OpenDDS Modeling SDK 使用指南

王进成
邮箱：jincheng8712@163.com
2019/4/25

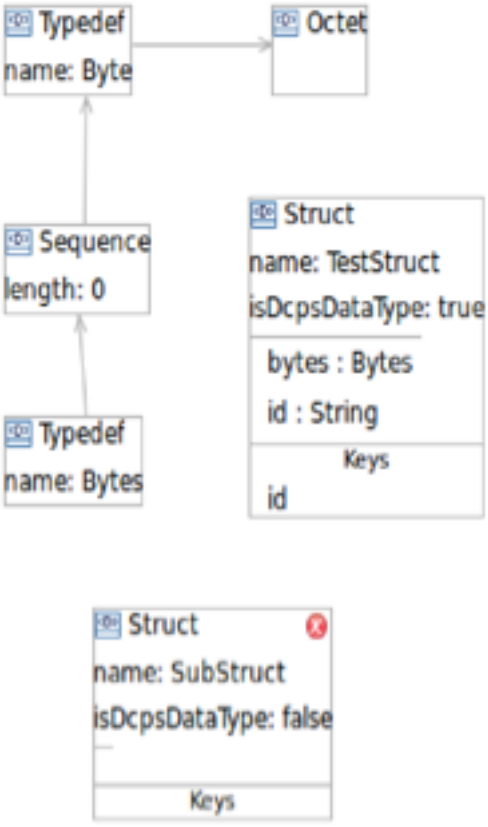
OpenDDS Modeling SDK 是一种建模工具，应用程序开发人员可以使用该工具将所需的中间件组件和数据结构定义为 UML 模型，然后生成使用 OpenDDS 实现模型的代码。 然后，可以编译生成的代码并将其与应用程序链接，以向应用程序提供无缝的中间件支持。

目录

1. 概述

1.1. 模型捕获

使用 Eclipse 插件中包含的图形建模捕获编辑器捕获定义 DCPS 元素和策略以及数据定义的 UML 模型。 UML 模型的元素遵循 DDS 规范中定义的 DDS UML 平台独立模型（PIM）的结构（OMG：formal / 2015-04-10）。
在插件中打开一个新的 OpenDDS 模型，从顶层主图开始， 此图包括要包含在模型中的任何包结构以及模型的本地 QoS 策略定义，数据定义和 DCPS 元素。可以包括零个或多个策略或数据定义元素。任何给定模型中都可以包含零个或一个 DCPS 元素定义。



支持为 QoS 策略，数据定义或 DCPS 元素创建单独的模型。 对其他模型的引用允许外部定义的模型包含在模型中。 这允许在不同的 DCPS 模型之间共享数据定义和 QoS 策略，以及在一组新的数据定义中包括外部定义的数据。

1.2. 代码生成

捕获模型后，可以从中生成源代码。 然后可以将此源代码编译到链接库中，从而将模型中定义的中间件元素提供给应用程序。代码生成使用单独的基于表单的编辑器完成。

代码生成的规范对于各个生成形式是唯一的，并且与正在执行生成的模型分开。代码生成一次在单个模型上执行，并且包括定制生成的代码以及指定在构建时用于定位资源的搜索路径的能力。

可以生成模型变体（相同模型的不同自定义），然后可以在相同的应用程序或不同的应用程序中创建。 还可以指定在构建时搜索标题文件和链接库的位置。

1.2.1. 编程

为了使用由模型定义的中间件，应用程序需要链接生成的代码。 这是通过头文

件和链接库完成的。 使用 MPC 可移植构建工具构建应用程序的支持包含在模型的生成文件中。

2. 开发应用程序

为了使用 OpenDDS Modeling SDK 构建应用程序，必须了解一些关键概念。这些概念涉及：

1）支持库

2）生成的模型代码

3）应用程序代码

2.1. 建模支持库

OpenDDS Modeling SDK 包含一个支持库，可在$ DDS_ROOT / tools / modeling / codegen / model 中找到。此支持库与 Modeling SDK 生成的代码结合使用，可以大大减少构建 OpenDDS 应用程序所需的代码量。

支持库是一个 C ++库，由 OpenDDS Modeling SDK 应用程序使用。大多数开发人员需要的支持库中的两个类是 Application 和 Service 类。

2.1.1. Application 类

OpenDDS :: Model :: Application 类负责 OpenDDS 库的初始化和完成。使用 OpenDDS 的任何应用程序都需要实例化 Application 类的单个实例，并且在使用 OpenDDS 进行通信时不会销毁 Application 对象。

Application 类初始化用于创建 OpenDDS 参与者的工厂。此工厂需要用户提供的命令行参数。为了提供它们，必须为 Application 对象提供相同的命令行参数。

2.1.2. Service 类

OpenDDS :: Model :: Service 类负责创建 OpenDDS Modeling SDK 模型中描述的 OpenDDS 实体。 由于模型可以是通用的，描述比单个应用程序使用的更广泛的

域，因此 Service 类使用延迟实例化来创建 OpenDDS 实体。
为了正确实例化这些实体，它必须知道：
•实体之间的关系
•实体使用的传输配置

2.2. 生成的代码

OpenDDS Modeling SDK 生成模型特定代码，供 OpenDDS Modeling SDK 应用程序使用。 从.codegen 文件（指向.opendds 模型文件）生成，表 1 中描述了这些文件。 生成代码的过程记录在 Eclipse 帮助中。

表格 1 生成的文件

File Name	Description
<ModelName>.idl	Data types from the model's DataLib
<ModelName>_T.h	C++ class from the model's DcpsLib
<ModelName>_T.cpp	C++ implementation of the model's DcpsLib
<ModelName>.mpc	MPC project fle for the generated C++ library
<ModelName>.mpb	MPC base project for use by the application
<ModelName>_paths.mpb	MPC base project with paths, see section 11.3.3.7
<ModelName>Traits.h	Transport confguration from the .codegen fle
<ModelName>Traits.cpp	Transport confguration from the .codegen fle

2.2.1. DCPS 模型类

DCPS 库建模了 DDS 实体之间的关系，包括 Topics，DomainParticipants，Publishers，Subscribers，DataWriters 和 DataReaders 及其相应的域。
对于你的模型中的每个 DCPS 库，OpenDDS Modeling SDK 会生成一个以 DCPS

库命名的类。此 DCPS 模型类以 DCPS 库命名，可在代码生成目标目录中的 <ModelName> _T.h 文件中找到。
模型类包含一个名为 Elements 的内部类，为库中建模的每个 DCPS 实体和库主题引用的每种类型定义枚举标识。 此 Elements 类包含以下各项的枚举定义：

- DomainParticipants
- Types
- Topics
- Content Filtered Topics
- Multi Topics
- Publishers
- Subscribers
- Data Writers
- Data Readers

此外，DCPS 模型类捕获这些实体之间的关系。实例化 DCPS 实体时，Service 类将使用这些关系。

2.2.2. Traits 类

DCPS 模型中的实体按名称引用其传输配置。Codegen 文件编辑器的 “模型自定义” 选项卡用于定义每个名称的传输配置。
对于特定代码生成文件，可以定义多组配置。这些配置集被分组为实例，每个实例由名称标识。可以定义多个实例，表示使用该应用程序的模型的不同部署方案。
对于每个实例，都会生成一个 Traits 类。traits 类提供在 Codegen 编辑器中建模的传输配置，以获取特定的传输配置名称。

2.2.3. Service 类型定义

服务是一个需要两个参数的模板：（1）实体模型，在 DCPS 模型中的 Elements 类，（2）传输配置，在 Traits 类中。 OpenDDS Modeling SDK 为 DCPS 库和传输配置模型实例的每个组合生成一个 typedef。 typedef 名为<InstanceName> <DCPSLibraryName> Type。

2.2.4. 数据库生成的代码

从数据库中生成 IDL，IDL 由 IDL 编译器处理。 IDL 编译器生成类型支持代码，用于序列化和反序列化数据类型。

2.2.5. QoS 策略库生成的代码

QoS 策略库中没有生成特定的编译单元。相反，DCPS 库存储其建模的实体的 QoS 策略。稍后将由 Service 类查询此 QoS 策略，该类在实体创建时设置 QoS 策略。

2.3. 应用代码要求

2.3.1. 要求的头文件

除了 Tcp.h 头文件（用于静态链接）之外，应用程序还需要包含 Traits 头文件。这些将包含构建发布应用程序所需的所有内容。以下是示例发布应用程序 MinimalPublisher.cpp 的#include 部分。

```
#ifdef ACE_AS_STATIC_LIBS
#include <dds/DCPS/transport/tcp/Tcp.h>
#endif
#include "model/MinimalTraits.h"
```

2.3.2. 异常处理

推荐 Modeling SDK 应用程序捕获 CORBA :: Exception 对象和 std :: exception 对象。

```
int ACE_TMAIN(int argc, ACE_TCHAR* argv[]) {

  try {
    // Create and use OpenDDS Modeling SDK (see sections below)
  } catch (const CORBA::Exception& e) {
```

```
    // Handle exception and return non-zero
  } catch (const OpenDDS::DCPS::Transport::Exception& te) {
    // Handle exception and return non-zero
  } catch (const std::exception& ex) {
    // Handle exception and return non-zero
}

return 0;

}
```

2.3.3. 实例化

如上所述，OpenDDS Modeling SDK 应用程序必须在其生命周期内创建一个 OpenDDS :: Model :: Application 对象。反过来，此 Application 对象将传递给由 traits 头中的 typedef 声明之一指定的 Service 对象的构造函数。
然后，该服务用于创建 OpenDDS 实体。 要使用 Elements 类中指定的枚举标识符之一指定要创建的特定实体。 服务为实体创建提供此接口：

```
DDS::DomainParticipant_var participant(Elements::Participants::Values part);
DDS::TopicDescription_var topic(Elements::Participants::Values part,
                                Elements::Topics::Values topic);
DDS::Publisher_var publisher(Elements::Publishers::Values publisher);
DDS::Subscriber_var subscriber(Elements::Subscribers::Values subscriber);
DDS::DataWriter_var writer(Elements::DataWriters::Values writer);
DDS::DataReader_var reader(Elements::DataReaders::Values reader);
```

请务必注意，该服务还会在必要时创建任何所需的中间实体，例如 DomainParticipants，Publishers，Subscribers 和 Topics。

2.3.4. 发布者代码

使用上面显示的 writer（）方法，MinimalPublisher.cpp 继续：

```
int ACE_TMAIN(int argc, ACE_TCHAR* argv[])
{
  try {
    OpenDDS::Model::Application application(argc, argv);
    MinimalLib::DefaultMinimalType model(application, argc, argv);

    using OpenDDS::Model::MinimalLib::Elements;
    DDS::DataWriter_var writer = model.writer(Elements::DataWriters::writer);
```

剩下的就是将 DataWriter 缩小到类型特定的数据写入器，并发送样本。

```
  data1::MessageDataWriter_var msg_writer =
      data1::MessageDataWriter::_narrow(writer);
  data1::Message message;
  // Populate message and send
  message.text = "Worst. Movie. Ever.";
  DDS::ReturnCode_t error = msg_writer->write(message, DDS::HANDLE_NIL);
  if (error != DDS::RETCODE_OK) {
    // Handle error
  }
```

在整个发布应用程序中，MinimalPublisher.cpp 就像这样：

```
#ifdef ACE_AS_STATIC_LIBS
#include <dds/DCPS/transport/tcp/Tcp.h>
#endif

#include "model/MinimalTraits.h"

int ACE_TMAIN(int argc, ACE_TCHAR* argv[])
{
  try {
    OpenDDS::Model::Application application(argc, argv);
    MinimalLib::DefaultMinimalType model(application, argc, argv);

    using OpenDDS::Model::MinimalLib::Elements;
    DDS::DataWriter_var writer = model.writer(Elements::DataWriters::writer);

    data1::MessageDataWriter_var msg_writer =
        data1::MessageDataWriter::_narrow(writer);
    data1::Message message;
    // Populate message and send
    message.text = "Worst. Movie. Ever.";
    DDS::ReturnCode_t error = msg_writer->write(message, DDS::HANDLE_NIL);
    if (error != DDS::RETCODE_OK) {
      // Handle error
    }
  } catch (const CORBA::Exception& e) {
    // Handle exception and return non-zero
  } catch (const std::exception& ex) {



    // Handle exception and return non-zero
  }
  return 0;
}
```

请注意，此最小示例忽略了日志记录和同步，这些问题不是特定于 OpenDDS Modeling SDK 的问题。

2.3.5. 订阅者代码

订阅者代码与发布者非常相似。为简单起见，OpenDDS Modeling SDK 订阅者可能希望利用 Reader Listeners 的基类，称为 OpenDDS :: Modeling :: NullReaderListener。 NullReaderListener 实现整个 DataReaderListener 接口并记录每个回调。

订阅者可以通过从 NullReaderListener 派生类并覆盖感兴趣的接口（例如 on_data_available）来创建侦听器。

```
#ifdef ACE_AS_STATIC_LIBS
#include <dds/DCPS/transport/tcp/Tcp.h>
#endif

#include "model/MinimalTraits.h"
#include <model/NullReaderListener.h>

class ReaderListener : public OpenDDS::Model::NullReaderListener {
public:
  virtual void on_data_available(DDS::DataReader_ptr reader)
                       ACE_THROW_SPEC((CORBA::SystemException)) {
    data1::MessageDataReader_var reader_i =
      data1::MessageDataReader::_narrow(reader);

    if (!reader_i) {
      // Handle error
      ACE_OS::exit(-1);
    }

    data1::Message msg;
    DDS::SampleInfo info;

    // Read until no more messages
    while (true) {
      DDS::ReturnCode_t error = reader_i->take_next_sample(msg, info);
      if (error == DDS::RETCODE_OK) {
        if (info.valid_data) {
          std::cout << "Message: " << msg.text.in() << std::endl;
        }
      } else {
        if (error != DDS::RETCODE_NO_DATA) {
          // Handle error
        }
        break;
      }
    }
  }
};
```

在 main 函数中，从服务对象创建数据读取器：

```
DDS::DataReader_var reader = model.reader(Elements::DataReaders::reader);
```

当然，DataReaderListener 必须与数据读取器相关联才能获得其回调。

```
DDS::DataReaderListener_var listener(new ReaderListener);
reader->set_listener(listener, OpenDDS::DCPS::DEFAULT_STATUS_MASK);
```

剩余的订阅者代码与任何 OpenDDS Modeling SDK 应用程序具有相同的要求，因为它必须通过 OpenDDS :: Modeling :: Application 对象初始化 OpenDDS 库，并使用适当的 DCPS 模型 Elements 类和 traits 类创建 Service 对象。

下面是一个示例订阅应用程序 MinimalSubscriber.cpp。

```
#ifdef ACE_AS_STATIC_LIBS
#include <dds/DCPS/transport/tcp/Tcp.h>
#endif

#include "model/MinimalTraits.h"
#include <model/NullReaderListener.h>

class ReaderListener : public OpenDDS::Model::NullReaderListener {
public:
  virtual void on_data_available(DDS::DataReader_ptr reader)
                        ACE_THROW_SPEC((CORBA::SystemException)) {
    data1::MessageDataReader_var reader_i =
      data1::MessageDataReader::_narrow(reader);

    if (!reader_i) {
      // Handle error
      ACE_OS::exit(-1);
    }

    data1::Message msg;
    DDS::SampleInfo info;

    // Read until no more messages
    while (true) {
      DDS::ReturnCode_t error = reader_i->take_next_sample(msg, info);
      if (error == DDS::RETCODE_OK) {
        if (info.valid_data) {
          std::cout << "Message: " << msg.text.in() << std::endl;
        }
      } else {
        if (error != DDS::RETCODE_NO_DATA) {
          // Handle error
        }
        break;
      }
    }
  }
};
```

```
int ACE_TMAIN(int argc, ACE_TCHAR* argv[])
{
  try {
    OpenDDS::Model::Application application(argc, argv);
    MinimalLib::DefaultMinimalType model(application, argc, argv);

    using OpenDDS::Model::MinimalLib::Elements;

    DDS::DataReader_var reader = model.reader(Elements::DataReaders::reader);

    DDS::DataReaderListener_var listener(new ReaderListener);
    reader->set_listener(listener, OpenDDS::DCPS::DEFAULT_STATUS_MASK);

    // Call on_data_available in case there are samples which are waiting
    listener->on_data_available(reader);

    // At this point the application can wait for an exteral "stop" indication
    // such as blocking until the user terminates the program with Ctrl-C.

  } catch (const CORBA::Exception& e) {
    e._tao_print_exception("Exception caught in main():");
    return -1;
  } catch (const std::exception& ex) {
    // Handle error
    return -1;
  }
  return 0;
}
```

2.3.6. MPC 项目

为了使用 OpenDDS Modeling SDK 支持库，OpenDDS Modeling SDK MPC 项目应该从 dds_model 项目库继承。 这是非 Modeling SDK 项目继承的 dcpsexe 基础的补充。

```
project(*Publisher) : dcpsexe, dds_model {
  // project configuration
}
```

生成的模型库将在目标目录中生成 MPC 项目文件和基础项目文件，并负责构建模型共享库。OpenDDS 建模应用程序必须（1）在其构建中包含生成的模型库，以及（2）确保在生成的模型库之后构建项目。

```
project(*Publisher) : dcpsexe, dds_model {
  // project configuration
  libs  += Minimal
  after += Minimal
}
```

这两个都可以通过继承模型库的项目库来完成，该项目库以模型库命名。

```
project(*Publisher) : dcpsexe, dds_model, Minimal {
  // project configuration
```

}

请注意，现在必须在项目文件创建期间由 MPC 找到 Minimal.mpb 文件。 这可以通过-include 命令行选项来完成。

使用任何一种形式，MPC 文件必须告诉构建系统在哪里查找生成的模型库。

```
project(*Publisher) : dcpsexe, dds_model, Minimal {
  // project configuration
  libpaths += model
}
```

此设置基于 Codegen 文件编辑器中提供给目标文件夹设置的内容。

最后，像任何其他 MPC 项目一样，必须包括其源文件：

```
  Source_Files {
    MinimalPublisher.cpp
  }
```

最终的 MPC 项目对于发布者来说是这样的：

```
project(*Publisher) : dcpsexe, dds_model, Minimal {
  exename   = publisher
  libpaths += model

  Source_Files {
    MinimalPublisher.cpp
  }
}
```

订阅者类似：

```
project(*Subscriber) : dcpsexe, dds_model, Minimal {
  exename   = subscriber
  libpaths += model

  Source_Files {
    MinimalSubscriber.cpp
  }
}
```

2.3.7. 模型之间的依赖关系

最后一个考虑因素 - 生成的模型库本身可以依赖于其他生成的模型库。 例如，可能存在生成到不同目录的外部数据类型库。

随着时间的推移，模型改变其依赖性，这种可能性可能导致项目文件的大量维护。为了帮助克服这一负担，生成的模型库在名为<ModelName> _paths.mpb 的单独 MPB 文件中记录其所有外部引用的模型库的路径。从此路径继承基础项目将

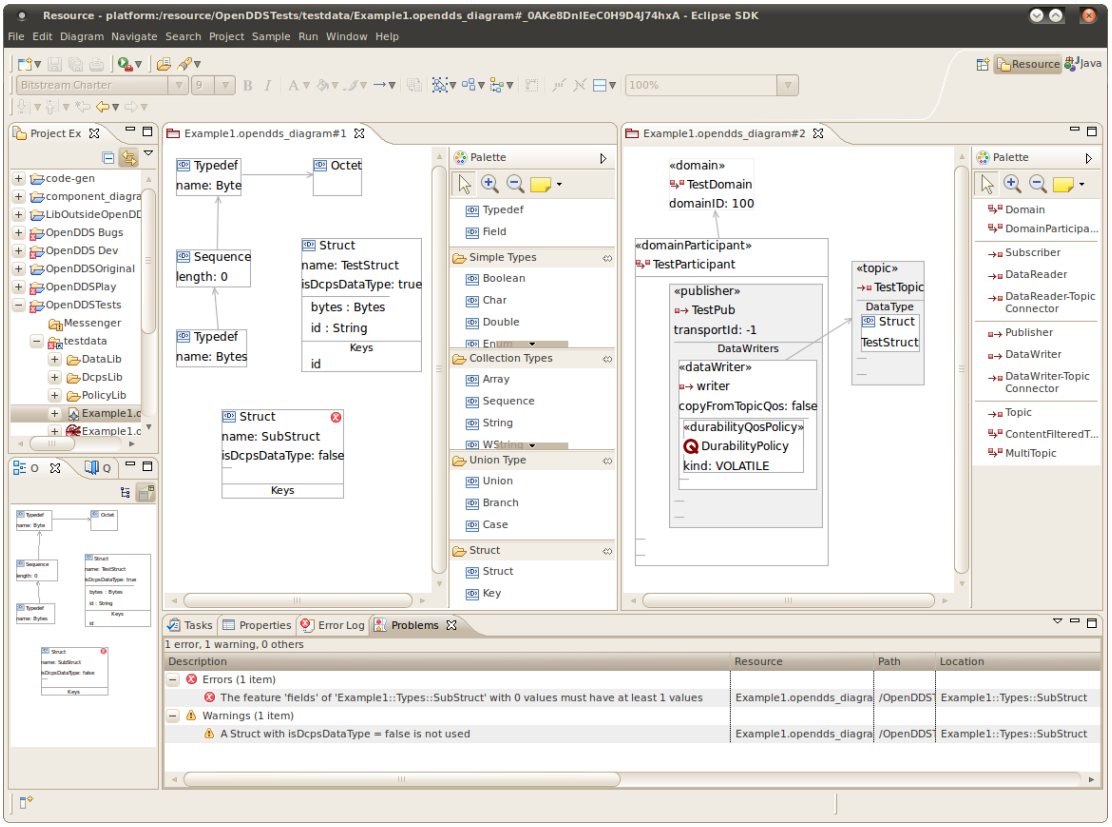
继承所需的设置以包括依赖模型。

我们的完整 MPC 文件看起来像这样：

```
project(*Publisher) : dcpsexe, dds_model, Minimal, Minimal_paths {
  exename   = publisher
  libpaths += model

  Source_Files {
    MinimalPublisher.cpp
  }
}

project(*Subscriber) : dcpsexe, dds_model, Minimal, Minimal_paths {
  exename   = subscriber
  libpaths += model

  Source_Files {
    MinimalSubscriber.cpp
  }
}
```

3. 建模

图形建模 Eclipse 插件用于构建 OpenDDS 模型并将其转换为 C ++代码。



建模以下列图表类型之一完成：

- 主图。这是建模的入口点，可以包含“库”，它们是可重用的包。 下面是对库的描述。
- 策略库。这包含一组 QoS 策略，这些策略通常协同工作以实现整体 QoS 特性。 例如，一个策略库可能具有针对可靠数据传输而调整的策略，而另一个库具有针对高性能数据传输而调整的策略。
- 数据库。包含用于定义要发布的数据结构的数据类型。
- DCPS 库。这是定义发布和订阅实体的拓扑的位置。 其他库中的元素（如 QoS 策略和 Structs）将在此库中使用。

代码生成

OpenDDS Modeling SDK 还提供了一个自定义代码生成的插件。

代码生成插件允许您：

- 选择生成源模型和目标目录
- 定制传输用于生成
- 自定义构建的搜索位置

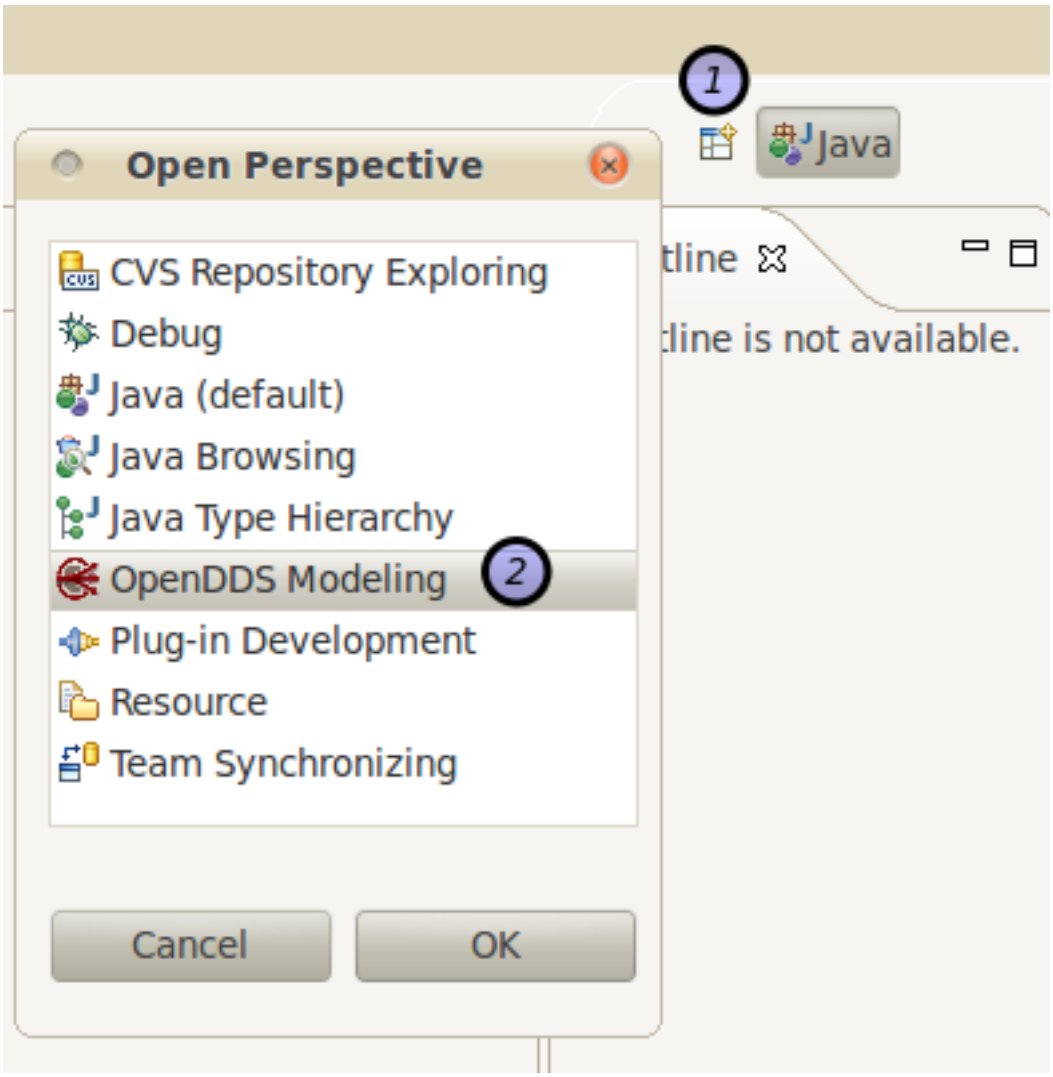
3.1. 使用图表

3.1.1. 使用 OpenDDS Modeling SDK 透视图

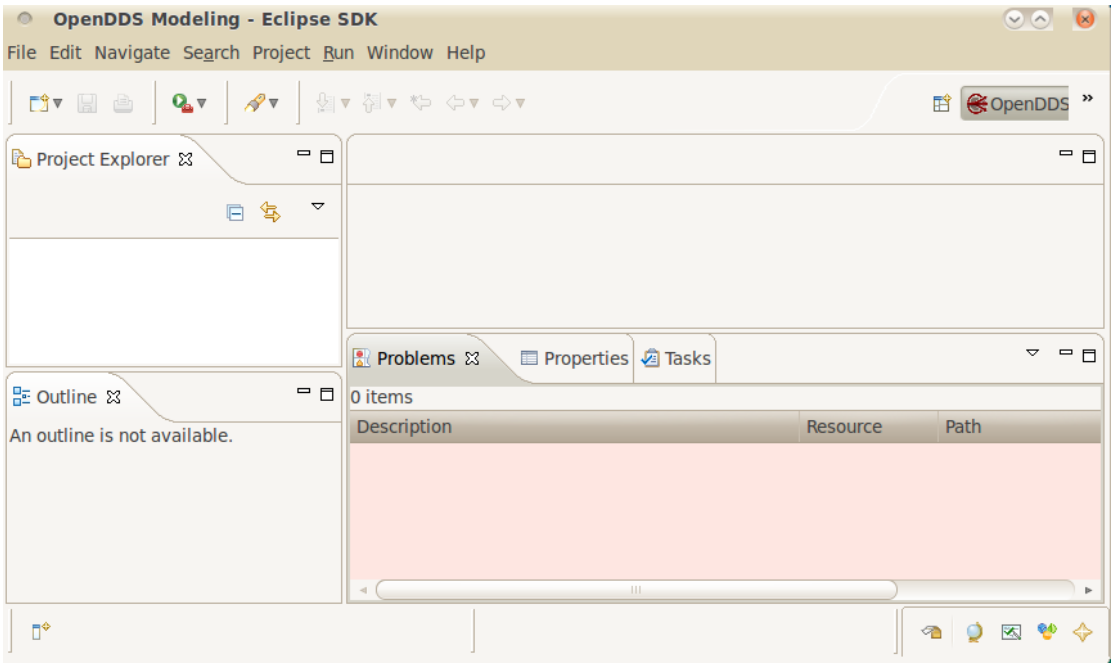
Eclipse 具有透视图的概念，该透视图显示预定义视图，例如工作台上特定位置的项目浏览器和属性。 OpenDDS Modeling SDK 的透视图可以方便地使用模型图。这个透视图可以被激活，通过

1. 单击“透视”按钮，然后单击“其他”...
2. 选择 OpenDDS 建模

这些步骤如下图所示。



然后，您应该看到一个类似于以下内容的 Eclipse 工作台：



3.1.2. 创造图形

可以通过以下三种方式之一将图形添加到画布中：

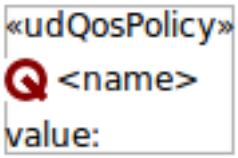
1. 单击调色板中的图形类型，然后在拖动画布上的图形角落时按住鼠标以指定图形的大小。
2. 双击调色板中的图形类型。
3. 单击调色板中的图形类型，然后单击画布上的某个位置。

> 第一种方法的缺点是如果图形具有隔间并且内容被添加到隔间中，则在添加内容之后将不会扩展该图形。 这意味着，根据图形的大小，在展开父图形之前，可能看不到隔间内容。 使用第二和第三种方法具有自动图形重新调整大小以适应添加的子图的优点。

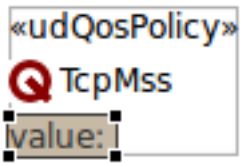
有关自动调整大小的更多信息，请参阅排列图。

3.1.3. 设置元素属性

许多 OpenDDS 建模元素都有可供您设置的属性。 要设置属性，请在选择属性后单击它并为其输入值。 步骤如下所示，使用 QoS 策略作为示例。
创建元素。 例如，双击 Palette 中的元素。 这将使元素具有未定义的属性（在此示例中，为策略名称和值）：



要设置值属性，请先选择它。 （可能需要先在图形外部的某处单击以取消选中它，然后重新选择该图形。）

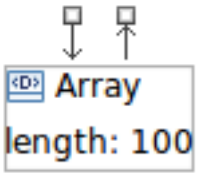


现在单击该属性，将出现一个输入框，允许您键入属性值。 在编辑时，属性的名称将暂时消失。

编辑后，您应该看到属性的名称，后跟它的值：

3.1.4. 连接图形

对于旨在彼此连接的图，可以使用连接创建助手。将鼠标悬停在要连接的图形上时，连接处理程序将如下所示。



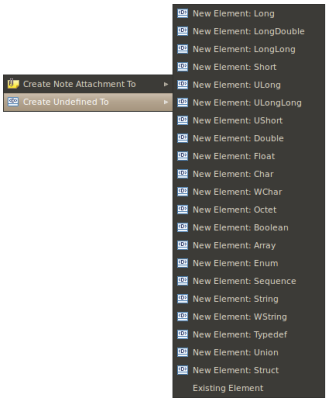
如果图中已存在源图和目标图，则可以通过以下两种方式之一直接连接它们：

1. 如果所选图形是连接源，则可以通过单击传出箭头上的框并拖动到目标来连接到目标图形。
2. 如果所选图形是连接目标，则可以单击传入箭头上的框并拖动到源图形。

另一种建立连接的方法如下：

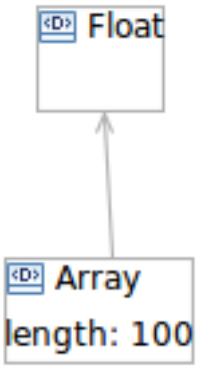
1. 如果选择的图形是源图形，则单击连接到连接手柄的传出箭头的框并拖动到画布的空白部分。然后，系统将提示您创建允许的目标图形，或从图表中的现有图形中进行选择。
2. 如果所选图形是目标图形，则单击连接到连接手柄的传入箭头的框并拖动到画布的空白部分。然后，系统将提示您创建允许的源图。

源图的示例连接菜单如下所示。



正如您现在可能已经发现的那样，您还可以创建与 Note 图形的连接。 虽然这是帮助记录模型的好方法，但请记住，注释仅与图表相关联，而不是域模型本身的一部分。

连接后，目标图形将连接箭头。 示例连接如下所示。



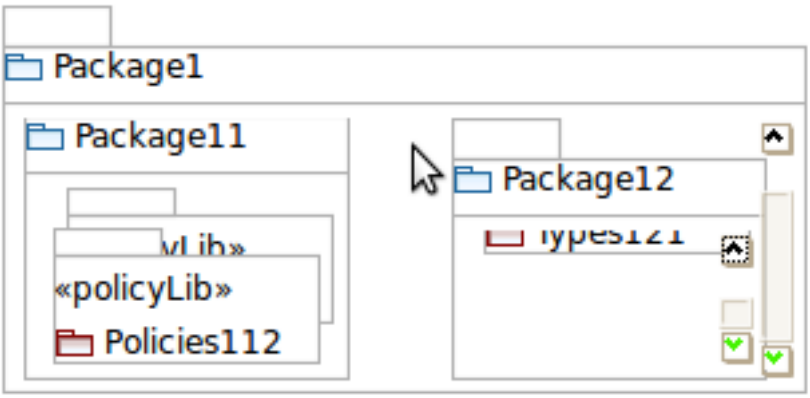
3.1.5. 在图表之间移动

在使用 OpenDDS 模型中的各种库的过程中，您可能会发现自己打开了许多图表编辑窗口。 为了便于在这些窗口之间来回移动，您可以使用 Eclipse 工具栏上的后退和前进按钮，如下所示。

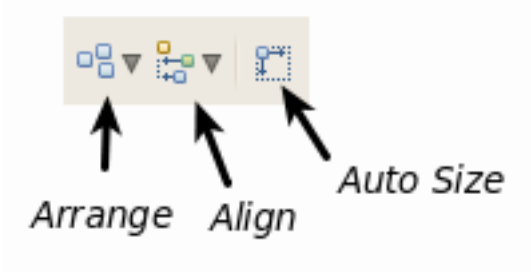
如果要在图表中进行更改并激活另一个图表或 Eclipse 会话中的任何其他窗口，则会自动保存模型和图表文件。 这样做是为了避免在引用相同文件的不同图编辑器中进行更改时可能发生的冲突。
请注意，撤消跨文件保存，以便您可以撤消在以后决定不想保存时保存的更改。

3.1.6. 安排图形

一些 OpenDDS 建模元素（例如包和域参与者）可以具有嵌套的图形区域。有时这些隔间会在您向其添加图形时变得混乱，如下所示。



图表工具栏包含一些按钮，用于自动调整图元素的大小和重新定位：

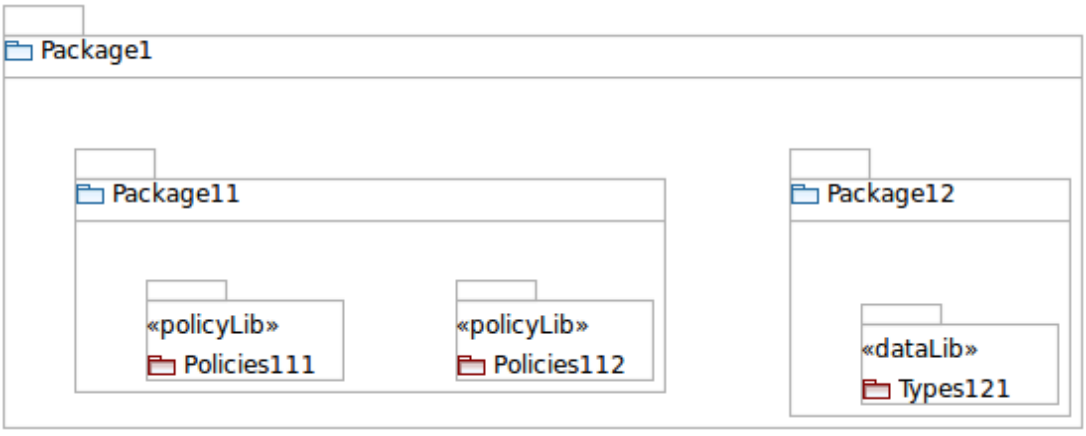


“排列”按钮将重新定位图形，使它们不会重叠。如果您选择一个隔间内的图形，然后选择全部排列，则该图形和隔间内的其他图形将重新排列。

“自动调整大小”按钮将根据需要扩展父图形以容纳其子项。如果自动尺寸有未使用的空间，它也会缩小数字。

> 如果您手动调整带有隔间的图形，折叠隔间不会导致图形缩小，如您所料。选择该图，然后启用“自动调整大小”以使图形缩小。

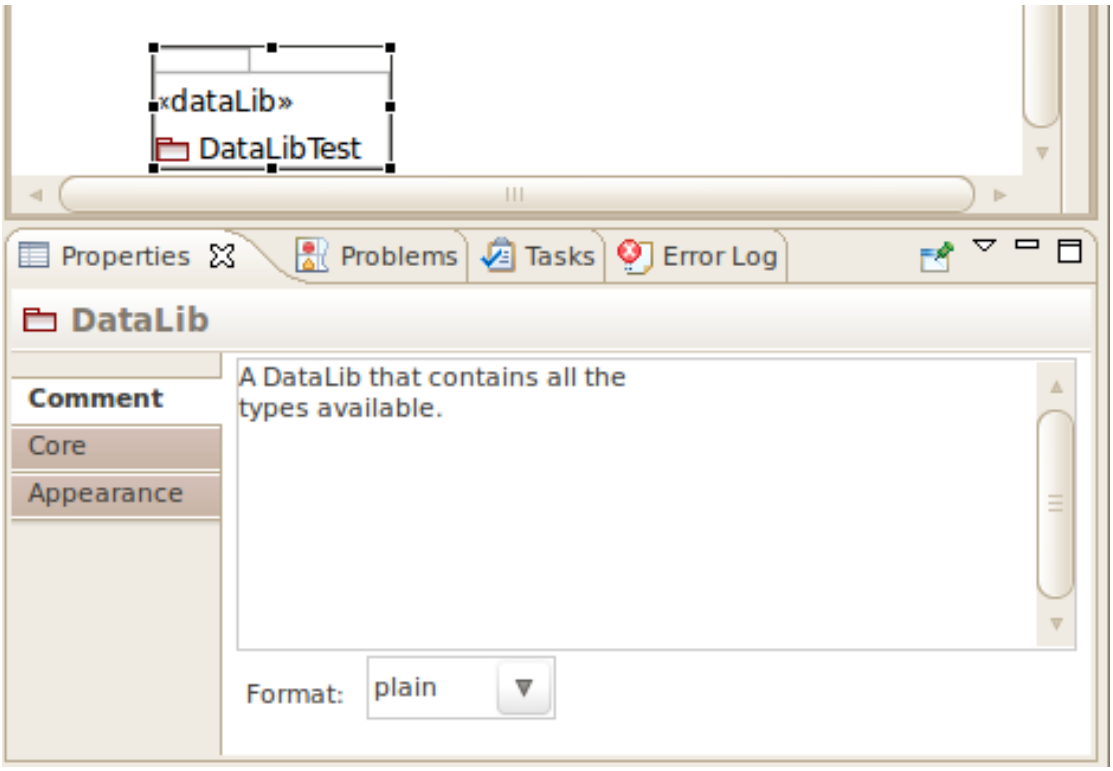
自动尺寸和排列所有几次后，您可以解除隔间的混乱：



3.1.7. 添加注释

您可以为图表中的每个元素提供多行注释，以帮助传达元素背后的意图以及任何其他信息。 这些注释保留在生成的代码中。

要添加注释，请转到属性表并选择“注释”选项卡。 然后，您可以在工作表中键入注释。 一个例子如下所示。



注释下方是一个下拉框，用于指示注释的格式是纯文本还是应该作为 Doxygen 注释映射到 C ++源代码。

注释也可以与图表本身相关联。 如果单击图表的空白部分，您将在属性表中看到“注释”选项卡。 对于主图，注释适用于模型本身。 对于库图，注释适用于库。

> 对于 DataLibs，可以将注释添加到“字段和键”。 例如，选择一个 Field，然后单击 Comment 选项卡以输入 Field 的注释。

3.1.8. 将图表导出为图像

虽然不是很明显，但您可以将图表保存为图像，方法是右键单击图表上的任意位

置以显示上下文菜单，然后选择文件 - >另存为图像文件....

3.2. 使用 OpenDDS 模型

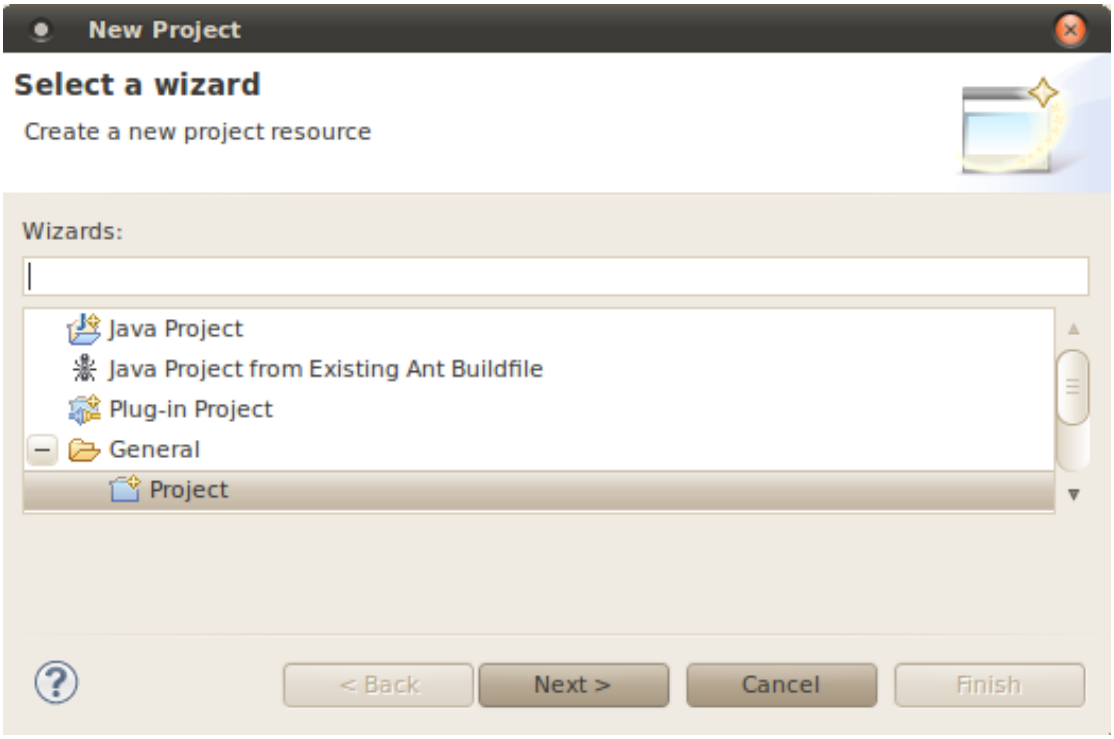
3.2.1. OpenDDS 模型

OpenDDS 模型由可能重复使用的 OpenDDS 元素的 “库” 和可以包含这些库的可选包组成。

> 此上下文中的库与 OpenDDS 中使用的 C ++目标文件库的概念不同。 即使您可能正在使用不同的建模库，生成的代码也将针对单个目标文件库。

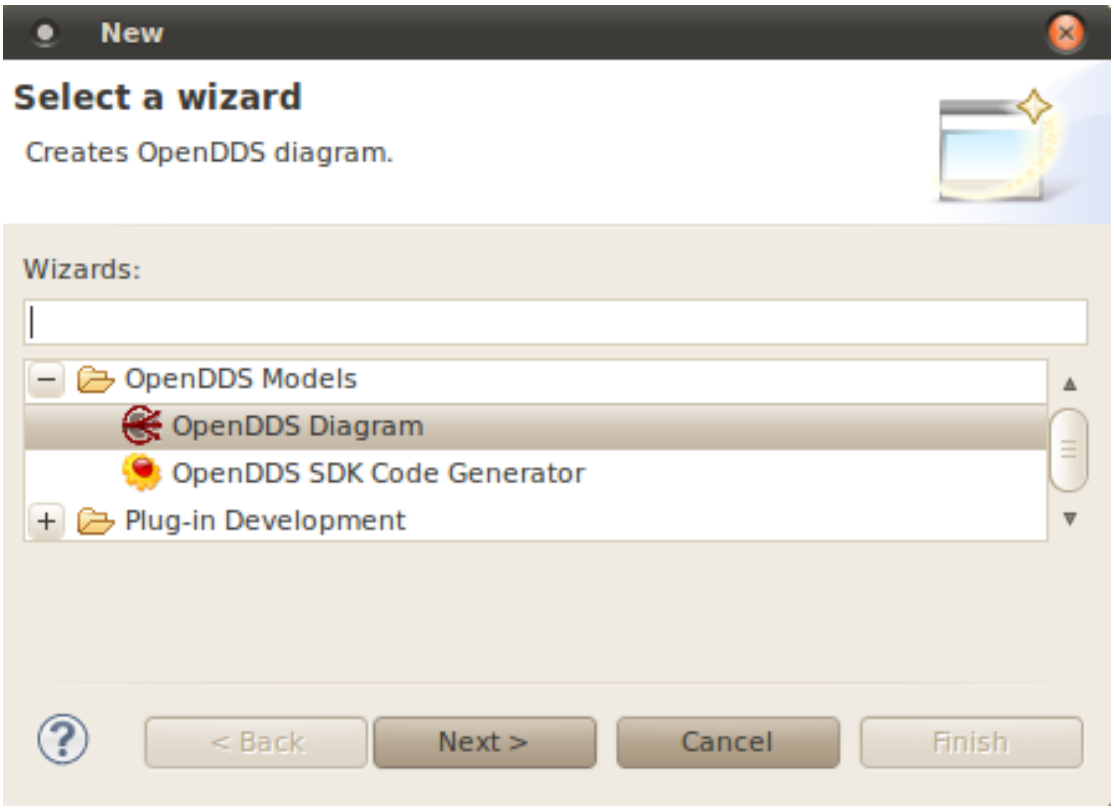
3.2.2. 创建模型

要创建 OpenDDS 模型，必须标识 Eclipse 项目以包含模型。可以使用任何类型的现有项目，也可以从菜单 Eclipse File - > New - > Project ...创建项目。可以使用一般项目类型，如下所示。



选择项目后，可以从 Eclipse File - > New - > Other ...（Ctrl + N）菜单项创建一个新的 OpenDDS 模型，选择 OpenDDS Models 类别下的 OpenDDS Diagram，

如下图所示。



Create OpenDDS Diagram 向导将提示输入模型的名称。此名称用于自动生成的源文件名，并用作共享库的名称。它还用作 C ++命名空间的标识符。

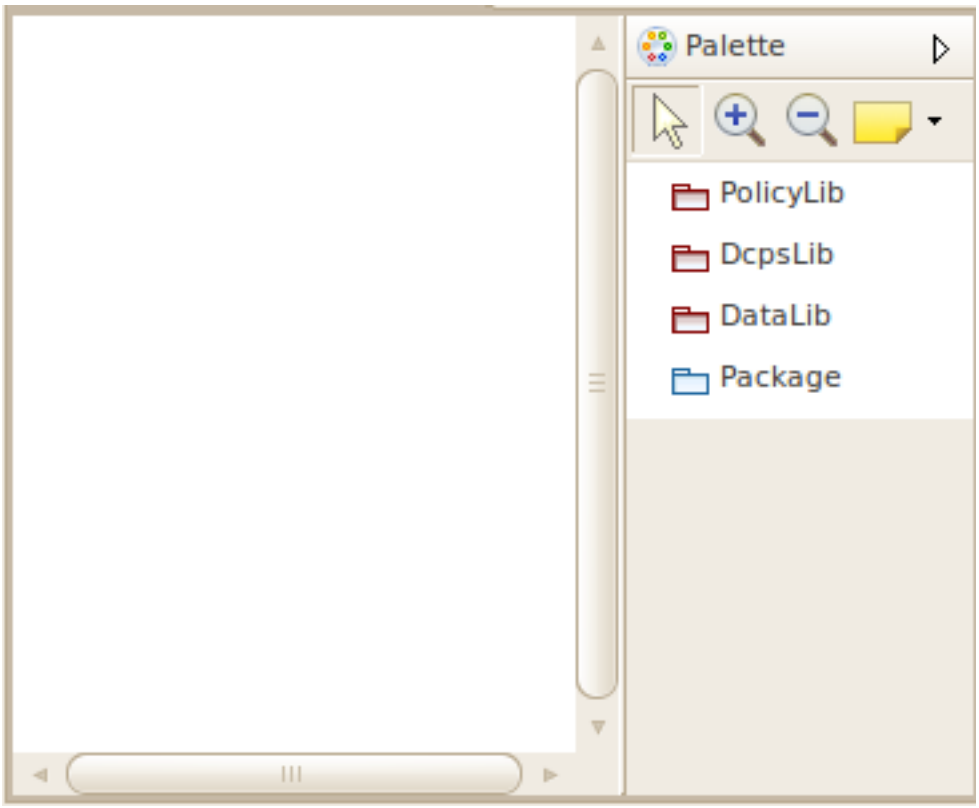
> 如果您决定稍后更改模型名称，请单击主图表的 emtpy 部分，以在图表下方的“属性”选项卡的属性表中显示该名称。

选择模型名称后，Create OpenDDS Diagram 向导将提示输入图表文件的名称以及包含域模型本身的文件的名称。建议在指定这些名称时使用默认值，即模型本身的名称。

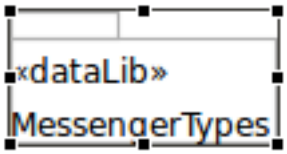
许多基于图表的建模工具将模型和图表元素保存在单个文件中。对于 OpenDDS SDK，这些文件是分开的。虽然这样可以考虑模型的替代表示，但在共享 OpenDDS 模型时应注意包含域模型文件和图表文件。

3.2.3. 填充模型

在选择新的 OpenDDS 图表之后，您将看到一个编辑器，其中包含一个画布和一个“库”调色板以及一个可以拖到画布上的包。该编辑器的示例如下图所示。



您可以首先选择它，然后双击它或按 Enter 键打开画布上的库。当您在图中看到一个粗边框时，可以验证它是否已被选中，并在其上重新调整大小，如下所示。



打开库时，编辑器窗口中会有一个新的编辑器窗口。有关使用多个图编辑器的信息，请参阅在图表之间移动。

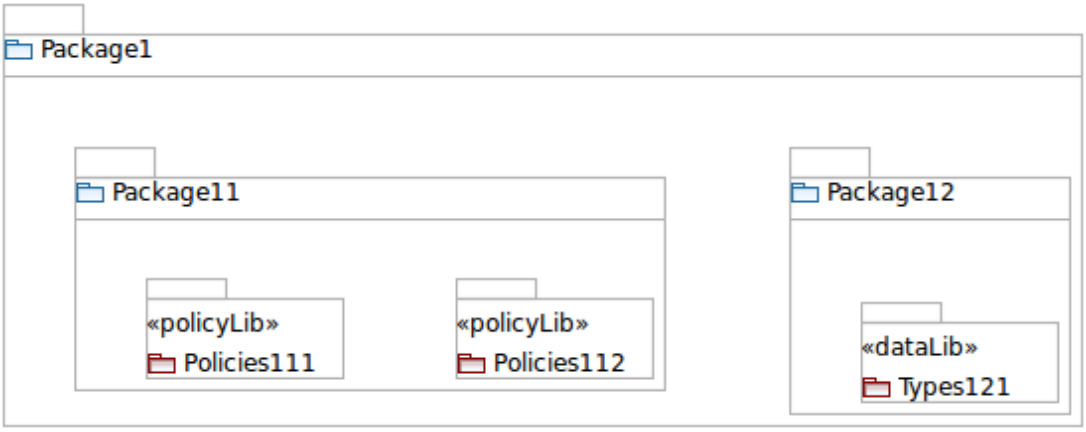
可以选择按包组织库。有关更多信息，请参阅使用包。

> 尽管 OpenDDS Modeling 透视图包含在 “Core” 选项卡中公开底层模型的 Properties 视图，但已尽力减少图和 Properties 视图之间的需求。在 “属性” 视图中进行修改时应小心，因为可能违反构建图表组件的某些建模假设。

3.2.4. 使用包

虽然您可以通过将库放在单独的模型文件中来对库进行分区，但是包提供了一种

在同一模型文件中组织库的方法。 包可以是分层的，可以包含库或其他包：



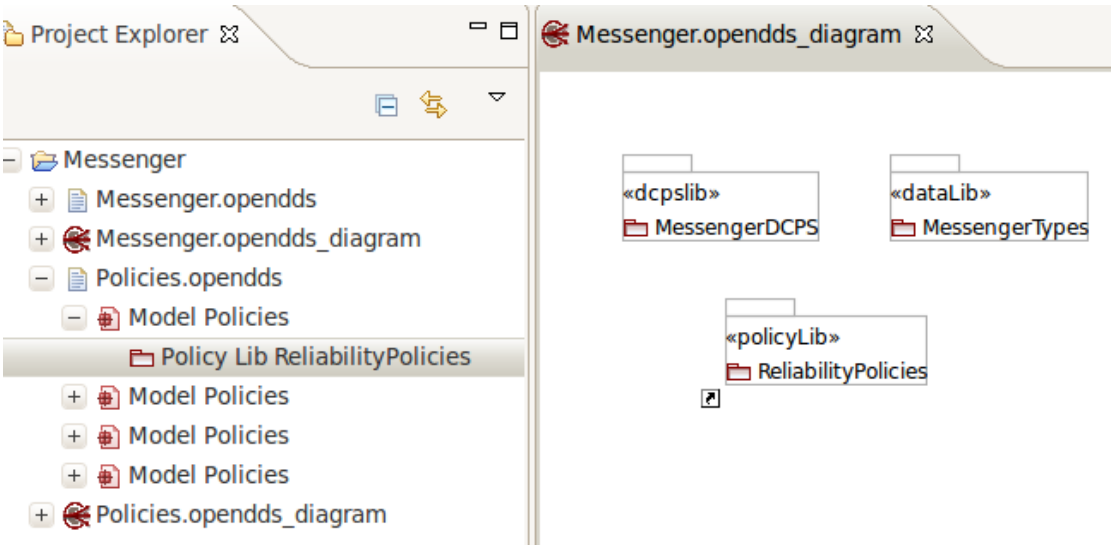
生成 C ++代码时，包将映射到名称空间。

> 如果您有许多嵌套包，您可能需要查看有关如何自动调整大小和重新定位包和包内容的数据，以便它们不会重叠。

3.2.5. 使用其他模型文件中的库

您可以通过将这些库从 Project Explorer 视图拖动到主图画布来明确表示您将使用其他 OpenDDS 模型文件中的库。 这些快捷方式将附加一个箭头，表示它们是其他文件中元素的别名。

例如，在下图中，Policies.opendds 文件中的 ReliabilityPolicies 库在主图上显示为快捷方式。 然后，可以从引用共享策略时选择此库中的策略。



可以显示多个包（例如，上图中所示的四个“模型策略”）。 这是无害的，可以

使用任何一个包。

3.2.6. 验证库和包

在使用模型生成代码之前，您应该首先执行验证。 请参阅执行验证以了解如何进行验证。

3.3. 使用策略库

3.3.1. 策略库定义

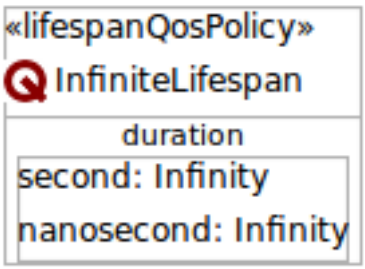
策略库允许由 DCPS 库中定义的各种 DCPS 域元素重用 QoS 策略设置。

3.3.2. 创建策略库

您可以创建由不同 DCPS 域元素重用的 QoS 策略“库”。
在主图中，可以通过双击选项板中的 PolicyLib 工具，或者单击 PolicyLib 工具，然后拖动策略库图应占用的图表面上的区域来创建策略库。 之后，指定用于库的名称。
然后可以通过双击库来打开库。 将打开一个新的图表编辑器（使用与主图表相同的图表文件）。

3.3.3. 添加策略

对于要定义的策略，从选项板中选择策略类型并将其添加到画布。 提供策略的名称并根据需要编辑策略字段。
在 DcpsLib 图表中为域元素选择共享策略时会列出策略名称，因此请确保它们是可描述性的。
某些策略的持续时间默认为无限期。在这种情况下，持续时间的秒和纳秒字段将显示“无穷大”。（这对应于 DDS IDL 模块中的 DURATION_INFINITE_SEC 和 DURATION_INFINITE_NSEC 常量。）



通过在编辑字段时输入 “Infinity” （或简称 “inf” ），可以指示秒或纳秒字段是无限的。

3.3.4. 验证策略

在使用策略库之前，您应首先执行验证。请参阅执行验证以了解如何进行验证。

3.4. 使用数据库

3.4.1. 数据库定义

您可以在数据库中定义 Structs 和支持类型。 然后可以将结构用作 DCPS 库中的主题类型。

3.4.2. 创建数据库

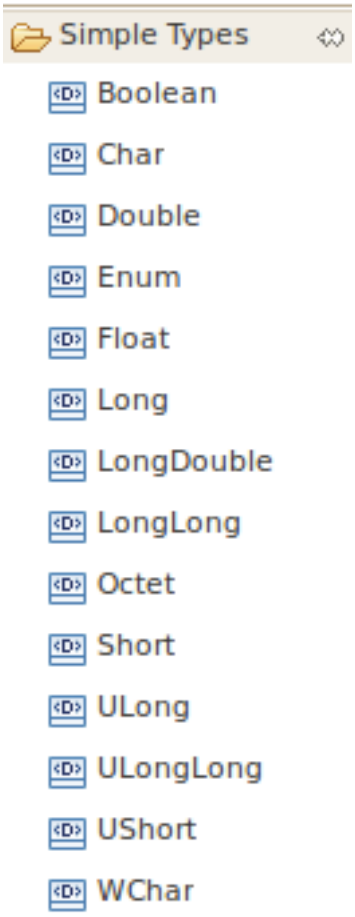
在主图中，可以通过双击选项板中的 DataLib 工具，或者单击 DataLib 工具，然后拖动数据库图形应占据的图表面上的区域来创建数据库。 之后，指定用于库的名称。

然后可以通过双击库来打开库。 将打开一个新的图表编辑器（使用与主图表相同的图表文件）。

3.4.3. 添加类型

基本类型

最终，其他类型必须引用 Long，Octet 或 String 等基本类型。 使用连接器时，选项是将这些基本类型添加到图表中，然后连接到这些类型。 或者，您可以使用图形连接器手柄在需要时快速创建基本类型。 （如果您愿意，可以在同一图表中使用相同基本类型的多个图形。）



带字段的类型

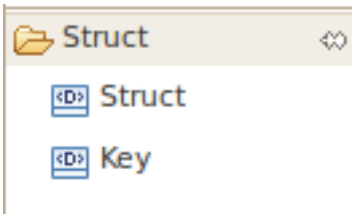
Struct 和 Union 具有被命名为类型引用的字段（类似于类属性）。 要添加字段，请单击调色板中的 “字段” 工具，然后单击 “结构” 或 “联合” 以包含 “字段” 。然后使用 “name：type” 形式填写 Field。 例如，要指示 Field 将被命名为 count 并指向 Long，您将输入 “count：Long” 。（不需要首先创建基本类型。）除了当

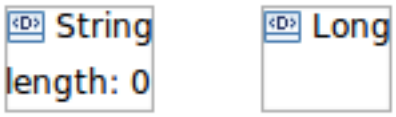
前打开的图表中的类型之外，类型可以来自任何活动 DataLib。

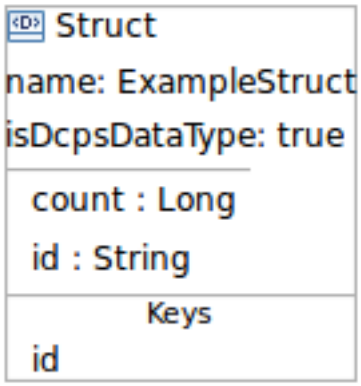
如果未正确定义字段（例如，引用不存在的类型），则会在其位置看到<...>。 使用 “名称：类型” 表单编辑字段或从结构中删除字段。

结构

属性 isDcpsDataType 设置为 true 的结构是用于主题类型的候选类型。 Struct 示例如下所示。





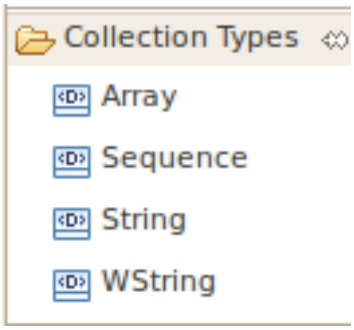


通过从调色板中选择 Struct 并添加到图中来创建 Struct。
在为 Struct 赋予名称之后，您将需要向 Fields 添加 Fields，如带字段的类型中所述。
为 Struct 定义 Fields 后，您可能需要重新排序它们。您可以通过拖动字段并将其放入新位置来完成此操作。
键值

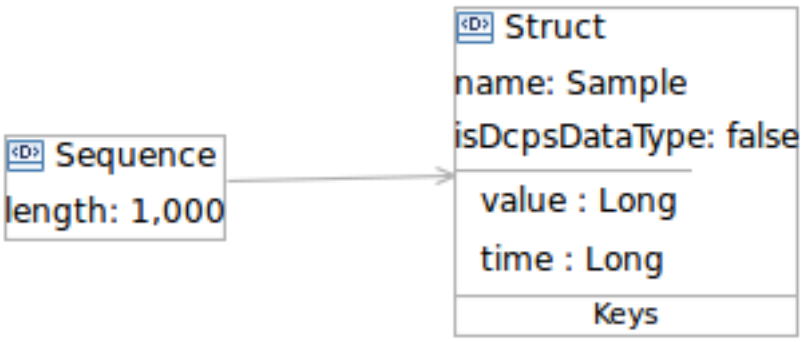
如 OpenDDS 开发人员指南中的 OpenDDS 下载页面所述，键值是用于标识主题中不同实例的字段。这些关键字段由 Struct 的 Keys 区域中的元素标识，其名称是充当键的字段的名称。
创建 Struct 的键与创建 Fields 的方式类似。从选项板中选择 “Key” 工具，然后将其添加到 Struct。然后在 Struct 图中键入应该用于 Key 的 Field 的名称。与可以重新排序字段的方式类似，可以通过拖放重新排序键值。

数组和序列类型

对于集合类型 Array 和 Sequence，必须指定用于集合的类型。 这是通过将 Array 或 Sequence 连接到集合的类型来完成的。



结构序列的一个例子如下所示。

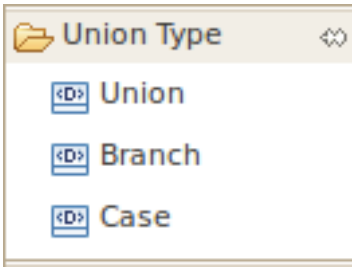


将 Arrays 和 Sequences 连接到其他类型时，如果您不熟悉如何进行连接，请查看如何建立连接可能会有所帮助。

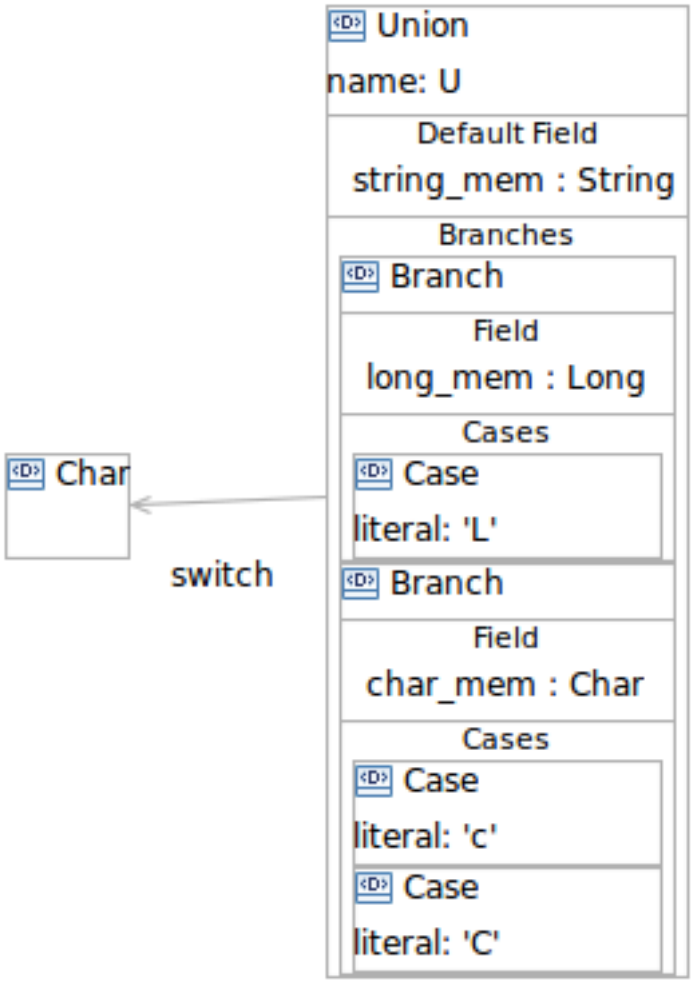
联合

如果任何案例文字（Case literals）与交换规范匹配，则联合包含用于保存案例列

表的分支以及要应用的字段。



放置在可折叠隔间中的这些分支和盒子显示在下面的示例中。



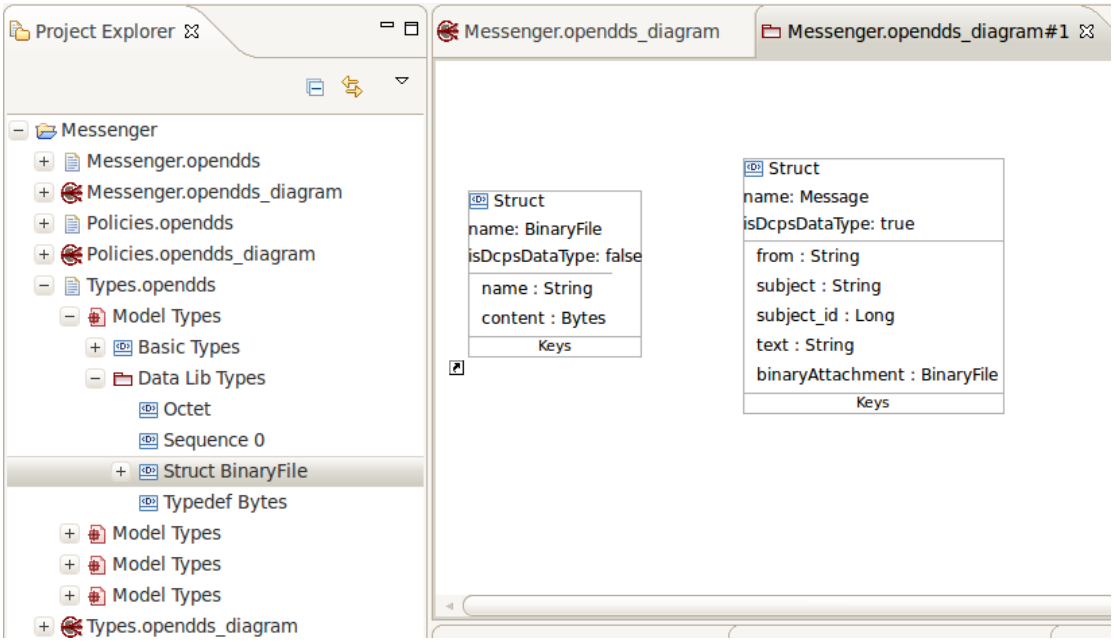
此示例对应于 IDL：

```
union U switch (char) {
case 'L':
    long long_mem;
case 'c':
case 'C':
    char char_mem;
```

```
default:
    string string_mem;
};
```

3.4.4. 引用其他 DataLibs 中的类型

您可以使用快捷方式从其他 DataLibs 引用类型（用于将库从其他模型文件放置到主图上的相同机制）。
以这种方式使用快捷方式的示例如下图所示。 在这种情况下，Messenger Struct 使用在另一个模型文件中定义的类型 BinaryFile。



3.4.5. 验证数据类型

在使用数据库之前，您应首先执行验证。请参阅执行验证以了解如何进行验证。

3.5. 使用 DCPS 库

3.5.1. DCPS 库定义

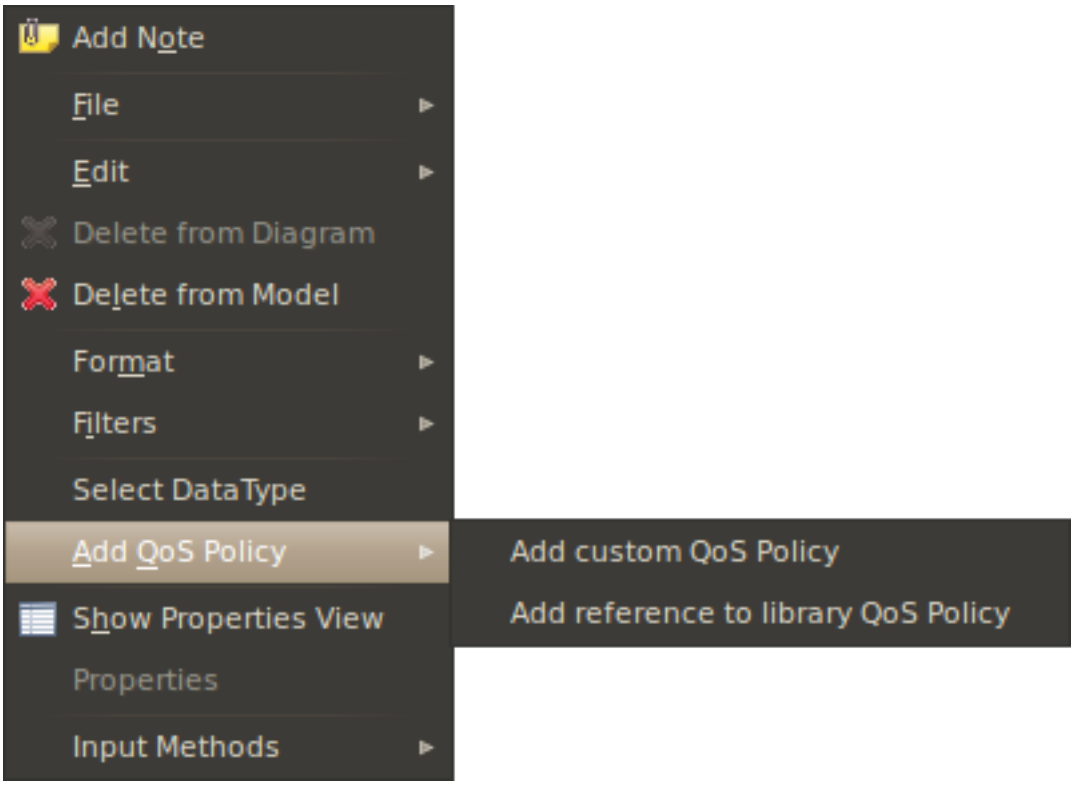
以数据为中心的发布 - 订阅 (DCPS) 库是您定义构成发布/订阅拓扑的实体的位置。

3.5.2. 创建 DCPS 库

在主图中, 可以通过双击调色板中的 DcpsLib 工具, 或者通过单击 DcpsLib 工具, 然后拖动 DCPS 库图形应占据的图表面上的区域来创建 DCPS 库。 之后，指定用于库的名称。
然后可以通过双击库来打开库。 将打开一个新的图表编辑器（使用与主图表相同的图表文件)。

3.5.3. 使用 QoS 策略

许多 DCPS 域实体可以具有与它们相关联的某些 QoS 策略。 策略可以是该实体特有的自定义策略，也可以是策略库中定义的共享策略。
要向实体添加策略, 请右键单击实体以显示上下文菜单, 然后选择 “添加 QoS 策略” 。 从那里，您可以选择是应该创建自定义策略还是应该使用对库中策略的引用。



可折叠隔离专区用于自定义策略，可折叠隔离专区用于共享策略。 如果您不确定哪个隔间是哪个，您可以将鼠标悬停在任何隔间上，并显示隔间的名称。

3.5.4. 创建一个域

应至少定义一个 Domain 元素。 这可以通过从调色板添加域工具来完成。

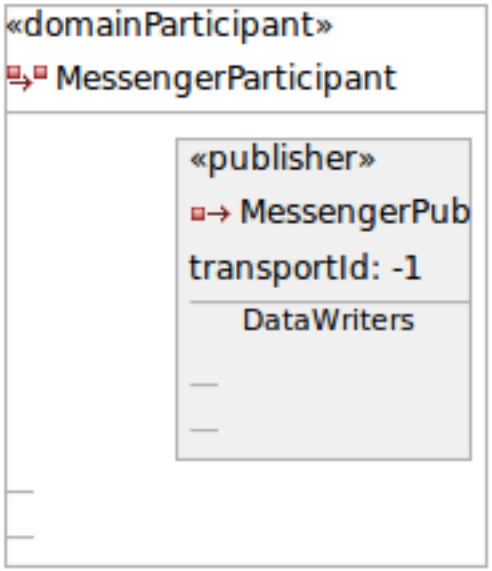
3.5.5. 创建 DomainParticipant

要创建 DomainParticipant，请从选项板中选择 DomainParticipant 并添加到画布。在给参与者一个名字之后，这将是将参与者与其相应的域连接的好时机（此时您可能想要查看连接图形）。
DomainParticipant 可以具有与之关联的 QoS 策略。

3.5.6. 创建发布者

可以将 Publisher 添加到 DomainParticipant。 要添加发布者，请从调色板中单击“发布者”，然后单击“DomainParticipant”图。 DomainParticipant 可以包含

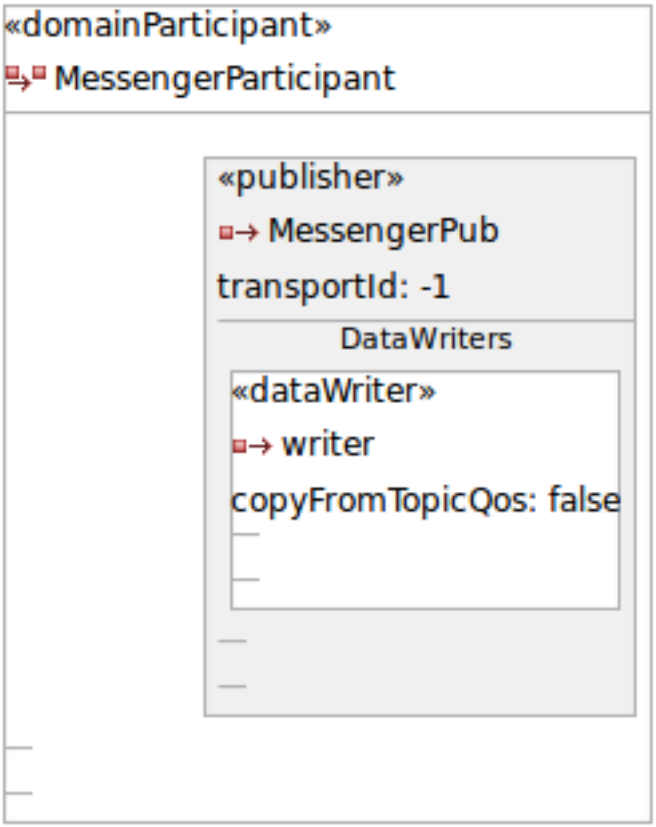
多个 Publisher。



可以根据需要将 QoS 策略添加到发布者。

3.5.7. 创建 DataWriter

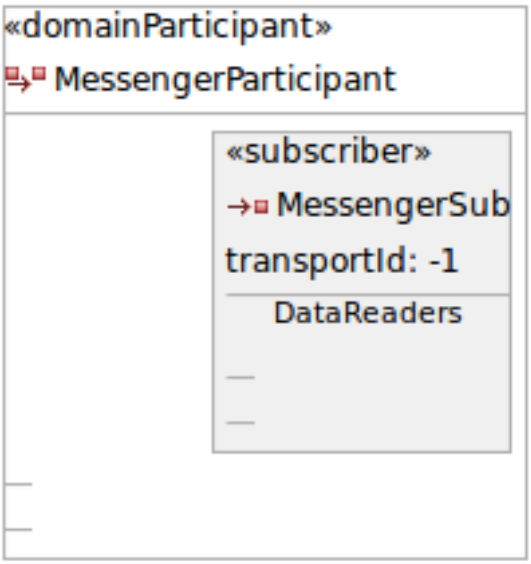
创建发布者时，默认情况下会将 DataWriter 添加到发布者。可以通过从调色板中选择 DataWriter 并将其添加到 Publisher 的 DataWriter 隔离专区来创建其他 DataWriters。 可以根据需要将 QoS 策略添加到 DataWriter。



必须将 DataWriter 连接到主题才能在语义上完成。 有关如何将 DataWriter 连接到主题的信息，请参阅连接图。

3.5.8. 创建订阅者

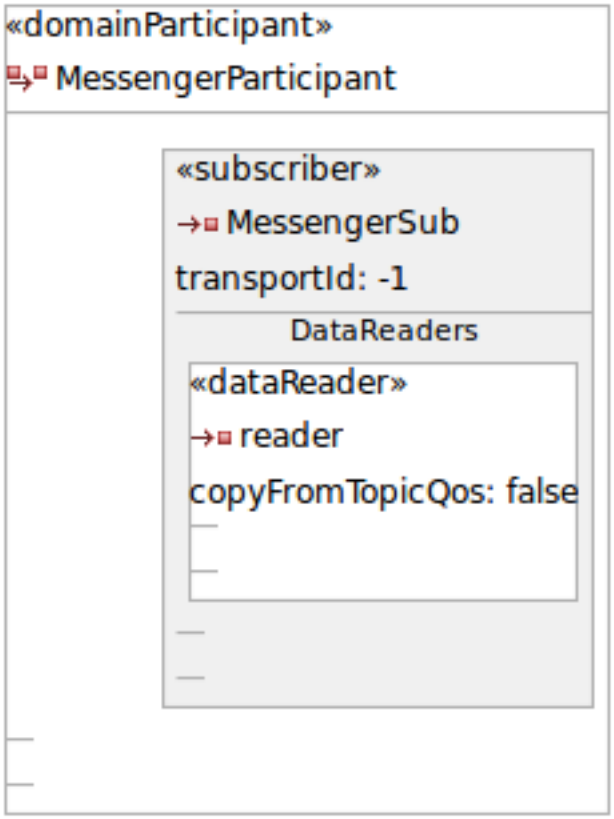
订阅者可以添加到 DomainParticipant。要添加订阅服务器，请从调色板中单击订阅服务器，然后单击 DomainParticipant 数字。 DomainParticipant 可以包含多个订阅服务器。



可以根据需要将 QoS 策略添加到发布者。

3.5.9. 创建 DataReader

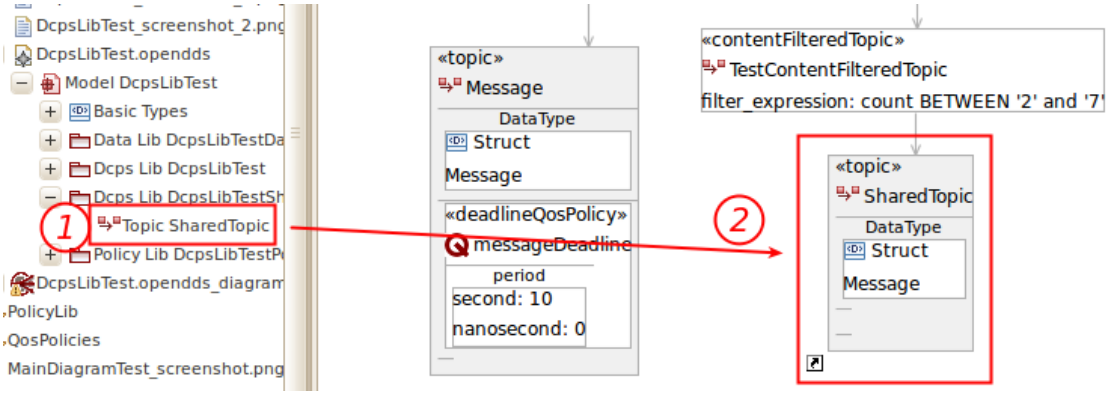
创建订阅者时，默认情况下会将 DataReader 添加到订阅者。 可以通过从选用板中选择 DataReader 并将其添加到订阅者的 DataReader 隔离专区来创建其他 DataReader。 可以根据需要将 QoS 策略添加到 DataReader。



DataReader 必须连接到 Topic，ContentFilteredTopic 或 MultiTopic 才能在语义上完成。 有关如何将 DataReader 连接到主题相关图的信息，请参阅连接图。

3.5.10. 使用主题

Dcpslib 应该定义至少一个 Topic 或引用另一个 DcpsLib 中定义的 Topic。
可以通过从调色板中选择主题来创建主题。或者，可以引用在另一个 DcpsLib 中定义的主题。这避免了需要在多个 DcpsLib 中复制主题。要将另一个 DcpsLib 中的主题放到当前图表上：
在 Project Explorer 视图中，展开与.opendds_diagram 文件关联的.opendds 文件，然后选择要共享的主题。
将主题拖动到画布上。
这些步骤如下所示：



您现在可以链接到此主题，就好像它是在原始 DcpsLib 中定义的一样。

需要为 Topic 的 DataType 指定 Struct。要从允许的 Structs 中进行选择，请右键单击 Topic 以显示上下文菜单。选择 “选择数据类型” 以显示一个对话框，显示您可以从中选择的所有允许的结构。

有关将 DataReader / DataWriter 连接到 Topic / ContentFilteredTopic / MultiTopic 的步骤，请参阅将 DataReader / DataWriter 连接到主题。

ContentFilteredTopic

DataReader 可以选择连接到 ContentFilteredTopic。每个 ContentFilteredTopic 都必须连接到主题。与 Topic 类似，ContentFilteredTopic 必须为其 DataType 指定 Struct。

多主题

DataReader 可以选择连接到 MultiTopic。与 Topic 类似，MultiTopic 必须为其 DataType 指定 Struct。

3.5.11. 使用隔间内的隔间

当您从 DomainParticipant 向下拉到具有 QoS 策略的 DataReader 和 DataWriters 时，隔离专区内将有许多隔离专区。 为了帮助在这些周围进行操作，您可以折叠目前不感兴趣的隔间，并打开您感兴趣的隔间。此外，您可能想要查看有关如何自动调整隔间中元素的大小和重新定位的图形，以便他们不要 重叠。

3.5.12. 验证 DCPS 类型

在使用 DCPS 库之前，您应首先执行验证。 请参阅执行验证以了解如何进行验证。

3.6. 执行验证

可以对建模元素执行验证检查，以帮助确保它们在语义上是正确的，并且适合用作自动生成代码的基础。

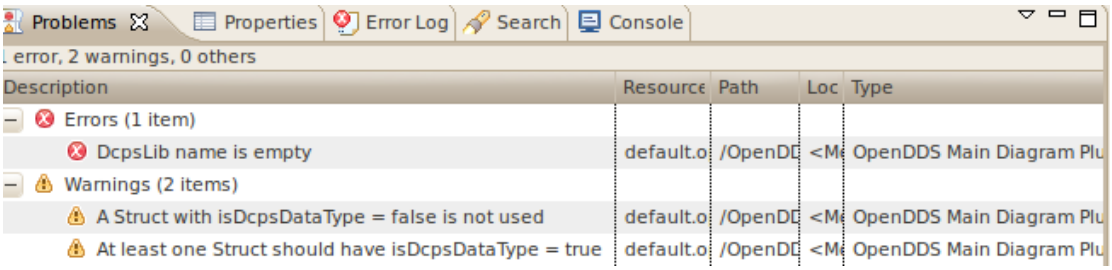
要验证图中显示的建模元素，请选择菜单项“编辑” - > “验证” 。

在可行的情况下，如果存在与该元素相关的错误或警告，则将使用小图标修饰单个元素。下图显示左图上的黄色警告装饰和右图上的红色错误装饰

错误表示模型存在语义问题，导致代码生成问题。警告表示建模构造可能不完整，在某些情况下可以忽略。

如果将鼠标悬停在错误或警告图标上，则弹出窗口将显示该元素的验证失败。

验证失败也可以在 Problems 视图中看到，如下图所示。要打开 “问题” 视图，请选择菜单项窗口 - >显示视图 - >问题



当对同一模型文件中的两个或多个相同类型的库进行验证时，完成的最后一次验证是唯一保留的验证。例如，假设您有一个带有 PolicyLibs PolicyLibA 和 PolicyLibB 的模型，两者都有验证错误。如果打开模型文件，然后

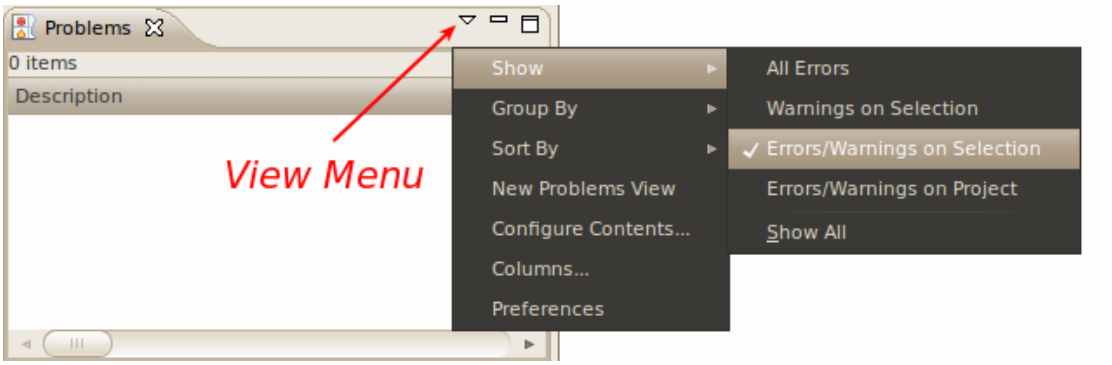
1. 打开 PolicyLibA
2. 执行验证
3. 打开 PolicyLibB
4. 执行验证

只有 PolicyLibB 中的验证错误才会出现在 Problems 视图中。

限制问题视图中显示的错误

默认情况下， “问题” 视图中显示的验证消息的范围包括工作区中的所有文件。

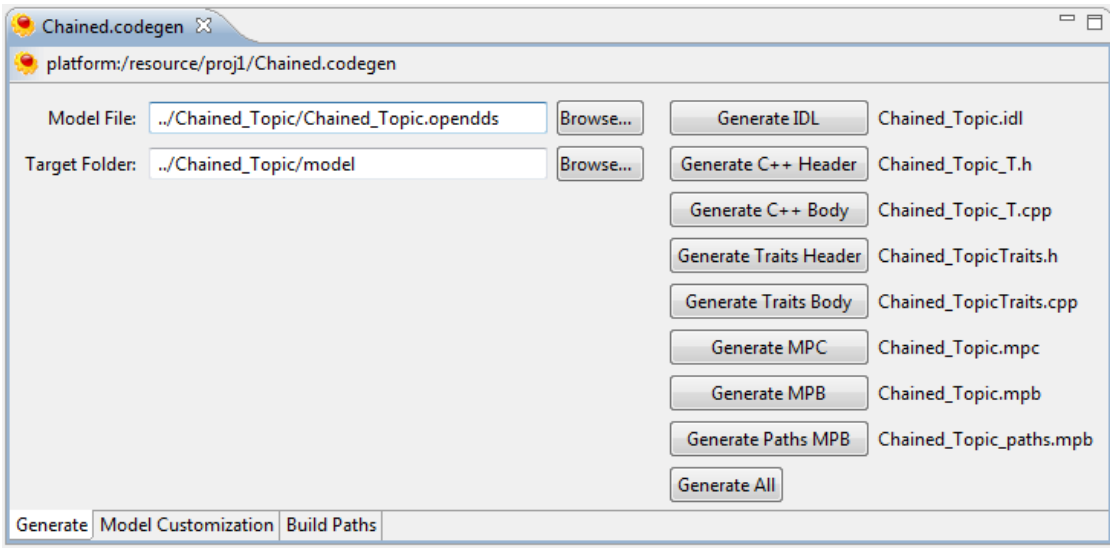
如果您有多个带有验证消息的模型，则可能很难找到特定于打开的图表的消息。您可以配置问题视图以限制显示的消息范围。这可以通过从问题视图的视图菜单中选择显示并指示消息的范围来完成。



4. 代码生成

4.1. 选择生成参数

在 Codegen 编辑器的 Generate 选项卡上选择 generate 参数。



4.1.1. 代码生成参数

生成参数包括：

模型文件： - 模型文件的路径，用作生成的基础。

目标文件夹： - 用作生成目标的目录的路径。

4.1.2. 模型文件

可以使用“模型文件”文本字段旁边的“浏览”按钮选择模型文件。导航到所需的.opendds 模型文件，然后单击“确定”。

为了生成模型的代码，模型需要具有非空模型名称。

目标目录
可以使用“目标目录”文本字段旁边的“浏览”按钮选择目标目录。导航到所需的生成目录，然后单击“确定”。

如果目标目录不存在，代码生成器将创建目标目录。但是，“目标选择”对话框没有用于创建目录的界面。在这种情况下，可以在目标目录文本框中键入目标目录。

4.1.3. 生成按钮

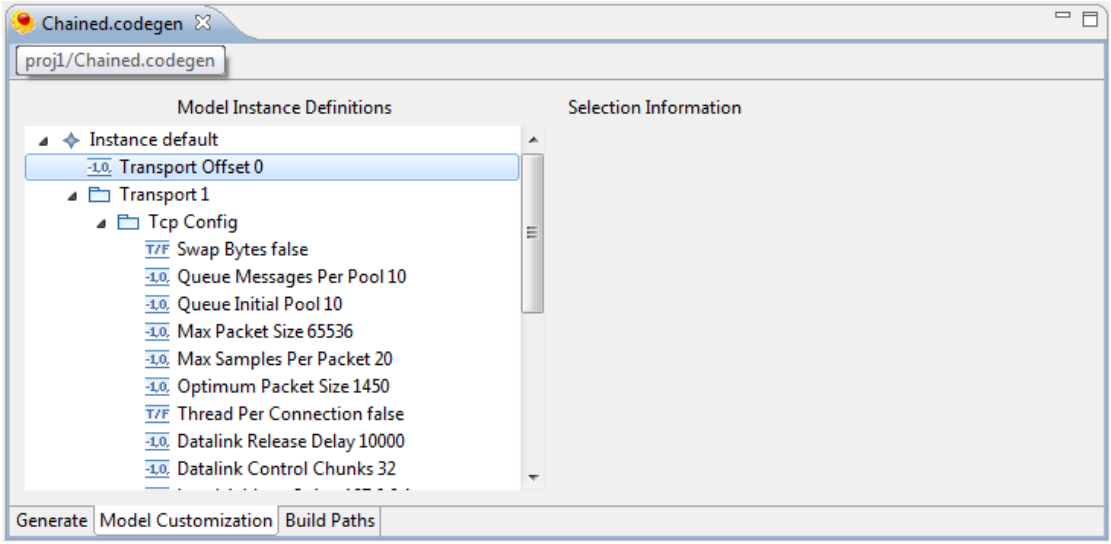
“生成”选项卡上还有一系列“生成”按钮。这些按钮用于生成各种输出文件。每个生成按钮旁边是它将根据模型名称生成的文件的名称。

4.1.4. 生成所有按钮

全部生成按钮是按下每个生成按钮的快捷方式。

4.2. 自定义代码生成

代码生成允许对传输配置进行一些自定义。



4.2.1. 自定义实例

自定义包括一系列实例，这些实例表示底层应用程序的潜在部署方案。

例如，可以指定一个表示 TCP 部署的实例，而第二个实例表示多播部署。

实例传输偏移

在多个实例的情况下，生成的代码允许实例化所有传输配置。实例中的传输 ID 都与基础模型中的传输 ID 匹配，但生成的代码仍必须具有唯一 ID。

为此，生成的 ID 是添加到实例传输偏移量的实例传输 ID，以创建唯一 ID。以这种方式，可以从单个模型生成多个传输配置。

4.2.2. 自定义传输

每个传输都需要子运输配置。为此，请使用上下文菜单添加子 TCP，多播或 UDP 传输。

自定义一个传输

每个传输都可以通过其上下文菜单进一步自定义，再次通过添加子项菜单。

4.2.3. 默认自定义内容

默认情况下，创建 codegen 文件时，编辑器会从所选的.opendds 模型文件中查找传输 ID 列表。它创建一个名为 default 的实例，偏移量为 0 \，并且模型文件

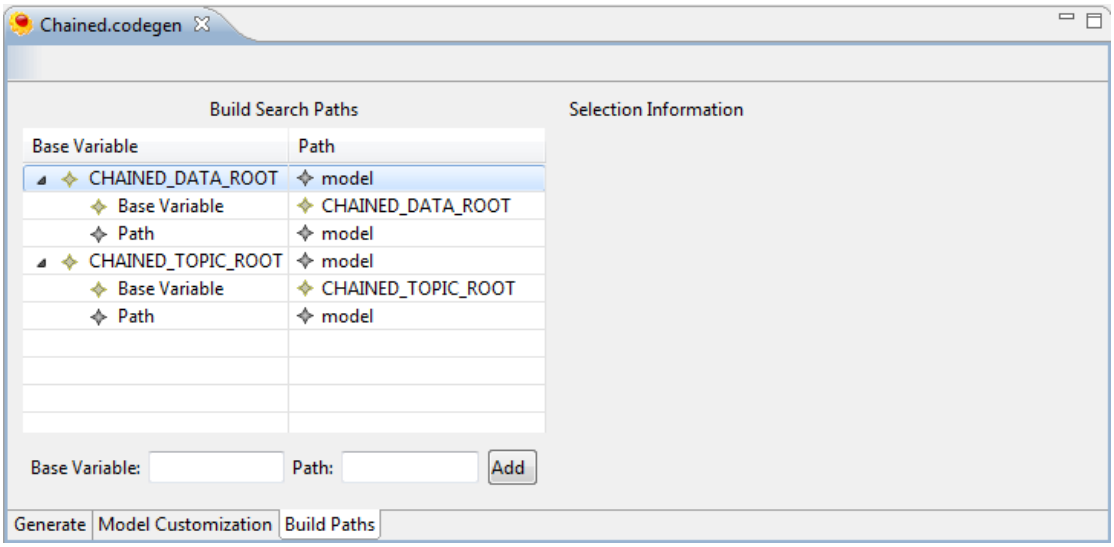
中引用的每个 ID 都为空传输。

至少应将特定配置添加到这些通用传输中，以便选择正确的传输类型。

4.3. 与其他模型集成

当模型引用其他模型时，需要使用“构建路径”选项卡进行其他配置。

> 这些构建路径在构建时用于定位生成的模型库代码以进行编译和链接，而不是在代码生成期间查找模型文件。



4.3.1. 构建路径的参数

构建路径由一系列搜索路径组成，用于查找其他模型。

搜索路径内容

每个搜索路径由两部分组成：

基础变量 - 用作搜索路径基础的环境变量。

路径 - 相对于基础变量值的附加路径，用于创建路径。

搜索路径的每个部分都可以省略。它可以仅包含 Base 变量，也可以仅包含搜索路径。

4.3.2. 默认搜索路径

可以使用“模型文件”文本字段旁边的“浏览”按钮选择模型文件。导航到所需的.opendds 模型文件，然后单击“确定”。

可以通过现有搜索路径的上下文菜单添加其他搜索路径，选择 Add Sibling。

> 如果要从模型文件中删除最后一个构建路径，请使用“大纲”视图，导航到 searchLocations 并在其上下文菜单中选择“添加子项”。