# JADE .NET Exposures

Version 2018

# Contents

# JADE .NET Exposures

## Introduction

JADE provides for integration with Microsoft .NET systems through use of the **JoobContext** object.

In this module, the Banking system that is built in the JADE Developer's course is exposed to .NET, allowing for a WPF front-end application to display and add to the JADE back-end database.

As the Banking system was designed with a separation between the model and the view, the view can be substituted with a completely different view without the need for any change to the model code.

## DotNetConnection Application

For Microsoft .NET to use JADE classes, it maintains a reference to a **JoobContext** object that is generated by JADE and it provides an interface to the JADE classes by delegating requests to the appropriate JADE classes.

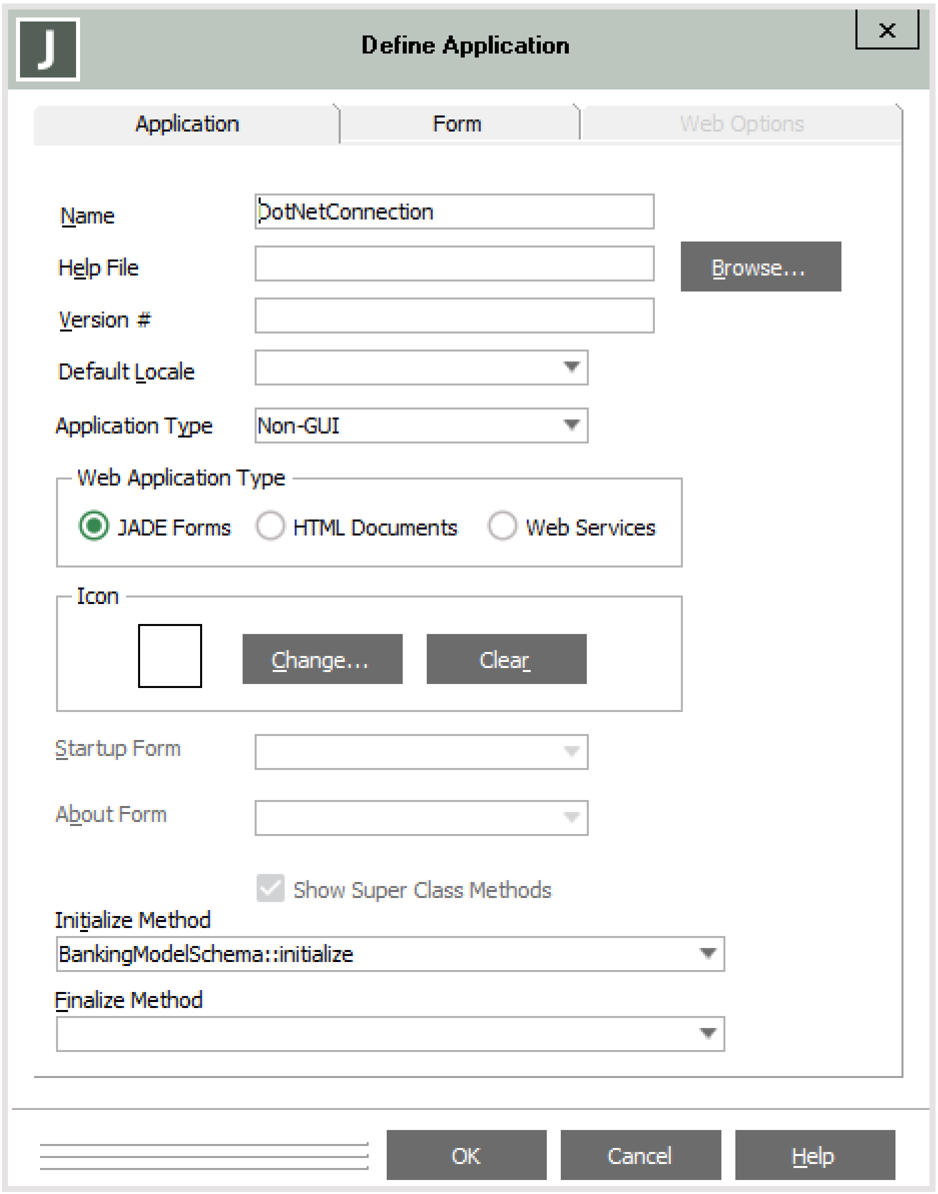
As the **JoobContext** object needs to have a dedicated process to be able to read properties or run methods of JADE classes, we use a non-GUI application called **DotNetConnection** to generate this process.

### Exercise 1 – Creating a DotNetConnection Application

In this exercise, you will create the **DotNetConnection** application that the **JoobContext** object will use to delegate requests from .NET to JADE.

1. Select **BankingViewSchema** in the Schema Browser and then open the Application Browser by pressing Ctrl+L or clicking on the **Browse Applications** icon in the toolbar.

2. Select the **Add** command from the Application menu and then fill out the **Application** sheet of the Define Application dialog as follows.



Name: DotNetConnection
Help File:
Version #:
Default Locale:
Application Type: Non-GUI
Web Application Type: [✓ JADE Forms] HTML Documents Web Services
Icon
Startup Form:
About Form:
✓ Show Super Class Methods
Initialize Method: BankingModelSchema::initialize
Finalize Method:

a. Enter **DotNetConnection** in the Name text box.

b. Select **Non-GUI** in the **Application Type** drop-down list box.

c. Check the **Show Super Class Methods** check box, so that you can set the initialize method to **BankingModelSchema::initialize** in the **Initialize Method** combo box.

d. Click **OK**.

## JADE Exposures

Rather than letting external applications have full access to JADE databases, JADE defines an exposure, which contains only the properties and methods that the specific external application requires.

The advantage of this is twofold, as it:

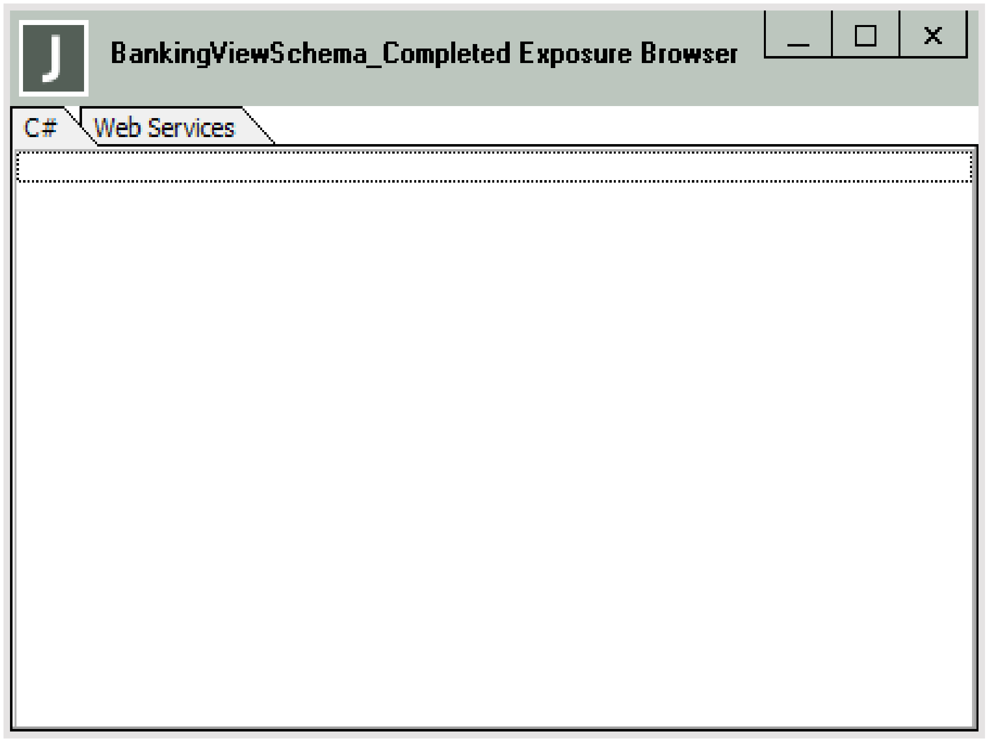
- Provides a more-simple interface to the JADE database
- Allows for greater control over which JADE properties and methods external applications can access

## Exercise 2 – Using the JADE Exposure Wizard

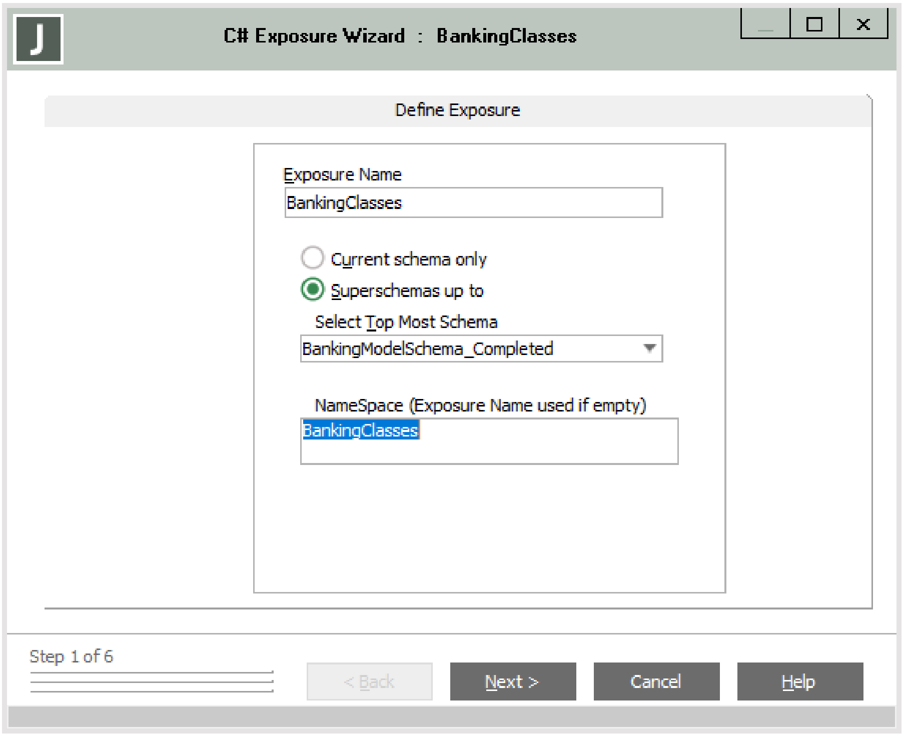
In this exercise, you will define the exposure that the .NET application will access.

1. Select **BankingViewSchema** in the Schema Browser, and then select the **Exposures** command from the Browse menu.

   Right-click on the **Exposure Browser** and then select the **Add** command, to create a new C# exposure.



2. Name the exposure **BankingClasses** and then select **BankingModelSchema** option in the **Superschemas up to** drop-down list box.


C# Exposure Wizard : BankingClasses

Define Exposure

Exposure Name: BankingClasses

( ) Current schema only
(●) Superschemas up to

Select Top Most Schema: BankingModelSchema_Completed

NameSpace (Exposure Name used if empty): BankingClasses

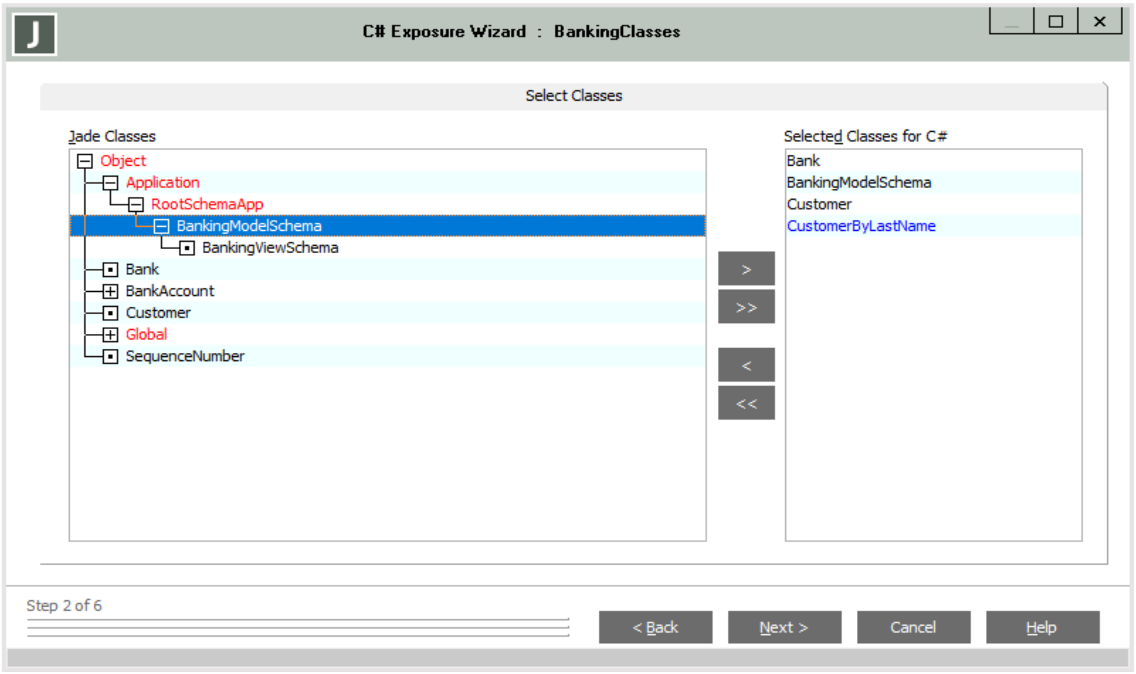
3. Select the following classes.

   a. **Bank**

   b. **BankingModelSchema** (a subclass of the **RootSchemaApp** application class)

   c. **Customer**.

   These will be the classes that are accessible from .NET.


C# Exposure Wizard : BankingClasses

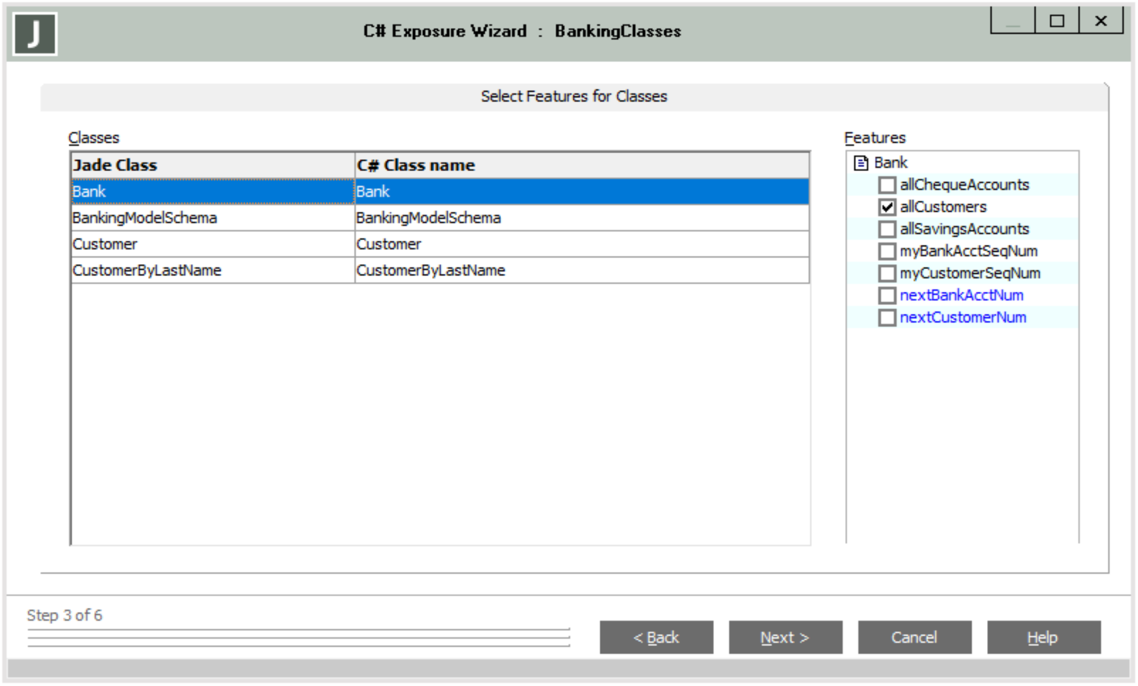
Select Classes

Jade Classes:
```
Object
  Application
    RootSchemaApp
      BankingModelSchema
        BankingViewSchema
  Bank
  BankAccount
  Customer
  Global
  SequenceNumber
```

Selected Classes for C#:
```
Bank
BankingModelSchema
Customer
CustomerByLastName
```

**Note** Selecting only the classes that you intend to use provides a simpler interface to JADE. You can always add additional classes to the exposure later, if required.

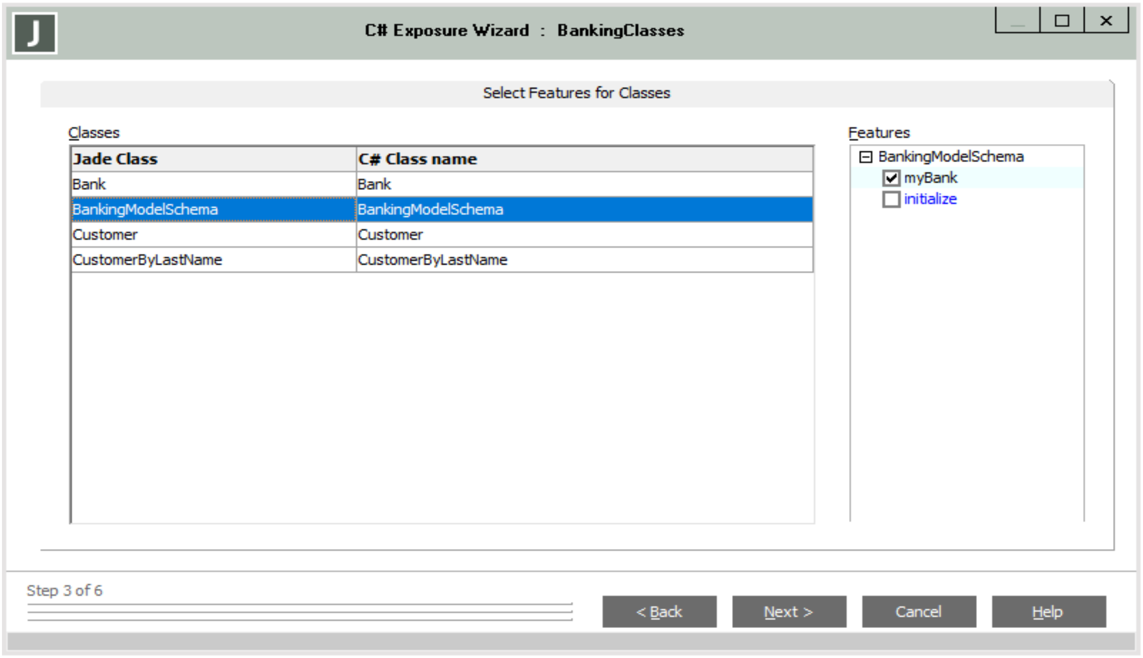
4. Select the **Bank** class and then select (check) the **allCustomers** property from the features on the right of the sheet. This exposes the **allCustomers** collection to .NET; the other methods and properties are *not* exposed.


C# Exposure Wizard : BankingClasses
Select Features for Classes
Classes
Features
Step 3 of 6
< Back
Next >
Cancel
Help

| Jade Class | C# Class name |
|---|---|
| Bank | Bank |
| BankingModelSchema | BankingModelSchema |
| Customer | Customer |
| CustomerByLastName | CustomerByLastName |

Bank
- [ ] allChequeAccounts
- [✓] allCustomers
- [ ] allSavingsAccounts
- [ ] myBankAcctSeqNum
- [ ] myCustomerSeqNum
- [ ] nextBankAcctNum
- [ ] nextCustomerNum

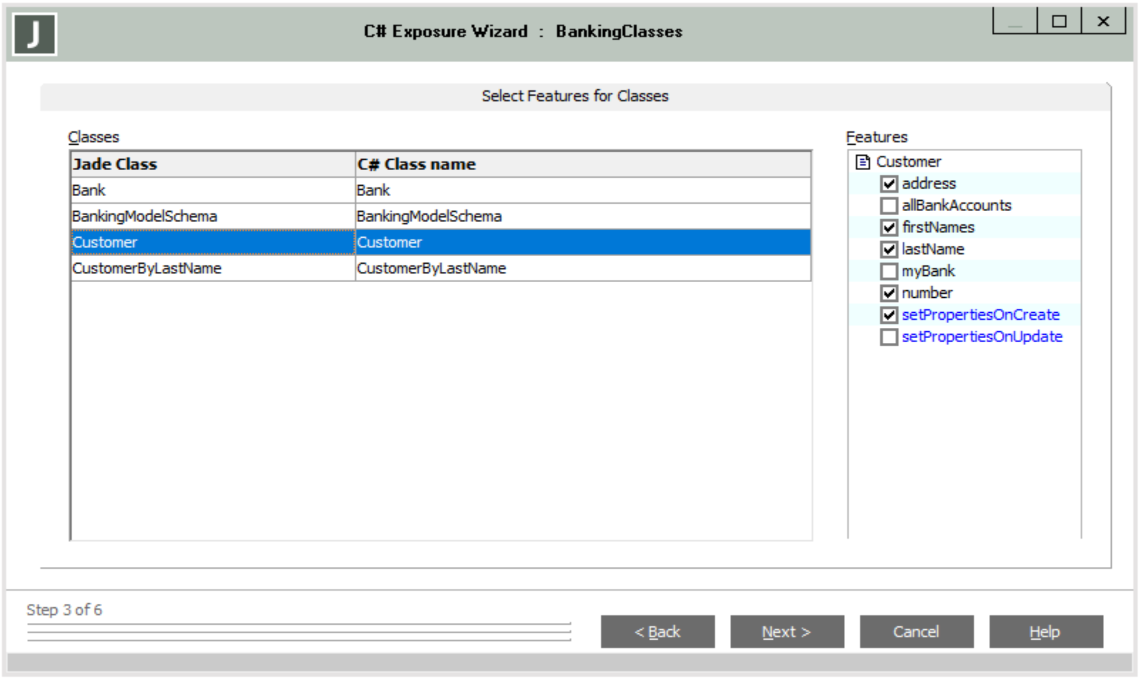
5. In the **BankingModelSchema** class, check the **myBank** feature.


C# Exposure Wizard : BankingClasses
Select Features for Classes
Classes
Features
Step 3 of 6
< Back
Next >
Cancel
Help

| Jade Class | C# Class name |
|---|---|
| Bank | Bank |
| BankingModelSchema | BankingModelSchema |
| Customer | Customer |
| CustomerByLastName | CustomerByLastName |

BankingModelSchema
- [✓] myBank
- [ ] initialize

6. In the **Customer** class, check the **address**, **firstNames**, **lastName**, **number**, and **setPropertiesOnCreate** feature properties.


C# Exposure Wizard : BankingClasses
Select Features for Classes
Step 3 of 6
< Back
Next >
Cancel
Help

Classes

| Jade Class | C# Class name |
|---|---|
| Bank | Bank |
| BankingModelSchema | BankingModelSchema |
| Customer | Customer |
| CustomerByLastName | CustomerByLastName |

Features

- Customer
  - [✓] address
  - [ ] allBankAccounts
  - [✓] firstNames
  - [✓] lastName
  - [ ] myBank
  - [✓] number
  - [✓] setPropertiesOnCreate
  - [ ] setPropertiesOnUpdate

7. As the steps on sheets 4 and 5 of this wizard do not require any changes, click **Next** until you reach step 6 (that is, the **Generate** sheet).

   Fill out this sheet, as follows.

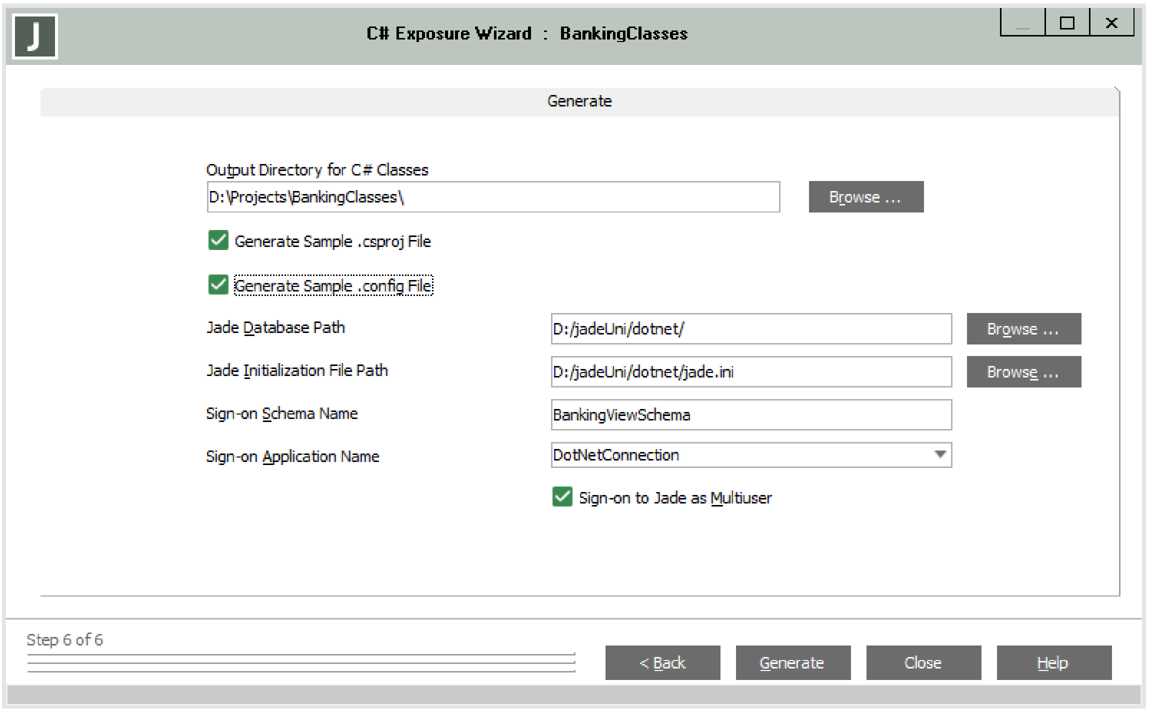
   a. Enter **C:\Projects\BankingClasses** in the **Output Directory** text box. This is the directory where the C# project file and class files will be created.

   b. Check the **Generate Sample .csproj File** and **Generate Sample .config File** check boxes.

   c. Enter the absolute path of the JADE database directory (including the **system** folder) in the **Jade Database Path** text box.

   d. Enter the location and file name of your JADE initialization file (which is probably called **jade.ini**) in the **Jade Initialization File Path** text box.

e. Check the **Sign-on to Jade as Multiuser** check box.



C# Exposure Wizard : BankingClasses

Generate

Output Directory for C# Classes: D:\Projects\BankingClasses\

✓ Generate Sample .csproj File

✓ Generate Sample .config File

Jade Database Path: D:/jadeUni/dotnet/

Jade Initialization File Path: D:/jadeUni/dotnet/jade.ini

Sign-on Schema Name: BankingViewSchema

Sign-on Application Name: DotNetConnection

✓ Sign-on to Jade as Multiuser

Step 6 of 6

8. Click **Generate**.

   The wizard does not close, but *Generation completed* should be displayed at the lower left of the wizard. You can now click **Close**.

## Dynamic Link Libraries (*.dll Files)

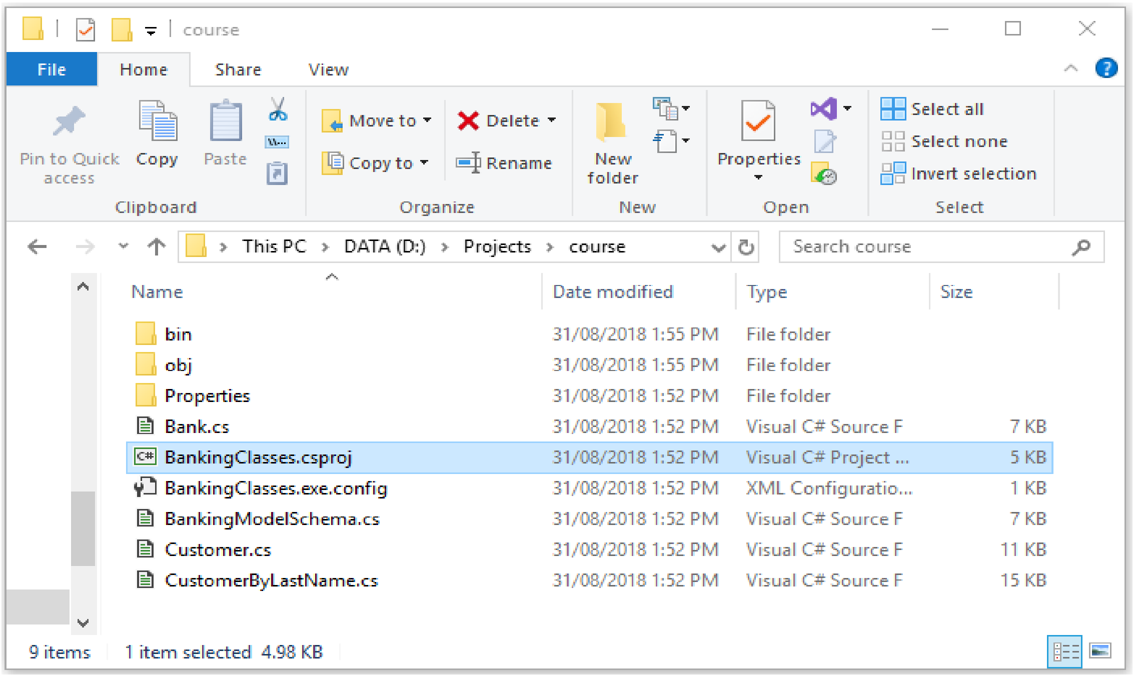
A Dynamic Link Library is an external method (or routine) written in any language that can create a Dynamic Link Library (DLL). DLL methods can then be executed as standard JADE methods from within JADE.

The purpose of creating DLLs is to allow different environments to integrate; for example, allowing an application written in C# to use JADE methods.

# Exercise 3 – Creating a DLL File

In this exercise, you will use the generated C# project to create the **BankingClasses.dll** file.

1. In the File Explorer, navigate to **C:\Projects\BankingClasses**.

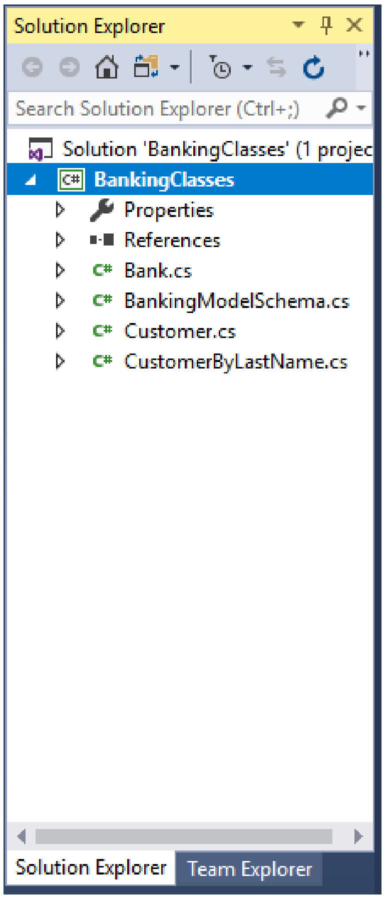


| Name | Date modified | Type | Size |
|---|---|---|---|
| bin | 31/08/2018 1:55 PM | File folder | |
| obj | 31/08/2018 1:55 PM | File folder | |
| Properties | 31/08/2018 1:52 PM | File folder | |
| Bank.cs | 31/08/2018 1:52 PM | Visual C# Source F | 7 KB |
| BankingClasses.csproj | 31/08/2018 1:52 PM | Visual C# Project ... | 5 KB |
| BankingClasses.exe.config | 31/08/2018 1:52 PM | XML Configuratio... | 1 KB |
| BankingModelSchema.cs | 31/08/2018 1:52 PM | Visual C# Source F | 7 KB |
| Customer.cs | 31/08/2018 1:52 PM | Visual C# Source F | 11 KB |
| CustomerByLastName.cs | 31/08/2018 1:52 PM | Visual C# Source F | 15 KB |

9 items    1 item selected 4.98 KB

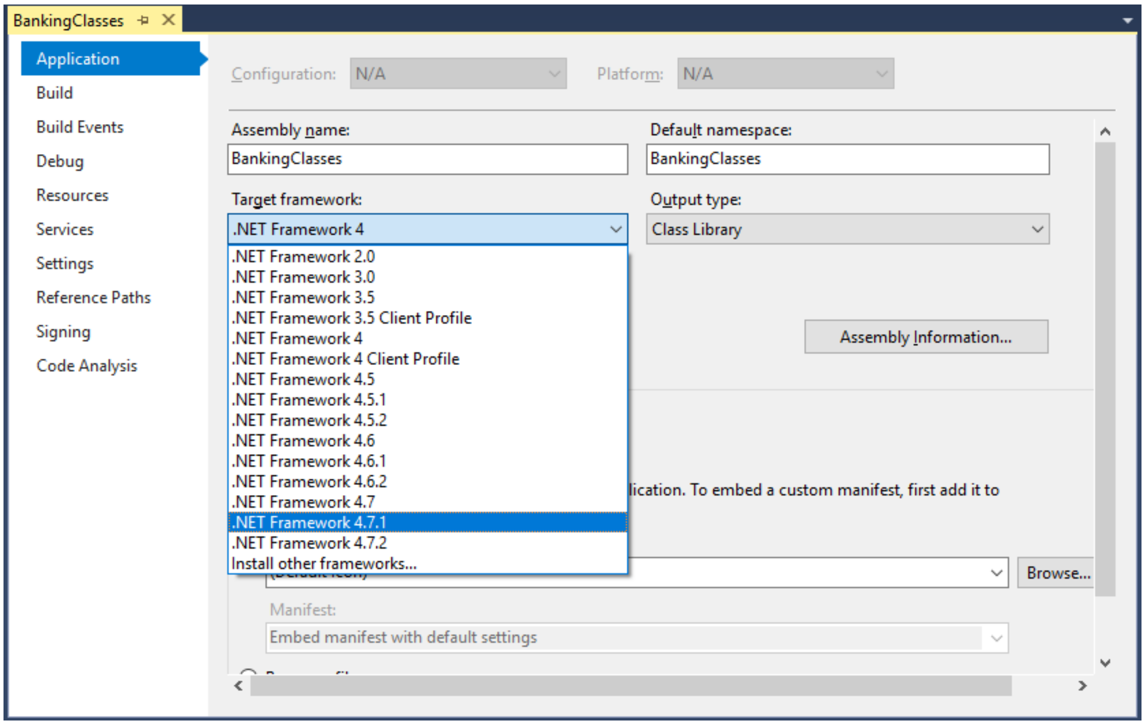
2. Open **BankingClasses.csproj** in Microsoft Visual Studio 2017.

Find the Solution Explorer and then select **BankingClasses** under Solution 'BankingClasses'.



3. Press Alt+Enter to open the Properties menu.

Change the Target framework to **.NET Framework 4.7.1**.



Assembly name: BankingClasses

Default namespace: BankingClasses

Target framework: .NET Framework 4

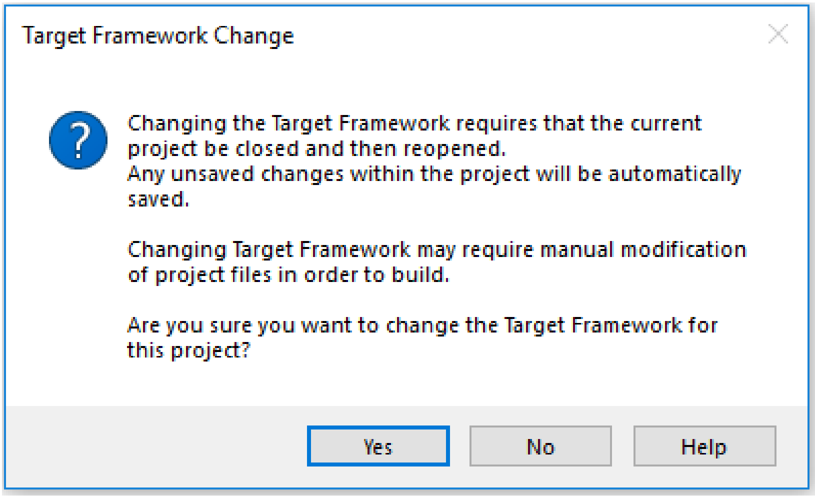
Output type: Class Library

.NET Framework 2.0
.NET Framework 3.0
.NET Framework 3.5
.NET Framework 3.5 Client Profile
.NET Framework 4
.NET Framework 4 Client Profile
.NET Framework 4.5
.NET Framework 4.5.1
.NET Framework 4.5.2
.NET Framework 4.6
.NET Framework 4.6.1
.NET Framework 4.6.2
.NET Framework 4.7
.NET Framework 4.7.1
.NET Framework 4.7.2
Install other frameworks...

lication. To embed a custom manifest, first add it to

Manifest: Embed manifest with default settings

4. In the following message box that is then displayed, click **Yes**.



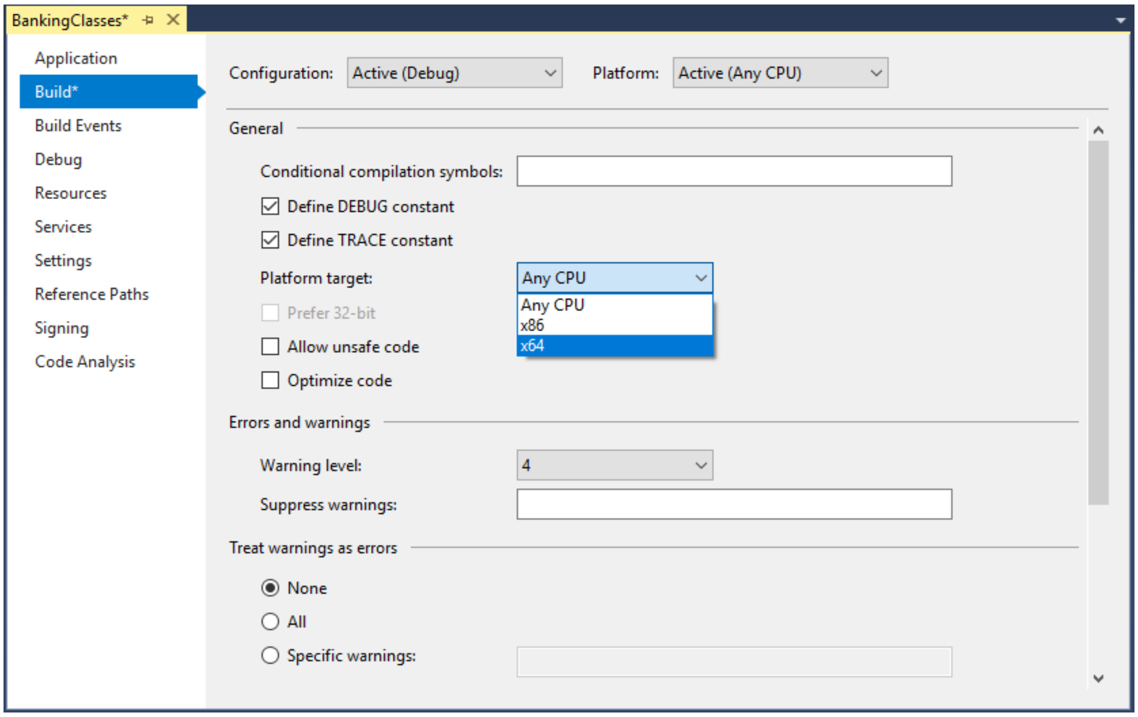
Target Framework Change

Changing the Target Framework requires that the current project be closed and then reopened.
Any unsaved changes within the project will be automatically saved.

Changing Target Framework may require manual modification of project files in order to build.

Are you sure you want to change the Target Framework for this project?

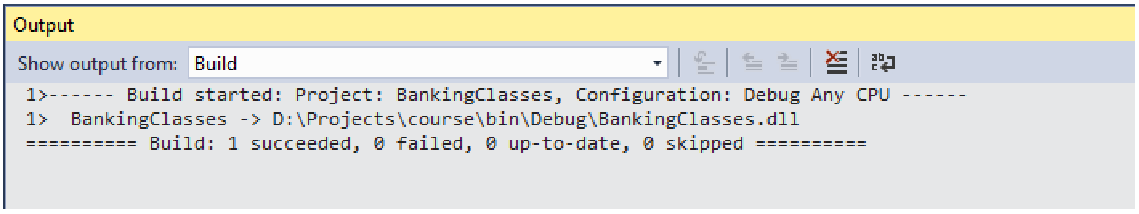
5. Click the **Build** tab, and then change the platform target from **Any CPU** to **x64**.



Configuration: Active (Debug)
Platform: Active (Any CPU)

Conditional compilation symbols:
[✓] Define DEBUG constant
[✓] Define TRACE constant
Platform target: Any CPU
[ ] Prefer 32-bit
[ ] Allow unsafe code
[ ] Optimize code
Warning level: 4
Suppress warnings:
(●) None
( ) All
( ) Specific warnings:

6. Build the project by pressing Ctrl+Shift+B. You should then see a message like the following displayed in the Output window.



```
1>------ Build started: Project: BankingClasses, Configuration: Debug Any CPU ------
1>  BankingClasses -> D:\Projects\course\bin\Debug\BankingClasses.dll
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

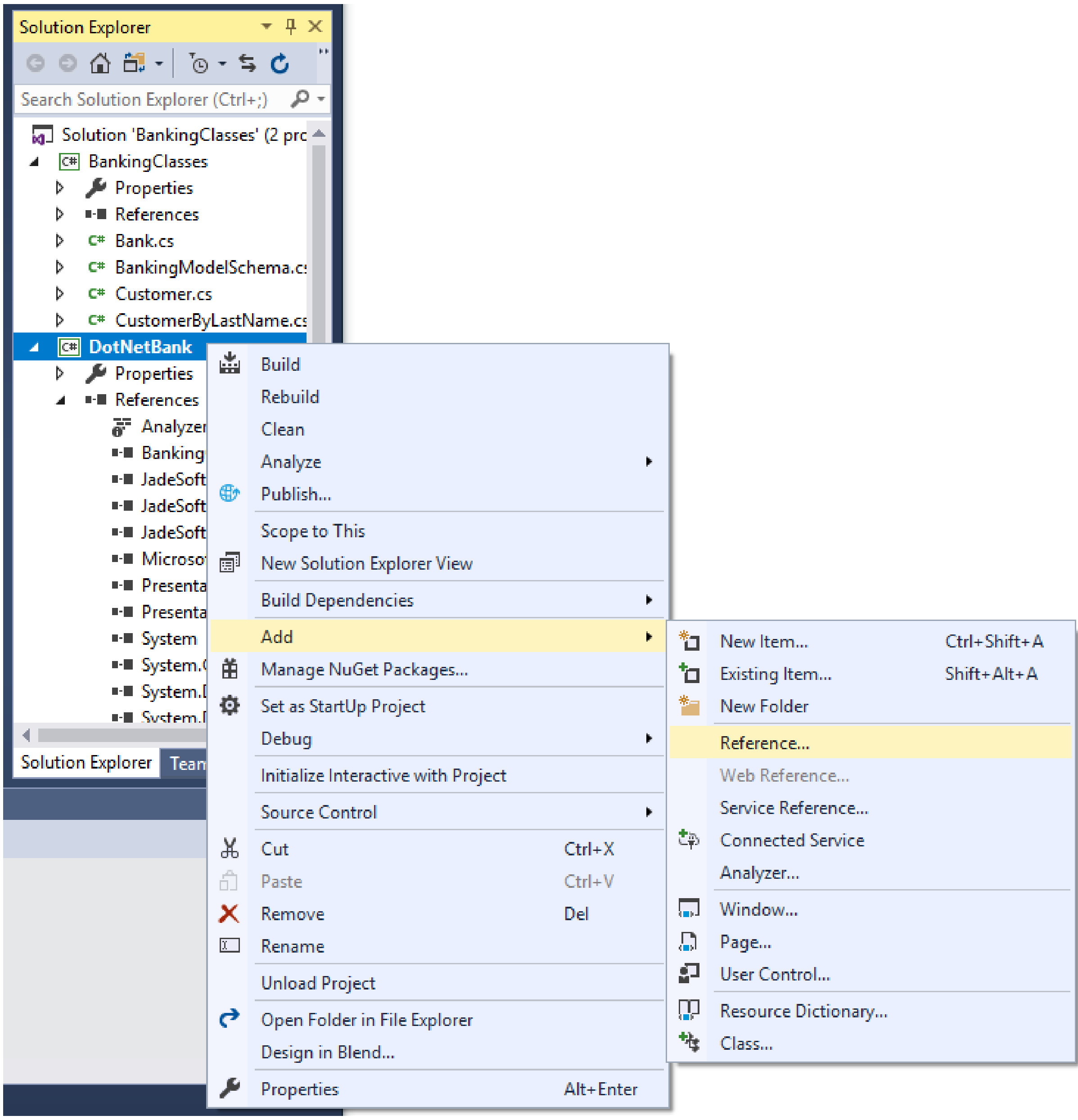
Building this project with the correct settings creates a **.dll** file for the JADE exposure defined in Exercise 2 of this module.

## Exercise 4 – Importing JADE DLL Files into .NET

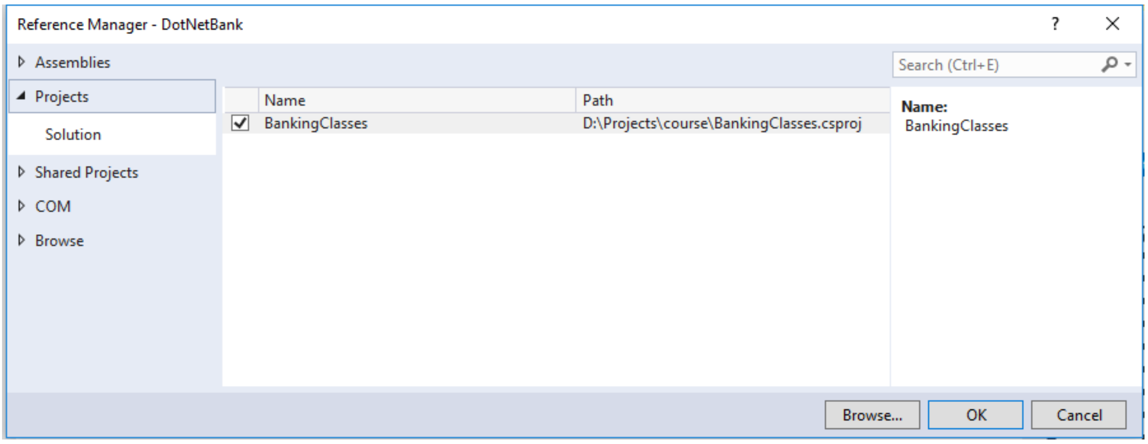
In this exercise, you will set the required DLL files as references in a .NET application.

1. Add a new project called **DotNetBank** to the Solution 'BankingClasses' in the Solution Explorer.

2. Add a new reference to the **DotNetBank** project by right-clicking on the **DotNetBank** project and then selecting the **Reference** command in the **Add** menu.



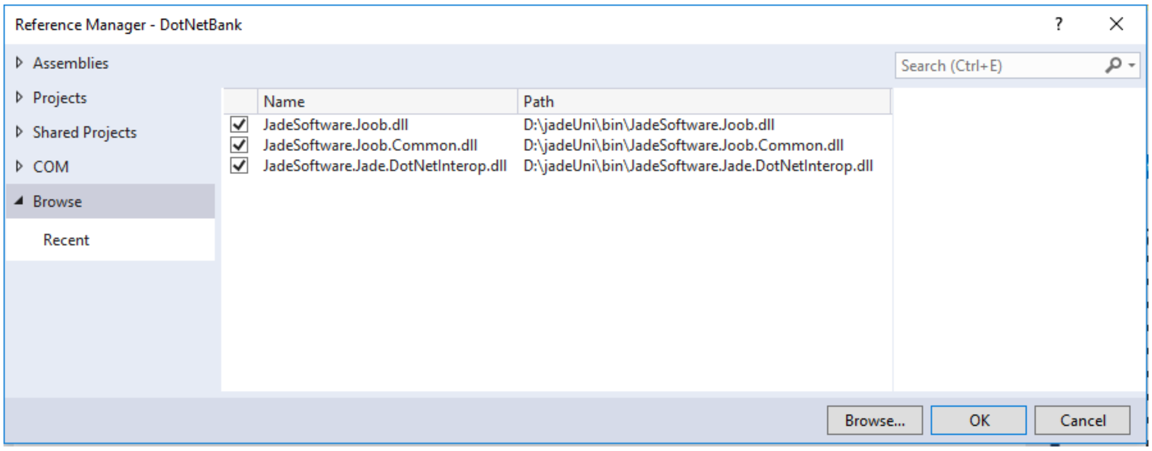
3. Check **BankingClasses** in the **Projects** tab.



| | Name | Path |
|---|---|---|
| ✓ | BankingClasses | D:\Projects\course\BankingClasses.csproj |

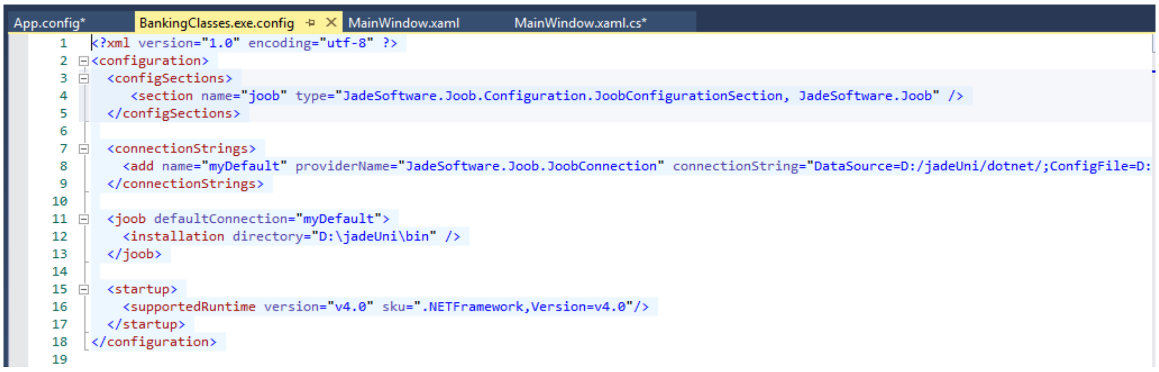
**Name:**
BankingClasses

4. In the **Browse** tab, click the **Browse** button and then add the following three references.



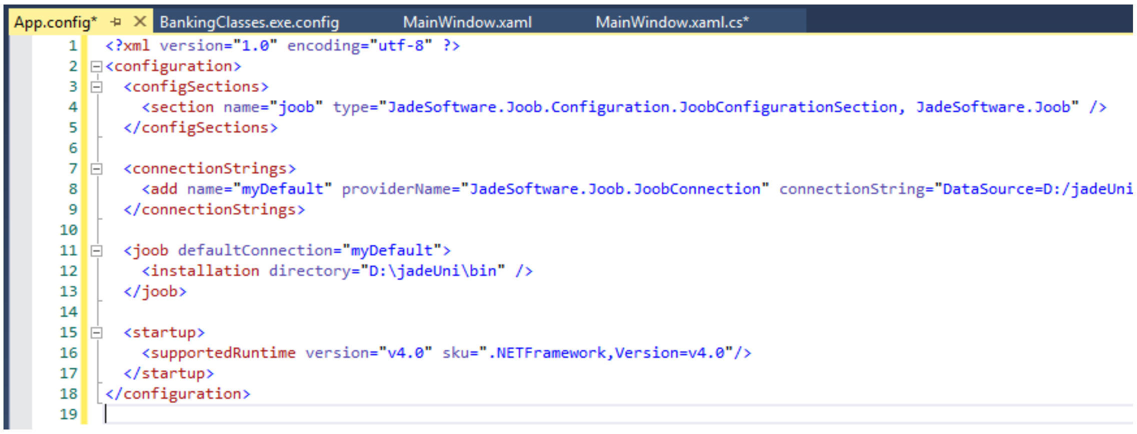
| | Name | Path |
|---|---|---|
| ✓ | JadeSoftware.Joob.dll | D:\jadeUni\bin\JadeSoftware.Joob.dll |
| ✓ | JadeSoftware.Joob.Common.dll | D:\jadeUni\bin\JadeSoftware.Joob.Common.dll |
| ✓ | JadeSoftware.Jade.DotNetInterop.dll | D:\jadeUni\bin\JadeSoftware.Jade.DotNetInterop.dll |

5. Navigate to **C:\Projects\BankingClasses** in the File Explorer and then open **BankingClasses.exe.config** in Visual Studio.



```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
     <section name="joob" type="JadeSoftware.Joob.Configuration.JoobConfigurationSection, JadeSoftware.Joob" />
  </configSections>

  <connectionStrings>
    <add name="myDefault" providerName="JadeSoftware.Joob.JoobConnection" connectionString="DataSource=D:/jadeUni/dotnet/;ConfigFile=D:
  </connectionStrings>

  <joob defaultConnection="myDefault">
    <installation directory="D:\jadeUni\bin" />
  </joob>

  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
  </startup>
</configuration>
```

6. Copy the contents of **BankingClasses.exe.config** and then paste it into **app.config**.



```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="joob" type="JadeSoftware.Joob.Configuration.JoobConfigurationSection, JadeSoftware.Joob" />
  </configSections>

  <connectionStrings>
    <add name="myDefault" providerName="JadeSoftware.Joob.JoobConnection" connectionString="DataSource=D:/jadeUni
  </connectionStrings>

  <joob defaultConnection="myDefault">
    <installation directory="D:\jadeUni\bin" />
  </joob>

  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
  </startup>
</configuration>
```

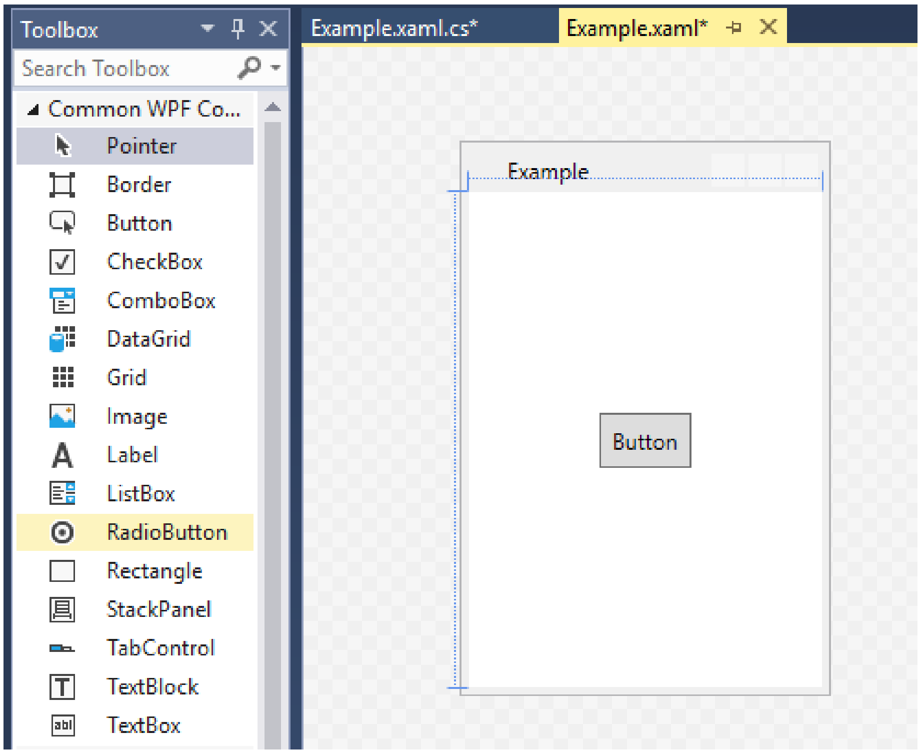
7. Ensure that the **Application** and **Build** settings are set to the same as those of the **BankingClasses** application; that is:

   - Target Framework is set to **.NET Framework 4.7.1** in the **Application** tab of the project properties.
   - Platform Target is set to **x64** in the **Build** tab of the project properties.

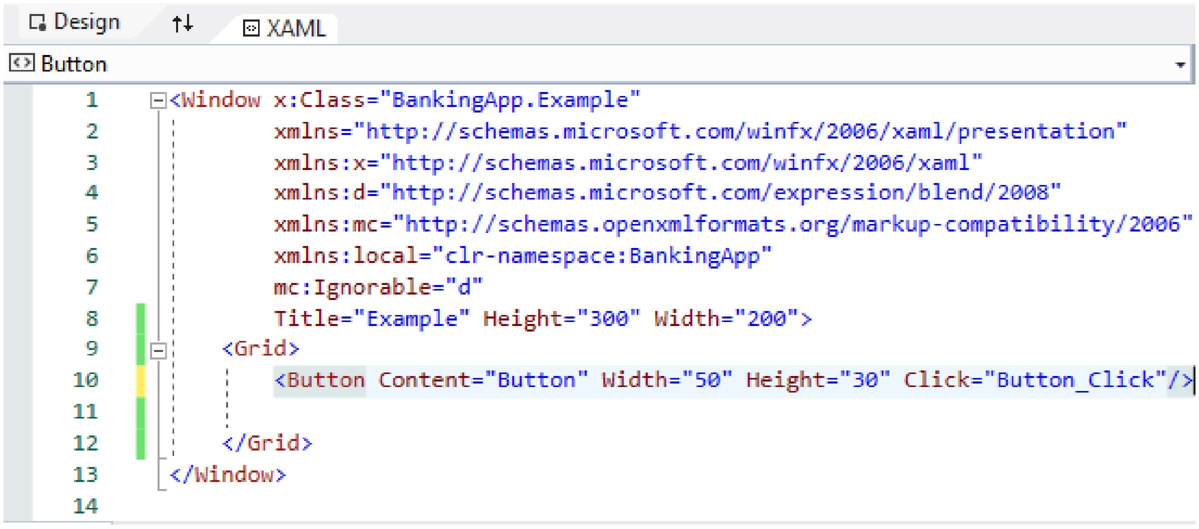
## C#, XAML, and the Windows Presentation Framework

When developing .NET applications, one typically uses the Windows Presentation Framework (WPF), which uses eXtensible Application Markup Language (XAML, pronounced "Zammel") to define the presentation of the application and the C# programming language to define the logic of the application.

XAML can be generated in Visual Studio by using the Designer - a What-You-See-Is-What-You-Get (WYSIWYG) editor.

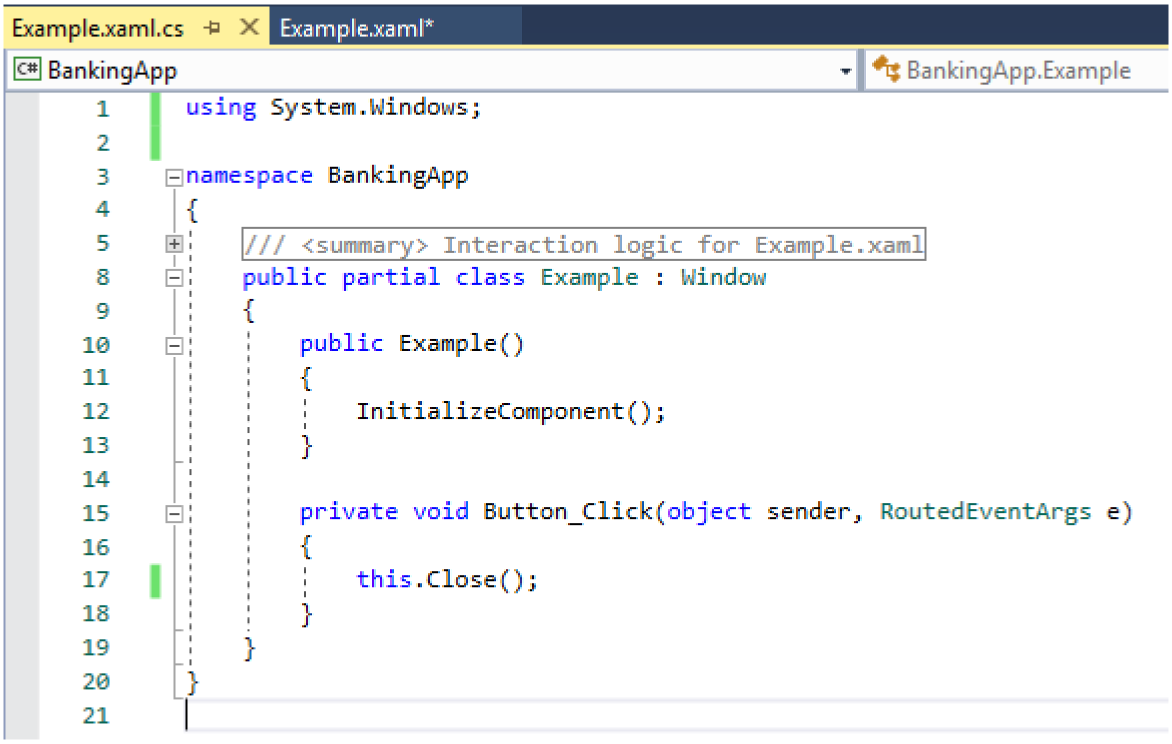


XAML elements can also be created by manually typing in the required markup.



```
<Window x:Class="BankingApp.Example"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:BankingApp"
        mc:Ignorable="d"
        Title="Example" Height="300" Width="200">
    <Grid>
        <Button Content="Button" Width="50" Height="30" Click="Button_Click"/>

    </Grid>
</Window>
```

C# code is used to define the logic of the form; for example, the behavior when a button is clicked.

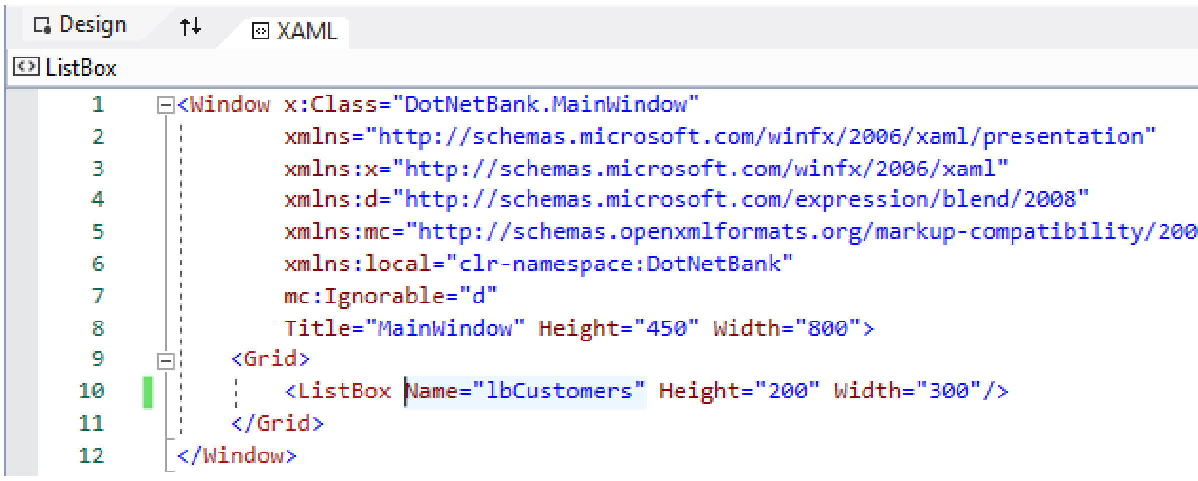


```
using System.Windows;

namespace BankingApp
{
    /// <summary> Interaction logic for Example.xaml
    public partial class Example : Window
    {
        public Example()
        {
            InitializeComponent();
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
        }
    }
}
```

## Exercise 5 – Creating a WPF Application

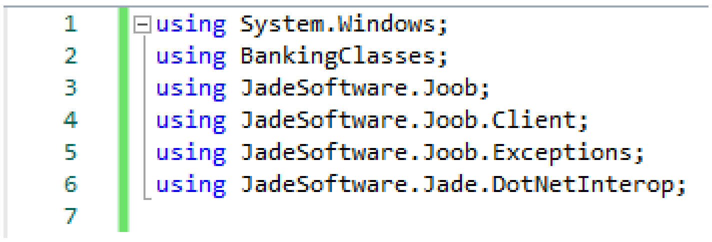
In this exercise, you will create a WPF application that will provide a .NET interface to the JADE banking system exposure. At first, you will simply display all customers of the bank in a list box.

1. Create a **ListBox** control in the center of the window using the Designer and then add **Name="lbCustomers"** to the **ListBox** element in the XAML.



```
<Window x:Class="DotNetBank.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/200
        xmlns:local="clr-namespace:DotNetBank"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>
        <ListBox Name="lbCustomers" Height="200" Width="300"/>
    </Grid>
</Window>
```

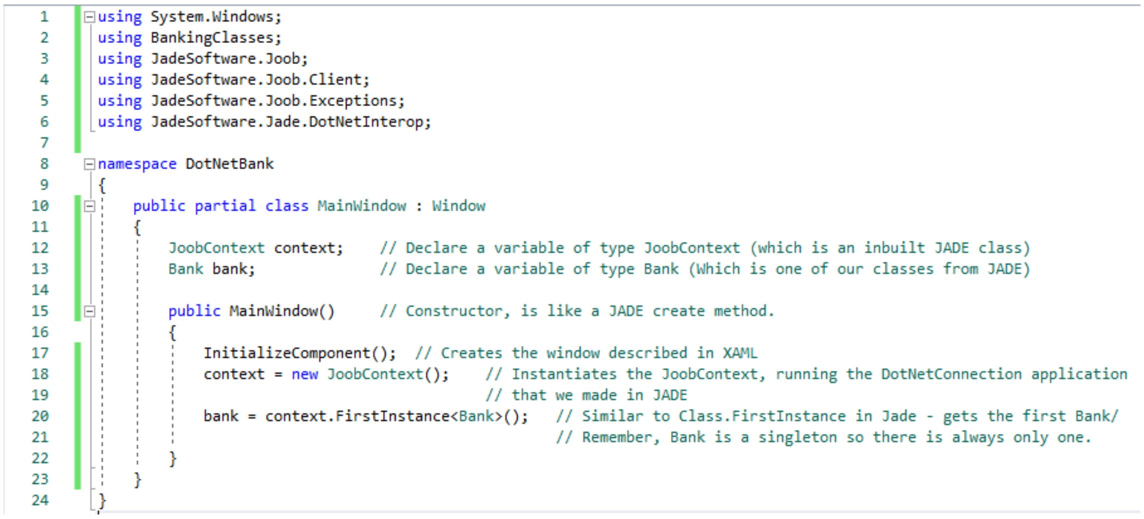
2. Switch to Code View by pressing F7 and then add the following **using** statements.



```
using System.Windows;
using BankingClasses;
using JadeSoftware.Joob;
using JadeSoftware.Joob.Client;
using JadeSoftware.Joob.Exceptions;
using JadeSoftware.Jade.DotNetInterop;
```

**Note** There may be superfluous **using** statements when the code is first opened. To remove them, select any **using** statement, press Alt+Enter, and then click **Remove Unnecessary Usings**.

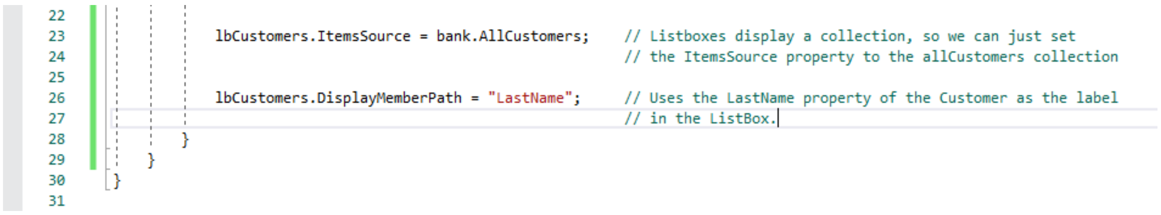
3. Add the following code. This will create a reference called **bank**, which through the **JoobContext** object, refers to the **Bank** class defined in JADE.



```
using System.Windows;
using BankingClasses;
using JadeSoftware.Joob;
using JadeSoftware.Joob.Client;
using JadeSoftware.Joob.Exceptions;
using JadeSoftware.Jade.DotNetInterop;

namespace DotNetBank
{
    public partial class MainWindow : Window
    {
        JoobContext context;    // Declare a variable of type JoobContext (which is an inbuilt JADE class)
        Bank bank;              // Declare a variable of type Bank (Which is one of our classes from JADE)

        public MainWindow()     // Constructor, is like a JADE create method.
        {
            InitializeComponent();  // Creates the window described in XAML
            context = new JoobContext();    // Instantiates the JoobContext, running the DotNetConnection application
                                            // that we made in JADE
            bank = context.FirstInstance<Bank>();   // Similar to Class.FirstInstance in Jade - gets the first Bank/
                                                    // Remember, Bank is a singleton so there is always only one.
        }
    }
}
```

**Note** The comments are for illustrative purposes and can be omitted.

4. To populate the **lbCustomers** list box with the customers of the bank, add the following to the **MainWindow** constructor method.



```
            lbCustomers.ItemsSource = bank.AllCustomers;     // Listboxes display a collection, so we can just set
                                                             // the ItemsSource property to the allCustomers collection

            lbCustomers.DisplayMemberPath = "LastName";      // Uses the LastName property of the Customer as the label
                                                             // in the ListBox.
        }
    }
}
```

5. Press F5 to run the application. You should now see the last names of the bank's customers displayed in the list box.