Custom Apps

Module Description

In this module, we will learn on how to create a custom app, that will send interest packets to the forwarder, and will receive data packets from the forwarder.

Task

1. To create a consumer class, that will send constant number of interest packets per second to the forwarder, and will receive the data packets from the forwarder.

2. To create a producer class, that will only respond to the interest packets that are sent by the consumer class.

Prerequisites

1. Faces
2. Forwarder
3. Knowledge about inheritance in C++

Procedure

Task 1 + 2

1. To create our own application, we can either make use of the pre-existing application base classes, or we can create our own application base class. We will do both of these things in this module.

2. The type of base class we use depends on the context. If suppose i am going to implement a consumer application, then i will use the `Consumer` base class. If suppose i am going to implement a producer application, then i will use the `Producer` base class. Else i don't use either and create my own base class.

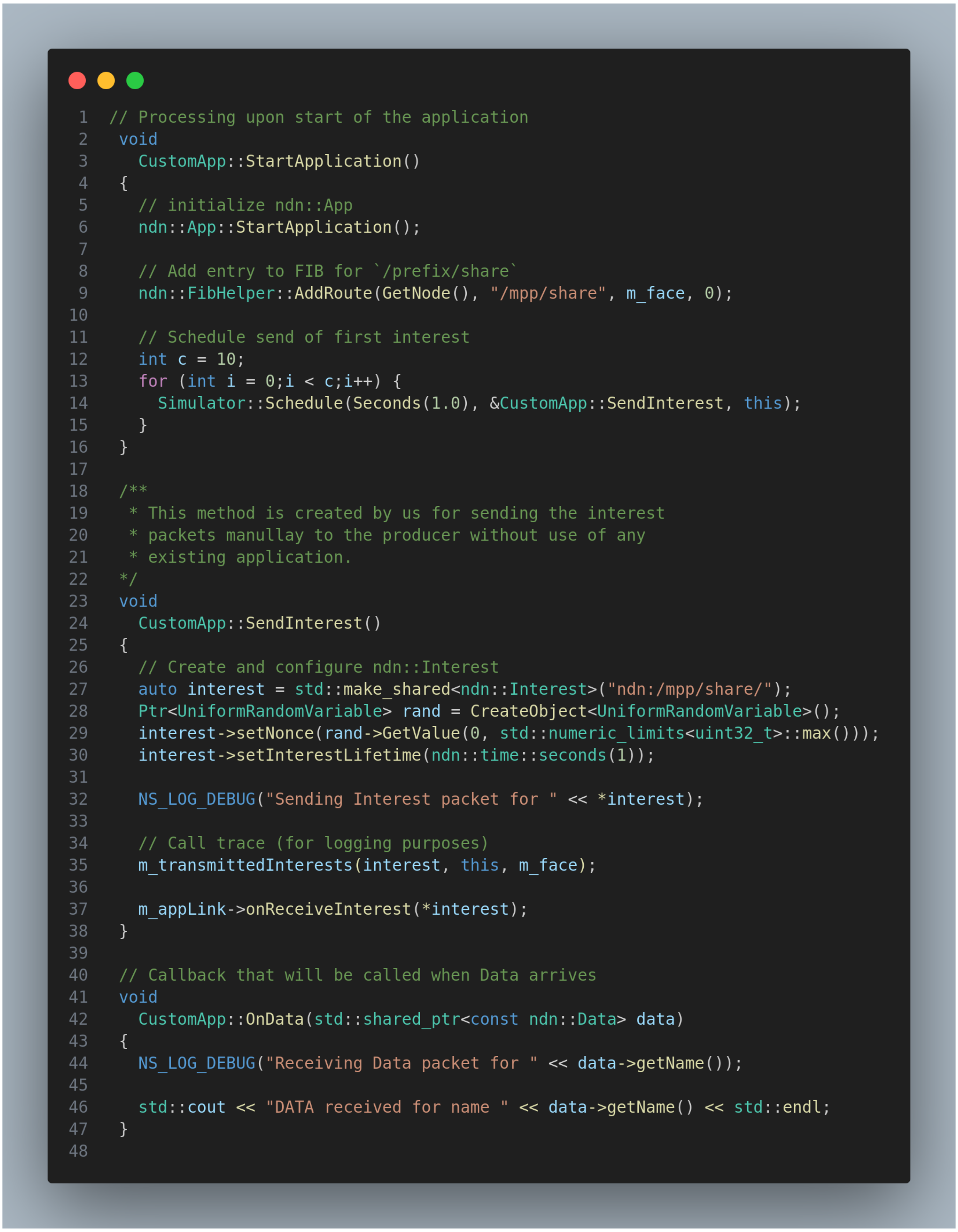
The `Consumer` base class is defined in the ndn-consumer.hpp

The `Producer` base class is defined in the ndn-producer.hpp

3. We will create a class that derives from the base class that we want to use. In this case, we will create a class that derives from the `App` base class.

In this type of application, we use our custom methods to send the interest packet as we are not using any pre-existing methods to send the interest packet.

> See the `SendInterest` method in the picture below.
>
> Also notice how i use the `SendInterest` method in the `StartApplication` method to send the interest packet repeatedly with a constant delay.



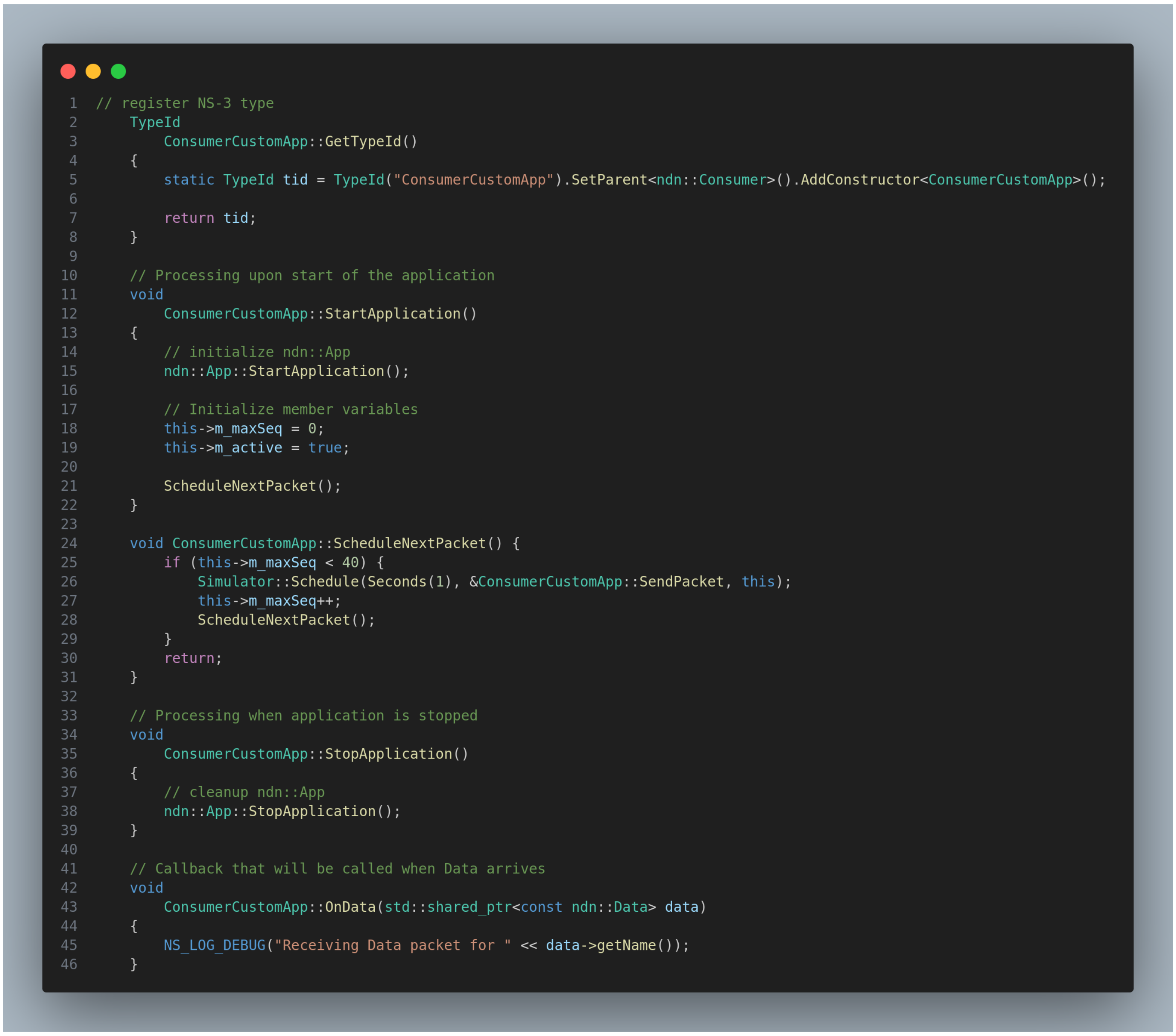
```cpp
// Processing upon start of the application
void
  CustomApp::StartApplication()
{
  // initialize ndn::App
  ndn::App::StartApplication();

  // Add entry to FIB for `/prefix/share`
  ndn::FibHelper::AddRoute(GetNode(), "/mpp/share", m_face, 0);

  // Schedule send of first interest
  int c = 10;
  for (int i = 0;i < c;i++) {
    Simulator::Schedule(Seconds(1.0), &CustomApp::SendInterest, this);
  }
}

/**
 * This method is created by us for sending the interest
 * packets manullay to the producer without use of any
 * existing application.
*/
void
  CustomApp::SendInterest()
{
  // Create and configure ndn::Interest
  auto interest = std::make_shared<ndn::Interest>("ndn:/mpp/share/");
  Ptr<UniformRandomVariable> rand = CreateObject<UniformRandomVariable>();
  interest->setNonce(rand->GetValue(0, std::numeric_limits<uint32_t>::max()));
  interest->setInterestLifetime(ndn::time::seconds(1));

  NS_LOG_DEBUG("Sending Interest packet for " << *interest);

  // Call trace (for logging purposes)
  m_transmittedInterests(interest, this, m_face);

  m_appLink->onReceiveInterest(*interest);
}

// Callback that will be called when Data arrives
void
  CustomApp::OnData(std::shared_ptr<const ndn::Data> data)
{
  NS_LOG_DEBUG("Receiving Data packet for " << data->getName());

  std::cout << "DATA received for name " << data->getName() << std::endl;
}
```

Task 1

4. Now, We will create a class that derives from the `Consumer` base class.

Here we need not to create our own `SendInterest` method but rather we use the `SendPacket` method that is already defined in the `Consumer` base class.

5. This method will make use of the attributes defined in the `Consumer` base class to send the interest packet. So make sure that we set the parent of the class to be consumer class so that the attributes defined in the consumer class are available to the class that we are creating.



```
// register NS-3 type
    TypeId
        ConsumerCustomApp::GetTypeId()
    {
        static TypeId tid = TypeId("ConsumerCustomApp").SetParent<ndn::Consumer>().AddConstructor<ConsumerCustomApp>();

        return tid;
    }

    // Processing upon start of the application
    void
        ConsumerCustomApp::StartApplication()
    {
        // initialize ndn::App
        ndn::App::StartApplication();

        // Initialize member variables
        this->m_maxSeq = 0;
        this->m_active = true;

        ScheduleNextPacket();
    }

    void ConsumerCustomApp::ScheduleNextPacket() {
        if (this->m_maxSeq < 40) {
            Simulator::Schedule(Seconds(1), &ConsumerCustomApp::SendPacket, this);
            this->m_maxSeq++;
            ScheduleNextPacket();
        }
        return;
    }

    // Processing when application is stopped
    void
        ConsumerCustomApp::StopApplication()
    {
        // cleanup ndn::App
        ndn::App::StopApplication();
    }

    // Callback that will be called when Data arrives
    void
        ConsumerCustomApp::OnData(std::shared_ptr<const ndn::Data> data)
    {
        NS_LOG_DEBUG("Receiving Data packet for " << data->getName());
    }
```

In the `SendInterest` method, i create my own interest packet and send to the forwarder using the applink service.

`ScheduleNextPacket` method is used to schedule the next interest packet to be sent to the forwarder.

6. For, producer class we will do the same thing as we did for the consumer class. We will create a class that derives from the `Producer` base class or from base `App` class. And depending on the context, either create our own `OnInterest` method or use the `OnInterest` method that is already defined in the `Producer` base class.

> Note: The `OnInterest` method is used because there is no special method to send the data packet. The data packet is sent as a response to the interest packet that is received.