Разработка адаптеров на платформе .NET Framework/.NET Core

В данном документы описаны шаги для построения веб-сервисов, взаимодействующие через платформу X-Road версии 6.

Чтобы облегчить материал вам предоставляется решение, исходный код которого расположен по следующему адресу <https://github.com/tunduk-gov-kg/dotnet-adapters-demo>.

Решение разбито на следующие проекты:

* *Consumer.XRoad\_End*
* *Consumer.Xroad\_Start*
* *Producer.Data*
* *Producer.Service*
* *Producer.Xroad\_End*
* *Producer.Xroad\_Start*

Краткое описание проектов:

*Producer.Data* – проект .NET Core Library содержит доменные сущности и прилагаемые к ним конфигурации для взаимодействия с базой данных через ORM Entity Framework Core. Папка SQL\_Scripts содержит скрипт для создания таблиц в СУБД Postgres и тестовые данные.

*Producer.Service* – слой бизнес логики также является проектом .NET Core Library и содержит сервисы которые служат в качестве связующего звена между слоем представления и слоем данных.

*Хочется отметить, что эти два проекта в ни в коем образе не влияют на инфраструктуру Xroad.*

*Producer.Xroad\_End* – веб-сервис на Asp.NET Core версии 2.2, готовая реализация Producer – ДЕМО адаптера.

*Producer.Xroad\_Start* – это пустой шаблон проекта на Asp.NET Core 2.2 и именно с него мы и начнем обучение разработке адаптеров типа Producer.

*Consumer.XRoad\_End* – WinForms проект содержит готовую реализацию ДЕМО адаптера типа Consumer.

*Consumer.Xroad\_Start* – пустой шаблон проекта WinForms для разработки Consumer.

В конечном итоге у нас должна появиться две системы, взаимодействующие через систему X-Road. Первая система будет запрашивать данные о человеке по ПИН и выводить эти данные на форму. Вторая система будет являться источником этих данных и будет по запросу выдавать их.

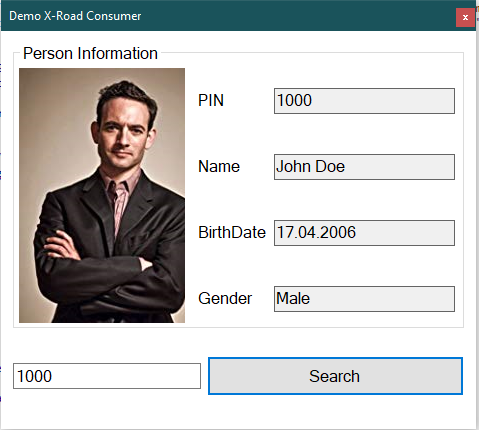


Рисунок 1 Программа для отображения данных о человеке

# Разработка Producer

Разработка провайдера услуг или так называемого Producer мы будем вести на платформе .NET Core 2.\*

*Выбор платформы неограничен, потому что протокол SOAP описывает лишь спецификации к реализуемым веб-сервисам с помощью языка описания веб-сервисов WSDL. Как мы знаем существует две версии протокола SOAP, но Xroad реализует только версию 1.1. Если у вас уже есть RESTful веб-сервисы с выходом системы X-Road версии 6.21 ожидается поддержка REST и OpenAPI спецификация.*

И так нам предстоит решить следующую задачу:

На сервере располагается база данных с информацией о людях. БД содержит таблицу Person с колонками:

* PIN – персональный идентификационный номер
* Name – ФИО человека
* Gender – пол человека
* BirthDate – дата рождения человека
* Photo – фотография человека

**Требуется реализовать SOAP сервис называемый GetPerson который будет выдавать информацию о человеке по входящему пин и зарегистрировать этот сервис в системе X-Road;**

Для разработки Producer вам предоставлен тестовый проект называемый Producer.Xroad\_Start. Предупреждение: В этом материале не рассматривается обучение технологии ASP.NET core, EF Core

Для начала вам потребуется подключить библиотеку XRoadLib.AspNetCore версии 1.3.7 из Nuget репозитории. Чтобы убедиться, что зависимость добавлена в нужный проект вам нужно раскрыть вкладку Зависимости/Nuget и там в списке вы увидите XRoadLib.Extensions.AspNetCore.

И начнём мы с создания папки DataContract где будут размещаться типы для сериализации в формат XML и для десериализации типов во входящих запросах. Эти типы будут далее описываться в XML схеме при генерации WSDL файла.

Для задания имени XML типа и пространства имен создаваемого класса его нужно пометить стандартным атрибутом XmlType и в качестве параметров указать TypeName = имя типа, Namespace = пространство имен.

Но вы можете опустить эти аргументы и просто указать что данный класс просто является XmlType, и будет использоваться при пересылке сообщений.

Далее вам следует унаследоваться от абстрактного класса XRoadSerializable иначе данный тип будет игнорироваться при генерации XSD схемы

Пример:

[XmlType(TypeName = "Person", Namespace = "http://producer.xroad.com")]

public class PersonModel : XRoadSerializable

Генерирует следующий тип XML который будет находиться в пространстве имен http://producer.xroad.com

<xsd:complexType name="Person">

Атрибут XRoadXmlElement наследует атрибут XmlElement и добавляет новые свойства такие как IsOptional и UseXop

Параметр IsOptional указывает что поле является опциональными и при сериализации поле со значением NULL не включается в XML.

Параметр UseXop указывает следует ли включать поле типа Stream в тело сообщения в виде закодированной base64 строки либо выносить в отдельную часть запроса таким образом пометив запрос как multipart/related; по умолчанию значение True; (.net core wcf client не поддерживает возможность обработки multipart/related таким образом если вы пометите его как false то клиент может обработать attachments как base64 строку)

Пример:

[XRoadXmlElement(ElementName = "Photo", IsOptional = false, Order = 4)]

public Stream Photo { get; set; }

Создается обязательное поле Photo в типе Person идущий 4ым по очереди.

А вот так выглядит xml схема

<xsd:element xmime:expectedContentTypes="application/octet-stream" name="Photo" type="xsd:base64Binary"/>

Как вы заметили добавился атрибут expectedContentTypes со значением application/octet-stream это указывает что это поле будет содержать двоичные данные закодированные в base64.

И так создадим в папке DataContract два типа:

[XmlType(TypeName = "Gender")]

public enum GenderEnum

{

Male = 1,

Female = 0

}

[XmlType(TypeName = "Person", Namespace = "http://sample.com")]

public class PersonModel : XRoadSerializable

{

[XRoadXmlElement(ElementName = "Pin", IsOptional = false, Order = 0)]

public string Pin { get; set; }

[XRoadXmlElement(ElementName = "Name", IsOptional = false, Order = 1)]

public string Name { get; set; }

[XRoadXmlElement(ElementName = "BirthDate", IsOptional = false, Order = 2)]

public DateTime BirthDate { get; set; }

[XRoadXmlElement(ElementName = "Gender", IsOptional = false, Order = 3)]

public GenderEnum Gender { get; set; }

[XRoadXmlElement(ElementName = "Photo", IsOptional = false, Order = 4)]

public Stream Photo { get; set; }

}

Далее создаем папку ServiceContract и создаем интерфейс:

public interface IPersonSoapService

{

[XRoadService("GetPerson", AddedInVersion = 1)]

[XRoadTitle("RU","Сервис для выдачи персональных данных по пин")]

PersonModel GetPerson(string pin);

}

Атрибут XRoadService обязателен. И в нем вы указываете название вашего сервиса и версию если вы предполагаете, что она будет меняться со временем.

Атрибут XRoadTitle не обязателен, он служит для добавления информации для разработчиков. При добавлении такого атрибута в wsdl появится следующий тег с внутренними полями

<wsdl:documentation>

<xrd:title xml:lang="RU">

Сервис для выдачи персональных данных по пин

</xrd:title>

</wsdl:documentation>

В целом wsdl будет выглядет следующим образом

<wsdl:portType name="PortTypeName">

<wsdl:operation name="GetPerson">

<wsdl:documentation>

<xrd:title xml:lang="RU">Сервис для выдачи персональных данных по пин</xrd:title>

</wsdl:documentation>

<wsdl:input message="tns:GetPerson"/>

<wsdl:output message="tns:GetPersonResponse"/>

</wsdl:operation>

</wsdl:portType>

# Разработка Consumer