This sample can act as a generic data receiver, to be used with coreDS C++. You can request a free trial on https://www.ds.tools/contact-us/trial-request/

The Sample project uses the following coreDS concepts:

- Connect (createFederationExecution, joinFederationExection,
- Disconnect
- Receive Update Object (Receive EntityStatePDU or UpdateAttributeValues)
- Delete Objects (EntityStatePDU timeout or RemoveObjectInstance)

Runtime commands:

- Pressing S will stop the connection then open the configuration selection window. This allows to change from one distributed simulation protocol to the other without having to close the application.
- Pressing Q will quit the application.

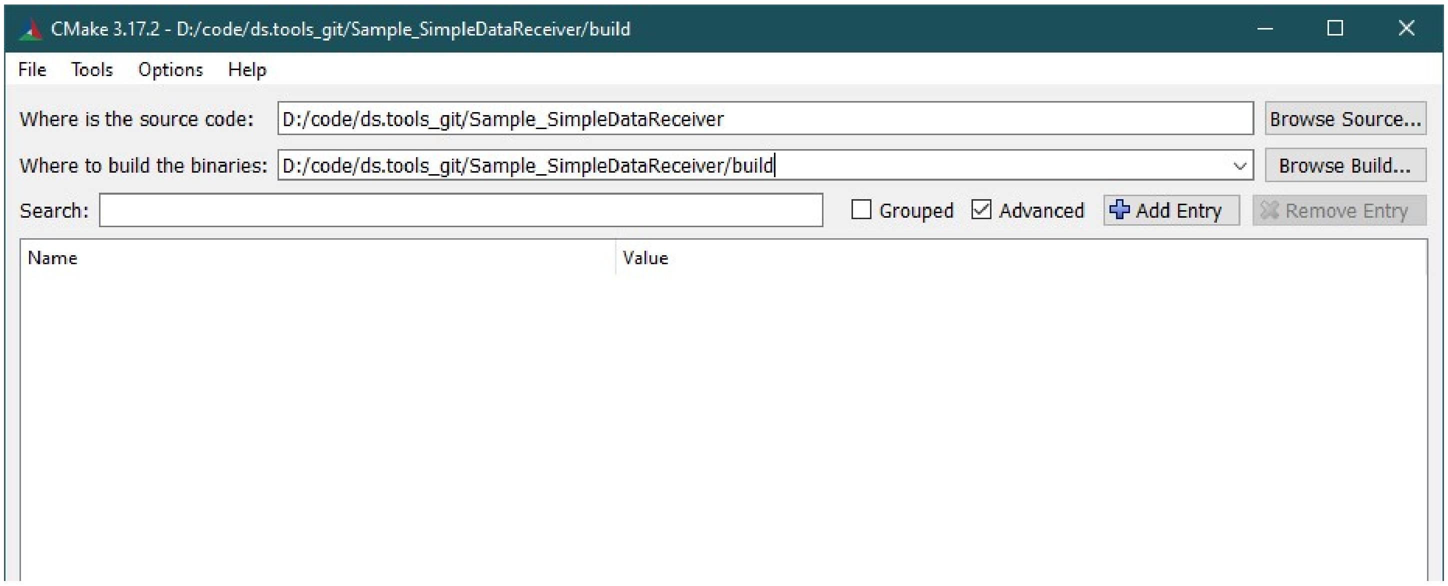
Getting started

How to compile

This tool can be built using CMake (www.cmake.org). Only the x64 built is supported on Windows.

If you've never used CMake before, let's see how it works first.

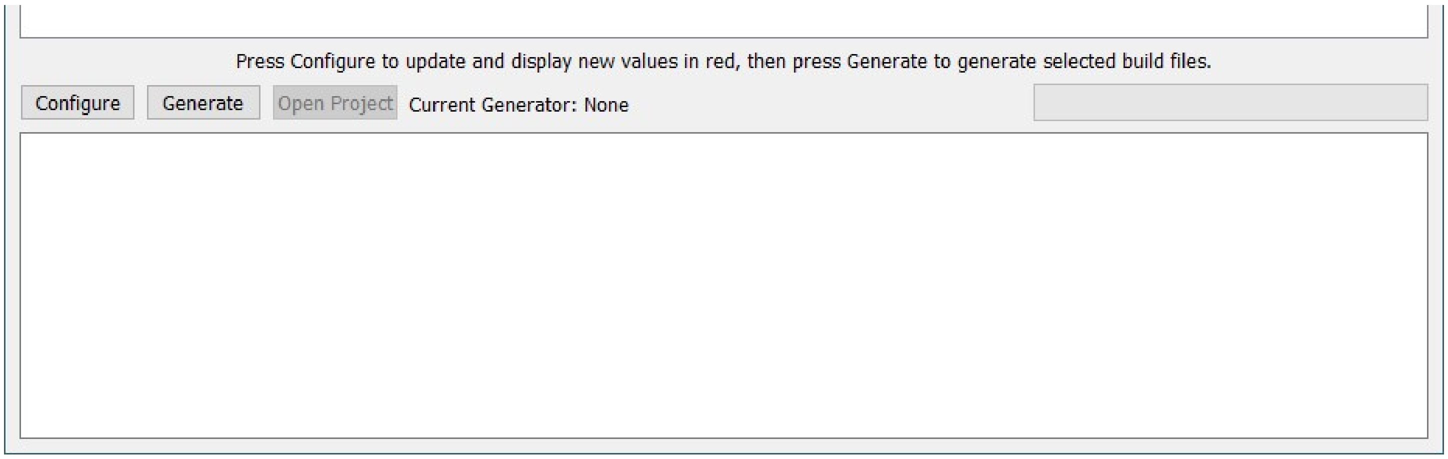
First, download, install and launch cmake-gui



Where is the source code: D:/code/ds.tools_git/Sample_SimpleDataReceiver

Where to build the binaries: D:/code/ds.tools_git/Sample_SimpleDataReceiver/build

Search:



Press Configure to update and display new values in red, then press Generate to generate selected build files.

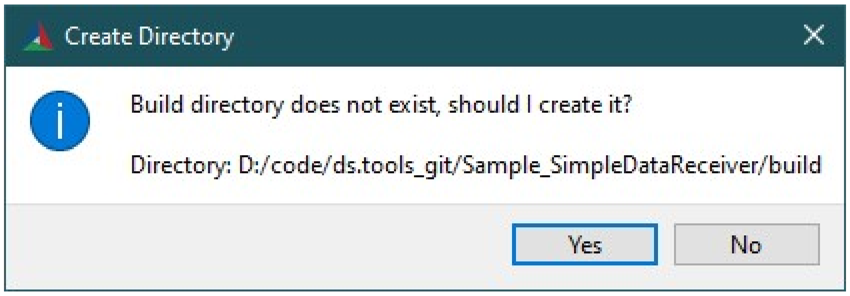
Current Generator: None

Click on "Browse Source" then select the sample source folder (where the CMakeList.txt file is)

The click on "Browse Build..." to select where the generated, and compiled files, will be saved.

Once you are done, click on the "Configure" button. This will parse the CMakeList.txt, extract some configuration variables and make sure everything is correct.

If the build folder does not exist, you will be asked to create it:



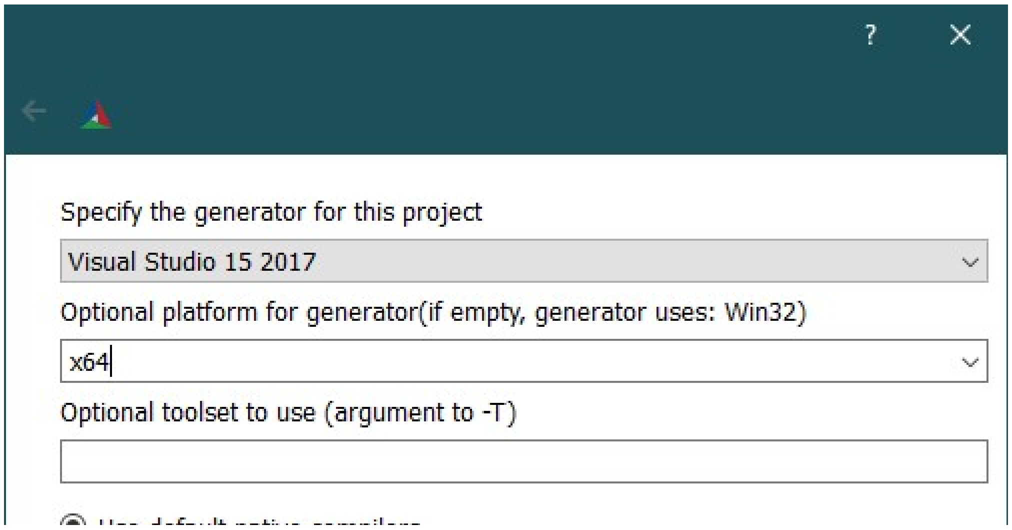
Create Directory

Build directory does not exist, should I create it?

Directory: D:/code/ds.tools_git/Sample_SimpleDataReceiver/build

Then you will be asked to select the compiler that you want to use. This will depends on the installed coreDS Version - be careful, incorrect configuration at this step could cause hard to catch crashes.

If you are build a Win64 target, make sur you select "x64" in the "Optional platform for generation"



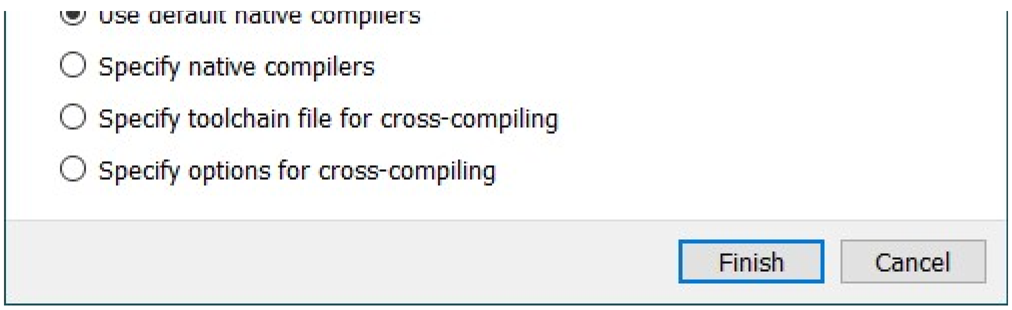
Specify the generator for this project

Visual Studio 15 2017

Optional platform for generator(if empty, generator uses: Win32)

x64

Optional toolset to use (argument to -T)

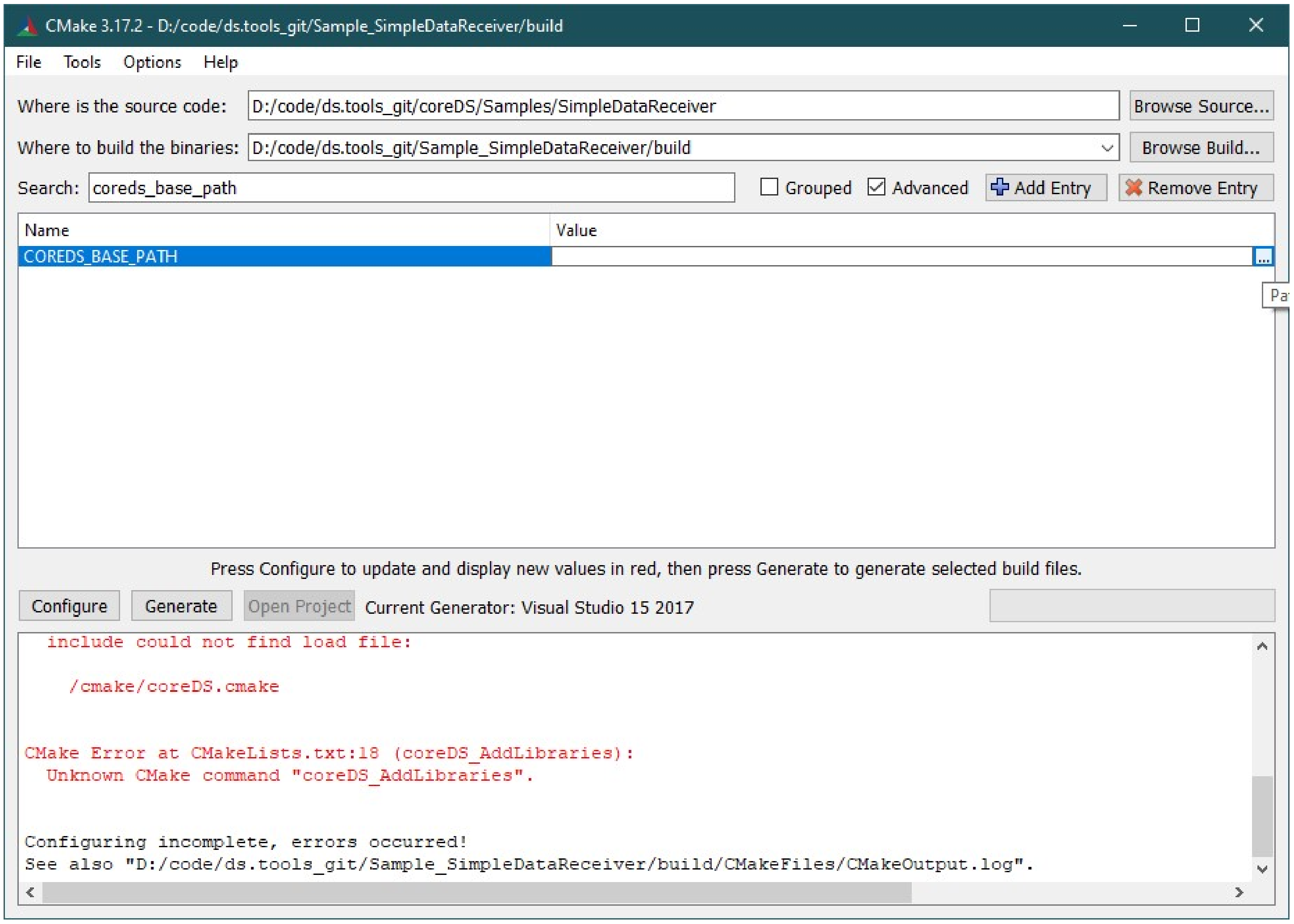


[✓ Use default native compilers]
Specify native compilers
Specify toolchain file for cross-compiling
Specify options for cross-compiling
Finish
Cancel

At this point, you will get and error and this is normal: CMake can't find coreDS!

Now, we need to set two variables:
COREDS_BASE_PATH:
CMAKE_INSTALL_PATH:

First, let's configure the coreDS path. You can search for a specific variable using the "Search" file.



Where is the source code: D:/code/ds.tools_git/coreDS/Samples/SimpleDataReceiver
Where to build the binaries: D:/code/ds.tools_git/Sample_SimpleDataReceiver/build
Search: coreds_base_path

Name	Value
COREDS_BASE_PATH	

Press Configure to update and display new values in red, then press Generate to generate selected build files.

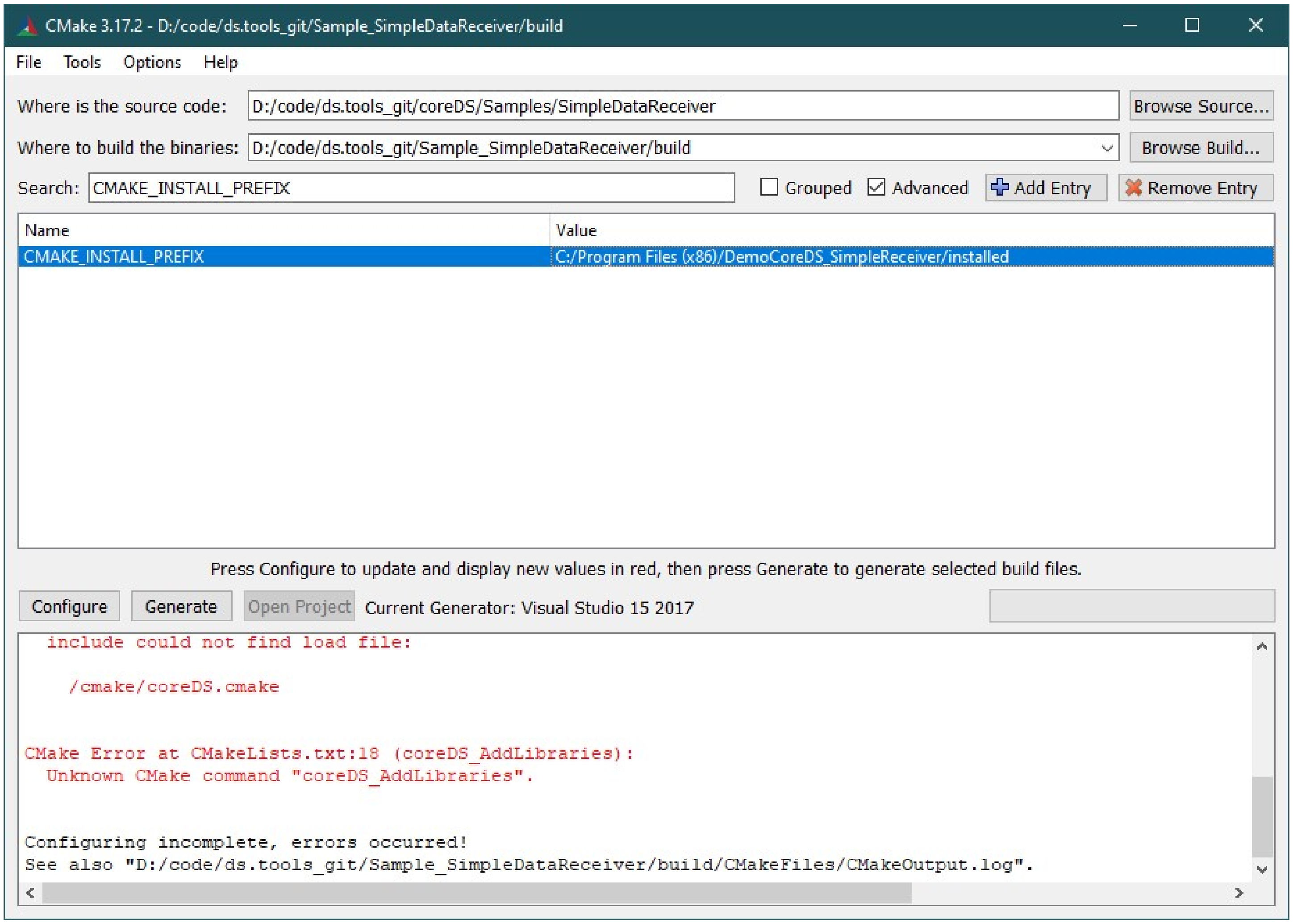
Current Generator: Visual Studio 15 2017

```
  include could not find load file:

    /cmake/coreDS.cmake


CMake Error at CMakeLists.txt:18 (coreDS_AddLibraries):
  Unknown CMake command "coreDS_AddLibraries".


Configuring incomplete, errors occurred!
See also "D:/code/ds.tools_git/Sample_SimpleDataReceiver/build/CMakeFiles/CMakeOutput.log".
```

Once you have found the correct varible, click on the "..." button to select the coreDS installation folder.

Set the "COREDS_BASE_PATH" to the root of your coreDS installation folder

Then, do the same to set CMAKE_INSTALL_PATH. The CMAKE_INSTALL_PATH is where the final binaries will be installed. Default is to install in the Program Files folder but in most cases, this is not recommanded (administrator privileges are required). We simply install to a local folder as shown below.



Where is the source code: D:/code/ds.tools_git/coreDS/Samples/SimpleDataReceiver

Where to build the binaries: D:/code/ds.tools_git/Sample_SimpleDataReceiver/build

Search: CMAKE_INSTALL_PREFIX

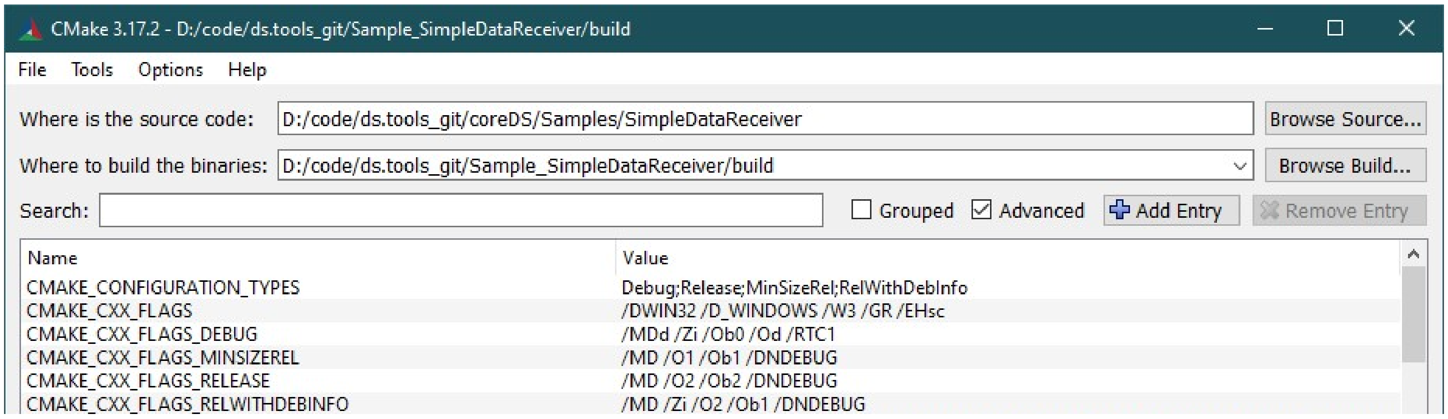
Name	Value
CMAKE_INSTALL_PREFIX	C:/Program Files (x86)/DemoCoreDS_SimpleReceiver/installed

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Visual Studio 15 2017

```
  include could not find load file:

    /cmake/coreDS.cmake


CMake Error at CMakeLists.txt:18 (coreDS_AddLibraries):
  Unknown CMake command "coreDS_AddLibraries".


Configuring incomplete, errors occurred!
See also "D:/code/ds.tools_git/Sample_SimpleDataReceiver/build/CMakeFiles/CMakeOutput.log".
```

Finally, click on the "Generate", the the "Open Project" button. This will launch the configured VisualStudio version. Make sure there is no error.

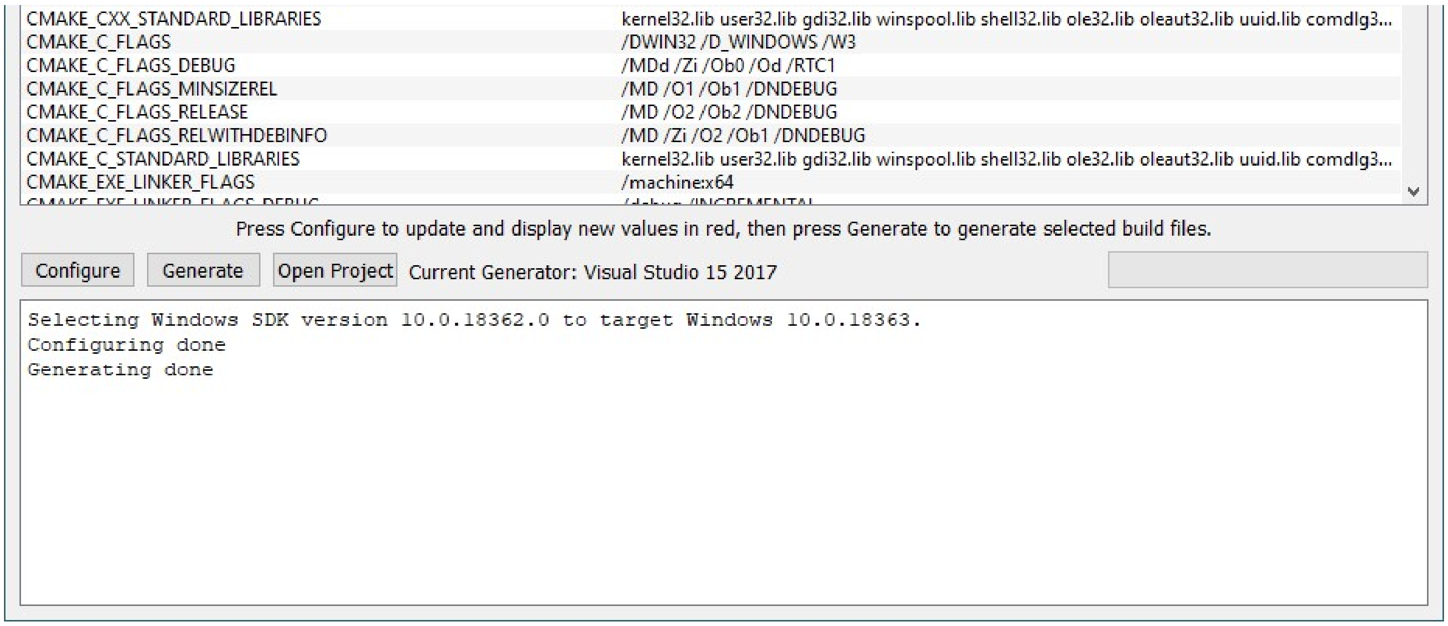


Where is the source code: D:/code/ds.tools_git/coreDS/Samples/SimpleDataReceiver

Where to build the binaries: D:/code/ds.tools_git/Sample_SimpleDataReceiver/build

Search:

Name	Value
CMAKE_CONFIGURATION_TYPES	Debug;Release;MinSizeRel;RelWithDebInfo
CMAKE_CXX_FLAGS	/DWIN32 /D_WINDOWS /W3 /GR /EHsc
CMAKE_CXX_FLAGS_DEBUG	/MDd /Zi /Ob0 /Od /RTC1
CMAKE_CXX_FLAGS_MINSIZEREL	/MD /O1 /Ob1 /DNDEBUG
CMAKE_CXX_FLAGS_RELEASE	/MD /O2 /Ob2 /DNDEBUG
CMAKE_CXX_FLAGS_RELWITHDEBINFO	/MD /Zi /O2 /Ob1 /DNDEBUG



CMAKE_CXX_STANDARD_LIBRARIES	kernel32.lib user32.lib gdi32.lib winspool.lib shell32.lib ole32.lib oleaut32.lib uuid.lib comdlg3...
CMAKE_C_FLAGS	/DWIN32 /D_WINDOWS /W3
CMAKE_C_FLAGS_DEBUG	/MDd /Zi /Ob0 /Od /RTC1
CMAKE_C_FLAGS_MINSIZEREL	/MD /O1 /Ob1 /DNDEBUG
CMAKE_C_FLAGS_RELEASE	/MD /O2 /Ob2 /DNDEBUG
CMAKE_C_FLAGS_RELWITHDEBINFO	/MD /Zi /O2 /Ob1 /DNDEBUG
CMAKE_C_STANDARD_LIBRARIES	kernel32.lib user32.lib gdi32.lib winspool.lib shell32.lib ole32.lib oleaut32.lib uuid.lib comdlg3...
CMAKE_EXE_LINKER_FLAGS	/machine:x64

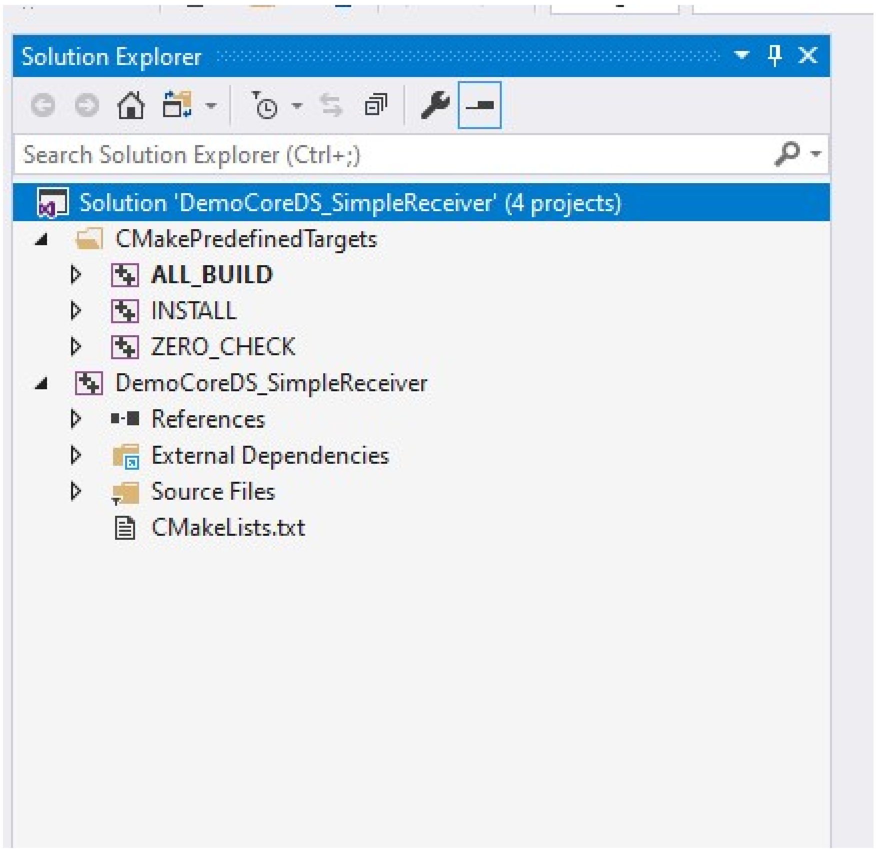
Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Visual Studio 15 2017

```
Selecting Windows SDK version 10.0.18362.0 to target Windows 10.0.18363.
Configuring done
Generating done
```

Once you are within VisualStudio, you will notice a few targets:

"All_BUILD", "INSTALL", "ZERO_CHECK" and the main project "DemoCoreDS_SimpleReceiver"



```
Solution 'DemoCoreDS_SimpleReceiver' (4 projects)
  CMakePredefinedTargets
    ALL_BUILD
    INSTALL
    ZERO_CHECK
  DemoCoreDS_SimpleReceiver
    References
    External Dependencies
    Source Files
    CMakeLists.txt
```

Let's focus on the two most important: the project and the "INSTALL" target. Building the "DemoCoreDS_SimpleReceiver" will compile the project. The "INSTALL" target on the other hand will compile and install all the dependencies into the CMAKE_INSTALL_PATH as configured before. You will need to "Install" the project in order to run it, otherwise you will be missing key dependencies.

Now, it's time to take a look at some code!

Header files

Most functions are included in these header files

```
#include "DSimUI.h"                       // coreDS API UI include
#include "DSimAPI.hpp"          // coreDS API main functions
```

Define the data to be exchanged

The first step is to define which objects, object attributes, messages and message parameters your simulator will support. Keep in mind that the names you define are not related to the distributed simulation protocol you plan on using. These names will only be used internally when using the coreDS Mapping interface.

```
std::map<std::string, std::set< std::string > > lInputObjectVariables;
lInputObjectVariables["ItemInput"].insert("X");
lInputObjectVariables["ItemInput"].insert("Y");
lInputObjectVariables["ItemInput"].insert("Z");

lInputObjectVariables["ItemInput1"].insert("X");
lInputObjectVariables["ItemInput1"].insert("Y");
lInputObjectVariables["ItemInput1"].insert("Z");

setInputObjectVariables(lInputObjectVariables);

std::map<std::string, std::set< std::string > > linputMessageVariables;
linputMessageVariables["CollisionIn"].insert("CollisionX");
linputMessageVariables["CollisionIn"].insert("CollisionY");
linputMessageVariables["CollisionIn"].insert("CollisionZ");
setInputMessageVariables(linputMessageVariables);
```

You will notice that we create a map containing the objects or messages name then fill a set with the corresponding attributes/parameters. These data structures are then being registered using

setInputObjectVariables: Register objects getting from the distributed simulation network to your simulator.
setInputMessageVariables: Register messages/interactions getting from the distributed simulation

network to your simulator.
setOutputObjectVariables: Register objects getting from your simulator to the distributed simulation network.
setOutputMessageVariables: Register messages/interactions getting from your simulator to the distributed simulation network.

These four functions are global and will affect coreDS UIs only.

In this particular sample, we can receive two objects type, "ItemInput" and "ItemInput1". We can also receive a message, "CollisionIn". All messages and objects have 3 properties, X, Y, Z.

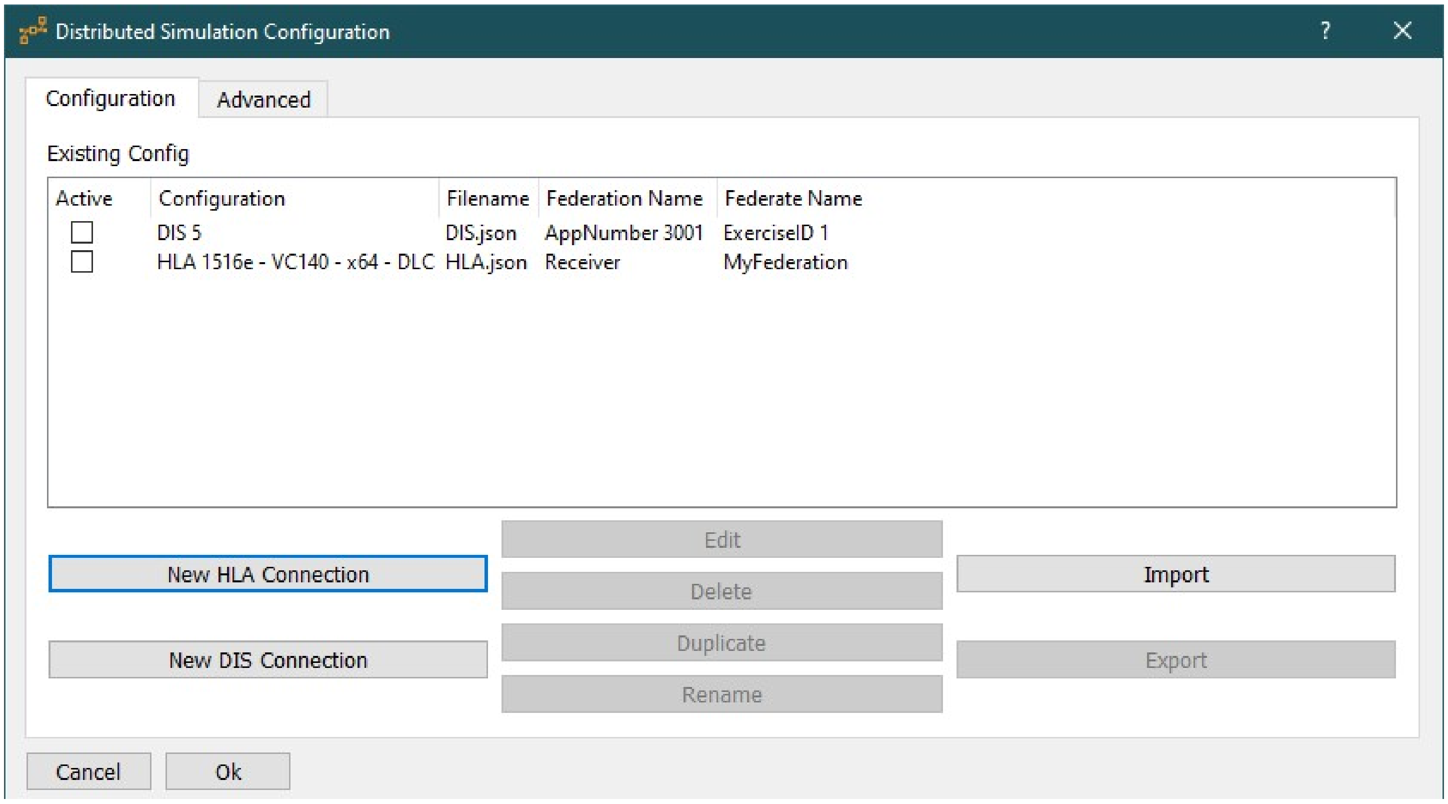
Understanding the UIs

coreDS has two main configuration interface.

Configuration Selection

This sample comes with 4 pre-configured settings:

- DIS: DIS v6
- HLA: HLA 1516e with RPRFOM 2.0



Active	Configuration	Filename	Federation Name	Federate Name
☐	DIS 5	DIS.json	AppNumber 3001	ExerciseID 1
☐	HLA 1516e - VC140 - x64 - DLC	HLA.json	Receiver	MyFederation

The listed configurations are read from the "config" folder, next to your executable. The configuration is a simple JSON file. Right now, it is not possible to change the configuration without the UIs.

Configuration Edition

In order to connect to a distributed simulation framework, you have to create or use an existing configuration.

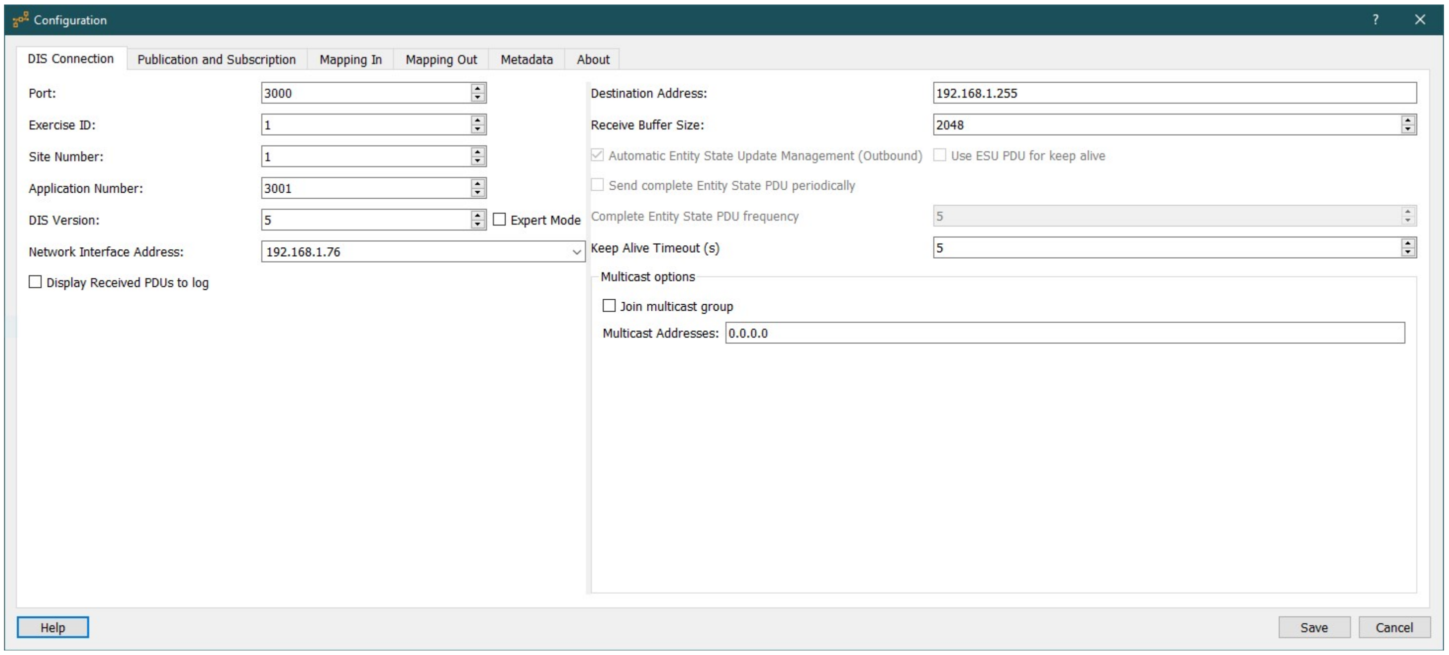
To create a new HLA (High-Level Architecture) configuration, click on the “New HLA Connection” button.

To create a new DIS (Distributed Interactive Simulation) configuration, click on the “New DIS Connection” button.

To enable a configuration, the “Active” checkbox must be enabled next it. Only one configuration can be activated for a simulator at the same time. To disable coreDS, make sure no configuration is active.

Editing a DIS or a HLA configuration is pretty much the same, exept for the first pane which includes parameters specifics to each protocol.

DIS



Port: 3000
Exercise ID: 1
Site Number: 1
Application Number: 3001
DIS Version: 5 ☐ Expert Mode
Network Interface Address: 192.168.1.76
☐ Display Received PDUs to log
Destination Address: 192.168.1.255
Receive Buffer Size: 2048
☑ Automatic Entity State Update Management (Outbound) ☐ Use ESU PDU for keep alive
☐ Send complete Entity State PDU periodically
Complete Entity State PDU frequency: 5
Keep Alive Timeout (s): 5
☐ Join multicast group
Multicast Addresses: 0.0.0.0

Port Port on which coreDS will listen for data.

Exercice ID: Each DIS exercise shall be distinguished from other exercises by the use of an exercice number.

Site Number: A site is defined as a facility, an installation, an organizational unit, or a geographic location that has one or more simulation applications capable of participating in a distributed event. A facility, an installation, an organizational unit, or a geographic location may have multiple sites associated with it. Each site participating in an event (e.g., training exercise, experiment, and test) shall be assigned a unique Site Number that is different from any other site that is part of the same event.

Application Number:Each application participating in an event (e.g., training exercise) shall be assigned a unique Application Number for the site with which the application is associated.

DIS Version: Version of the DIS standard to be used.

Export mode: When enabled, enumerators can be configuration manually.

Display received PDUs to Log: When checked, all the received PDU, including decoded data, will be sent to the log.

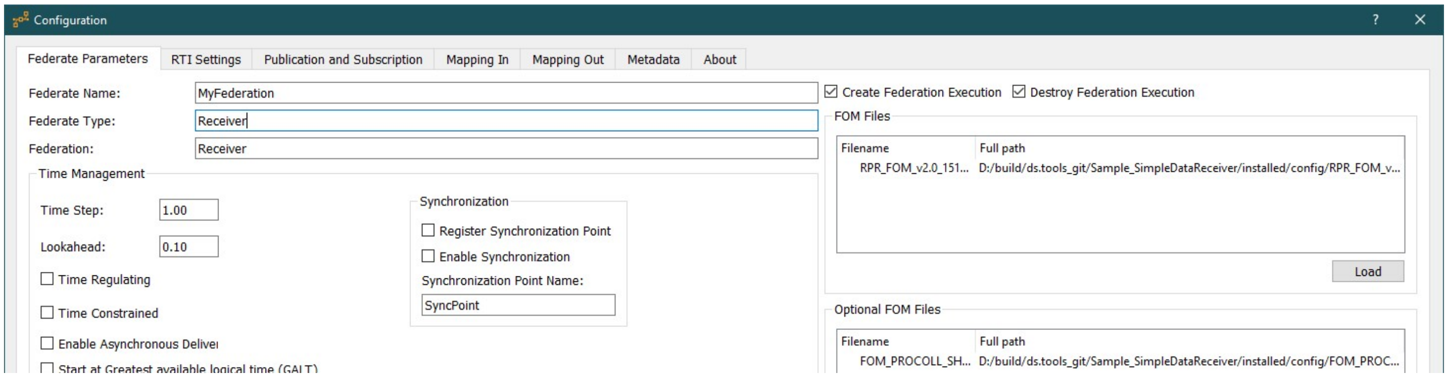
Network Interface Address: Network adapter sending and receiving packets.

Destination Address: Destination address of the UDP Packet. Most of the time, this value should be a broadcast address from the same subnet

Receive Buffer size: Maximum size of the receive buffer. If a packet is bigger than the set value, the packet will be truncated.

Multicast options
Join multicast group: Toggle to join a multicast group.
Multicast addresses: Multicast group address

HLA



Federate Name: MyFederation
Federate Type: Receiver
Federation: Receiver

☑ Create Federation Execution ☑ Destroy Federation Execution

Time Management
Time Step: 1.00
Lookahead: 0.10
☐ Time Regulating
☐ Time Constrained
☐ Enable Asynchronous Delivei
☐ Start at Greatest available logical time (GALT)

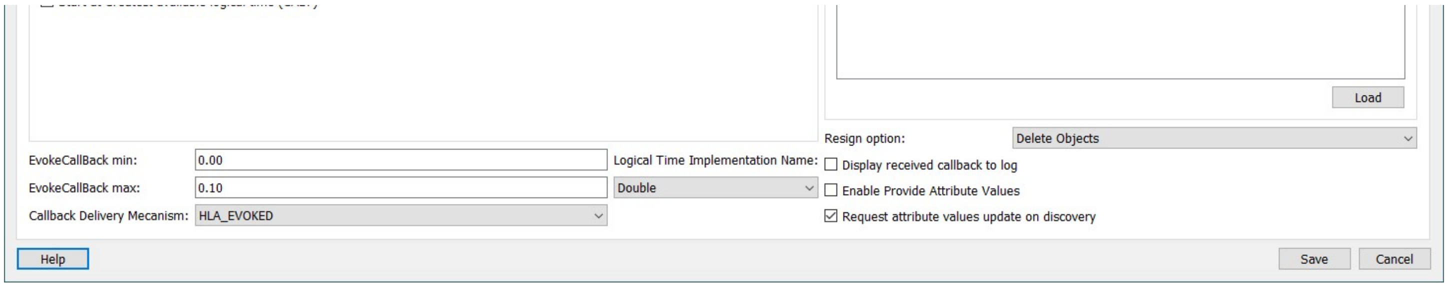
Synchronization
☐ Register Synchronization Point
☐ Enable Synchronization
Synchronization Point Name: SyncPoint

FOM Files

Filename	Full path
RPR_FOM_v2.0_151...	D:/build/ds.tools_git/Sample_SimpleDataReceiver/installed/config/RPR_FOM_v...

Optional FOM Files

Filename	Full path
FOM_PROCOLL_SH...	D:/build/ds.tools_git/Sample_SimpleDataReceiver/installed/config/FOM_PROC...



EvokeCallBack min: 0.00
EvokeCallBack max: 0.10
Callback Delivery Mecanism: HLA_EVOKED
Logical Time Implementation Name: Double
Resign option: Delete Objects
☐ Display received callback to log
☐ Enable Provide Attribute Values
☑ Request attribute values update on discovery

Federate Name: Name of the federate in the given federation.
Federate Type: Federate type.
Federation: Name of the federation you want to join or create.
Create Federation Execution: Try to create the federation before joining.
Destroy Federation Execution: Try to destroy the federation after resignation.

FED / FDD File
The FOM Files section let you specify Federation Object Model (FOM) used for the federation creation. You can select a HLA 1.3 compliant fed file (.fed), a IEEE 1516 compliant fdd file (.xml or .fdd) or a IEEE 1516-2010 compliant fdd file (.xml or .fdd). If you are creating an IEEE 1516-2010 federate, you can select more than one fdd file (additional file are named FOM Module). Press the “Load” button in order to populate the tree view with the FED file information. You a can remove a fdd file by right-clicking on it and selecting “Remove”.

NOTE: You must select a FOM file compatible with your RTI and standard. That is, if you use a 1516 RTI, you must use a FDD file.

Once the fed file is loaded, you can navigate the Object and Interaction classes defined by the FED file and access the Attributes and Parameters from the tree view.

Time management:
Time Step: Increment of time for this federate

Lookahead: Look ahead used by this federate

Time Regulating: If set, the federate will be time regulating

Time Constrained: If set, the federate will be time constrained

Enable Asynchronous Delivery: If set, the federate will enable asynchronous delivery

Synchronization

This section gives you the opportunity to handle a synchronization point right after the complete initialisation of the federate.

Synchronization Point Name: Name of the synchronisation point.

Enable Synchronization: The federate with wait for the given point to be announced and reached by all the federates in the federation.

Register synchronization point: the federate will register the given synchronization point and wait until all the federates are synchronized.

Resign Action: Action to pose upon resignation of the federation

EvokeCallback min: Minimum time to wait during a call to EvokeMultipleCallbacks

EvokeCallback max: Maximum time available for callback executions

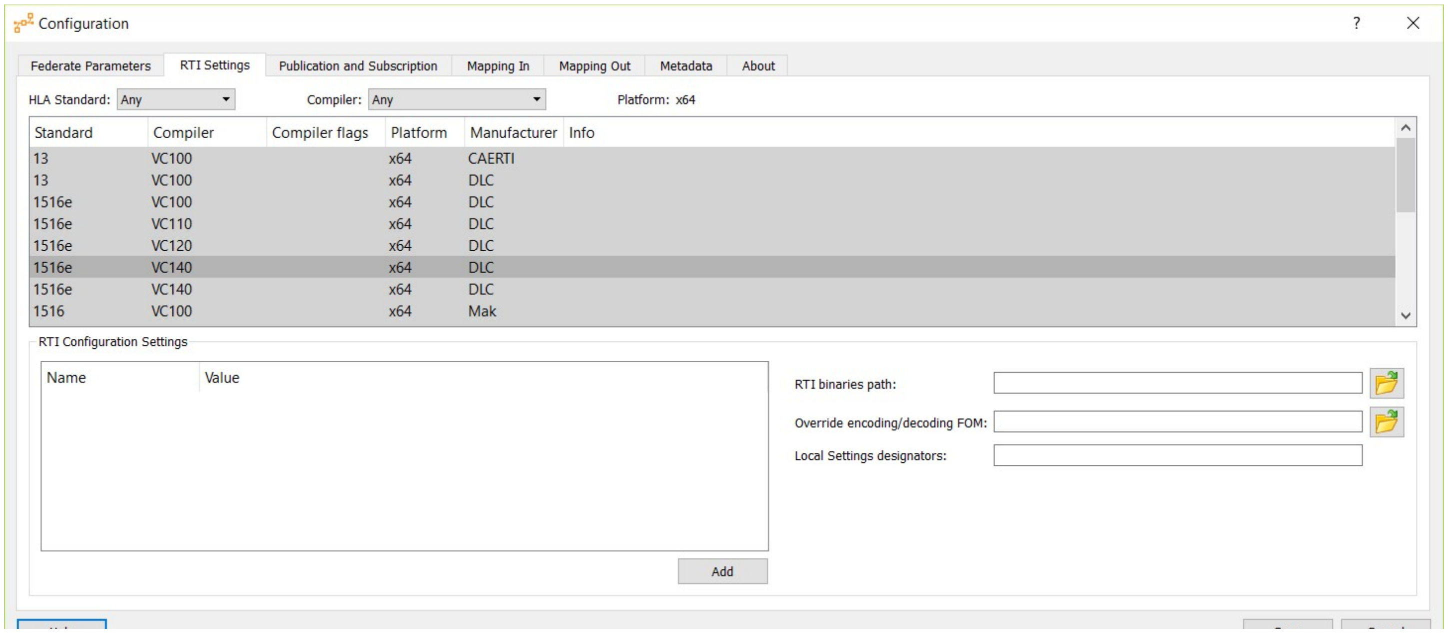
Display received callback: If checked, received callback will be displayed in the debug window and/or the log file.

Enable Provide Attribute Values: If checked, coreDS will automatically send a reply when receiving a ProvideAttributeValueUpdate callback.

Logical time implementation: Dropdown to select the time representation. Value can be double or integer. This option is only relevant with a HLA 1516e federation.

Callback delivery mechanism: Evoked or Immediate. Using immediate, callbacks are send directly to the simulation upon receipt. The simulator must support asynchronous calls. Using Evoked, callbacks are send to the simulator then the coreDS "step" function is called.

RTI Settings



HLA Standard: Any   Compiler: Any   Platform: x64

Standard	Compiler	Compiler flags	Platform	Manufacturer	Info
13	VC100		x64	CAERTI	
13	VC100		x64	DLC	
1516e	VC100		x64	DLC	
1516e	VC110		x64	DLC	
1516e	VC120		x64	DLC	
1516e	VC140		x64	DLC	
1516e	VC140		x64	DLC	
1516	VC100		x64	Mak	

RTI Configuration Settings

Name	Value

RTI binaries path:

Override encoding/decoding FOM:

Local Settings designators:

coreDS Supports a wide range of RTI, binding and HLA version. If a configuraiton is missing, please contact us. In most case, you can safety use a DLC, 1516e configuration. Make sure to select a "Compilier" that matchs the installed RTI bindings. Incorrect configuration will result in crashes.

RTI Configuration Settings
The Pitch RTI and MAK RTI support passing additionnal configuration in the form key=value (for example crcHost=192.168.1.23).

RTI Binaries path
To be used when you have multiple RTI installed on the same computer. You can force the path to the RTI. The application needs to be restarted.

Override encoding/decoding FOM: HLA 1.3 FOM file did not include encoding information. In order to use the automatic encoding/decoding feature, a matching FOM file, including encoding data, must be created and available to coreDS.

This can also be used when the master FOM used by the federation is incorrect or lacks some encoding/decoding information.

The file must be a valid HLA 1516 or 1516e file. All names from the override FOM must be the same as the FOM used to connect to the federation.

RTI-S

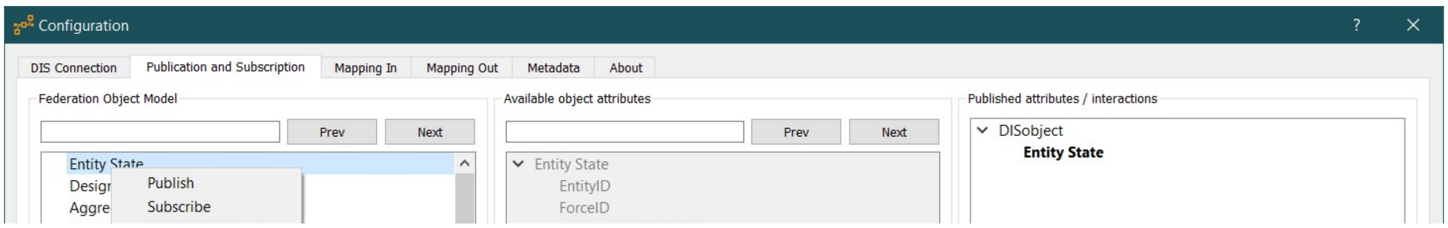
If you need to pass a rdr file to the RTI, enter the full path to the file in the "Local Settings Designators" field.

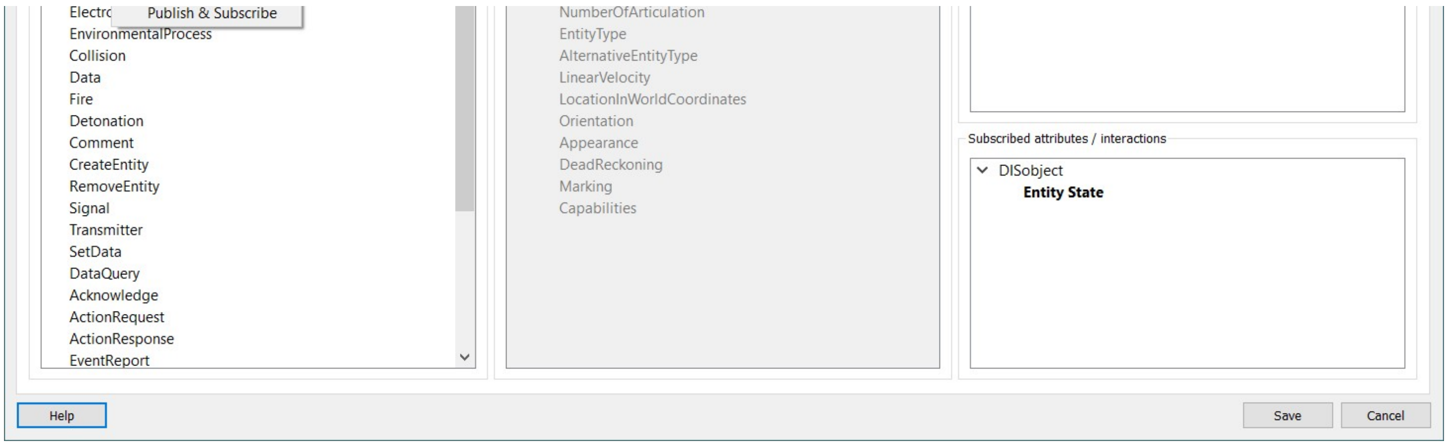
Publication and Subscription

HLA users are familiar with the concept of subscription and publication. This concept instructs to RTI which data we want to send and receive.

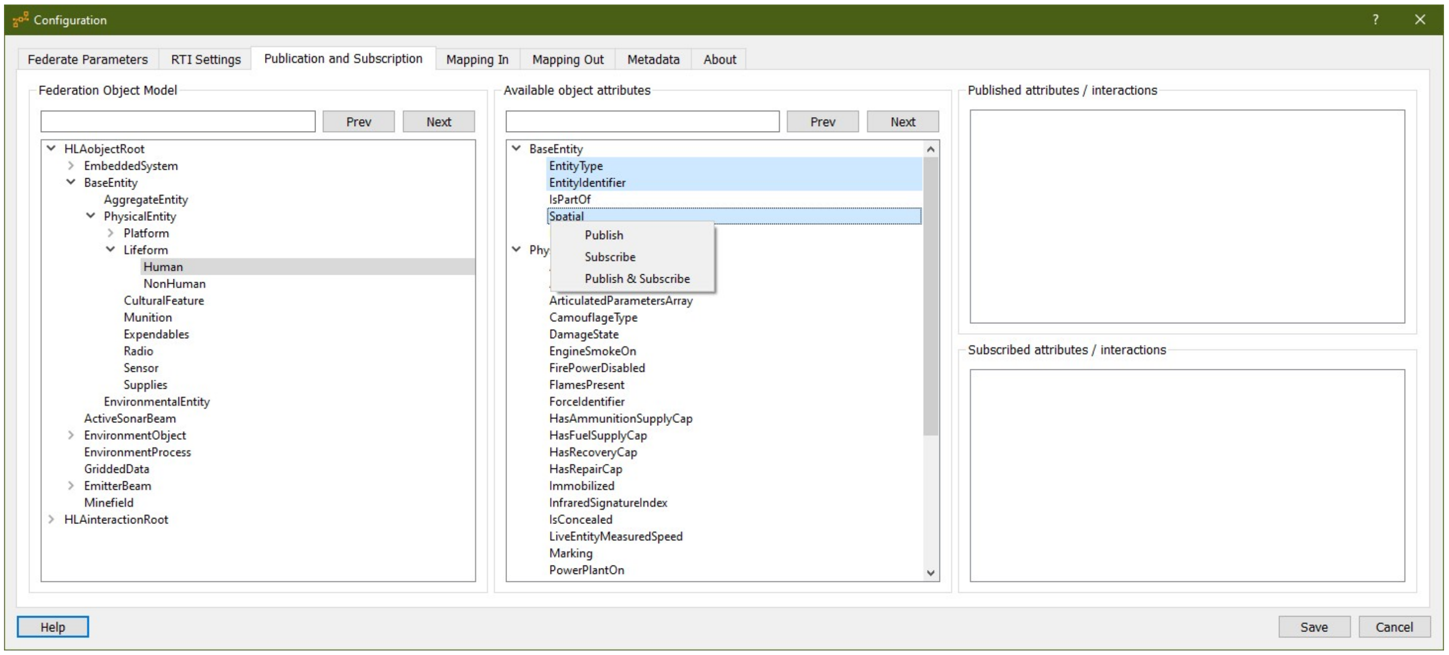
Although this concept does not exist when using DIS, coreDS allows to filter some PDU types.

When using DIS





When using HLA



Each column have a search field at the top allowing you to quickly find information.

Select an Object class in order to see the parameters associated with it in the list view to in the middle. From the list view, you can select the attributes you want to publish or subscribe and right click on them to save your selection in the Publish and/or Subscribe list displayed to the right.

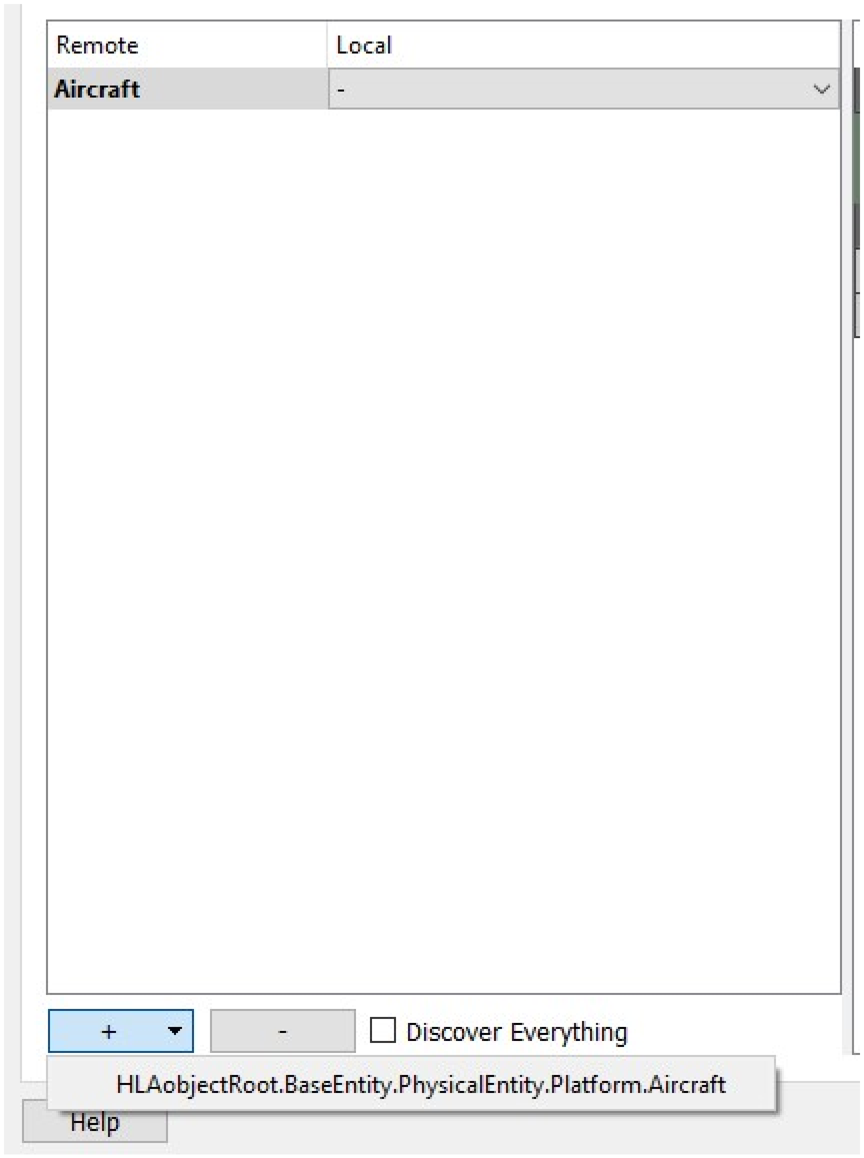
Working with interactions is the same as working with Objects except that you cannot publish or subscribe directly to a parameter. You must publish or subscribe a complete Interaction (HLA standard requirement).

You can delete published/subscribed element by right clicking on them and selecting "Delete" or "Delete all"

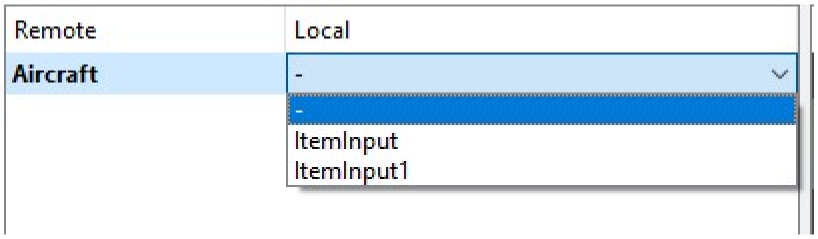
Mapping In

Mapping In defines data going from the simulation network to you simulator. This is the place where we link the distributed simulation names to locals names.

First step is to add a subscribed object or interaction (message) to our list. Clicking the "+" button will show you a list of the subscribed objects and messages. Selecting an item will add a row. You can now map your remote object to your local object

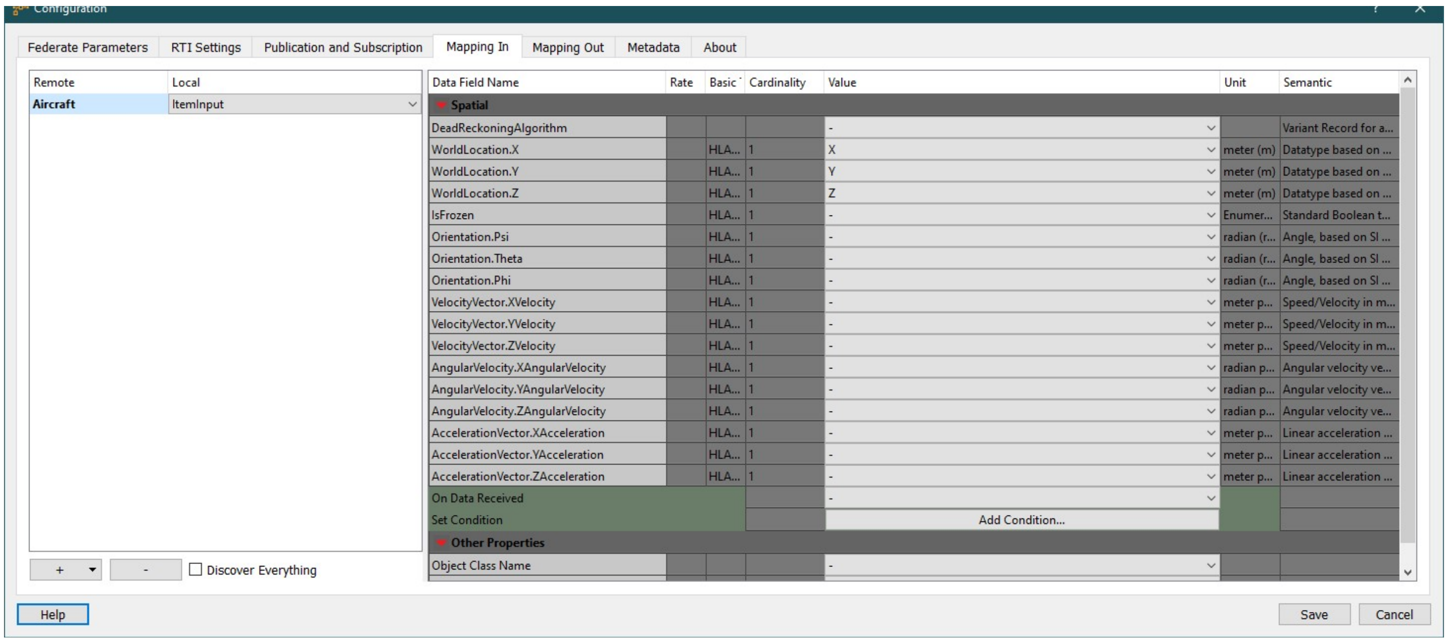


Remote	Local
Aircraft	-



Remote	Local
Aircraft	-

You can now map each and every properties from the remote object to your local object. There is no need to map everything! Map only the needed properties.



Remote	Local
Aircraft	ItemInput

Data Field Name	Rate	Basic	Cardinality	Value	Unit	Semantic
Spatial						
DeadReckoningAlgorithm				-		Variant Record for a...
WorldLocation.X		HLA...	1	X	meter (m)	Datatype based on ...
WorldLocation.Y		HLA...	1	Y	meter (m)	Datatype based on ...
WorldLocation.Z		HLA...	1	Z	meter (m)	Datatype based on ...
IsFrozen		HLA...	1	-	Enumer...	Standard Boolean t...
Orientation.Psi		HLA...	1	-	radian (r...	Angle, based on SI ...
Orientation.Theta		HLA...	1	-	radian (r...	Angle, based on SI ...
Orientation.Phi		HLA...	1	-	radian (r...	Angle, based on SI ...
VelocityVector.XVelocity		HLA...	1	-	meter p...	Speed/Velocity in m...
VelocityVector.YVelocity		HLA...	1	-	meter p...	Speed/Velocity in m...
VelocityVector.ZVelocity		HLA...	1	-	meter p...	Speed/Velocity in m...
AngularVelocity.XAngularVelocity		HLA...	1	-	radian p...	Angular velocity ve...
AngularVelocity.YAngularVelocity		HLA...	1	-	radian p...	Angular velocity ve...
AngularVelocity.ZAngularVelocity		HLA...	1	-	radian p...	Angular velocity ve...
AccelerationVector.XAcceleration		HLA...	1	-	meter p...	Linear acceleration ...
AccelerationVector.YAcceleration		HLA...	1	-	meter p...	Linear acceleration ...
AccelerationVector.ZAcceleration		HLA...	1	-	meter p...	Linear acceleration ...
On Data Received				-		
Set Condition				Add Condition...		
Other Properties						
Object Class Name				-		

We only mapped X, Y, Z for a local object of type "ItemInput".

So now, everything time we receive an HLA object of type Aircraft, you local simulator will receive the corresponding data (more on the C++ code required later).

HLA and DIS uses Geocentric coordinates. What if my simulator prefers LLA? What if my federation agreement/FOM File use a different coordinate system? We got you covered with the buildin LUA scripting engine.

You can hook a script to any values to handle data filtered, data conversion or even to send the information to a database. Let's say we want to convert our coordinates to Longitute, Latitude, Altitude (LLA), next to the "On Data Received", select "ConvertLocationFromHLA.lua". This couldn't be simplier.

Scripts are location in the config folder next to your application. The main LUA function must have the same name as the filename. The global state is shared accross all scripts meaning you can use global variables.

Below is the script that convert from Geocentric to local coordinates (convertLocationFromDIS.lua)

```
angleConversions = require("angleConversions")
require("ecef2lla")


lastReceivedLat = 0;
lastReceivedLong = 0;
lastReceivedAlt = 0;

function convertLocationFromDIS ()
        --convert geocentric to lat/log, values returned in degree
        assert(type(DSimLocal.X) ~= "number", "DSimLocal.X is not a number, received: " .. tost
        assert(type(DSimLocal.Y) ~= "number", "DSimLocal.Y is not a number, received: " .. tost
        assert(type(DSimLocal.Z) ~= "number", "DSimLocal.Z is not a number, received: " .. tost

        ok, DSimLocal.Y,DSimLocal.X,DSimLocal.Z =  pcall(ecef2lla, DSimLocal.X,DSimLocal.Y,DSim

        if ok then
                lastReceivedLat = DSimLocal.Y
                lastReceivedLong = DSimLocal.X
                lastReceivedAlt = DSimLocal.Z
        else
                lastReceivedLat = 0
                lastReceivedLong = 0
                lastReceivedAlt = 0
                DeleteValues = 1 --error while converting - ignore value
        end
end
```

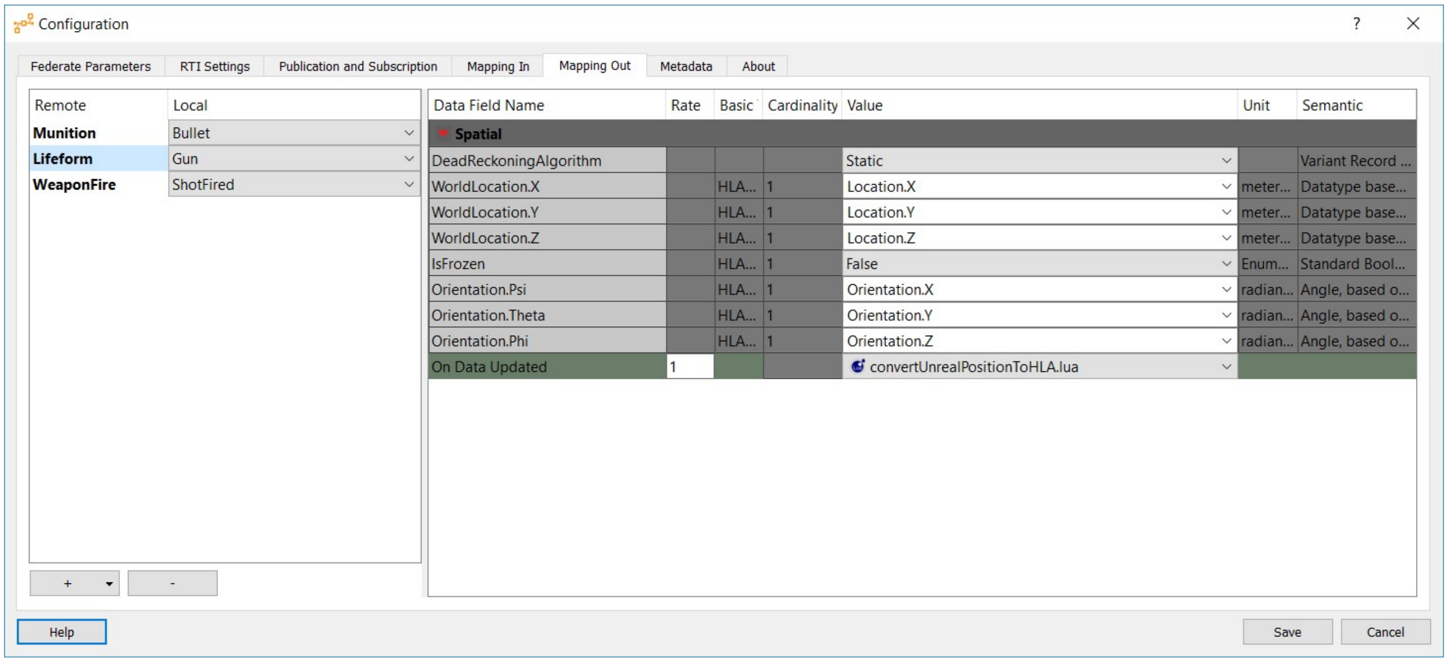
Data filtering and data selection

In some cases, you might have interest for a single object, or a few objects. When required, add a LUA script to the variable used to filter. In the script, simply set the variable DeleteValues=1.

For example, since DIS does not allow for explicity object type subscription, we can for a given object type by using a filtering script. The following script will only accept "Munition". Everything else will be discarded.

```
function FilterGun()
    -- Available variables
    -- DSimLocal.Category
    -- DSimLocal.CountryCode
    -- DSimLocal.Domain.Category
    -- DSimLocal.Domain.CountryCode
    -- DSimLocal.Domain.DomainDiscriminant
    -- DSimLocal.EntityKind
    -- DSimLocal.Extra
    -- DSimLocal.On Data Received
    -- DSimLocal.Specific
    -- DSimLocal.Subcategory

  if((DSimLocal.EntityKind == "3") ~= true) then
                DeleteValues = 1;
        end
end
```

Mapping Out



Remote	Local
Munition	Bullet
Lifeform	Gun
WeaponFire	ShotFired

Data Field Name	Rate	Basic	Cardinality	Value	Unit	Semantic
Spatial						
DeadReckoningAlgorithm				Static		Variant Record ...
WorldLocation.X		HLA...	1	Location.X	meter...	Datatype base...
WorldLocation.Y		HLA...	1	Location.Y	meter...	Datatype base...
WorldLocation.Z		HLA...	1	Location.Z	meter...	Datatype base...
IsFrozen		HLA...	1	False	Enum...	Standard Bool...
Orientation.Psi		HLA...	1	Orientation.X	radian...	Angle, based o...
Orientation.Theta		HLA...	1	Orientation.Y	radian...	Angle, based o...
Orientation.Phi		HLA...	1	Orientation.Z	radian...	Angle, based o...
On Data Updated	1			convertUnrealPositionToHLA.lua		

Integration coreDS

Calling the UIs

The showUI function will take care of opening the Configuraiton Selection window, the returned value is the configuration name, or empty if no configuration has been selected.

```
//the UI share globals states
showUI(lSelectedConfig, useless,  mEnableScripting, lSelectedLUAEditor, mDisableWord, mDebugEna
```

Registering the callbacks

In order to receive data, you must register a callback for each local datatype. You must use the LOCAL NAMES used in the configuration window. Various handlers are available. Just to name a few handles

- ObjectUpdate
- NewObjectFound
- ObjectRemoved
- MessageReceived

The registered function can either be a lambda function or a std::function.

```
mDSimManager.registerObjectUpdateHandler("ItemInput", updateAircraftPos);
mDSimManager.registerNewObjectFoundHandler("ItemInput", objectDiscovered);
mDSimManager.registerObjectRemovedHandler("ItemInput", objectRemoved);
mDSimManager.registerMessageReceivedHandler("CollisionIn", messageReceived);
```

Below are some sample callbacks

```
void messageReceived(const std::string &iMessageName, const std::map<std::string, std::string >
{
        std::map<std::string, std::string >::const_iterator iter = oDataStruct.begin();
        for (; iter != oDataStruct.end(); iter++)
        {
                printf("Message: %s\t%s\n", (*iter).first.c_str(), (*iter).second.c_str());
        }
}

void objectRemoved(const std::string &objectclassName)
{
        printf("Object deleted: %s\n", objectclassName.c_str());
}

void objectDiscovered(const std::string &objecttypeName, const std::string &objectUniqueIdentif
{
        printf("Object discovered: %s\t%s\t%s\n", objecttypeName.c_str(), objectUniqueIdentifie
}
```

Connecting to the simulation

To connecto the distributed simulation, simply call the Init function.

```
mDSimManager.init(mSelectedConfiguration, mDSIMParameters);
```

Do not forget to call the "step" function to receive the callbacks!

```
        mDSimManager.step();
```

Sending data

Sending data is done using the "UpdateObject" function

```
std::map< std::string, std::string > lOutputObjectVariablesValues;
lOutputObjectVariablesValues["X"] = "1";
lOutputObjectVariablesValues["Y"] = "2";
lOutputObjectVariablesValues["Z"] = "3";
mDSimManager.updateObject("UniqueName", "ItemOutput", lOutputObjectVariablesValues);
```

Then again, you must use the LOCAL names. "UniqueName" is a unique identifier that you must define to distinguish between your object instances. When calling the updateObject function for the first time, the corresponding HLA registerObjectInstance or DIS CreateObjectPDU will be used.