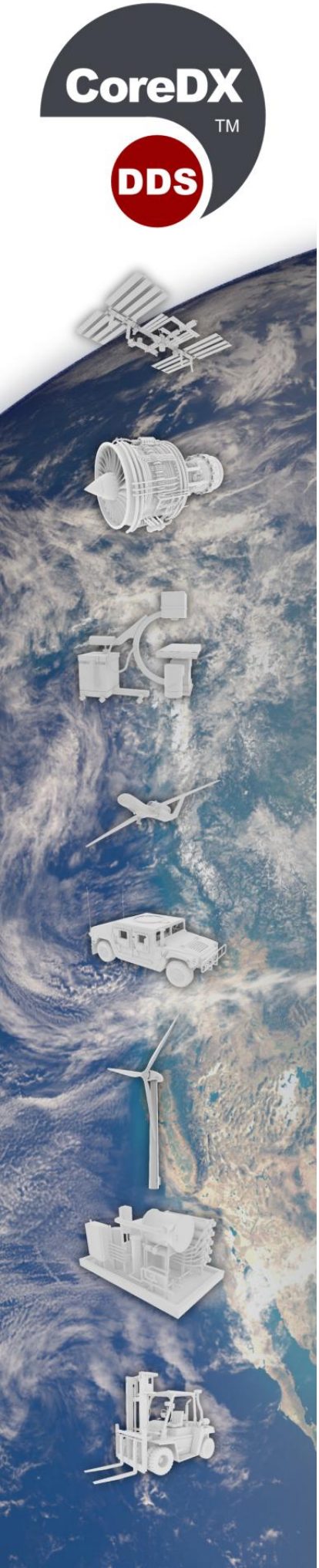


CoreDX DDS

Secure

Powerful protection for data communications

Programmer's Guide

Version 5.0

March 2020

TWINOAKS COMPUTING INC.
PRACTICAL MIDDLEWARE EXPERTISE

This document describes how to install and use the CoreDX DDS Secure software.

Preface

CoreDX DDS Secure is a secure version of CoreDX DDS - small-footprint, high-performance communications middleware compliant with the OMG Data Distribution Service (DDS) standard. CoreDX DDS Secure supports multiple hardware architectures and operating systems, and is intended to facilitate the development of robust, near real-time, highly distributed, and secure systems.

This manual describes how to install and use CoreDX DDS Secure. It is for developers who want to integrate a high-performance, OMG compliant data distribution middleware service into their application.

How this Guide is Organized

This document contains a number of Sections that describe the standardized DDS Security Plug-ins, the CoreDX DDS Security Plug-in implementation, and how to configure and use the CoreDX DDS Security plug-ins.

Related Documentation

CoreDX DDS Programmer's Guide

CoreDX DDS Reference Manuals

CoreDX DDS Type System

Intended Audience

This document is intended for software developers who are deploying CoreDX DDS Secure systems. The guide assumes that the reader is competent in programming languages and software development concepts. CoreDX DDS supports multiple programming languages, and this guide includes examples in C++.

Typographic Conventions

Typeface	Meaning	Examples

Courier	Example code	struct StringMsg
		{
		  string msg;
		};

Courier	Example Commands	gunzip –c coredx-2.x.tar.gz

Figure 0-1: Typographic Conventions

Feedback

Twin Oaks Computing welcomes your comments. We are interested in improving our products and we welcome your comments and suggestions. You can provide email feedback about this document to **documents@twinoakscomputing.com**.

Table of Contents

Table of Figures

Part 1: Introduction

This section provides an introduction of the Data Distribution Service (DDS) and the CoreDX DDS implementation from Twin Oaks Computing, Inc.

Chapter 1 An Introduction to CoreDX DDS

Welcome to CoreDX DDS, a high-performance, small-footprint implementation of the OMG Data Distribution Service (DDS) standard. The CoreDX DDS Data-Centric, Publish-Subscribe messaging infrastructure provides high-throughput, low-latency data communications.

This chapter provides an overview of the Data Distribution Service (DDS), how applications might use DDS to meet their communication requirements, and features of the CoreDX DDS product.

1.1 Why DDS?

Today's enterprise systems, embedded systems, and all systems in between, need flexible, open information systems. Most systems span multiple technologies, hardware platforms, operating systems, and programming languages. In addition, components of these systems have real-time requirements. CoreDX DDS is an open standards-based, communication middleware solution to meet the needs of these real-time distributed systems.

1.2 The case for Middleware

Middleware is a class of software that exists between an application and the Operating System. In deeply embedded environments, middleware exists between the functional software and a network stack of the device. It provides useful capabilities that are above and beyond those found in standard Operating Systems. In the case of CoreDX DDS, the middleware provides a facility for both publish-subscribe and client-server communications.Figure 1-1 illustrates where middleware components fit in the application, and how they logically bridge across multiple operating systems and hardware architectures.

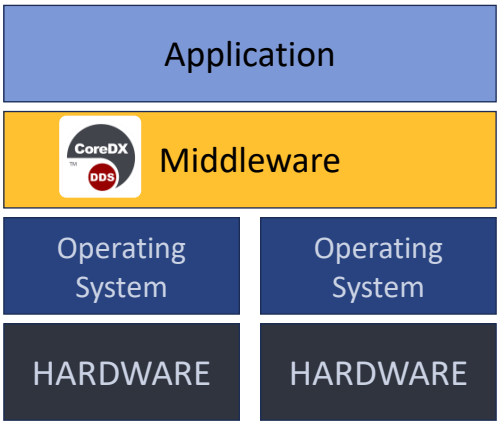


Figure 1-1: Middleware

Applications that employ a communications middleware like CoreDX DDS realize many benefits. The requirements and complexity of data communications in a distributed system are met by the middleware component - leaving developers more time to focus on the important application logic. CoreDX DDS middleware supports many operating systems and hardware architectures - the task of porting complex communications software is already complete.

1.3 The case for Publish SubscribeDDS

Many communication middleware technologies are available. Most are based on a functional model. For example, RPC (Remote Procedure Call) and CORBA (Object Request Broker) are two examples of middleware that allow function calls to be distributed across the network between a client and a server. However, these architectures lead to tight coupling between the client and the server; this makes these systems difficult to extend.

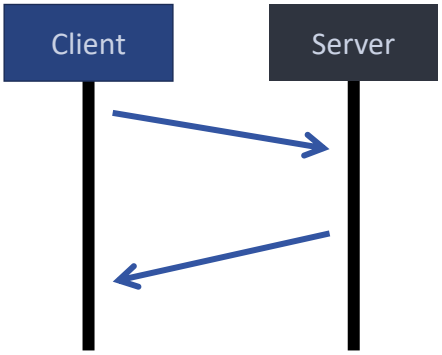


Figure 1-2: Client Server Architecture

The client-server architecture is appropriate for centralized data processing and works well in some systems and some use cases. In some client-server technologies, the drawbacks are increased integration costs for new capabilities and potential single point of failure.

An alternative to this approach is the Publish-Subscribe architecture embodied in DDS. This architecture promotes a loose coupling between data producers and data consumers. The architecture is flexible and dynamic; it is easy to adapt and extend systems to changing environments and requirements.Figure 1-3 illustrates the DDS Publish Subscribe architecture where multiple Publishers and Subscribers exchange strongly typed data through a common Topic. The communications are controlled by a Quality of Service model.

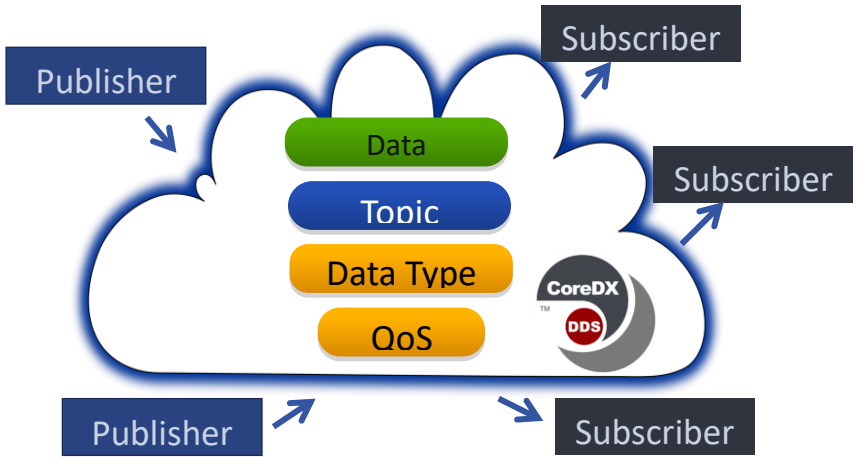


Figure 1-3: Publish Subscribe Architecture

Figure 1-4is an example of how DDS might be applied in a system. This example has several sources of "raw data", a data processor that performs some processing on the raw data to produce "processed data", several end users working with the processed data, and an administrative user performing analysis, maintenance, or auditing functions.

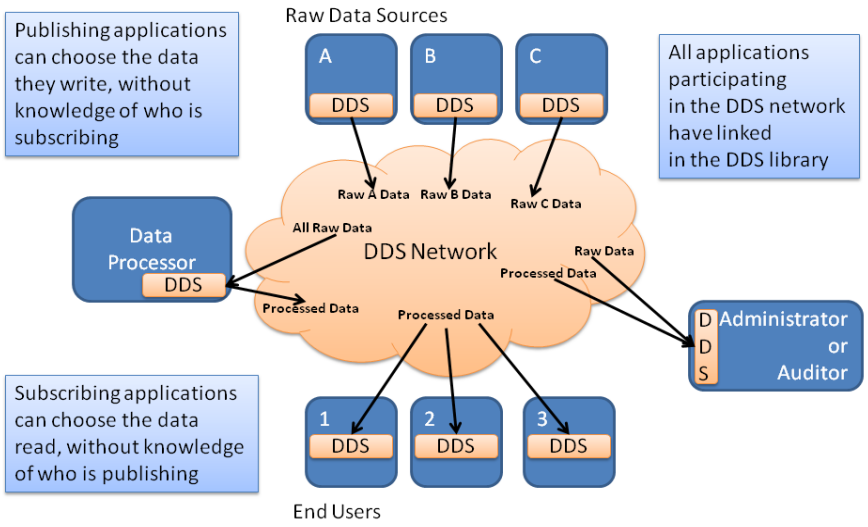


Figure 1-4: Example DDS Usage

In this example, the darker blue boxes represent applications communicating over a DDS network. These applications might be running together on 1 host, or they might be distributed over multiple hosts. A DDS application simply publishes or subscribes to their data, without concern for what, if anything, might be on the other end of its communications. Any of the applications can be dynamically removed (and new applications may be added) without impacting the existing network.

Because many systems include some natural publish-subscribe use cases as well as some natural client-server use cases, the DDS standards include both communication mechanisms. This document focuses on the publish-subscribe interface to DDS. For information on programming with the remote procedure call (RPC) or request reply APIs, please use the CoreDX DDS RPC over DDS Programmer's Guide.

1.3.1 DDS is an Open Standard

DDS is an open specification (documented by multiple standards) managed by the Object Management Group (OMG). The OMG is an international, open membership, non-profit organization that develops and manages computer industry specifications. Hundreds of organizations, including software end-users and commercial vendors, make up the OMG. Together they develop and manage many of the standards widely used in the computer industry today. The set of Data Distribution Service (DDS)

standards is an example of one of the technology standards managed by the OMG. Other examples include the Unified Modeling Language (UML), Model Driven Architecture (MDA) and the Common Object Request Broker Architecture (CORBA).

There are several advantages to using a technology that conforms to an open standard, and more advantages if that open standard is managed by an open membership organization like the OMG. First, an open standard promotes interoperability. Anyone, even if they are not connected with the managing organization, can pick up an Open Standard and write a conforming application. Second, open standards reduce the dependence on a particular vendor. When an open standard product is available from multiple vendors, the consumer can easily change between them. Finally, anyone can join the managing organization and vote on the direction and advancement of the technology. In the case of DDS, this means vendors and users, both public and private, can influence the future of the technology.

1.3.2 DDS is More than a Communications Middleware

The DDS standards specify the mechanism for moving data – a typical communications middleware technology standard. However, DDS is so much more. In addition to communications, DDS provides advanced data management, storage, organization, filtering, redundancy, extensibility, and security. With a rich set of features, interoperability across languages, operating systems, hardware platforms, and implementations, DDS provides a robust, secure infrastructure foundation for your small-scale, large-scale, enterprise, embedded, and everything in between software system.

1.3.3 Remote Procedure Call (RPC) in addition to Publish-Subscribe

The Data Distribution Service is a publish-subscribe technology, which provides a flexible, loosely coupled architecture suitable for many real-time applications. However, many sophisticated projects are a natural mix of publish-subscribe and client-server (or RPC) requirements.

The DDS Standards include API's for publish-susbscribe, request-response, and RPC – all implemented on top of the original DDS publish-subscribe architecture. DDS request-response and RPC have some unique features over other client-server type middleware, including automatic discovery, security, and the ability to use the full set of DDS Quality of Service (QoS) configurations.

The CoreDX DDS RPC API is fully described in the CoreDX DDS RPC Programmer's Guide.

1.3.4 DDS is flexible and scalable

Applications communicating with DDS might be running together on 1 host, or they might be distributed over multiple hosts, each with different architectures and operating systems. Applications using DDS for communications do not need to know the details of where the other applications are residing, or even if they exist.

The *discovery* mechanism built into DDS allows applications to come and go from a DDS network without requiring any changes to the applications or the network. This means a new system can be brought into the network, and start sending or receiving data, without any changes to existing applications.

1.3.5 DDS is secure

The DDS Security standard contains a complete state-of-the-art security solution that is completely integrated into the DDS protocols (**not** simply layered on top of SSL). DDS Security includes: Identification, Authentication, Access Control, Integrity, and Confidentiality, allowing the designer full flexibility on a topic-by-topic level.

Security configuration and usage is documented in the CoreDX DDS Security Programmer's Guide.

1.3.6 DDS Features

A DDS application can be a publisher of data, a subscriber of data, or both.

A ***Publisher*** is responsible for data distribution. It may publish data of different data types. The application uses a typed ***DataWriter*** attached to the publisher to communicate the data to be published. Both the Publisher and the DataWriter have a Quality of Service (***QoS***) that affects the behavior of the publication.

A ***Subscriber*** is responsible for receiving published data and making it available to the receiving application. It may receive data of different data types. The application uses a typed ***DataReader*** attached to the subscriber to access the data. Both the Subscriber and DataReader have a QoS that affects the behavior of the subscription. The subscribing application can

choose to block waiting for data using ***WaitSets*** or receive data asynchronously, using ***Listeners***.

A Topic fits between publications and subscriptions. Subscriptions must be able to refer to specific publications. A topic fulfills this purpose: it associates a name, a data-type, and a QoS related to the data itself.

When an application wants to publish data of a given type, it must use a Publisher and DataWriter with all the characteristics of the desired publication. When an application wants to subscribe to data of a given type, it must use a Subscriber and DataReader with all the characteristics of the desired subscription.

The following figure depicts the common DDS objects used in exchanging data.

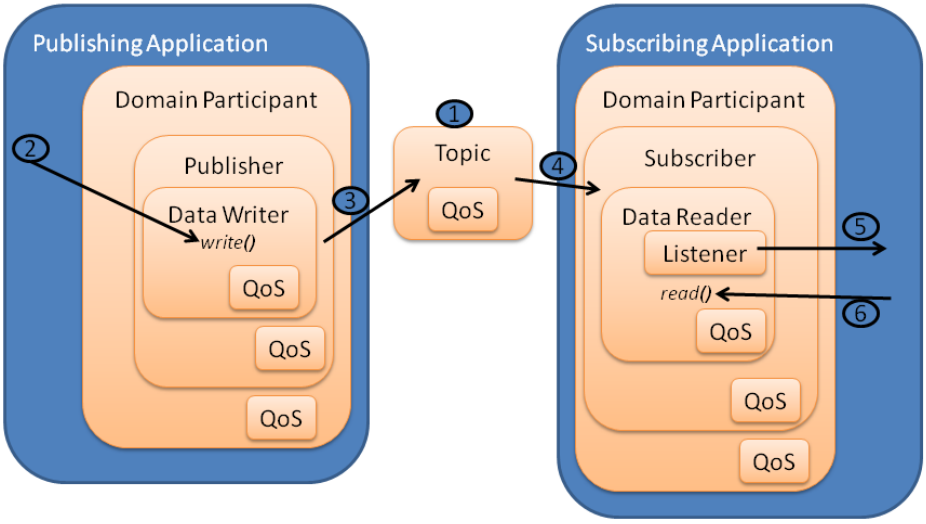


Figure 1-5: DDS Architecture

The following describes the actions depicted in Figure 1-5.

1. *DataReaders* and *DataWriters* are associated with a *Topic*
2. The publishing application calls *DataWriter*::write() to write the data
3. The *Publisher* publishes the data
4. The *Subscriber* receives the data
5. The *Listener* notifies the subscribing application of available data

6. The subscribing application calls *DataReader*::read() to access the data

1.4 The case for CoreDX DDS

The CoreDX DDS provides a quality, high-performance, very small footprint implementation of the DDS standards, including the original publish-subscribe DDS API, RTPS wire protocol, X-Types, RPC over DDS, and DDS Security.

1.4.1 CoreDX DDS is Fast

CoreDX DDS was built from the ground up with performance in mind. The engineering staff at Twin Oaks Computing has a long history of writing and maintaining real-time and near real-time software, and this expertise was used in creating CoreDX DDS. CoreDX DDS is written in 'C' (with additional application language bindings available)for low overhead and memory savings. The CoreDX DDS baseline is tested and enhanced for performance at every step of the development process. The result is a quality DDS implementation with extremely low latency and high throughput capacity.

CoreDX DDS data aggregation, multi-core data pipeline, and low latency event notification provide for throughput in the +900Mbps range and latencies below 75 usec over a 1Gbps ETHERNET network. But don't take our word for it. The CoreDX DDS release includes source code for example benchmarking applications. Use these examples to compile your own benchmark tests and see how CoreDX DDS performs in your environment, with your data.

1.4.2 CoreDX DDS is Small

The CoreDX DDS product is 100% designed and developed by Twin Oaks Computing to meet the OMG's DDS specification. There is no historical code, no code borrowed from the open source community, no code retrofitted to meet the CoreDX DDS requirements. This allows us to deliver a quality, fully-functional DDS implementation with the smallest footprint. Our entire core library is less than 500 KB, and runs in environments with as little as 100 KB of RAM. The full CoreDX DDS implementation is deployed on FPGA's, DSP's, PLC's, ECU's and other embedded environments.

This small library size comes with a proportionally small Line of Code Count, perfect for safety critical applications requiring DO-178B certification.

CoreDX DDS is modular and contains additional run-time memory tuning parameters. Space constrained projects can select components of CoreDX DDS to meet their requirements, and tune those components to reduce unnecessary memory utilization.

For those environments that are even smaller: true microcontrollers, CoreDX DDS Micro requires no more than 8K of RAM, allowing the benefit of the interoperability DDS protocols down to the component level of any system.

CoreDX DDS Micro is documented in the CoreDX DDS Micro Programmer's Guide.

1.4.3 CoreDX DDS is Proven & Robust

The small footprint CoreDX DDS software has over 10 years of deployment usage in a wide variety of mission-critical, and business-critial applications.

With over 1 million deployed instances around the world and in space, connecting components in surgical devices, military and commercial vehicles, space exploration platforms, electrical grids, CoreDX DDS has a proven track record of reliability, robustness, and compentent technical and business support.

1.4.4 CoreDX DDS is Secure

CoreDX DDS complies with the DDS Security standards, providing integrated and sophisticated security features that are fully configurable. Using state-of-the-art security algorithms, The DDS Security standard was designed to meet the requirements of military and critical national infrastructure systems.

System designers may choose to use the standards compliant Twin Oaks Computing developed security plug-ins for identification, authentication, access control, integrity, and confidentiality, or develop their own with the standardized plug-in API.

1.4.5 CoreDX DDS Uses Multi-Core Technologies

Hardware is moving to multiple core technology. Even embedded processors are shipping with more than one core. This presents a challenge to application developers, because making use of multiple cores requires complex code that is difficult and expensive to develop and maintain. The solution: use a multithreaded communications middleware like CoreDX DDS.

CoreDX DDS was architected from the start to take advantage of multi-core environments. With advanced threading and protections, each CoreDX DDS participant will use a minimum of 3 cores, and typical CoreDX DDS applications will use between 4and 8 cores. These are single threaded applications, taking advantage of quad-core and higher hardware, just by using CoreDX DDS for data communications.

1.4.6 CoreDX DDS is Self Contained

In order to use CoreDX DDS for communications, the application links in the appropriate CoreDX DDSlibraries and that is it. With no daemons and no operating system services that need to be started and maintained, there is no place for data to become "stuck" or for communication states to become corrupted.

1.4.7 CoreDX DDS has Comprehensive Platform Support

With the wide array of language binding, operating system and architecture support, CoreDX DDS runs on a wide variety of platforms, from enterprise servers, to common desktop configurations, to embedded environments and real-time operating systems, to FPGA's and 'bare-metal' configurations.

1.4.8 CoreDX DDS has a great team behind it

A quality DDS implementation is important. But the organization behind the implementation is critical. When you make a commitment to purchase a software product, you are not only obtaining the rights to run the software contained on the installation disk (or downloaded from the web). You are also obtaining support services, training services, and product enhancements for at least the next year.

The staff at Twin Oaks Computing has been developing and supporting large software systems and global software companies for over 50 years. We have worked beside soldiers in Kuwait, sailors onboard aircraft carriers, and other warfighters around the world. We have supported commercial IoT and IIoT companies with millions of products deployed world-wide. We understand not only the importance of delivering a software product that works, but also the importance of helping companies and their end users make the most of their investment.

We will do the same for you. Give us a call or send us an email. We promise you will receive prompt, friendly, and helpful service.

Part 2: CoreDX DDS Secure

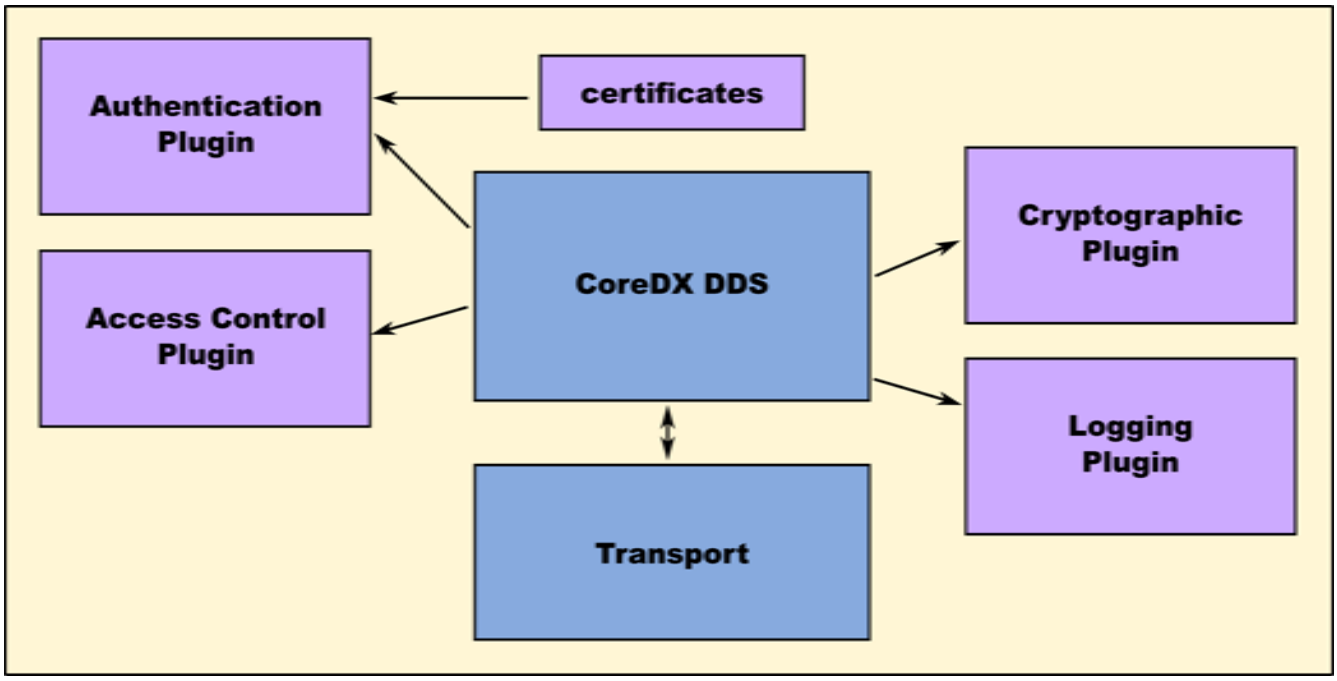
The section describes CoreDX DDS Secure – configuration and use.

Chapter 1 Introduction

CoreDX DDS versions 4.0 and later support the standardized DDS Security protocol and plug-in architecture. The DDS Security model provides for the following:

- Authentication of DDS entities,
- Authorization of DDS entities,
- Integrity of data and protocol messages,
- Confidentiality of data, and
- Non-repudiation of data

The security architecture is based on a plug-in approach, where the security functions are provided by a set of plug-in components. As such, a system may provide alternate implementations for each security function. For example, a user could provide his own implementation of the cryptographic algorithms.



CoreDX DDS includes an implementation of the standardized security plug-ins. These plug-ins can be used as-is, or can form the basis of a customized implementation.

This document presumes that the reader is familiar with DDS concepts. For the appropriate DDS background, please see the CoreDX DDS Programmer's Guide [57].

Chapter 2 CoreDX DDS Security Plug-in Overview

The **CoreDX DDS Security Plug-in** provides an implementation that complies with the DDS Security standards. The plug-in implementation includes identification, access control, integrity, cryptographic, and logging functions.

2.1 Integration

The CoreDX DDS Security Plug-in integrates with the CoreDX DDS middleware infrastructure via the standardized API defined in the DDS Security standard [57]. The **CoreDX DDS Secure** distribution includes the standardized plug-in ports to support integration of a secure plug-in implementation. It is possible to operate the secure distribution without installing a plug-in, in which case, the DDS middleware operates the standard DDS protocol without any specific security facilities.

2.2 Cryptographic Technology

The CoreDX DDS Security Plug-in utilizes PKI certificates for identification and authentication. It also employs the Diffie-Hellman key exchange algorithm (ECDH or DH+MODP) to establish symmetric keys used for data integrity and encryption. The plug-in uses the GCM and GMAC algorithms for fast and efficient symmetric key cryptography; these algorithms support AES-128 and AES-256 bit encryption.

2.3 Configuration

The CoreDX DDS Security Plug-in supports configuration through the use of two XML formatted documents: the **Domain Governance** document and the **Permissions** document. Together, these two documents control the behavior of the security plug-in, including how domains are controlled and which entities have permission to publish or subscribe on various topics. These documents are described in detail in subsequent sections of this document.

Chapter 3 CoreDX DDS Security Plug-in Configuration

3.1 Domain Governance Document

The Domain Governance document configures how DDS Domains will be controlled from a security perspective. This includes item such as:

- if all DDS data will be clear text, signed, or encrypted and signed,
- whether access is restricted to only authenticated Participants (that is, Participants in possession of a valid identity certificate signed by the Authentication CA)
- if and how DDS discovery data is protected
- if and how DDS liveliness data is protected

The DomainGovernance document is an XML document that complies with the standardized XSD in the OMG Consolidated XML standard. The document should have a top-level **<dds>** element that contains a single **<domain_access_rules>** element. The **<domain_access_rules>** element can include any number of **<domain_rule>** elements.

3.1.1 3.1.1 domain_rule element

The <**domain_rule>**element defines the security configuration for one or more DDS Domains. Each <**domain_rule**> element contains the following elements and sections:

- <domains> element
- <discovery_protection_kind> element
- <liveliness_protection_kind> element
- <allow_unauthenticated_join> element
- <enable_join_access_control> element
- <topic_access_rules> element, containing topic rules

The contents and delimiters of each Section are described below.

3.1.2 domains element

The <**domain_rule**>applies to the DDS domains specified in the **<domains>** element. The **<domains>** element contains one or more **<id>** and **<id_range>** elements. The <**id>** element text body should contain a single integer number that identifies a DDS Domain ID. The <**id_range>**element contains a **<min>** and **<max>** element, each of which contain a single integer number as their text body.

For example, a single domain:

```
<domains>
  <id>0</id>
</domains>
```

Or a single range of domains:

```
<domains>
  <id_range>
    <min>10</min>
    <max>20</max>
  </id_range>
</domains>
```

Or a combination:

```
<domains>
  <id>0</id>
  <id_range>
    <min>10</min>
    <max>20</max>
  </id_range>
</domains>
```

A domain_rule element applies to a DomainParticipant if the Participant's domain_id falls in the set of ids specified in the **<domains>**element. During operation, the **<domain_rule>** elements are searched in the order they are listed in the document. If multiple rule elements match, the first is selected. If no rule elements apply, then the domain is uncontrolled.

3.1.3 discovery_protection_kind element

The discovery protection element specifies the protection kind applied to the secure builtin DataWriter and DataReader entities used for discovery: SEDPbuiltinPublicationsSecureWriter, SEDPbuiltinSubscriptionsSecureWriter,

SEDPbuiltinPublicationsSecureReader, SEDPbuiltinSubscriptionsSecureReader. The discovery protection kind element may take three possible values: NONE, SIGN, or ENCRYPT. The specified protection is applied at the metadata (submesage) level.

For example:

```
<liveliness_protection_kind>SIGN</liveliness_protection_
kind>
```

3.1.4 liveliness_protection_kind element

The liveliness protection element specifies the protection kind applied to the builtin DataWriter and DataReader associated with the ParticipantMessageSecure builtin Topic used for communication of liveliness. The discovery protection kind element may take three possible values: NONE, SIGN, or ENCRYPT. The specified protection is applied at the metadata (submesage) level.

For example:

```
<liveliness_protection_kind>SIGN</liveliness_protection_
kind>
```

3.1.5 allow_unauthenticated_join element

This element controls if unauthenticated participants are allowed to join the DDS Domain. If set to TRUE, then the Participant will match with remote Participants even if they cannot be authenticated. If set to FALSE, then a remote participant that cannot be authenticated will essentially be ignored. This element may take the binary values TRUE or FALSE.

For example:

```
<allow_unauthenticated_participants>FALSE</allow_unauthe
nticated_participants>
```

3.1.6 enable_join_access_control element

This element determines whether or not remote Participant matching is controlled by the Permissions document. If set to TRUE, then matching a remote Participant is controlled by the Permissions document (that is, match only if the remote participant permissions must indicate that they are allowed to publish or subscribe to a topic in the domain). If set to FALSE,

then a remote participant is matched without consulting the Permissions. This element may take the binary values TRUE or FALSE. For example:

```
<enable_join_access_control>TRUE</enable_join_access_con
trol>
```

3.1.7 topic_access_rules element

The **topic_access_rules** element contains any number of **<topic_rule>** elements.

3.1.7.1 topic_rule element

Each **<topic_rule>** element contains

- topic_expression element
- enable_discovery_protection element
- enable_read_access_control element
- enable_write_access_control element
- metadata_protection_kind element
- data_protection_kind element

3.1.7.1.1 topic_expression element

The value in this element identifies the set of DDS Topic names to which the rule applies. The rule will apply to any DataReader or DataWriter associated with a Topic whose name matches the value. The Topic name expression syntax and matching complies with the syntax and rules of the POSIX fnmatch() function as specified in POSIX 1003.2-1992, Section B.6 [38].

For example:

```
<topic_expression>Square*</topic_expression>
```

3.1.7.1.2 enable_discovery_protection element

This element may take the binary values TRUE or FALSE. It controls whether the discovery information for the topic[s] covered by this rule is exchanged in the clear via the standard discovery builtin topics, or protected with the secured builtin topics.

For example:

```
        <enable_discovery_protection>TRUE</enable_discovery_prot
ection>
```

3.1.7.1.3 *enable_read_access_control element*

This element may take the binary values TRUE or FALSE. This controls whether or not access controls are applied to the reading permission on the topic[s] covered by this rule.

For example:

```
        <enable_read_access_control>TRUE</enable_read_access_con
trol>
```

3.1.7.1.4 *enable_write_access_control element*

This element may take the binary values TRUE or FALSE. This controls whether or not access controls are applied to the writing permission on the topic[s] covered by this rule.

For example:

```
        <enable_write_access_control>TRUE</enable_write_access_c
ontrol>
```

3.1.7.1.5 *metadata_protection_kind element*

This element may take the binary values: NONE, SIGN, or ENCRYPT.

The setting of this element controls the protection kind applied to the RTPS SubMessages (for example: Data, HeartBeat, AckNack), sent by any DataWriter and DataReader on the topic[s] covered by this rule.

For example:

```
        <metadata_protection_kind>ENCRYPT</metadata_protection_k
ind>
```

Because this protection applies to the entire sub-message, it will effectively apply to any application data contained therein.

3.1.7.1.6 data_protection_kind element

This element may take three possible values: NONE, SIGN, or ENCRYPT.

The setting of this element shall specify the protection kind applied to the payload data (application data) sent by any DataWriter on the topic[s] covered by this rule.

For example:

```
<data_protection_kind>ENCRYPT</data_protection_kind>
```

With this setting, only the application data is protected; the meta-data contained in the the data header or in handshaking sub-messages is not protected (see metadata_protection_kind).

3.2 Permissions Document

The Permissions document grants or denies publish and subscribe permission on topics to individual Participants. The participants are identified by the 'Subject Name' contained in their Identity certificate. The Permissions document is an XML document that complies with the standardized XSD in the OMG Consolidated XML standard**.**

ThePermissions document may be consulted to check for the permission to create or match a participant, topic, reader or writer, based on the settings in the DomainGovernance file.

The document should have a **<dds>** element containing a **<permissions>** element that contains any number of **<grant>** elements. Each **<grant>** element should include the following elements:

- <subject_name> element
- <validity> element
- Rules
  - <allow_rule> element[s]
  - <deny_rule> element[s]
  - <default> element

3.2.1 subject_name element

The subject name section identifies the DomainParticipant to which the grant section applies. Each grant section in the Permissions document should have a unique subject name. That is, a subject name should appear only once in the Permissions document.

The contents of the <subject_name> element must must match the X.509 subject name for the DomainParticipant as is given in its Identity Certificate.

The X.509 subject name is a set of name-value pairs. The format of X.509 subject name shall be the string representation of the X.509 certificate subject name as defined in IETF RFC 4514 "Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names" [51].

For example:

```
<subject_name>/C=US/ST=CO/O=Acme Inc/CN=CN_TEST_DDS-
SECURITY_ACME/emailAd dress=so@acme.com</subject_name>
```

3.2.2 validity element

The contents of the **validity** element indicate the valid dates for the grant element. It contains both the starting date and the end date in GMT formatted as YYYYMMDDHH.

A grant section with a validity date that does not include the current date at which the permissions are being evaluated shall be ignored.

3.2.3 Rules

The permissions assigned to the DomainParticipant are described as a set of rules (allow, deny, and default). The rules are applied in the same order that appear in the document. Each rule includes the domain_id, topic name, the partitions, and the data-tags associated with the DataWriter or DataReader; these are referred to as the 'matching criteria' or simply 'criteria'.

If the criteria for the rule matches the domain join or publish/subscribe operation that is being attempted, then the allow or deny decision is applied. For example, rules are checked when the application creates a DataReader or DataWriter and also when a considering matching with a remote DataReader or DataWriter.

If the criteria for a rule does not match the operation being attempted, the evaluation shall proceed to the next rule. If all rules have been examined without a match, then the decision specified by the "default" rule is applied. If the default rule is not present, the default decision is DENY.

For a rule to match, the domain_id, topic, partitions, and data-tags criteria must match. For the 'topic' criterion to match it is sufficient that one of the topic expressions listed matches (i.e., an OR of the expressions with the <topics> section). For the 'partition' criterion to match, it is sufficient for one of the partition names associated with the entity (Reader/Writer) to match one of the partition expressions within one of the <partitions> elements. If the <partitions> element is not present, it is treated as if a single 'empty string' partition is specified. For the data-tag criterion to match, it is required that all of the data-tags associated with the entity (Reader/Writer) be present in one of the <**data_tags**> elements.

Each rule (whether it be an allow or deny rule) includes:

- a <domains> element
- any number of <publish> elements
- any number of <subscribe> elements

The <domains> element indicates which domain_id[s] are impacted by the rule; it includes any number of <id> and <id_range> elements specifying a set of domain_ids to which the rule applies. The <publish> and <subscribe> elements each identify one or more topic names (or topic name patterns) and optionally the associated partitions and data tags.

3.2.4 allow_rule element

The <**allow_rule**> element is a member of the <**grant**> element. Allow rules appear inside the <**allow_rule**> XML Element. Each allow rule contains the domain IDs to which the rule applies, and any number of <**publish**> and <**subscribe**> elements. The <**publish**> and <**subscribe**> elements contain the following:

- a set of the topic names (topic name expressions) that are allowed to be published and subscribed;
- a list of lists of partition names, allowing access if at least one of the names listed matches one of the names in the Partition QoS of the entity in question; and,

- a list of lists of data tags that must be present in the DataTags QoS of the entity in question.

3.2.5 deny_rule element

The <**deny_rule**> element is a member of the <**grant**> element. Deny rules appear inside the <**deny_rule**> XML Element. Each deny rule contains the domain IDs to which the rule applies, and any number of <**publish**> and <**subscribe**> elements. The <**publish**> and <**subscribe**> elements contain the following:

- a set of the topic names (topic name expressions) that are denied from being published and subscribed;
- a list of lists of partition names, denying access if at least one of the names listed matches one of the names in the Partition QoS of the entity in question; and,
- a list of lists of data tags that must not be present in the DataTags QoS of the entity in question.

3.2.6 default element

The <**default**> element is a member of the <**grant**> element. It should contain the text "ALLOW" or "DENY". If no allow_rule or deny_rule is found that matches the entity in question, then the specified default action is applied. If the default element is absent, then the default action is set to DENY.

3.2.7 domains element

The **<domains>** element contains one or more **<id>** or **<id_range>** elements. Its format is the same as the **domains** element in the DomainGovernance.

3.2.8 publish element

The <**publish>** element can include any number of **<topics>**, **<partitions>**, and **<data_tags>** elements. An entity is tested against each of these criteria to determine a match for the containing rule (allow or deny).

3.2.9 subscribe element

The <**subscribe>** element can include any number of **<topics>**, **<partitions>**, and **<data_tags>** elements. An entity is tested against each of these criteria to determine a match for the containing rule (allow or deny).

3.2.10 topics element

The <**topics>** element can include one or more **<topic>** elements. Each <**topic**> element contains a topic name expression text string. The topic name expression syntax follows the syntax and rules of the POSIX fnmatch() function as specified in POSIX 1003.2-1992, Section B.6 [38].

For example:

```
<topics>
<topic>Secure_*</topic>
<topic>Square</topic>
</topics>
```

3.2.11 partitions element

The <**partitions>**element contains one or more **<partition>** elements. Each <**partition>**element contains a DDS Partition name. Partition names may be given explicitly or by means of Partition name expressions. Each partition name or partition-name expression appears separately in a **<partition>** element within the **<partitions>** element.

The partition name expression syntax and matching uses the syntax and rules of the POSIX fnmatch() function as specified in POSIX 1003.2-1992, Section B.6 [38]. If there is no **<partitions>** Section then the rule allows publishing only in the "empty string" partition. See PARTITION QosPolicy entry in Qos Policies table of section 2.2.3 (Supported Qos) of the DDS Specification version 1.4.

For example:

```
<partitions>
<partition>A1</partition>
<partition>B*</partition>
</partitions>
```

3.2.12 data_tags element

The <**data_tags>** element contains one or more **<data_tag>**elements. Each <**data_tag>** element must include a **<name>** and **<value>** element. Each of the <**name>** and **<value>** elements contains a string, together identifying a (name,value) pair.

For example:

```
      <data_tags>
      <data_tag><name>varA</name><value>123</value></dat
a_tag>
      <data_tag><name>varB</name><value>123 East
Main</value></data_tag>
      </data_tags>
```

3.2.13 Example <grant> elements

The following <grant> section would allow publishing of only a topic named "Square" on domain 0 under partition "A".

```
<grant name="TWINOAKS_grant_all_domain_0">
<subject_name>/C=US/ST=CO/O=Twin Oaks
Computing/CN=CN_TEST_DDS/emailAddress=sup
      port@twinoakscomputing.com</subject_name>
<validity>
<not_before>2016-01-01T00:00:00</not_before>
<not_after>2018-01-01T00:00:00</not_after>
</validity>
<allow_rule>
<domains>
<id>0</id>
</domains>
<publish>
<topics>
<topic>Square</topic>
</topics>
      <partitions>
      <partition>A</partition
      </partitions>
</publish>
</allow_rule>
<default>DENY</default>
</grant>
```

3.3 Certificate Authorities and Identity Certificates

The CoreDX DDS Security Plugin utilizes standard PKI certificates to support identification and authentication.

The plugins can make use of two distinct Certificate Authorities (CAs); one for Authentication and one for Permissions. The system can also be configured such that a single CA is used for both purposes. The creation and management of certificates is outside the scope of the plugin functionality, but is described below for the purpose of example.

The **Authentication CA** is used to validate identity credentials. Each 'identity certificate' used to identify a DomainParticipant must be signed by the Authentication CA, and will be validated against the Authentication CA at runtime.

The **Permissions CA** is used to validate the Domain Governance and Permissions XML documents. These configuration documents must be signed by the Permissions CA, and will be validated against the Permissions CA at runtime.

The **Identity Certificate** is used to identify a DomainParticipant. This certificate must include a Subject Name (CN) field. This value (subject name) is used as an index into the Permissions Document to select the appropriate permissions that apply to the associated Participant. Further, the plugin must have access to the 'private key' that is associated with this certificate, so that it can sign authentication handshake tokens that will be validated against the public Identity Certificate as part of the the Authentication process. The private key is often protected with a passphrase to control access. If the private key is so protected, the passphrase can be configured so that the plugin can decode the private key.

3.4 Impact of DomainGovernance and Permissions

The DDS Security Plugins, once configured, operate internally to the middleware. The plugins augment the behavior of the standard DDS API. The application software does not interact directly with the plug-in implementation. Some DDS API routines can fail or return different error codes based on the configuration and operation of the security plugins. For example, the security configuration disallow the creation (or enabling) of a Reader on a particular topic; as a result, Subscriber::create_datareader() or DataReader::enable() might fail where they normally would succeed.

The following sections indicate how the standard security plug-in implementation impacts the behavior of the DDS API.

3.4.1 Enable Participant

If the applicable DomainGovernance document **enable_join_access_control** element is set to TRUE, then the creation of a DomainParticipant is controlled by the DomainGovernance document and the Permissions document; otherwise, the creation of Participants is not controlled.

If participant creation is controlled, then the following conditions are applied:

If the DomainGovernance document specifies any topics on the DomainParticipant domain_id with **enable_read_access_control** set to FALSE or with **enable_write_access_control** set to FALSE, (that is, if the participant has access to an uncontrolled topic) then the operation shall succeed and return TRUE.

If the Permissions document contains a grant element for the DomainParticipant and the grant contains an allow rule on the DomainParticipant's domain_id, then the enable operation shall proceede.

3.4.2 Create Topic

The DomainGovernance document is searched for a matching topic name or topic-expression for the associated domain_id. If found, then the **enable_write_access_control** and **enable_read_access_control** flags are checked – if either is set to FALSE then the Participant is allowed to create the topic. If both read and write access control is enabled (TRUE), then the Permissions document is consulted to located a matching 'grant' element that allows publish or subscription on the topic. If such element is found, the topic creation is allowed; otherwise, the 'default' element determines if the creation is allowed or denied.

3.4.3 Create DataWriter

The create_datawriter() operation is not directly impacted by the Security Plug-in. However, if the Publisher QoS "autoenable_created_entities" is TRUE, then the create_datawriter() operation will call 'enable()' on the DataWriter. If the enable fails (see 37Enable DataWriter) then create_datawriter() will return NULL.

3.4.4 Enable DataWriter

During the DataWriter enable() operation, the Security Plug-in is consulted to determine if the DataWriter is permitted. First, the DomainGovernance document is searched for a matching topic name or topic-expression for the associated domain_id. If found, then the **enable_write_access_control** flag is checked – if it is set to FALSE then the DataWriter is allowed, and the enable() operation is allowed to proceed. If write access control is enabled (TRUE), then the Permissions document is consulted to located a matching 'grant' element that allows publishing on the topic. If such element is

found, the DataWriter enable is allowed; otherwise, the 'default' element determines if the enable operation is allowed or denied. A matching grant rule considers the topic_name, as well as the partition[s], and data_tag[s] in the DataWriter QoS.

If the enable operation is denied by the Security Plug-in, then an error of NOT_ALLOWED_BY_SEC is returned.

3.4.5 Create DataReader

The create_datareader() operation is not directly impacted by the Security Plug-in. However, if the Subscriber QoS "autoenable_created_entities" is TRUE, then the create_datareader() operation will call 'enable()' on the DataReader. If the enable fails (see 38Enable DataReader) then create_datareader() will return NULL.

3.4.6 Enable DataReader

During the DataReader enable() operation, the Security Plug-in is consulted to determine if the DataReader is permitted. First, the DomainGovernance document is searched for a matching topic name or topic-expression for the associated domain_id. If found, and the associated **enable_read_access_control** flag is set to FALSE, then the DataReader is allowed, and the enable operation can proceed. If not found, or if the read access control is enabled (TRUE), then the Permissions document is consulted to located a matching 'grant' element that allows subscribing on the topic. If such element is found, the DataReader enable is allowed; otherwise, the 'default' element determines if the enable is allowed or denied. A matching grant rule considers the topic_name, as well as the partition[s], and data_tag[s] in the DataReader QoS.

If the enable operation is denied by the Security Plug-in, then an error of NOT_ALLOWED_BY_SEC is returned.

3.4.7 Matching Remote Participant

If the applicable DomainGovernance document **enable_join_access_control** element is set to TRUE, then the matching of a DomainParticipant is limited by the DomainGovernance document and the Permissions document.

3.4.8 Matching Remote DataReader

The DomainGovernance document is searched for a matching topic or topic-expression for the participant's domain_id. If found, and **enable_read_access_control** set to FALSE, then the match is allowed.

Otherwise, the Permissions document is searched for a applicable grant element (allow or deny rule), considering the subject_name, domain_id, topic_name, the partition[s], and the data_tag[s]. If an 'allow subscription' rule is found, then the match is allowed. If a 'deny subscription' rule is found, the match is denied.

If no matching rule is found, then the 'default' rule is applied, either allowing or denying the match.

3.4.9 Matching Remote DataWriter

The DomainGovernance document is searched for a matching topic or topic-expression for the participant's domain_id. If found, and **enable_write_access_control** set to FALSE, then the match is allowed.

Otherwise, the Permissions document is searched for a applicable grant element (allow or deny rule), considering the subject_name, domain_id, topic_name, the partition[s], and the data_tag[s]. If an 'allow publication' rule is found, then the match is allowed. If a 'deny publication' rule is found, the match is denied.

If no matching rule is found, then the 'default' rule is applied, either allowing or denying the match.

3.5 CoreDX DDS Security Plug-In Run-Time Configuration

The configuration of the security plugins is provided via the DomainParticpantQos; specifically, the 'properties' QosPolicy. The 'properties' policy is a sequence of name-value pairs. There are several standardized property names that are used to configure the Security plugins.

The following properties are required to configure properties of the CoreDX DDS Security plug-ins:

Property Name	Value Description
"dds.sec.auth.identity_ca"	The URL that identifies the Certificate of the Authentication CA. This certificate is used to validate authentication information.
"dds.sec.auth.identity_certificate"	The URL that identifies the Certificate of the DomainParticipant. This is the 'identity' of the participant, and must be signed via the provided Authentication CA.
"dds.sec.auth.private_key"	This is the URL that identifies the PRIVATE key associated with the participant 'identity' certificate.
"dds.sec.auth.password"	This is a 'passphrase' used to access the PRIVATE key (if required).
"dds.sec.access.permissions_ca"	A URL that identifies the certificate of the Permissions CA. This certificate is used to validate permissions information.
"dds.sec.access.governance"	A URL that contains the Domain Governance XML document. This document must be 'signed' by the Permissions CA.
"dds.sec.access.permissions"	A URL that contains the Permissions XML document. This document must be signed by the Permissions CA.
"com.toc.sec.create_plugins"	This identifies the name of the dynamic library containing the implementation of the security plugins. Further, it includes a function name to call to initialize the library.
"com.toc.sec.kx_aesbits"	Sets the key size for the KeyExchange builtin topic. (default 256)
"com.toc.sec.dr_aesbits"	Sets the key size for application created DataReader[s]. (default 128)
"com.toc.sec.dw_aesbits"	Sets the key size for application created DataWriter[s]. (default 128)

"com.toc.sec.log_level"	Set the level of security logging: EMERGENCY, ALERT, CRITICAL, etc. Impacts the verbosity of the security logging.
"com.toc.sec.log_file"	Set the name of the local file where security logging is recoreded.
"com.toc.sec.log_publish"	Boolean flag (0 or 1) indicating if security logging entries should be published on the built-in DDS Logging topic.

CoreDX DDS supports the 'file:' and 'data:' URL kinds for the various property values that identify a URL. Each property is described in detail below.

3.5.1 dds.sec.auth.identity_ca

This property must be configured with a URL that contains the certificate of the Authentication CA. This certificate is used to validate authentication information. The certificate should be in PEM format.

For example:

```
"file:CA_Identity_cert.pem", or

"data:,-----BEGIN CERTIFICATE-----
MIIDjDCCAnQCCQCxXfSqVVsCvjANBgkqhkiG9w0BAQUFADCBhzELMAkG
A1UEBhMC
...
NmwIbJfu+GOhaP/MW6Pio1eTNI7HusuMFj+tsphc79bDkU1UxW8wmqBs
OVJFFlIq
-----END CERTIFICATE-----"
```

3.5.2 dds.sec.auth.identity_certificate

This property must be configured with a URL that contains the Identity Certificate that will be used to identify this DomainParticipant to the network. The certificate should be in PEM format.

For example:

```
"file:DP1_cert.pem", or

"data:,-----BEGIN CERTIFICATE-----
VQQIDAJNQTEPMA0GA1UEBwwGQm9zdG9uMSIwIAYDVQQKDBlPTUctRERT
IFNJRyAo
...
```

```
NBgApgohasrvKSK3ZrORuehSsvA3fqkatIJIGry+0/N1Z7sf3RhCl9sB
urW2o5EC
r64v
-----END CERTIFICATE-----"
```

3.5.3 dds.sec.auth.private_key

This property must be configured with the private key that corresponds to the Identity Certificate. This key is used during security handshake to establish identification with peers. The certificate should be in PEM format.

For example:

```
"file:DP1_private_key.pem", or

"data:,-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,AC4536BB9894B622

HlDuyaM2I0YzI7CvXU1XtgRrAcsBzOmIt4fIZaoWqJJDXFsjb2UVnifj
akeVkEPR
...
u36htaR2wdf9wZk9FD3qjoXkrqS1Y7QUE830fcQ4dYAeRzHaMCq7dL1C
A47zeSWN
-----END RSA PRIVATE KEY-----"
```

NOTE: The private key associated with the participant identity certificate is sensitive, and must be protected from disclosure. Invalid access to a public identity cert and the private key would allow a participant to masquerade as that identity. System level procedures and protections are required to mitigate this risk.

3.5.4 dds.sec.auth.password

If the private key is protected with a pass-phrase, that pass-phrase must be configured in this property.

For example:

```
"data:,the_pass_phrase"
```

NOTE: The private key (and pass-phrase) associated with the participant identity certificate is sensitive, and must be protected from disclosure. Invalid access to a public identity cert and the private key would allow a participant to masquerade as that identity. System level procedures and protections are required to mitigate this risk.

3.5.5 dds.sec.access.permissions_ca

This property must be configured with a URL that contains the certificate of the Permissions CA. This certificate is used to validate permissions information. The certificate should be in PEM format.

For example:

```
"file:CA_Permissions_cert.pem", or

"data:,-----BEGIN CERTIFICATE-----
MIIEszCCA5ugAwIBAgIJAP5qAf9jZ9GBMA0GCSqGSIb3DQEBBQUAMIGX
MQswCQYD
...
6Zg41sG0AdbxjEb8A+6LSUOIgV4rS9wTU2NrG91MoHpgL4atz1RI81gk
NtzTZxO4
K8KV2pG15w==
-----END CERTIFICATE-----"
```

3.5.6 dds.sec.access.governance

This property must be configured with a URL that contains the signed DomainGovernance document. This data should be in S/MIME format.

For example:

```
"file:test_domain_gov.pkcs7"
```

3.5.7 dds.sec.access.permissions

This property must be configured with a URL that contains the signed Permissions document. This data should be in S/MIME format.

For example:

```
"file:test_permissions.pkcs7"
```

3.5.8 com.toc.sec.create_plugins

This property must be configured with the library name and function name used to initialize the security plugins. At run-time, the CoreDX DDS code will attempt to load the named dynamic library and locate the named function. The format of the property is "library_name:function_name".

For example:

```
"dds_security:DSREF_create_plugins"
```

This will cause CoreDX DDS to search for a library named "dds_security", "dds_security.so", "libdds_security.so", "libdds_security.dylib", and "dds_security.dll", in that order. The first matching library found will be opened, and will be searched for the function name.

Alternatively, the plugin can be linked directly into the application. In this case, the library_name should be left blank, and the current executable will be search for the named function.

If the function name is found, it is invoked and the return value is expected to be a pointer to a set of security plugins, in the following structure:

```
typedef struct DDS_SecurityPlugins
{
  DDS_Security_Authentication     * auth;
  DDS_Security_AccessControl      * access_control;
  DDS_Security_CryptoKeyFactory   * crypto_key_factory;
  DDS_Security_CryptoKeyExchange  * crypto_key_exchange;
  DDS_Security_CryptoTransform    * crypto_transform;
  DDS_Security_Logging            * logging;
  DDS_ReturnCode_t               (* destroy_plugins ) ( struct
     DDS_SecurityPlugins * plugins );
} DDS_SecurityPlugins;
```

The CoreDX DDS Security Plug-in implementations use the function name **DSREF_create_plugins** for the entry point.

3.5.9 com.toc.sec.kx_aesbits

This property can be configured to override the default AES encryption bits (256) used by the builtin Key Exchange topic. It can be set to "128" or "256". For interoperability, the default of 256 bits is recommended. Further, the Key Exchange topic should be protected with encryption that is at least as strong as the keys being exchanged in the topic data.

3.5.10 com.toc.sec.dr_aesbits

This property can be configured to override the default AES encryption bits (128) used by any application created DataReader. It can be set to "128" or "256".

3.5.11 com.toc.sec.dw_aesbits

This property can be configured to override the default AES encryption bits (128) used by any application created DataWriter. It can be set to "128" or "256".

3.5.12 com.toc.sec.log_level

Set the level of security logging. This impacts the verbosity of the security logging. The valid values are the following:

Level	Value
EMERGENCY	0
ALERT	1
CRITICAL	2
ERROR	3
NOTICE	4
INFORMATIONAL	5
DEBUG	6

3.5.13 com.toc.sec.log_file

This property is used to specify the name of a file to send security logging information. By default, the file is "/tmp/DDS_SecurityLog_GUIDBYTES". Where GUIDBYTES is made up of the hex values of the twelve bytes of the Domain Participant GUID prefix.

3.5.14 com.toc.sec.log_publish

This property controls whether or not security logging messages are published on the built-in DDS Security Logging topic. If the property is set to a non-zero value, then the messages are published; otherwise, they are not.

3.6 CoreDX DDS Security Plug-In Run-Time Environment

If configured to load the Security Plug-in from a dynamic library, then the library must be included in the dynamic library search paths. For example LD_LIBRARY_PATH in Unix environments and PATH in Windows environments.

Further, the CoreDX DDS Security Plug-in implementation requires the OpenSSL crypto (eg, libcrypto.so or crypto.dll) library be available. It can be

linked dynamically or statically. If linked dynamically, it must be included in the dynamic library search path.

Chapter 4 Security Logging

The Security Plug-in API includes a facility to access security logging information either through a callback mechanism, a logging file, or by a built-in DDS "SecurityLogging" topic. Configuration of the security logging is configured when the plug-in is created via properties, as described in Section 39CoreDX DDS Security Plug-In Run-Time Configuration.

Chapter 5 5 Creating Certs and Signing Docs with OpenSSL

The OpenSSL project includes a comprehensive cryptographic library and a command line application that can be used to perform many cryptographic tasks including generation of PKI certifcates and signing documents. While most deployments will use a key management service to ensure the proper handling of key material, we show the process of using the OpenSSL utility to generate the necessary tasks for the CoreDX DDS security plugin.

5.1 Certificates

The CoreDX DDS Security Plugin utilizes standard PKI certificates to support identification and authentication. These certificates can be created through the use of the '**openssl**' command line utility.

5.1.1 Identity Certificate Authority

The openssl command line utility can generate PKI certificates and keys. The certificate generation process is controlled by a configuration file. The configuration file controls aspects of the certificate generation process. For further information about this configuration file, see the OpenSSL documentation. An example is provided below:

```
#
# OpenSSL example Certificate Authority configuration file.
#
####################################################################
[ ca ]
default_ca  = CA_default            # The default ca section

####################################################################
[ CA_default ]

dir               = ./identity_ca_files   # Where everything is kept
certs             = $dir/certs            # Where the issued certs
are kept
crl_dir           = $dir/crl        # Where the issued crl are kept
database          = $dir/index.txt  # database index file.
#unique_subject   = no              # Set to 'no' to allow creation of
                                    # several ctificates with same subject.
new_certs_dir     = $dir
```

```
certificate = ./TESTONLY_identity_ca_cert.pem   # The CA certificate
serial      = $dir/serial           # The current serial number
crlnumber   = $dir/crlnumber        # the current crl number
                                    # must be commented for a V1 CRL
crl         = $dir/crl.pem          # The current CRL
private_key = $dir/private/TESTONLY_identity_ca_private_key.pem
RANDFILE    = $dir/private/.rand    # private random number file

#x509_extensions  = usr_cert        # The extentions to add to the
cert

# Comment out the following two lines for the "traditional"
# (and highly broken) format.
name_opt          = ca_default            # Subject Name options
cert_opt          = ca_default            # Certificate options

# Extension copying option: use with caution.
# copy_extensions       = copy

# Extensions to add to a CRL. Note: Netscape communicator chokes on
# V2 CRLs so this is commented out by default to leave a V1 CRL.
# crlnumber must also be commented out to leave a V1 CRL.
# crl_extensions              = crl_ext

default_days                  = 365       # how long to certify for
default_crl_days              = 30        # how long before next CRL
default_md                    = sha256    # which md to use.
preserve                      = no        # keep passed DN ordering

# A few differece way of specifying how similar the request should
# look For type CA, the listed attributes must be the same, and the
# optional and supplied fields are just that :-)
policy                  = policy_match    # For the CA policy
[ policy_match ]
countryName             = optional
stateOrProvinceName     = optional
organizationName        = optional
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types.
[ policy_anything ]
countryName                 = optional
stateOrProvinceName         = optional
localityName                = optional
organizationName            = optional
organizationalUnitName      = optional
```

```
commonName                      = supplied
emailAddress                    = optional

[ req ]
prompt                          = no
#default_bits                   = 1024
#default_keyfile                = privkey.pem
distinguished_name              = req_distinguished_name
#attributes                     = req_attributes
#x509_extensions                = v3_ca

[ req_distinguished_name ]
#countryName                    = Country Name (2 letter code)
#countryName_default            = US
countryName                     = US
#countryName_min                = 2
#countryName_max                = 2

#stateOrProvinceName            = State or Province Name (full name)
#stateOrProvinceName_default    = CA
stateOrProvinceName             = MA

localityName                    = Boston

#0.organizationName             = Organization Name (eg, company)
0.organizationName              = OMG-DDS SIG (Identity CA)

#commonName                     = Common Name (eg, YOUR name)
#commonName_max                 = 64
commonName                      = OMG-DDS (Iden CA)

#emailAddress                   = Email Address
#emailAddress_max               = 64
emailAddress                    = dds@omg.org
```

After creating the configuration file, the following steps are required to generate a Identity Certificate Authority and then create an Identity Certificate for a participant.

Step 1: Initialization of Index and Serial number files

```
touch identity_ca_files/index.txt
echo "01" > identity_ca_files/serial
```

Step 2: Generate the Identity CA private key:

```
openssl genrsa -out \
      identity_ca_files/private/EXAMPLE_identity_ca_private_key.pem  2048
```

Step 3: Generate the self-signed certificate signing request **(CSR) for the Identity CA**:

```
openssl req -config identity_ca_files/identity_ca_openssl.cnf -new \
      -key identity_ca_files/private/EXAMPLE_identity_ca_private_key.pem \
      -out identity_ca_files/identity_ca.csr
```

Step 4: Generate the Identity CA Certificate:

```
openssl x509 -req -days 3650 -in identity_ca_files/identity_ca.csr  \
      -signkey identity_ca_files/private/EXAMPLE_identity_ca_private_key.pem \
      -out identity_ca_files/EXAMPLE_identity_ca_cert.pem
```

5.1.2 Permissions Certificate Authority

The Permissions CA should have an OpenSSL configuration file similar to the Identity CA configuration file shown above.

Step 1: Initialization of Index and Serial number files

```
touch permissions_ca_files/index.txt
echo "01" > permissions_ca_files/serial
```

Step 2: Generate the Permissions CA private key:

```
openssl genrsa -out \
permissions_ca_files/private/TESTONLY_permissions_ca_private_key.pem  2048
```

Step 3: Generate the self-signed certificate signing request **(CSR) for the Permissions CA**:

```
openssl req -config permissions_ca_files/permissions_ca_openssl.cnf -new \
      -key permissions_ca_files/private/EXAMPLE_permissions_ca_private_key.pem \
      -out permissions_ca_files/permissions_ca.csr
```

Step 4: Generate the Identity CA Certificate:

```
openssl x509 -req -days 3650 -in  permissions_ca_files/permissions_ca.csr \
      -signkey permissions_ca_files/private/EXAMPLE_permissions_ca_private_key.pem \
      -out permissions_ca_files/EXAMPLE_permissions_ca_cert.pem
```

5.1.3 Identity Certificate

Each participant that participates in secure communications must have an Identity Certificate. The certificate must include a "Subject Name". The subject name is used as an index into the Permissions document to find the rules that apply to the participant.

Step 1: Generate a CSR for the Participant Identity Certificate:

```
openssl req -config identity_ca_files/identity_ca_openssl.cnf -new \
      -key identity_ca_files/private/EXAMPLE_identity_ca_private_key.pem \
      -out example_identity_dds_participant.csr
```

Step 2: Generate the **Participant Identity Certificate** (by signing the CSR with Identity CA cert)

```
openssl ca -config identity_ca_files/identity_ca_openssl.cnf -days 3650 \
      -in example_identity_dds_participant.csr \
      -out EXAMPLE_participant_identity_cert.pem
```

5.2 Signing Documents

The DomainGovernance and Permissions documents that control the plug-in behavior must be signed by a trusted Permissions Certificate Authority. The resulting signed documents can be verified using PKI techniques. The CoreDX DDS Security Plug-ins accept the documents in SMIME format.

5.2.1 Signing the DomainGovernance Document

Once the Permissions CA has been established, its certificate and private key can be used to sign the DomainGovernance file.

```
openssl smime -sign -in EXAMPLE_governance.xml \
      -text \
      -out EXAMPLE_governance_signed.p7s \
      -signer permissions_ca_files/EXAMPLE_permissions_ca_cert.pem \
      -inkey  permissions_ca_files/private/EXAMPLE_permissions_ca_private_key.pem
```

5.2.2 Signing the Permissions Document[s]

The Permissions CA is also used to sign the Permissions file[s].

```
openssl smime -sign \
      -in toc_coredx_dds_certs/EXAMPLE_participant_permissions.xml \
```

```
-text \
-out EXAMPLE_participant_permissions_signed.p7s \
-signer permissions_ca_files/EXAMPLE_permissions_ca_cert.pem \
-inkey  permissions_ca_files/private/EXAMPLE_permissions_ca_private_key.pem
```

Chapter 6 Example Code

There is some additional code that must be included in an application to integrate the CoreDX DDS Security Plug-ins. The CoreDX DDS distribution must support the security plug-in ports; such distributions include the tag "-sec" in their name.

By configuring a set of properties in the DomainParticipantQos before creating the participant, the middleware is informed of the need to instantiate and integrate with the specified security plug-in. The configuration properties are passed to the plug-ins, and calls are made to the plug-ins at the appropriate times to check for authentication, access-control, and perform crypto operations.

The following C++ code illustrates how to set the Security Plug-in configuration properties. The pattern is similar for other programming languages.

```
/* These property names are standardized for the Security plug-in
 * configuration:
 */

#define DDSSEC_PROP_IDENTITY_CA        "dds.sec.auth.identity_ca"
#define DDSSEC_PROP_IDENTITY_CERT      "dds.sec.auth.identity_certificate"
#define DDSSEC_PROP_IDENTITY_PRIVKEY   "dds.sec.auth.private_key"
#define DDSSEC_PROP_IDENTITY_PASSWORD  "dds.sec.auth.password"

#define DDSSEC_PROP_PERM_CA            "dds.sec.access.permissions_ca"
#define DDSSEC_PROP_PERM_GOV_DOC       "dds.sec.access.governance"
#define DDSSEC_PROP_PERM_DOC           "dds.sec.access.permissions"

/* The application must configure the URL property values as appropriate
 *  for the run-time environment:
 */

static const char *auth_ca_file  = "file:./EXAMPLE_identity_ca_cert.pem";
static const char *perm_ca_file  = "file:./EXAMPLE_permissions_ca_cert.pem";

static const char *id_cert_file  =
  "file:./dds_certs/EXAMPLE_participant_identity_cert.pem";
static const char *id_key_file   =
  "file:./dds_certs/private/EXAMPLE_participant_identity_private_key.pem";
static const char * governance_uri  = "file:./GOVERNANCE_signed.p7s";
static const char * permissions_uri = "file:./PERMISSIONS_signed.p7s";
```

```
/*************************************************************************
 *
 ************************************************************************/
static void set_property(DDS::PropertySeq * properties, const char * name,
                         const char * value)
{
  DDS::Property_t prop;
  prop.name      = cpp_strdup(name);
  prop.value     = cpp_strdup(value);
  prop.propagate = 0;
  properties->push_back( prop ); /* shallow copy */
}


...


DDS::DomainParticipantFactory    * dpf = DomainParticipantFactory::get_instance();
DDS::DomainParticipant           * participant = NULL;
DDS::DomainParticipantQos          dp_qos;

dpf->get_default_participant_qos(dp_qos);

DDS::PropertySeq * properties = &dp_qos.properties.value;

/* AUTHENTICATION: */
set_property( properties, DDSSEC_PROP_IDENTITY_CA,      auth_ca_file );
set_property( properties, DDSSEC_PROP_IDENTITY_CERT,    id_cert_file );
set_property( properties, DDSSEC_PROP_IDENTITY_PRIVKEY, id_key_file );
/* (optional) passphrase to acces 'private key' */
// set_property(properties, DDSSEC_PROP_IDENTITY_PASSWORD,
//              "data:,thereisnopassword");

/* ACCESS CONTROL: */
set_property( properties, DDSSEC_PROP_PERM_CA,       perm_ca_file );
set_property( properties, DDSSEC_PROP_PERM_GOV_DOC,  governance_uri );
set_property( properties, DDSSEC_PROP_PERM_DOC,      permissions_uri );

/* PLUGIN CREATION: */
set_property(properties,
         "com.toc.sec.create_plugins",
         ":DSREF_create_plugins");   /* this loads from 'main' executable */
         // "dds_security_log:DSREF_create_plugins");  /* this loads from
                                                        * dynamic library
                                                        * 'dds_security_log' */

/* once all QoS is initialized, create the participant */
participant = dpf->create_participant(domain_id, dp_qos,
                                      NULL, STATUS_MASK_NONE);
```

Chapter 7 Caveats

7.1 rtps_protection = SIGN

~~This setting is currently not interoperable with RTI Connext. We are working (with other members of the OMG) to address related issues in the DDS Security Standard. Once the issues with the standard are resolved, we expect to offer an interoperable implementation of this option.~~ This incompatibility has been resolved as of 2017-10-01.

7.2 rtps_protection = ENCRYPT

This setting is currently not supported by CoreDX DDS. We are working (with other members of the OMG) to address related issues in the DDS Security Standard. Once the issues with the standard are resolved, we expect to offer a working implementation of this option.

Chapter 8 References

1. CoreDX DDS Programmer's Guide,
   http://www.twinoakscomputing.com/documents/CoreDX_DDS_Programmers_Guide_v4.pdf

2. DDS Security Standard v1.0, http://www.omg.org/cgi-bin/doc?formal/16-08-01.pdf

Chapter 9 About Twin Oaks Computing

Twin Oaks Computing, Inc is a company dedicated to developing and delivering quality software solutions. We leverage our technical experience and abilities to provide innovative and useful services in the domain of intelligence systems.

Twin Oaks Computing specializes in high-performance and embedded communications solutions for commercial and DoD applications. Our CoreDX DDS was first released in 2008. In March 2009, Twin Oaks Computing participated in the first public multi-vendor DDS interoperability demonstration. For more information on our products, please visit our website at http://www.twinoakscomputing.com.

Twin Oaks Computing is headquartered in Castle Rock, CO. Our staff has over 30 years of experience developing and supporting DoD systems. We have performed installs and upgrades of critical mission systems at U.S. military facilities around the world. Through this experience, we understand the importance of the systems that collect, manage, and distribute information for the warfighter.

We apply our technical experience to develop solutions in the following Intelligence Domains:

- Tactical Communications - Link 16, IBS, Link 11, Link 11B
- Tactical Data Correlation - Single and Multi-INT Correlation
- Situational Awareness - consolidated display of tactical data

We have Technical experience in the following areas:

- Networking - Ethernet, IP, UDP, TCP, RDMA
- Device Drivers - MILSTD-1553, Serial, Network Interface
- Interprocess Communication - DDS, Sockets, CORBA, RPC, SysV IPC
- Operating Systems - Solaris™, Linux™, FreeBSD™, VxWorks™, and others
- Database Technologies - Sybase™, Oracle™, MySQL™, and others

- Network Services - email servers, HTTP servers, DNS servers, firewalls
- System Security - DCID 6/3 security accreditation
- System Administration - scripting languages, backup/restore, storage management, software installation/configuration

We would be happy to discuss how we can help you. Please contact us at contact@twinoakscomputing.com.

Chapter 10 Contact Information

Have a question? Don't hesitate to contact us by any means convenient for you:

Web Site: http://www.twinoakscomputing.com

Email: support@twinoakscomputing.com

Twitter: @CoreDX_DDS

Phone: 720.733.7906

Address:

230 Third Street
Suite 260
Castle Rock, CO, 80104