Best Practices using RTI Connext DDS
Best Practices

• Introduction

• Types of best practices
  – Architectural
  – Application design and implementation
  – Network and QoS configuration

• Where to find more information
Introduction

• Best practices guide you in building a system that is
  – Future-proof
  – Scalable
  – High performance
  – Testable

• Without best practices, may be building suboptimal systems
  – Lose time figuring out best approach
  – System quality suffers
Categories

• Architectural
  – Creating a network data model that meets your requirements

• Application design and implementation
  – Making optimal design decisions within a single application

• Network configuration and QoS
  – Tuning for your system requirements
Create a Network Data Model

• Determine the structure of the data you will send over the network based on
  – The state you want to represent
  – The delivery characteristics of the data

• For example:
  – Representing state data for a fleet of trucks
  – Trucks have license plate numbers, position, oil levels, maintenance information
    – These change at different rates, or aperiodically

• This can map to Topics:
  – TruckGPS, TruckOilLevel, and TruckMaintenance
Use Typed Data

*Give your data a type instead of sending binary*

- Middleware handles endiness
- Automatically plug in applications
  - Data visualization and debugging (Excel, RTI DDS Spy)
  - Data recording, replay, and database integration (RTI Recorder, RTI Replay, Real-Time Connect)
  - Mediation and transformation (RTI Routing Service)
- RTI Connext DDS can filter data based on the content
Use Keyed Data

Represent data-objects within the middleware

- Keying data tells Connext DDS about unique data-objects – or *instances* – inside your Topics
  - **Example:** VIN is the unique ID of a truck, and maps to a key field in all data types representing trucks
- Connext notifies you about instance lifecycle
  - New instance, instance alive, instance not alive
- Connext can have behavior per-instance
  - Notifications of delayed instance updates
  - Cache allocated per-instance
  - Failure/failover mechanisms per-instance
Use Keyed Data (Example)

Represent data-objects within the middleware

```c
struct TruckGPS
{
  string<18> VIN; //@key
  LatLong position;
  long speed;
  long direction;
};
```

<table>
<thead>
<tr>
<th>Key</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIN</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>position</td>
<td>LatLong</td>
<td></td>
</tr>
<tr>
<td>speed</td>
<td>long</td>
<td></td>
</tr>
<tr>
<td>direction</td>
<td>long</td>
<td></td>
</tr>
</tbody>
</table>
Create Few DomainParticipants

Create the minimum number of DomainParticipants required by your data model

- DomainParticipant objects are responsible for:
  - Discovering Entities in the domain
  - Maintaining a discovery database
  - Creating threads
- It is not necessary to create one DomainParticipant per DataWriter or DataReader in an application
  - Doing this uses unnecessary local resources (threads, sockets, memory)
  - Because of discovery, this also uses unnecessary remote resources (network bandwidth, memory)
- An application typically creates one DomainParticipant per domain it wants to join
Create Entities Early

Do not create DataWriters and DataReaders just before sending or receiving

• When you create a DataWriter or DataReader, Connext:
  – Allocates the queues for that Entity
  – Sends discovery messages notifying other applications about that Entity

• Discovery process matches DataReaders and DataWriters over the network

• With default QoS, DataReaders receive data only after discovery is complete

• Better to create entities early so discovery has time to complete
Never Block a Callback

*Blocking callbacks causes non-deterministic pain*

- Listener callbacks are called from Connext threads that may do various tasks
  - Receive data for one or more DataReaders
  - Send and receive reliability metadata for one or more DataWriters
  - Handle periodic events such as deadlines and liveliness
- Blocking these callbacks will prevent the middleware from working correctly
  - May not receive expected data on one or more DataReaders
  - May fail to notify other applications of liveliness
- If you are creating a layer above Connext DDS, be careful about exposing callbacks to your users
Unless you need extreme performance, WaitSets are safer

- Listeners receive data in the middleware threads
  - Lowest latency receive of data
  - Blocking is a problem
- WaitSets receive data in a user thread
  - Can do long processing, block, call all Connext APIs
  - Slight increase in latency due to thread context switch
Unless you need to manipulate or combine data

- read or take from the queue depending on your use case
- The queue characteristics is controlled by QoS
- Reduces the amount of copies
- Getting state of objects
- Querying the state of certain objects
Decoupling protocol and data definition allows you to specify different reliability per-channel

• If your data is modeled as discrete updates to object state, you often do not need strict reliability
  — Example: “I need the current state of this turbine rotor, and the previous state.”
• Strict reliability requires higher overhead to store and deliver all data
• Choose reliability level depending on your requirements instead of defaulting to strict reliability:
  — **Strict reliability**: DataWriter is not allowed to overwrite data if existing DataReaders have not received it.
  — **Non-strict reliability**: DataWriter keeps the last N data samples to reliably deliver to DataReaders. Is allowed to overwrite samples even if DataReaders have not received it.
  — **Best effort**: DataWriters do not keep a queue of data for DataReaders and make no guarantees of delivery
Configure QoS Through Profiles

Unless you don’t want to use a configuration file and hardcode the configuration

• Use built-in profiles
• Give it useful names (e.g. pattern of data flow)
• Don’t use default QoS
• Inherit from base profiles
• Use the QoS library and profile name when creating entities (e.g. use create_datawriter_with_profile)
More Information

• Go to:
  – community.rti.com/best-practices
  – community.rti.com/forum