Edge C SDK Developer's Guide

Version 2.2.1

February 2019

PTC Inc., 140 Kendrick Street, Needham, MA 02494 USA

Contents

Document Revision History

The following table briefly describes changes to this document for each release in which it was updated. For all interim releases (for example, 2.0.5, 2.1.1), please refer to the release notes for the changes in those releases.

Revision Date	Version	Description of Change
February 2019	2.2.1	Changed version of OpenSSL to 1.0.2q for the NON-FIPS distributions of this SDK. This release includes non-FIPS OpenSSL 32– and 64–bit libraries that on Windows is based on Visual Studio 2015 runtime library. The FIPS versions of this SDK are still based on Visual Studio 2012. Also removed mention of AxTLS since it is no longer provided in the distribution bundle.
December 2018	2.2.0	New callbacks to secure Application Keys, digests, passwords for opening certificates, and passphrases for opening keystore files. For details, see the information about the callback function for Application keys in Initialize API Singleton on page 57, the section of the topic on initializing the TunnelManager called Passwords (C SDK 2.2.0 and later) on page 64.
September 2018	2.1.3	Update for addition of support for *LastUpdated* optional parameter for

Revision Date	Version	Description of Change
		GetPropertySubscriptions in the **SubscribedPropsMgr** (Subscribed Properties Manager). The C SDK now supports the Software Content Management (SCM) Edge Extension. See the ThingWorx SCM Edge Extension for the *ThingWorx Edge C SDK Developer's Guide* for details about this extended capability for the C SDK.
April 2018	2.1.2	Added information about setting up a custom set of cipher suites to use with OpenSSL and the C SDK and a note about additional logging (DEBUG builds only) for staging directory failures when transferring files.
January 2018	2.1.0	Added the new feature of compression for all WebSocket communications, including file transfers.
September 2017	2.0.0	Added a new chapter on Edge Extensions and organized all the security information into a separate chapter. The OpenSSL v.1.0.2k is now provided with the C SDK. Revised topics for that. Added the new feature where the C SDK prints

Revision Date	Version	Description of Change
		version information on startup.
June 2017	1.5.2	Minor edits to fix typos.
May 2017	1.5.1	Added a tip about using `twDict` with the new `Foreach` iterator to process entries in a list or dictionary. instead of `twList`. AxTLS v.2.1.2 unable to connect to to SSL Tomcat servers that do not specify `pathLenConstraint` in their root certificate (which include PTC Cloud Services production instances of ThingWorx). Added Caution note to the section, Using SSL/TLS for Security on page 72.
May 2017	1.5.0	Updated information about file transfer timeouts and a new deadband push type. See the section, Defining Properties on page 41, for information about the new push type.
April 2017	1.4.1	Added note about tick resolution ()Initializing the Tunnel Manager (Optional) on page 62 and the section, Running the C SDK on Windows-based Operating Systems on page 70.

Revision Date	Version	Description of Change
March 2017	1.4.0	Updated the version of AxTLS to 2.1.2. Replaced the existing build information with information about building with CMake. Added section about Synchronized State Handler and updated the section about the Subscribed Properties. Updated the section on InfoTables. Added note about avoiding a possible deadlock in Linux by NOT calling `twList_Remove` from within a `foreach` handler. Added a subsection about lost messages to the section, Handling Offline Messages on page 36.
December 2016	1.3.4	Added Caution about using `twApi_BindThingWithoutDefaultServices`. See the section, Binding Your Entities on page 60.
December 2016	1.3.3	Added information about bulk binding to the section Binding Your Entities on page 60, support for `libcfu`, the `ForEach` iterator for the `tw_List` API, use of `twMap` instead of `twList` to improve performance,

Revision Date	Version	Description of Change
		andd the addition of a list-backed `twMap` and `twDict`.
June 2016	1.3.2	Added workaround for Known Issue for Apache Tomcat 8.0.35 and ciphers that were disabled in that release.
January 2016	1.3.1	Release fixed issues. Minor documentation edits, no major changes.
October 2015	1.3.0	Changed file name `list.h` to `twList.h` and added notes about the change to the default value for the socket read timeout in `twDefaultSettings.h`. Added the order of preference for proxy authentication types.
March 2015	1.2.0	Initial version of this guide. Included new features for releases 1.1.0 and 1.1.1 in addition to 1.1.2. See the release notes for details.

About this Guide

ThingWorx offers Software Development Kits (SDKs) for Edge devices, machines, and systems in several programming languages. These SDKs allow companies to incorporate connectivity functionality into their products, and to easily connect those products to an instance of ThingWorx platform. These SDKs can either be implemented as a gateway to several connected products, or be embedded directly within a product on a one-to-one basis.

All ThingWorx Edge SDKs share a common reference implementation and provide a secure communication channel to an instance of ThingWorx platform, allowing a machine/device to be a full participant in a ThingWorx IoT solution.

This document describes how to use the ThingWorx Edge C SDK. The complete API reference (javadoc) is available in the C SDK bundle.

Pre-requisites

This document assumes that you have a solid background in the C/C++ programming language. Further, it assumes that you have had at least basic training in ThingWorx. For example, you know how to use the ThingWorx Composer and understand the main concepts of things, data shapes, properties, events, and services.

To develop an application using the C SDK, you need to have a C/C++ development environment. No specific compiler version is required, but the compiler must be C89 (the C language spec) compatible. You can use CMake, version 2.6.1 or later to build projects or make files, which then are used to build the applications that you develop with the C SDK.

To get started, it is recommended that you review the sample projects provided in the SDK. To use these examples, you can use any IDE to build them with CMake. For information about the versions of Visual Studio that you can use with the C

SDK, see the *ThingWorx Edge and Connection Services Compatibility Reference*, which is available on the ThingWorx Edge Reference documents page of the PTC Support site.

Technical Support

Contact PTC Technical Support via the PTC Web site, phone, fax, or e-mail if you encounter problems using your product or the product documentation.

For complete details, refer to Contacting Technical Support in the *PTC Customer Service Guide*. This guide can be found under the Related Resources section of the PTC Web site at:

http://www.ptc.com/support/

The PTC Web site also provides a search facility for technical documentation of particular interest. To access this search facility, use the URL above and search the knowledge base.

You must have a Service Contract Number (SCN) before you can receive technical support. If you do not have an SCN, contact PTC Maintenance Department using the instructions found in your *PTC Customer Service Guide* under Contacting Your Maintenance Support Representative.

Documentation for PTC ThingWorx Products

You can access PTC ThingWorx documentation, using the following resources available through the PTC Support site:

- PTC ThingWorx Edge SDKs and WebSocket-Based Edge MicroServer Help Center — This Help Center includes documentation for all of the ThingWorx Edge SDKs and for the ThingWorx WebSocket-Based Edge MicroServer (WS EMS). You can browse the entire documentation set, or use the search capability to perform a keyword search. The Help Center contains all the release notes for all of the ThingWorx Edge SDKs and WS EMS.
- PTC ThingWorx Connection Services Help Center — This Help Center includes documentation for the ThingWorx Connection Server, the ThingWorx AWS IoT Connector, and the ThingWorx Azure IoT Hub Connector. You can browse the entire documentation set, or use the search capability to perform a keyword search.
- PTC ThingWorx Help Center — This Help Center includes documentation for the ThingWorx platform, ThingWorx Composer, and ThingWorx Mashup Builder. You can browse the entire documentation set, or use the search capability to perform a keyword search.
- PTC ThingWorx Utilities Help Center.— This Help Center includes release notes for each release of the utilities and all the information you need to use these out-ot-the-box utilities and applications.

Note

With the 8.3.0 release of ThingWorx Utilities, ThingWorx Asset Management and ThingWorx Alert Management have been deprecated and are no longer being developed. They are still included in the 8.3.0 release to support existing customers and will be removed in a future release. Asset and Alert Management functionality has been improved and is now part of the Asset Advisor package. New and existing customers are advised to use Asset Advisor in place of these utilities. Current ThingWorx Utilities customers should contact their PTC sales representative for information about moving to Asset Advisor. See also the release notes for v.8.3.0 of ThingWorx Utilities for information about the changes in packaging for the 8.3.0 release of ThingWorx Utilities.

- PTC ThingWorx Reference Documentation — The Reference Documents pages provide access to the PDF documents available for all PTC ThingWorx products. You can search for documentation in multiple ways. See the online documentation for searching the portal.

  A Service Contract Number (SCN) is required to access the PTC documentation from the Reference Documents website. If you do not know your SCN, see "Preparing to contact TS" on the Processes tab of the PTC Customer Support Guide for information about how to locate it: http://support.ptc.com/appserver/support/csguide/csguide.jsp. When you enter a keyword in the Search Our Knowledge field on the PTC eSupport portal, your search results include both knowledge base articles and PDF guides.

Comments

PTC welcomes your suggestions and comments on our documentation. To submit your feedback, you can:

- Send an e-mail to documentation@ptc.com. To help us more quickly address your concern, include the name of the PTC product and its release number with your comments. If your comments are about a specific help topic or book, include the title.
- Click the feedback icon in any ThingWorx Help Center toolbar and complete the feedback form. The title of the help topic you were viewing when you clicked the icon is automatically included with your feedback.

Terminology Used in This Document

The following table lists and defines terms that as they are used in this document and the ThingWorx C SDK environment:

Term	**Definition**
ThingWorx platform	An instance of the ThingWorx application server that communicates with remote devices. This server is also referred to as “the platform” or “an instance of ThingWorx platform”.
EdgeThing (ET)	An instance of “Thing” on a remote system that inherits the properties, services, and events of a pre-defined EdgeThingTemplate. Even though C does not have the concept of an object, the ThingWorx C SDK supports the construction of a Thing by managing a collection of these capabilities under a single “Thing” name. This collection is recognized as an EdgeThing and appears to ThingWorx platform as an object that can be manipulated.
EdgeThing-Template (ETT)	A collection of properties, services, and events that provide functionality to support a specific activity or hardware device. All EdgeThingShape-based Things have one EdgeThingTemplate.
EdgeThing-Shape (ETS)	A collection of properties, services, and events that provide functionality to support a specific activity or hardware device. More than one EdgeThingShape’s functionality can be used in conjunction with a single EdgeThingTemplate. This behavior is similar to the Composite Pattern, allowing multiple EdgeThingShapes to be compounded into a single EdgeThing at runtime.
Edge Extension	A collection of EdgeThingShapes used in an EdgeThingTemplate that defines one or more Edge things in ThingWorx.

1

Introducing the ThingWorx Edge C SDK

This section provides an introduction to the ThingWorx Edge C SDK, explains its purpose, requirements for using it, and main features. It then explains how to install the SDK and provides a table that shows the directories and files in the installation. Finally, this section provides a Getting Started section, which contains an overview of the process for creating an application using this SDK. This process references later sections of this document where you can find more details.

About the C SDK

The ThingWorx Edge C SDK is a lightweight, but fully functional implementation of the ThingWorx ™AlwaysOn protocol. It is designed to minimize memory footprint while making it easy to integrate applications into the ThingWorx distributed computing environment of the Internet of Things (IoT). The goal of the C SDK is to make creating applications that use it simple, but to also give developers enough flexibility to create very sophisticated applications. For example, the SDK contains a simple “tasker” framework that you can use to call functions repeatedly at a set interval. You can use the tasker framework to drive not only the connectivity layer of your application, but also the functionality of your application. However, it is not required to use the tasker at all. The API is thread safe and can be used in a complex, multi-threaded environment as well. Other examples of this flexibility are highlighted in this document.

Note

The ThingWorx C SDK assumes the use of the C language as defined by the ANSI C89 specification (http://www.open-std.org/jtc1/sc22/wg14/www/projects#9899).

Purpose

The primary functions of the C SDK are as follows:

- Establish and manage a secure AlwaysOn connection with an instance of ThingWorx platform. This includes SSL/TLS negotiation, duty-cycle modulation, and connection maintenance such as re-establishing a connection after network connectivity is lost and restored.
- Enable easy programmatic interaction with the properties, services, and events that are exposed by entities running on ThingWorx platform.
- Implement a callback infrastructure that makes it easy to expose a set of properties and services to ThingWorx platform. These properties and services can be surfaced from multiple entities. When a request is made from ThingWorx platform for a registered property or service, a callback is made to a function that you supply during the registration process.

The C SDK uses callback functions to notify your application of requests for property reads and writes as well as requests to execute a service. The callback function signatures are defined in the `twApi.h` file. Your application can register properties and services (and their metadata) with the API. The metadata is used when browsing remote entities from ThingWorx Composer, making it simple to import functionality created in your application as a thing or thing template into your application model.

The properties, services, and events for ThingWorx platform-side things are easily accessed through appropriate API calls: `twApi_ReadProperty`/`twApi_WriteProperty`, `twAPI_InvokeService`, and `twApi_FireEvent`, respectively.

Features

The C SDK supports the following functionality that allows your machine, device, or application to work with ThingWorx platform:

- Secure Connections—As of v.2.0.0,.the C SDK provides the OpenSSL library and defaults to rejecting self-signed certificates. It also defaults to building with these binaries. It also provides a template for you to use if you want to use another SSL/TLS implementation in your application. The C SDK supports client and server certificate validation. You can enable or disable SSL/TLS certificate validation.

- Security for Passwords—As of v.2.2.0, the C SDK uses a callback function to retrieve an Application Key, digest, certificate password, or proxy user password, or a passphrase for a keystore. This change improves the security of these sensitive pieces of information. See Initializing the API Singleton on page 57 for information on using the callback to retrieve an Application Key for authentication with ThingWorx platform.
- Compression—As of v.2.1, the C SDK supports WebSocket compression at the edge for all WebSocket communications, including file transfers. The zlib compression utility is used to compress and extract files from a ZIP archive. For the version of zlib, see the release notes for v.2.1 of the C SDK.
- Edge Extensions — Provide building blocks of functionality that enable you to create re-usable components. For example, a component that parses files. For more information see [Missing cross reference text].
- Software Content Management (SCM) Edge Extension — As of v.2.1.3, the C SDK includes an Edge Extension that supports ThingWorx SCM capability to create and deploy a package that contains a script to be run on the edge device. Any scripting language is supported, as long as the edge device has the executable for that scripting language installed. For example, you could use Python, Javascript, or nodejs.
- File transfer — The file transfer functionality of the C SDK allows browsing of remote directories and files browsing on an instance of ThingWorx platform, and permits bidirectional file transfer between an edge device and an instance of the platform.
- Data Shapes — You can create Data Shape definitions that model types of metadata for a remote machine/device.
- Tunneling — The tunneling functionality of the C SDK allows you to establish secure, firewall-transparent application tunnels for applications that use TCP, such as VNC and SSH.
- Proxy settings — If your environment requires edge devices to communicate through a proxy server, you can set up your application to connect to ThingWorx platform through a proxy server.
- Offline message storage — Enabled by default, this features queues outgoing messages if the network is down or if the duty cycle is in the “off” state.
- Subscribed properties — Events can subscribe to changes in property values and in aspects of properties.
- Easy build environment — The C SDK supports building with CMake, enabling you to build an application for multiple environments.
- Version information — On startup, the C SDK prints the version number of the C SDK, the SSL/TLS library in use, and its version number. If FIPS mode is enabled, it prints `FIPS Enabled`.

Note

As of release 2.2.1, the non-FIPS distribution bundles of the C SDK include OpenSSL 32– and 64–bit libraries, version 1.0.2q, which, on Windows platforms are based on the Visual Studio 2015 runtime library. The FIPS distribution bundles include 32–bit OpenSSL libraries, v.1.0.2l, which are based on the Visual Studio 2012 runtime library. In addition, the axTLS library is no longer included in any of the distribution bundles..

Installing and Navigating the Directories of the C SDK

Installation

To install the ThingWorx Edge C SDK, go to the PTC Support site (http://www.ptc.com/support/), Software Downloads page, and download the bundle to your computer, and extract the files. After you extract the files, the top level directory is called `<MED-nnnnn-CD-055_<datecode>_C_SDK-v.v.v.bbbb` where `v.v.v.bbbb` is the release number plus the build number. For example, `2-2-1-108` is the release number 2.2.1, with build number 108.

Note

As of v.2.2.1, the non-FIPS distribution bundles of the C SDK include OpenSSL 32– and 64–bit libraries, version 1.0.2q, which, on Windows platforms are based on the Visual Studio 2015 runtime library. The FIPS distribution bundles of this SDK include OpenSSL 32–bit libraries, v.1.0.2l, which are based on the Visual Studio 2012 runtime library.

In addition, the axTLS library is no longer included in the C SDK distribution bundles as of release 2.2.1.

Directories and Files

The following table lists and briefly describes the directories and files of the C SDK. Note that the notation `./` indicates the top-level directory, which is named for the release, `c-sdk-v.v.v.bbbb` and `<version>` in the second-level directory replaces `v.v.v.bbbb`.

This Directory	Contains	See Also
`./c-sdk-<version>`	The file, `version.properties`, which provides the major, minor, and revision numbers that comprise the SDK version. The file, `CMakeLists.txt`, which provides the options for building a C SDK application. Note that such a file is provided for each of the C SDK examples.A `README_BUILDING.txt` file provides instructions for building the SDK with CMake This directory also contains the subdirectories, `doc`, `examples`, `src`, and `test`. The rest of this table describes the contents of these main directories.	Building Your Applications with CMake on page 103
`./c-sdk-<version>/doc`	The PDF files for this document and the release notes; also the `mainpage.md` file for the Doxygen documentation.	The ThingWorx Edge SDKs and WS EMS Help Center, available at PTC ThingWorx Support site, http://support.ptc.com/help/thingworx_hc/thingworx_edge_sdks_ems/
`./c-sdk-<version>/doc/html`	All the files and the `search` subdirectory for the Doxygen documentation.	The `mainpage.md` file provides an overview of the C SDK source code organization. To view the Doxygen documentation in a browser, open any of the *.html files in the `html` directory.
`./c-sdk-<version>/examples`	Subdirectories for the various examples: `ExtUseExample`, `simpleextlib`, `SteamSensor`, and `warehouseextlib`. Each subdirectory contains the source files for the example. Two of the examples also provide an XML file that contains entities for the example. You need to import this XML file into ThingWorx platform (using ThingWorx Composer) before running the example. Finally each example provides the `CMake.txt` file needed to build the example with CMake.	
`./c-sdk-<version>/src`	This directory contains subdirectories that contain all of the source code (*.c and *.h files) for the C SDK. For details, see the table below.	
`./c-sdk-<version>/test`	This directory contains subdirectories for the test suite for the C SDK. For details, see the table, below.	

Source Files Directory (`src`)

This Subdirectory	Contains	See Also
`../src/api`	The API source (*.c) and header (*.h) files. `twApi.c, twApi.h`.	Initialize the API Singleton on page 57
	`twDefinitions.h` contains the enumerated message types, message codes (status, errors), as well as type definitions (characteristic, BaseType, entityType).	twPrimitiveStructure on page 87 Base Types on page 88 twInfoTable on page 89 CallBack Function Return Codes on page 133
	`twErrors.h` contains definitions for different types of errors, including websocket, messaging, primitive/infotable, api, tasker, logger, utils, system socket, file transfer, tunneling, and managed property. It also contains the `#defines` for the `msgCodeEnum` errors.	C SDK Error Codes on page 117
	`twProperties.h` contains definition structures and metadata functions for creating properties. `twProperties.c` contains the implementations of functions for creating and deleting property definitions.	Register Properties and Services on page 59 Property Access Callbacks on page 92 Service Callbacks on page 94 SDK Application-Initiated Interaction on page 96
	`twServices.h` contains service definition structures and metadata functions for services. `twProperties.c`	

Source Files Directory (`src`) (continued)

This Subdirectory	Contains	See Also
	contains the implementations of functions for creating and deleting services.	
	`twVersion.h` contains the `#define` for the version of the C SDK. `twVersion.h.template` contains	
`../src/config`	Two configuration files, `twConfig.h` and `twDefaultSettings.h`. As its name implies, the `twDefaultSettings.h` file contains the default settings for many of the C SDK, parameters, should you decide to use default values and not include your own specific settings in your project. You can also change values here so that you have different default values for your projects. The `twConfig.h` file is provided should you need to override common settings provided in the `CMakeList.txt` file. You can also use it if you are using Windows Solution (sln) or gcc Makefiles (the use of sln and make files is deprecated). Use this file only if you are not using one of these provided	Configuring Components for an Application on page 33 Building Your Applications with CMake on page 103

Source Files Directory (`src`) (continued)

This Subdirectory	Contains	See Also
	files to do per project configuration. Note that the settings in these files apply to ALL of your projects that use the SDK. **Note** You can also edit CMake options at any time by editing them in the CMakeCache.txt files created when you generate your CMake build.	
`../src/fileTransfer`	This directory contains the source and header files for the file transfer functionality of the C SDK, `twFileManager.c`, `twFileManager.h`, `twFileTransferCallbacks.c`, and `twFileTransferCallbacks.h`.	Initializing the File Manager (Optional) on page 61 Create a File Transfer Event Handler (Optional) on page 53 File System Functions on page 115
`../src/messaging`	The following source and header files: - `twBaseTypes.c`, `twBaseTypes.h` contain the definitions of the Base Types of the SDK. - `twInfoTable.c`, `twInfoTables.h` contain the definitions of functions related to creating an infotable with the SDK. - `twMessages.c`, `twMessages.h`	twPrimitiveStructure on page 87 Base Types on page 88 twInfoTable on page 89 Handling Offline Messaging on page 36

Source Files Directory (`src`) (continued)

This Subdirectory	Contains	See Also
	- `twMessaging.c, twMessaging.h`	
`../src/porting`	Source and header files that contain wrappers for OS-specific functionality (`twIos.c, twIos.h, twLinux.c, twLinux.h, twLinux-opensll.h, twMarvellExtras.c, twMarvelThreads.c twOSPort.h, twPThreads.c, twThreads.h, twTlSimplelink.c, twTlSimplelink.h, twWin32Threads.c, twWindows.c, twWindows.h, twWindows-openssl.h`)	Porting to Another Platform on page 108
`../src/stubs`	The stubs files in this directory are for compiling.	Building Your Applications with CMake on page 103
`../src/subscribedProps`	Source and header files (`subscribedProps.c` and `subscribedProps.h`) that contain the functionality to support subscribed properties.	Defining Properties on page 41
`../src/thirdParty`	Third-party libraries for the C SDK, including `cJSON`, `libcfu`, `ntlm`, `openssl-<version>_fips-<version>_sdk`,	For OpenSSL, see Using SSL/TLS for Security on page 72

Source Files Directory (`src`) (continued)

This Subdirectory	Contains	See Also
	`tomcrypt`, and `wildcard`.	
`../src/tls`	The files needed to use SSL/TLS with the SDK.	Using SSL/TLS for Security on page 72
`../src/tunneling`	The source (`twTunnelManager.c`) and header (`twTunnelManager.h`) files for the Tunneling feature.	Configuring Application Tunneling on page 35 Initializing the Tunnel Manager (Optional) on page 62

Source Files Directory (`src`) (continued)

This Subdirectory	Contains	See Also
`../src/utils`	The source and header files for utilities provided by the SDK: • `cryptoWrapper.c`, `cryptoWrapper.h` • `list.c` (for doubly-linked list utilities) • `stringUtils.c`, `stringUtils.h` • `twDict.c`, `twDict.h` • `twHttpProxy.c`, `twHttpProxy.h` • `twList.h` • `twLogger.c`, `twLogger.h` • `twMap.c`, `twMap.h` • `twNtlm.c`, `twNtlm.h` • `twOfflineMsgStore.c`, `twOfflineMsgStore.h` • `twTasker.c`, `twTasker.h`	Using the Utilties of the C SDK on page 65 Handling Offline Messaages on page 36 Using Lists, Maps, and Dictionaries on page 67 Configuring the Tasker on page 34
`tw-c-sdk/src/websocket`	The source (`twWebsocket.c`) and header (`twWebsocket.h`) files for the Websocket Client (abstraction layer).	Interacting with ThingWorx platform on page 86 WebSocket Error Codes on page 118

The following table lists and briefly describes the contents of the `test` subdirectory, which contains all the source code files, entity files, and `CMakeList.txt` files needed to build and run the tests.

Test Subdirectory

This Subdirectory	Contains
`chart-js-client`	`chart-js-client.c` and `chart-js-client.h`
`etc`	Subdirectories and files to support the test applications, including XML files that contain entities used in the tests. The entities need to be imported into ThingWorx platform for the tests to run successfully.
`graphite-c-client`	`graphite-client.c` and `graphite-client.h`l This test is a simple, pure C client for Graphite that allows you to send metrics to Graphite/ Carbon, using Graphite plain-text protocol.
`include`	Header files to support the test utilities.
`integration`	Integration tests for the C SDK that test different features, including file transfers, FIPS, offline message storage, property writes, services, and more.
`integration_slow`	`BindingIntegrationTestsSlow.c`, which tests multiple combinations of bindng and connecting. `OfflineMsgStoreIntegrationTestsSlow.c`, which tests filling up the offline message store and flushing it, in different ways.
`performance`	Performance tests that exercise binding, file transfer, **ListForeach**, property writes, and service execution. Also includes Kepware tests that represent high bandwidth throughput of properties (up to 10000 unique integer properties pushed up to a single thing in under one second).
`src`	. `TestServices.c`,

Test Subdirectory (continued)

This Subdirectory	Contains
	`TestUtilities.c`, `crossPlatformTestSupport.c`, `testmain.c`, and `twPrimitiveUtils.c`
`unit`	Subdirectories and files for unit tests of **twApi**, `twFileManager`, `twOfflineMessageStore`, and more. Also contains the `CMakeLists.txt` file that is needed to build the tests.
`unity`	The Unity Project test framework for C from MIT.
`CMakeLists.txt`	File required for building the test suite using CMake.

2

Getting Started

The best place to start is by examining the examples provided in the `tw-c-sdk/examples` directory. In addition to the *.c and *.h files, each example subdirectory contains the `CMakeLists.txt` file that you need to build the examples using CMake (see Building Applications with CMake on page 103 for more information). The `SteamSensor` and `ExtUseExample` also contain an `import` subdirectory, in which you will find the XML file that you need to import into your ThingWorx platform prior to running the example. The XML files set up everything needed to run and view the results of these examples on ThingWorx platform.

As of v.2.0.0, two examples of Edge Extensions are provided, `simpleextlib` and `warehouseextlib`, and the `ExtUseExample` shows how to load these two example extensions. In addition, the SteamSensor example was rewritten to use the macros that were introduced with the Edge Extensions. For information about these examples, see .[Missing cross reference text].

ThingWorx Configuration

The SDK requires that a `RemoteThing` be created in ThingWorx in order to communicate. Creating a `RemoteThing` is as simple as creating a thing that is derived from one of the `RemoteThing` thing templates and optionally has an Identifier. If an Identifier is supplied, the SDK must use the same identifier as well. Without an Identifier, the `RemoteThing` is referenced by name. The Identifier may be used if a device has access to its serial number via firmware, for instance.

If many Things are to be created with the same properties, services, and events, it is recommended that a Thing Template be derived from one of the supplied `RemoteThing` templates. It will be much easier to maintain the Things and will require less memory on ThingWorx platform. One way to do this is to create a thing, using a `RemoteThing` template, and then browse the client application created with the SDK for its properties, services, and events. Once this work has been completed, create a template based on this thing. Then use the template instead of recreating all the properties, services, and events on each thing.

Application Development

This section provides an overview of the main steps for developing an application using the C SDK.

1. Configure the components that your application will use. The components may include the following:
    - Tasker (optional, Configuring the Tasker on page 34)
    - File Transfer (Configuring File Transfers on page 34)
    - Application Tunneling (Configuring Tunneling on page 35)
    - Handling of Offline Messages (Handling Offline Messages on page 36)
    - Any additional settings (Configuring Additional Settings on page 35)
2. If you need to minimize code footprint, follow the instructions in the section, Minimizing Code Footprint. on page 38
3. Define the properties, events, and services that you want to expose to the server and create the required callback functions. Callback functions can be created to handle individual properties and services, or a single property or service callback can be created to handle all of those types of entities. Refer to the following sections:
    - Defining Properties on page 41
    - Defining Events on page 48
    - Define Property Callback Functions on page 48
    - Defining Service Callback Functions on page 50
4. If your application requires, set up the following:
    - Tasks—See Create Your Tasks (Optional) on page 52.
    - Bind Event Handler—See Create a Bind Event Handler (Optional) on page 53.
    - Synchronized State Handler—See Implementing a Synchronized State Handler on page 55
    - Event Handler for File Transfers—See Create a File Transfer Event Handler (Optional) on page 53.

- Event Handler for Tunneling—See Create a Tunnel Event Handler (Optional) on page 54.
- SSL/TLS for secure communications—See .Using SSL/TLS for Security on page 72 for details.

5. Initialize the API Singleton on page 57.

Note

This initialization function initializes the Subscribed Properties Manager automatically.

6. Register Properties and Services on page 59.

   Register Events on page 60
7. Bind Your Entities on page 60 (Things).
8. If your application requires it, initialize the following components:
   - File Manager — Refer to Initialize the File Manager (Optional) on page 61.
   - Tunnel Manager — Refer to Initialize the Tunnel Manager (Optional) on page 62.
9. Connect to the Server and Initiate Tasks on page 68.
10. Once your connection is alive and active, any requests made on the server for registered properties and services are automatically forwarded to your application, and the appropriate callback function is called. For information about server-initiated actions and callback functions, refer to Server-Initiated Interaction on page 92.

    Helper functions are available to push properties to the server, execute a service on another entity in the system, or trigger an event on the server. Refer to SDK Application-Initiated Interaction on page 96.
11. Build your application using CMake. Follow the instructions in .Building Applications with CMake on page 103

Note

As of v.2.1, WebSocket compression is enabled by default for ALL WebSocket communications, including file transfers and pushing property values to ThingWorx platform. If you need to disable compression, use the `twApi_DisableWebSocketCompression()` function. You can find information for the function in the files, `twApi.h` and `twApi.c` in the `../src/twApi/` subdirectory of your C SDK installation.

Configuring Components of the C SDK

Once you have decided which components your application requires, you must define the components as explained in this section. This section assumes that you will use CMake to build your application.

Configure the desired components to include in a `CMakeLists.txt` file, and verify that the SDK supports your platform/OS. If not, refer to the sections on building (Building Applications with CMake on page 103) and porting (Porting to Another Platform on page 108), which describes the requirements and process for porting the SDK.

Provided within the SDK examples directory are example applications that demonstrate various capabilities of the SDK. Within each of those directories are a `win32`, `osx`, and a `linux` subdirectory, each with their own source code and `CMakeLists.txt` file. The section,Building Applications with CMake on page 103 includes a table that lists and briefly describes the options that you can set at the command line with CMake. It is HIGHLY RECOMMENDED that you use one of these examples to gain an understanding of what source files and configuration settings need to be included in your build environment.

To learn about the configuration settings for applications, open the following files in the `src/config/` subdirectory of the C SDK installation:

- `twConfig.h` — This file provides a place for general settings for C SDK applications, including offline message store. Any settings in this file will apply to ALL of your projects. To override the settings for specific applications, use a `CMakeLists.txt` file to change the default configuration. However, if you are NOT building with CMake, use this file to configure the SDK for your application.
- `twDefaultSettings.h` — This file explains all of the configuration settings for the C SDK. The comments in this file provide specific information about the settings.

Note

Configuration affects the footprint and RAM usage of the SDK, and consequently, any application built using the SDK.

Editing the `CMakeLists.txt` file is not required. You can set options from the CMake command line. For example:

```
cmake =DMyOption=ON MyProjectFolder
```

OR

```
cmake -DUSE_OPENSSL=ON
```

Continue to the following configuration tasks:

- Configuring the Tasker on page 34
- Configuring File Transfers on page 34
- Configuring Tunneling on page 35
- Configuring Additional Settings on page 35

Configuring the Tasker

The built-in tasker is a simple round-robin execution engine that will call all registered functions at a rate defined when those functions are registered. If using a multitasking or multi-threaded environment you may want to disable the tasker and use the native environment. If you choose to disable the tasker, you must call **twApi_TaskerFunction()** and **twMessageHandler_msgHandlerTask()** on a regular basis (every 5 milliseconds or so). Un-define this setting if you are using your own threads to drive the API, as you do not want the tasker running in parallel with another thread running the API.

To properly initialize the Tasker, you must define **ENABLE_TASKER**:

```
/*********************************/
/*  Tasker Configuration       */
/*********************************/
#define ENABLE_TASKER 1
```

The Windows-based operating systems have a tick resolution (15ms) that is higher than the tick resolutions requested by the C SDK (5ms). For information about achieving the best performance in a Windows-based operating system, see Running on a Windows-based Operating System on page 70.

Back to top on page 33

Configuring File Transfers

The C SDK has full support for all the remote directory/file browsing capabilities of ThingWorx platform as well as bidirectional file transfer. To use this functionality, define `ENABLE_FILE_XFER`. This module will add ~15KB of code space to your application, so severely constrained environments may want to omit this functionality.

```
/*********************************/
/* File Transfer Configuration   */
/*********************************/
#define ENABLE_FILE_XFER 1
```

As of v.2.1, the C SDK uses WebSocket compression (zlib) for all WebSocket communications, including file transfers, by default. In general, leaving compression turned on reduces bandwidth used by the edge application, but requires more memory and CPU usage. Choose whether to leave compression enabled, based on your the capabilities of your devices and on the available bandwidth. If you need to disable compression, use the `twApi_DisableWebSocketCompression()` function. This function is in the `twApi.c` file, which is located in the `./src/api/`directory of your C SDK installation. Its description is in the `twApi.h` file, which is located in the `./src/api/` directory. To call the function, add it to your code after you initialize the API (by calling `twApi_Initialize()` and where you configure the API. For example:

```
/* Configure API */
...
tw_Api_SetSelfSignedOk();
twApi_DisableWebSocketCompression();
twApi_SetOfflineMsgStoreDir(offlineMsgStoreDir);
...
```

This example sets up an offline message store location and disables compression. When the messages that are stored while the device is offline are eventually transmitted, the messages will use more memory and CPU. Disabling compression is useful if you have limited bandwidth or want to keep bandwidth costs down. .

Note

This example allows the use of a self-signed certificate. In general, use self-signed certificates ONLY while developing and testing an application. In production, use a highly secure connection.

Configuring Tunneling

The C SDK has full support for application tunneling. Application tunnels allow for secure, firewall transparent tunneling of TCP client server applications such as VNC and SSH. To use this functionality, you must define `ENABLE_TUNNELING`. This module will add ~5KB of code space to your application, and upwards of 100KB RAM, depending on usage, so severely constrained environments may want to omit this functionality.

```
a/*********************************/
/*    Tunneling Configuration    */
/*If defined, the tunneling system will be enabled.
*/
#define ENABLE_TUNNELING 1
```

The Windows-based operating systems have a tick resolution (15ms) that is higher than the tick resolutions requested by the C SDK (5ms). For information about achieving the best performance for tunneling in a Windows-based operating system, see Running on a Windows-based Operating System on page 70.

Configuring Additional Settings

The C SDK has several settings that you can modify, based on the needs of your application for things such as minimizing RAM usage or improving performance. The defaults for these settings are found in the file `src/config/`

`twDefaultSettings.h`. In most cases you do not need to change these settings. If you must change them, exercise caution when making the changes. With the exception of `TW_MAX_TASKS`, all of the settings can be modified at runtime by changing the appropriate setting in the global **`twcfg`** structure. The structure definition can be found in `src/config/twDefaultSettings.h`.

Note

The default setting for `DEFAULT_SOCKET_READ_TIMEOUT` in `twDefaultSettings.h` is 500 ms. If you are using SSL/TLS to connect to ThingWorx platform and a websocket read times out in the middle of reading a record, the SSL state is lost. As a result, the SDK tries to start reading the record header again, and the operation fails. To detect this situation, check the log for the SDK for the error, `twTlsClient_Read: Timed out after X milliseconds`, and consider increasing the value of the `DEFAULT_SOCKET_READ_TIMEOUT`. You can change the setting at runtime by modifying the value of `twcfg.socket_read_timeout`.

Handling Offline Messages

The C SDK has multiple options for offline message storage. In general, offline message storage will queue up outgoing request messages for later delivery if the network is down or the duty cycle modulation component of the AlwaysOn protocol happens to be in the “off” state. When offline message storage is enabled and the device is offline, outgoing messages are placed in a queue, up to a limit of `OFFLINE_MSG_QUEUE_SIZE`. When connectivity is re-established, all the messages in this queue are sent out to the ThingWorx platform.

The default setting for offline message storage is that offline message storage is enabled and will persist the messages to a file. If storage space is not available on the device, you can still enable offline message storage that stores messages in RAM only. For the smallest footprint, you can disable this feature.

Note

If you enable offline message store but limit the storage to RAM only, keep in mind that, if there is a power outage or the system is shut down for any reason, these messages will be lost and not delivered to the ThingWorx platform.

To configure offline message store, open either a `CommonSettings` file or the file, `twConfig.h`, and add the following parameters:

- `OFFLINE_MSG_STORE` — To disable the feature, change the 2 to 0. To enable it but store only in RAM, change the 2 to 1.
- `OFFLINE_MSG_STORE_DIR` — If you are using the feature and storing messages in files, specify the directory in which you want to store the messages.
- `OFFLINE_MSG_QUEUE_SIZE` — If you are using the feature, specify the maximum size of the message queue (number of messages).

In both the RAM-based, and file-based offline message stores, when connectivity is re-established all the messages in this queue are sent out to the ThingWorx platform. Note that it is quite likely that all of the original messages will time out while waiting for a response from the platform, so you will not receive any indication or confirmation that these messages were successfully processed by the platform. Also in either use case (RAM-based or file-based), if the total size of the queued messages exceeds the limit defined in `OFFLINE_MSG_QUEUE_SIZE`, any subsequent attempt to queue more messages will fail and those new messages will be lost.

The structure for offline message store is defined in `./src/utils`. A singleton instance of this structure is automatically created when `* twOfflineMsgStore_Initialize()` is called. You should not need to manipulate this structure directly. You can make the following types of requests to the offline message store:

- `OFFLINE_MSG_STORE_FLUSH` — Request to flush the offline message store buffer.
- `OFFLINE_MSG_STORE_WRITE` — Request to write a message into the offline message store.

As of release 1.3.3 of the C SDK, the offline message store is automatically initialized by `twApi_Initialize()`. However, if your application requires a more complex offline message store model, you can initialize it separately by calling `twOfflineMsgStore_Initialize()`, and passing in the following arguments:

- `enabled` — A boolean value to enable/disable the offline message store.
- `filePath` — The path to the offline message store directory.
- `size` — The maximum size of the offline message store.

If the initialization is successful, this function returns `TW_OK`. If not, it returns an error (see `twErrors.h` for definitions of the errors).

The following additional functions are available for offline message storage as of release 1.3.3:

- Specify the directory in which to store the offline messages — `twOfflineMsgStore_SetDir()`.

- Free memory that is associated with the offline message store singleton — `twOfflineMsgStore_Delete()`.
- Process requests to the offline message store — `twOfflineMsgStore_HandleRequest()`.

Lost Messages

If the connection to ThingWorx platform is lost and offline message store is enabled, the messages currently waiting to be sent to ThingWorx are stored in the offline message store, as configured. Once the application reconnects and authenticates with ThingWorx platform again, the messages are sent based on your configuration, as described above. However, if the ping timeout and connection retries are exhausted, the SDK will disconnect and not reconnect unless the application invokes `twApi_Connect`. Messages destined for ThingWorx platform will be lost.

This situation may occur in an environment where network connectivity is forcibly removed. To avoid the loss of messages, consider adjusting the setting for retrying the connection to ThingWorx platform (`connection_retries` in `twApi_initialize()`). In addition, if it is not already enabled, enable automatic re-connection. To enable automatic re-connection, set `autoreconnect` to `true` in `twApi_initialize()` so that the application automatically tries to reconnect when a connection is lost.

If these changes do not resolve the situation, set the number of connection retries to -1 (`connection_retries= -1`). The SDK will try indefinitely to reconnect. Be aware that with these settings, it might appear that the application is in an infinite loop. In reality, the SDK just is not able to connect and is constantly retrying the connection.

Minimizing Code Footprint

To attempt to create the smallest possible code footprint, define `TW_LEAN_AND_MEAN`. Using `TW_LEAN_AND_MEAN` disables optional, resource-consuming entities, such as offline message storage, tunneling, and file transfer. The default behavior is to remove all logging from the system.

Another way to minimize code footprint is to disable the resource-consuming entities you do not require.

The following code example shows the definition for `TW_LEAN_AND_MEAN`:

```
/*********************************/
/*   Minimize Code Footprint     */
/*********************************/
/*
Attempts to minimize the code footprint at the
expense of functionality.  Check your OS port
 header file to see what is disabled.
```

```
*/
#define TW_LEAN_AND_MEAN
```

Tips for Minimizing Footprint and Maximizing Performance

The C SDK has several settings that can significantly impact code footprint and performance. For performance, key among them is disabling verbose logging mode. Verbose logging parses every message sent between your application and ThingWorx platform. While extremely valuable for debugging, it can have a significant impact on performance. It is recommended that you disable verbose logging by calling `twLogger_SetIsVerbose(FALSE);`

Several areas impact code footprint. Support for connecting through HTTP Proxies adds ~5KB to your final code size. If not needed, follow this example: Suppose you are connecting over a cellular connection. To disable the support for HTTP Proxies, use `#undef ENABLE_HTTP_PROXY_SUPPORT`.

In addition, support for NTLM proxies adds ~45KB of code. To disable this support, use `#undef USE_NTLM_PROXY`.

File Transfer and Tunneling add ~15KB and 5KB respectively. You can disable them, using `#undef ENABLE_FILE_XFER` and `#undef ENABLE_TUNNELING`.

Finally logging itself adds ~20KB of code. Logging can be disabled with macros in parts by defining the log functions as empty as follows:

```
#define TW_LOG(level, fmt, ...)
#define TW_LOG_HEX(msg, preamble, length)
#define TW_LOG_MSG(msg, preamble)
```

The `twWindows.h` or `twLinux.h` files provide examples of using `TW_LEAN_AND_MEAN` to minimize the code footprint.

The SteamSensor example, `SteamSensorWithMinimalFootprint`, in the `/examples` subdirectory of the installation shows how to set up an application that minimizes the footprint. When updating properties, you can minimize footprint by processing the property updates individually. To minimize the use of bandwidth, process the property updates all at once. The example shows both ways to update properties.

.

3

Steps for Setting Up Applications

What do you need to do to set up the application using the C SDK? This section explains some of those steps, including defining the properties and services that you want to expose to the server and implementing the required callback functions. Callback functions can be used to handle individual properties and services or a single property or service callback can be created to handle all of those types of entities. This decision is left to the application developer.

Optionally, you may need to create tasks on page 52 as well as event handlers:

- Bind event handler on page 53, so the application can determine which entities are bound to ThingWorx platform),
- File transfer event handler on page 53 for file transfers to and from ThingWorx platform.
- Tunneling event handler on page 54 for open and close events.
- Synchronized state handler on page 55

The C SDK uses a callback mechanism to handle server-initiated requests to read or write properties and invoke services. The signatures of the callback functions and the registration functions themselves are found in the file, `src/api/twApi.h`.

Defining Properties

In the ThingWorx environment, a property represents a data point, which has a name, a value, a timestamp, and optionally, a quality. In ThingWorx platform, properties can also have *aspects*, which provide additional details about the property. Once a client application binds an entity to a corresponding RemoteThing on the ThingWorx platform, you can associate properties with the RemoteThing, using ThingWorx Composer.

The C SDK supports two types of properties, properties that do not have Remote Binding Information ”aspects” and so-called *subscribed* properties that have Remote Binding Information aspects that are displayed in ThingWorx Composer. These aspects are described in the *Property Definitions* section below.

Two types of structures are used by the C SDK to define properties:

- Property Definitions (`twPropertyDef`) to describe the basic information for the properties that are going to be available to the ThingWorx platform and can be added to a client application.
- Property Values (`twProperty`) to associate the property name with a value, timestamp, and quality.

The structures are defined in the file, `twProperties.h`.

The following example of a simple property structure from the Steam Sensor example shows how the declaration of properties works:

```
/*****************
A simple structure to handle
properties.
******************/
struct  {
double TotalFlow;
char FaultStatus;
char InletValve;
double Pressure;
double Temperature;
double TemperatureLimit;
twLocation Location;
char * BigGiantString;
} properties;
```

To store the values sent by ThingWorx platform, you must use a callback method to either allocate a new variable or set the memory in an already allocated variable. For information about registering callbacks for properties, refer to Registering Properties and Services on page 59. For additional information, see also Property Access Callbacks on page 92 and the sections on reading, writing, and pushing properties in the section, SDK Application-Initiated Interaction on page 96.

Property Definitions

The basic information that you provide for a Property Definition includes the following attributes:

- `name` — Specifies the name of the property that will appear in ThingWorx Composer when users browse the related Thing while the platform is binding to the Thing.
- `description` — Provides a description of the property that gives further understanding of the meaning of the property.
- `baseType` — Specifies the type of the property. For a list of base types supported by the SDK, refer to Base Types on page 88.
- `aspects` — Define the ways to interact with a property. All properties have the following aspects:
    - `isPersistent` — Set to `TRUE` for the ThingWorx platform to persist the value even if it restarts. It is extremely expensive to have persistent values, so it is recommended to set this value to `FALSE` unless absolutely necessary.
    - `isReadOnly` — Set to `TRUE` to inform the ThingWorx platform that this value is only readable and cannot be changed by a request from the server.
    - `dataChangeType` — Describes how the ThingWorx platform responds when the value changes in the client application. Subscriptions to these value changes can be modeled in the ThingWorx platform. If nothing needs to react to the property change, set this value to `NEVER`. The possible values are:

Value	Description
`ALWAYS`	Always notify of the value change even if the new value is the same as the last reported value.
`VALUE`	Only notify of a change when a newly reported value is different than its previous value.
`ON`	For BOOLEAN types, notify only when the value is `true`.
`OFF`	For BOOLEAN types only, notify when the value is `false`.
`NEVER`	Ignore all changes to this value.

    - `dataChangeThreshold` — Defines how much the value must change to trigger a change event. For example 0 (zero) indicates that any change triggers an event. A value of 10 (ten) for example would not trigger an update unless the value changed by an amount greater than or equal to 10.

  - `defaultValue` — The default value is the value that the ThingWorx platform uses when the RemoteThing connected to the device first starts up and has not received an update from the device. The value is different based on the different value for each base type.

- Only properties defined as *subscribed* properties have the following Remote Binding aspects:
  - `cacheTime` — The amount of time that the ThingWorx platform caches the value before reading it again. A value of -1 informs the server that the client application always sends its value and the server should never go and get it. A value of 0 (zero) indicates that every time the platform uses the value, it should go and get it from the client application. Any other positive value indicates that the server caches the value for that many seconds and then retrieves it from the client application only after that time expired.

Note

For the client application to set the value every time it changes, set this value to -1.

  - `pushType` — Informs the ThingWorx platform how the client application pushes its values to the platform. The possible values are as follows:

Select	For the Client to
`ALWAYS`	Send updates even if the value has not changed. It is common to use a `cacheTime` setting of `-1` in this case.
`NEVER`	Never send the value, which indicates that ThingWorx platform only writes to this value.It is common to use a `cacheTime` setting of 0 or greater in this case.

Select	For the Client to
`VALUE`	Send updates only when the value changes. It is common to use a `cacheTime` setting of `-1` in this case. **Note** As of v.1.5.0, if a property is using this push type and only the quality for that property changes, the update is propagated to ThingWorx platform.
`DEADBAND`	Added to support KEPServer, this push type is an absolute deadband (no percentages). It provides a cumulative threshold, such that the Edge device should send an update if its current data point exceeds Threshold compared to the last value sent to ThingWorx platform. It follows existing threshold fields limits.

Properties need to be registered so that ThingWorx platform can browse them. Refer to Registering Properties and Services on page 59.

Property Values

You can define the property value in two ways – one with specific settings for timestamp and quality and one with the default quality.

Note

Updating a property value does not send the value to the ThingWorx platform. To send the value to the platform, the `twSubscribedPropsMgr_PushSubscribedProperties` function must be called.

Helper functions for creating property values include:

- `setPropertyVTQ` — Sets a property's value using a VTQ (value, time, and quality) structure.
    - `name` — The name of the property.
    - `value` — The VTQ (value, time, and quality) for the property's value.
    - `forceChange` — Set this value to true to force the value to be sent to the ThingWorx platform even if it has not changed. This option is a good option for sending the first value or sending a value immediately after reconnect.
- `setPropertyValue` — Sets the value of a property, using a `Primitive` type.

  - `name` — The name of the property.
  - `value` — The `Primitive` type for the value.
- `setProperty` — Sets a property's value from an object.
  - `name` — The name of the property.
  - `value` — The value to set. The value will be cast to the type of property if possible; otherwise an exception will be thrown.

Setting Up the Subscribed Properties Manager

The *subscribed properties* have a separate manager, called Subscribed Properties Manager (defined in the source file, *`./subscribedProps/twSubscribedProps.h`*). . To set up the use of the Subscribed Properties Manager, the `twConfig.h` files should have the following members:

- `#subscribed_props_enabled` — Controls whether the Subscribed Property Manager will persist offline property updates.
- `#subscribed_props_queue_size` — Limits the maximum size of the Subscribed Properties bin that stores the offline property updates.
- `#subscribed_props_dir` — Specifies the path to the directory for subscribed properties.

As of v.1.3.0 of the C SDK, these properties can be set independently during initialization. You can copy them from the `twConfig` structure in the `twDefaultConfig.h` file to your `twConfig.h` file and set the values according to the requirements of your application. Previously these properties were stored in the `tw_api struct` and were derived from the offline message store settings. For this reason, the default values shown in the `twDefaultConfig.h` file still point to the default values in the offline message store.

Note

The Subscribed Properties Manager is initialized automatically when you call `twAPI_initialize()`. You do not need to initialize it separately.

Setting Subscribed Properties

For efficient throughput, the functions `twApi_SetSubscribedPropertyVTQ` and `twApi_PushSubscribedProperties` are essential. For even better throughput, consider using the asynchronous version of the call, `twApi_PushSubscribedPropertiesAsync`.

After setting properties individually using `twApi_SetSubscribedPropertyVTQ`, you can alternatively use the Subscribed Properties Manager (SPM) to push the subscribed properties all at once to ThingWorx platform (`twSubscribedPropsMgr_PushSubscribedProperties`).

See the table below for more information about these functions.

Subscribed Properties Functions

Function	Description
`twApi_SetSubscribedPropertyVTQ`	This function sets a specified subscribed property for a specified thing, including the name of the property, its value, timestamp, and quality. Note that this call is blocked while the `twApi_PushSubscribedProperties` is being called.
`twApi_PushSubscribedProperties`	This function pushes subscribed properties for a specified thing to ThingWorx platform. When this function is being called, it blocks the function `twApi_SetSubscribedPropertyVTQ`. This function does not return until the data is sent to ThingWorx platform and a response is received (or times out).

Subscribed Properties Functions (continued)

Function	Description
`twSubscribedPropsMgr_PushSubscribedProperties()`	This function sends ("pushes") subscribed properties to ThingWorx platform. When this function is called, the calling thread is blocked. This call also blocks other threads from invoking `twApi_PushSubscribedProperties` at the same time. If you need better throughput, use the asynchronous `twApi_PushSubscribedPropertiesAsync` instead.
`twApi_PushSubscribedPropertiesAsync(char * entityName, char forceConnect,PushSubscribedPropertiesAsyncCallback cb, void* userdata)`	This function sends ("pushes") subscribed properties to ThingWorx platform asynchronously. The calling thread does not block nor does it bock other threads. The `userdata` is used as a correlation Id and will be passed into the callback. You can use the callback to handle the asynchronous results from this call. When this function is called, the calling thread is not blocked nor does this call block other threads from making the same call.. If `twApi_PushSubscribedPropertiesAsync` results in multiple `UpdateSubscribedPropertyValues` calls (due to message size), then the callback will be invoked multiple times. In all cases, the final callback invocation will be indicated by a `char` parameter on the callback function.

Defining Events

Event definitions describe interrupts that ThingWorx platform users can subscribe to if they want to be notified when something happens.

Events require that a data shape for event data be defined in code. Events can be defined in code or by using the following attributes:

- ThingWorxEventDefinition — Defines the event.
- name — Name of the event.
- description — A description for the event.
- dataShape — The name of the data shape for the event data.

Events must be registered. Refer to Register Events on page 60 for details. The registered event is reported back to the server when it is browsing. Note that Events do not have callbacks since they cannot be invoked from ThingWorx platform to the Edge. You can add aspects to an Event that is already registered, using `twApi_AddAspectToService`.

Define Property Callback Functions

The property callback function is registered to be called when a request for a specific property is received from ThingWorx platform; for example, if a service or a mashup references a property.

```
typedef enum msgCodeEnum (*property_cb)
(const char * entityName, const char* propertyName,
twInfoTable** value, char isWrite, void * userdata)
```

The following parameters are passed to this function:

- `entityName` — the name of the entity this request is for
- `propertyName` — the name of the property the request is for
- `twInfoTable ** value` — a pointer to an `twInfoTable` that will contain the new property value if this is a write or will be populated with the current property value if this is a read. (For information on InfoTables, see the section, twInfoTable on page 89.)
- `isWrite` — a Boolean indicator saying whether this is a read or a write
- `userdata` — any user data value that was passed in when the callback was registered.

The return value of the function should be a message code enumeration as defined in `src/api/twDefinitions.h`. These message codes reflect the overall success or failure of your read or write operation locally. For more information about the return values, refer to the appendix, Callback Function Return Codes on page 133.

Pushing Property Changes from ThingWorx to Edge Devices.

When properly bound to a remote property on an Edge device, properties on ThingWorx `RemoteThing` instances can be used for both reading values from and writing new values to the device. For C SDK implementations, use the function, `twApi_RegisterPropertyCallback()`, to register properties for which you expect ThingWorx platform to push down values. Then use the property handler callbacks to update the property values received from ThingWorx platform.

When a new value is set for a remote property on a `RemoteThing` instance, the value is sent down to the edge drive. For Remote Properties configured with a `Cache Option` of `Read from Server Cache`, ThingWorx platform continues to show the old value of the property until the edge device confirms the new value by sending it back in a property update. This behavior gives the device the ability to decide if the new value is valid before updating the value in ThingWorx platform.

Tip

To ensure that a property displays an accurate value at all times, you can set the `Cache Option` to `Fetch from Remote on Every Read.` This setting increases the amount of data sent between ThingWorx platform and the edge device because every request for the property retrieves the data directly from the device. Use this option sparingly with devices on metered connection.

Example

```
/*****************
Property Handler Callbacks
******************/
enum msgCodeEnum propertyHandler(const char * entityName,
const char * propertyName,  twInfoTable ** value,
char isWrite, void * userdata) {
  TW_LOG(TW_TRACE,"propertyHandler - Function called for Entity %
s,
        Property %s", entityName, propertyName);
  if (value) {
    if (isWrite && *value) {
        /* Property Writes */
        if (strcmp(propertyName, "InletValve") == 0)
           twInfoTable_GetBoolean(*value, propertyName, 0,
           &properties.InletValve);
        else if (strcmp(propertyName, "FaultStatus") == 0)
           twInfoTable_GetBoolean(*value, propertyName, 0,
           &properties.FaultStatus);
        else if (strcmp(propertyName, "TemperatureLimit") == 0)
```

```
            twInfoTable_GetNumber(*value, propertyName,
            0, &properties.TemperatureLimit);
        else return NOT_FOUND;
        return SUCCESS;
 } else {
        /* Property Reads */
        if (strcmp(propertyName, "InletValve") == 0)
          *value = twInfoTable_CreateFromBoolean(propertyName,
                   properties.InletValve);
        else if (strcmp(propertyName, "Temperature") == 0)
             *value = twInfoTable_CreateFromNumber(propertyName,
                       properties.Temperature);
        else if (strcmp(propertyName, "TemperatureLimit") == 0)
             *value = twInfoTable_CreateFromNumber(propertyName,
                      properties.TemperatureLimit);
        else if (strcmp(propertyName, "Location") == 0)
             *value = twInfoTable_CreateFromLocation(propertyName,
                      &properties.Location);
        else if (strcmp(propertyName, "BigGiantString") == 0)
             *value = twInfoTable_CreateFromString(propertyName,
                      properties.BigGiantString, TRUE);
        else return NOT_FOUND;
        }
        return SUCCESS;
 } else {
     TW_LOG(TW_ERROR,"propertyHandler - NULL pointer for value");
     return BAD_REQUEST;
     }
}
```

Define Service Callback Functions

The service callback function is registered to be called when a request for a specific service is received from ThingWorx platform.

```
typedef enum msgCodeEnum (*service_cb)
(const char * entityName, const char * serviceName,
twInfoTable * params,twInfoTable** content, void * userdata)
```

The following parameters are passed to this callback function:

- `entityName` — the name of the entity this request is for (Thing, Resource, for example). Guaranteed to not be `NULL`.
- `serviceName` — the name of the service being requested
- `twInfoTable *params` — a pointer to an `twInfoTable` that contains all the parameters for the service. May be NULL if service has no parameters. (For information on InfoTables, see the section, twInfoTable on page 89)

- `twInfoTable ** content` — a pointer to a pointer to a `twInfoTable`. `content` is guaranteed to not be `NULL`. `*content` is not.

  Note

  A new instance of a `twInfoTable` should be created on the heap and a pointer to it returned.

- `userdata` — any user data value that was passed in when the callback was registered.

The return value of the function is `TWX_SUCCESS` if the request completes successfully or an appropriate error code if not (should be a message code enumeration as defined in `twDefinitions.h`).

Example

Here is an example of handling a single service in a callback:

```
/*****************
Service Callbacks
******************/
/* Example of handling a single service in a callback */
enum msgCodeEnum addNumbersService(const char * entityName,
const char * serviceName, twInfoTable * params,
twInfoTable ** content, void * userdata) {
        double a, b, res;
        TW_LOG(TW_TRACE,"addNumbersService - Function called");
        if (!params || !content) {
                TW_LOG(TW_ERROR,"addNumbersService -
                  NULL params or content pointer");
                return BAD_REQUEST;
        }

        twInfoTable_GetNumber(params, "a", 0, &a);
        twInfoTable_GetNumber(params, "b", 0, &b);
        res = a + b;
        *content = twInfoTable_CreateFromNumber("result", res);
        if (*content) return SUCCESS;
        else return INTERNAL_SERVER_ERROR;
}
```

Create Your Tasks (Optional)

If using the built-in tasker to drive data collection or other types of repetitive or periodic activities, create a function for the task. Task functions are registered with the Tasker and then called at the rate specified after they are registered. The Tasker is a very simple, cooperative multitasker, so these functions should not take long to return and most certainly must not go into an infinite loop.

The signature for a task function is found in `src/utils/twTasker.h`. The function is passed a `DATETIME` value with the current time and a void pointer that is passed into the Tasker when the task is registered.

Here is an example of a data collection task:

```
/***************
Data Collection Task
****************/
/*
This function is called at the rate defined in the task creation.

The SDK has a simple cooperative multitasker, so the function
cannot infinitely loop.
Use of a task like this is optional and not required in a
multithreaded
environment where this functionality could be provided in a
separate thread.
*/
#define DATA_COLLECTION_RATE_MSEC 2000
void dataCollectionTask(DATETIME now, void * params) {
  /* TW_LOG(TW_TRACE,"dataCollectionTask: Executing"); */
      properties.TotalFlow = rand()/(RAND_MAX/10.0);
      properties.Pressure = 18 + rand()/(RAND_MAX/5.0);
      properties.Location.latitude = properties.Location.latitude
+

                       ((double)(rand() - RAND_MAX))/RAND_MAX/5;
        properties.Location.longitude = properties.Location.
longitude +

                       ((double)(rand() - RAND_MAX))/RAND_MAX/5;
        properties.Temperature  = 400 + rand()/(RAND_MAX/40);
        /* Check for a fault.  Only do something if we haven't
already */
        if (properties.Temperature > properties.TemperatureLimit
&&

                                     properties.FaultStatus ==
FALSE) {
                twInfoTable * faultData = 0;
                char msg[140];
                properties.FaultStatus = TRUE;
                properties.InletValve = TRUE;
                sprintf(msg,"%s Temperature %2f exceeds threshold
of %2f",
                        thingName, properties.Temperature,

                        properties.TemperatureLimit);
                faultData = twInfoTable_CreateFromString("msg",
msg, TRUE);
                twApi_FireEvent(TW_THING, thingName,

                                "SteamSensorFault", faultData, -1,
TRUE);
                twInfoTable_Delete(faultData);
        }
        /* Update the properties on the server */
        sendPropertyUpdate();
}
```

Creating a Bind Event Handler (Optional)

You may want to track exactly when your edge entities are successfully bound to or unbound from ThingWorx platform. The reason for this is that only bound items should be interacting with ThingWorx platform and it will never forward a request to a corresponding remote thing in its database when the request is targeted at an entity that is not bound.

```
/* Register a bind event handler */
/* Callbacks only when thingName is bound/unbound */
    twApi_RegisterBindEventCallback(thingName, BindEventHandler,
NULL);

/* First NULL says "tell me about all things that are bound */

/* twApi_RegisterBindEventCallback(NULL, BindEventHandler, NULL
```

Create a File Transfer Event Handler (Optional)

If you are using the File Transfer capability of the C SDK, you may want to create an event handler for any file transfer events. This handler will be called whenever a new file is successfully sent from the server to your application, and when an asynchronous file transfer from your device to the service has completed either successfully or unsuccessfully.

The signature for a file transfer event callback is as follows:

```
typedef void (*file_cb) (char fileRcvd, twFileTransferInfo *
info);
```

The input parameters for this callback function are as follows:

- `fileRcvd` — a Boolean. `TRUE` is the file was received, `FALSE` if it was being sent
- `info` — a pointer to the file transfer info structure. The called function retains ownership of this pointer and must delete it with `twFileTransferInfo_Delete()` when it has finished using it

Return:

- None

The structure definition of `twFileTransferInfo` can be found in the file `src/fileTransfer/twFileManager.h`.

Create a Tunnel Event Handler (Optional)

If you are using the Tunneling capability of the C SDK, you may want to create an event handler for any tunneling events. This handler will be called whenever a new tunnel is established or when a tunnel closes. The `twTunnelManager` also provides functions to list active tunnels as well as to force a shutdown of an active tunnel.

The signature for a tunnel event callback is as follows:

```
typedef void (*tunnel_cb) (char started, const char * tid,
const char * thingName,
    const char * peerName,
const char * host, int16_t port, DATETIME startTime,

  DATETIME endTime, uint64_t bytesSent, uint64_t bytesRcvd,

const char * type, const char * msg, void * userdata);
```

The Input parameters for this callback function are as follows:

- `started` — Boolean. TRUE is the tunnel is started, FALSE if tunnel has ended.
- `tid` — the unique id of the tunnel
- `thingName` — the name of the thing this tunnel is targeted at
- `peerName` — the name of the peer user of the tunnel
- `host` — the hostname of the local connection that is tunneled to
- `port` — the port number of the local connection that is tunneled to
- `startTime` — the time the tunnel started (0 if it never started)
- `endTime` — the time the tunnel ended (0 if it hasn't ended yet)
- `bytesSent` — the total number of bytes that were sent to the peer
- `bytesRcvd` — the total number of bytes that were received from the peer
- `type` — the type of the tunnel (tcp, udp, or serial)
- `userdata` — an opaque pointer that was passed in during registration

Return:

- None

The definition of the `twTunnelManager` singleton's functions can be found in the file `src/tunneling/twTunnelManager.h`.

Implementing a Synchronized State Handler

As of version 1.4.0, the C SDK provides the ability to install handlers that notify you when `getPropertySubscriptions()` is being called on ThingWorx platform in response to a `notifyPropertyUpdates` message. This mechanism works in a similar manner to the bind event handler.

Tip

Consider the Synchronized State event handler as a best practice for edge applications.

4

Running the C SDK

After developing the callback handler functions, it is now time to do something with them. Continue here to learn what you should typically do in your 'main' function (or in a function called by main).

See the following topics:

- Initializing the API Singleton on page 57
- Registering Properties and Services on page 59
- Registering Events on page 60
- Binding Your Entities on page 60
- Initializing the File Manager (Optional) on page 61
- Initializing the Tunnel Manager (Optional) on page 62
- Using the Utilities of the C SDK on page 65
    - Using Linked Lists, Maps, and Dictionaries on page 67
- Connecting to the Server and Initiating Defined Tasks on page 68
- Running the C SDK on Windows-based Operating Systems on page 70

Initializing the API Singleton

Initializing the API singleton configures the connection to the server, but does NOT establish the connection. Typically, only the `host` and the application key need to be modified, all other defaults can be used. For security purposes, the API defaults to rejecting self-signed certificates. If you choose to override this behavior, you can tell the API to allow them.

To initialize the API, use the **twAPI_initialize** function. Here is an example from the Steam Sensor example application (`main.c` file):

```
/* Initialize the API */
err = twApi_Initialize(hostname, port, TW_URI, appKeyCallback, NULL, MESSAGE_
CHUNK_SIZE,
MESSAGE_CHUNK_SIZE, TRUE);
if (TW_OK != err) {
TW_LOG(TW_ERROR, "Error initializing the API");
exit(err);
}
```

In this example, notice that this function no longer takes an application key variable. Instead, starting with v.2.2.0 of the C SDK, it uses the output of the **appKeyCallback** function. This callback function is called whenever the C SDK requires the current application key to authenticate with the ThingWorx platform. Here is its signature:
`void appKeyCallback(char* appKeyBuffer,unsigned int maxLength);`

where `appKeyBuffer` is an allocated buffer in which to copy the application key, and `maxLength` is the size of the buffer. Do not copy an Application Key that is longer than this buffer size into `appKeyBuffer`. This callback usage applies to all C SDK functions that require an Application Key as input.

⚠ **Caution**

In production, this callback should obtain an Application Key from a secure source.

The signature for the **twApi_Initialize()** function and definitions of its parameters can be found in the file, `twApi.h`.

Note

The **twApi_Initialize()** function initializes the Subscribed Properties Manager. You do not need to initialize this manager separately.

Version 2.0.0 of the C SDK added an `init` callback that is fired when the SDK is initialized by **twApi_Initialize**. The registration function **twApi_RegisterInitCallback** for this callback allows you to provide a `void*` 'user data' pointer, which is cached in a data structure owned by **twcfg**. This type of user data pointer is useful for integrating C with C++ because it allows a C++ static member function to invoke a private member via the C++ **this** pointer.

Simpler Initialization (C SDK v.2.0.0 and later)

Version 2.0.0 of the ThingWorx Edge C SDK has introduced initialization functions that simplify startup, namely `twExt_Start()` and `twExt_Idle()`. Located in the `/src/threadUtils/twThreadUtils.c` file in the C SDK installation, these functions provide simple thread management for C SDK functionality. Both functions can be called to establish the minimum number of support threads and services to manage an AlwaysOn connection to ThingWorx platform. The main difference between them is that `twExt_Idle()` will not return until your application is terminated. Any thread used to call this function will not exit. `twExt_Idle()` is useful in situations where your application only wants to start up services and then idle until it is exited. Call this function if you want this thread to take control of polling any registered polled functions.

`twExt_Start()` assumes that you are starting up AlwaysOn services as part of your application's normal startup process and need the calling thread to return once. The return is done to continue with operations that you may need to perform as part of your startup process. This function starts a thread to monitor all things with registered polled functions. Use this function if you want control of the calling thread to perfom other work inside your application. This function relies on the tasker to call polled functions on a thread that it creates. Here is the function signture from `src/threadUtils/twThreadUtils.h`:

```
 void twExt_Start(uint32_t dataCollectionRate, enum twThreadingModel
threadingModel,
 uint32_t messageHandlerThreadCount);
```

where:

- `intervalMsec` is the polling period in milliseconds.
- `threadingModel` is the threading model that you want to use.
- `messageHandlerThreadCount` is the number of message handling threads to spawn.

The signature of the `twExt_Idle()` function from `twThreadUtils.h`:

```
 void twExt_Idle(uint32_t intervalMsec, enum twThreadingModel
threadingModel, uint32_t messageHandlerThreadCount);
```

where:

- `intervalMsec` is the polling period in milliseconds.
- `messageHandlerThreadCount` is the number of message handling threads to spawn.

- `twThreadingModel` specifies which threading model you want to use:
    - `TW_THREADING_SINGLE`— Use the thread on which this function is called to service registered polled functions.
    - `TW_THREADING_TASKER` — Use the built-in tasker functionality of the C SDK to call all polled functions.

Both the `twExt_Start()` and `twExt_Stop()` functions set up periodic calls to any polled functions that you declare and that will be bound to specific Thing Shapes or Thing Templates. Often referred to as "Process Scan Request" functions, periodic polled functions can be declared against any Edge Thing Shape or Edge Thing Template, using the functions, `twExt_RegisterPolledTemplateFunction()` and `twExt_RegisterPolledShapeFunction()`. These periodic polled functions can be used to generate simulated data or to poll hardware for new data in your Thing or Shape. For details about these functions, see `threadUtils.c` and `threadUtils.h` in the `src/threadUtils` subdirectory of your C SDK installation.

The `twExt_Stop()` function shuts down all threads associated with your current threading model. Call `twExt_Stop()` before calling `twApi_Disconnect()`. Here is the signature of the `twExt_Stop()` function from `twThreadUtils.h`:

```
int twExt_Stop()
```

For details about each of these functions, see the `twThreadUtils.h` and `twTrheadUtils.c` files in the C SDK installation directory, `../src/threadUtils`.

Registering Properties and Services

Registering properties and services with the API accomplishes two things:

1. Tells the API what callback function to invoke when a request for that property or service comes in from ThingWorx platform.
2. Gives the API information about the property or service so that when ThingWorx browses the Edge device, it can be informed about the availability and the definition of that property or service.

To register services and properties, follow these examples:

```
/* Register our services */
ds = twDataShape_Create(twDataShapeEntry_Create("a",NULL,TW_NUMBER));
twDataShape_AddEntry(ds, twDataShapeEntry_Create("b",NULL,TW_NUMBER));
twApi_RegisterService(TW_THING, thingName,
  "AddNumbers", NULL, ds, TW_NUMBER, NULL, addNumbersService, NULL);


/* Register our properties */
twApi_RegisterProperty(TW_THING, thingName,
 "InletValve", TW_BOOLEAN, NULL, "ALWAYS", 0, propertyHandler, NULL);
twApi_RegisterProperty(TW_THING, thingName,
 "Pressure", TW_NUMBER, NULL, "ALWAYS", 0, propertyHandler, NULL);
twApi_RegisterProperty(TW_THING, thingName,
 "BigGiantString", TW_STRING, NULL, "ALWAYS", 0, propertyHandler, NULL);
```

For more information about using the callbacks, refer to the section, Server-Initiated Interactions on page 92.

Registering Events

Events do not have callbacks because they cannot be invoked from ThingWorx platform as a request to the edge device running your application. For your application to report events back to ThingWorx platform, use the `twApi_RegisterEvent` function to register the events. For more information about the function, refer to the Doxygen documentation that accompanies the C SDK.

Binding Your Entities

For an edge device to communicate with ThingWorx platform, its application must bind with the server, effectively establishing the connection with the RemoteThing that represents the edge device on ThingWorx platform. Bind each entity (Thing) so that when the API connects (and reconnects) to the server, it will announce that your entity is connected and available for interaction. The API supports unbinding entities so transient “Things” are supported.

To bind an entity, use its `thingName`, as shown here:

```
/* Bind our thing */
twApi_BindThing(thingName);
```

⚠ **Caution**

The ThingWorx Edge .NET SDK uses the function, `twApi_BindThingWithoutDefaultServices`, internally. Do NOT use this function for C SDK development. It will not register handlers for `GetMetadata` or `NotifyPropertyUpdate`. As a result, it will not properly bind things to ThingWorx platform.

Bulk Binding

To bind multiple entities at the same time, use the API that enables bulk binding, `twApi_BindThings(twList * entityNames)`. With this function, you provide a list of remote things when you want to bind multiple remote devices using a single call. To avoid any attempt to send a message larger than the configured maximum message size, the SDK checks the size of the message and generates an error when the maximum message size is reached. Once it detects the error, it sends the current message and starts a new one.

Initializing the File Manager (Optional)

If using the directory browsing and file transfer capability of the SDK, perform the following steps:

1. Set the staging directory — You must set the staging directory before initializing the FileManager. The default directory of the FileManager is most likely owned by root and will require a change to either the location of the staging directory and the ownership of the staging directory, or running the application as a user with the correct permissions. For example:

   ```
   /* Staging Directory Variable */
   /*  must be set before initializing file manager*/
   twcfg.file_xfer_staging_dir=”/home/user/stagingdir”;
   ```

2. Initialize the FileManager singleton. For example:

   ```
   /* Initialize the FileTransfer Manager */
   twFileManager_Create();
   ```

3. Define any virtual directories — Virtual directories allow you to expose only a subset of the entire file system of the device to the server for browsing and file transfer. This restriction is for both performance and security reasons. For example:

   ```
   /* Create our virtual directories */
   twFileManager_AddVirtualDir(thingName, "tw", "/opt/thingworx");
   twFileManager_AddVirtualDir(thingName, "tw2", "/twFile_tmp");
   ```

   Registering a virtual directory with the FileManager consists of mapping a unique name to an absolute path of a directory in your file system. Note that all subdirectories of the specified directory in the file system will be exposed to the server. Multiple virtual directories can be defined and there is no requirement that they be contiguous.

4. Register the `FileCallback` function that was previously defined so that the FileManager will call that function when any file transfer events occur. You can provide a wildcard filter so that only file transfer events of files that match the filter call the callback function. In addition, callbacks can be set up as “one-shots” such that the callback is unregistered automatically after it is invoked the first time. For example:

   ```
   /* Register the file transfer callback function */
   twFileManager_RegisterFileCallback(fileCallbackFunc, NULL,
   FALSE, NULL);
   ```

5. OPTIONAL: By default, WebSocket compression will be used for all WebSocket communications, including file transfers. If you need to disable compression, call the `tw_Api_DisableWebSocketCompression()` function. Note that this function does not return anything. In general, compression can be disabled after the API is initialized by calling the function:

```
twApi_DisableWebSocketCompression ();
```

When you disable compression, compression will not be used for any WebSocket communications, including file transfers.

Note

The C SDK v.2.1.2 provides improved logging for staging directory failures. If a file transfer fails to complete when the staging directory is on a separate partition (RAMDisk) from the final destination, use a DEBUG build of our application to see the logging around this failure (in `twListEntities()`).

Initializing the Tunnel Manager (Optional)

If using the tunneling capability of the C SDK you must create `#define ENABLE_TUNNELING`. A tunnel manager singleton is automatically created for you when you initialize the API. If you wish to disable tunneling for any reason you may call `twTunnelManager_Delete()`. The tunnel manager may be started up again by calling `twTunnelManager_Create()`. Once the tunnel manager is running you may register any callback functions. Passing a NULL for the id parameter when registering a callback will result in callbacks for all tunnel events.

```
/* Register the tunnel callback function */
twTunnelManager_RegisterTunnelCallback(tunnelCallbackFunc, NULL,
NULL);
```

When new tunnels are requested by the server, the tunnel manager creates a new tunnel. These tunnels establish an independent websocket back to the server. By default these websockets connect back to the same host/port that the API's websocket uses as well as the same TLS certificate validation criteria. You can override these defaults by using the built-in tunnel manager functions as found in the file, `twTunnelManager.h`:

```
int twTunnelManager_UpdateTunnelServerInfo(char * host,
   uint16_t port, appKeyCallback);
void twTunnelManager_SetProxyInfo(char * proxyHost, uint16_t proxyPort,
     char * proxyUser, twPasswdCallbackFunction proxyPassCallback);
void twTunnelManager_SetSelfSignedOk(char state);
void twTunnelManager_EnableFipsMode(char state);
void twTunnelManager_DisableCertValidation(char state);
void twTunnelManager_DisableEncryption(char state);
void twTunnelManager_SetX509Fields(char * subject_cn, char * subject_o,
   char * subject_ou, char * issuer_cn,
   char * issuer_o, char * issuer_ou);
void twTunnelManager_LoadCACert(const char *file, int type);
void twTunnelManager_LoadClientCert(char *file);
void twTunnelManager_SetClientKey(const char *file, char * passphrase,
int type);
```

Notice that output of two callback functions is used by the **TunnelManager** for security:

- The **twTunnelManager_UpdateTunnelServerInfo()** function uses the output of the `appKeyCallback()` function, which is represented as `appKeyCallback`.
- The **twTunnelManager_setProxyInfo()** function uses the output of the **twPasswdCallbackFunction**, which is represented here by `proxyPassCallback`.

This change is present in v.2.2.0 and later versions of the C SDK. For details about the **appKeyCallback** function, see the explanation below the first example code listing in "Initializing the API Singleton" on page . The password callback is explained in the section, Passwords (C SDK 2.2.0 and later) on page 64.

Is the Built-in Tasker Function Used?

If you are using the built-in tasker, continue to the next section. However, if you are not using the built-in tasker, you must call the function `twTunnelManager_TaskerFunction` on a very frequent basis (the examples default to 5 ms). The setting to use here is highly dependent on use case and environment. Before using this function, read the section, .twTunnelManager_TaskerFunction and Tick Resolution on page 63.

Connection Information for the Tunnel Manager

By default the `twTunnelManager` uses the same `twConnectionInfo` structure as `twApi` so that all `twConnectionInfo` settings should be shared by **twApi** and `twTunnelManager`. However, if `twTunnelManager_EnableFipsMode` is called, a new `twConnectionInfo` structure is allocated for the `twTunnelManager`, assigned the current values of the `twConnectionInfo` structure of `twApi`, and then updated by this function. Therefore, after this function is called, any `::twConnectionInfo` settings applied to the `::twApi` are *not* reflected in the `::twTunnelManager`'s connection structure.

twTunnelManager_TaskerFunction and Tick Resolution

Tunnel performance can be greatly affected by the thread’s `tick_resolution` of the `twTunnelManager_TaskerFunction`. When the tunnel manager thread is being created, the tick resolution determines how fast a tunnel manager checks the status of its managed tunnels. The smaller this value, the faster the tunnel responds. Tick resolution is especially important when running multiple tunnels concurrently, but be aware that a smaller tick resolution consumes more CPU resources. For an example, see the example application called “SteamSensorWithThreads.” See also Running the C SDK on Windows-based Operating Systems on page 70

Tunneling and Proxy Servers

When using a proxy server, you must set both the initial proxy with `twApi_SetProxyInfo` AND the proxy for tunneling with `twTunnelManager_SetProxyInfo`. Otherwise, the tunneling will fail. To set up the initial proxy, see Proxy Server Authentication on page 76.

Use the following function to set up communication through a proxy server for tunneling:

```
twTunnelManager_SetProxyInfo(char * proxyHost, uint16_t proxyPort,
    char * proxyUser, twPasswdCallbackFunction proxyPassCallback);
```

The following table lists and describes the parameters you can specify:

Parameter	Description
`proxyHost`	The IP address or host name of the proxy server to use for tunneling. .
`proxyPort`	The number of the port on the proxy server to use.
`proxyUser`	If the proxy server requires Basic or Digest authentication, you need to use the callback function, **twPassword**
`passwdCallback`	For a password, you must provide a password callback function so the ThingWorx SDK can obtain a copy of the tunnel password from your application. See the next section for more information about passwords.

Passwords (C SDK 2.2.0 and later)

As of v.2.2.0 of the C SDK, password protection is your responsibility. For the C SDK to use a password, you need to develop a way for the password to be provided to the following callback function:

```
typedef void (*twPasswdCallbackFunction)(char * passwdBuffer,
unsigned int maxPasswdSize);
```

As a result, the **twTunnelManager_SetProxyInfo** function no longer takes password variable. Instead, starting with v.2.2.0 of the C SDK, it uses the output of **twPasswdCallbackFunction**. This callback function is called whenever the C SDK requires the current password for a proxy server user or a digest to authenticate with the ThingWorx platform.

Caution

In production, this callback should obtain a password or digest from a secure source.

Tunnel Manager and OpenSSL

When the tunnel manager is initialized, it points its `{{tm->info}}` struct at `{{tw_api->connectionInfo}}`, so that any API settings are realized in the tunnel manager. However, if you call any functions that set tunnel manager settings (such as `{{twTunnelManager_DisableCertValidation()}}` or `{{twTunnelManager_EnableFipsMode()}}`), then the C SDK will actually create a new struct to set the tunnel manager specific settings and any subsequent calls to set API connection information (like loading a cert) will no longer be realized in the tunnel manager. See also the API documentation for the C SDK.

Creating a Bind Event Handler (Optional)

You may want to track exactly when your edge entities are successfully bound to or unbound from ThingWorx platform. The reason for this is that only bound items should be interacting with ThingWorx platform and it will never forward a request to a corresponding remote thing in its database when the request is targeted at an entity that is not bound.

```
/* Register a bind event handler */
/* Callbacks only when thingName is bound/unbound */
    twApi_RegisterBindEventCallback(thingName, BindEventHandler,
NULL);

/* First NULL says "tell me about all things that are bound */

/* twApi_RegisterBindEventCallback(NULL, BindEventHandler, NULL
```

Using the Utilities of the C SDK

The `/src/utils` subdirectory of the C SDK provides several utilities that you may find useful in your applications:

- Utilities that support proxy servers:
    - `HTTP Proxy` — This utility allows you to open a socket and connect to ThingWorx platform through the HTTP proxy server specified in the `twSocket` structure. The functions provided support establishing, authenticating, and otherwise managing connections through an HTTP

  proxy server. The files that define and implement the functions are `twHttpProxy.h` and `twHttpProxy.c`.
  - `twNTLM` — This utility enables you to establish a connection to ThingWorx platform through an NTLM proxy server. The files that define and implement the functions are `twNtlm.h` and `twNtlm.c`.
- Utilities that enable you to use lists, maps (hashmaps), and dictionaries:
  - `list` — This utility provides functions for creating a linked list, adding entries to the list, updating values of entries in the list, removing entries from the list, iterating over entries in a list, and deleting lists. The linked lists that the `ForEach` function can iterate over include maps and dictionaries. Note that `twList` is dynamically sized, thread-safe, untyped, and doubly linked. See also Using Linked Lists on page 67
  - `twMap` — `twMap` is a pointer to an internally maintained data structure. The `twMap.h` file defines the prototypes for the functions that enable you to create a hashmap, add elements to a hashmap, retrieve an element from a hashmap, remove elements from a hashmap, delete a hashmap, search a map for an element and value, replace a value of an element, retrieve the number of elements in a map, and iterate over the elements in a hashmap, using the `twMap_Foreach` function. See Maps on page 67.
  - `twDict` — This is an abstraction that lets the C SDK decide if a list is implemented as a map or a list. This utility mightbe useful on systems with low memory. See Dictionaries on page 68.
- Utility that supports logging:
  - `twLogger` — The `twLogger.h` file defines the structures and prototypes for the functions that support the logging functionality of the C SDK, including the log level enumeration and the ability to turn on verbose logging for purposes of debugging. The `twLogger.c` file provides the functions that support logging with the C SDK.
- Utilities that provide encryption and string manipulation:
  - `crypto_wrapper` — This utility is a wrapper around the libtomcrypt DES encryption functions. The functions support DES encryption/ decryption, MD4 Message-Digest algorithm, and creation of a DES key.
  - `twString` — The string utilities enable an application to modify characters (strings), including changing the case (upper to lower and lower to upper) and copying a string. The

Using Linked Lists, Maps, and Dictionaries

The `twList` functions provide a set of utility features that you can use to perform the following tasks:

- Create / delete a linked list.
- Add a new entry to an existing linked list.
- Remove an existing entry from a linked list.
- Clear all entries from a linked list.
- Iterate over the entries in a linked list.

The `twList.h` file defines functions that support these activities. What is important to note is that when you want to iterate over a list, you should now use the `twList_ForEach()` function instead of `twList_Next()`, which is deprecated as of release 1.3.3 of the C SDK. The `twList_ForEach()` function provides a way to iterate over a specified list quickly and in a thread-safe manner. Further, it allows you to iterate over different kinds of structures, namely lists, maps, and dictionaries. It first determines whether the `list` passed in is a list or not and checks if `list->count == 0`. If these conditions are true, it exits. If it is a list, it locks the list, defines a `node` as the first entry in the list, and then iterates over the list, using the `listHandler`.

For more details about these functions, see the source files and/or the generated documentation that accompanies the C SDK. The generated documentation is located in the `/documentation` subdirectory of the C SDK installation.

Maps

The `twMap.h` and `twMap.c` files provide functions and mock list interfaces to perform the following tasks on a hashmap:

- Create a hashmap.
- Add a new element to a hashmap.
- Retrieve an existing element from a hashmap.
- Search for an element in a hashmap.
- Remove an existing element from a hashmap.
- Remove all elements from a hashmap.
- Free the memory associated with a hashmap.
- Delete a hashmap.
- Determine the current size of a hashmap.
- Iterate over the elements in a map.
- Replace the value of an element in a map.

For more details about these functions, see the source files and/or the generated documentation that accompanies the C SDK. The generated documentation is located in the `/documentation` subdirectory of the C SDK installation.

Dictionaries

The `twDict` is an abstraction that lets you treat lists and maps the same way. A `twDict` can be implemented with either a map or a list. Lists use less memory and are faster at inserting new items. Lists are slow for finding an item. Maps are slower to insert new items, but are much faster at finding items. Maps do use more memory. By providing the `twDict` abstraction, you can decide if you want a low memory implementation (list) to run in a smaller memory footprint or a higher performing implementation that will require more memory. This can be set at compile time with `twDictionaryMode tw_dictionary_mode=TW_DICTIONARY_MODE`, in `twDict.c` before compiling.

Note

This has not been tested in any mode other than `TW_DICTIONARY_MODE` as of version 1.4.0 of the C SDK.

For more information about `twDict`, see the source files and/or the generated documentation that accompanies the C SDK. The generated documentation is located in the `/doc` subdirectory of the C SDK installation.

Example

Here is a snippet that creates a dictionary called `list_Foreach`, containing three items. It uses the `twMap_forEach` to iterate over the dictionary.

```
list_Foreach = twDict_Create(NULL, NULL);
twDict_Add(list_Foreach,(void*)"A");
twDict_Add(list_Foreach,(void*)"B");
twDict_Add(list_Foreach,(void*)"C");
twDict_Foreach(list_Foreach, twDict_ForEach_ForEachHandler, (void *)
userData);}
```

Tip

It is recommended to use `twDict` instead of `tw_List`. Further, `twList_Next()` is deprecated; instead use the new `Foreach` iterator to process entries in a list or dictionary. The `Foreach` iterator significantly improves performance.

Connecting to the Server and Initiating Defined Tasks

Connecting to the server first and then initiating tasks is the preferable order, especially if your tasks are pushing data to the server. If you start the tasks earlier, they may attempt to send property updates or invoke services on the server before the connection is established. While reversing the order does not cause any lasting problems, it tends to keep the system very busy with retries before the connection is established.

The connection to the server is attempted and retried with the parameters specified to the **twApi_Connect()** function. By default, the API automatically reconnects using the same parameters if the connection is subsequently lost. This behavior can be overridden when the API is initialized by setting the *autoreconnect* parameter to `FALSE`.

Note

The default setting for **DEFAULT_SOCKET_READ_TIMEOUT** in `twDefaultSettings.h` is 500 ms. If a websocket read times out in the middle of reading a record, the SSL state is lost. As a result, the SDK tries to start read the record header again, and the operation fails. To detect this situation, check the log for the SDK for the error, `twTlsClient_Read: Timed out after X milliseconds`, and consider increasing the value of the **DEFAULT_SOCKET_READ_TIMEOUT**. You can change the setting at runtime by modifying the value of **twcfg.socket_read_timeout.**

The API also supports callback notifications when a connection is successfully made and when a connection is lost. The signature for “event callback” functions can be found in the file, `src/messaging/twMessaging.h`, and the task registration functions are found in the file, `twApi.h`.

```
/* Connect to server */
if (!twApi_Connect(CONNECT_TIMEOUT, twcfg.connect_retries)) {
/* Register our "Data collection Task" with the tasker */
twApi_CreateTask(DATA_COLLECTION_RATE_MSEC, dataCollectionTask);
}
```

Performance Tip - Socket Read Timeout

If you are experiencing slow performance during high traffic C SDK operations, it could be beneficial to decrease the `twcfg.socket_read_timeout`. This change will allow more blocked threads to access the receive socket to look for the message that they are expecting. While smaller values will lead to increased performance, it is important to keep in mind that the smaller the value of `twcfg.socket_read_timeout`, the higher the CPU usage. This increased CPU usage should be monitored, especially on power constrained (battery operated, for example) devices.

Running the C SDK on Windows-based Operating Systems

When running the C SDK on Windows-based operating systems, it is possible for the Windows OS to have a tick resolution that is higher that the tick resolutions requested by the C SDK. For example, the default Windows tick resolution is 15ms and the recommended API task tick resolution is 5ms. In this scenario the API thread executes only at the limit interval of 15ms instead of the requested 5ms interval. To achieve the best performance, it is recommended that the Windows tick resolution be changed, using the Windows API functions, to one half of the maximum sampling rate (Nyquist Sampling). Note that some systems will experience high CPU load due to the increased tick timer.

5

Setting Up Security

Using SSL/TLS for Security

OpenSSL provides a more secure and more-frequently updated library for securing your Edge applications than the Open Source axTLS library, which was previously provided with the ThingWorx Edge C SDK. As of v. 2.2.1 the ThingWorx C SDK distribution bundles include only the OpenSSL libraries and not the axTLS library. In addition, as of release 2.2.1, the non-FIPS distribution bundles include the OpenSSL 32– and 64–bit libraries, version 1.0.2q, which, on Windows platforms are based on the Visual Studio 2015 runtime library. The FIPS distribution bundles include the OpenSSL libraries v.1.0.2l, which are based on the Visual Studio 2012 runtime library.

The C SDK prints not only its version number but also the SSL/TLS library and version number being used. If FIPS is enabled, it includes that information as well.

Tip

For best security practices, use OpenSSL, which is provided in the distribution bundle.

The C SDK supports Apache Tomcat default ciphers up to and including Tomcat 8.0.33. Subsequent versions of Tomcat may *exclude* ciphers that are used in older versions of OpenSSL and therefore will prevent the ThingWorx C SDK from connecting to the server in question (a ThingWorx platform).

If you prefer to use your own security implementation, note that the C SDK provides wrapper functions that closely follow the OpenSSL API to make it easy to use in your applications. If you want to use another SSL/TLS implementation, you need to set up the C SDK to use your implementation by following the template provided in the file, `twTemplateSSL.h`, located in the subdirectory, `/src/tls`, of the C SDK installation. This file contains a template for an SSL/TLS wrapper layer for your SSL/TLS implementation.

Tip

The OpenSSL library supports client authentication for an application that you are developing with the C SDK.

Use of OpenSSL is the default setting when generating the make or project files using CMake. If you are using your own security implementation, it is possible to turn OpenSSL off and your implementation on. Here is an example of enabling a custom implementation and disabling OpenSSL:

```
cmake /path/to/tw-c-sdk -DUSE_CUSTOMSECURITY=ON -DUSE_OPENSSL=OFF
```

Caution

Using an insecure connection is strongly discouraged, especially in a production environment.

The first argument for `cmake` is always the path to the source directory.

Setting Up Secure Connections

By default the C SDK is set up to ensure the most secure connection possible, using the OpenSSL 1.0.2q library as the default library in the non-FIPS distribution bundles of the SDK and OpenSSL 1.0.2l in the FIPS versions of the SDK.

Note

Starting with release 2.2.1 of the C SDK, the non-FIPS distribution bundles provide version 1.0.2q of the OpenSSL 32– and 64–bit libraries, which on Windows platforms are based on the Visual Studio 2015 runtime library. The FIPS distribution bundles of this SDK provide the 32–bit OpenSSL libraries, v.1.0.2l, which are based on the Visual Studio 2012 runtime library.

For the most secure connection, set the `issuer` and `subject` fields of your server certificates before starting the connection by using the `twApi_SetX509Fields()` function. These settings mean that it will attempt to validate certificates and reject self-signed certificates.

Several functions are available to modify the default behavior and may provide some level of convenience during development, such as allowing self-signed certificates. These functions can be found in the file, `twApi.h`, and are as follows:

```
int twApi_SetProxyInfo(char * proxyHost, uint16_t proxyPort,
    char * proxyUser, twPasswdCallbackFunction proxyPassCallback);
void twApi_SetSelfSignedOk();
int twApi_EnableFipsMode();
void twApi_DisableCertValidation();
void twApi_DisableEncryption();
int twApi_SetX509Fields(char * subject_cn,
  char * subject_o, char * subject_ou,
    char * issuer_cn,
  char * issuer_o, char * issuer_ou);
int twApi_LoadCACert(const char *file, int type);
int twApi_LoadClientCert(char *file);
int twApi_SetClientKey(const char *file, twPasswdCallbackFunction
passphraseCallback, int type);
```

In the **twApi_SetProxyInfo()** function, note that this function no longer takes a variable for the proxy password. Instead, it uses the output of the **twPasswdCallbackFunction()**. For details, see Passwords (C SDK 2.2.0 and later) on page 64.

In the **twApi_SetClientKey()** function, note that this function no longer takes a variable for the passphrase. Instead it uses the output of the **twPasswdCallbackFunction()**. For details, see Passwords (C SDK 2.2.0 and later) on page 64.

Note

Although you may want to enable self-signed certificates for development purposes, make sure that you disable self-signed certificates and set up the proper certificates before putting your application into production. Modifying the most secure settings possible for production is NOT recommended.

The functions defined in `twTLS.h` can be used for any SSL/TLS connections that your application needs to make. These functions are the abstracted interface that sit on top of the underlying TLS implementation.

Consistent with the OpenSSL APIs, the C SDK uses a structure for an SSL/TLS context that manages all the SSL/TLS sessions, as well as a structure for an SSL/TLS session itself. In addition, the APIs expose several functions for operations. The definitions and functions are exposed with preprocessor definitions. For these details, refer to the Doxygen documentation provided with the SDK. The following table lists and briefly describes the structures and functions defined in `twTLS.h`.

Item	Description
`TW_SSL_CTX`	The SSL context structure as defined by the implementation.
`TW_SSL`	The SSL session structure as defined by the implementation.
`TW_SSL_SESSION_ID_SIZE`	The SSL session structure as defined by the implementation.
`TW_SSL_SESSION_ID_SIZE`	The size of an SSL session ID as defined by the implementation. This ID is used for session resumption.
`TW_GET_CERT_SIZE`	Returns the maximum number of certificates allowed by the implementation.
`TW_GET_CA_CERT_SIZE`	Returns the maximum number of CA certificates allowed by the implementation.
`TW_NEW_SSL_CTX`	Creates and initializes new instance of an `SSL_CTX`.
`TW_NEW_SSL_CLIENT(a,b,c,d)`	Creates and initializes a new instance of an SSL structure within the provided `SSL_CTX`. Parameters: • `a` — pointer to a `TW_SSL_CTX` structure.

Item	Description
	• `b` — a `TW_SOCKET_TYPE` value that is the descriptor of the socket to be used. The underlying socket should not be opened before calling this function. • `c` — session id. The session ID if session resumption is being used. The SDK does not use session resumption and sets this to `NULL`. • `d` — size of the session ID that was passed in.
`TW_HANDSHAKE_SUCCEEDED`	Returns a `Boolean (char)` value, TRUE if the SSL handshake succeeded and data can be securely exchanged, FALSE if otherwise.
`TW_SSL_FREE(a)`	Close any socket and free up any memory associated with an SSL session. Parameter: • `a` — pointer to the `TW_SSL` structure to free.
`TW_SSL_CTX_FREE(a)`	Free up any memory associated with an SSL context. Parameter: • `a` — pointer to the `TW_SSL_CTX` structure to free.
`TW_SSL_WRITE(a,b,c)`	Writes data out the secure connection. Parameters: • `a` — pointer to the `TW_SSL` structure to write to. • `b` — pointer to the buffer containing the data to write. • `c` — the amount of data to write. This result of this macro should contain the number of bytes sent, or a negative number if an error occurred.
`TW_SSL_READ(a, b, c, d)`	Reads data from the secure connection. Parameters: • `a` — pointer to the `TW_SSL` structure to read from. • `b` — pointer to the buffer that the data should be placed in. • `c` — the amount of data to read. • `d` — the number of milliseconds to wait while trying to read the desired amount of data. This result of this macro should contain the number of bytes read, or a negative number if an error occurred.
`TW_USE_CERT_FILE(a,b,c)`	Loads an X509 certificate in PEM or DER format from the file specified. Parameters: • `a` — pointer to the TW_SSL_CTX structure load the certificate into. • `b` — name of the file containing the certificate. • `c` — a password to access the certificate (if required).
`TW_USE_KEY_ FILE(a,b,c,d)`	Loads an encrypted key in PEM or DER format from the file specified. Parameters: • `a` — pointer to the `TW_SSL_CTX` structure to read from

Item	Description
	- `b` — name of the file containing the key - `c` — the type of key - `d` — a password to access the key.
`TW_USE_CERT_CHAIN_FILE(a,b,c)`	Loads a certificate chain in PEM or DER format from the file specified. Parameters: - `a` — pointer to the `TW_SSL_CTX` structure load the certificate into. - `b` — name of the file containing the certificate chain. - `c` — a password to access the certificate (if required).
`TW_SET_CLIENT_CA_LIST(a,b)`	Sets the list of supported CAs from the file specified. Parameters: - a — pointer to the `TW_SSL_CTX` structure load the certificate into. - b — pointer to the CA list.
`TW_VALIDATE_CERT(TW_SSL * ssl, char selfSignedOk)`	Inline function that validates the received certificate. Parameters: - `ssl` — pointer to the `TW_SSL` structure that has received the certificate - `selfSignedOk` — boolean, set to `TRUE` if self-signed certificates are allowed, `FALSE` if not. Default is `FALSE`. Returns zero if the certificate is valid, non-zero if not.
`TW_ENABLE_FIPS_MODE(a)`	Enables FIPS mode. Parameters: - `a` – pointer to the `TW_SSL_CTX` structure Returns zero if successful or an error code if FIPS is supported but enabling failed or `TW_FIPS_MODE_NOT_SUPPORTED` if the TLS layer does not support FIPS
`TW_GET_X509_FIELD(TW_SSL * ssl, char field)`	Inline function that gets the value of a field in the certificate. Parameters: - `ssl` — pointer to the `TW_SSL` structure that has received the certificate - `field` – `char`, the field to retrieve. Fields supported must be `SUBJECT_CN`, `SUBJECT_O`, `SUBJECT_OU`, `ISSUER CN`, `ISSUER_O`, `ISSUER_OU` Returns the value of the field, or `NULL` if the field is not found.

Proxy Server Authentication

The C SDK supports the following authentication options for communicating with ThingWorx platform through a proxy server:

- No authentication

- Basic authentication
- Digest authentication
- NTLM

As of v.2.2.0, the C SDK uses a callback function to retrieve a basic password or a digest when it needs to authenticate with a proxy server. The `twPasswords.h` file provides the signatures for this callback function:

```
typedef void (*twPasswdCallbackFunction)(char * passwdBuffer,
unsigned int maxPasswdSize);
```

where `passwdBuffer` allocates a buffer in which to copy the password or digest and `maxPasswdSize` is the size of the buffer. Do not copy a password or digest that is longer than `maxPasswdSize` into `passwdBuffer`.

⚠ **Caution**

In production, this callback should obtain a password or digest from a secure source.

Use the following function to set up communication through a proxy server:

```
int twApi_SetProxyInfo(char * proxyHost, uint16_t proxyPort,
  char * proxyUser, char * passwdCallback);
```

The following table lists and briefly describes the parameters you can specify:

Parameter	Description
`proxyHost`	The IP address or host name of the proxy server to use when connecting to the ThingWorx platform.
`proxyPort`	The number of the port on the proxy server to use.
`proxyUser`	If the proxy server requires Basic or Digest authentication, provide a user nameto present to the proxy server on connection. These credentials are only for the proxy server. They are not passed beyond the proxy server.
`passwdCallback`	For a password, you must use the **twPasswordCallbackFunction** to obtain an encrypted password from a source of your choosing and store the encrypted password in `passwdCallback`. For example, your application might request the credentials in a user interface or command line interface. The C SDK retrieves and uses the password when it must access the proxy server. It then zeroes out the `passwdCallback` and discards it from memory. See also Passwords (C SDK 2.2.0 and later) on page 64.

Tunneling with Proxy Servers

When using a proxy server, you must set both the initial proxy as shown above using `twApi_SetProxyInfo` AND the proxy for tunneling with `twTunnelManager_SetProxyInfo`. If you set only the initial proxy and not the tunneling proxy, the connection succeeds but the tunneling fails. See Tunneling with Proxy Servers on page 64 for more information about setting up the proxy server for tunneling.

FIPS Mode

Your application can use an embedded FIPS-140-2-validated cryptographic module (Certificate #1747; OpenSSL FIPS module version 2.0.2) running on all supported platforms per FIPS 140-2 Implementation Guidance section G.5 guidelines. The C SDK with FIPS requires the OpenSSL toolkit to be used in conjunction with the OpenSSL FIPS Object Module 2.0.2. Do not attempt to use any libraries other than the OpenSSL library provided with the C SDK. The current version of OpenSSL in the FIPS distribution bundle is 1.0.2l.

Note

Not all hardware platforms where applications written using the C SDK can support FIPS-140-2-validated cryptography. For example, on platforms based on IA32 architecture, the processor must support the SSE2 instruction set. The SSE2 instruction set is available in Intel x86 CPUs, starting with Pentium 4. The application log will have a message that FIPS-140-2-validated cryptography is enabled. If you enable it, be sure that your certificates include only FIPS approved encryption algorithms. The FIPS approved algorithms are AES, Triple-DES, RSA, DSA, DH, SHA1, and SHA2.

If the FIPS module is enabled and the application directly communicates with a Java-based SSL/TLS server (such as ThingWorx platform), the cipher suite list should include `!kEDH` (as shown below). Otherwise, ephemeral Diffie-Hellman (EDH) key exchange may fail:

```
<CipherSuites>DEFAULT:!kEDH</CipherSuites>
```

In addition, depending on the Java version, the Apache Tomcat server used by your ThingWorx platform may or may not be FIPS compliant:

- Java 7 — By default, the strong encryption ciphers necessary for the FIPS mode edge client to connect are NOT enabled. To enable them, you must add the following line to the Apache Tomcat `server.xml` configuration file's `<Connector>` tag:

  ```
  ciphers="TLS_RSA_WITH_AES_128_CBC_SHA256,
           TLS_RSA_WITH_AES_128_CBC_SHA,
           TLS_RSA_WITH_AES_256_CBC_SHA256,
           TLS_RSA_WITH_AES_256_CBC_SHA,SSL_RSA_WITH_RC4_128_SHA"
  ```

- Java 8 — By default, the strong encryption ciphers necessary for the FIPS mode edge client to connect ARE enabled. You do not need to modify the Apache Tomcat file.

By default in both Java 7 and Java 8, weak encryption ciphers are enabled. To disable weak encryption ciphers for running in FIPS mode, update the following two lines in the Java configuration file, `java.security`:

```
jdk.certpath.disabledAlgorithms=MD2, DSA, RSA keySize < 2048
jdk.tls.disabledAlgorithms=MD5, SHA1, DSA, RSA keySize < 2048
```

With weak encryption ciphers disabled, the FIPS mode edge client will connect to the server, but the non-FIPSmode edge client will NOT connect to the server.

For information about building your edge client with FIPS mode, see the section on building with FIPS mode enabled in the topic, How to Build with FIPS Mode Enabled on page 107.

Support for Cipher Suites

The C SDK supports the default cipher suites of Apache Tomcat up to and including Tomcat 8.0.33. Subsequent versions of Tomcat may exclude ciphers that are used by earlier versions of OpenSSL and therefore could prevent the C SDK from connecting to the server in question (a ThingWorx platform).

With OpenSSL, you can choose from 110 ciphers. For more information about the supported cipher suites, visit https://www.openssl.org/docs/man1.0.2/apps/ciphers.html.

Note

As of release 2.2.1 of the C SDK, axTLS is no longer provided in the distribution bundle. For best security practices, use OpenSSL. In addition, as of release 2.2.1, the version of OpenSSL in the non-FIPS 32– and 64–bit distribution bundles of the C SDK is 1.0.2q. The version of OpenSSL in the FIPS distribution bundle is 1.0.2l.

Custom Cipher Suites

As of v.2.1.2 of the C SDK, you can customize what cipher suites are used at run time through a C SDK parameter. Called `cipher_set`, this parameter has been added to the `twcfg` data structure of the C SDK. This parameter allows you to specify a string that contains your cipher suite configuration. This parameter is supported only for builds that are based on OpenSSL. When specifying a string, use the OpenSSL cipher list configuration format, which you can find at http://openssl.cs.utah.edu/docs/apps/ciphers.html#cipher_list_format.

If you do not specify any cipher suites, secure defaults are used. The default string is set in `twOpenSSL.h` as follows:

```
#define TW_SSL_DEFAULT_CIPHER_STRING
ALL:!aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!PSK:!DSS:!RC4:!SEED:!ADH:!IDEA:!3
DES:!SRP:!SSLv3
```

If FIPS mode is enabled, any configuration that you may have entered is ignored. Instead, the following configuration string is used:

```
TLSv1.2+FIPS:kRSA+FIPS:!eNULL:!aNULL
```

The file, `twNoTls.h`, sets the cipher suite to `null` because the functionality is not supported in the build.

Note

You will see a warning if the C SDK detects a different OpenSSL version being used at run time than the version with which the application was built.

A Note About Cipher Suites

If your application communicates with an instance of the ThingWorx platform that uses Java 1.7, the cipher suite list should include !kEDH (as shown below) to disable Ephemeral Diffie-Hellman ciphers. Otherwise, Ephemeral Diffie-Hellman (EDH) key exchange will fail, and your device application will be unable to connect to the ThingWorx platform.

```
<CipherSuites>DEFAULT:!kEDH</CipherSuites>
```

Debugging with GDB and OpenSSL on ARM Platforms

When debugging the C SDK with OpenSSL on ARM platforms, it is possible to receive a SIGILL with the default debugging configuration. Here is an example of the SIGILL message from GDB:

```
Program received signal SIGILL, Illegal instruction.
0x400864c0 in ?? () from /usr/lib/arm-linux-gnueabihf/libcrypto.
so.1.0.0
```

Continuing the execution should be enough to continue debugging. However, it is possible to write a custom handler in GDB that will automatically handle this during the debug process. Please refer to the official OpenSSL documentation for more information: https://www.openssl.org/docs/faq.html#PROG15 (entry #16).

Troubleshooting Connection Errors (C SDK v.1.4.0 and earlier)

If your Edge application uses an earlier version of the C SDK than 1.4.0 and it cannot establish a secure (SSL/TLS) websocket Connection using the C SDK, you may see one of the following errors:

- `Error initializing SSL connection`
- `twWs_Connect: Error restarting socket. Error 0.`

These errors are known to occur after the version of Tomcat used for ThingWorx platform has been upgraded to 8.0.35 or higher and versions of the C SDK earlier than 1.4.0 are in use in the application. The later versions of Tomcat have disabled RSA-based ciphers by default due to forward secrecy concerns. The ciphers supported by the axTLS libraries earlier than v.2.1.2.1 (the version in the C SDK 1.4.0) are disabled by default by this change to Tomcat. When a C SDK device tries to connect, an error “No compatible ciphers” is returned.

It is strongly recommended that you upgrade to the latest version of the C SDK and use the OpenSSL library provided in the distribution bundle. You then will need to rebuild your application. As of release 2.2.1, the axTLS libraries are no longer provided in the distribution bundle.

6

Using Edge Extensions

ThingWorx Edge SDK Extensions for the C SDK

Creating a Directory of Registered Shapes and Templates

Loading Shape Libraries

Tasks for EdgeThingShape and EdgeThingTemplate Constructors

Macros for the Edge Extensions

Macros That Take Actions

Macros to Create twPrimitives from C Primitives

Macros to Create Data Shapes and Single Columns

Macros to Create InfoTables for Data Shapes

Declaring Edge Things

Defining Aspects for Properties

Services

Events

Best Practices for Developing Edge Extensions

Examples of Using Edge Extensions with the C SDK

7

Advanced Use of Edge Extensions

Modifying Property Values at Runtime

Property Change Listeners

Applying EdgeThingShapes at Runtime

Inter-Shape Communication

Calling ThingWorx Platform Functions

Polling Updates for EdgeThingShapes

8

Interacting with ThingWorx

Basic Data Structures

Once your connection is alive and active, any requests made to the server for registered properties and services will automatically be forwarded to your application, and the appropriate callback function will be called. To push properties to the server, execute a service on another entity in the system, or trigger an event on the server. Helper functions are available for these actions. These functions are described in the section, SDK Application-Initiated Interaction on page 96.

Data in the C SDK are represented in the form of a `twPrimitive` structure. Collections of data values are represented in the form of a `twInfoTable` structure. Each of these structures is defined below and the API functions to access them are found in `src/messaging/twBaseTypes.h` and `twInfoTable.h`, respectively.

twPrimitive Structure

The `twPrimitive` structure is a form of a variant that can represent any of the base types supported in ThingWorx platform. The structure is defined in `src/messaging/twBaseTypes.h` as follows:

```
typedef struct twPrimitive {
  enum BaseType type;
  enum BaseType typeFamily;
  uint32_t length;
  union {
         int32_t integer;
         double number;
         DATETIME datetime;
         twLocation location;
         char boolean;
         struct {
               char * data;
               uint32_t len;
         } bytes;
         struct twInfoTable * infotable;
         struct twPrimitive * variant;
  } val;
} twPrimitive;
```

The key fields are the `type` enumeration and the `val` union. The fields `typeFamily` and `length` are for internal API use and are typically not used by an application.

There are many helper functions for creating `twPrimitive` structures from base types in the ThingWorx C SDK, so that you will rarely have to create one manually. These function definitions can be found in `src/messaging/twBaseTypes.h`.

ThingWorx Base Types

The supported base types consist of the following:

Base Types

Base Type	Description
`TW_NOTHING`	An empty `val`.
`TW_STRING`	A modified UTF8 encoded string. Data and length are stored in `val.bytes` and `val.len`, respectively. The `twPrimitive` owns the data pointer and will free it when deleted. `TW_STRING` types are null terminated.
`TW_NUMBER`	A C double value, stored in `val.double`.
`TW_BOOLEAN`	Represented as a single `char`, stored in `val.boolean`.
`TW_DATETIME`	A DATETIME value, which is an unsigned 64 bit value representing milliseconds since the epoch 1/1/1970. Data is stored in `val.datetime`.
`TW_INFOTABLE`	A pointer to a complex structure (defined in the next section) and stored in `val.infotable`. The `twPrimitive` owns this pointer and will free up the memory pointed to when the `twPrimitive` is deleted.
`TW_LOCATION`	A structure consisting of three double floating point values – longitude, latitude, and elevation. Stored as `val.location`.
`TW_BLOB`	A pointer to a character array. Data and length are stored in `val.bytes` and `val.len`, respectively. Differs from `TW_STRING` in that the array may contain nulls. The `twPrimitive` owns the data pointer and will free it when deleted.
`TW_IMAGE`	Identical to `TW_BLOB` except for the type difference.
`TW_INTEGER`	Assigned 4 by integral value. Stored as `val.integer`.

Base Types (continued)

Base Type	Description
`TW_VARIANT`	Pointer to a structure that contain a `type enum` and a `twPrimitive` value. The pointer is stored as `val.variant`. The `twPrimitive` owns the pointer and will free the structure when deleted.
`TW_XML,TW_JSON, TW_QUERY, TW_HYPERLINK, TW_IMAGELINK, TW_PASSWORD, TW_HTML, TW_TEXT, TW_TAGS, TW_GUID,TW_THINGNAME, TW_THINGSHAPENAME, TW_THINGTEMPLATENAME, TW_DATASHAPENAME, TW_MASHUPNAME, TW_MENUNAME, TW_BASETYPENAME, TW_USERNAME, TW_GROUPNAME, TW_CATEGORYNAME, TW_STATEDEFINITIONNAME, TW_STYLEDEFINITIONNAME, TW_MODELTAGVOCABULARYNAME, TW_DATATAGVOCABULARYNAME, TW_NETWORKNAME, TW_MEDIAENTITYNAME, TW_APPLICATIONKEYNAME, TW_LOCALIZATIONTABLENAME, TW_ORGANIZATIONNAME`	These base types are all of the `TW_STRING` family and are stored similarly.

twInfoTable

The `twInfoTable` is a non-ordered collection, composed of columns and rows. Infotables serve as the primary mechanism for sending data to and from ThingWorx platform. Data that is sent to ThingWorx platform using an infotable is NOT guaranteed to be in any particular order. That said, the data shape used to specify an infotable supports an ordinal that can be used to display values in a specific order, but not through the C SDK. Note that infotable properties store field values by key, not by index. The properties should be referenced by key and not by index.

Structure of an InfoTable

A `twInfoTable` is essentially a self-describing collection of `twPrimitive` values. The structure of a `twInfoTable` follows:

```
typedef struct twInfoTable {
  twDataShape * ds;
  twList * rows;
  uint32_t length;
  TW_MUTEX mtx;
} twInfoTable;
```

The `ds` element is a pointer to a `twDatashape` structure that describes what each field (column) of the table is – its name, description, and the base type of that field. The base type of a field can be any one of the base types described in ThingWorx Base Types on page 88, including a `twInfoTable`, as the SDK and ThingWorx platform allow nesting of these tables.

The `rows` element is a pointer to a list of values. Each entry in the list is a pointer to a `twInfoTableRow` structure. The `twInfoTableRow` structure contains values for each of the fields described in the data shape and must contain the values in the same order as in the data shape. The number of rows in an `twInfoTable` is a 32-bit value and therefore only practically limited to how much memory you wish to allow the `twInfoTable` to consume.

The `length` and `mtx` elements of the `twInfoTable` structure are for internal use and are typically not accessed directly. All the pointer elements of an infotable are owned and managed by the `twInfoTable` and should not be deleted or freed on their own.

Creating a twInfoTable

Creating an`InfoTable` is a three step process, as follows:

1. Create your data shape and add any necessary entries (fields) to the data shape.

```
twDataShapeEntry * twDataShapeEntry_Create(const char * name,
    const char description, enum BaseType type);

twDataShape * twDataShape_Create(twDataShapeEntry * firstEntry);

int twDataShape_AddEntry(struct twDataShape * ds,
    struct twDataShapeEntry * entry);
```

⚠ **Caution**

You must create a data shape to hold the schema for the `twInfoTable` BEFORE creating the table. Once the table is created, data is added one row at a time. When a row is created, data must be added to the row in the same order that it is in data shape. If the data is not added in the correct order, the table does not form correctly. There is no warning about this, and it becomes evident only when a user attempts to view the data in ThingWorx Composer or a mashup that the data is being added incorrectly.

---

2. Create the `twInfoTable`, which requires its data shape to be passed in as a parameter.

```
twInfoTable * twInfoTable_Create(twDataShape * shape)
```

3. Add data to the `twInfoTable` by individually creating the rows and adding them to the it.

```
twInfoTableRow * twInfoTableRow_Create(twPrimitive * firstEntry)
int twInfoTableRow_AddEntry(twInfoTableRow * row, twPrimitive * entry)
int twInfoTable_AddRow(twInfoTable * it, twInfoTableRow * row)
```

Adding Ordinal to the Data Shape in the C SDK

If you want to display the data in an infotable in a particular order, you can add the ordinal aspect to the data shape, as follows:

1. Create the data shape entry, using `twDataShapeEntry_Create(<name>,<descr>,<BaseType>);`.
2. Create a `twPrimitive` to hold the ordinal value, using `twPrimitive_CreateFromInteger(<ordinal>);`.
3. Add the ordinal aspect, using `twDataShapeEntry_AddAspect(<DataShapeEntry pointer>,"Ordinal",<twPrimitive pointer>);`.
4. Create the data shape, using `twDatashape_Create()`, or add to an existing data shape, using `twDataShape_AddEntry()`, passing the `DataShapeEntry` pointer created above.

Helper Functions for InfoTables

One very common pattern is a `twInfoTable` that contains a single field and a single row, for example the current value of a single property. The API provides several helper functions that make it easy to create these simple tables, using just a single function call.

```
twInfoTable * twInfoTable_CreateFromString(const char * name,
   char * value, char duplicate);
twInfoTable * twInfoTable_CreateFromNumber(const char * name,
   double value);
twInfoTable * twInfoTable_CreateFromInteger(const char * name,
   int32_t value);
twInfoTable * twInfoTable_CreateFromLocation(const char * name,
   twLocation * value);
twInfoTable * twInfoTable_CreateFromDatetime(const char * name,
   DATETIME value);
twInfoTable * twInfoTable_CreateFromBoolean(const char * name,
   char value);
twInfoTable * twInfoTable_CreateFromPrimitive(const char * name,
   twPrimitive * value);
twInfoTable * twInfoTable_CreateFromBlob(const char * name,
   char * value, int32_t length, char isImage, char duplicate);
```

Accessing data contained in a `twInfoTable` is also easy with several helper functions defined to assist with the common usage patterns. You simply pass in the name of the field and which row you wish to retrieve the value from.

```
int twInfoTable_GetString(twInfoTable * it, const char * name,
    int32_t row, char ** value);
int twInfoTable_GetNumber(twInfoTable * it, const char * name,
    int32_t row, double * value);
int twInfoTable_GetInteger(twInfoTable * it, const char * name,
    int32_t row,int32_t * value);
int twInfoTable_GetLocation(twInfoTable * it, const char * name,
    int32_t row, twLocation * value);
int twInfoTable_GetBlob(twInfoTable * it, const char * name,
    int32_t row, char ** value, int32_t * length);
int twInfoTable_GetDatetime(twInfoTable * it, const char * name,
    int32_t row, DATETIME * value);
int twInfoTable_GetBoolean(twInfoTable * it, const char * name,
    int32_t row, char * value);
int twInfoTable_GetPrimitive(twInfoTable * it, const char * name,
    int32_t row, twPrimitive ** value);
```

Server-Initiated Interaction

To respond to requests for properties and services from the server, the API provides the property access and service access callbacks. The next two sections describe these callbacks, their parameters, and return values, and provide examples of using these callbacks.

Property Access Callbacks

Property access callbacks are the functions that are called when a request comes from the server to either read or write a specific property. These functions have the following signature:

```
enum msgCodeEnum myPropCallback (
    const char * entityName,
    const char * propertyName,
    twInfoTable ** value,
    char isWrite,
    void * userdata
)
```

The following table lists and describes the parameters:

Parameter	Type	Description
`entityName`	Input	Pointer to a character array. The name is represented as a modified UTF-8 string with the name of the entity targeted in this request. This parameter is guaranteed not to be null.
`propertyName`	Input	Pointer to a character array. This is the name of the property, represented in modified UTF-8. This value may be null or '*" which means the request is to return the value of all properties registered for this entity.
`value`	Input/Output	Pointer to a pointer to a `twInfoTable`. If this is a request to read the value of a property a new `twInfoTable` structure should be created and it pointer should

Parameter	Type	Description
		assigned to value. If this is a write, the value will contain a pointer to the infotable that contains the data to be written. This pointer is guaranteed to be non-NULL. In either case, the calling function will assume ownership of the pointer in `*value`, so the callback function does not need to worry about memory management of any infotables passed in or created and returned as values.
`isWrite`	Input	A Boolean value describing whether this is a read (`FALSE`) or write (`TRUE`) request for the property.
`userdata`	Input	The same pointer value that was passed in when this property was registered. This pointer can be used for anything. A typical use is to specify the `this` pointer when using C++ class wrappers.

The return value of the callback is an indicator of the success or failure of the function. You are free to choose any of the return codes defined in the `msgCodeEnum` enumeration type, defined in `src/api/twDefinitions.h`, starting with `SUCCESS` or any applicable larger value.

Below is a simple example of a property handler callback function.

```
enum msgCodeEnum propertyHandler(const char * entityName,
                                 const char * propertyName,
                                 twInfoTable ** value,
                                 char isWrite,
void * userdata) {
        char * asterisk = “*”;
   if (!propertyName) propertyName = asterisk;
   TW_LOG(TW_TRACE,"propertyHandler - Function called for Entity %
s,
     Property %s", entityName, propertyName);
        if (value) {
                if (isWrite && *value) {
                        /* Property Writes */
                        if (strcmp(propertyName,
"TemperatureLimit") == 0) {
               twInfoTable_GetNumber(*value, propertyName, 0,
                                     &properties.TemperatureLimit);
                        } else return NOT_FOUND;
                        return SUCCESS;
                } else {
                        /* Property Reads */
                        if (strcmp(propertyName,
"TemperatureLimit") == 0)
                      {*value = twInfoTable_CreateFromNumber
(propertyName,
                            properties.TemperatureLimit);
                        } else return NOT_FOUND;
                }
                return SUCCESS;
        } else {
                TW_LOG(TW_ERROR,"propertyHandler - NULL pointer
for value");
                return BAD_REQUEST;
        }
}
```

Service Callbacks

Service callbacks are the functions that are called when a request comes from ThingWorx platform to execute a service on a particular entity. These functions have the following signature:

```
typedef enum msgCodeEnum (*service_cb) (
                                const char * entityName,
                                const char * serviceName,
                                twInfoTable * params,
                                twInfoTable ** content);
```

The following table defines the parameters:

Parameters for msgCodeEnum()

Parameter	Type	Description
`entityName`	Input	Pointer to a character array. The name is represented as a modified UTF-8 string with the name of the entity targeted in this request. This parameter is guaranteed not to be NULL.
`serviceName`	Input	Pointer to a character array. This is the name of the service to be executed, represented in modified UTF-8. This parameter is guaranteed not to be NULL.
`params`	Input	Pointer to a `twInfoTable`. This is a pointer to an infotable that contains all of the parameters specified for this invocation of the service. This pointer may be NULL if the service in question has no input parameters. The API owns this pointer and

Parameters for msgCodeEnum() (continued)

Parameter	Type	Description
		will manage any memory associated with it.
`content`	Output	Pointer to a pointer to a twInfoTable. This is used to return any data the service returns back to the server. The callback function should create a twInfoTable as described previously and pass a pointer to that structure to *content. If the service does not return any data it is OK to set *content to NULL. The API will assume ownership of the pointer in *value, so the callback function does not need to worry about memory management of any infotables passed in or created and returned as values.
`userdata`	Input	The same pointer value that was passed in when this property was registered. This pointer can be used for anything, a typical use is to specify the 'this' pointer when using C++ class wrappers.

The return value of the callback is an indicator of the success or failure of the service call. You are free to choose any of the return codes defined in the `msgCodeEnum` enumeration type, defined in `src/api/twDefinitions.h`, starting with `SUCCESS` or any applicable larger value. Here is an example of a service handler callback:

```
enum msgCodeEnum addNumbersService(const char * entityName,
                                   const char * serviceName,
                                   twInfoTable * params,
  twInfoTable ** content,
  void * userdata) {
    double a, b, res;
    TW_LOG(TW_TRACE,"addNumbersService - Function called");
    if (!params || !content) {
      TW_LOG(TW_ERROR,"addNumbersService - NULL params or content
pointer");
      return BAD_REQUEST;
    }
    if (twInfoTable_GetNumber(params, "a", 0, &a) ||
        twInfoTable_GetNumber(params, "b", 0, &b)) {
            TW_LOG(TW_ERROR,"addNumbersService – Missing parameter
data");
            return BAD_REQUEST;
```

```
 }
    res = a + b;
    *content = twInfoTable_CreateFromNumber("result", res);
    if (*content) return SUCCESS;
    else return INTERNAL_SERVER_ERROR;
}
```

SDK Application-Initiated Interaction

The SDK provides functions to make it easy for an application to initiate interaction with ThingWorx platform. Assuming all the proper visibility, permissions, and other security aspects are correct, an entity built using the C SDK can read or write properties, create a list of subscribed properties, set values of subscribed properties, invoke services, and trigger events on itself or other entities in the system. The following sections describe the helper functions

Read a Property

This helper function retrieves the current value of a property of a specific entity on ThingWorx platform.

```
enum msgCodeEnum twApi_ReadProperty(enum entityTypeEnum
entityType,
                                    char * entityName, char *
propertyName,
                                    twPrimitive ** result, int32_t
timeout,
                                    char forceConnect)
```

The following table lists and describes the parameters for this helper function:

Parameter	Type	Description
`entityType`	Input	The type of entity that the property belongs to. Enumeration values can be found in `twDefinitions.h`
`entityName`	Input	The name of the entity that the property belongs to.
`propertyName`	Input	The name of the property to retrieve.
`result`	Input/Ouput	A pointer to a `twPrimitive` pointer. In a successful request, this parameter will end up with a valid pointer to a `twPrimitive` value. The caller is responsible for deleting the returned primitive using `twPrimitive_Delete`. It is possible for the returned pointer be a NULL if an error occurred.

Parameter	Type	Description
`timeout`	Input	The time (in milliseconds) to wait for a response from the server. A value of `-1` uses the `DEFAULT_MESSAGE_TIMEOUT` as defined in `twDefaultSettings.h`
`forceConnect`	Input	A Boolean value. If `TRUE` and the API is in the disconnected state of the duty cycle, the API will force a reconnect to send the request.

Return:

- `msgCodeEnum` — the result of the call. See `twDefinitions.h` for the enumeration definition.

Write a Property

This helper function writes a new value for a property of a specific entity on ThingWorx platform.

```
enum msgCodeEnum twApi_WriteProperty(enum entityTypeEnum
entityType,
char * entityName, char * propertyName,
twPrimitive * value, int32_t timeout, char forceConnect)
```

The following table lists and describes the parameters for this helper function:

Parameter	Type	Description
`entityType`	Input	The type of entity that the property belongs to. Enumeration values can be found in `twDefinitions.h`.
`entityName`	Input	The name of the entity that the property belongs to.
`propertyName`	Input	The name of the property to retrieve.
`value`	Input	A pointer to a `twPrimitive` that contains the value to set for the property. Once called, the calling function will retain ownership of this pointer and must manage the memory lifecycle. NOTE: The called function WILL alter the contents of this primitive, so the original contents cannot be relied upon after the function returns..

Parameter	Type	Description
`timeout`	Input	The time (in milliseconds) to wait for a response from the server. A value of `-1` uses the `DEFAULT_MESSAGE_TIMEOUT` as defined in `twDefaultSettings.h`.
`forceConnect`	Input	A Boolean value. If `TRUE` and the API is in the disconnected state of the duty cycle, the API will force a reconnect to send the request.

Return:

- `msgCodeEnum` — the result of the call. See `twDefinitions.h` for the enumeration definition.

Push Properties

Use this function to update one or more properties with a single message to ThingWorx platform. You can also use it to send multiple values of the same property to ThingWorx platform in a single message.

```
enum msgCodeEnum twApi_PushProperties(enum entityTypeEnum
entityType,
char * entityName, propertyList * properties, int32_t timeout,
 char forceConnect)
```

The following table lists and describes the parameters for this helper function:

Parameter	Type	Description
`entityType`	Input	The type of entity that the properties belong to. Enumeration values can be found in the file, `twDefinitions.h`
`entityName`	Input	The name of the entity that the properties belong to.
`properties`	Input	A pointer to a list of `twPrimitives`. The calling function will retain ownership of this pointer and is responsible for cleaning up the memory after the call is complete.

Parameter	Type	Description
`timeout`	Input	The time (in milliseconds) to wait for a response from the server. A value of `-1` uses the `DEFAULT_MESSAGE_TIMEOUT` as defined in `twDefaultSettings.h`
`forceConnect`	Input	A Boolean value. If `TRUE` and the API is in the disconnected state of the duty cycle, the API will force a reconnect to send the request.

Return:

- `msgCodeEnum` — the result of the call. See `twDefinitions.h` for the enumeration definition.

An example usage of the `twApi_PushProperties` function is as follows:

```
void sendPropertyUpdate() {propertyList * proplist =
      twApi_CreatePropertyList("FaultStatus",
      twPrimitive_CreateFromBoolean(properties.FaultStatus), 0);
if (!proplist) {
 TW_LOG(TW_ERROR,"sendPropertyUpdate: Error allocating property
list");
   return;
   }
  twApi_AddPropertyToList(proplist,"InletValve",
          twPrimitive_CreateFromBoolean(properties.InletValve),
0);
twApi_AddPropertyToList(proplist,"Temperature",
          twPrimitive_CreateFromNumber(properties.Temperature),
0);
  twApi_AddPropertyToList(proplist,"TotalFlow",
          twPrimitive_CreateFromNumber(properties.TotalFlow), 0);
  twApi_AddPropertyToList(proplist,"Pressure",
          twPrimitive_CreateFromNumber(properties.Pressure), 0);
  twApi_AddPropertyToList(proplist,"Location",
          twPrimitive_CreateFromLocation(&properties.Location),
0);
  twApi_PushProperties(TW_THING, thingName, proplist, -1, FALSE);
  twApi_DeletePropertyList(proplist);
}
```

Execute a Service

This helper function executes a service on a named entity on ThingWorx platform.

```
enum msgCodeEnum twApi_InvokeService(enum entityTypeEnum
entityType,
     char * entityName, char * serviceName,
twInfoTable * params, twInfoTable ** result, int32_t timeout,
    char forceConnect)
```

The following table lists and describes the parameters for this helper function:

Parameter	Type	Description
`entityType`	Input	The type of entity that the service belongs to. Enumeration values can be found in `twDefinitions.h`.
`entityName`	Input	The name of the entity that the service belongs to.
`serviceName`	Input	The name of the service to execute.
`params`	Input	A pointer to an infotable containing the parameters to be passed in to the service. The calling function will retain ownership of this pointer and is responsible for cleaning up the memory after the call is complete.
`result`	Input/Ouput	A pointer to a `twInfoTable` pointer. In a successful request, this parameter will end up with a valid pointer to a `twInfoTable` that is the result of the service invocation. The caller is responsible for deleting the returned primitive using `twInfoTable_Delete`. It is possible for the returned pointer be a NULL if an error occurred or no data is returned.
`timeout`	Input	The time (in milliseconds) to wait for a response from the server. A value of `-1` uses the `DEFAULT_MESSAGE_TIMEOUT` as defined in `twDefaultSettings.h`.
`forceConnect`	Input	A Boolean value. If `TRUE` and the API is in the disconnected state of the duty cycle, the API will force a reconnect to send the request.

Return:

- `msgCodeEnum`— the result of the call. See `twDefinitions.h` for the enumeration definition.

Trigger an Event

This helper function triggers a specific event on a named entity on ThingWorx platform.

```
enum msgCodeEnum twApi_FireEvent(enum entityTypeEnum entityType,
char * entityName, char * eventName,
twInfoTable * params, int32_t timeout, char forceConnect)
```

The following table lists and describes the parameters for this helper function:

Parameter	Type	Description
`entityType`	Input	The type of entity that the service belongs to. Enumeration values can be found in `twDefinitions.h`.
`entityName`	Input	The name of the entity that the service belongs to.
`eventName`	Input	The name of the event to trigger.
`params`	Input	A pointer to an infotable containing the parameters to be passed to the event. The calling function will retain ownership of this pointer and is responsible for cleaning up the memory after the call is complete.
`timeout`	Input	The time (in milliseconds) to wait for a response from the server. A value of `-1` uses the `DEFAULT_MESSAGE_TIMEOUT` as defined in `twDefaultSettings.h`.
`forceConnect`	Input	A Boolean value. If `TRUE` and the API is in the disconnected state of the duty cycle, the API will force a reconnect to send the request.

Return:

- `msgCodeEnum` — the result of the call. See `src/api/twDefinitions.h` for the enumeration definition.

9

Building a ThingWorx Edge C SDK Application

Use the information presented here to build the example applications as well as your own applications. As applicable, you can reuse build files from the examples or modify a build file to support a new platform.

See the following sections:

- Using CMake with ThingWorx C SDK Examples on page 103
- Building Applications with CMake on page 103
  - Configuring Options for a CMake Build on page 104
  - How to Build for Windows Platforms with CMake on page 107
  - How to Build for Linux Platforms with CMake on page 106
  - How to Build with FIPS Mode Enabled on page 107

Using CMake with ThingWorx C SDK Examples

The C SDK is a set of ANSI C header and source files that can be easily integrated into any build environment. As of version 1.4.0 of the C SDK, all sample applications in the `examples` directory provide `CMakeLists.txt` files for use with CMake. If you do not have CMake, download it from https://cmake.org/download/. When ready to generate make files or project files for building your application, create a subdirectory in the directory containing your source files. You will configure and run `cmake` within this directory.

Building Applications with CMake

CMake is an open-source, cross-platform set of tools that have been created to facilitate the tasks of building, testing, and packaging software. You can use CMake to control the compilation of your applications by using its platform- and compiler-independent configuration files and to generate native makefiles and projects to use in your compiler environment. The minimum required version of CMake to use with the ThingWorx Edge C SDK is 2.6. It is recommended that CMake be on your system path. If you installed CMake on Linux, it is on the system path already. If on Windows, you can use the CMake UI or, if using it from a Command Prompt, add CMake to your system PATH.

For complete information about CMake, visit the CMake web site, at https://cmake.org/. At the top of the home page, select **Resources ▸ Documentation** to find the training materials, reference documents for CMake, among other resources.

Note

It is STRONGLY recommended that you use one of the provided examples as a starting point for your custom application.

Configuring Options for a CMake Build

Note

The C SDK build process will now default to an OpenSSL-based build The axTLS is no longer available in the C SDK distribution. It is strongly recommended that you use the OpenSSL library provided with the C SDK. Release 2.2.1 of the C SDK include non-FIPS OpenSSL 32– and 64–bit libraries, version 1.0.2q, which, on Windows platforms is based on Visual Studio 2015 runtime library. The FIPS OpenSSL libraries of this SDK are still based on Visual Studio 2012 and therefore still are version 1.0.2l of the OpenSSL libraries.

The ThingWorx Edge C SDK provides a `CMakeLists.txt` file that shows the default values for the options for a C SDK application. Always check the statement at the top of the file, which indicates the minimum version of CMake that you can use.

You do not edit the `CMakeLists.txt` file to set options. Instead, when you generate a make file using `cmake`, use the `-D` argument with the command to set each desired option set. For example, to enable OpenSSL with FIPS mode, use the `cmake` command as follows:

```
cmake -DENABLE_OPENSSL=ON -DENABLE_FIPS_MODE=ON
```

Note

You can also edit CMake options at any time by editing them in the `CMakeCache.txt` files created when you generated your CMake build.

The following table lists and describes these options:

Option	Description	Default Value
`USE_OPENSSL`	It is recommended that you use the OpenSSL library that is provided with this SDK. By default CMake will build with this option.	`ON`
`ENABLE_TUNNELING`	Specifies whether your application uses the tunneling feature (to support remote sessions with a device that is	`ON`

Option	Description	Default Value
	running your application). If your application does not use tunneling, set this option to `OFF`.	
`ENABLE_FILE_XFER`	Specifies whether your application performs file transfers between the device and ThingWorx platform. If your application does not transfer files, set this option to `OFF`.	`ON`
`ENABLE_OFFLINE_MSG_STORE_RAM`	Specifies whether your application writes messages to RAM when the edge device is offline. Messages stored in RAM are lost if the device loses power. If you want messages to be persisted during a power outage, leave this option set to `OFF` and set the option, `ENABLE_OFFLINE_MSG_STORE_RAM`, to `ON`.	`OFF`

Option	Description	Default Value
`ENABLE_OFFLINE_MSG_STORE_FILE`	Specifies whether your application writes messages to a file when the edge device is offline. Messages stored in RAM are lost if the device loses power. If you want messages to be persisted during a power outage, leave this option set to `ON` and the option, `ENABLE_OFFLINE_MSG_STORE_RAM`, set to `OFF`.	`ON`
`ENABLE_FIPS_MODE`	If you are using the OpenSSL TLS library and want to use FIPS mode, set this option to `ON`.	`OFF`

How to Build for Linux Platforms with CMake

The following procedure assumes that you have downloaded and extracted CMake and that you have created a subdirectory where the output of `cmake` will be stored within the directory that contains your source files. To build for a Linux platform using CMake, follow these steps:

1. Navigate to the subdirectory that you created for running `cmake`.
2. If the parent directory contains your source code (as recommended), run `cmake ..`. If you want to use a toolchain other than the default, include the `-G` argument and specify the toolchain.
3. Run `make`.

CMake builds your application.

Example

For example, to build the C SDK using CMake

```
cd tw-c-sdk
mkdir buildoutput
cd buildoutput
cmake ..
make
```

Specifying a Custom Installation Directory for the C SDK

By default the C SDK headers and libraries install under `/usr/local/` when the `make install` command is run. To override this default location, use the CMake variable, `CMAKE_INSTALL_PREFIX` when running CMake. For example:

```
cmake .. -DCMAKE_INSTALL_PREFIX=/opt/thingworx/
```

Afterwards, running `make install` would install the C SDK to `/opt/thingworx/`.

How to Build for Windows Platforms with CMake

The following procedure assumes that you have downloaded and extracted CMake and that you have created a subdirectory where the output of `cmake` will be stored within the directory that contains your source files. To build for a Windows platform using CMake, follow these steps:

1. Navigate to the subdirectory that you created for running `cmake`.
2. Run `cmake -G` to see a list of all the IDE's that you can choose.
3. Run `cmake -G "Visual Studio 11 2012" ..`
4. Open `tw-c-sdk.sln` in Visual Studio.

How to Build with FIPS Mode Enabled

In addition to working with the OpenSSL library that is provided in the FIPS version of the C SDK, you need to run `cmake` with the argument, `-DENABLE_FIPS_MODE=ON`.

For more information about using FIPS mode, see FIPS Mode on page 78.

10

Porting to Another Platform

To port to a platform other than those that the SDK currently supports (with files specifically for the platforms), you'll need the information presented here. Included here is information about defining the OS, TLS support, and the various types of functions (logging, memory management, date/time, synchronization, socket).

Supporting New Platforms

If you are using a platform that is different than the provided options in CMake, CMake has its own custom toolchain support. Go to https://cmake.org/cmake/help/v3.6/manual/cmake-toolchains.7.html.

Requirements for Platforms

The ThingWorx Edge C SDK is designed for easy porting to even the most basic of platforms. The key requirements for the platform are as follows:

- ANSI C compiler and run time support
- TCP/IP stack
- Dynamic memory allocation (malloc, calloc, free)
- Millisecond granularity timer, preferably with a Real Time Clock
- Some form of Mutual Exclusion capability (Mutex, Critical Section, Spinlock, etc.)
- Tick Timer Interrupt/Callback capability (if using the built-in tasker)
- File System functions if using the File Transfer capability of the SDK
- Threads (optional)

All custom configurations for a platform are typically encapsulated in a single C source and header file pair. For example, the SDK comes with example ports for Windows and Linux (or any POSIX environment). The files are located in the porting directory and are `twWindows.h/twWindows.c` and `twLinux.h/twLinux.c` respectively. It is strongly recommended that you start with one of these files as the basis for your porting efforts. The Linux port will be used as an example in the sections that follow.

For the platforms supported by this release of the ThingWorx C SDK, refer to the *ThingWorx Edge Requirements and Compatibility Matrix*, which is available on the PTC Support site, Reference Documents page for ThingWorx products.

Defining the Chosen OS

Building with CMake

With CMake, you can build for a variety of operating systems. CMake automatically detects the native operating system and chooses the appropriate build tools.

If you would like to manually specify an IDE within the native operating system, see the CMake documentation for the list of supported generators.

If you would like to cross compile, see the CMake documentation on how to set up cross compiling with a specified toolchain. Some cross-compiling examples exist as the `PLATFORM` option in `CMakeLists.txt` file, but the required cross-compile tools must be downloaded separately from the ThingWorx C SDK.

If a new porting section is created, you need to add the section to the `Compiler and Linker` section of the `CMakeLists.txt` file. See the examples provided in the distribution bundle for the C SDK (each example has subdirectories for osx, linux, and win).

SSL/TLS Support

The C SDK has a pluggable security layer. As of release 2.0.0, it defaults to using the OpenSSL library that is provided with the SDK for full TLS 1.1 compliant certificate-based authentication and 128-bit AES encryption.

Good reasons for using the OpenSSL library include compliance with cipher suites supported by Tomcat (critical for connecting to an instance of ThingWorx platform), HW-based acceleration, and/or a need for a FIPS compliant implementation based on OpenSSL.

If you choose to use some library other than the provided OpenSSL library, point `TW_TLS_INCLUDE` to the required header file for your implementation. In addition, you need to set up the C SDK to use your implementation by following the template provided in the file, `twTemplateSSL.h`, located in the subdirectory, `/src/tls`, of the C SDK installation. This file contains a template for developing an SSL/TLS wrapper layer for your SSL/TLS implementation.

Note

The `NO_TLS` option will result in clear-text communications between your application and ThingWorx platform. The `NO_TLS` option is provided as a convenience for development purposes, but is NOT recommended for any production implementations. If you choose to use that setting you must also `#define NO_TLS`.

```
/***
#define TW_TLS_INCLUDE "twOpenSSL.h"
#define TW_TLS_INCLUDE "twNoTls.h"
#define NO_TLS
***/
```

Tip

A much more practical way to build without using HTTPS for development purposes is to use the function, **twApi_DisableEncryption()**. For details, see also the Doxygen documentation provided with the SDK bundle.

Logging Functions

The C SDK has a pluggable logging provider that defaults to simple `printf` statements. The function definition is in the `utils/twLogger.c` file. Your platform/OS specific header file also defines some macros for logging, as shown below.

```
/* Logging */
#ifdef _DEBUG
#ifndef DBG_LOGGING
#define DBG_LOGGING
#endif
#endif
#ifdef DBG_LOGGING
#define TW_LOGGER_BUF_SIZE 4096 /* Max size of log buffer */
#define TW_LOG(level, fmt, ...)   twLog(level, fmt, ##__VA_ARGS__)
#define TW_LOG_HEX(msg, preamble, length)   twLogHexString(msg, preamble,
length)
#define TW_LOG_MSG(msg, preamble)   twLogMessage(msg, preamble)
#else
#define TW_LOGGER_BUF_SIZE 1
#define TW_LOG(level, fmt, ...)
#define TW_LOG_HEX(msg, preamble, length)
#define TW_LOG_MSG(msg, preamble)
#endif
```

To minimize the code footprint of a released application, the default for logging is that it is enabled for debug builds and entirely disabled for release builds. Both the logging functions and buffer size need to be defined if logging is enabled. The macros `TW_LOG_HEX` and `TW_LOG_MSG` are used to display the hex bytes moving over the wire and the actual message content, respectively. These functions tend to have a serious impact on performance and are not recommended for use in a released system.

The logging system also provides a convenient way for you to define you own logging function without changing these macros. This function is

```
int twLogger_SetFunction(log_function f);
```

For details about this function, refer to the Doxygen documentation provided with the SDK bundle.

Memory Management Functions

The SDK uses dynamic memory allocation and de-allocation. In all but the most basic of platforms, this means the use of the standard C malloc, calloc, and free functions. The SDK does not use realloc itself, but any underlying TLS library may. To create an abstraction layer, the SDK uses `#defines` to give you the flexibility of creating your own implementations of these functions. These definitions, which are required, and their most basic implementations are as follows:

```
#define TW_MALLOC(a) malloc(a)
#define TW_CALLOC(a, b) calloc(a,b)
#define TW_REALLOC(a, b) realloc(a, b)
#define TW_FREE(a) free(a)
```

Date/Time Functions

The SDK requires a timer with millisecond granularity for things such as messaging timeouts and task scheduling. In addition, some form of real-time clock may be required if using `DATETIME` base types or the standard logging plugin. The `DATETIME` base type uses the standard javascript representation of milliseconds since the epoch of midnight 1/1/1970. In the Linux environment this is represented as an unsigned 64-bit integer with a direct correlation to the number of milliseconds, but the SDK makes no requirement that a `DATETIME` must be a simple element.

```
/* Time */
typedef uint64_t DATETIME;  /* AS DEFINED IN LINUX.H */
```

To support potentially complex `DATETIME` structures, a port of the SDK must provide a few `DATETIME` manipulation and comparison functions. The function definitions are in the file, `twOSPort.h`, but the implementations are typically in your OS-specific C file, or in the file, `twLinux.c` for a Linux port. The required functions are listed and described in the table that follows. For the signature and parameter definitions for the functions, refer to the Doxygen documentation provided with the SDK bundle.

Function	**Description**
`twTimeGreaterThan`	Compare two `DATETIME` entities and returns a value of `TRUE` if `t1` > `t2` or `FALSE` if not.
`twTimeLessThan`	Compare two `DATETIME` entities and returns a `TRUE` value if `t1` < `t2` or `FALSE` if not.
`twAddMilliseconds`	Add a number (msec) of milliseconds to the value in `t1`.
`twGetSystemMillisecondCount`	Get the current millisecond count since the system started (or since the epoch if the system time has millisecond granularity).

Function	Description
	On systems where the real-time clock has a millisecond granularity, it is recommended that this value be the same as the current system time, representing the current date/time.
`twGetSystemTime`	Get the current system time, representing milliseconds since the epoch. If `utc` is `TRUE` (the default for the SDK), the time is corrected to Universal Coordinate Time (UTC).
`twGetSystemTimeString`	Get the current system time and converts it to a string using `strftime` formatting.
`twGetTimeString`	Convert a `DATETIME` to a string using `strftime` formatting.
`twSleepMsec`	Delay execution. In a single-threaded, single-processor system, this may be a blocking call.

Synchronization Functions

The SDK may run in a multi-threaded or multitasking environment. Therefore, it is important to protect access to certain data structures. The functions described in the following table provide such access protection. While they may be stubbed out in a single-tasking environment, it is highly recommended that these functions be fully implemented with whatever facility your OS provides. Note that functions using the `TW_MUTEX typedef` assume that this will be a pointer to whatever structure or synchronization mechanism you wish to use.

Function	Description
`twMutex_Create`	Create a synchronization entity.
`twMutex_Delete`	Delete a synchronization entity and free up its memory.
`twMutex_Lock`	Lock the synchronization entity.
`twMutex_Unlock`	Unlock the synchronization entity.

For more information about these functions, refer to the Doxygen documentation provided with the SDK bundle.

Socket Functions

The C SDK does not include a TCP/IP stack. Rather, it assumes that the underlying platform provides that functionality. To that end, the SDK has defined a series of wrapper functions to mask the underlying native socket functions. The function definitions use an underlying `twSocket` structure that abstracts away some of the differences in how certain platforms deal with socket descriptors – for example, Linux uses an `int` while Windows uses a `HANDLE`. The structure is defined in the file, `src/porting/twOSPort.h`, as follows:

```
typedef struct twSocket {
        TW_SOCKET_TYPE sock;  /* socket descriptor  */
        TW_ADDR_INFO addr; /* address to use */
        TW_ADDR_INFO * addrInfo; /* Addr Info struct head - use to 
free */
        char state;
} twSocket;
```

The actual definition of and `TW_ADDR_INFO` and the implementation of the functions above should be done in your platform-specific C file. The following table lists and describes the socket functions that must be provided by a port. For signatures, parameter details, and return information, refer to the Doxygen documentation provided with the SDK.

Function	**Description**
`twSocket_Create`	Allocate and initialize a socket structure.
`twSocket_Connect`	Establish a connection to the specified host/port pair.
`twSocket_Reconnect`	Re-establish a connection to the specified host/port pair. The underlying socket will be torn down and recreated, but all other `twSocket` parameters should remain intact.
`twSocket_Close`	Close a previously opened connection.
`twSocket_WaitFor`	Check to see if data is available on a socket. Use this function to prevent a `twSocket_Read` call from blocking permanently if no data is available. This function is especially important if using the built-in tasker, which cannot have tasks that block.
`twSocket_Read`	Read data from a socket.
`twSocket_Write`	Write data to a socket.
`twSocket_Delete`	Delete a `twSocket` structure. This function should close the socket if it is still open before deleting the structure.
`twSocket_GetLastError`	Get the error code of the last error that occurred while using a socket. Note that this is typically a system-wide call and not a call to a specific socket.

Tasker Functions

The C SDK has a simple built-in tasker that can be used in conjunction with or in place of an underlying OS. The key requirement for the underlying architecture is to provide some sort of tick-timer that allows the execution of what could be a relatively long running callback function at one millisecond intervals. The callback function is `twTaskerStart`. This function initializes the tasker by setting up a mechanism to call the `tickTimerCallback` function every millisecond. This function call is blocking, so it is best to use some separate thread of execution, or at least re-enable priority interrupts before making this call. This function is called only once when a process using the API starts.

To shut down the `tickTimerCallback` mechansim, use the `twTaskerStop` function. Call this function only once when a process using the API ends.

For signatures, parameter details, and return information for these functions, refer to the Doxygen documentation provided with the SDK.

File System Functions

To use the file transfer or directory browsing capability of the C SDK, implement the functions listed in the following table. For signatures, parameter details, and return information, refer to the Doxygen documentation provided with the SDK.

Function	Description
`twDirectory_GetFileInfo`	Retrieve information about a directory entry (file or subdirectory).
`twDirectory_FileExists`	Check if a directory entry (file or directory) exists.
`twDirectory_CreateFile`	Create a file.
`twDirectory_MoveFile`	Move a file.
`twDirectory_DeleteFile`	Delete a file.
`twDirectory_CreateDirectory`	Create a directory.
`twDirectory_DeleteDirectory`	Delete the specified directory (and all its contents).
`twDirectory_IterateEntries`	Iterate through a directory, retrieving the information of the next file or subdirectory.
`twDirectory_GetLastError`	Retrieve the last error that occurred as a result of a file system activity.

Native Threads

With the built-in tasker, the C SDK has does not depend on a threading OS. However, if one is present, there are advantages to using native threads. Therefore, the C SDK provides a wrapper layer around native threads that maps tasks as defined for the built-in tasker to native threads. Porting the wrapper to a native threading model is straightforward and requires the implementation of only a few functions. These functions are defined in the file `src/porting/twThreads.h`.

The `twThread` structure follows:

```
typedef struct twThread {
     TW_THREAD_ID id;
     twTaskFunction func;
     uint32_t rate;
     char isRunning;
     char isPaused;
     char shutdownRequested;
     char hasStopped;
     void * opaquePtr;
}twThread;
```

The following table lists and describes the functions available for threads. For details, refer to the Doxygen documentation provided with the SDK.

Function	Description
`twThread_Create`	Create a new thread and optionally start it.
`twThread_Delete`	Stop a thread and free up the thread structure memory.
`twThread_Start`	Start a thread.
`twThread_Stop`	Stop a thread and optionally specify a number of milliseconds to wait for the thread to exit before forcefully killingit.
`twThread_Pause`	Pause the execution of a thread.
`twThread_Resume`	Resume the execution of a thread.
`twThread_IsRunning`	Check if a specified thread is running.
`twThread_IsPaused`	Check if a specified thread is paused.
`twThread_IsStopped`	Check if a specified thread is stopped.

A

Error Codes

This section lists and briefly describes the error codes that you may see when working with the ThingWorx Edge C SDK.

General Errors

The following table lists general errors and their corresponding codes:

Code	Message	Troubleshooting
100	`TW_UNKNOWN_ERROR`	An error occurred, but it was not recognized by the SDK. You should not see this error
101	`TW_INVALID_PARAM`	The parameter value is not allowed. Typically indicative of a NULL pointer being passed in where a NULL pointer is not allowed.
102	`TW_ERROR_ALLOCATING_MEMORY`	The specified amount of memory could not be allocated. Make sure that components free memory when they exit. Make sure you free up memory when finished using data structures. This error is very serious, and your application will usually terminate soon after.
103	`TW_ERROR_CREATING_MTX`	An error occurred while creating a mutex.
104	`TW_ERROR_WRITING_FILE`	An error occurred while writing to a file.
105	`TW_ERROR_READING_FILE`	An error occurred while reading a file sent from ThingWorx platform.
106	`TW_ERROR_ITEM_EXISTS`	The referenced entity already exists.
107	`TW_ERROR_ITEM_DOES_NOT_EXIST`	The referenced entity does not exist.
108	`TW_ERROR_SHAPE_DOES_NOT_EXIST`	The referenced data shape does not exist.
109	`TW_ERROR_TEMPLATE_DOES_NOT_EXIST`	The referenced Edge Thing Template does not exist.

Websocket Errors

A Websocket connection is run using a system socket; a system socket sits one layer lower in the networking stack. All Websocket errors indicate some general issue communicating with ThingWorx platform. The following table lists websocket errors, their corresponding codes, and an explanation of the issue.

Note

As of release 1.2 of the C SDK, the default setting for `DEFAULT_SOCKET_READ_TIMEOUT` in `twDefaultSettings.h` is 500 ms. If you are using axTLS (v.2.2.0 and earlier of the C SDK) and a websocket read times out in the middle of reading a record, the SSL state is lost. As a result, the SDK tries to start reading the record header again, and the operation fails. To detect this situation, check the log for the SDK for the error, `twTlsClient_Read:TimedoutafterXmilliseconds`, and consider increasing the value of the `DEFAULT_SOCKET_READ_TIMEOUT`. You can change the setting at runtime by modifying the value of `twcfg.socket_ read_timeout`.

Code	Message	Troubleshooting
200	`TW_UNKNOWN_WEBSOCKET_ERROR`	An unknown error occurred on the websocket. You should not see this error.
201	`TW_ERROR_INITIALIZING_WEBSOCKET`	An error occurred while initializing the websocket. Check your websocket configuration parameters for validity.
202	`TW_TIMEOUT_INITIALIZING_WEBSOCKET`	A timeout occurred while initializing the websocket. Check the status of the connection to ThingWorx platform.
203	`TW_WEBSOCKET_NOT_CONNECTED`	The websocket is not connected to ThingWorx platform. The requested operation cannot be performed.
204	`TW_ERROR_PARSING_WEBSOCKET_DATA`	An error occurred while parsing websocket data. The parser could not break down the data from the websocket.
205	`TW_ERROR_READING_FROM_WEBSOCKET`	An error occurred while reading data from the websocket. Retry the read operation. If necessary, resend the data.
206	`TW_WEBSOCKET_FRAME_TOO_LARGE`	The SDK is attempting to send a websocket frame that is too large. The Maximum Frame Size is set when calling `twAPI_`

Code	Message	Troubleshooting
		`Initialize` and should always be set to the Message Chunk Size (`twcfg.message_chunk_size`).
207	`TW_INVALID_WEBSOCKET_FRAME_TYPE`	The type of the frame coming in over the websocket is invalid.
208	`TW_WEBSOCKET_MSG_TOO_LARGE`	The application is attempting to send a message that has been broken up in to chunks that are too large to fit in a frame. You should not see this error.
209	`TW_ERROR_WRITING_TO_WEBSOCKET`	An error occurred while writing to the Web socket.
210	`TW_INVALID_ACCEPT_KEY`	The Accept key sent earlier from ThingWorx platform is not valid.

Messaging Errors

The following table lists the error codes and messages for Messaging errors and provides some troubleshooting information.

Code	Message	Troubleshooting
300	`TW_NULL_OR_INVALID_MSG_HANDLER`	The message handler singleton has not been initialized.
301	`TW_INVALID_CALLBACK_STRUCT`	The callback structure was not valid. Check that your application properly implements the callback.
302	`TW_ERROR_CALLBACK_NOT_FOUND`	The specified callback was not found. Check the callback parameters passed to the function.
303	`TW_INVALID_MSG_CODE`	An attempt to set an invalid message code was made. Valid message codes are defined in `twDefinitions.h`. You should not see this internal error in your code.
304	`TW_INVALID_MSG_TYPE`	A function was called with an invalid message code.

Code	Message	Troubleshooting
		Valid message codes are defined in `twDefinitions.h`. You should not see this internal error.
305	`TW_ERROR_SENDING_MSG`	An error occurred while sending the message. Check the network connections and the destination host. If network connections and the destination host are working properly, check the configuration of the destination host to be sure it is correct.
306	`TW_ERROR_WRITING_OFFLINE_MSG_STORE`	An error occurred while writing to the offline message store.
307	`TW_ERROR_MESSAGE_TOO_LARGE`	The message was too large. Check that the size you configured for messages is adequate for all expected traffic. Consider increasing the size.
308	`TW_WROTE_TO_OFFLINE_MSG_STORE`	The message was not sent to ThingWorx platform, but was stored in the offline message store. The message will be delivered next time the websocket is connected.
309	`TW_INVALID_MSG_STORE_DIR`	The directory for the message store was not correct. Make sure the path is valid and that you have write permission.
310	`TW_MSG_STORE_FILE_NOT_EMPTY`	The on-disk file that is uses to store offline messages contains some messages that have not been sent yet. The file name cannot be changed.

Code	Message	Troubleshooting
311	`TW_NULL_MSG_BODY`	The body of the message is empty. The message is not saved.
312	`TW_BIND_MESSAGE_FULL`	
313	`TW_NULL_OR_INVALID_OFFLINE_MSG_STORE_SINGLETON`	
314	`TW_ERROR_OFFLINE_MSG_STORE_FULL`	The offline message store has reached its maximum size.
315	`TW_ERROR_INFLATING_RECEIVED_MSG`	An error occurred while extracting a compressed file that was received from ThingWorx platform.

Primitive and InfoTable Errors

The following table lists the errors related to the data structures, `twPrimitive` and `twInfoTable`, and their supporting functions. It also provides suggestions for troubleshooting. For more information about these data structures, refer to twPrimitiveStructure on page 87 and twInfoTable on page 89.

Note

When creating an infotable, keep in mind that the `twInfoTableRow` structure must contain the field values of the data shape **in the same order** as in the data shape.

Code	Message	Troubleshooting
400	`TW_ERROR_ADDING_DATASHAPE_ENTRY`	An error occurred while attempting to add an entry (field) to the data shape.
401	`TW_INDEX_NOT_FOUND`	Attempted to access a non-existent field from a row in an infotable. The index value must be less than the number of fields defined in the data shape.

Code	Message	Troubleshooting
402	`TW_ERROR_GETTING_PRIMITIVE`	The function `twInfoTable_GetPrimitive` failed to retrieve the requested primitive from the infotable.
403	`TW_INVALID_BASE_TYPE`	The specified base type is not valid. Check the spelling in your code, or select a different base type. For a table of the available base types, refer to Base Types on page 88

List Errors

The following table lists the error related to lists (for example, subscribed properties):

Code	Message	Troubleshooting
500	`TW_LIST_ENTRY_NOT_FOUND`	The entry was not found in the list. For example, the requested property was not found in the list of subscribed properties.

API Errors

The following table lists the errors related to the API:

Code	Message	Troubleshooting
600	`TW_NULL_OR_INVALID_API_SINGLETON`	The API singleton is either null or invalid. This error occurs if the API was not initialized properly. Check the parameters that you are passing to the initialize function. Check the log.
601	`TW_ERROR_SENDING_RESP`	An error occurred while sending a response message to ThingWorx platform.
602	`TW_INVALID_MSG_BODY`	A message was received from ThingWorx platform that had an invalid or malformed message body.
603	`TW_INVALID_MSG_PARAMS`	A Property PUT was received from ThingWorx platform with an empty parameters infotable. The

Code	Message	Troubleshooting
		property value will not be changed.
604	`TW_INVALID_RESP_MSG`	The response message was not valid. You should not see this internal error.
605	`TW_NULL_API_SINGLETON`	The API singleton was null. This message indicates that the API was not initialized properly. Check the parameters that you are passing to the initialize function. Check the log.
606	`TW_ERROR_CREATING_MSG`	An error occurred while creating the message. This error typically indicates an out-of-memory condition.
607	`TW_ERROR_INITIALIZING_API`	An error occurred while initializing the API. Check the parameters that you are passing to the initialize function. Check the log.

Tasker Errors

The following table lists the errors related to the Tasker:

Code	Message	Troubleshooting
700	`TW_MAX_TASKS_EXCEEDED`	You have attempted to create more tasks than are allowed for the built-in tasker. The maximum number of tasks allowed is set at compile time with the constant `TW_MAX_TASKS` which is defined in `twDefinitions.h`. If you have many tasks running you may wish to consider using native threads if your platform supports them.
701	`TW_TASK_NOT_FOUND`	The specified task ID was not found. Make sure the task ID passed to this function is correct. The task ID is returned from the function call `twTasker_CreateTask`.
702	`TW_THREADING_MODEL_FAILED_SHUTDOWN`	The task thread failed to shut down.

Logger Errors

The following table lists the error related to logging:

Code	Message	Troubleshooting
800	`TW_NULL_OR_INVALID_LOGGER_SINGLETON`	The logger singleton was not initialized properly. This error indicates a memory allocation error. Check your `TW_LOGGER_BUF_SIZE` setting in your platform-specific header file in the `src/porting` directory.

Utils Errors

The SDK uses Base64 encoding/decoding. The following table lists the related errors. At this time, the code does not use them.

Code	Message
900	`TW_BASE64_ENCODE_OVERRUN`
901	`TW_BASE64_DECODE_OVERRUN`
910	`TW_ERROR_INITIALIZING_OFFLINE_MSG_STORE`

System Socket Errors

System Sockets are Operating System-provided networking APIs. The `TW_ERROR_WRITING_TO_SOCKET` error in the System Socket category is a general socket write error. All errors in this category are in the context of a connection to ThingWorx platform.

As appropriate, first check the network connection between the Thing where your application is running and ThingWorx platform to resolve the problem. If a proxy server is used between your edge device and ThingWorx platform, check that the proxy server is operating properly. If so, check the configuration for the connection to the proxy server.

Code	Message	Troubleshooting
1000	`TW_ERROR_WRITING_TO_SOCKET`	General socket write error encountered while writing to ThingWorx platform.
1001	`TW_SOCKET_INIT_ERROR`	An error occurred while initializing the socket. The network connection may have dropped.
1002	`TW_INVALID_SSL_CERT`	The SSL certificate provided by the server was not valid or was self-signed. Check your certificate settings.
1003	`TW_SOCKET_NOT_FOUND`	The socket was not found. The network connection may have dropped.
1004	`TW_HOST_NOT_FOUND`	The specified ThingWorx platform was not found. Check network connections and make sure that your application configuration specifies a valid host address.
1005	`TW_ERROR_CREATING_SSL_CTX`	An error occurred creating the SSL context.
1006	`TW_ERROR_CONNECTING_TO_PROXY`	An error occurred connecting to the specified proxy server. Make sure the proxy server address is correctly specified. Check network connections.
1007	`TW_TIMEOUT_READING_FROM_SOCKET`	An attempt to read from a socket timed out with no data available.

Code	Message	Troubleshooting
1008	`TW_ERROR_READING_RESPONSE`	An error occurred while reading the response from the proxy server. Check your proxy configuration in your application.
1009	`TW_INVALID_PROXY_CREDENTIALS`	The credentials presented to the proxy server were not valid. Check with the administrator for the proxy server and re-enter the credentials for the proxy server. NOTE: While the connection to the proxy server is not encrypted, the credentials are obfuscated using standard HTTP Basic, Digest, or NTLM encoding.
1010	`TW_UNSUPPORTED_PROXY_AUTH_TYPE`	The specified authentication type for the proxy server is not supported. Make sure that the authentication type is correctly specified in your application.
1011	`TW_ENABLE_FIPS_MODE_FAILED`	FIPS Mode could not be enabled. Ensure that you are using an OpenSSL library with FIPS validated cryptographic algorithms.
1012	`TW_FIPS_MODE_NOT_SUPPORTED`	FIPS Mode is not supported. Ensure that you are using an OpenSSL library with FIPS validated cryptographic algorithms.
1013	`TW_DISABLE_FIPS_MODE_FAILED`	FIPS mode could not be disabled. Ensure that the call to disable FIPS mode occurs during initialization and not later, after the application has started and connected to ThingWorx platform.

Message Code Errors

The message code errors can be returned when the SDK makes a request to ThingWorx platform. They can also be the return values for property/service requests executed by the application using the SDK. For example, if the server queried the SDK application for the property ‘temperature’, but the application did not have that property, it could return `TW_NOT_FOUND`. The server could also return the same code if the application asked the server for a property that it did not have defined.

Most of these are standard HTTP error codes. You can see more information about them at http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html.

Code	Message	Troubleshooting
1100	`TW_BAD_REQUEST`	The HTTP request contained syntax errors, so the server did not understand it. Modify the request before attempting it again..
1101	`TW_UNAUTHORIZED`	The request requires authentication. This error results from a failed login attempt — whether from credentials that were not valid or from the request being sent before authentication occurred.
1102	`TW_ERROR_BAD_OPTION`	An option or a parameter for a function has a value that is not valid or is not spelled correctly (and so is not recognized).
1103	`TW_FORBIDDEN`	ThingWorx platform is denying access to the requested resource. Check your permission settings on ThingWorx platform.
1104	`TW_NOT_FOUND`	This message is returned for anything that was not found — a property, a service, a thing, a data shape, and so on.
1105	`TW_METHOD_NOT_ALLOWED`	The specified method is not allowed. Check the spelling

Code	Message	Troubleshooting
		and syntax of your code.
1106	`TW_NOT_ACCEPTABLE`	Not acceptable.
1107	`TW_PRECONDITION_FAILED`	The precondition for the operation was not met.
1108	`TW_ENTITY_TOO_LARGE`	This error occurs if you attempt to send a Property, or Service or Event parameters that are too large for ThingWorx platform to handle.
1109	`TW_UNSUPPORTED_CONTENT_FORMAT`	This error occurs if you attempt to send a Property, or Service or Event parameter that has the wrong baseType as defined on ThingWorx platform.
1110	`TW_INTERNAL_SERVER_ERROR`	An error occurred on ThingWorx platform while processing this request.
1111	`TW_NOT_IMPLEMENTED`	ThingWorx platform may return this error if you attempt a function that is not implemented.
1112	`TW_BAD_GATEWAY`	A gateway could be bad if it cannot communicate to the next component in the chain.
1113	`TW_SERVICE_UNAVAILABLE`	The requested service is not defined. You could also use the `TW_NOT_FOUND` error code, but this one is more specific.
1114	`TW_GATEWAY_TIMEOUT`	If the application sends a request to ThingWorx platform and does not get a response within some amount of time, the service call results in this error. The amount of time is configurable.

Subscribed (Managed) Property Errors

The following table lists the errors related to subscribed properties and the Subscribed Properties Manager:

Code	Message	Troubleshooting
1200	`TW_SUBSCRIBEDPROP_MGR_NOT_INTIALIZED`	The Subscribed Properties Manager is initialized by `twApi_Initialize` automatically. For this error to occur, it is most likely that other, more serious errors have occurred. Investigate the other errors first.
1201	`TW_SUBSCRIBED_PROPERTY_NOT_FOUND`	The requested subscribed property was not found.
1202	`TW_PROPERTY_CHANGE_BUFFER_FULL`	The buffer that is storing the name/value pairs to be sent with a single push is full. If an attempt is made to add another property/value, the oldest name/value pair is dropped.
1203	`TW_SUBSCRIBED_PROPERTY_LIST_PERSISTED`	The list of subscribed properties remains in memory.
1204	`TW_SUBSCRIBED_PROPERTY_LIST_PERSIST_ERROR`	An error occurred when attempting to persist the subscribed property list.
1205	`TW_SUBSCRIBED_PROPERTY_LIST_UNABLE_TO_PERSIST_ERROR`	The SPM was unable to persist the list of property/value pairs.
1206	`TW_SUBSCRIBED_PROPERTY_SYNCHRONIZATION_TIMEOUT`	During a synchronization of subscripted properties, a timeout occurred. The synchronization did not complete.

File Transfer Errors

The following table lists the errors for the File Transfer component:

Code	Message	Troubleshooting
1300	`TW_FILE_XFER_MANAGER_NOT_INITIALIZED`	The File Transfer Manager has not been initialized. The File Transfer Manager is initialized when `twApi_Initialize` is called only if `ENABLE_FILE_XFER` is defined. If you wish to use file transfer functionality make sure `ENABLE_FILE_XFER` is defined.
1301	`TW_ERROR_CREATING_STAGING_DIR`	An error occurred while creating the staging directory. The error happens if there is an invalid path or if you do not have the proper permissions to create the directory specified.
1302	`TW_FILE_NOT_FOUND`	The specified file for the transfer was not found. Check the name of the file specified. If it is correct, check for the presence of the file in the file system at the specified location.
1303	`FILE_TRANSFER_FAILED`	The file transfer operation failed. The network connection may have dropped during the transfer, the destination for the transfer may be unavailable (down for maintenance or power outage), or the MD5 checksum of the file indicated invalid file content.

Tunneling Errors

The following table lists the errors related to the Tunneling Manager

Code	Message	Troubleshooting
1400	`TW_TUNNEL_MANAGER_NOT_INITIALIZED`	The Tunnel Manager has not been initialized. The Tunnel Manager is initialized when `twApi_Initialize` is called only if `ENABLE_TUNNELING` is defined. If you wish to use tunneling functionality make sure `ENABLE_TUNNELING` is defined.
1401	`TW_TUNNEL_CREATION_FAILED`	The tunnel was not created. This error could be because of an out-of-memory condition.

TLS Errors

The following table lists the error related to TLS security component configured for the SDK:

Code	Message	Troubleshooting
1501	`TW_TLS_ERROR_LOADING_FILE`	The specified certificate file could not be loaded. Check the path specified for the certificate and that the file is in the specified location.

B

Callback Function Return Codes

The following table contains the acceptable return codes (`msgCodeEnum`) for all Property and Service callback functions. These codes are defined in `src/api/twDefinitions.h`. The callback functions are invoked as a result incoming requests from ThingWorx platform. The property and service callback function signatures are defined in `src/api/twApi.h`.

Return Code	Returned When
HTTP Client Error Status Codes	
`TWX_SUCCESS` = `0x40`	0x40 (2.00) Success. The request completes successfully.
`TWX_BAD_REQUEST` = `0x80`	0x80 (4.00) Bad request. The HTTP request contains syntax errors, so the server cannot understand it. Modify the request before attempting it again.
`TWX_UNAUTHORIZED`	0x81 (4.01) Unauthorized. The request requires authentication. This error results from a failed login attempt — whether from credentials that were not valid or from the request being sent before authentication occurred.
`TWX_BAD_OPTION`	0x82 (4.02) Bad option. An option or a parameter for a function has a value that is not valid or is not spelled correctly (and so is not recognized).

Return Code	Returned When
`TWX_FORBIDDEN`	0x83 (4.03) Forbidden. ThingWorx platform is denying access to the requested resource. Check your permission settings on ThingWorx platform.
`TWX_NOT_FOUND`	0x84 (4.04) Not found. Anything is not found — a property, a service, a thing, a data shape, and so on.
`TWX_METHOD_NOT_ALLOWED`	0x85 (4.05) Method not allowed. The specified method is not allowed. Check the spelling and syntax of your code.
`TWX_NOT_ACCEPTABLE`	0x86 (4.06) Not acceptable.
`TWX_PRECONDITION_FAILED = 0x8C`	0x8C (4.12) Precondition failed. The precondition for the operation is not met.
`TWX_ENTITY_TOO_LARGE`	0x8D (4.13) Entity too large. An attempt is made to send a Property, or Service or Event parameter that is too large for ThingWorx platform to handle.
`TWX_UNSUPPORTED_CONTENT_FORMAT = 0x8F`	0x8F (4.15) Unsupported content format. An attempt is made to send a Property, or Service or Event parameter that has the wrong baseType as defined on ThingWorx platform.
HTTP Server Error Status Codes	
`TWX_INTERNAL_SERVER_ERROR = 0xA0`	0xA0 (5.00) Internal server error. An error occurs on ThingWorx platform while processing this request.
`TWX_NOT_IMPLEMENTED`	0xA1 (5.01) Not implemented. ThingWorx platform may return this error if you attempt a function that is not implemented.
`TWX_BAD_GATEWAY`	0xA2 (5.02) Bad gateway. A gateway could be bad if it cannot communicate to the next component in the chain.
`TWX_SERVICE_UNAVAILABLE`	0xA3 (5.03) Service unavailable. The requested service is not defined. You could also use the `TW_NOT_FOUND` error code, but this one is more

Return Code	Returned When
	specific.
`TWX_GATEWAY_TIMEOUT`	0xA4 (5.04) Gateway timeout. If the application sends a request to ThingWorx platform and does not get a response within some amount of time, the service call results in this error. The amount of time is configurable.
`TWX_WROTE_TO_OFFLINE_MSG_STORE`	Wrote to offline message store. The message is not sent to ThingWorx platform, but instead is stored in the offline message store. The message will be delivered next time the websocket is connected.

thingworx®