**Alarms Data Modelling**

In this Alarms data modeling example, we will:

- describe the data model structure

- the use of keyed data and the benefits

- the use of optional data and the benefits

**Structure**

A useful tool provided by RTI is System Designer; it is GUI based and can assist you in creating your data model as well as create QoS settings. It runs in a web browser environment, creates and continuously saves an XML representation of both data model and QoS while you are designing your system.

This Alarm example makes use of this tool and a representation of the *Alarms data model* is shown below.

*module Common* contains the many basic data type definitions used in this model and these definitions are then referenced in module Alarms. *module Alarms* is the section that defines the final data structures that become the Topics used by Connext DDS; in this example there are 4 structures supported and each structure can become a separate Topic in your application. All you have to do is instantiate it and use it.

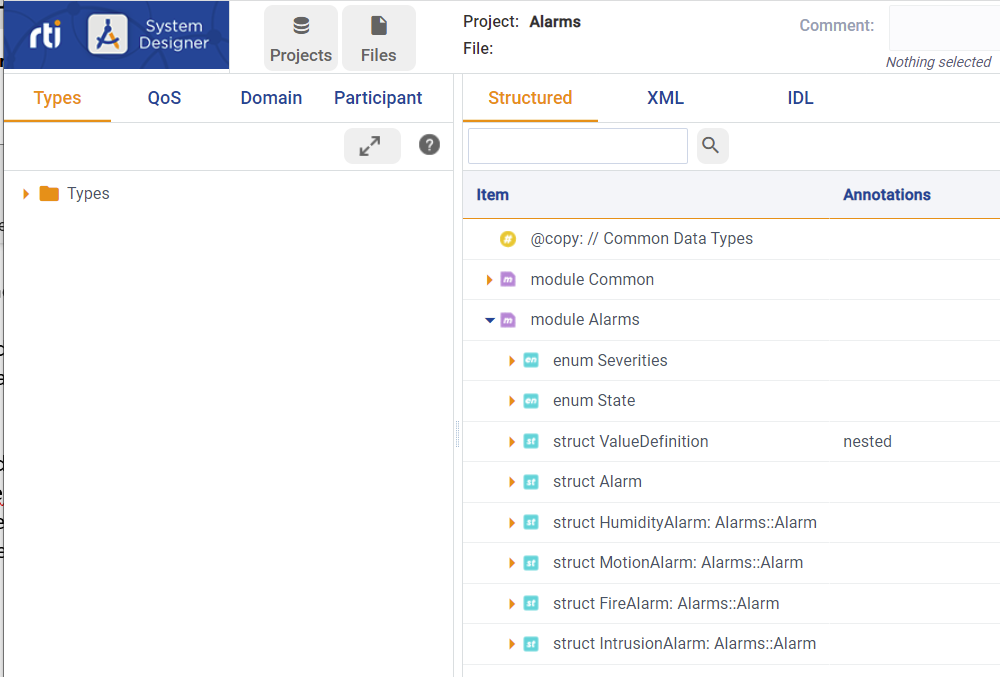
HumidityAlarm

MotionAlarm

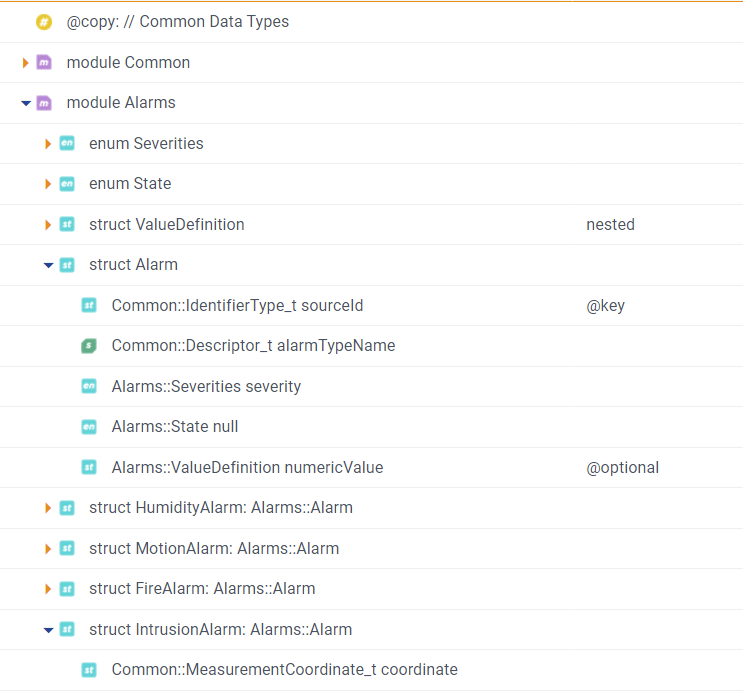
FireAlarm

IntrusionAlarm

In the code examples presented later , we will be using the IntrisionAlarm as the instantiated Topic.



Referring to the screen shot below outlining the Alarm data model, the *intrusionAlarm* is defined as a specific Topic and will hold *coordinate* data; it also inherits *Alarm* data and this holds sourceID (an ID associated with the alarm), TypeName (a description of the alarm), severity (self explanatory), null (the alarm state), and a numericValue (used to further describe the alarm).



**Keyed Data - Instances**

Of note are the tags associated with data:

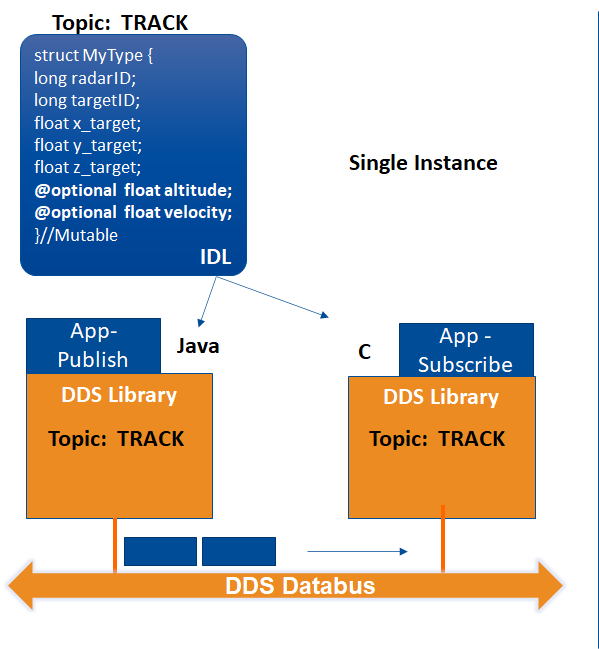
@key

@optional

sourceID is a data field tagged with a *key* value and this is a powerful tool. Any topic that contains the key identifier can be separated into unique *instances*.

*Instances* are flavors of Topics that are treated separately – they are managed in separate queues on the publisher and subscriber sides and can be configured uniquely. Typically, instances are used to convey commonly defined data with each instance being associated with a specific object.

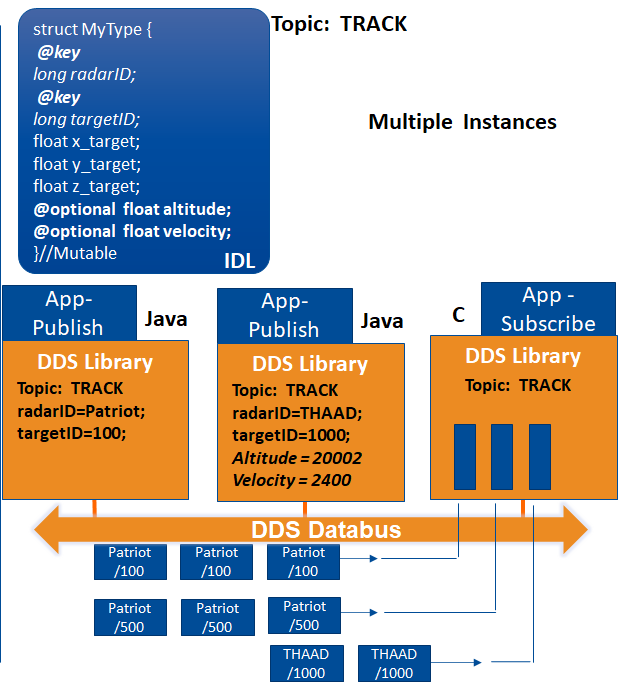
Let’s look at an example as shown below. In a system where there is one Topic and no instances (or keys) defined, that Topic is sent from publisher to subscriber and is managed in a single queue on both sides.



In the second case below , we have defined multiple instances. As a real world example, we are looking at a defensive missile system with two types of radar – low altitude Patriot missile defense and THAAD, a high altitude ICBM defensive system. Both radar types can identify a target and can send x, y, and z coordinates of the target to a central control system using a Topic called TRACK. Without Instances, sorting out the incoming Topics and associating target locations with specific targets can be a significant problem and would tax the application software sitting atop Connext DDS.

Instances can be used to segregate targets easily.

The Topic name TRACK is used but radarID and targetID are used as keyed fields; for each unique combination of keys, one instance is created. In this case the radar type and target ID are unique and will send an instance of TRACK that identifies an incoming missile with x, y and z coordinates. Each instance is managed in a separate queue and presented to the application as a grouping of data that is associated with one target. Quality of Service settings governing each Topic can also be set on a per instance basis.



In terms of coding, and returning to the original data model above, we can create two instances of IntrusionAlarm in the following way.

Using the RTI Code Generator and using the Alarms.xml file generated by System Designer, a code template will be created yielding source that still requires modification to be useful. Withon the code you will see the following snippet, creating the data structure called *data* (that is linked to the Topic name which is defined earlier in the code).

Alarms\_IntrusionAlarm \*data = Alarms\_IntrusionAlarmTypeSupport::create\_data();

if (data == NULL) {

return shutdown\_participant(

participant,

"Alarms\_IntrusionAlarmTypeSupport::create\_data error",

EXIT\_FAILURE);

}

This creates only one instance of the Topic. To create a second instance you must create a separate data structure (called *data2* in this example):

Alarms\_IntrusionAlarm\* data2 = Alarms\_IntrusionAlarmTypeSupport::create\_data();

if (data == NULL) {

return shutdown\_participant(

participant,

"Alarms\_IntrusionAlarmTypeSupport::create\_data error",

EXIT\_FAILURE);

}

You application can then use instances in the following way:

Instance 1 (*data* structure) is populated with data as shown…. Note that the keyed data, source.ID.id and sourceID.resourceID are defined and would be associated with an alarm device (serial number, model number etc. )

data->sourceId.id = 1;

data->sourceId.resourceId = 10;

data->alarmTypeName = "alarm name";

data->severity = Critical;

data->null = Open;

Instance 2 (*data2* structure) is also populated with data (from a different alarm sensor) and its keyed values are different than those of instance 1:

data2->sourceId.id = 2;

data2->sourceId.resourceId = 20;

data2->alarmTypeName = "alarm name";

data2->severity = Critical;

data2->null = Open;

Each of these instance will be managed separately and a unique QoS can be assigned to each instance if required; QoS assignment will be addressed later in this note.

**Optional Data**

Optional data is useful when creating a system where a Topic is defined but not all of the fields are used by a particular device. In cases where the (optional) data field is unpopulated, that data is not sent through the network.

Going back to the previous example of a radar defense system, you will note that the Patriot system provides only x, y and z coordinates. THAAD provides x, y, z, altitude, and velocity parameters. These additional parameters can be tagged as optional since they are not used in all systems that use the TRACK topic.

Another situation where a system is subject to a variable bandwidth network connection (WiFi, tactical radio) can make use of @optional. When bandwidth is restricted, the Topic can be populated with critical data and all other (optional) data can be omitted. With the restoration of bandwidth, the entire Topic can be sent through the network.

In our data modeling example, note that numericValue is a field tagged with @optional. There are a few considerations to be mindful of when using this capability:

1. The data creation (of data or data2) must be setup correctly; refer to the snippet below. Note the use of : DDS\_TypeAllocationParams\_t().set\_allocate\_optional\_members(DDS\_BOOLEAN\_TRUE)); in the parameter section of the create\_data call

Alarms\_IntrusionAlarm \*data = Alarms\_IntrusionAlarmTypeSupport::create\_data(DDS\_TypeAllocationParams\_t().set\_allocate\_optional\_members(DDS\_BOOLEAN\_TRUE));

if (data == NULL) {

return shutdown\_participant(

participant,

"Alarms\_IntrusionAlarmTypeSupport::create\_data error",

EXIT\_FAILURE);

Populating the data field requires a different approach. The optional data is reference by a pointer and if data is not present, the pointer is assigned a NULL value and the optional field is not sent as part of the Topic. When data that is written into the optional field is sent; the pointer is set to a value and then used to include the data field in the Topic when published. The example below illustrates this example. In the first instance (*data*), the optional value is set to NULL by writing NULL into the contents of the pointer and Connext DDS will recognize this and not send this data field

\*data->numericValue.number = NULL;

In the second instance (data2), data is not NULL and the data will be sent.

\*data2->numericValue.number = 111;

data->sourceId.id = 1;

data->sourceId.resourceId = 10;

data->alarmTypeName = "alarm name";

data->severity = Critical;

data->null = Open;

\*data->numericValue.number = NULL;

data->numericValue.Units = Celcius;

data2->sourceId.id = 2;

data2->sourceId.resourceId = 20;

data2->alarmTypeName = "alarm name";

data2->severity = Critical;

data2->null = Open;

\*data2->numericValue.number = 111;

data2->numericValue.Units = Farenheit;

As in the example of the Patriot missile defense system, the optional fields can be set to NULL and never used. For more information refer to Section 3.2 Optional Members in:

<https://community.rti.com/static/documentation/connext-dds/6.1.0/doc/manuals/connext_dds_professional/extensible_types_guide/index.htm#extensible_types/XTypes_Title.htm%3FTocPath%3D_____1>