Quick Start Guide

for C#

Version 5.0

Castle Rock, CO

80104

TWINOAKS COMPUTING INC.
PRACTICAL MIDDLEWARE EXPERTISE

Welcome

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

This Quick Start will guide you through the basic installation of CoreDX DDS, including installation and compiling and running an example C# application. You will learn how easy it is to integrate CoreDX DDS into an application. This Quick Start Guide is tailored for C# applications, and the examples differ slightly for other programming languages.

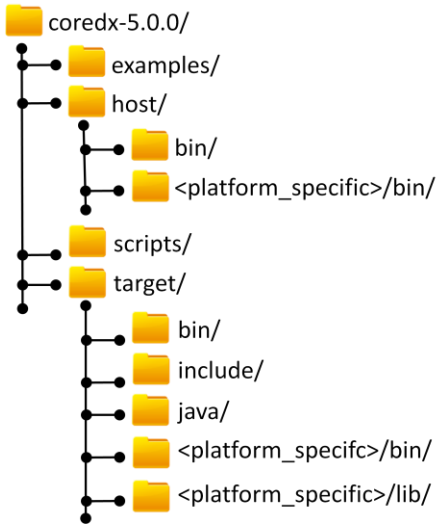
Installation

First things first: get CoreDX DDS onto your development system! Here's what you need to do:

1. Once you have obtained CoreDX DDS from Twin Oaks Computing, unpack the appropriate distribution for your machine somewhere on your system. We'll refer to this directory throughout this guide as COREDX_HOME. For example, on a UNIX system this command will extract the distribution into the current directory:

   ```
   gunzip -c coredx-5.0.0-Linux_2.6_x86_64_gcc5-Release.tgz | tar xf -
   ```

   CoreDX DDS is available for multiple platform architectures, and multiple platform architectures of CoreDX DDS can be installed in the same top level (COREDX_TOP) directory. The directory structure under COREDX_TOP will look like:

2. If you are using an evaluation copy of CoreDX DDS, follow the instructions you received when you downloaded the software to obtain an evaluation license. Otherwise, use the purchased license provided by Twin Oaks Computing. Once you have the license, put this file somewhere on your system. We'll refer to the full path name to this file throughout this guide as LICENSE_FILE.

Building an Application

Next, integrate CoreDX DDS into an application. We've provided a sample data type and application with the distribution (located in COREDX_TOP/examples). You can use these examples or create your own while going though the following steps. Our example Makefiles were built for **mono** (UNIX systems) and Visual Studio **cl.exe** (Windows systems).

1. Create the IDL file for the data type(s) you will use for communications. IDL is a language independent way to specify data types and interfaces and is standardized by the OMG. [NOTE: CoreDX DDS example data types files often use the historical ".ddl" extension. The syntax is IDL compliant, even though the file is named ".ddl".] Here is the "hello world" example IDL provided with the distribution:

hello.idl

```
struct StringMsg
{
  string msg;
};
```

2. Tell the CoreDX DDS type compiler where your evaluation license is located. You can copy the license file into the directory where you will run the compiler; or set the following environment variable, using the full path to your license file. (This assumes a *bash* style shell):

```
% export TWINOAKS_LICENSE_FILE=LICENSE_FILE
```

(Windows style):

```
% set TWINOAKS_LICENSE_FILE=LICENSE_FILE
```

3. Compile the IDL to generate the type specific code using the CoreDX DDS type compiler. The type compiler is a *host* tool, and is located in the host subdirectory. Actually, there may be more than one type compiler, if you have multiple platform versions of CoreDX DDS installed. In this case, choose the appropriate compiler for your architecture. Assuming the name of the IDL file is hello.idl (Linux example):

```
% COREDX_TOP/host/bin/Linux_2.6_x86_64_gcc5_coredx_ddl
   -f hello.idl -l csharp
```

(Windows example):

```
% COREDX_TOP\host\bin\Windows_x86_vs2008_coredx_ddl.exe
   -f hello.idl -l csharp
```

The compilation will generate the following files in a "gen" subdirectory (file names are based on the data type name):

- StringMsg.cs
- StringMsgTypeSupport.cs
- StringMsgDataReader.cs
- StringMsgDataWriter.cs

4. Create code to publish data of this data type. Our sample Hello World publisher is located in COREDX_HOME/examples/hello_csharp/hello_pub.cs.

5. Create code to subscribe to data of this data type. Our sample Hello World subscriber is located in COREDX_HOME/examples/hello_csharp/hello_sub.cs.

6. Compile your application(s). Our Hello World example creates two applications, one for the publisher and one for the subscriber. This is not necessary, and is completely dependent on your application architecture. Your application will require the objects from the generated type support code above, as well as your publisher and/or subscriber code.

   CoreDX DDS will require the following paths and libraries when compiling your application:
   (Linux):

```
Library Path:
        -L${COREDX_TOP}/target/${COREDX_TARGET}/lib
Libraries:  dds_csharp.so (native library)
            coredx_csharp.dll
```

   (Windows):

```
Library Path: -L$(COREDX_TOP)\target\$(COREDX_TARGET)\lib
Libraries:    coredx_csharp.dll dds_csharp.dll
```

   We've provided a Makefile and NMakefile for compiling our example, along with a Windows VisualStudio solution file. You can use these as a reference for compiling your application. Our Makefiles (and VisualStudio solutions) require three environment variables to successfully compile the examples:

   1. COREDX_TOP
   2. COREDX_HOST
   3. COREDX_TARGET

   The COREDX_TOP is a path name to the location of your CoreDX DDS distribution(s). COREDX_HOST and COREDX_TARGET are the platform architectures you are compiling on and compiling for (these can be the same). These values can be set manually, or determined by running the script: COREDX_TOP/scripts/cdxenv.sh or cdxenv.bat.

Using our Makefiles

Our Makefiles will run the CoreDX DDS type compiler to generate the type specific code as well as compile the applications. To compile the Hello World sample application using our Linux Makefiles you will need mono and coredx_ddl in your path. To compile using the Windows NMakefile you will need the MS Visual Studio environment (32-bit or 64-bit as appropriate for the CoreDX DDS package you are using) and coredx_ddl in your path. Then, simply type 'make' for UNIX or 'nmake –f NMakefile' for Windows in the appropriate directory.

This will compile two applications: hello_pub and hello_sub.

Using Visual Studio

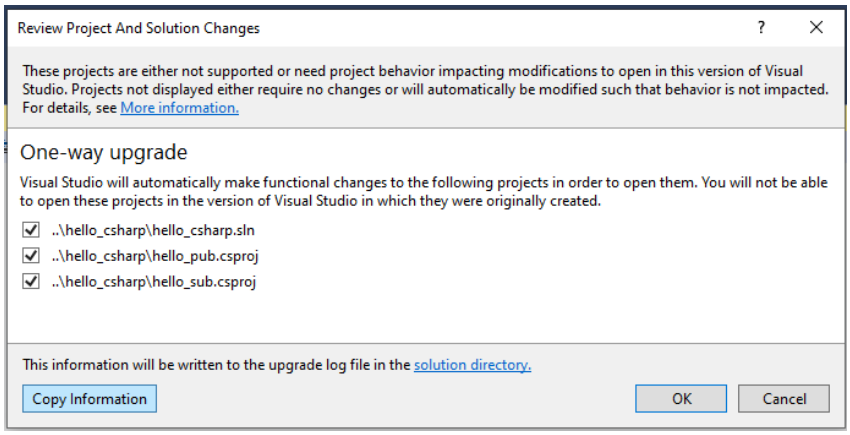
To compile using Microsoft's Visual Studio, the environment variables listed above, including the TWINOAKS_LICENSE_FILE variable must be set. They can be set as part of your user configuration, or you can set them in a Visual Studio command prompt. If you set the environment variables in a Visual Studio command prompt, you must then run Visual Studio from that same command prompt.

Next, open the appropriate example Visual Studio project solution file. In the hello_c example, there will be subdirectories for various versions of Visual Studio containing the solution and projects configured for that version of Visual Studio. If your version of Visual Studio is not included, use the closest (older) version to your own.

This solution contains 2 projects: hello_pub and hello_sub.

```
File->Open
Navigate to
COREDX_TOP\examples\vs<version>\hello_csharp, and then
select "hello_csharp.sln"
```

If the Visual Studio solution was built with an older version of Visual Studio than you are using, you will be prompted to 'Upgrade' the solution:



Click "OK" to upgrade the solution and contained projects.

The 'conversion' process to a newer Visual Studio version may leave the target .NET Framework version at a default of 2.0. In order to get the example to build, you will need to manually update the .NET Framework version to 3.5.

To update the project's framework version to 3.5:

1. Click on (open) the 'Properties' sheet under the hello_pub project

2. Select ".NET framework 3.5" in the 'Target framework' option menu
3. A dialog box will pop-up, asking for confirmation about changing the target framework: click "Yes"

Note, only the hello_pub project needs to be updated, the hello_sub will automatically pick up the change.

Next, compile the hello_world example:

```
In the "Solution Explorer", Right click on the
'hello_csharp' solution
Select "build solution"
```

This will compile two applications: hello_pub and hello_sub.

Running a Test Application

You've written some code, generated some code, and compiled it all. Now for seeing it all work! You will need at least one environment variable to run:

TWINOAKS_LICENSE_FILE = The full path to your evaluation license

On Windows

In addition, you will need to set your PATH environment variable to include the path to the CoreDX DDS dll's.

```
set PATH=%PATH%;%COREDX_TOP%\target\%COREDX_TARGET%\lib
```

Run your application(s). The sample Hello World has two applications: hello_pub and hello_sub. You can run these from a Windows command line:

```
COREDX_TOP\examples\hello_csharp\hello_sub.exe
COREDX_TOP\examples\hello_csharp\hello_pub.exe
```

Or, you can run these two projects from VisualStudio. The project files are configured to copy the required '.dll' files into the target bin directory (either bin/Release or bin/Debug, depending on the build type selected). Therefore, the exe files will run in place without further configuration.

On Linux

To run the examples on Linux, you will need to configure your LD_LIBRARY_PATH and MONO_PATH environment variables:

```
export LD_LIBRARY_PATH=
${COREDX_TOP}/target/${COREDX_TARGET}/lib:${LD_LIBRARY_PATH}
```

```
export MONO_PATH=
${COREDX_TOP}/target/${COREDX_TARGET}/lib:${MONO_PATH}
```

Then run the applications:

```
COREDX_TOP/examples/hello_csharp/hello_sub
COREDX_TOP/examples/hello_csharp/hello_pub
```

Congratulations! You have now built and run two applications that are communicating using CoreDX DDS.

Figure 1 shows a picture of what you have built:

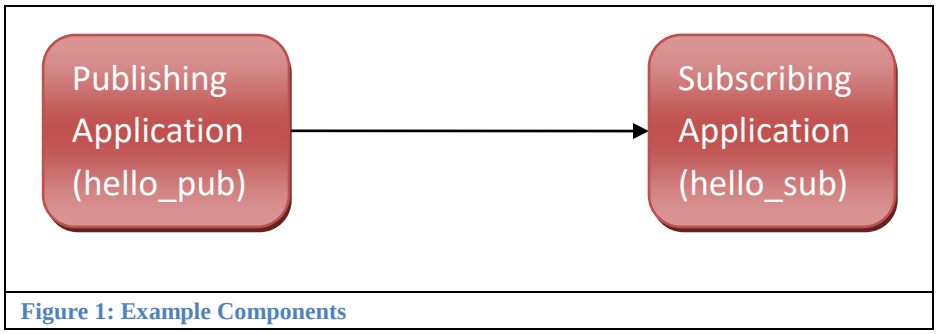


Figure 1: Example Components

You can run multiple Publishers and multiple Subscribers to immediately see the dynamic nature of the DDS network infrastructure. These Publishers and Subscribers can be run on the same host or on multiple hosts across a network.

A few notes about the Transport

The CoreDX DDS transport conforms to the Real-Time Publish-Subscribe (RTPS) Wire Protocol. This transport does not use a stand-alone transport daemon, and does not require configuration of any operating system services.

Basic Transport Configuration

The CoreDX DDS examples will run in most development environments without any further configuration. However, in some cases, basic transport configuration is desirable, or will help limit network traffic to desired interfaces. The CoreDX DDS transport may be configured via API and/or environment variables. The following table lists some of the common configuration items and the corresponding environment variables that may be used to modify their configuration.

Environment Variable	Values	Description
COREDX_IP_ADDR	Valid, Local IP address	As part of discovery, each DDS DomainParticipant advertises local IP addresses that peers can use to communicate with it. If your machine has multiple network interfaces, CoreDX DDS will by default advertise (and use) all interfaces for DDS communications. This may generate unnecessary network traffic on some of those networks. This environment variable will limit DDS traffic to just one interface – the interfaces specified by the IP address.
COREDX_USE_MULTICAST	"YES", "NO"	By default, CoreDX DDS RTPS will use multicast for data communications between DDS participants where it can. To specify unicast data communications, set this environment variable to "NO". This only effects data communications, discovery will still use multicast.
COREDX_MIN_TX_BUFFER_SIZE	400 - 65400	(in bytes) The CoreDX DDS RTPS transport will combine multiple data packets to send over the network to reduce network overhead and improve performance. By default, the transmit buffer size is *dynamic*. For cases where IP fragmentation and reassembly is not implemented well in either the network hardware or operating system, it is necessary to fix the transmit buffer to a smaller size. This environment variable controls the minimum size of the transmit buffer on every DataWriter within a DomainParticipant (including Built-in

Environment Variable	Values	Description
		DataWriters).
COREDX_MAX_TX_BUFFER_SIZE	401-65400	(in bytes) The CoreDX DDS RTPS transport will combine multiple data packets to send over the network to reduce network overhead and improve performance. By default, the transmit buffer size is *dynamic*. For cases where IP fragmentation and reassembly is not implemented well in either the network hardware or operating system, it is necessary to fix the transmit buffer to a smaller size. This environment variable controls the maximum size of the transmit buffer on every DataWriter within a DomainParticipant (including Built-in DataWriters).

About License Files

CoreDX DDS uses development and run-time license keys. A development license key is required for using the CoreDX DDS type compiler (coredx_ddl). A run-time license key is required for making CoreDX DDS library function calls. Both licenses are contained in a license file provided by Twin Oaks Computing. Here is an example license file containing evaluation licenses for both development and run-time:

coredx.lic

```
#=====================================================================
# CoreDX DDS Evaluation License file
#
# Created:  <today> by Twin Oaks Computing, Inc.
# Contains: 30 day evaluation development licenses, evaluation run-time licenses
#
#=====================================================================

LICENSE PRODUCT=coredx_ddl EXP=2020_06_01 BUILD=Evaluation
CUSTOMER=Company_X SIG=
4ccad329d5a10b93460ff3b249cea6733f6bd408d22b5fe9cb2a2c69b0d575e69a5d
c14b436b90c2ed6b516930452b862133cf7d2a9301d46ce99865f78c998311adeb99
3f68da82b74f1583511edab1d0de61dbe065f38955dd6596f0b564639fed231b1af8
61b6df122040173804e0e61b0dba37d6913cfc66d319217df099
LICENSE PRODUCT=coredx_c EXP=2020_06_01 BUILD=Evaluation
CUSTOMER=Company_X SIG=
30b3c5d6f941c5ff5e46384eb1b74bd1809dfbd53ca11fa4d7442054bb260846588
c4bd7a5c7f7a986a12905b22dbdc428a67ee2d2c806ed5f1a14c35deb03e3a8ce6a
2fda8fdb7e5728c3103f239b51aca3b3911901e2e959fe020a21b7b7cb72dee8ca8
da8fa73cc69d3572738259025c212815aef2f94111580f51583e437
```

This evaluation license file contains two LICENSE lines. The first is the development license key for the CoreDX DDS type compiler. The second is the run-time license key for the CoreDX DDS library. All evaluation licenses have expiration dates. In the example file above, the licenses expire on June 1, 2020.

The easiest way to configure CoreDX DDS software is by using the environment variable: TWINOAKS_LICENSE_FILE. This environment variable can contain either:

- The fully qualified name of the license file
- The entire LICENSE line from the license file contained in angle brackets: < >

For development (to run the coredx_ddl compiler), you must set the TWINOAKS_LICENSE_FILE environment variable to the license file.

For run-time, you can use either method listed above. If you have access to the license file from your run-time environment, this is the simplest way to use the license. Simply set a TWINOAKS_LICENSE_FILE environment variable to the license file.

If you do not have access to the license file at run-time, you can set the TWINOAKS_LICENSE_FILE environment variable to the LICENSE line. For the run-time license in the above example license file, set your TWINOAKS_LICENSE_FILE like:

```
% export TWINOAKS_LICENSE_FILE="<LICENSE PRODUCT=coredx_c BUILD=Evaluation
EXP=2020_06_01 CUSTOMER=Company_X SIG=30b3c5d6f941c5ff5e46384eb1b74bd1
809dfbd53ca11fa4d7442054bb260846588c4bd7a5c7f7a986a12905b22dbdc428a67ee
2d2c806ed5f1a14c35deb03e3a8ce6a2fda8fdb7e5728c3103f239b51aca3b3911901e2
e959fe020a21b7b7cb72dee8ca8da8fa73cc69d3572738259025c212815aef2f9411158
0f51583e437>"
```

A few notes about the Data Type Compiler (coredx_ddl)

The coredx_ddl compiler handles a few command line arguments. The following briefly describes the command line options and arguments. (The '-h' argument can be used to list all the command line arguments accepted by the coredx_ddl compiler.)

-f <filename>	Specifies the IDL file to compile. This is a required argument.
-l <language>	Specifies the language to generate: 'csharp' for C# code. The default if not specified is 'c'.
-d <dest dir>	Specifies the destination directory for generated files. By default, this is the current directory.

Changes from Previous Release

For current release notes, visit the Twin Oaks Computing website at:
http://www.twinoakscomputing.com/documents/CoreDX_DDS_release_notes.txt

Contact Information

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

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

Support:

Email: support@twinoakscomputing.com

Phone: 720.733.7906

Primary Sales Office:

Email: sales@twinoakscomputing.com

Phone: 720.733.7906

EMEA Sales Office:

Email: emea.sales@twinoakscomputing.com

Phone: +44 7717 790404

About Twin Oaks Computing

With corporate headquarters located in Castle Rock, Colorado, USA, Twin Oaks Computing 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 data communications. Founded in 2005, Twin Oaks Computing, Inc delivered the first version of CoreDX DDS in 2008. The next two years saw deliveries to over 100 customers around the world. We continue to provide world class support to these customers while ever expanding.

Contact

Twin Oaks Computing, Inc.
(720) 733-7906
230 Third Street, Suite 260
Castle Rock, CO. 80104
www.twinoakscomputing.com