OpenDDS FOUNDATION™

WEBINAR

OpenDDS with Dynamic Data

Webinar Host: Jen Wiese
Presenters: Adam Mitz, Justin Wilson, Fred Hornsey, Son Dinh

December 14, 2022

OpenDDS Foundation™
12140 Woodcrest Exec. Dr., Ste. 300
Saint Louis, MO 63141 USA

Limits of Liability and Disclaimer of Warranty

While every precaution has been taken in preparing this material, including research, development and testing, OpenDDS Foundation assumes no responsibility for errors or omissions. No liability is assumed by OpenDDS Foundation for any damages resulting from the use of this information.

Opportunities to learn more about OpenDDS

https://opendds.org

Online Training Classes:

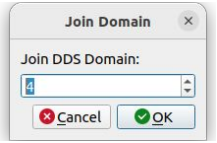
- Introduction to OpenDDS Programming (C++, Java)
- Building OpenDDS Applications with DDS Security (C++, Java)
- OpenDDS Essentials I (C++, Java)
  - QoS, Keys and Instances, & Built-In Topics
- OpenDDS Essentials II (C++, Java)
  - Configuration, Listeners, Conditions, & Content-Subscription

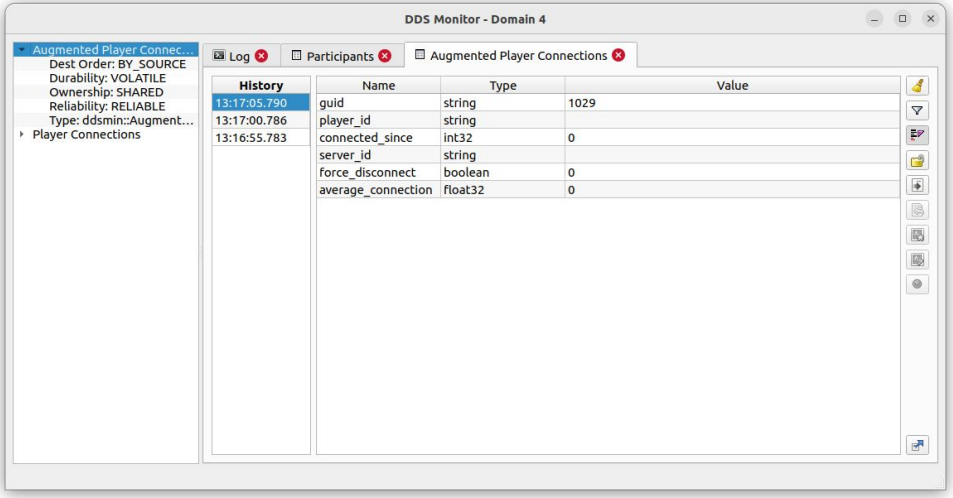
Related training classes from Object Computing, Inc.:

https://objectcomputing.com/services/training/catalog/middleware

Agenda

- Overview of XTypes and Dynamic Data
- Common Dynamic Data usage patterns
- Existing Dynamic Data in OpenDDS
- New Dynamic Data capabilities
- Demo of Dynamic Data applications
- Next steps for Dynamic Data support
- Q&A





Overview of XTypes

XTypes = eXtensible Types specification https://www.omg.org/spec/DDS-XTypes

XTypes gets around the assumption that all participants are using the same types, which facilitates interoperability and evolvability.

Introduces portable Type Objects for topic data types and an extensible encoding

Type Objects describe the schema of a type and the extensibility of its members

Extensibility allows the type to differ while remaining interoperable, e.g.,

- New fields can be added
- Default values provided for missing fields

Overview of XTypes

XTypes can be used “statically” or “dynamically”

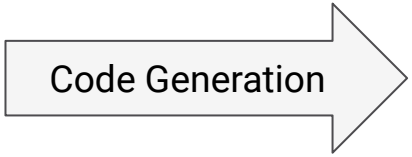
In the static scenario:

- Type Objects are derived from annotated IDL
- A writing application uses a statically-typed DataWriter
  - The DataWriter accepts objects of a type (C++/Java class) mapped directly from the topic's IDL structure
- A reading application uses a statically-typed DataReader
- The middleware makes the sample “right” for the reader

"Static" usage with Plain Language Binding

IDL:

```
@topic
struct LightControl {
  boolean state;
  float level;
  float temperature;
};
```



C++:

```
struct LightControl {
  bool state;
  float level;
  float temperature;
};
```

In this application, the structure above is used for a DDS topic. Other applications in the domain may have different but compatible type definitions (XTypes). Each application may use static (plain) or dynamic language binding.

Application:

```
LightControl ctrl{true, 1.f, 2.f};
data_writer->write(ctrl, handle);
```

Overview of Dynamic Data

In the dynamic scenario

- Type Objects are constructed programmatically or received from peer processes
- A writing application uses a dynamically typed DataWriter
- A reading application uses a dynamically typed DataReader
- The APIs use an abstract interface "DynamicData" that includes a type descriptor

“Dynamic” writers can interoperate with “static” readers and vice versa

The subset of XTypes that deals with dynamically constructed types and samples is called *Dynamic Data* and the *Dynamic Language Binding*.

Dynamic Language Binding and Dynamic Data

Application:

```
// The 'type' object below is the DynamicType for this topic.
// The DynamicType can be obtained from Built-In Endpoints (discovery) or
// loaded from a dynamic library used as a plugin.
DynamicData_var dd = DynamicDataFactory::get_instance()->create_data(type);
dd->set_boolean_value(MEMBER_ID_STATE, true);
dd->set_float32_value(MEMBER_ID_LEVEL, 1.f);
dd->set_float32_value(MEMBER_ID_TEMPERATURE, 2.f);
data_writer->write(dd, handle);
```

This application "knows" the structure used for a DDS topic, but there's no direct C++ mapping of that structure. Instead, a general container called DynamicData can store typed data for each member. Data samples are still strongly-typed, but type checks happen at runtime instead of at compile time. The DynamicType interface can be used for reflection/introspection – determining Member IDs and types at runtime.

Common Dynamic Data usage patterns

Recorders - applications that record samples for analysis or playback

Replayers - applications that replay recorded or dynamically generated samples

Monitoring and Diagnostic tools - applications that can show and/or analyze samples on arbitrary topics (without compile-time knowledge of those types)

Test Harnesses and Drivers - applications that can write samples on arbitrary topics

Generic Message Handling - logging, redaction, etc. (Dynamic Data interface to statically defined type)

DDS Security using Dynamic Data for instance-level security

Existing Dynamic Data in OpenDDS

OpenDDS 3.22 has support for

- Recorder: receive untyped data (raw byte stream)
- Replayer: send untyped data (raw byte stream)
- Static XTypes (not all annotations and types are supported)
- Partial support for reading Dynamic Data
  - Since there is no Dynamic Data Reader, data is received via Recorder
  - The byte stream received from Recorder is then processed as DynamicData
  - This is used by the *inspect* tool

New Dynamic Data capabilities

OpenDDS 3.23 (not released yet) will have

- Spec-defined interfaces for Dynamic Data / Dynamic Language Binding
  - Types can be loaded from generated code or received from peers in domain
  - Doesn't support creating types at runtime
  - Compiled type libraries can be used as plugins (dlopen/LoadLibrary)
- Dynamic Data Writer
- Dynamic Data Reader
- OpenDDS Monitor - replacement for existing monitor tool; based on XTypes
  https://github.com/OpenDDS/opendds-monitor

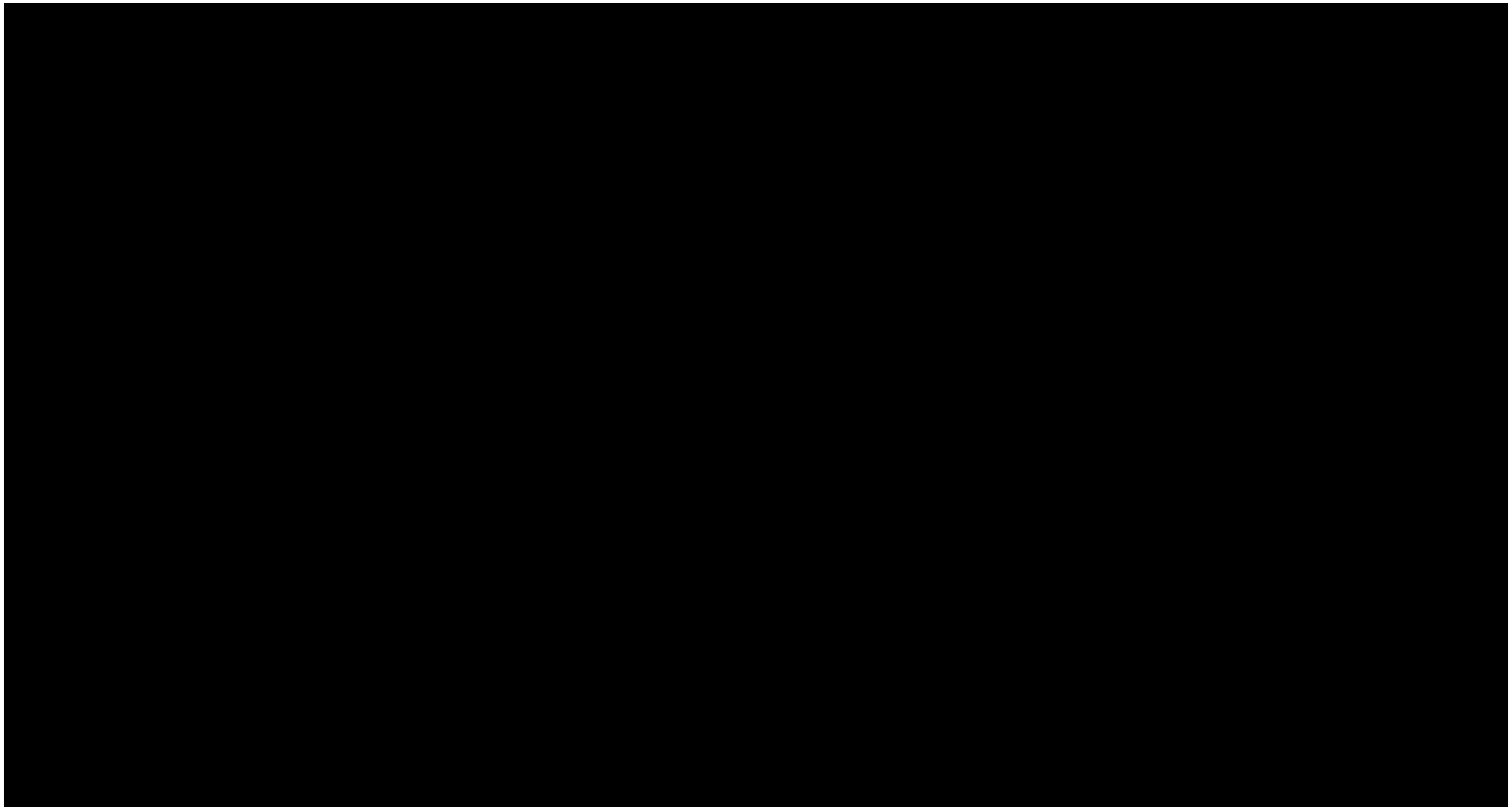
Demo of Dynamic Data applications

- Example IDL for vehicle status in a rental car fleet:

```
@topic
@mutable
struct VehicleHealth {
  @id(0) string vin;
  @id(1) uint16 oil_level;
  @id(2) double fuel;
};
```

- Demo 1: Dynamic publishing application → Static subscribing application
  - This writer loads a Type Support library as a plugin

Demo of Dynamic Data applications



Getting DynamicType from a TypeSupport Shared Library

```
ACE_DLL ts_plugin;
if (ts_plugin.open("VehicleHealth")) { /* Error */ }
// libVehicleHealth.so or VehicleHealth.dll

DDS::TypeSupport_var ts = Registered_Data_Types->lookup(0, "VehicleHealth1");
if (!ts) { /* Error */ }

DDS::DynamicType_var type = ts->get_type();
DDS::DynamicTypeSupport_var dts = new DDS::DynamicTypeSupport(type);
DDS::ReturnCode_t rc = dts->register_type(dp, "");
if (rc != DDS::RETCODE_OK) { /* Error */ }
```

Getting DynamicType from a Remote Topic

```
// Get key from publication or subscription builtin topic
DDS::DynamicType_var type;
const DDS::ReturnCode_t rc =
  TheServiceParticipant->get_dynamic_type(type, part, key);
if (rc != DDS::RETCODE_OK) { /* ERROR */ }

DDS::DynamicTypeSupport_var dts = new DDS::DynamicTypeSupport(type);
DDS::ReturnCode_t rc = dts->register_type(dp, "");
if (rc != DDS::RETCODE_OK) { /* Error */ }
```

Demo of Dynamic Data applications

- Demo 2: Static publishing application → Dynamic subscribing application
  - OpenDDS Monitor as the reader
  - OpenDDS's inspect tool as the reader

Next steps for Dynamic Data support

Integration with Content-Filtered Topics and other Content Subscription features

OpenDDS Monitor (Qt GUI):

- Rendering more complex types in the UI
- Writing data samples using Dynamic Data

Constructing types at runtime

XCDRv1 Data Representation

Thank you!

Any Questions?

OpenDDS FOUNDATION™

Articles and Talks

On-demand Webinars:

- Designing a Secure Cloud-Enabled Peer-to-Peer IoT Application
  - objectcomputing.com/resources/publications/mnb/2019/06/20/interoperable-internet-enabled-dds-applications
- Using OpenDDS's RtpsRelay to Connect IoT/IIoT Applications
  - objectcomputing.com/products/opendds/resources/rtpsrelay-iot

Articles:

- Interoperable Internet-Enabled DDS Applications
  - objectcomputing.com/resources/publications/mnb/2019/06/20/interoperable-internet-enabled-dds-applications
- Bringing Multicast to the Cloud for Interoperable DDS Applications
  - objectcomputing.com/resources/publications/mnb/2019/03/01/bringing-multicast-cloud-interoperable-dds-applications

Further Study: On-Demand Webinars

Data Distribution with an Open and Secure DDS (DDS Security)

objectcomputing.com/resources/events/webinars/opendds-security

Designing a Distributed Application using DDS QoS

brighttalk.com/webcast/12231/281491

XTypes in OpenDDS 3.16

objectcomputing.com/products/opendds/resources/introducing-xtypes

Getting Started as an OpenDDS Code Contributor

objectcomputing.com/products/opendds/resources/opendds-code-contribution-tutorial

OpenDDS Resources

OpenDDS project

opendds.org

Source repo

github.com/objectcomputing/OpenDDS

OpenDDS support, training, consulting, development

objectcomputing.com/products/opendds

OpenDDS 3.22 Release Notes

github.com/objectcomputing/OpenDDS/releases/tag/DDS-3.22

OpenDDS Foundation

OpenDDS Foundation is a not-for-profit organization that exists to support and collectively lead the open source OpenDDS® project. The Foundation is supported by a Technology Advisory Board that ensures the technology continues to reflect and serve its diverse and growing user community.

OpenDDS Foundation works to ensure technical innovation and advancement of the OpenDDS project, evangelize and promote the project as a leading technology in the data distribution space, and build and support an ecosystem of complementary documentation, functionality, and services.

As a not-for-profit organization, OpenDDS Foundation relies on the financial support of contributing members to support and grow the project. Businesses and community members are encouraged to actively participate in the project's success by becoming contributing members through one of our sponsorship programs.

LET'S CONNECT!


info@opendds.org


opendds.org/foundation


@OpenDDS


linkedin.com/showcase/opendds


github.com/objectcomputing/OpenDDS