTYPES_USHORT        errorTextLength;
        TMWTYPES_UCHAR         errorText[DNPAUTH_MAX_ERROR_TEXT_LENGTH + 1];
      } authError;

      /* This field is used when getting an Authentication Security Statistic
        * from the database
        */
      struct {
        TMWTYPES_USHORT        assocId;
        TMWTYPES_ULONG         ulValue;
      } authSecStat;

    } value;
  } SDNPDATA_GET_EVENT;

  /* function: sdnpdata_umEventAdd
   * purpose: add event to user managed event queue in database
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  group - object group indicating event type being added
   *   DNPDEFS_OBJ_2_BIN_CHNG_EVENTS  or
   *   DNPDEFS_OBJ_4_DBL_CHNG_EVENTS  or
   *   DNPDEFS_OBJ_11_BIN_OUT_EVENTS  or
   *   DNPDEFS_OBJ_13_BIN_CMD_EVENTS  or
   *   DNPDEFS_OBJ_22_CNTR_EVENTS     or
   *   DNPDEFS_OBJ_23_FCTR_EVENTS     or
   *   DNPDEFS_OBJ_32_ANA_CHNG_EVENTS or
   *   DNPDEFS_OBJ_42_ANA_OUT_EVENTS  or
   *   DNPDEFS_OBJ_43_ANA_CMD_EVENTS  or
   *   DNPDEFS_OBJ_111_STRING_EVENTS  or
```

```
 *   DNPDEFS_OBJ_113_VTERM_EVENTS
 *  point - point number of point that changed
 *  classMask - class that this change event should be reported in.
 *   TMWDEFS_CLASS_MASK_ONE   or
 *   TMWDEFS_CLASS_MASK_TWO   or
 *   TMWDEFS_CLASS_MASK_THREE
 *  defaultVariation - what variation to use for this event if a read of
 *   variation 0 is received.
 *   This is used when per point variation is compiled in.
 *   (SDNPDATA_SUPPORT_EVENT_MODE_POINT)
 *  flags - flags value for this point
 *  pValue - pointer to union containing value for this point
 *   group 2, 4 will not have a value other than flags
 *   group 22, 23 will have value contained in ulValue
 *   group 30 will have value pointed to by analogPtr
 *   group 111 and 113 will have length and string pointed to by stringPtr
 *  pTimeStamp - pointer to time stamp indicating when this point changed
 * returns:
 *  TMWDEFS_TRUE if event was successfully added to queue,
 *  TMWDEFS_FALSE if this event could not be added, OR another event had to be
 *   deleted in order to add this event. The SCL will set the OVERFLOW IIN bit
 *   indicating 1 or more events were lost.
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_umEventAdd(
  void *pHandle,
  TMWTYPES_UCHAR group,
  TMWTYPES_USHORT point,
  TMWDEFS_CLASS_MASK classMask,
  TMWTYPES_UCHAR defaultVariation,
  TMWTYPES_UCHAR flags,
  SDNPDATA_ADD_EVENT_VALUE *pValue,
  TMWDTIME *pTimeStamp);

/* function: sdnpdata_umEventNotSentCount
 * purpose: retrieve count of events of this type that have not
 *  been marked as sent, in user managed event queue in database
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  group - object group indicating event type
 *   DNPDEFS_OBJ_2_BIN_CHNG_EVENTS  or
 *   DNPDEFS_OBJ_4_DBL_CHNG_EVENTS  or
 *   DNPDEFS_OBJ_11_BIN_OUT_EVENTS  or
 *   DNPDEFS_OBJ_13_BIN_CMD_EVENTS  or
 *   DNPDEFS_OBJ_22_CNTR_EVENTS     or
 *   DNPDEFS_OBJ_23_FCTR_EVENTS     or
 *   DNPDEFS_OBJ_32_ANA_CHNG_EVENTS or
 *   DNPDEFS_OBJ_42_ANA_OUT_EVENTS  or
 *   DNPDEFS_OBJ_43_ANA_CMD_EVENTS  or
 *   DNPDEFS_OBJ_111_STRING_EVENTS  or
 *   DNPDEFS_OBJ_113_VTERM_EVENTS
 *  classMask - which class to check for events.
 *   TMWDEFS_CLASS_MASK_ONE    or
 *   TMWDEFS_CLASS_MASK_TWO    or
 *   TMWDEFS_CLASS_MASK_THREE  or
 *   TMWDEFS_CLASS_MASK_ALL
 * returns:
 *  number of events for this group and class in queue THAT HAVE
 *   NOT BEEN MARKED AS SENT
 */
TMWTYPES_USHORT TMWDEFS_GLOBAL sdnpdata_umEventNotSentCount(
  void *pHandle,
  TMWTYPES_UCHAR group,
  TMWDEFS_CLASS_MASK classMask);

/* function: sdnpdata_umEventGet
 * purpose: retrieve event from user managed event queue in database
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  group - object group indicating event type
 *   DNPDEFS_OBJ_2_BIN_CHNG_EVENTS  or
 *   DNPDEFS_OBJ_4_DBL_CHNG_EVENTS  or
 *   DNPDEFS_OBJ_11_BIN_OUT_EVENTS  or
 *   DNPDEFS_OBJ_13_BIN_CMD_EVENTS  or
 *   DNPDEFS_OBJ_22_CNTR_EVENTS     or
 *   DNPDEFS_OBJ_23_FCTR_EVENTS     or
 *   DNPDEFS_OBJ_32_ANA_CHNG_EVENTS or
 *   DNPDEFS_OBJ_42_ANA_OUT_EVENTS  or
 *   DNPDEFS_OBJ_43_ANA_CMD_EVENTS  or
 *   DNPDEFS_OBJ_111_STRING_EVENTS  or
 *   DNPDEFS_OBJ_113_VTERM_EVENTS
 *  NOTE: To support a read of both Obj2 and Obj4 events including both of those
 *   groups intermixed in order of occurrence, this function should return the
 *   next Obj2 or Obj4 event that occurred when group argument is either 2 or 4.
 *  classMask - which class to get event for.
 *   TMWDEFS_CLASS_MASK_ONE    or
 *   TMWDEFS_CLASS_MASK_TWO    or
 *   TMWDEFS_CLASS_MASK_THREE  or
 *   TMWDEFS_CLASS_MASK_ALL
 *  firstEvent - if TMWDEFS_TRUE, return data for first event in queue that
 *   matches group and classMask
 *  pEvent - pointer to structure to hold fields describing this event.
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_umEventGet(
  void *pHandle,
  TMWTYPES_UCHAR group,
  TMWDEFS_CLASS_MASK classMask,
  TMWTYPES_BOOL firstEvent,
  SDNPDATA_GET_EVENT *pEvent);

/* function: sdnpdata_umEventSent
 * purpose: Mark the last event returned by sdnpdata_umEventGet() as
 *  having been sent, but not yet acked by application confirm.
 *  Event cannot be removed from queue yet.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  group - object group indicating event type
 *   DNPDEFS_OBJ_2_BIN_CHNG_EVENTS  or
 *   DNPDEFS_OBJ_4_DBL_CHNG_EVENTS  or
 *   DNPDEFS_OBJ_22_CNTR_EVENTS     or
 *   DNPDEFS_OBJ_23_FCTR_EVENTS     or
 *   DNPDEFS_OBJ_32_ANA_CHNG_EVENTS or
 *   DNPDEFS_OBJ_42_ANA_OUT_EVENTS  or
 *   DNPDEFS_OBJ_43_ANA_CMD_EVENTS  or
 *   DNPDEFS_OBJ_111_STRING_EVENTS  or
 *   DNPDEFS_OBJ_113_VTERM_EVENTS
 *  point - point number in event
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL sdnpdata_umEventSent(
  void *pHandle,
  TMWTYPES_UCHAR group,
  TMWTYPES_USHORT point);

/* function: sdnpdata_umEventNotSent
 * purpose: Mark all events in this user managed queue as not yet sent.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  group - object group indicating event type
 *   DNPDEFS_OBJ_2_BIN_CHNG_EVENTS  or
 *   DNPDEFS_OBJ_4_DBL_CHNG_EVENTS  or
 *   DNPDEFS_OBJ_22_CNTR_EVENTS     or
 *   DNPDEFS_OBJ_23_FCTR_EVENTS     or
 *   DNPDEFS_OBJ_32_ANA_CHNG_EVENTS or
 *   DNPDEFS_OBJ_42_ANA_OUT_EVENTS  or
 *   DNPDEFS_OBJ_43_ANA_CMD_EVENTS  or
 *   DNPDEFS_OBJ_111_STRING_EVENTS  or
 *   DNPDEFS_OBJ_113_VTERM_EVENTS
 * returns:
 *  TMWDEFS_TRUE if specified queue is full
 *  TMWDEFS_FALSE if queue is not full
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_umEventNotSent(
  void *pHandle,
  TMWTYPES_UCHAR group);

/* function: sdnpdata_umEventRemove
 * purpose: remove all events, that have been marked
 *  as sent, from this user managed event queue in database
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  group - object group indicating event type
 *   DNPDEFS_OBJ_2_BIN_CHNG_EVENTS  or
 *   DNPDEFS_OBJ_4_DBL_CHNG_EVENTS  or
 *   DNPDEFS_OBJ_22_CNTR_EVENTS     or
 *   DNPDEFS_OBJ_23_FCTR_EVENTS     or
 *   DNPDEFS_OBJ_32_ANA_CHNG_EVENTS or
 *   DNPDEFS_OBJ_42_ANA_OUT_EVENTS  or
 *   DNPDEFS_OBJ_43_ANA_CMD_EVENTS  or
 *   DNPDEFS_OBJ_111_STRING_EVENTS  or
```

```
  *   DNPDEFS_OBJ_113_VTERM_EVENTS
  * returns:
  *  TMWDEFS_TRUE if specified queue is full
  *  TMWDEFS_FALSE if queue is not full
  */
 TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_umEventRemove(
   void *pHandle,
   TMWTYPES_UCHAR group);
#endif

#if SDNPDATA_SUPPORT_OBJ0
 /* Device Attributes */

 /* Object Group 0. Variations 0-255 are used to specify particular attributes
  * Index 0 is used for standard attributes defined by DNP User Group in
  *  Tech Bulletin TB2003-001 and TB2005-002.
  *  Defines DNPDEFS_OBJ0xxx in dnpdefs.h can be used to identify the standard
  *  attributes for Index 0.
  *
  * Indexes other than 0 are used to specify user-specific attribute sets.
  */

 /* function: sdnpdata_deviceAttrQuantity
  * purpose:  Return quantity of device attribute sets present in this device,
  *  including 1 for the standard DNP set and 1 for each User-specific set.
  * arguments:
  *  pHandle - handle to database returned from sdnpdata_init
  * returns:
  *  Number of device attribute sets present (see note for
  *  sdnpdata_xxxQuantity at top of this file)
  */
 TMWTYPES_USHORT TMWDEFS_GLOBAL sdnpdata_deviceAttrQuantity(
   void *pHandle);

 /* function: sdnpdata_deviceAttrGetPoint
  * purpose: Return a handle to a specific device attribute point or set
  *  of attributes
  * arguments:
  *  pHandle - handle to database returned from sdnpdata_init
  *  pointNum - point number or device attribute set to return a handle to.
  *   0 for standard attributes, other values for user-specific attribute set
  * returns:
  *  Handle for specified point (device attribute set) or TMWDEFS_NULL if this set
  *   is currently disabled.
  */
 void * TMWDEFS_CALLBACK sdnpdata_deviceAttrGetPoint(
   void *pHandle,
   TMWTYPES_USHORT pointNum);

 /* function: sdnpdata_deviceAttrNext
  * purpose:  Return next device attribute variation that is supported and the
  *  value of its property field. The SCL will use this when reading all device
  *  attributes.
  * arguments:
  *  pPoint - handle returned by sdnpdata_deviceAttrGetPoint. It points to a
  *   set of attributes.
  *  variation - variation of device attribute.
  *   NOTE: Variation is used as an "index" when referring to device
  *   attributes.
  *   if variation == 0, return the first variation supported in this set of
  *   device attributes.
  *   if variation != 0, return next variation after specified variation,
  *   This function supports gaps in the device attribute variations supported.
  *  pNextVariation - pointer to variation variable to be filled in
  *  pProperty - pointer to property variable to be filled in
  *   0x00 indicates attribute is NOT writable by master
  *   0x01 indicates attribute is writable by master
  * returns:
  *  TMWDEFS_TRUE if there is a next variation.
  */
 TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_deviceAttrNext(
   void *pPoint,
   TMWTYPES_UCHAR variation,
   TMWTYPES_UCHAR *pNextVariation,
   TMWTYPES_UCHAR *pProperty);

 /* function: sdnpdata_deviceAttrGetVar
  * purpose: Return a handle to a specific device attribute. This handle is
  *  used by the routines below to read and write values to this attribute.
  * arguments:
  *  pPoint - handle to device attribute set returned by
```

```
   *   sdnpdata_deviceAttrGetPoint
   *  point - point number of device attribute to get handle for,
   *   0 for standard attributes, other values for user-specific attribute set
   *  variation - variation of device attribute within that set.
   * returns:
   *  Pointer to specified device attribute or TMWDEFS_NULL if this attribute
   *   does not exist
   */
  void * TMWDEFS_GLOBAL sdnpdata_deviceAttrGetVar(
    void *pPoint,
    TMWTYPES_UCHAR variation);

  /* function: sdnpdata_deviceAttrRead
   * purpose: Read specified device attribute
   * arguments:
   *  pAttribute - handle to device attribute returned from
   *   sdnpdata_deviceAttrGetVar().
   *  pData - pointer to DNPDATA_ATTRIBUTE_VALUE structure to be filled in.
   *   pData->type indicates type of data in the union pData->value
   *     DNPDEFS_ATTRIBUTE_TYPE_VSTR,
   *     DNPDEFS_ATTRIBUTE_TYPE_UINT,
   *     DNPDEFS_ATTRIBUTE_TYPE_INT,
   *     DNPDEFS_ATTRIBUTE_TYPE_FLT,
   *     DNPDEFS_ATTRIBUTE_TYPE_OSTR,
   *     DNPDEFS_ATTRIBUTE_TYPE_BSTR,
   *     DNPDEFS_ATTRIBUTE_TYPE_DNP3TIME
   *   NOTE: Types are specified for the device attributes defined in
   *    Tech Bulletin TB2003-001 and TB2005-002. For quantities UINT is used.
   *    To indicate support INT (0 or 1) is used. Most of the others are VSTR.
   * returns:
   *  TMWDEFS_TRUE if read was successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_deviceAttrRead(
    void *pAttribute,
    DNPDATA_ATTRIBUTE_VALUE *pData);

  /* function: sdnpdata_deviceAttrWrite
   * purpose: Write values to specified device attribute
   *  NOTE: this function should fail if property for this
   *  device attribute indicates it is not writable.
   * arguments:
   *  pAttribute - handle to attribute returned from sdnpdata_deviceAttrGetVar().
   * returns:
   *  TMWDEFS_TRUE if write was successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_deviceAttrWrite(
    void *pAttribute,
    DNPDATA_ATTRIBUTE_VALUE *pData);
#endif

  /* DATA SETS  */

  /* function: sdnpdata_datasetGetDescription
   * purpose: Get description of this point
   *  NOTE: this is used only by sdnpxml/sdnpxml2.c if SDNPDATA_SUPPORT_XMLxx is
TRUE
   *  code to generate the configuration file or Device Profile for this device.
   * arguments:
   *  pPoint - handle to data point returned from 'getPoint' function.
   * returns:
   *  pointer to string describing point
   *  TMWDEFS_NULL if failure
   */
TMWTYPES_CHAR * TMWDEFS_GLOBAL sdnpdata_datasetGetDescription(
  void *pPoint);

#if SDNPDATA_SUPPORT_OBJ85
  /* Data Set Prototypes */

  /* function: sdnpdata_datasetProtoQuantity
   * purpose: Return the number of Data Set prototypes in the
   *  specified database.
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   * returns:
   *  The number of Data Set prototypes
   *  NOTE: these must be sequential starting from prototype id
   *   or point index 0.
   */
```

```
TMWTYPES_USHORT TMWDEFS_GLOBAL sdnpdata_datasetProtoQuantity(
  void *pHandle);

/* function: sdnpdata_datasetProtoGetID
 * purpose: Return a prototype ID or point index for a Data Set prototype
 *  with this UUID, if one exists in the database.
 *  This point number is used by sdnpdata_datasetProtoGetPoint to prepare
 *  a prototype for reading.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  pUUID - pointer to 16 byte UUID string to be looked up
 *  pPointNum - point number or prototype id to be filled in.
 * returns:
 *  TMWDEFS_TRUE if prototype was found
 *  TMWDEFS_FALSE otherwise
 */
TMWTYPES_BOOL TMWDEFS_CALLBACK sdnpdata_datasetProtoGetID(
  void *pHandle,
  TMWTYPES_UCHAR *pUUID,
  TMWTYPES_USHORT *pPointNum);

/* function: sdnpdata_datasetProtoGetPoint
 * purpose: Return a handle to a specific Data Set prototype. This handle is
 *  used by the routine below to read values for this prototype.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  pointNum - point number or prototype id to return a handle to.
 * returns:
 *  Handle for a specified prototype or TMWDEFS_NULL if this prototype
 *   is currently disabled or not found.
 */
void * TMWDEFS_CALLBACK sdnpdata_datasetProtoGetPoint(
  void *pHandle,
  TMWTYPES_USHORT pointNum);

/* function: sdnpdata_datasetProtoRead
 * purpose: Get a pointer to array of Data Set prototype element
 *   structures to support read of object group 85 variation 1
 *  pPoint - handle to Data Set prototype "data point" returned by
 *   sdnpdata_datasetProtoGetPoint function.
 *  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 sdnpdata_datasetProtoRelease() is called
 *   NOTE: The array of elements should not contain the mandatory Prototype ID
 *   and UUID. Prototype ID was specified when xxxGetPoint was called 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.
 */
DNPDATA_DATASET_DESCR_ELEM * TMWDEFS_GLOBAL sdnpdata_datasetProtoRead(
  void *pPoint,
  TMWTYPES_UCHAR *pNumberElems,
  TMWTYPES_UCHAR *pUUID);

/* function: sdnpdata_datasetProtoRelease
 * purpose: Release the pointer that was returned by in sdnpdata_datasetProtoRead
 *  The database is free to deallocate or reuse the memory that was pointed to.
The
 *  SCL will not attempt to reference that pointer anymore.
 *  pPoint - handle to Data Set descriptor "data point" returned by
 *   sdnpdata_datasetDescrGetPoint function.
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL sdnpdata_datasetProtoRelease(
  void *pPoint);

/* function: sdnpdata_datasetProtoCreatePoint
 * purpose: Create a Data Set prototype if it does not exist and return a handle
 *  to it. This handle is used by the routines below to write values for this
 *  prototype.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
```

```
  *  pointNum - point number or prototype id to return a handle to.
  *   This should start after the last index for prototypes already found on the
  *   outstation(slave).
  *  pUUID - pointer to 16 byte string containing UUID to be stored
  * returns:
  *  Handle for a specified prototype or TMWDEFS_NULL if this prototype does
  *   not exist or cannot be created for writing to.
  */
 void * TMWDEFS_GLOBAL sdnpdata_datasetProtoCreatePoint(
   void *pHandle,
   TMWTYPES_USHORT pointNum,
   TMWTYPES_UCHAR *pUUID);

 /* function: sdnpdata_datasetProtoWrite
  * purpose: Write a Data Set prototype element to support write of object
  *   group 85 variation 1
  *  pPoint - handle to Data Set prototype "data point" returned by
  *   sdnpdata_datasetProtoCreatePoint function.
  *  index - index of element in Data Set prototype to be written starting
  *   with index 0. The mandatory Data Set prototype ID and UUID will not be
  *   written using this function. They will be written to the database using
  *   the sdnpdata_datasetProtoCreatePoint() function.
  *  pElem - pointer to structure containing prototype element to be written
  *   NOTE: if pElem->ancillaryValue->type == DNPDATA_VALUE_STRPTR, this is just
  *    a pointer, the string itself must be copied somewhere.
  * returns:
  *  TMWDEFS_TRUE if write is successful
  *  TMWDEFS_FALSE if write failed
  */
 TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_datasetProtoWrite(
   void *pPoint,
   TMWTYPES_UCHAR index,
   DNPDATA_DATASET_DESCR_ELEM *pElem);
#endif

#if SDNPDATA_SUPPORT_OBJ86
 /* Data Set Descriptors */

 /* function: sdnpdata_datasetDescrQuantity
  * purpose: Return the number of Data Set descriptors in the
  *  specified database.
  * arguments:
  *  pHandle - handle to database returned from sdnpdata_init
  * returns:
  *  The number of Data Set descriptors (see note for
  *  sdnpdata_xxxQuantity at top of this file)
  */
 TMWTYPES_USHORT TMWDEFS_GLOBAL sdnpdata_datasetDescrQuantity(
   void *pHandle);

 /* function: sdnpdata_datasetDescrGetPoint
  * purpose: Return a handle to a specific data descriptor. This handle is
  *  used by the routines below to read values for this descriptor.
  * arguments:
  *  pHandle - handle to database returned from sdnpdata_init
  *  pointNum - point number or Data Set id to return a handle to.
  * returns:
  *  Handle for a specified data descriptor or TMWDEFS_NULL if this
  *   is currently disabled or not found.
  */
 void * TMWDEFS_CALLBACK sdnpdata_datasetDescrGetPoint(
   void *pHandle,
   TMWTYPES_USHORT pointNum);

 /* function: sdnpdata_datasetDescrReadCont
  * purpose: Get a pointer to array of Data Set descriptor contents
  *   structures to support read of object group 86 variation 1
  *  pPoint - handle to Data Set descriptor "data point" returned by
  *   sdnpdata_datasetDescrGetPoint function.
  *  pNumberElems - number of elements returned in specified Data Set 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 sdnpdata_datasetDescrRel() is called
  *  NOTE: Data Set id is a mandatory element and was specified in
  *  sdnpdata_datasetDescrGetPoint(). 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.
  */
 DNPDATA_DATASET_DESCR_ELEM * TMWDEFS_GLOBAL sdnpdata_datasetDescrReadCont(
   void *pPoint,
   TMWTYPES_UCHAR *pNumberElems);

 /* function: sdnpdata_datasetDescrReadChars
  * purpose: Read characteristics of Data Set descriptor specified
  *   to support read of object group 86 variation 2.
  *  pPoint - handle to Data Set descriptor "data point" returned by
  *   sdnpdata_datasetDescrGetPoint function.
  *  pValue - pointer to characteristics variable to be filled in by this
  *   function. The following bit definitions may be OR'ed together.
  *    DNPDEFS_DATASET_CHAR_RD  set if data set is readable
  *    DNPDEFS_DATASET_CHAR_WR  set if data set is writable
  *    DNPDEFS_DATASET_CHAR_ST  set if outstation maintains a static data set
  *      Note: a read of object 87 would fail if this bit is not set.
  *    DNPDEFS_DATASET_CHAR_EV  set if outstation generates a data set event
  *    DNPDEFS_DATASET_CHAR_DF  set if defined by master
  * returns:
  *  void
  */
 void TMWDEFS_GLOBAL sdnpdata_datasetDescrReadChars(
   void *pPoint,
   TMWTYPES_UCHAR *pValue);

 /* function: sdnpdata_datasetDescrReadIndex
  * purpose: Get a pointer to array of Data Set descriptor index
  *   structures to support read of object group 86 variation 3
  *  pPoint - handle to Data Set descriptor "data point" returned by
  *   sdnpdata_datasetDescrGetPoint function.
  *  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 sdnpdata_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).
  */
 DNPDATA_DATASET_DESCR_INDEX * TMWDEFS_GLOBAL sdnpdata_datasetDescrReadIndex(
   void *pPoint,
   TMWTYPES_UCHAR *pNumberElems);

 /* function: sdnpdata_datasetDescrRelease
  * purpose: Release the pointer that was returned by
  *  sdnpdata_datasetDescrReadCont or sdnpdata_datasetDescrReadIndex.
  *  The database is free to deallocate or reuse the memory that was pointed to.
  *  The SCL will not attempt to reference that pointer anymore.
  *  pPoint - handle to Data Set descriptor "data point" returned by
  *   sdnpdata_datasetDescrGetPoint function.
  * returns:
  *  void
  */
 void TMWDEFS_GLOBAL sdnpdata_datasetDescrRelease(
   void *pPoint);

 /* function: sdnpdata_datasetDescrCreatePoint
  * purpose: Create a Data Set descriptor if it does not exist and return a
  *  handle to it. This handle is used by the routines below to write values
  *  for this descriptor.
  * arguments:
  *  pHandle - handle to database returned from sdnpdata_init
  *  pointNum - point number or Data Set id to return a handle to.
  *   This should start after the last index for descriptors already found on
  *   the outstation(slave).
  * returns:
  *  Handle for a specified descriptor or TMWDEFS_NULL if this descriptor does
  *   not exist or cannot be created for writing to.
  */
 void * TMWDEFS_GLOBAL sdnpdata_datasetDescrCreatePoint(
   void *pHandle,
   TMWTYPES_USHORT pointNum) ;

 /* function: sdnpdata_datasetDescrWriteCont
  * purpose: Write a Data Set descriptor contents structure to support
  *   write of object group 86 variation 1
  *  pPoint - handle to Data Set descriptor "data point" returned by
  *   sdnpdata_datasetDescrCreatePoint function.
  *  index - index of element in Data Set descriptor to be written starting
  *   with index 0. The mandatory Data Set descriptor ID will not be written
  *   using this function. It will be written to the database using
  *   the sdnpdata_datasetDescrCreatePoint() function.
```

```
   *  pElem - pointer to structure containing data to be written
   *   NOTE: if pElem->ancillaryValue->type == DNPDATA_VALUE_STRPTR, this is just
   *    a pointer, the string itself must be copied somewhere.
   * returns:
   *  TMWDEFS_TRUE if write is successful
   *  TMWDEFS_FALSE if write failed
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_datasetDescrWriteCont(
    void *pPoint,
    TMWTYPES_UCHAR index,
    DNPDATA_DATASET_DESCR_ELEM *pElem);

  /* function: sdnpdata_datasetDescrWriteIndex
   * purpose: Write a Data Set descriptor index structure to support
   *   write of object group 86 variation 3
   *  pPoint - handle to Data Set descriptor "data point" returned by
   *   sdnpdata_datasetDescrGetPoint function.
   *  index - index of element in Data Set descriptor to be written 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.
   *  pElem - pointer to structure containing data to be written
   * returns:
   *  TMWDEFS_TRUE if write is successful
   *  TMWDEFS_FALSE if write failed
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_datasetDescrWriteIndex(
    void *pPoint,
    TMWTYPES_UCHAR index,
    DNPDATA_DATASET_DESCR_INDEX *pElem);

  /* function: sdnpdata_datasetAssignClass
   * purpose: Assign the class in which events for this Data Set descriptor
   *  will belong.
   * arguments:
   *  pPoint - handle to Data Set returned from sdnpdata_datasetDescrGetPoint
   *   function.
   *  classMask - new class in which to generate events from this point
   * returns:
   *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_datasetDescrAssignClass(
    void *pPoint,
    TMWDEFS_CLASS_MASK classMask);

  /* function: sdnpdata_datasetDescrEventClass
   * purpose: Return the class in which events for this Data Set descriptor
   *  belong.
   * arguments:
   *  pPoint - handle to Data Set returned from sdnpdata_datasetDescrGetPoint
   * returns:
   *  Class in which these events (Object Group 88) will be returned
   */
  TMWDEFS_CLASS_MASK TMWDEFS_GLOBAL sdnpdata_datasetDescrEventClass(
    void *pPoint);
#endif

#if SDNPDATA_SUPPORT_OBJ87
  /* Data Set Contents */

  /* function: sdnpdata_datasetQuantity
   * purpose: Return the number of datasets in the specified database.
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   * returns:
   *  The number of datasets (see note for
   *  sdnpdata_xxxQuantity at top of this file)
   */
  TMWTYPES_USHORT TMWDEFS_GLOBAL sdnpdata_datasetQuantity(
    void *pHandle);

  /* function: sdnpdata_datasetGetPoint
   * purpose: Return a handle to a specific Data Set point. This handle is
   *  used by the routines below to read values from this Data Set.
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  pointNum - point number or Data Set ID to return a handle to.
   * returns:
   *  Handle for a specified Data Set or TMWDEFS_NULL if this Data Set
   *   is currently disabled or not found.
   */
  void * TMWDEFS_CALLBACK sdnpdata_datasetGetPoint(
    void *pHandle,
    TMWTYPES_USHORT pointNum);

  /* function: sdnpdata_datasetRead
   * purpose: Get a pointer to array of Data Set structures to support read
   *   of object group 87 variation 1
   *  pPoint - handle to Data Set "data point" returned by
   *   sdnpdata_datasetGetPoint function.
   *  pNumberElems - number of elements returned in Data Set array
   *  pTimeStamp - location in memory to copy timeStamp for this Data Set to.
   * returns:
   *  Pointer to array of Data Set value structures SDNPDATA_DATASET_VALUE.
   *   This points to memory maintained by the database. This pointer will need
   *   to be valid until sdnpdata_datasetRelease() is called.
   *   NOTE: The array of elements should not contain the mandatory Data Set ID
   *   and timeStamp. Data Set ID was specified when xxxGetPoint was called and
   *   timeStamp should be returned by pTimeStamp parameter.
   */
  DNPDATA_DATASET_VALUE * TMWDEFS_GLOBAL sdnpdata_datasetRead(
    void *pPoint,
    TMWTYPES_UCHAR *pNumberElems,
    TMWDTIME *pTimeStamp);

  /* function: sdnpdata_datasetRelease
   * purpose: Release the pointer that was returned by sdnpdata_datasetRead().
   *  The database is free to deallocate or reuse the memory that was pointed to.
The
   *  SCL will not attempt to reference that pointer anymore.
   *  pPoint - handle to Data Set descriptor "data point" returned by
   *   sdnpdata_datasetDescrGetPoint function.
   * returns:
   *  void
   */
  void TMWDEFS_GLOBAL sdnpdata_datasetRelease(
    void *pPoint);

  /* function: sdnpdata_datasetCreatePoint
   * purpose: Create a Data Set if it does not exist (if desired) and return a
   *  handle to it. This handle is used by the routines below to write or control
   *  values for this Data Set.
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  pointNum - point number or Data Set id to return a handle to.
   *   This should start after the last index for datasets already found on the
   *   outstation(slave).
   *  mode - indicates whether this call is caused by write, select or operate
   *   SDNPDATA_DATASET_MODE_WRITE,
   *   SDNPDATA_DATASET_MODE_SELECT,
   *   SDNPDATA_DATASET_MODE_OPERATE
   *  pTimeStamp - pointer timeStamp to be stored or TMWDEFS_NULL when cancel select
is being performed.
   * returns:
   *  Handle for a specified Data Set or TMWDEFS_NULL if this Data Set does
   *   not exist or cannot be created for writing or controlling.
   */
  void * TMWDEFS_GLOBAL sdnpdata_datasetCreatePoint(
    void *pHandle,
    TMWTYPES_USHORT pointNum,
    SDNPDATA_DATASET_MODE mode,
    TMWDTIME *pTimeStamp);

   /* function: sdnpdata_datasetWrite
    * purpose: Write a Data Set value structure to support
    *   write of object group 87 variation 1
    *  pPoint - handle to Data Set "data point" returned by
    *   sdnpdata_datasetCreatePoint function.
    *  index - index of element in Data Set to be written starting
    *   with index 0. The mandatory Data Set ID and timeStamp will not be
    *   written using this function. They will be written to the database
    *   using the sdnpdata_datasetCreatePoint() function.
    *  pElem - pointer to structure containing value to be written
    *   NOTE: if pElem->type == DNPDATA_VALUE_STRPTR, this is just
    *    a pointer, the string itself must be copied somewhere.
    * returns:
    *  TMWDEFS_TRUE if write is successful
    *  TMWDEFS_FALSE if write failed
    */
   TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_datasetWrite(
    void *pPoint,
    TMWTYPES_UCHAR index,
    DNPDATA_DATASET_VALUE *pElem);
```

```
 /* function: sdnpdata_datasetSelect
  * purpose: Determine whether operate would succeed for a Data Set to support
  *   select of object group 87 variation 1
  *  pPoint - handle to Data Set "data point" returned by
  *   sdnpdata_datasetCreatePoint function.
  *  pCtrlValues - pointer to array of data set control value structures
  *   indicating element index in this data set and value for that element.
  *  numberCtrlValues - number of control values passed in pCtrlValues array
  * returns:
  *  status of select operation. Valid values are:
  *    DNPDEFS_DATASET_ST_SUCCESS - the operate command would succeed
  *    DNPDEFS_DATASET_ST_FORMAT_ERROR - the request was not accepted due
  *      to formatting errors in the request
  *    DNPDEFS_DATASET_ST_NOT_SUPPORTED - the request was not accepted because
  *      Control operations are not supported for this point
  *    DNPDEFS_DATASET_ST_ALREADY_ACTIVE - the request was not accepted because
  *      the control queue is full or the point is already active
  *    DNPDEFS_DATASET_ST_HARDWARE_ERROR - the request was not accepted due to
  *      control hardware problems
  */
 DNPDEFS_DATASET_ST TMWDEFS_GLOBAL sdnpdata_datasetSelect(
   void *pPoint,
   DNPDATA_DATASET_CTRL_VALUE *pCtrlValues,
   TMWTYPES_UCHAR numberCtrlValues);

/* function: sdnpdata_datasetOperate
 * purpose: Issue operate on a Data Set to support operate function code
 *   on object group 87 variation 1
 *  pPoint - handle to Data Set "data point" returned by
 *   sdnpdata_datasetCreatePoint function.
 *  pCtrlValues - pointer to array of data set control value structures
 *   indicating element index in this data set and value for that element.
 *  numberCtrlValues - number of control values passed in pCtrlValues array
 * returns:
 *  status of DATASET operation. Valid values are:
 *    DNPDEFS_DATASET_ST_SUCCESS - the command was performed successfully
 *    DNPDEFS_DATASET_ST_FORMAT_ERROR - the request was not accepted due
 *      to formatting errors in the request
 *    DNPDEFS_DATASET_ST_NOT_SUPPORTED - the request was not accepted because
 *      Control operations are not supported for this point
 *    DNPDEFS_DATASET_ST_ALREADY_ACTIVE - the request was not accepted because
 *      the control queue is full or the point is already active
 *    DNPDEFS_DATASET_ST_HARDWARE_ERROR - the request was not accepted due to
 *      control hardware problems
 */
DNPDEFS_DATASET_ST TMWDEFS_GLOBAL sdnpdata_datasetOperate(
  void *pPoint,
  DNPDATA_DATASET_CTRL_VALUE *pCtrlValues,
  TMWTYPES_UCHAR numberCtrlValues);

/* function: sdnpdata_datasetCancelSelect
 * purpose: cancel outstanding Select on the specified point
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL sdnpdata_datasetCancelSelect(
  void *pPoint);
#endif

/* function: sdnpdata_activateConfig
 * purpose: Process activate configuration (FC31) command received from master
 *  and provide result status and time delay to be sent back in Object Group 91
 *  variation 1 response.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  pointNum - point number received in request from master if request contained
 *    object 110 string. Its meaning is a local matter. A request containing
 *    object 70 variation 8 does not contain a point number and pointNum will
 *    be set to zero.
 *  lastString - TMWDEFS_TRUE  - this is the last string in the activate config
 *    request.   TMWDEFS_FALSE - otherwise.
 *  pString - pointer to string received from master
 *  stringLength - length of string
 *  pDelay - pointer to ULONG that should be filled in to indicate how long
 *   outstation expects to be busy activating the configuration.
 *   Only the last delay returned by this function
 *  (when lastString == TMWDEFS_TRUE) will be sent back to the master.
 *  pErrorText - pointer to optional error text that can be filled in
 *  pErrorLength - max length of error text allowed. Should be filled in
 *   to indicate the length of pErrorText returned to SCL
 * returns:
 *  status of activate config command
 *   Possible status values:
 *    DNPDEFS_ACTCONFIG_SUCCESS    - success
 *    DNPDEFS_ACTCONFIG_REQERROR   - error in the request object
 *    DNPDEFS_ACTCONFIG_DATAERROR  - error in the configuration data
 *    DNPDEFS_ACTCONFIG_ERROR      - any other error
 *    DNPDEFS_ACTCONFIG_NOTCHECKED - not checked
 */
 TMWTYPES_UCHAR TMWDEFS_GLOBAL sdnpdata_activateConfig(
   void *pHandle,
   TMWTYPES_USHORT pointNum,
   TMWTYPES_BOOL   lastString,
   TMWTYPES_UCHAR *pString,
   TMWTYPES_USHORT stringLength,
   TMWTYPES_ULONG *pDelay,
   TMWTYPES_UCHAR *pErrorText,
   TMWTYPES_UCHAR *pErrorLength);

/* function: sdnpdata_authAssignClass
 * purpose: Set class to which secure authentication g120v7 error events belong
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  classMask - Class in which these events will be returned
 * returns:
 *  TMWDEFS_TRUE if successful
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_authAssignClass(
  void *pHandle,
  TMWDEFS_CLASS_MASK classMask);

/* function: sdnpdata_authErrorEventClass
 * purpose: Return class in which secure authentication g120v7 error events belong
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 * returns:
 *  Class in which these events will be returned
 */
TMWDEFS_CLASS_MASK TMWDEFS_GLOBAL sdnpdata_authErrorEventClass(
  void *pHandle);

/* function: sdnpdata_authIsCriticalReq
 * purpose: Determine if this message should be considered critical.
 *  The Secure Authentication spec lists function codes that are mandatory to
 *  be considered critical. Other function codes are optional.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  fc - function code received
 *  pRxMsg - pointer to received message
 *  msgLength - length of received message
 * returns:
 *  TMWDEFS_TRUE if this message is considered critical and should be challenged
 *  TMWDEFS_FALSE otherwise
 * NOTE: Application Confirms are a special case in the specification. To make
 *  Appl Confirms Critical you must call sdnpsesn_authPreChallApplConf() instead
 *  of returning TMWDEFS_TRUE from this function.
 *  See sdnpsesn.h for more information about sdnpsesn_authPreChallApplConf.
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_authIsCriticalReq(
  void           *pHandle,
  TMWTYPES_UCHAR fc,
  TMWTYPES_UCHAR *pRxMsg,
  TMWTYPES_USHORT msgLength);

/* function: sdnpdata_authGetRole
 * purpose: Return the role for this user.
 *  This function is currently only called by the sample code for
 *  sdnpdata_authRequestAllowed and not by the SDNP Library itself.
 *  It is here to make it clear that you should be considering the
 *  configured role for a user when deciding of a request should be allowed.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  userNumber - user number
 * returns: The following are defined in the Secure Authentication Specification
 *  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
 */
TMWTYPES_USHORT TMWDEFS_GLOBAL sdnpdata_authGetRole(
  void           *pHandle,
  TMWTYPES_USHORT userNumber);

/* function: sdnpdata_authRequestAllowed
 * purpose: Determine if this user is allowed to perform this request which has
 *  been challenged and the challenge reply from the master was valid.
 *  NOTE: If not authorized, this event should be logged.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  userNumber - user number
 *  fc - function code received
 *  pRxMsg - pointer to received message
 *  msgLength - length of received message
 * returns:
 *  TMWDEFS_TRUE if this request is allowed for this user
 *  TMWDEFS_FALSE otherwise
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_authRequestAllowed(
  void           *pHandle,
  TMWTYPES_USHORT userNumber,
  TMWTYPES_UCHAR  fc,
  TMWTYPES_UCHAR *pRxMsg,
  TMWTYPES_USHORT msgLength);

/* function: sdnpdata_authLogErrorRx
 * purpose: Log error message (Obj120V7) received from master.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  userNumber - user number from error message
 *  assocId - Association Id from error message
 *  sequenceNumber - Sequence number from error message
 *  errorCode - error code from error message
 *  pTimeStamp - pointer to time stamp from message
 *  pErrorText - optional error text from message
 *  errorTextLength - length of optional error text
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL sdnpdata_authLogErrorRx(
  void            *pHandle,
  TMWTYPES_USHORT  userNumber,
  TMWTYPES_USHORT  assocId,
  TMWTYPES_ULONG   sequenceNumber,
  TMWTYPES_UCHAR   errorCode,
  TMWDTIME        *pTimeStamp,
  TMWTYPES_CHAR   *pErrorText,
  TMWTYPES_USHORT  errorTextLength);

/* function: sdnpdata_authLogErrorTx
 * purpose: Log error for this user number.
 *  1815-2012 7.5.2.4 says it is recommended that error messages be transmitted on
 *  an association other than the one on which authentication is performed.
 * arguments:
 *  pHandle - handle to database returned from sdnpdata_init
 *  userNumber - user number
 *  assocId - association id indicating which master/outstation association
 *  sequenceNumber - Sequence number from error message
 *  errorCode - error identifier
 *  pErrorText - pointer to optional error text
 *  errorTextLength - length of optional error text.
 *  msgSent - indicates if the error message was sent to the master. When count
 *   exceeds max, error message g120v7 will not be sent.
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL sdnpdata_authLogErrorTx(
  void            *pHandle,
  TMWTYPES_USHORT  userNumber,
  TMWTYPES_USHORT  assocId,
  TMWTYPES_ULONG   sequenceNumber,
  TMWTYPES_UCHAR   errorCode,
  TMWDTIME        *pTimeStamp,
  TMWTYPES_CHAR   *pErrorText,
  TMWTYPES_USHORT  errorTextLength,
  TMWTYPES_BOOL    msgSent);
```

```
#if SDNPCNFG_SUPPORT_SA_VERSION5

#if TMWCNFG_SUPPORT_CRYPTO_AESGMAC
  /* function: sdnpdata_authStoreKSQ
   * purpose: The optional MAC algorithm AES-GMAC added in SAv5 requires the
   *   Key Change Sequence Number(KSQ) be retained over restarts.
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  KSQ - KSQ value to be stored so that it can be retained.
   * returns:
   *  void
   */
  void TMWDEFS_GLOBAL sdnpdata_authStoreKSQ(
    void            *pHandle,
    TMWTYPES_ULONG   KSQ);

  /* function: sdnpdata_authGetKSQ
   * purpose: The optional MAC algorithm AES-GMAC added in SAv5 requires the
   *   Key Change Sequence Number(KSQ) be retained over restarts.
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  pKSQ - pointer to memory to store the returned KSQ
   * returns:
   *  void
   */
  void TMWDEFS_GLOBAL sdnpdata_authGetKSQ(
    void            *pHandle,
    TMWTYPES_ULONG  *pKSQ);
#endif

#if DNPCNFG_SUPPORT_AUTHKEYUPDATE
  /* The following functions are required if Update Keys can be sent over DNP as
   * specified in DNP Secure Authentication Version 5. Symmetric update algorithms
   * are required, asymmetric algoritms are optional.
   */

  /* function: sdnpdata_authKeyChgMethodSupport
   * purpose: Determine if this key change method for sending update keys is
supported
   * arguments:
   *  pHandle - database handle returned from sdnpdata_init
   *  keyChangeMethod - less than 64 are symmetric
   *    DNPAUTH_KEYCH_SYMAES128_SHA1   = Symmetric AES-128
   *    DNPAUTH_KEYCH_SYMAES256_SHA256 = Symmetric AES-256
   *    DNPAUTH_KEYCH_SYMAES256_GMAC   = Symmetric AES-256 / AES-GMAC
   *
   *    DNPAUTH_KEYCH_ASYM_RSA1024_SHA1 = Asymmetric RSA-1024 / DSA / SHA1
   *    DNPAUTH_KEYCH_ASYM_RSA2048_SHA256 = Asymmetric RSA-2048 / DSA / SHA256
   *    DNPAUTH_KEYCH_ASYM_RSA3072_SHA256 = Asymmetric RSA-3072 / DSA / SHA256
   *    DNPAUTH_KEYCH_ASYM_RSA2048_SHA256_GMAC = Asymmetric RSA-2048 / DSA / AES-
GMAC
   *    DNPAUTH_KEYCH_ASYM_RSA3072_SHA256_GMAC = Asymmetric RSA-3072 / DSA / AES-
GMAC
   *    DNPAUTH_KEYCH_ASYM_RSA1024_RSA_SHA1 = Asymmetric RSA-1024 / RSA / SHA1
   *    DNPAUTH_KEYCH_ASYM_RSA2048_RSA_SHA256 = Asymmetric RSA-2048 / RSA / SHA256
   *    DNPAUTH_KEYCH_ASYM_RSA3072_RSA_SHA256 = Asymmetric RSA-3072 / RSA / SHA256
   *    DNPAUTH_KEYCH_ASYM_RSA2048_RSA_SHA256_GMAC = Asymmetric RSA-2048 / RSA /
AES-GMAC
   *    DNPAUTH_KEYCH_ASYM_RSA3072_RSA_SHA256_GMAC = Asymmetric RSA-3072 / RSA /
AES-GMAC
   * returns:
   *  TMWDEFS_TRUE if supported
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_authKeyChgMethodSupport(
    void           *pHandle,
    TMWTYPES_UCHAR  keyChangeMethod);

  /* function: sdnpdata_authGetOSName
   * purpose: Get the preconfigured name of this outstation. The master must also
   *  know this same name to send update keys over DNP.
   * arguments:
   *  pHandle - database handle returned from sdnpdata_init
   *  userNumber - user number that was returned by {sdnpdata_authUpdateKeyChangeReq
   *  pOSName - pointer to buffer where name should be copied
   *  pOSNameLength -  when called this is the maximum length allowed for
   *    outstation name, on return this should be set to the length of name returned
   * returns:
   *  TMWDEFS_TRUE if successful
   *  TMWDEFS_FALSE otherwise
   */
```

```
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_authGetOSName(
  void            *pHandle,
  TMWTYPES_CHAR   *pOSName,
  TMWTYPES_USHORT *pOSNameLength);

/* function: sdnpdata_authGetUserName
 * purpose: Get the globally unique name for this user
 *  that was stored by sdnpdata_authUserStatusChngRcvd
 * arguments:
 *  pHandle - database handle returned from sdnpdata_init
 *  userNumber - user number that was returned by sdnpdata_authUpdateKeyChangeReq
 *  pUserName - pointer to buffer where name should be copied
 *  pUserNameLength -  when called this is the maximum length allowed for
 *    user name, on return this should be set to the length of name to be filled
in
 * returns:
 *  TMWDEFS_TRUE if successful
 *  TMWDEFS_FALSE otherwise
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_authGetUserName(
  void            *pHandle,
  TMWTYPES_USHORT  userNumber,
  TMWTYPES_CHAR   *pUserName,
  TMWTYPES_USHORT *pUserNameLength);

/* function: sdnpdata_authUserCertRcvd
 *  NOTE: This only needs to be implemented if optional IEC 62351 Certificate sent
 *        in g120v8 for asymmetric key update is supported
 * purpose: Parse the User IEC 62351-8 Certificate data that was received
 *  in g120V8 message. The certificate will contain the user name,
 *  operation, user role, and user role expiry interval, status change sequence
(SCS) number,
 *  and area of responsibility.
 *  Verify the Area of Responsibility text string in the certificate matches
 *  at least one such string configured for this outstation.
 *  This function should return the user number for this user.
 *  NOTE: The SDNP Library will call a separate function to verify that sequence
number in the certificate incremented.
 *  NOTE: The SDNP library will call a separate function to store this certificate
 * arguments:
 *  pHandle - database handle returned from sdnpdata_init
 *  keyChangeMethod -
 *    DNPAUTH_KEYCH_ASYM_RSA1024_SHA1 = Asymmetric RSA-1024 / DSA / SHA1
 *    DNPAUTH_KEYCH_ASYM_RSA2048_SHA256 = Asymmetric RSA-2048 / DSA / SHA256
 *    DNPAUTH_KEYCH_ASYM_RSA3072_SHA256 = Asymmetric RSA-3072 / DSA / SHA256
 *    DNPAUTH_KEYCH_ASYM_RSA2048_SHA256_GMAC = Asymmetric RSA-2048 / DSA / AES-
GMAC
 *    DNPAUTH_KEYCH_ASYM_RSA3072_SHA256_GMAC = Asymmetric RSA-3072 / DSA / AES-
GMAC
 *    DNPAUTH_KEYCH_ASYM_RSA1024_RSA_SHA1 = Asymmetric RSA-1024 / RSA / SHA1
 *    DNPAUTH_KEYCH_ASYM_RSA2048_RSA_SHA256 = Asymmetric RSA-2048 / RSA / SHA256
 *    DNPAUTH_KEYCH_ASYM_RSA3072_RSA_SHA256 = Asymmetric RSA-3072 / RSA / SHA256
 *    DNPAUTH_KEYCH_ASYM_RSA2048_RSA_SHA256_GMAC = Asymmetric RSA-2048 / RSA /
AES-GMAC
 *    DNPAUTH_KEYCH_ASYM_RSA3072_RSA_SHA256_GMAC = Asymmetric RSA-3072 / RSA /
AES-GMAC
 *  certType - DNPAUTH_ID_CERTIFICATE or DNPAUTH_ATTRIB_CERTIFICATE
 *  pCertData - pointer to certificate data in message
 *  certDataLength - length of certificate data in message
 *  pStatusChangeSequence - pointer to value of SCS received from the master
 *   to be filled in by this function.
 *  pOperation - pointer to operation variable to be filled in by this function
 *  pError - pointer to error variable to be filled in by this function if it
 *   fails for any reason and returns 0 for a user number.
 *   DNPAUTH_ERROR_NONE if success
 *   DNPAUTH_ERROR_UNKNOWN_USER if the user does not exist
 *   DNPAUTH_ERROR_INVALIDSIG if the signature does not validate using configured
 *    Authority Public Key
 *   DNPAUTH_ERROR_INVALIDCERTDATA if the SCS value is not greater than previous
SCS
 *    or Area Of Responsibility does not match a configured value or other
 *    certification data errors.
 * returns:
 *  user number for this user name, or 0 if any failure
 *   ADD will need to determine a new unused userNumber.
 *   DELETE must still return correct nonzero user number if successful.
 *   For DELETE, if successful, invalidate the Update Key for this User
 *              (delete the user).
 *              return 0==failure if the user does not exist.
 */
TMWTYPES_USHORT TMWDEFS_GLOBAL sdnpdata_authUserCertRcvd(
    void              *pHandle,
    TMWTYPES_UCHAR     keyChangeMethod,
    TMWTYPES_UCHAR     certType,
    TMWTYPES_UCHAR    *pCertData,
    TMWTYPES_USHORT    certDataLength,
    TMWTYPES_ULONG    *pStatusChangeSequence,
    TMWTYPES_UCHAR    *pOperation,
    TMWTYPES_UCHAR    *pError);

  /* function: sdnpdata_authUserStatusChgRcvd
   * purpose: Store the User Status Change data that was received in g120V10 message
   *  The library will call this after verifying the certification data received.
   *  This data must be stored and will be retrieved by other sdnpdata_xx functions.
   *  This function should assign an available user number for this user.
   *  NOTE: the role and userRoleExpiryInterval should not take affect until the
   *         tmwcrypto_commitKey(true) is called!
   * arguments:
   *  pHandle - database handle returned from sdnpdata_init
   *  pUserName - pointer to globally unique identifier representing a user.
   *  userNameLength - length of user name
   *  keyChangeMethod - less than 64 are symmetric
   *    DNPAUTH_KEYCH_SYMAES128_SHA1    = Symmetric AES-128
   *    DNPAUTH_KEYCH_SYMAES256_SHA256 = Symmetric AES-256
   *    DNPAUTH_KEYCH_SYMAES256_GMAC    = Symmetric AES-256 / AES-GMAC
   *
   *    DNPAUTH_KEYCH_ASYM_RSA1024_SHA1 = Asymmetric RSA-1024 / DSA / SHA1
   *    DNPAUTH_KEYCH_ASYM_RSA2048_SHA256 = Asymmetric RSA-2048 / DSA / SHA256
   *    DNPAUTH_KEYCH_ASYM_RSA3072_SHA256 = Asymmetric RSA-3072 / DSA / SHA256
   *    DNPAUTH_KEYCH_ASYM_RSA2048_SHA256_GMAC = Asymmetric RSA-2048 / DSA / AES-
GMAC
   *    DNPAUTH_KEYCH_ASYM_RSA3072_SHA256_GMAC = Asymmetric RSA-3072 / DSA / AES-
GMAC
   *    DNPAUTH_KEYCH_ASYM_RSA1024_RSA_SHA1 = Asymmetric RSA-1024 / RSA / SHA1
   *    DNPAUTH_KEYCH_ASYM_RSA2048_RSA_SHA256 = Asymmetric RSA-2048 / RSA / SHA256
   *    DNPAUTH_KEYCH_ASYM_RSA3072_RSA_SHA256 = Asymmetric RSA-3072 / RSA / SHA256
   *    DNPAUTH_KEYCH_ASYM_RSA2048_RSA_SHA256_GMAC = Asymmetric RSA-2048 / RSA /
AES-GMAC
   *    DNPAUTH_KEYCH_ASYM_RSA3072_RSA_SHA256_GMAC = Asymmetric RSA-3072 / RSA /
AES-GMAC
   *  operation -  DNPAUTH_USER_STATUS_ADD, DNPAUTH_USER_STATUS_DELETE,
   *            or DNPAUTH_USER_STATUS_CHANGE
   *   For CHANGE  return 0==failure if the user does not exist.
   *   For DELETE, invalidate the Update Key for this User (delete the user).
   *               return 0==failure if the user does not exist.
   *  role - user role definition, what this user is allowed to do.
   *   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
   *  userRoleExpiryInterval - Number of days after receiving the new
   *   User Update Key in the g120v13 request.
   * returns:
   *  user number for this user name, or 0 if failure
   *  DELETE must still return correct nonzero user number if successful.
   */
  TMWTYPES_USHORT TMWDEFS_GLOBAL sdnpdata_authUserStatusChgRcvd(
    void              *pHandle,
    TMWTYPES_CHAR     *pUserName,
    TMWTYPES_USHORT    userNameLength,
    TMWTYPES_UCHAR     keyChangeMethod,
    TMWTYPES_UCHAR     operation,
    TMWTYPES_USHORT    userRole,
    TMWTYPES_USHORT    userRoleExpiryInterval);

  /* function: sdnpdata_authUpdateKeyChgReq
   * purpose: Process received Update Key Change Request g120v11
   * arguments:
   *  pHandle - database handle returned from sdnpdata_init
   *  keyChangeMethod - less than 64 are symmetric
   *    DNPAUTH_KEYCH_SYMAES128_SHA1    = Symmetric AES-128
   *    DNPAUTH_KEYCH_SYMAES256_SHA256 = Symmetric AES-256
   *    DNPAUTH_KEYCH_SYMAES256_GMAC    = Symmetric AES-256 / AES-GMAC
   *
   *    DNPAUTH_KEYCH_ASYM_RSA1024_SHA1 = Asymmetric RSA-1024 / DSA / SHA1
   *    DNPAUTH_KEYCH_ASYM_RSA2048_SHA256 = Asymmetric RSA-2048 / DSA / SHA256
   *    DNPAUTH_KEYCH_ASYM_RSA3072_SHA256 = Asymmetric RSA-3072 / DSA / SHA256
   *    DNPAUTH_KEYCH_ASYM_RSA2048_SHA256_GMAC = Asymmetric RSA-2048 / DSA / AES-
```

```
GMAC
   *    DNPAUTH_KEYCH_ASYM_RSA3072_SHA256_GMAC = Asymmetric RSA-3072 / DSA / AES-
GMAC
   *    DNPAUTH_KEYCH_ASYM_RSA1024_RSA_SHA1 = Asymmetric RSA-1024 / RSA / SHA1
   *    DNPAUTH_KEYCH_ASYM_RSA2048_RSA_SHA256 = Asymmetric RSA-2048 / RSA / SHA256
   *    DNPAUTH_KEYCH_ASYM_RSA3072_RSA_SHA256 = Asymmetric RSA-3072 / RSA / SHA256
   *    DNPAUTH_KEYCH_ASYM_RSA2048_RSA_SHA256_GMAC = Asymmetric RSA-2048 / RSA /
AES-GMAC
   *    DNPAUTH_KEYCH_ASYM_RSA3072_RSA_SHA256_GMAC = Asymmetric RSA-3072 / RSA /
AES-GMAC
   *  pUserName - pointer to globally unique identifier representing user.
   *  userNameLength - length of user name
   * returns:
   *  user number for this user name (should be same number returned from
   *  sdnpdata_authUserCertRcvd or sdnpdata_authUserStatusChgRcvd)
   *  or 0 if failure
   */
  TMWTYPES_USHORT TMWDEFS_GLOBAL sdnpdata_authUpdateKeyChgReq(
    void            *pHandle,
    TMWTYPES_UCHAR   keyChangeMethod,
    TMWTYPES_CHAR   *pUserName,
    TMWTYPES_USHORT  userNameLength);

#endif  /* DNPCNFG_SUPPORT_AUTHKEYUPDATE */

  /* function: sdnpdata_authSecStatQuantity
   * purpose: Return the number of security statistics in the
   *  specified database.
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   * returns:
   *  The number of security statistics (see note for
   *  sdnpdata_xxxQuantity at top of this file)
   */
  TMWTYPES_USHORT TMWDEFS_GLOBAL sdnpdata_authSecStatQuantity(
    void *pHandle);

  /* function: sdnpdata_authSecStatGetPoint
   * purpose: Return a handle to a specific data point. This handle is
   *  used by the routines below to read values from this statistic
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  pointNum - point number to return
   * returns:
   *  Pointer to specified data point or TMWDEFS_NULL if this point
   *   is currently disabled for reading.
   */
  void * TMWDEFS_GLOBAL sdnpdata_authSecStatGetPoint(
    void *pHandle,
    TMWTYPES_USHORT pointNum);

  /* function: sdnpdata_authSecStatThreshold
   * purpose: Retrieve the Security Statistic max or threshold value
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  index - index indicating which statistic
   *   for example DNPAUTH_UNEXPECTED_MSG_INDEX
   */
  TMWTYPES_USHORT TMWDEFS_GLOBAL sdnpdata_authSecStatThreshold(
    void            *pHandle,
    TMWTYPES_USHORT  index);

  /* function: sdnpdata_authSecStatSet
   * purpose: Set the Security Statistic for this index.
   * arguments:
   *  pPoint - handle to data point returned from 'getPoint' function.
   *  value - value to set for this statistic
   * returns:
   *  void
   */
  void TMWDEFS_GLOBAL sdnpdata_authSecStatSet(
    void            *pPoint,
    TMWTYPES_ULONG   value);

  /* function: sdnpdata_authSecStatEventClass
   * purpose: Retrieve the event class for this point
   * NOTE IEEE 1815-2012 Table 5-2 says this statistic SHALL be in
   *  class 1, 2,or 3. It must be in an event class.
   * arguments:
   *  pPoint - handle to data point returned from 'getPoint' function.
   * returns:
```

```
 *  Class in which these events will be returned
 */
TMWDEFS_CLASS_MASK TMWDEFS_GLOBAL sdnpdata_authSecStatEventClass(
  void *pPoint);

/* function: sdnpdata_authSecStatIsClass0
 * purpose: Should this point be reported in response to an object 60
 *   variation 1 read request. This allows individual points to be excluded
 *   from a class 0 response but still readable by a specific object group
 *   read request.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  TMWDEFS_TRUE if point should be reported.
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_authSecStatIsClass0(
  void *pPoint);

/* function: sdnpdata_authSecStatEventDefVariation
 * purpose: Determine default variation for this security statistic
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  classMask - mask indicating what event class is being requested
 * returns:
 *  default variation for this point
 *  NOTE: this will only be called if the default variation for security
 *   statistic change events obj122DefaultVariation is configured as zero
 */
TMWTYPES_UCHAR TMWDEFS_GLOBAL sdnpdata_authSecStatEventDefVariation(
  void *pPoint,
  TMWDEFS_CLASS_MASK classMask);

/* function: sdnpdata_authSecStatEventMode
 * purpose: Determine event mode for this point
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 * returns:
 *  event mode for this point, SOE or LAST
 *  NOTE: this will only be called if the event mode for security statistics
 *   if securityStatEventMode is configured as TMWDEFS_EVENT_MODE_PER_POINT
 */
TMWDEFS_EVENT_MODE TMWDEFS_GLOBAL sdnpdata_authSecStatEventMode(
  void *pPoint);

/* function: sdnpdata_authSecStatAssignClass
 * purpose: Assign the class in which events from this data point
 *  will belong.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  classMask - new class in which to generate events from this point
 * returns:
 *  TMWDEFS_TRUE if successful, else TMWDEFS_FALSE
 */
TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_authSecStatAssignClass(
  void *pPoint,
  TMWDEFS_CLASS_MASK classMask);

/* function: sdnpdata_authSecStatRead
 * purpose: Return the current value of the specified point.
 * arguments:
 *  pPoint - handle to data point returned from 'getPoint' function.
 *  pValue - pointer to location to store current value
 *  pFlags - pointer to location to store current DNP3 flags
 *   pFlags contains a status indication and the current state of the point.
 *   The following values (or OR'd combinations) are valid for this type:
 *      DNPDEFS_DBAS_FLAG_OFF_LINE - the point is off-line, and the returned
 *        state of this point may not be correct
 *      DNPDEFS_DBAS_FLAG_ON_LINE - the binary counter 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 device
 *        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_STAT_ROLLOVER - the accumulated value has exceeded
 *        has exceeded its maximum and rolled over to zero. The counter
 *        value should be set to 0 upon rollover, and counting is resumed as
 *        normal. The Rollover bit should be cleared when the counter value
 *        and roll-over state have been reported.
 *      DNPDEFS_DBAS_FLAG_DISCONTINUITY - value cannot be compared against
 *        a prior value to obtain the correct count difference
 * returns:
 *  void
 */
void TMWDEFS_GLOBAL sdnpdata_authSecStatRead(
  void *pPoint,
  TMWTYPES_ULONG *pValue,
  TMWTYPES_UCHAR *pFlags);

/* function: sdnpdata_authSesnKeyStatCount
 * purpose:  Number of Key Status Requests within Expected Session Key Change
 *  Interval has been exceeded. Notify a human and if other associations exist,
 *  send an Error Event g120v7 with code 12 Max Session Key Status Requests
 *  Exceeded on those associations.
 * arguments:
 *  pHandle - database handle returned from sdnpdata_init
 *  assocId - association id for status request
 *  userNumber - userNumber from status request
 *  sequenceNumber - key change sequence number from status request
 * returns:
 */
void TMWDEFS_GLOBAL sdnpdata_authSesnKeyStatCount(
  void            *pHandle,
  TMWTYPES_USHORT  assocId,
  TMWTYPES_USHORT  userNumber,
  TMWTYPES_ULONG   sequenceNumber);

/* function: sdnpdata_authLogUnexpectedMsg
 * purpose:  log that an unexpected message was received
 * arguments:
 *  pHandle - database handle returned from sdnpdata_init
 *  state - current master state ie SDNPAUTH_STATE_XXX
 *  event - event being processes ie SDNPAUTH_EVT_XX
 *  pRxFragment - pointer to message that was received.
 * returns:
 */
void TMWDEFS_GLOBAL sdnpdata_authLogUnexpectedMsg(
  void            *pHandle,
  TMWTYPES_UCHAR   state,
  TMWTYPES_ULONG   event,
  TMWSESN_RX_DATA *pRxFragment);

/* function: sdnpdata_authLogTx
 * purpose:  log the Secure Authentication message that is being sent
 *  to Master. g120v7 says this event should be queued for transmission on other
 *  associations if they exist. (only the most recent shall be buffered)
 * arguments:
 *  pHandle - database handle returned from sdnpdata_init
 *  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 sdnpdata_authLogTx(
  void            *pHandle,
  TMWTYPES_UCHAR   variation,
  TMWTYPES_USHORT  userNumber,
  TMWTYPES_ULONG   sequenceNumber,
  TMWTYPES_UCHAR  *pMsgBuf,
  TMWTYPES_USHORT  msgLength);

/* function: sdnpdata_authLogRx
 * purpose:  log the Secure Authentication message that was received
 *  from Master
 * arguments:
 *  pHandle - database handle returned from sdnpdata_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
 */
```

```
  void TMWDEFS_GLOBAL sdnpdata_authLogRx(
    void            *pHandle,
    TMWTYPES_UCHAR   variation,
    TMWTYPES_USHORT  userNumber,
    TMWTYPES_ULONG   sequenceNumber,
    TMWTYPES_UCHAR  *pMsgBuf,
    TMWTYPES_USHORT  msgLength);

  /* function: sdnpdata_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 sdnpdata_init
   * returns:
   */
  void TMWDEFS_GLOBAL sdnpdata_authLogMaxRekeyTCPClose(
    void            *pHandle);

#endif  /* SDNPCNFG_SUPPORT_SA_VERSION5 */

#if SDNPCNFG_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
   */

  /* function: sdnpdata_authLogChallTx
   * purpose:  log that a challenge has been sent to the master
   *    NOTE: Only required for SA_VERSION2
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  userNumber - user number sent in challenge (0)
   *  sequenceNumber - Challenge sequence number transmitted
   *  macAlgorithm - MAC Algorithm Id transmitted
   *  reason - Reason challenge was transmitted
   *   Authentication Spec now only defines DNPAUTH_REASON_CRITICAL==1
   * returns:
   *  void
   */
  void TMWDEFS_GLOBAL sdnpdata_authLogChallTx(
    void            *pHandle,
    TMWTYPES_USHORT  userNumber,
    TMWTYPES_ULONG   sequenceNumber,
    TMWTYPES_UCHAR   macAlgorithm,
    TMWTYPES_UCHAR   reason);

  /* function: sdnpdata_authLogChallRplyTx
   * purpose:  log that a challenge reply has been sent to the master
   *    NOTE: Only required for SA_VERSION2
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  userNumber - user number sent in challenge reply
   *  sequenceNumber - Challenge sequence number transmitted
   * returns:
   *  void
   */
  void TMWDEFS_GLOBAL sdnpdata_authLogChallRplyTx(
    void            *pHandle,
    TMWTYPES_USHORT  userNumber,
    TMWTYPES_ULONG   sequenceNumber);

  /* function: sdnpdata_authLogKeyStatusTx
   * purpose:  log that a key status request has been sent to the master
   *    NOTE: Only required for SA_VERSION2
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  userNumber - user number
   *  sequenceNumber - Sequence number transmitted
   *  keyWrapAgorithm - Key Wrap Algorithm ID transmitted
   *  keyStatus - key status transmitted
   *  macAlgorithm - MAC Algorithm Id transmitted
   * returns:
   *  void
   */
  void sdnpdata_authLogKeyStatusTx(
    void            *pHandle,
    TMWTYPES_USHORT  userNumber,
    TMWTYPES_ULONG   sequenceNumber,
    TMWTYPES_UCHAR   keyWrapAlgorithm,
    TMWTYPES_UCHAR   keyStatus,
    TMWTYPES_UCHAR   macAlgorithm);

  /* function: sdnpdata_authLogAggrTx
   * purpose:  log that an aggressive mode request has been sent
   *    NOTE: Only required for SA_VERSION2
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  userNumber - user number
   *  sequenceNumber - Challenge Sequence number transmitted
   * returns:
   *  void
   */
  void TMWDEFS_GLOBAL sdnpdata_authLogAggrTx(
    void            *pHandle,
    TMWTYPES_USHORT  userNumber,
    TMWTYPES_ULONG   sequenceNumber);

  /* function: sdnpdata_authLogChallRx
   * purpose:  log that a challenge request has been received
   *    NOTE: Only required for SA_VERSION2
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  userNumber - user number
   *  sequenceNumber - Challenge sequence number from message
   *  macAlgorithm - MAC Algorithm ID from message
   *  reason - reason from message
   * returns:
   *  void
   */
  void TMWDEFS_GLOBAL sdnpdata_authLogChallRx(
    void            *pHandle,
    TMWTYPES_USHORT  userNumber,
    TMWTYPES_ULONG   sequenceNumber,
    TMWTYPES_UCHAR   macAlgorithm,
    TMWTYPES_UCHAR   reason);

  /* function: sdnpdata_authLogChallRplyRx
   * purpose:  log that a challenge reply has been received
   *    NOTE: Only required for SA_VERSION2
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  userNumber - user number from message
   *  sequenceNumber - Sequence number from message
   *  status - TMWDEFS_TRUE if challenge reply was valid
   *   TMWDEFS_FALSE otherwise
   * returns:
   *  void
   */
  void TMWDEFS_GLOBAL sdnpdata_authLogChallRplyRx(
    void            *pHandle,
    TMWTYPES_USHORT  userNumber,
    TMWTYPES_ULONG   sequenceNumber,
    TMWTYPES_BOOL    status);

  /* function: sdnpdata_authLogAggrRx
   * purpose:  log that an aggressive mode request has been received
   *    NOTE: Only required for SA_VERSION2
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  userNumber - user number from message
   *  sequenceNumber - Sequence number from message
   *  status - TMWDEFS_TRUE if aggressive mode request was valid
   *   TMWDEFS_FALSE otherwise
   * returns:
   *  void
   */
  void TMWDEFS_GLOBAL sdnpdata_authLogAggrRx(
    void            *pHandle,
    TMWTYPES_USHORT  userNumber,
    TMWTYPES_ULONG   sequenceNumber,
    TMWTYPES_BOOL    status);

  /* function: sdnpdata_authLogKeyStatRqRx
   * purpose:  log that a key status request has been received
   *    NOTE: Only required for SA_VERSION2
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  userNumber - user number from message
   * returns:
   *  void
   */
  void TMWDEFS_GLOBAL sdnpdata_authLogKeyStatRqRx(
    void            *pHandle,
    TMWTYPES_USHORT  userNumber);

  /* function: sdnpdata_authLogKeyChangeRx
   * purpose:  log that a key change request has been received
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *    NOTE: Only required for SA_VERSION2
   *  userNumber - user number from message
   *  sequenceNumber - Key Change Sequence number from message
   *  status - TMWDEFS_TRUE if key change request was valid
   *   TMWDEFS_FALSE otherwise
   * returns:
   *  void
   */
  void TMWDEFS_GLOBAL sdnpdata_authLogKeyChangeRx(
    void            *pHandle,
    TMWTYPES_USHORT  userNumber,
    TMWTYPES_ULONG   sequenceNumber,
    TMWTYPES_BOOL    status);

  /* function: sdnpdata_authDecryptKeyWrapData
   * purpose: Decrypt the data using the update key provided,
   *   return the decrypted data in pValue setting *pLength to
   *   the length of the returned data
   *    NOTE: Only required for SA_VERSION2, see utils/tmwcrypto.h for SA_VERSION5
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  userNumber - user number
   *  algorithm - Decryption algorithm to use
   *   DNPAUTH_KEYWRAP_AES128 is the only one currently specified.
   *   other values reserved for future use or vendor specific choices
   *  pEncryptedData - pointer to data to be decrypted
   *  encryptedDataLength - length of data to be decrypted
   *  pPlainValue - where to copy the plain (decrypted) data (including padding
  bytes
   *   that may have been added at master).
   *  pPlainLength - when called this is the maximum length allowed for plain data,
   *   on return this should be set to the length of the plain data. This will
   *   contain any padding bytes that were added at master. This function should
   *   not try to determine what padding might have been added.
   * returns:
   *  TMWDEFS_TRUE of successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_authDecryptKeyWrapData(
    void            *pHandle,
    TMWTYPES_USHORT  userNumber,
    TMWTYPES_UCHAR   algorithm,
    TMWTYPES_UCHAR  *pEncryptedData,
    TMWTYPES_USHORT  encryptedValueLength,
    TMWTYPES_UCHAR  *pPlainValue,
    TMWTYPES_USHORT *pPlainLength);

    /* function: sdnpdata_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
     *    NOTE: Only required for SA_VERSION2, see utils/tmwcrypto.h for
     *          SA_VERSION5
     * arguments:
     *  HMACAlgorithm - 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 sdnpdata_authHMACSupport(
      TMWTYPES_UCHAR HMACAlgorithm);

  /* function: sdnpdata_authHMACValue
```

```
   * purpose: using the specified algorithm and key calculate the
   *  Keyed-Hash Message Authentication Code (MAC) value of the
   *   data provided.
   *   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
   *  pMACValue - pointer to where hashed data should be copied
   *  pMACValueLength - 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 of successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_authHMACValue(
    TMWTYPES_UCHAR    algorithm,
    DNPDATA_AUTH_KEY *pKey,
    TMWTYPES_UCHAR   *pData,
    TMWTYPES_ULONG    dataLength,
    TMWTYPES_UCHAR   *pHMACValue,
    TMWTYPES_USHORT  *pHMACValueLength);

  /* function: sdnpdata_authRandomChallengeData
   * purpose:  generate pseudo-random data,
   *  using algorithm specified in Secure Authentication Spec
   *  and 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,
   *   on return this should be set to the length of the random data.
   * returns:
   *  TMWDEFS_TRUE of successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_authRandomChallengeData(
    TMWTYPES_UCHAR  *pBuf,
    TMWTYPES_USHORT  minLength,
    TMWTYPES_USHORT *pLength);
#endif

#ifdef __cplusplus
}
#endif

#if SDNPDATA_SUPPORT_XML2
/* The following defines and structures are used for retrieving information to
 * create the device profile for this device
 */

/* This is used in many places, for names and other strings */
#define SDNPXML_MAXSTRINGLENGTH 255

/* deviceConfig notableAdditions, max number of notes and notable additions */
#define SDNPXML_MAX_NOTABLEADDITIONS 2

/* deviceConfig configuration methods, max number of software types */
#define SDNPXML_MAX_CONFIGSOFTWARE 2

/* deviceConfig configuration methods, max other types */
#define SDNPXML_MAX_CONFIGOTHERMETHODS 2

/* deviceConfig online and offlineXmlFileNames, max number xml filenames and notes
*/
#define SDNPXML_MAX_XMLFILENAMES 2

/* deviceConfig connectionsSupported, max other connection types */
#define SDNPXML_MAX_OTHERCONNECTIONS 2

/* networkConfig ipAddressOfRemoteDevice, max ip addresses of remote devices */
```

```
#define SDNPXML_MAX_IPADDRESSES 4

typedef struct {
  /* note is optional null terminated string
   * Empty string will not be output
   */
  TMWTYPES_CHAR note[SDNPXML_MAXSTRINGLENGTH];
  TMWTYPES_BOOL readAccess;
  TMWTYPES_BOOL writeAccess;

  /* null terminated strings
   * These two strings are required by schema.
   */
  TMWTYPES_CHAR filename[SDNPXML_MAXSTRINGLENGTH];
  TMWTYPES_CHAR description[SDNPXML_MAXSTRINGLENGTH];
} SDNPDATA_XMLFILETYPE;


/* device config structure */
typedef struct {

  /* vendor name, null terminated string */
  TMWTYPES_CHAR vendorName[SDNPXML_MAXSTRINGLENGTH];

  /* device name, null terminated string */
  TMWTYPES_CHAR deviceName[SDNPXML_MAXSTRINGLENGTH];

  /* hardware version, null terminated string */
  TMWTYPES_CHAR hardwareVersion[SDNPXML_MAXSTRINGLENGTH];

  /* software version, null terminated string */
  TMWTYPES_CHAR softwareVersion[SDNPXML_MAXSTRINGLENGTH];

  /* Device Profile Document Version Number */
  TMWTYPES_ULONG documentVersionNumber;

  /* Set all that apply to TMWDEFS_TRUE */
  struct {
    TMWTYPES_BOOL none;
    TMWTYPES_BOOL level1;
    TMWTYPES_BOOL level2;
    TMWTYPES_BOOL level3;
    TMWTYPES_BOOL level4;
  } dnpLevelSupported;

  /* Set all that apply to TMWDEFS_TRUE */
  struct {
    /* SCL knows about these
     * selfAddressReservation
     * dataSets
     * fileTransfer
     * virtualTerminal;
     * FC31ActivateConfiguration
     * secureAuthentication
     */
    TMWTYPES_BOOL mappingToIEC61850ObjectModels;
  } supportedFunctionBlocks;

  /* notableAdditions */
  /* Null terminated strings */
  /* Leave notableAdditions[x].note[0]==0 if it is not to be output. */
  /* Leave notableAdditions[x].notableAddition[0]==0 if it is not to be output. */
  struct {
    TMWTYPES_CHAR note[SDNPXML_MAXSTRINGLENGTH];
    TMWTYPES_CHAR notableAddition[SDNPXML_MAXSTRINGLENGTH];
  } notableAdditions[SDNPXML_MAX_NOTABLEADDITIONS];

  /* configurationMethods */
  /* Set all that apply to TMWDEFS_TRUE
   * software and other are null terminated strings
   */
  struct {
    TMWTYPES_BOOL xmlViaFileTransfer;
    TMWTYPES_BOOL xmlViaOtherTransportMechanism;
    TMWTYPES_BOOL terminal;

    /* software */
    /* null terminated strings,
     * Empty strings, where name[0]==0,will not be output
     * Empty strings, where version[0]==0,will not be output
```

```
         */
        struct {
          TMWTYPES_CHAR name[SDNPXML_MAXSTRINGLENGTH];
          TMWTYPES_CHAR version[SDNPXML_MAXSTRINGLENGTH];
        } software[SDNPXML_MAX_CONFIGSOFTWARE];

        TMWTYPES_BOOL proprietaryFileViaFileTransfer;
        TMWTYPES_BOOL proprietaryFileViaOtherTransportMechanism;
        TMWTYPES_BOOL direct;
        TMWTYPES_BOOL factory;
        TMWTYPES_BOOL protocol;

        /* other
         * null terminated strings
         * Empty strings, where other[x][0]==0, will not be output
         */
        TMWTYPES_CHAR other[SDNPXML_MAX_CONFIGOTHERMETHODS][SDNPXML_MAXSTRINGLENGTH];
      } configurationMethods;

      /* onlineXmlFileNames
       * Set all that apply to TMWDEFS_TRUE
       */
      struct {
        TMWTYPES_BOOL dnpDPReadSupported;
        TMWTYPES_BOOL dnpDPCapReadSupported;
        TMWTYPES_BOOL dnpDPCfgReadSupported;

        SDNPDATA_XMLFILETYPE xmlFile[SDNPXML_MAX_XMLFILENAMES];

        /* null terminated string
         * Empty string will not be output
         */
        TMWTYPES_CHAR note[SDNPXML_MAX_XMLFILENAMES][SDNPXML_MAXSTRINGLENGTH];
      } onlineXmlFileNames;

      /* offlineXmlFileNames
       * Set all that apply to TMWDEFS_TRUE
       */
      struct {
        TMWTYPES_BOOL dnpDPReadSupported;
        TMWTYPES_BOOL dnpDPWriteSupported;
        TMWTYPES_BOOL dnpDPCapReadSupported;
        TMWTYPES_BOOL dnpDPCapWriteSupported;
        TMWTYPES_BOOL dnpDPCfgReadSupported;
        TMWTYPES_BOOL dnpDPCfgWriteSupported;

        SDNPDATA_XMLFILETYPE xmlFile[SDNPXML_MAX_XMLFILENAMES];

        /* null terminated string
         * Empty string will not be output
         */
        TMWTYPES_CHAR note[SDNPXML_MAX_XMLFILENAMES][SDNPXML_MAXSTRINGLENGTH];
      } offlineXmlFileNames;

      /* connectionsSupported */
      /* Set all that apply to TMWDEFS_TRUE */
      struct {
        TMWTYPES_BOOL serial;
        TMWTYPES_BOOL network;

        /* other is a null terminated string
         * Empty strings where other[x][0]==0 will not be output
         */
        TMWTYPES_CHAR other[SDNPXML_MAX_OTHERCONNECTIONS][SDNPXML_MAXSTRINGLENGTH];
      }connectionsSupported;

    } SDNPDATA_XML_DEVICECONFIG;


    /* Serial Config */
    /* The following enumeration is used to specify what serialConfig
     * serialParameters value is specified
     */
    typedef enum {
      SDNPDATA_XML_SERIALPARAMAEMPTY,
      SDNPDATA_XML_SERIALPARAMASYNC,
      SDNPDATA_XML_SERIALPARAMOTHER
    } SDNPDATA_XML_SERIALPARAMTYPE;

    /* The following enumeration is used to specify what serialConfig
     * interCharacterTimeOut value is specified
```

```
  */
typedef enum {
  /* not output */
  SDNPDATA_XML_INTERCHAREMTPY,
  /* notChecked */
  SDNPDATA_XML_INTERCHARNOTCHECKED,
  /* noGapPermitted */
  SDNPDATA_XML_INTERCHARNOGAPPERM,
  /* valueBitTimes */
  SDNPDATA_XML_INTERCHARVALUEBIT,
  /* valueMilliseconds */
  SDNPDATA_XML_INTERCHARVALUEMILL,
  /* variable, variable field should be filled in */
  SDNPDATA_XML_INTERCHARVARIABLE
} SDNPDATA_XML_INTERCHARTOTYPE;

/* The following enumeration is used to specify what serialConfig
 * interCharacterGap value is specified
 */
typedef enum {
  /* no output */
  SDNPDATA_XML_INTERCHARGAPEMPTY,
  /* none */
  SDNPDATA_XML_INTERCHARGAPNONE,
  /* maximumBitTimes, set value field */
  SDNPDATA_XML_INTERCHARGAPMAXBIT,
  /* maximumMilliseconds, set value field */
  SDNPDATA_XML_INTERCHARGAPMAXMILL
} SDNPDATA_XML_INTERCHARGAPTYPE;


/* Serial Config structure */

/* The following enumeration is used to specify whether flow control option
 * should be Asserted, DeAsserted or Neither
 */
typedef enum {
  /* Neither asserted nor deasserted */
  SDNPDATA_XML_ASSERTED_EMPTY,
  /* asserted */
  SDNPDATA_XML_ASSERTED_ASSERTED,
  /* deasserted */
  SDNPDATA_XML_ASSERTED_DEASSERTED
} SDNPDATA_XML_ASSERTEDTYPE;

typedef struct {

  /* serial port name, null terminated string */
  TMWTYPES_CHAR portName[SDNPXML_MAXSTRINGLENGTH];

  /* serial parameters */
  struct {
    /* if type == SDNPDATA_XML_SERIALPARAMOTHER, fill other in. */
    SDNPDATA_XML_SERIALPARAMTYPE type;
    TMWTYPES_CHAR other[SDNPXML_MAXSTRINGLENGTH];
  } serialParameters;

  /* baud rate */
  TMWTYPES_ULONG baudRate;

  /* rs232 options */
  struct {
    /* sequence of 0 or more of the following.
     * Leave other.length 0 if other is not to be output.
     */
    TMWTYPES_BOOL none;
    struct {
      TMWTYPES_BOOL assertsRTSBeforeTx;
      TMWTYPES_BOOL assertsDTRBeforeTx;
      TMWTYPES_BOOL assertsRTSBeforeRx;
      TMWTYPES_BOOL assertsDTRBeforeRx;
      TMWTYPES_BOOL alwaysAssertsRTS;
      TMWTYPES_BOOL alwaysAssertsDTR;
      SDNPDATA_XML_ASSERTEDTYPE requiresCTSBeforeTx;
      SDNPDATA_XML_ASSERTEDTYPE requiresDCDBeforeTx;
      SDNPDATA_XML_ASSERTEDTYPE requiresDSRBeforeTx;
      SDNPDATA_XML_ASSERTEDTYPE requiresRIBeforeTx;
      TMWTYPES_BOOL rxInactive;
      SDNPDATA_XML_ASSERTEDTYPE requiresCTSBeforeRx;
      SDNPDATA_XML_ASSERTEDTYPE requiresDCDBeforeRx;
      SDNPDATA_XML_ASSERTEDTYPE requiresDSRBeforeRx;
```

```
        SDNPDATA_XML_ASSERTEDTYPE requiresRIBeforeRx;
        TMWTYPES_BOOL alwaysIgnoresCTS;
        TMWTYPES_BOOL alwaysIgnoresDCD;
        TMWTYPES_BOOL alwaysIgnoresDSR;
        TMWTYPES_BOOL alwaysIgnoresRI;
        TMWTYPES_CHAR other[SDNPXML_MAXSTRINGLENGTH];
      }rs232Options;

      /* rs422 options */
      struct {
        /*       * sequence of 0 or more of the following.
          * Other is a null terminated string. Leave other[0]==0
          * if other is not to be output.
          */
        TMWTYPES_BOOL requiresIndicationBeforeRx;
        TMWTYPES_BOOL assertsControlBeforeTx;
        TMWTYPES_CHAR other[SDNPXML_MAXSTRINGLENGTH];
      }rs422Options;

      /* 485 options*/
      struct {
        /* sequence of 0 or more of the following.
          * Leave other[0]==0 if other is not to be output.
          */
        TMWTYPES_BOOL requiresRxInactiveBeforeTx;
        TMWTYPES_CHAR other[SDNPXML_MAXSTRINGLENGTH];
      }rs485Options;
    }flowControl;

    /* SCL knows link status interval
      * linkStatusInterval
      */

    struct {
      /* Choice of one of these */
      /* If supported is set to TMWDEFS_FALSE it will be output as
        * "<no />" to schema
        */
      TMWTYPES_BOOL supported;
      TMWTYPES_CHAR yes[SDNPXML_MAXSTRINGLENGTH];
    } supportsCollisionAvoidance;

    struct {
      SDNPDATA_XML_INTERCHARTOTYPE type;
      union {
        TMWTYPES_ULONG value;
        TMWTYPES_CHAR variable[SDNPXML_MAXSTRINGLENGTH];
      }value;
    }interCharacterTimeout;

    struct {
      SDNPDATA_XML_INTERCHARGAPTYPE type;
      TMWTYPES_ULONG value;
    } interCharacterGap;

  } SDNPDATA_XML_SERIALCONFIG;


  /* Network Config */
  /* The following enumeration is used to specify what networkConfig
    * typeOfEndPoint value is specified
    */
  typedef enum {
    /* tcpInitiating */
    SDNPDATA_XML_TCPINITIATING,
    /* tcpListening */
    SDNPDATA_XML_TCPLISTENING,
    /* tcpDual */
    SDNPDATA_XML_TCPDUAL,
    /* udpDatagram */
    SDNPDATA_XML_UDPDATAGRAM
  } SDNPDATA_XML_ENDPOINTTYPE;

  /* The following enumeration is used to specify what networkConfig
    * tcpConnectionEstablishment value is specified
    */
  typedef enum {
    /* Choose this one to cause "<>" empty value */
    SDNPDATA_XML_TCPEMPTY,
    /* allowsAll */
    SDNPDATA_XML_TCPALL,
    /* basedOnIPAddress */
    SDNPDATA_XML_TCPBASEDONIP,
    /* basedOnListOfIPAddresses */
    SDNPDATA_XML_TCPBASEDONLIST,
    /* basedOnWildcardIPAddress */
    SDNPDATA_XML_TCPBASEDONWILD,
    /* basedOnListOfWildcardIPAddresses */
    SDNPDATA_XML_TCPBASEDONWILDLIST,
    /* other */
    /* If this is specified, null terminated other string should be filled in. */
    SDNPDATA_XML_TCPOTHER,
  } SDNPDATA_XML_TCPCONNECTTYPE;

  /* The following enumeration is used to specify what networkConfig
   * udpPortForResponses value is specified
   */
  typedef enum {
    /* Choose this one to cause "<>" empty value */
    SDNPDATA_XML_UDPEMPTY,
    /* none */
    SDNPDATA_XML_UDPNONE,
    /* useSourcePortNumber */
    SDNPDATA_XML_UDPUSESOURCE,
    /* value, set value field to value to be output */
    SDNPDATA_XML_UDPVALUE
  } SDNPDATA_XML_UDPRESPONSETYPE;

  /* The following enumeration is used to specify what networkConfig
   * timeSynchronization value is specified
   */
  typedef enum {
    /* notSupported */
    SDNPDATA_XML_TIMESYNCNOTSUP,
    /* dnpLANProcedure */
    SDNPDATA_XML_TIMESYNCLAN,
    /* dnpWriteTimeProcedure */
    SDNPDATA_XML_TIMESYNCWRITE,
    /* other, set other field to be output*/
    SDNPDATA_XML_TIMESYNCOTHER
  } SDNPDATA_XML_TIMESYNCTYPE;

  /* network config structure */
  typedef struct {

    /* network port name */
    TMWTYPES_CHAR portName[SDNPXML_MAXSTRINGLENGTH];

    SDNPDATA_XML_ENDPOINTTYPE typeOfEndPoint;

    /* ip address of this device */
    TMWTYPES_CHAR ipAddress[SDNPXML_MAXSTRINGLENGTH];

    /* subnet mask */
    TMWTYPES_CHAR subnetMask[SDNPXML_MAXSTRINGLENGTH];

    /* Gateway IP Address */
    TMWTYPES_CHAR gatewayIPAddress[SDNPXML_MAXSTRINGLENGTH];

    /* tcpConnectionEstablishement
     * choice, specify type to indicate which one
     */
    struct {
      SDNPDATA_XML_TCPCONNECTTYPE type;
      TMWTYPES_CHAR other[SDNPXML_MAXSTRINGLENGTH];
    } tcpConnectionEstablishment;

    /* Ip Address of Remote Device, multiple addresses allowed*/
    TMWTYPES_CHAR
ipAddressOfRemoteDevice[SDNPXML_MAX_IPADDRESSES][SDNPXML_MAXSTRINGLENGTH];

    /* TCP Listen Port */
    struct {
      /* Choice of one of these two
       * If notApplicable is TMWDEFS_TRUE, "<notApplicable /> will be output
       * otherwise the value specified will be output
       */
      TMWTYPES_BOOL notApplicable;
      TMWTYPES_ULONG value;
    } tcpListenPort;

    /* TCP Listen Port of remote device (when Dual End Point) */
    struct {
      /* Choice of one of these two
       * If notApplicable is TMWDEFS_TRUE, "<notApplicable /> will be output
       * otherwise the value specified will be output
       */
      TMWTYPES_BOOL notApplicable;
      TMWTYPES_ULONG value;
    } tcpPortOfRemoteDevice;

    /* SCL knows value of TCP Keep Alive Timer
     * tcpKeepAliveTimer;
     */

    /* Local UDP Port */
    struct {
      /* Choice of one of these two
       * If letSystemChoose is TMWDEFS_TRUE, "<letSystemChoose /> will be output
       * otherwise the value specified will be output
       */
      TMWTYPES_BOOL letSystemChoose;
      TMWTYPES_ULONG value;
    } localUDPPort;

    /* destinationUDPPort (Masters Only), not used, should be removed. */
    TMWTYPES_ULONG destinationUDPPort;

    /* Destination UDP port for unsolicited null responses */
    struct {
      /* Choice of one of these two
       * If none is TMWDEFS_TRUE, "<none /> will be output
       * otherwise the value specified will be output
       */
      TMWTYPES_BOOL none;
      TMWTYPES_ULONG value;
    } udpPortForUnsolicitedNullResponses;

    /* Destination UDP port for responses (if UDP only) */
    struct {
      /* Choice
       * Set type to the correct enum,
       * if type==SDNPDATA_XML_UDPVALUE, value will be output
       */
      SDNPDATA_XML_UDPRESPONSETYPE type;
      TMWTYPES_ULONG value;
    } udpPortForResponses;

    /* multipleOutstationConnections is a master only value */

    /* multipleMasterConnections */
    struct {
      /* If notSupported is TMWDEFS_TRUE only <notSupported /> will be output.
       * If notSupported is TMWDEFS_FALSE, 0 or more of the other three can be
       *  set to TMWDEFS_TRUE
       */
      TMWTYPES_BOOL notSupported;
      TMWTYPES_BOOL basedOnIPAddress;
      TMWTYPES_BOOL basedOnIPPortNumber;
      TMWTYPES_BOOL browsingForStaticData;
    } multipleMasterConnections;

    /* timeSynchronization */
    struct {
      /* Choice
       * Set type to the correct enum,
       * if type==SDNPDATA_XML_TIMESYNCOTHER, other string will be output
       */
      SDNPDATA_XML_TIMESYNCTYPE type;
      TMWTYPES_CHAR other[SDNPXML_MAXSTRINGLENGTH];
    } timeSynchronization;

  } SDNPDATA_XML_NETWORKCONFIG;

  /* Outstation Config */
  /* The following enumeration is used to specify what outstationConfig
   * timeSyncRequired value is specified
   */
  typedef enum {
    /* "never" */
    SDNPDATA_XML_TIMESYNCREQNEVER,
    /* "withinSecondsOfIIN14", value */
    SDNPDATA_XML_TIMESYNCREQWITHIN,
    /* "periodically", value */
    SDNPDATA_XML_TIMESYNCREQPERIOD
  } SDNPDATA_XML_TIMESYNCREQTYPE;

  /* The following enumeration is used to specify what outstationConfig
   * deviceTroubleBit value is specified
   */
  typedef enum {
    /* "neverUsed" */
    SDNPDATA_XML_DEVICETROUBLENEVER,
    /* "reasonForSetting", reasonForSetting null terminated string */
    SDNPDATA_XML_DEVICETROUBLEREASON,
  } SDNPDATA_XML_DEVICETROUBLE;

  /* The following enumeration is used to specify what outstationConfig
   * eventBufferOverflowBehavior value is specified
   */
  typedef enum {
    /* "discardOldest" */
    SDNPDATA_XML_EVENT_DISCOLD,
    /* "discardNewest" */
    SDNPDATA_XML_EVENT_DISCNEW,
    /* "other", null terminated other[] */
    SDNPDATA_XML_EVENT_OTHER,
  } SDNPDATA_XML_EVENTBUFOVTYPE;

  /* outstation config */
  typedef struct {

    /* applicationLayerConfirmTimeout, SCL will determine this */

    /* timeSyncRequired */
    struct{
      SDNPDATA_XML_TIMESYNCREQTYPE type;
      TMWTYPES_ULONG value;
    } timeSyncRequired;

    /* deviceTroubleBit*/
    struct {
      SDNPDATA_XML_DEVICETROUBLE type;
      /* null terminated string */
      TMWTYPES_CHAR reasonForSetting[SDNPXML_MAXSTRINGLENGTH];
    } deviceTroubleBit;

    /* fileHandleTimeout, SCL will determine this */

    /* eventBufferOverflowBehavior */
    struct {
      SDNPDATA_XML_EVENTBUFOVTYPE type;
      /* null terminated string */
      TMWTYPES_CHAR other[SDNPXML_MAXSTRINGLENGTH];
    } eventBufferOverflowBehavior;

    /* eventBufferOrganization
    * This changed from a string to a more complex type in schema 2.09
    * TMWTYPES_CHAR eventBufferOrganization[SDNPXML_MAXSTRINGLENGTH];
    * This is perObjectGroup in the SDNP SCL.
    */

    /* sendsMultiFragmentResponses, SCL will determine this */

    /* requestsLastFragmentConfirmation, SCL will determine this */

    /* settingsPreservedThroughDeviceReset
     * (in 2.08 schema this was renamed ...DeviceRestart, but we will
     * keep the old name for backward code compatibility)
     */
    struct {
      /* set all that apply to TMWDEFS_TRUE */
      TMWTYPES_BOOL assignClass;
      TMWTYPES_BOOL analogDeadbands;
      TMWTYPES_BOOL dataSetPrototypes;
      TMWTYPES_BOOL dataSetDescriptors;
      TMWTYPES_BOOL FC31ActivateConfiguration;
    } settingsPreservedThroughDeviceReset;

  }SDNPDATA_XML_OUTSTATIONCONFIG;

  /* Outstation Performance */
  /* The following enumeration is used to specify outstationPerformance types that
   * use schema type timingPerformanceType such as maxTimeBaseDrift,
   * referenceErrorViaDNP etc
   */
  typedef enum {
    /* "value" set value */
    SDNPDATA_XML_TIMINGPERFVALUE,
    /* "other", set null terminated string */
    SDNPDATA_XML_TIMINGPERFOTHER
  } SDNPDATA_XML_TIMINGPERFTYPE;

  /* This is used by a number of outstationPerformance parameters */
  typedef struct {
    SDNPDATA_XML_TIMINGPERFTYPE type;
    union {
      /* milliseconds, if type is SDNPDATA_XML_TIMINGPERFVALUE */
      TMWTYPES_ULONG value;
      /* null terminated string, if type is SDNPDATA_XML_TIMINGPERFOTHER */
      TMWTYPES_CHAR other[SDNPXML_MAXSTRINGLENGTH];
    }value;
  }SDNPDATA_TIMINGPERFORMANCETYPE;

  /* configuration oustationPerformance */
  typedef struct {

    /* Maximum Time Base Drift (milliseconds per minute) */
    SDNPDATA_TIMINGPERFORMANCETYPE maxTimeBaseDrift;

    /* When Does Outstation Set IIN 1.4?
     * This can be never, or 1 or more of the other 4 values
     * If never is TMWDEFS_TRUE it will be output
     * If never is TMWDEFS_FALSE
     *    If atStartup is TMWDEFS_TRUE it will be output
     *    If the ULONG values are nonzero they will be output.
     */
    struct {
      TMWTYPES_BOOL  never;
      TMWTYPES_BOOL  atStartup;
      TMWTYPES_ULONG periodically;
      TMWTYPES_ULONG afterLastTimeSync;
      TMWTYPES_ULONG whenTimeErrorExceeds;
    } outstationSetsIIN14;

    /* Maximum Internal Time Reference Error When et Via DNP (ms) */
    SDNPDATA_TIMINGPERFORMANCETYPE referenceErrorViaDNP;

    /* Maximum Delay Measurement (ms) */
    SDNPDATA_TIMINGPERFORMANCETYPE delayMeasurementError;

    /* Maximum Response Time in (ms) */
    SDNPDATA_TIMINGPERFORMANCETYPE responseTime;

    /* Maximum Time From Startup to IIN 1.4 Assertion (ms)  */
    SDNPDATA_TIMINGPERFORMANCETYPE startupToIIN14;

    /* Maximum Event Time-tag Error for Local Binary and Double Bit I/O (ms)  */
    SDNPDATA_TIMINGPERFORMANCETYPE binaryOrDoubleBitEventError;

    /* Maximum Event Time-tag Error for Local I/O Other Than Binary and Double Bit
     * Data Types (ms)
     */
    SDNPDATA_TIMINGPERFORMANCETYPE nonBinaryOrDoubleBitEventError;

  } SDNPDATA_XML_OUTSTATIONPERFORM;

  /* field config */
  typedef struct {
    /* These are all null terminated strings,
     * Empty strings, where first byte==0, will not be output
     */
    /* Location Name Or Code */
    TMWTYPES_CHAR outstationLocation[SDNPXML_MAXSTRINGLENGTH];
    /* Outstation Field ID Code/Number */
    TMWTYPES_CHAR outstationId[SDNPXML_MAXSTRINGLENGTH];
    /* Outstation Name */
    TMWTYPES_CHAR outstationName[SDNPXML_MAXSTRINGLENGTH];
    /* Device Serial Number */
    TMWTYPES_CHAR deviceSerialNumber[SDNPXML_MAXSTRINGLENGTH];

  } SDNPDATA_XML_FIELDCONFIG;

  /* Security Config */
  /* Number of other critical fragments */
```

```
#define SDNPXML_MAX_CRITICALFRAGMENTS 2

/* The following enumeration is used to specify current value
 * for securityConfig TLSCipherSuites
 */
typedef enum {
  /* "notApplicable" */
  SDNPDATA_XML_TLSNA,

  /* "TLSRSAEncryptedAES128" */
  SDNPDATA_XML_TLSRSARAES128,

  /* "TLSRSAEncryptedRC4" */
  SDNPDATA_XML_TLSRSARC4,

  /* "TLSRSAEncrypted3DES" */
  SDNPDATA_XML_TLSRSADES,

  /* "TLSDHSignedDSSEncrypted3DES" */
  SDNPDATA_XML_TLSCHDSSDES,

  /* "TLSDHSignedRSAEncrypted3DES" */
  SDNPDATA_XML_TLSDHRSADES,

  /* "TLSDHESignedDSSEncrypted3DES" */
  SDNPDATA_XML_TLSDHEDSSDES,

  /* "TLSDHESignedRSAEncrypted3DES" */
  SDNPDATA_XML_TLSDHERSADES,

  /* "TLSDHSignedDSSEncryptedAES128" */
  SDNPDATA_XML_TLSDSSAES128,

  /* "TLSDHSignedDSSEncryptedAES256" */
  SDNPDATA_XML_TLSDSSAES256,

  /* "TLSDHEncryptedAES128" */
  SDNPDATA_XML_TLSAES128,

  /* "TLSDHEncryptedAES256" */
  SDNPDATA_XML_TLSAES256,

  /* "other", set null terminated string */
  SDNPDATA_XML_TLSOTHER

} SDNPDATA_XML_TLSCIPHERTYPE;

/* The following enumeration is used to specify securityconfig
 * changeCipherRequestTimeout
 */
typedef enum {
  /* "notApplicable" */
  SDNPDATA_XML_SECURITYVALUENA,

  /* "value" set value */
  SDNPDATA_XML_SECURITYVALUEVALUE,

  /* "other", set null terminated string */
  SDNPDATA_XML_SECURITYVALUEOTHER
} SDNPDATA_XML_SECURITYVALUETYPE;

/* This structure is used for some security config */
typedef  struct {
  SDNPDATA_XML_SECURITYVALUETYPE type;
  union {
    TMWTYPES_ULONG value;
    /* null terminated string, if type is SDNPDATA_XML_SECURITYVALUEOTHER */
    TMWTYPES_CHAR other[SDNPXML_MAXSTRINGLENGTH];
  }value;
} SDNP_XML_SECURITYVALUE;

/* security config structure */
typedef struct {

  /* Cipher Suites used with DNP implementations using TLS */
  struct {
    SDNPDATA_XML_TLSCIPHERTYPE type;
    TMWTYPES_CHAR other[SDNPXML_MAXSTRINGLENGTH];
  } TLSCipherSuites;

  /* Change cipher request timeout */
```

```
    SDNP_XML_SECURITYVALUE changeCipherRequestTimeout;

    /* Number of Certificate Authorities supported */
    TMWTYPES_ULONG numberCASupported;

    /* Certificate Revocation check time */
    SDNP_XML_SECURITYVALUE certificateRevocationCheckTime;

    /* Additional critical function codes, set the desired ones to TMWDEFS_TRUE */
    struct {
      TMWTYPES_BOOL FC0;
      TMWTYPES_BOOL FC1;
      TMWTYPES_BOOL FC7;
      TMWTYPES_BOOL FC8;
      TMWTYPES_BOOL FC9;
      TMWTYPES_BOOL FC10;
      TMWTYPES_BOOL FC11;
      TMWTYPES_BOOL FC12;
      TMWTYPES_BOOL FC22;
      TMWTYPES_BOOL FC23;
      TMWTYPES_BOOL FC25;
      TMWTYPES_BOOL FC26;
      TMWTYPES_BOOL FC27;
      TMWTYPES_BOOL FC28;
      TMWTYPES_BOOL FC30;
      TMWTYPES_BOOL FC129;
      TMWTYPES_BOOL FC130;
    } additionalCriticalFCs;

    /* Remote Update Key Change,
     * These changed in schema 2.09 from a boolean to a choice of one of each.
     * The SCL allows whatever the crypto library allows so it could be multiple.
     * The Authority chooses which to actually use.
     */
    struct {
      /* DNPAUTH_KEYCH_SYMAES128_SHA1
       * DNPAUTH_KEYCH_SYMAES256_SHA256
       * DNPAUTH_KEYCH_SYMAES256_GMAC
       */
      TMWTYPES_UCHAR symmetricCrypto;

      /* DNPAUTH_KEYCH_ASYM_RSA1024_SHA1
       * DNPAUTH_KEYCH_ASYM_RSA2048_SHA256
       * DNPAUTH_KEYCH_ASYM_RSA3072_SHA256
       * DNPAUTH_KEYCH_ASYM_RSA2048_SHA256_GMAC
       * DNPAUTH_KEYCH_ASYM_RSA3072_SHA256_GMAC
       * DNPAUTH_KEYCH_ASYM_RSA1024_RSA_SHA1
       * DNPAUTH_KEYCH_ASYM_RSA2048_RSA_SHA256
       * DNPAUTH_KEYCH_ASYM_RSA3072_RSA_SHA256
       * DNPAUTH_KEYCH_ASYM_RSA2048_RSA_SHA256_GMAC
       * DNPAUTH_KEYCH_ASYM_RSA3072_RSA_SHA256_GMAC
       */
      TMWTYPES_UCHAR asymmetricCrypto;
    } remoteUpdateKeyChangeSupported;

    /* Default User Credentials are permitted to expire */
    TMWTYPES_BOOL permitUserCredentialExpiry;

    /* Other critical fragments
     * Null terminated strings.
     * Empty strings, ie criticalFragments[x]criticalFragment[0]==0, and note[0]==0,
     *  will not be output
     */
    struct {
      TMWTYPES_CHAR note[SDNPXML_MAXSTRINGLENGTH];
      TMWTYPES_CHAR criticalFragment[SDNPXML_MAXSTRINGLENGTH];
    } otherCriticalFragments[SDNPXML_MAX_CRITICALFRAGMENTS];

  } SDNPDATA_XML_SECURITYCONFIG;

  /* Binary Output Group */
  /* The following enumeration is used to specify database binary output group max
   * and minimum pulse values
   */
  typedef enum {
    /* fixed, set fixedPulseValue field */
    SDNPDATA_XML_PULSEVALUEFIXED,
    /* basedOnPointIndex */
    SDNPDATA_XML_PULSEVALUEPERPOINT
  } SDNPDATA_XML_PULSEVALUETYPE;
```

```
typedef  struct {
  SDNPDATA_XML_PULSEVALUETYPE type;
  /* set this if type SDNPDATA_XML_PULSEVALUEFIXED */
  TMWTYPES_ULONG fixedPulseValue;
} SDNP_XML_PULSEVALUE;

/* The following enumeration is used to specify if/when command events are generated
 */
typedef enum {
  /* "never" */
  SDNPDATA_XML_COMMANDEVENTNEVER,
  /* "onSuccess" */
  SDNPDATA_XML_COMMANDEVENTSUCCESS,
  /* "allControlAttempts" */
  SDNPDATA_XML_COMMANDEVENTALL
} SDNPDATA_XML_COMMANDEVENTTYPE;


typedef struct {
  /* Set type to FIXED, or set value to minimum */
  SDNP_XML_PULSEVALUE minimumPulseTime;

  /* Set type to FIXED, or set value to maximum */
  SDNP_XML_PULSEVALUE maximumPulseTime;

  /* Set this to enum to indicate if binary output command events are generated */
  SDNPDATA_XML_COMMANDEVENTTYPE commandEvents;

} SDNPDATA_XML_BINOUTGROUPCONFIG;


/* bit values for determining XML Device Profile supportedControlOperations
 * for each point.
 */
typedef TMWTYPES_UCHAR SDNPDATA_XML_SUP;
#define SDNPDATA_XML_SUP_SELOP             0x01
#define SDNPDATA_XML_SUP_DIROP             0x02
#define SDNPDATA_XML_SUP_DIRNOACK          0x04
#define SDNPDATA_XML_SUP_MORE_ONE          0x08
#define SDNPDATA_XML_SUP_SEL_CANCEL        0x01

/* The following enumeration is used to specify database Analog Input group
 * deadband assignment
 */
typedef enum {
  /* fixed */
  SDNPDATA_XML_DDBNDASSIGNFIXED,

  /* configurableViaDNP3 */
  SDNPDATA_XML_DDBNDASSIGNDNP3,

  /* configurableViaOtherMeans*/
  SDNPDATA_XML_DDBNDASSIGNOTHERMEANS,

  /* basedOnPointIndex*/
  SDNPDATA_XML_DDBNDASSIGNPERPOINT,

  /* other, fill in other string */
  SDNPDATA_XML_DDBNDASSIGNOTHER,
} SDNPDATA_XML_DDBNDASSIGNTYPE;

/* The following enumeration is used to specify database analog Input group
 * deadband algorithm
 */
typedef enum {
  /* simple */
  SDNPDATA_XML_DDBNDALGOSIMPLE,

  /* integrating */
  SDNPDATA_XML_DDBNDALGOINTEG,

  /* basedOnPointIndex */
  SDNPDATA_XML_DDBNDALGOPERPOINT,

  /* other, fill in other string */
  SDNPDATA_XML_DDBNDALGOOTHER,
} SDNPDATA_XML_DDBNDALGOTYPE;

/* Analog Input Group */
typedef struct {
  struct {
    SDNPDATA_XML_DDBNDASSIGNTYPE type;

    /* if type is SDNPDATA_XML_DDBNDASSIGNOTHER
     * set this null terminated string
     */
    TMWTYPES_CHAR other[SDNPXML_MAXSTRINGLENGTH];
  } analogDeadbandAssignments;

  struct {
    SDNPDATA_XML_DDBNDALGOTYPE type;

    /* if type is SDNPDATA_XML_DDBNDALGOOTHER
     * set this null terminated string
     */
    TMWTYPES_CHAR other[SDNPXML_MAXSTRINGLENGTH];
  } analogDeadbandAlgorithm;

} SDNPDATA_XML_ANLGINGROUPCONFIG;

/* The following enumeration is used to specify database counter rollover */
typedef enum {
  /* sixteenBits */
  SDNPDATA_XML_CNTRROLLOVER16,

  /* thirtyTwoBits */
  SDNPDATA_XML_CNTRROLLOVER32,

  /* value */
  SDNPDATA_XML_CNTRROLLOVERVALUE,

  /* other */
  SDNPDATA_XML_CNTRROLLOVEROTHER,

  /* basedOnPointIndex */
  SDNPDATA_XML_CNTRROLLOVERPERPOINT

} SDNPDATA_XML_CNTRROLLOVERTYPE;

/* The following enumeration is used to specify database counter freeze behavior */
typedef enum {
  /* masterRequest */
  SDNPDATA_XML_CNTRFROZENMASTERREQ,

  /* localFreezeWithoutTimeOfDay */
  SDNPDATA_XML_CNTRFROZENLOCALWOTIME,

  /* localFreezeRequiredTimeOfDay */
  SDNPDATA_XML_CNTRFROZENLOCALTIME,

  /* other */
  SDNPDATA_XML_CNTRFROZENOTHER

} SDNPDATA_XML_CNTRFROZENTYPE;

/* counter group config */
typedef struct {

  struct {
    SDNPDATA_XML_CNTRROLLOVERTYPE type;
    TMWTYPES_ULONG value;
    TMWTYPES_CHAR other[SDNPXML_MAXSTRINGLENGTH];
  } counterRollOver;

  struct {
    SDNPDATA_XML_CNTRFROZENTYPE type;
    TMWTYPES_CHAR other[SDNPXML_MAXSTRINGLENGTH];
  } countersFrozen ;

} SDNPDATA_XML_COUNTERGROUPCONFIG;

/* The following enumeration is used to specify database file transfer
 * authentication and append mode
*/
typedef enum {
  /* never */
  SDNPDATA_XML_FILENEVER,
  /* always */
  SDNPDATA_XML_FILEALWAYS,
  /* sometimes, set field sometimes with string to be output */
  SDNPDATA_XML_FILESOMETIMES
} SDNPDATA_XML_FILETYPE;
```

```
typedef struct {
  SDNPDATA_XML_FILETYPE type;
  /* null terminated string, if type SDNPDATA_XML_FILESOMETIMES */
  TMWTYPES_CHAR sometimes[SDNPXML_MAXSTRINGLENGTH];
} SDNPDATA_XML_FILECAPABILITY;

/* Structure to support database sequentialFileTransfer */
typedef struct {

  /* File Authentication */
  SDNPDATA_XML_FILECAPABILITY fileAuthentication;

  /* File Append Mode */
  SDNPDATA_XML_FILECAPABILITY fileAppendMode;

  /* Permissions Support */
  struct {
    TMWTYPES_BOOL ownerReadAllowed;
    TMWTYPES_BOOL ownerWriteAllowed;
    TMWTYPES_BOOL ownerExecuteAllowed;
    TMWTYPES_BOOL groupReadAllowed;
    TMWTYPES_BOOL groupWriteAllowed;
    TMWTYPES_BOOL groupExecuteAllowed;
    TMWTYPES_BOOL worldReadAllowed;
    TMWTYPES_BOOL worldWriteAllowed;
    TMWTYPES_BOOL worldExecuteAllowed;
  }permissionsSupport;

} SDNPDATA_XML_FILECONFIG;

/* database pointListDefinition used in all data type Group description */
typedef enum {
  /* fixed */
  SDNPDATA_XML_POINT_LISTFIXED,
  /* configurable */
  SDNPDATA_XML_POINT_LISTCONFIGURABLE,
  /* other, set other field */
  SDNPDATA_XML_POINT_LISTOTHER
} SDNPDATA_XML_POINTLISTTYPE;

typedef struct {
  /* specify fixed, configurable or other */
  SDNPDATA_XML_POINTLISTTYPE type;

  /* null terminated string, if type is SDNPDATA_XML_POINT_LISTOTHER */
  TMWTYPES_CHAR other[SDNPXML_MAXSTRINGLENGTH];
} SDNPDATA_XML_POINTLISTVALUE;

/* point list config structure for all data object groups */
typedef struct {
  SDNPDATA_XML_POINTLISTVALUE binaryInputGroup;
  SDNPDATA_XML_POINTLISTVALUE doubleBitInputGroup;
  SDNPDATA_XML_POINTLISTVALUE binaryOutputGroup;
  SDNPDATA_XML_POINTLISTVALUE counterGroup;
  SDNPDATA_XML_POINTLISTVALUE analogInputGroup;
  SDNPDATA_XML_POINTLISTVALUE analogOutputGroup;
  SDNPDATA_XML_POINTLISTVALUE sequentialFile;
  SDNPDATA_XML_POINTLISTVALUE prototypeDefinition;
  SDNPDATA_XML_POINTLISTVALUE datasetDefinition;
  SDNPDATA_XML_POINTLISTVALUE octetStringGroup;
  SDNPDATA_XML_POINTLISTVALUE virtualTermGroup;
} SDNPDATA_XML_POINTLISTCONFIG;


/* Per Binary Input point structure */
typedef struct {
  /* null terminated strings, empty string will not be output */
  TMWTYPES_CHAR   nameState0[SDNPXML_MAXSTRINGLENGTH];
  TMWTYPES_CHAR   nameState1[SDNPXML_MAXSTRINGLENGTH];

  /* null terminated name of point */
  TMWTYPES_CHAR   name[SDNPXML_MAXSTRINGLENGTH];
} SDNPDATA_XML_BININPOINTCONFIG;

/* Per Double Bit Input point structure */
typedef struct {
  /* null terminated strings, empty string will not be output */
  TMWTYPES_CHAR   nameState0[SDNPXML_MAXSTRINGLENGTH];
  TMWTYPES_CHAR   nameState1[SDNPXML_MAXSTRINGLENGTH];
  TMWTYPES_CHAR   nameState2[SDNPXML_MAXSTRINGLENGTH];
  TMWTYPES_CHAR   nameState3[SDNPXML_MAXSTRINGLENGTH];

  /* null terminated name of point */
  TMWTYPES_CHAR   name[SDNPXML_MAXSTRINGLENGTH];
} SDNPDATA_XML_DBLINPOINTCONFIG;

/* Per Binary Output point structure */
typedef struct {
  TMWTYPES_DOUBLE minimumPulseWidth;
  TMWTYPES_DOUBLE maximumPulseWidth;

  /* null terminated strings, empty string will not be output */
  TMWTYPES_CHAR   nameState0[SDNPXML_MAXSTRINGLENGTH];
  TMWTYPES_CHAR   nameState1[SDNPXML_MAXSTRINGLENGTH];

  /* The following two are part of dnpData */
  TMWTYPES_UCHAR  control;
  TMWTYPES_UCHAR  status;

  /* null terminated name of point */
  TMWTYPES_CHAR   name[SDNPXML_MAXSTRINGLENGTH];
} SDNPDATA_XML_BINOUTPOINTCONFIG;

/* Per Counter point structure */
typedef struct {

  TMWTYPES_ULONG counterRollOver;

  /* name of point */
  /* null terminated name of point */
  TMWTYPES_CHAR   name[SDNPXML_MAXSTRINGLENGTH];
} SDNPDATA_XML_CNTRPOINTCONFIG;

/* Per Analog Input point structure */
typedef struct {
  /* In Jan2010 schema, min and max TransmittedValue were changed to
   * minInt, minFloat, maxInt and maxFloat TransmittedValue
   * In April2016 schema it became a choice between integer or float.
   * set minInt and maxInt to the same value to use the float values.
   */
  TMWTYPES_LONG   minIntTransmittedValue;
  TMWTYPES_LONG   maxIntTransmittedValue;

  TMWTYPES_DOUBLE minFloatTransmittedValue;
  TMWTYPES_DOUBLE maxFloatTransmittedValue;

  TMWTYPES_DOUBLE scaleOffset;
  TMWTYPES_DOUBLE scaleFactor;
  TMWTYPES_DOUBLE resolution;

  /* null terminated string, empty string will not be output */
  TMWTYPES_CHAR   units[SDNPXML_MAXSTRINGLENGTH];

  /* null terminated name of point */
  TMWTYPES_CHAR   name[SDNPXML_MAXSTRINGLENGTH];
} SDNPDATA_XML_ANLGINPOINTCONFIG;

/* Per Analog Output point structure */
typedef struct {
  TMWTYPES_DOUBLE minTransmittedValue;
  TMWTYPES_DOUBLE maxTransmittedValue;

  TMWTYPES_DOUBLE scaleOffset;
  TMWTYPES_DOUBLE scaleFactor;
  TMWTYPES_DOUBLE resolution;

  /* null terminated string, empty string will not be output */
  TMWTYPES_CHAR   units[SDNPXML_MAXSTRINGLENGTH];

  /* null terminated name of point */
  TMWTYPES_CHAR   name[SDNPXML_MAXSTRINGLENGTH];
} SDNPDATA_XML_ANLGOUTPOINTCONFIG;

/* Octet String point structure */
typedef struct {
  TMWTYPES_CHAR name[SDNPXML_MAXSTRINGLENGTH];
} SDNPDATA_XML_STRINGPOINTCONFIG;

/* Virtual Terminal point structure */
typedef struct {
  TMWTYPES_CHAR name[SDNPXML_MAXSTRINGLENGTH];
```

```
} SDNPDATA_XML_VTERMPOINTCONFIG;


/* Per File "point" structure, the SCL will call sdnpdata_XmlGetPerFileConfig
 * to get this information till function returns TMWDEFS_FALSE
 */
typedef struct {

  /* Null terminated string.
   * If empty string, ie filename[0]==0 this entire file record will not be output
   */
  TMWTYPES_CHAR filename[SDNPXML_MAXSTRINGLENGTH];

  TMWDEFS_CLASS_MASK eventClass;
  TMWTYPES_BOOL readAuthenticateRequired;
  TMWTYPES_BOOL writeAuthenticateRequired;
  TMWTYPES_BOOL deleteAuthenticateRequired;

  /* Null terminated string.
   * Empty string will not be output
   */
  TMWTYPES_CHAR description[SDNPXML_MAXSTRINGLENGTH];

} SDNPDATA_XML_PERFILECONFIG;


#ifdef __cplusplus
extern "C" {
#endif

  /* function: sdnpdata_deviceProfileFile
   * purpose: Check to see if this filename is to be used for retrieving the device
   * profile in xml format if so, is this file authentication key OK?
   * If this returns TMWDEFS_TRUE, but authentication is not OK
   *   the SCL will send back a denied status to the master.
   * if this returns TMWDEFS_FALSE, then the normal sdnpdata_openFile will be called
   * If this file name is for retrieving the Device Profile from the database
itself,
   * ie not generated by the SCL at this time, then this should return TMWDEFS_FALSE
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  pFileName - file name to be checked
   *  authKey - file authentication key sent from master
   *  pAuthKey - return value indicating if this authentication key was valid
   * returns:
   *  TMWDEFS_TRUE if this is the name for the SCL to generate the Device Profile
   *   in XML format.
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_deviceProfileFile(
    void *pHandle,
    TMWTYPES_CHAR *pFileName,
    TMWTYPES_ULONG authKey,
    TMWTYPES_BOOL *pAuthKeyOK);

  /* function: sdnpdata_XmlGetDeviceConfig
   * purpose: Get the information needed to generate the XML for the
   *  configuration/deviceConfig portion for the Device Profile XML output
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  pConfig - pointer to the configuration structure to be filled in.
   * returns:
   *  TMWDEFS_TRUE if successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_XmlGetDeviceConfig(
    void *pHandle,
    SDNPDATA_XML_DEVICECONFIG *pConfig);

  /* function: sdnpdata_XmlGetSerialConfig
   * purpose: Get the information needed to generate the XML for the
   *  configuration/serialConfig portion for the Device Profile XML output
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  pConfig - pointer to the configuration structure to be filled in.
   * returns:
   *  TMWDEFS_TRUE if successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_XmlGetSerialConfig(
    void *pHandle,
    SDNPDATA_XML_SERIALCONFIG *pConfig);

  /* function: sdnpdata_XmlGetNetworkConfig
   * purpose: Get the information needed to generate the XML for the
   *  configuration/network portion for the Device Profile XML output
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  pConfig - pointer to the configuration structure to be filled in.
   * returns:
   *  TMWDEFS_TRUE if successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_XmlGetNetworkConfig(
    void *pHandle,
    SDNPDATA_XML_NETWORKCONFIG *pConfig);

  /* function: sdnpdata_XmlGetOutstationConfig
   * purpose: Get the information needed to generate the XML for the
   *  configuration/outstationConfig portion for the Device Profile XML output
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  pConfig - pointer to the configuration structure to be filled in.
   * returns:
   *  TMWDEFS_TRUE if successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_XmlGetOutstationConfig(
    void *pHandle,
    SDNPDATA_XML_OUTSTATIONCONFIG *pConfig);

  /* function: sdnpdata_XmlGetOutstationPerform
   * purpose: Get the information needed to generate the XML for the
   *  configuration/outstationPerformance portion for the Device Profile XML output
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  pConfig - pointer to the configuration structure to be filled in.
   * returns:
   *  TMWDEFS_TRUE if successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_XmlGetOutstationPerform(
    void *pHandle,
    SDNPDATA_XML_OUTSTATIONPERFORM *pConfig);

  /* function: sdnpdata_XmlGetFieldConfig
   * purpose: Get the information needed to generate the XML for the
   *  configuration/fieldConfig portion for the Device Profile XML output
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  pConfig - pointer to the configuration structure to be filled in.
   * returns:
   *  TMWDEFS_TRUE if successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_XmlGetFieldConfig(
    void *pHandle,
    SDNPDATA_XML_FIELDCONFIG *pConfig);

  /* function: sdnpdata_XmlGetSecurityConfig
   * purpose: Get the information needed to generate the XML for the
   *  configuration/securityConfig portion for the Device Profile XML output
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  pConfig - pointer to the configuration structure to be filled in.
   * returns:
   *  TMWDEFS_TRUE if successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_XmlGetSecurityConfig(
    void *pHandle,
    SDNPDATA_XML_SECURITYCONFIG *pConfig);

  /* function: sdnpdata_XmlGetBinOutGroupConfig
   * purpose: Get the information needed to generate the XML for the
   *  database binaryOutputGroup portion of the Device Profile XML output
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  pConfig - pointer to the configuration structure to be filled in.
   * returns:
   *  TMWDEFS_TRUE if successful
   *  TMWDEFS_FALSE otherwise
   */
 TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_XmlGetBinOutGroupConfig(
   void *pHandle,
   SDNPDATA_XML_BINOUTGROUPCONFIG *pConfig);

  /* function: sdnpdata_XmlGetCntrGroupConfig
   * purpose: Get the information needed to generate the XML for the
   *  database counterGroup portion of the Device Profile XML output
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  pConfig - pointer to the configuration structure to be filled in.
   * returns:
   *  TMWDEFS_TRUE if successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_XmlGetCntrGroupConfig(
   void *pHandle,
   SDNPDATA_XML_COUNTERGROUPCONFIG *pConfig);

  /* function: sdnpdata_XmlGetAnlgInGroupConfig
   * purpose: Get the information needed to generate the XML for the
   *  database Analog Input Group portion of the Device Profile XML output
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  pConfig - pointer to the configuration structure to be filled in.
   * returns:
   *  TMWDEFS_TRUE if successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_XmlGetAnlgInGroupConfig(
   void *pHandle,
   SDNPDATA_XML_ANLGINGROUPCONFIG *pConfig);

  /* function: sdnpdata_XmlGetFileConfig
   * purpose: Get the information needed to generate the XML for the
   *  database sequentialFileTransfer portion of the Device Profile XML output
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  pConfig - pointer to the configuration structure to be filled in.
   * returns:
   *  TMWDEFS_TRUE if successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_XmlGetFileConfig(
   void *pHandle,
   SDNPDATA_XML_FILECONFIG *pConfig);

  /* function: sdnpdata_XmlGetPointListDefinition
   * purpose: Get the information needed to generate the XML for the
   *  pointListDefinition portion for each object group of the Device Profile
   *  XML output
   * arguments:
   *  pHandle - handle to database returned from sdnpdata_init
   *  pConfig - pointer to the configuration structure to be filled in.
   * returns:
   *  TMWDEFS_TRUE if successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_XmlGetPointListDefinition(
   void *pHandle,
   SDNPDATA_XML_POINTLISTCONFIG *pConfig);

  /* function: sdnpdata_XmlGetBinInPointConfig
   * purpose: Get the information needed to generate the XML for the
   *  database sequentialFileTransfer portion of the Device Profile XML output
   * arguments:
   *  pPoint - handle returned from sdnpdata_binInGetPoint function
   *  pConfig - pointer to the configuration structure to be filled in.
   * returns:
   *  TMWDEFS_TRUE if successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_XmlGetBinInPointConfig(
   void *pPoint,
   SDNPDATA_XML_BININPOINTCONFIG *pConfig);

  /* function: sdnpdata_XmlGetDblInPointConfig
   * purpose: Get the information needed to generate the XML for the
   *  database sequentialFileTransfer portion of the Device Profile XML output
   * arguments:
   *  pPoint - handle returned from sdnpdata_dblInGetPoint function
   *  pConfig - pointer to the configuration structure to be filled in.
   * returns:
   *  TMWDEFS_TRUE if successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_XmlGetDblInPointConfig(
    void *pPoint,
    SDNPDATA_XML_DBLINPOINTCONFIG *pConfig);

  /* function: sdnpdata_XmlGetBinOutPointConfig
   * purpose: Get the information needed to generate the XML for the
   *  database sequentialFileTransfer portion of the Device Profile XML output
   * arguments:
   *  pPoint - handle returned from sdnpdata_binOutGetPoint function
   *  pConfig - pointer to the configuration structure to be filled in.
   * returns:
   *  TMWDEFS_TRUE if successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_XmlGetBinOutPointConfig(
    void *pPoint,
    SDNPDATA_XML_BINOUTPOINTCONFIG *pConfig);

  /* function: sdnpdata_XmlGetCntrPointConfig
   * purpose: Get the information needed to generate the XML for the
   *  database sequentialFileTransfer portion of the Device Profile XML output
   * arguments:
   *  pPoint - handle returned from sdnpdata_binCntrGetPoint function
   *  pConfig - pointer to the configuration structure to be filled in.
   * returns:
   *  TMWDEFS_TRUE if successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_XmlGetCntrPointConfig(
    void *pPoint,
    SDNPDATA_XML_CNTRPOINTCONFIG *pConfig);

  /* function: sdnpdata_XmlGetAnlgInPointConfig
   * purpose: Get the information needed to generate the XML for the
   *  database sequentialFileTransfer portion of the Device Profile XML output
   * arguments:
   *  pPoint - handle returned from sdnpdata_anlgInGetPoint function
   *  pConfig - pointer to the configuration structure to be filled in.
   * returns:
   *  TMWDEFS_TRUE if successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_XmlGetAnlgInPointConfig(
    void *pPoint,
    SDNPDATA_XML_ANLGINPOINTCONFIG *pConfig);

  /* function: sdnpdata_XmlGetAnlgOutPointConfig
   * purpose: Get the information needed to generate the XML for the
   *  database sequentialFileTransfer portion of the Device Profile XML output
   * arguments:
   *  pPoint - handle returned from sdnpdata_anlgOutGetPoint function
   *  pConfig - pointer to the configuration structure to be filled in.
   * returns:
   *  TMWDEFS_TRUE if successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_XmlGetAnlgOutPointConfig(
    void *pPoint,
    SDNPDATA_XML_ANLGOUTPOINTCONFIG *pConfig);

  /* function: sdnpdata_XmlGetStringPointConfig
   * purpose: Get the information needed to generate the XML for the
   *  database sequentialFileTransfer portion of the Device Profile XML output
   * arguments:
   *  pPoint - handle returned from sdnpdata_strGetPoint function
   *  pConfig - pointer to the configuration structure to be filled in.
   * returns:
   *  TMWDEFS_TRUE if successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_XmlGetStringPointConfig(
    void *pPoint,
    SDNPDATA_XML_STRINGPOINTCONFIG *pConfig);

  /* function: sdnpdata_XmlGetVtermPointConfig
   * purpose: Get the information needed to generate the XML for the
```

```
   *  database sequentialFileTransfer portion of the Device Profile XML output
   * arguments:
   *  pPoint - handle returned from sdnpdata_vtermGetPoint function
   *  pConfig - pointer to the configuration structure to be filled in.
   * returns:
   *  TMWDEFS_TRUE if successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_XmlGetVtermPointConfig(
    void *pPoint,
    SDNPDATA_XML_VTERMPOINTCONFIG *pConfig);

  /* function: sdnpdata_XmlGetPerFileConfig
   * purpose: Get the information needed to generate the XML for the
   *  database sequentialFileTransfer portion of the Device Profile XML output
   * arguments:
   *  pPoint - handle returned sdnpdata_init function
   *  index - index 0 to n-1 for all files in database
   *  pConfig - pointer to the configuration structure to be filled in.
   * returns:
   *  TMWDEFS_TRUE if successful
   *  TMWDEFS_FALSE otherwise
   */
  TMWTYPES_BOOL TMWDEFS_GLOBAL sdnpdata_XmlGetPerFileConfig(
    void *pHandle,
    TMWTYPES_ULONG index,
    SDNPDATA_XML_PERFILECONFIG *pConfig);

  /* function: sdnpdata_binOutGetSupCtrl
   * purpose: Determine what device profile control operations are supported
   *  for this particular point.
   *  NOTE: this is used only by sdnpxml/sdnpxml2.c if SDNPDATA_SUPPORT_XMLxx is
TRUE
   *  code to generate the configuration file or Device Profile for this device.
   * arguments:
   *  pPoint - handle to data point returned from sdnpdata_binOutGetPoint function.
   * returns:
   *  bitmask indicating what operations are supported.
   *   SDNPDATA_XML_SUP_SELOP
   *   SDNPDATA_XML_SUP_DIROP
   *   SDNPDATA_XML_SUP_DIRNOACK
   *   SDNPDATA_XML_SUP_MORE_ONE
   *   SDNPDATA_XML_SUP_SEL_CANCEL
   */
  SDNPDATA_XML_SUP TMWDEFS_GLOBAL sdnpdata_binOutGetSupCtrl(
    void *pPoint);

  /* function: sdnpdata_anlgOutGetSupCtrl
   * purpose: Determine what device profile control operations are supported
   *  for this particular point.
   *  NOTE: this is used only by sdnpxml/sdnpxml2.c if SDNPDATA_SUPPORT_XMLxx is
TRUE
   *  code to generate the configuration file or Device Profile for this device.
   * arguments:
   *  pPoint - handle to data point returned from sdnpdata_anlgOutGetPoint function.
   * returns:
   *  bitmask indicating what operations are supported.
   *   SDNPDATA_XML_SUP_SELOP
   *   SDNPDATA_XML_SUP_DIROP
   *   SDNPDATA_XML_SUP_DIRNOACK
   */
  SDNPDATA_XML_SUP TMWDEFS_GLOBAL sdnpdata_anlgOutGetSupCtrl(
    void *pPoint);

#ifdef __cplusplus
}
#endif

#endif /* SDNPDATA_SUPPORT_XML2 */

#endif /* SDNPDATA_DEFINED */
```

7 Advanced Topics

7.1 DNP3 Device Profile in XML

DNP3 Specification Volume 8 Interoperability describes the content of a Device Profile Document that should be provided in XML format. A default Device Profile for the SDNP Source Code library is provided in the doc directory.

Triangle MicroWorks also provides a free Windows application that can be used to create, or edit DNP3 Device Profiles. This tool also provides useful information about how to configure the Source Code Library to match a device profile configuration. Click here to download DNP3 Forge.

The SDNP Source Code library can also be used to generate the current configuration portion of the Device Profile document. This can be generated in response to a file transfer read from the master or generated by the slave/outstation application directly.

When a file open request is received from the master and DNPDATA_SUPPORT_XML2 is set to TMWDEFS_TRUE, sdnpdata_deviceProfileFile will be called to determine if the specified file name is the one used for automatic generation of the standard device profile. If this function returns TMWDEFS_TRUE the Source Code Library will attempt to generate the current value portion of the device profile by calling a sequence of sdnpdata_xmlGetxxx functions to retrieve the portion of the device profile not already known by the Source Code Library. This data will be returned to the master using file transfer.

An application on the slave can also generate the current configuration portion and save it to a file by calling sdnpxml2_saveDatabaseToFile(..), see sdnpxml2.h for details. This function will use the sdnpdata_openFile and sdnpdata_writeFile functions. If a different mechanism is required, sdnpxml2_saveDatabaseToFile can serve as an example of how to implement your own device profile generation and save functionality.

7.2 DNP3 Device Profile as a Binary Configuration File

While the SDNP 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 an outstation device during its start up or operation. A DNP3 Binary Configuration file contains only the current values that could be used by the outstation to perform configuration tasks. The SDNP SCL contains a method that will read values from a DNP3 Binary Configuration file and apply the values to setting in the SCL. 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.

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 the 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 sdnpsesn_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, link, session, 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 and session.

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(…).

7.3 DNP3 Event Timestamps and Order of Occurrence

7.3.1 Order of Occurrence vs Timestamp

TB2018-001 Time Management, Common Time Of Occurrence and Event Ordering, DNP3 IED Certification Procedure Version 3.0 released in December 2020 and the resulting new version of IEEE 1815-TBD (2022?) make it clear that events are to be queued in Order of Occurrence independent of timestamp. Order should be determined by the when the change was observed and added to the event queue. Previously, IEEE 1815-2012 and other specifications indicated events were to be sorted by timestamp.

To provide compatibility with the previous time sorting behavior, a session configuration parameter sortEventsByTime has been added in SCL version 3.29.000. By default, this value is set to TMWDEFS_FALSE providing the required new functionality. It is not recommended to set this to TMWDEFS_TRUE but IS provided in case it is required for interoperability with certain DNP3 Masters.

TB2018-001 also indicates proper behavior if an Outstation does not support a clock and therefore timestamps are not available for events. It has always been possible to configure out support for objects that do not support timestamps. However, if this behavior is needed at runtime because sometimes a clock is not available, a new session configuration parameter supportEventsWithTime has been added to provide proper behavior as tested in the new IED Certification Procedure.

7.3.2 Binary Input and Double-bit Binary Input Events

A response containing both Binary Input Events (Obj2) and Double Bit Binary Input (Obj4) Events are required to be intermixed in the order they occur. Previously, the library maintained separate event queues for each type and ordered the events in time order in a response. To better support events that are not ordered by time including those without a timestamp, these two Binary Event objects have been combined into a single event queue. There is no change in configuration or external behavior with the move to a single queue.

7.3.3 User Managed Events

Previously, if User Managed Events support was compiled in to the SDNP SCL, Obj2 and Obj4 Events in a single response were sent sequentially and not properly intermixed in a response. Changes in SDNP SCL 3.29.000 allow these events to now be sorted properly intermixed. These changes include an additional optional field in the existing SDNPDATA_GET_EVENT interface structure. For existing customers, if no changes are made to the already implemented sdnpdata_umEventxxx functions, the behavior will remain unchanged. The new sdnpsim.c example implementation shows how to use the new group field to return obj2 and obj4 events in intermixed order when sdnpdata_umEventGet(group=2) is called. Internally, sdnpsim.c now has a single queue for both types of Binary events. A single queue is not required in your implementation as long as the events are returned across the interface in the proper order.

7.4 File Transfer

File Transfer in DNP can be used to provide Read, Write, GetInfo, and Delete functionality over the DNP Protocol if enabled. File Transfer functionality is compiled out of the SDNP C and .NET SCLs. It can be compiled in by setting #define SDNPDATA_SUPPORT_OBJ70 TMWDEFS_TRUE.

When compiled in File Transfer can be enabled or disabled at run time. In SDNP Session configuration structure SDNPSESN_CONFIG fileTransferTimeout can be set to TMWDEFS_TRUE or TMWDEFS_FALSE as desired.

```
/* If set to TMWDEFS_FALSE, reject all file transfer requests
 * even if file transfer code is compiled in.
 */
TMWTYPES_BOOL fileTransferEnabled;
```

The database functions sdnpdata_openFile, sdnpdata_getFileInfo, and sdnpdata_deleteFile also show examples of how to restrict access particular file name or directory paths.

In SDNP .NET there is a corresponding SDNPSession property FileTransferEnabled and also a Database Event AllowFileAccessEvent that can be registered to restrict file access to file or directory paths. You can look in the SDNP Sample code delivered with the .NET SCL to see examples.

7.5 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 SDNP SCL provides the interface function sdnpo088_addEvent to add Data Set Snapshot events. The sdnpdata database interface requires sdnpdata_datasetProtoxxx, sdnpdata_datasetDescrxxx and sdnpdata_datasetxxx functions to be implemented to read and write Data Set information from/to the database.

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 slave Data Set functionality can be found in sdnpsim.c. As always, the code in sdnpdata.c should be replaced with your own database implementation.

7.6 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 change 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 Authentication Version 5 (SAv5) was released in November 2011. This version of Secure Authentication will be 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.

7.6.1 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.

7.6.2 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
```

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 sdnpdata.c must be implemented. A description of these functions, named sdnpdata_authxxx(), is contained in the file sdnpdata.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 sdnpdata interface. If both SAv2 and SAv5 are required, both cryptographic function interfaces sdnpdata and tmwcrypto must be implemented. While this common crypto interface is not required for an SAv2 implementation, there is example code in sdnpdata.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 SDNPSESN_CONFIG structure should be set to TMWDEFS_TRUE, and the other authentication configuration fields in the SDNPSESN_AUTH_CONFIG structure should be set to the desired values. These configuration parameters are described in sdnpsesn.h

Example code for configuring and using Secure Authentication has been added to the sample application DNPSlave.cpp. This code is surrounded by #if SDNPDATA_SUPPORT_OBJ120 #endif. In order for Secure Authentication to function properly in these examples, the functions sdnpdata_authxxx() and (for SAv5) tmwcrypto_xxx() must be implemented. For information about these functions please read the next two sections of this manual.

7.6.3 sdnpdata_authxxx() implementation

Detailed information about the specific interface for each function is provided in sdnpdata.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 any 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 sdnpdata_authxxx() functions for *cryptography* related functions. Both SAv2 and SAv5 will call sdndata_authxxx() functions for *database* related functions that are used to support Secure Authentication. There is sample code in sdnpdata.c that maps the SAv2 database crypto functions to the new common crypto interface.

sdnpdata_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 sdnpdata_authLogxxx functions will be called by the SCL for all significant authentication events.

sdnpdata_authIsCriticalReq() - Determines if the received request should be considered a Critical Request that should be challenged. The Secure Authentication specification lists function codes that are considered critical. The list of Function Codes that are considered Mandatory Critical in SAv5 is used in the default implementation of this function.

sdnpdata_authRequestAllowed() - Determines if the received and authenticated critical request should be allowed for this particular user. If SAv5 a role can be sent by the Master to the Outstation, it is up to this function to enforce that role or set of permissions.

7.6.3.1 SAv2 Database Implementation

sdnpdata_authDecryptKeyWrapData() - Decrypt the key wrap data sent from the master to the outstation using the specified algorithm. Currently 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 on the master 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 sdnpdata.c

sdnpdata_authHMACValue() - Generates a Keyed-Hash Message Authentication Code (HMAC) value on the challenged message using the specified algorithm and session key. There are currently 2 algorithms (SHA1 and SHA256) and 2 data lengths for each algorithm specified.

sdnpdata_authRandomChallengeData() - Generates pseudo random data to be included in authentication messages. It should use the algorithm specified in the FIPS 186-2 Digital Signal Standard.

7.6.3.2 SAv5 Database Implementation

Mandatory Statistics functions

sdnpdata_authSecStatThreshold() - Retrieve the security statistic threshold value from the database.

sdnpdata_authSecStatSet() – Store a security statistic that has been generated on this outstation device.

sdnpdata_authSecStatRead() – Return a current security statistic value.

sdnpdata_authSecStatxxx() - Other security statistic database functions that should be implemented. They are described in sdnpdata.h

Optional Remote Key Update functions (`DNPCNFG_SUPPORT_AUTHKEYUPDATE)`

sdnpdata_authKeyChgMethodSupport() – Determine if this key change method for sending User Update Keys to the outstation is supported. There are mandatory symmetric and optional asymmetric or public key methods.

sdnpdata_authGetOSName () – Get the name of this outstation. This must be preconfigured to match on both the master and the outstation.

sdnpdata_authUserStatusChgRcvd() - Store the data received in the g120v10 User Status Change request that was received from the master as part of the remote key update procedure. This includes User Name, role and expiry time in days. Role and expiry time must be enforced by your application. sdnpdata_authRequestAllowed() should use role to decide if a particular request is allowed. When the User Status information expires, your application should invalidate that User including calling sdnpsesn_removeAuthUser().

sdnpdata_authUpdateKeyChgReq () - Process the Update Key Change Request data from the g120v11 message, determining what user number will be used for this globally unique User Name.

sdnpdata_authGetUserName() - Get the globally unique User Name for the specified user number that was stored when a key status change request was received from the master.

7.6.4 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.

tmwcrypto_getKeyData()- Master only Get 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.

7.6.4.1 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 is 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.

7.6.5 Secure Authentication Error Events

The DNP3 Secure Authentication specification indicates that error messages may be sent as events on DNP associations other than the one on which the authentication error occurred. When a Secure Authentication error occurs the SCL will call sdnpdata_authLogErrorTx(). If you support other sdnp sessions in your device, calling sdnpo120_addEvent() on those sessions will generate an ObjectGroup 120 Variation 7 authentication error event.

7.6.6 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 and 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 hash 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 hash 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. Specification says 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.

7.6.7 Update Key Management

The same Update Keys that are used for sending session keys from the Master to the Outstation must be configured 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 are used in the decryption of the session key data sent from the master to the outstation. These Update Keys will be used by the sdnpdata_authDecryptKeyWrapData()(for SAv2) or tmwcrypto_decryptData() functions. Depending on the target environment access to these Update Keys should be protected by encryption, password protection or other means as deemed necessary.

7.6.8 Building Secure Authentication in DNPSlave.cpp Sample

The sample application DNPSlave.cpp contains code for configuring and using Secure Authentication. This code is surrounded by #if SDNPDATA_SUPPORT_OBJ120 #endif which by default is set to false. To compile in Secure Authentication in the SDNP 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 sdnpdata_authxxx() and tmwcrypto_xxx() which by default return failure, 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 DNPSlave.cpp
Set myUseAuthentication = true;
And set myUseSAv2 = true if you want to use SAv2 instead of SAv5

Build the DNPSlave application

If the application builds successfully try to run the DNPSlave application

If you get the following errors in the Windows Command Prompt window. This indicates that the cryptography testing failed for the asymmetric tests. This normally indicates that DNPSlave.exe cannot find the key files TMWTestxxxKey.pem. These files are installed in the DNPSlave 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 ****
etc
```

7.6.9 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 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.

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
```

7.6.10 Debugging

The Triangle MicroWorks Protocol Test Harness is a good way to test your implementation of Secure Authentication on a DNP3 Outstation (slave). A master session can be opened with Secure Authentication enabled. By configuring the same 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 mdnp 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. Communications with the Test Harness can be used to verify that the key unwrap, decryption and hashing functions are properly implemented on the Outstation.

Most of the Test Harness MDNP Request Tcl commands allow Aggressive Mode to be specified and to specify which Authentication User number is to be used for Aggressive Mode or in response to an authentication challenge by the outstation.
For example :

```
  mdnpbincmd authUserNumber 20
  mdnpbincmd authAggressiveMode true authUserNumber 300
```

In the Test Harness Tcl command window the command mdnpauthentication can be used for some Secure Authentication testing. You can specify that the next response received is to be considered critical and should be challenged. You can force a key exchange sequence for a particular user number. You can also force a Secure Authentication Error Message to be sent when the next response is received on the Test Harness master. Enter mdnpauthentication in the Test Harness Tcl command window for more details.

If you have licensed Secure Authentication Version 5 (SAv5) in the Test Harness and SDNP SCL you can also use commands to simulate an Authority and a DNP Master Adding, Modifying, and Deleting Secure Authentication Users, their User Update Keys and roles (permissions) on the Outstation. See TCL commands mdnpauthuserstatuschange, mdnpauthuserupdatekey, and mdnpauthuser or the Test Harness manual for more details.

7.7 DNP3 DER AN2018-001 Example

DNP Application Note AN2018-001defines a DNP3 Profile for Communications with Distributed Energy Resources (DERs). The document describes a standard data point configuration, set of protocol services and settings for communicating with DERs using DNP3.

The Electric Power Research Institute Inc. (EPRI) has implemented an open source reference implementation of this Application Note. EPRI has published a description of the reference implementation in the document "Open Source DER Outstation for DNP Application Note AN2018-001". EPRI's open source reference implementation is available from the EPRI github repository (https://github.com/epri-dev/der-dnp3-an2018).

The SDNP DER example integrates the EPRI open source reference implementation with the Triangle MicroWorks DNP Slave library. This integration eliminates the following dependency from the EPRI open source:

- OpenDNP3
- Boost

A modified version of the EPRI open source used in this example is delivered in the der-an-2018 directory. This code is subject to EPRI's licensing terms described in the LICENSE file.

The interface to the DER outstation is defined in the file IDERCommandHandlerCallback.h. The EPRI example code provides an implementation of these functions in DERCommnadHandlerCallbackDefault.cpp which simply prints messages in each of the callbacks.

The interface to the DER Database has been modified for the Triangle MicroWorks example. The API is defined in DERDatabase.h and is implemented with the following methods:

- DERDatabaseBinInUpdate(bool value, uint8_t quality, uint16_t index);
- DERDatabaseBinOutUpdate(bool value, uint8_t quality, uint16_t index);
- DERDatabaseAnalogInUpdate(int32_t value, uint8_t quality, uint16_t index);
- DERDatabaseAnalogOutUpdate(int32_t value, uint8_t quality, uint16_t index);

The DER Database is initialized with the values contained in the EPRI provided CSV files. These files are delivered in the der-an-2018/Inputs directory.

The EPRI document, “Open Source DER Outstation for DNP Application Note AN2018-001” describes the curve and schedule functionality.