Workflow Services

Estimated time for completion: 45 minutes

Overview:

In this lab you will be creating a service that processes loan applications. On application the service will return the monthly payments and an application id. The customer then assesses the payment and confirms the loan application. However, if the loan is over a certain value then the workflow will request confirmation from another service. This will take place while the customer is considering their quote.

Goals:

- Learn how to expose a workflow as a service
- Learn how to invoke a service from a workflow
- Understand data orientated correlation

Lab Notes:

Part 1: Creating the Initial Service

In the first part of the lab you will create an version of the service to take the loan application and return an application id and repayment amout.

1. In Visual Studio, open the starter solution **before/WFServices.sln**
2. Add a new a New Project to the solution of type WCF Workflow Service Application (this is in the workflow section of the new project dialog) called `RequestLoan`
3. Add a public class to the project called `ApplyResponse`. This will be used to model the service response message.
    a. Create an integer property called `ApplicationId`
    b. Create a double property called `MonthlyRepayment`
    c. Annotate the class with the `DataContract` attribute and the properties with the `DataMember` attribute

```
[DataContract]
public class ApplyResponse
{
    [DataMember]
    public int ApplicationId { get; set; }
    [DataMember]
    public double MonthlyRepayment { get; set; }
}
```

4. Add an interface called `ILoanService`
    a. Add an `Apply` method that returns an `ApplyResponse` and takes an `amount` of type `double` and a `term` of type `int`
    b. Annotate the interface with a `ServiceContract` attribute and set the `Namespace` to be `"http://develop.com/"`
    c. Annotate the `Apply` method with the `OperationContract` attribute
5. Compile the project
6. Right click on the `RequestLoan` project and select `Import Service Contract`. Navigate to the `ILoanService` contract and press `OK`
7. Once the import has taken place recompile the project (this will generate activities for the contract operations)
8. Open the `Service1.xamlx` file and delete the Sequential Service activity
9. Go to the tool box and drop a `Apply_ReceiveAndSendReply` activity (from the `ILoanService` section) on to the design surface
10. Select the `Apply_Receive` and in the property window check the `CanCreateInstance` checkbox. This allows new instances of the workflow to be spun up when a message is received
11. Now add the following variables to the sequential service activitity
    a. `loanAmount` : `double`
    b. `loanTerm` : `Int32`
    c. `rnd` : `Random` – initialize this to `new Random()`
    d. `theResponse` : `ApplyResponse` – initialize this to `new ApplyResponse()`
12. Open the `Apply_Receive`'s Parameters dialog and map the incoming `loanAmount` to the `loanAmount` variable and the incoming `term` to the `loanTerm` variable
13. Now we need to generate the data for the response message. Drop an `Assign` between the request and response activities
    a. Set the `To` to `theResponse.ApplicationId`
    b. Set the `From` to `rnd.Next(1000)`. This sets the application id to a random value so we don't need to try to keep globally unique application ids in the lab (obviously this would never work in production)
14. Drop another `Assign` directly after the first
    a. Set the `To` to `theResponse.MonthlyRepayment`
    b. Set the `From` to `loanAmount / loanTerm * 2.0` (this sets the company to make a fairly extortionate amount of interest)
15. Now select the `Apply_SendReply` activity and open it's parameters
    a. Set the `ApplyResult parameter to applyResponse`
16. Compile and run this project. This should start IIS Express and a web browser (although the browser may say that directory browsing isn't allowed). Append Service1.xamlx?wsdl to the address displayed in the browser and you should see the WSDL for the service displayed.

Part 2: Creating a client for the service

In this part of the lab you will create a client application that uses the loan service.

1. Add a new Console Application project, called Client, to the solution

2. Ensure that IIS Express is running and in the `Client` project add a service reference that points to the loan service (this will be the URL of the WSDL document you viewed in part 1 of the lab)
3. In `Main` create an instance of generated `LoanServiceClient` proxy class
4. Call its `Apply` method passing the input parameters. Capture the response from the `Apply` operation
5. Write out, to the Console, the `ApplicationId` and `MonthlyRepayment` that were returned by the `Apply` operation

```
LoanServiceClient proxy = new LoanServiceClient();

ApplyResponse resp = proxy.Apply(new Apply { loanAmount = 4000, term = 60 });

Console.WriteLine("{0} : {1}", resp.ApplicationId, resp.MonthlyRepayment);
```

6. Compile and test your client

Part 3: Call Another Service to Request Loan Approval

In the 3rd part of the lab you will examine the requested loan amount and if it is over a £5000 then you will call another service to seek approval for the loan (loans under £5000 are automatically accepted). An ApproverService is already supplied for you to use

1. In the solution you will notice a project called `ApproverService`. This is an external service that will look at the loan and decide whether to approve it. To run this project you will need to ensure that the HTTP infrastructure on your machine will allow you to listen on port 9000. To do this you may need to run the following command at a Admin command prompt

```
NETSH HTTP ADD URLACL URL=http://+:9000/ USER=<Your user ID>
```

2. Now run the project - you will need to run this outside of the debugger (use CTRL-F5) and you should see the service start to listen successfully
3. In the `RequestLoan` project add a service reference to the `ApproverService` using the following address: **`http://localhost:9000/approval?wsdl`**. This will generate a new activity in your toolbox called `RequestApproval`
4. Open **Service1.xamlx** and add a new `Boolean` variable called `approverDecision`
5. After the `SendResponse` activity drop an `If` activity on to the design surface. We will use this to check the loan amount to assess whether the request can be auto-approved or whether we must request approval from the `ApproverService`
6. Select the `If` activity and set its `Condition` to `loanAmount > 5000`
7. Drop an `Assign` activity in the `Else` part of the `If` and set the `approverDecision` to `true`. This auto-approves the loan if it is 5000 or less
8. In the `Then` block of the `If` activity drop a `RequestApproval` activity (the one that was generated when you created the service reference)

Error! Reference source not found.

9. In the properties of the `RequestApproval` activity set the `amount` parameter to the `loanAmount` variable and the `RequestApprovalResult` to the `approverDecision` variable
10. Compile and run the project
11. Rerun your `Client` with an amount over 5000 in the request amount. You should see that the `ApproverService` is invoked

---

Part 4: Retrieving the Approval Decision

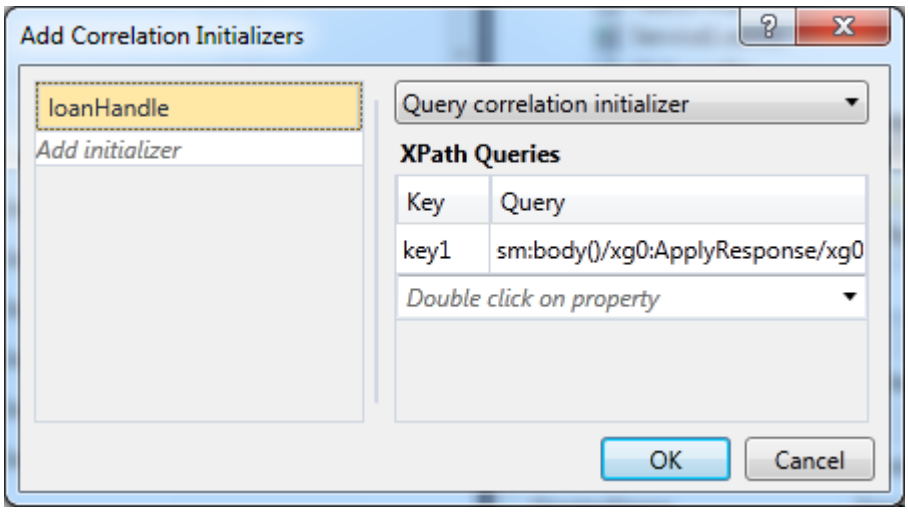
You will notice from the workflow structure that approval is decided asynchronously after the client has been informed of the repayment details. In the last part of the lab you will change the RequestLoan service to allow the client to find out what the decision was about their loan application.

1. Amend the ILoanService contract to add a Confirm operation that takes an applicationId of type int and returns a boolean for the loan decision

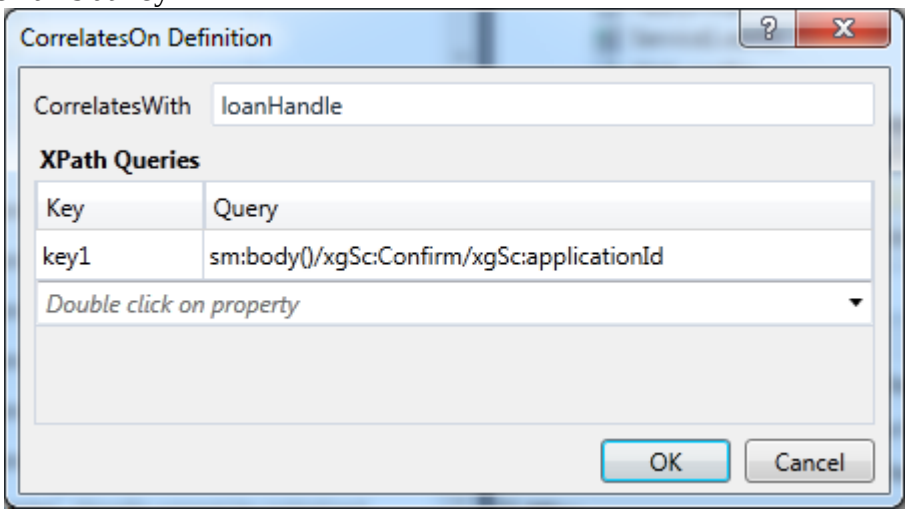
```
[ServiceContract(Namespace = "http://develop.com/")]
public interface ILoanService
{
   [OperationContract]
   ApplyResponse Apply(double amount, int term);

   [OperationContract]
   bool Confirm(int applicationId);
}
```

2. Compile and then right click on the ILoanService contract in the Service Contracts folder and select Update Service Contract.
3. Open Service1.xamlx. On recompilation a new activity should appear in your toolbox for the new operation
4. After the `If` activity drop a `Confirm_ReceiveAndSendReply` activity on to the designer
5. In the `Confirm_SendReply` activity map the `ConfirmResult` to the `approverDecision` variable
6. If you look at the variables in scope in the Confirm_ReceiveAndSendReply activity you will see a name collision for _handle. This is because the original request/reply activity's correlation handle is in scope as well. Add a **new** sequence at the start of the workflow and drag all of the activities for the initial receive/reply (including the two assigns) into this sequence. Then change the scope of this _handle to the new sequence. This should resolve the issue
7. You now need to set up correlation such that the `Confirm` message is routed to the correct instance of the loan application workflow
    a. At the top level sequence level create a new variable of type CorrelationHandle called loanHandle
    b. You will perform correlation on the applicationId. Go to the Apply_SendReply associated with the Apply operation and click on the correlationInitializers property to bring up the dialog

c. Add an initializer, set the handle to loanHandle, ensure the Correlation Type is Query Correlation Initializer and select the applicationId from the drop down in the XPath Query. Leave the key name as key1



8. Go to the Confirm operation receive and set CorrelatesWith to loanHandle
9. Open the Correlates On dialog and set the data to the incoming applicationId. Ensure that the key name remains at key1



10. Compile and run the service
11. Go to the client and update the service reference to the RequestLoan service
12. Now, in Main, add a call to the Confirm operation, passing the application id returned from the Apply operation
13. Print out whether the loan was agreed or rejected
14. Compile and test your client

Solutions

after\WFServices.sln