Deep Dive into Apps for Office with Excel

Office 365

Hands-on lab

In this lab you will get hands-on experience developing an App for Office which targets Microsoft Excel and creates bindings between the app and a spreadsheet.

This document is provided for informational purposes only and Microsoft makes no warranties, either express or implied, in this document. Information in this document, including URL and other Internet Web site references, is subject to change without notice. The entire risk of the use or the results from the use of this document remains with the user. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in examples herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Copyright 2014 © Microsoft Corporation. All rights reserved.

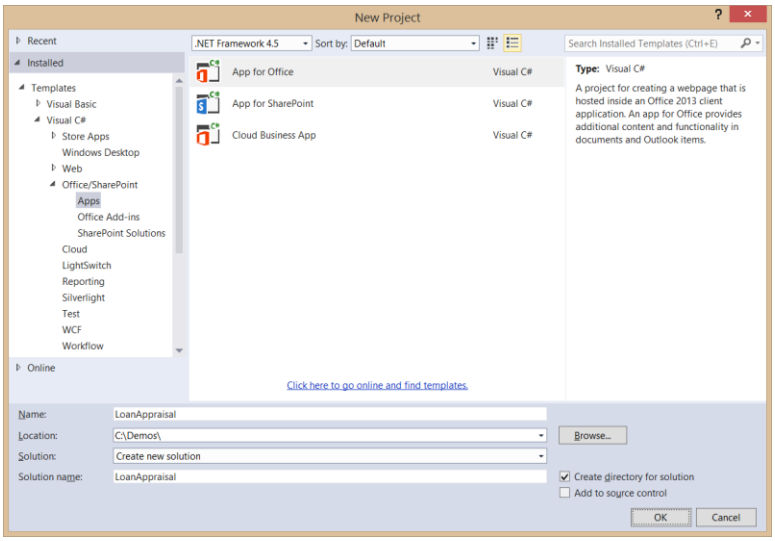
Microsoft, Internet Explorer, Microsoft Azure, Microsoft Office, Office 365, Visual Studio, and Windows are trademarks of the Microsoft group of companies.

All other trademarks are property of their respective owners.

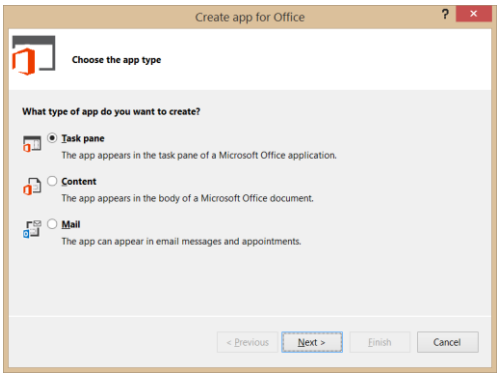
Exercise 1: Creating the LoanAppraisal App for Office Project

In this exercise you will create a new App for Office project in Visual Studio so that you can begin to write, test and debug an App for Office which targets Microsoft Excel.

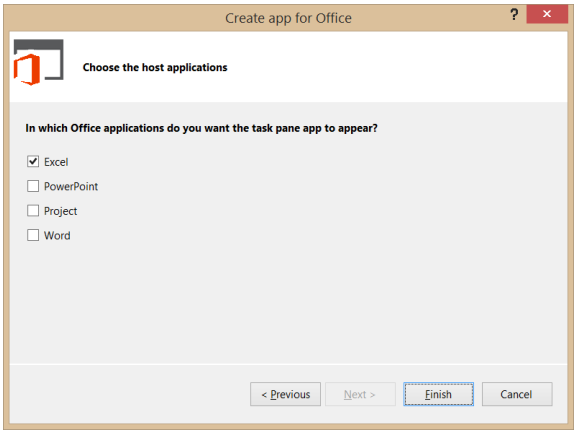
1. Launch Visual Studio 2013 as administrator.
2. From the **File** menu select the **New Project** command. When the **New Project** dialog appears, select the **App for Office** project template from the **Office/SharePoint** template folder as shown below. Name the new project **LoanAppraisal** and click **OK** to create the new project.



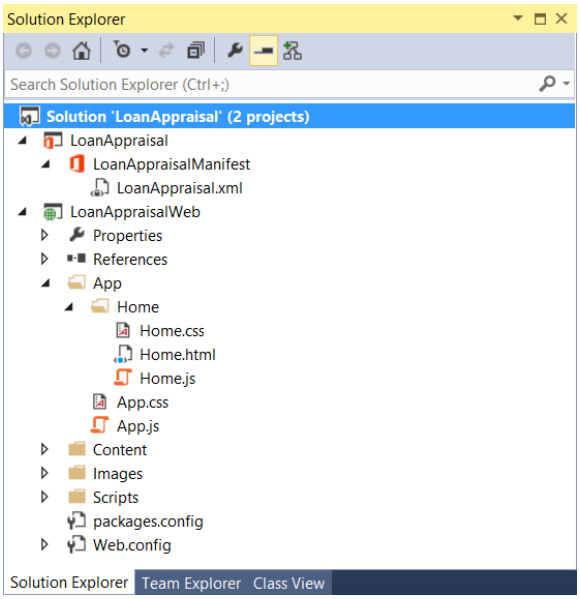
3. When you create a new App for Office project, Visual Studio prompts you with the **Choose the app type** page of the **Create app for Office** dialog. This is the point where you select the type of App for Office you want to create. Leave the default setting with the radio button titled **Task pane** and select **Next** to continue.



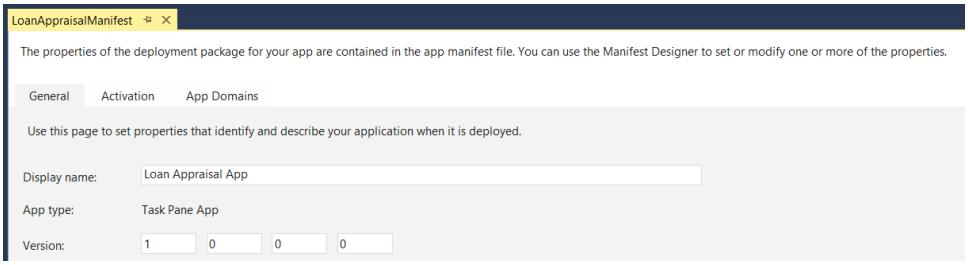
4. On the **Choose the host applications** page of the **Create app for Office** dialog, uncheck all the Office application except for **Excel** and then click **Finish** to create the new Visual Studio solution.



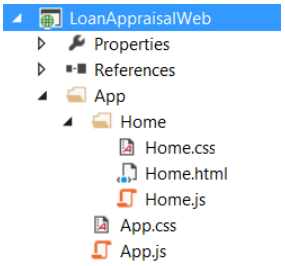
5. Take a look at the structure of the new Visual Studio solution once it has been created. At a high-level, the new solution has been created using two Visual Studio projects named **LoanAppraisal** and **LoanAppraisalWeb**. You should also observe that the top project contains a top-level manifest for the app named **LoanAppraisalManifest** which contains a single file named **LoanAppraisal.xml**.



6. In the Solution Explorer, double-click on the node named **LoanAppraisalManifest** to open the app manifest file in the Visual Studio designer. Update the **Display Name** settings in the app manifest from **LoanAppraisal** to **Loan Appraisal App**.



7. Save and close **LoanAppraisalManifest**.
8. Over the next few steps you will walk through the default app implementation that Visual Studio generated for you when the app project was created. Begin by looking at the structure of the **app** folder which has two important files named **app.css** and **app.js** which contain CSS styles and JavaScript code which is to be used on an app-wide basis.



9. You can see that inside the **app** folder there is a child folder named **Home** which contains three files named **Home.html**, **Home.css** and **Home.js**. Note that the app project is currently configured to use **Home.html** as the app's start page and that **Home.html** is linked to both **Home.css** and **Home.js**.
10. Double-click on **app.js** to open it in a code editor window. you should be able to see that the code creates a global variable named **app** based on the JavaScript *Closure* pattern. The global **app** object defines a method named **initialize** but it does not execute this method.

```
↪  var app = (function () {
↪    "use strict";
↪
↪    var app = {};
↪
↪    // Common initialization function (to be called from each page)
↪    app.initialize = function () {
↪      $('body').append(
↪        '<div id="notification-message">' +
↪        '<div class="padding">' +
↪        '<div id="notification-message-close"></div>' +
↪        '<div id="notification-message-header"></div>' +
↪        '<div id="notification-message-body"></div>' +
↪        '</div>' +
```

```
↪        '</div>');
↪
↪        $('#notification-message-close').click(function () {
↪          $('#notification-message').hide();
↪        });
↪
↪
↪        // After initialization, expose a common notification function
↪        app.showNotification = function (header, text) {
↪          $('#notification-message-header').text(header);
↪          $('#notification-message-body').text(text);
↪          $('#notification-message').slideDown('fast');
↪        };
↪      };
↪
↪        return app;
↪  })();
```

11. Close **app.js** and be sure not to save any changes.
12. Next you will examine the JavaScript code in **home.js**. Double-click on **home.js** to open it in a code editor window. Note that **Home.html** links to **app.js** before it links to **home.js** which means that JavaScript code written in **Home.js** can access the global **app** object created in **app.js**.
13. Walk through the code in **Home.js** and see how it uses a self-executing function to register an event handler on the **Office.initialize** method which in turn registers a document-ready event handler using jQuery. This allows the app to call **app.initialize** and to register an event handler using the **getDataFromSelection** function.

```
↪  (function () {
↪    "use strict";
↪
↪    // The initialize function must be run each time a new page is loaded
↪    Office.initialize = function (reason) {
↪      $(document).ready(function () {
↪        app.initialize();
↪        $('#get-data-from-selection').click(getDataFromSelection);
↪      });
↪    };
↪
↪    // Reads data from current document selection and displays a notification
↪    function getDataFromSelection() {
↪      Office.context.document.getSelectedDataAsync(Office.CoercionType.Text,
↪        function (result) {
↪          if (result.status === Office.AsyncResultStatus.Succeeded) {
↪            app.showNotification('The selected text is:', '"' + result.value + '"');
↪          } else {
```

```
            app.showNotification('Error:', result.error.message);
        }
      });
    }
  })();
```

14. Delete the **getDataFromSelection** function from **Home.js** and also remove the line of code that binds the event handler to the button with the id of **get-data-from-selection** so your code matches the following code listing.

```
(function () {
  "use strict";

  // The initialize function must be run each time a new page is loaded
  Office.initialize = function (reason) {
    $(document).ready(function () {
      app.initialize();
      // your app initialization code goes here
    });
  };

})();
```

15. Save your changes to **Home.js**. You will return to this source file after you have added your HTML layout to **Home.html**.
16. Now it time to examine the HTML that has been added to the project to create the app's user interface. Double-click **Home.html** to open this file in a Visual Studio editor window. Examine the layout of HTML elements inside the body element.

```
<body>
    <div id="content-header">
        <div class="padding">
            <h1>Welcome</h1>
        </div>
    </div>
    <div id="content-main">
        <div class="padding">
            <p><strong>Add home screen content here.</strong></p>
            <p>For example:</p>
            <button id="get-data-from-selection">Get data from selection</button>

            <p style="margin-top: 50px;">
                <a target="_blank" href="https://go.microsoft.com/fwlink/?LinkId=276812">Find more samples online...</a>
            </p>
```

```
↪         </div>
↪     </div>
↪ </body>
```

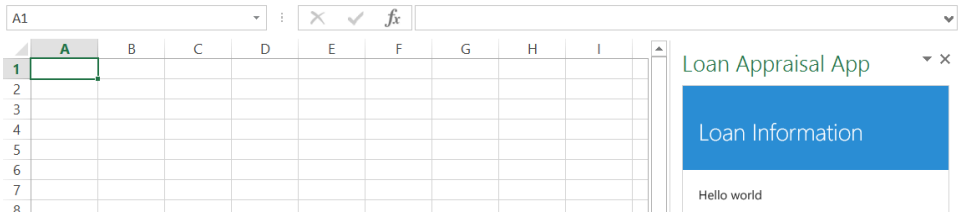
17. Replace the text message of **Welcome** inside the **h1** element with a different message such as **Loan Information**. Also trim down the contents of the **content-main** div element to match the HTML code shown below. You will start off your HTML layout using a single div element with an id of **results**.

```
↪ <body>
↪     <div id="content-header">
↪         <div class="padding">
↪             <h1>Loan Information</h1>
↪         </div>
↪     </div>
↪     <div id="content-main">
↪         <div class="padding">
↪             <div id="results"></div>
↪         </div>
↪     </div>
↪ </body>
```

18. Save and close **Home.html**.
19. Return to **Home.js** and modify to code to write a simple message to the **results** div using the following code.

```
↪ (function () {
↪     "use strict";
↪
↪     // The initialize function must be run each time a new page is loaded
↪     Office.initialize = function (reason) {
↪         $(document).ready(function () {
↪             app.initialize();
↪             $('#results').text("Hello world");
↪         });
↪     }
↪
↪ })();
```

20. Now it's time to test the app using the Visual Studio debugger. Press the **{F5}** key to run the project in the Visual Studio debugger. The debugger should launch Microsoft Excel 2013 and you should see your **LoanAppraisal** app in the task pane on the right side of a new Excel workbook as shown in the following screenshot.

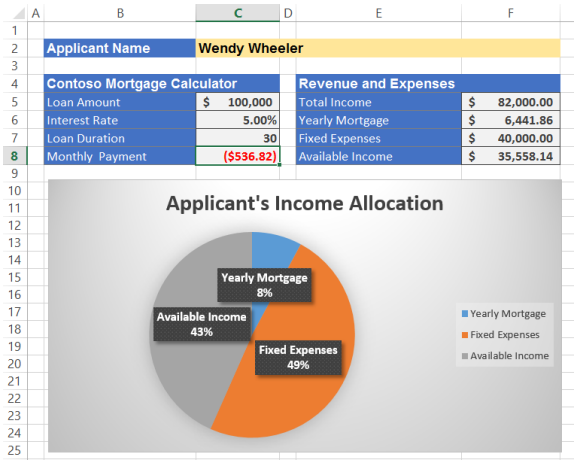


21. Close Microsoft Excel to terminate your debugging session and return to Visual Studio.

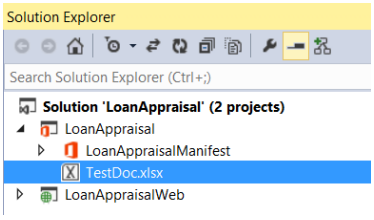
Exercise 2: Adding a Test Document to an Apps for Office project

In this exercise you continue to work on the LoanAppraisal project you created in the previous lab by integrating a preexisting Excel workbook into the development process. This will make it possible for you to develop an app binds to named ranges within the workbook.

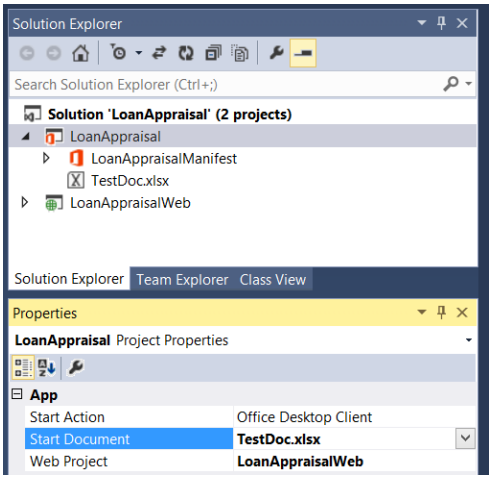
1. Ensure that you still have the **LoanAppraisal** app project opened in Visual Studio.
2. Using Windows Explorer, look in the **Starter Files** folder inside the folder for this lab and fins the workbook file named **TestDoc.xlsx**.
3. Double-click on **TestDoc.xlsx** to open the workbook in Microsoft Excel. You should see that the workbook provides morgage loan information and a chart as shown in the following screenshot.



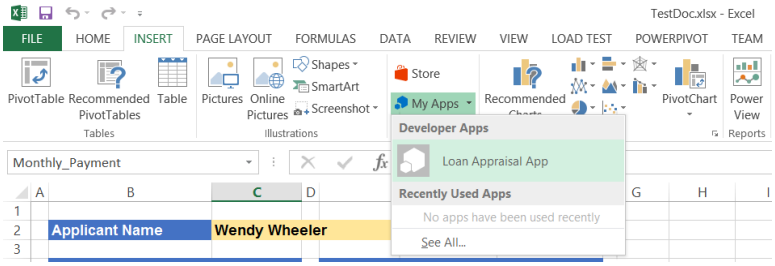
4. Close **TestDoc.xlsx** and also close Microsoft Excel.
5. Add the file **TestDoc.xlsx** into the **LoanAppraisal** project. The easiest you to do this is to copy the file to the clipboard in Windows Explorer and then to paste it into the root of the **LoanAppraisal** project. When you are done, you should be able to see **TestDoc.xlsx** at the root of the the **LoanAppraisal** project righ below **LoanAppraisalManifest** as shown in the following screenshot.



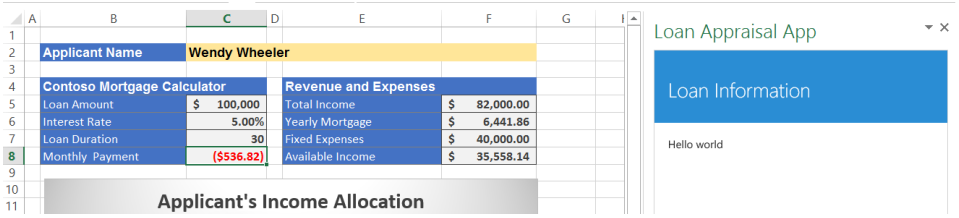
6. With the **LoanAppraisal** project selected in the Solution Explorer, locate the properties window and modify the **Start Document** property to **TestDoc.xslx**.



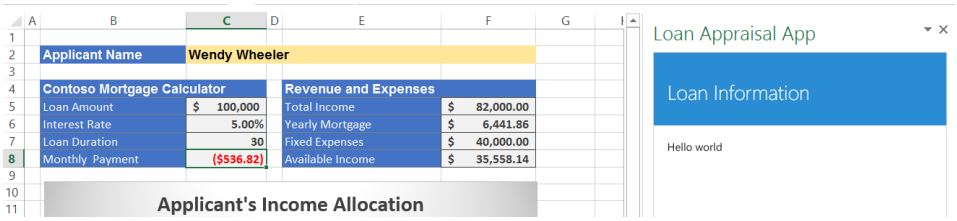
7. Press **{F5}** to begin a debugging session. You should see that Visual Studio initialize the debugging session with **TestDoc.xlsx** instead of using a new Excel workbook. However, you might notice that the **LoanAppraisal** app has not be activated. In the Excel ribbon, navigate to the **Insert** tab and select **Loan Appraisal App** from the **My Apps** drop down menu.



8. You should now see that the app has activated over in the task pane.



9. Inside Excel, save your changes to **TestDoc.xlsx** to update the test file to include the app in future debugging sessions.
10. Close **TestDoc.xlsx** and then close Microsoft Excel.
11. Return to Visual Studio and press **{F5}** to start another debugging session. Now the app should be initialized automatically when Visual Studio initialize a debugging session.



12. Now that you have integrated the test document into your project, it is time to move ahead to the next exercise where you will write code to bind to name ranges in the workbook.

Exercise 3: Adding Bindings Between an App and a Excel Workbook

In this exercise you will write code to create bindings on named ranges within the the Excel workbook named TestDoc.xlsx. You will also create event handlers so that the app responds to the user when updating the app user interface.

1. The workbook **TestDoc.xlsx** contains several cells that have already been defined as named ranges. Review the following list which shows the names of the Excel named ranges that you will be programming against in this exercise.
   - **Applicant_Name**
   - **Loan_Amount**
   - **Interest_Rate**
   - **Loan_Duration**
   - **Monthly_Payment**
   - **Total_Income**
   - **Yearly_Mortgage**
   - **Fixed_Expenses**
   - **Available_Income**
2. Open **Home.html** in an editor window.
3. Modify the contents of the **content-main** div element with the code shown in the following listing. If you would rather not type it all in by hand, you copy and paste this HTML code from **content-main.css.txt** which is located in the **Starter Files** folder for this lab.

```
↪  <div id="content-main">
↪      <div id="currentApplicantInfo">
↪          <table>
↪              <tr>
↪                  <td colspan="2" class="header_cell">Loan Application Detail</td>
↪              </tr>
↪              <tr>
↪                  <td>Name:</td>
↪                  <td id="applicant_name"> </td>
↪              </tr>
↪              <tr>
↪                  <td>Loan Amount:</td>
↪                  <td id="loan_amount"> </td>
↪              </tr>
↪              <tr>
↪                  <td>Interest Rate:</td>
↪                  <td id="interest_rate"> </td>
↪              </tr>
↪              <tr>
```

```
↪                   <td>Load Duration:</td>
↪                   <td id="loan_duration"> </td>
↪               </tr>
↪               <tr>
↪                   <td>Monthy Payment:</td>
↪                   <td id="monthly_payment"> </td>
↪               </tr>
↪               <tr>
↪                   <td colspan="2" class="header_cell">High-level Finanical
    Summary</td>
↪               </tr>
↪               <tr>
↪                   <td>Total Income:</td>
↪                   <td id="total_income"> </td>
↪               </tr>
↪               <tr>
↪                   <td>Yearly Morgage:</td>
↪                   <td id="yearly_mortgage"> </td>
↪               </tr>
↪               <tr>
↪                   <td>Fixed Expenses:</td>
↪                   <td id="fixed_expenses"> </td>
↪               </tr>
↪               <tr>
↪                   <td>Available Income:</td>
↪                   <td id="available_income"> </td>
↪               </tr>
↪           </table>
↪       </div>
↪
↪       <div class="padding">
↪           <h3>Interest Rate</h3>
↪           <div id="selectInterestRate" class="section"></div>
↪           <h3>Select a loan applicant</h3>
↪           <div id="selectApplicant" class="section"></div>
↪       </div>
↪
↪   </div>
```

4. Save and close **Home.html**.
5. Open **Home.css** in an editor window.
6. Modify the contents of **Home.css** with the set of CSS rules shown in the following listing. If you would rather not type it all in by hand, you copy and paste this HTML code from **Home.css.txt** which is located in the **Starter Files** folder for this lab.

```
↪   body {
```

```
↪    padding: 0px;
↪    background-color: #eee;
↪  }
↪
↪  h3 {
↪    margin: 2px;
↪  }
↪
↪  #currentApplicantInfo {
↪    margin: 0px;
↪    padding: 0px;
↪  }
↪
↪  .section {
↪    margin: 0px;
↪    padding: 0px;
↪    padding-top: 2px;
↪    padding-bottom: 4px;
↪  }
↪
↪  .section input[type="radio"] {
↪    margin: 0px;
↪    margin-left: 4px;
↪    padding: 0px;
↪  }
↪
↪  .section label {
↪    margin: 0px;
↪    padding: 0px;
↪    font-size: 0.8em;
↪  }
↪
↪  #currentApplicantInfo table {
↪    margin: 0px;
↪    width: 100%;
↪    box-sizing: border-box;
↪    border: 1px solid black;
↪    border-collapse: collapse;
↪  }
↪
↪  #currentApplicantInfo table td {
↪    min-width: 100px;
↪    border: 1px solid #ddd;
↪    border-collapse: collapse;
↪    padding: 2px;
↪    padding-left: 4px;
↪    background: white;
```

```
↪    font-size: 1.0em;
↪  }
↪
↪  #currentApplicantInfo table td.header_cell {
↪    color: #eee;
↪    background-color: navy;
↪    font-weight: bold;
↪    border: 1px solid black;
↪  }
↪
↪  #monthly_payment {
↪    color: red;
↪  }
```

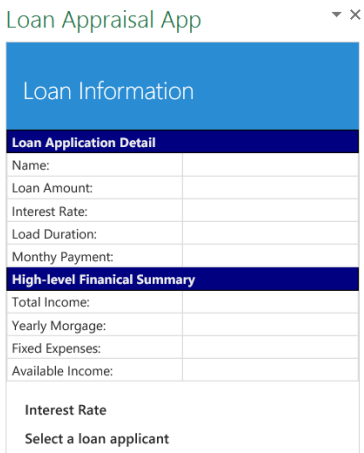
7. Save and close **Home.css**.
8. Open **Home.js** in a code editor widow. Remove the following line of code.

```
↪  $('#results').text("Hello world");
```

9. At this point, the code in **Home.js** should look like the following code listing.

```
↪  /// <reference path="../App.js" />
↪
↪  (function () {
↪      "use strict";
↪
↪      // The initialize function must be run each time a new page is loaded
↪      Office.initialize = function (reason) {
↪          $(document).ready(function () {
↪              app.initialize();
↪
↪          });
↪      }
↪
↪  })();
```

10. Start a debugging session by pressing the **{F5}** key to inspect the app's new HTML layout. You should see the user interface appears like the one in the following screenshot.



11. Close Excel and return to Visual Studio.
12. Inside **Home.js**, place the cursor under the **"use strict;"** statement at the top of the closure and add the following code. If you would rather not type this code by hand, you can copy and paste it from **Home.js_Part1.txt** inside the **Starter Files** folder for this lab.

```
var officeDoc;
var bindings;

var interestRates = [0.0425, 0.0500, 0.0750];
var currentRate = interestRates[0];

var applicants = [
  { name: "Brian Cox", loan_amount: 100000, loan_duration: 30, total_income: 82000, fixed_expenses: 22000 },
  { name: "Wendy Wheeler", loan_amount: 325000, loan_duration: 30, total_income: 145000, fixed_expenses: 40000 },
  { name: "Ken Sanchez", loan_amount: 225000, loan_duration: 30, total_income: 162000, fixed_expenses: 40000 },
  { name: "Joe Healy", loan_amount: 625000, loan_duration: 30, total_income: 182000, fixed_expenses: 72000 },
  { name: "Mke Fitzmaurice", loan_amount: 725000, loan_duration: 8, total_income: 320000, fixed_expenses: 120000 },
  { name: "Chris Sells", loan_amount: 1225000, loan_duration: 15, total_income: 325000, fixed_expenses: 167000 }
];
var currentApplicant = applicants[0];
```

13. After this step is complete, your **Home.js** file should match the following code listing.

```
/// <reference path="../App.js" />

(function () {
    "use strict";

```

```
    var officeDoc;
    var bindings;

    var interestRates = [0.0425, 0.0500, 0.0750];
    var currentRate = interestRates[0];

    var applicants = [
      { name: "Brian Cox", loan_amount: 100000, loan_duration: 30, total_income: 82000, fixed_expenses: 22000 },
      { name: "Wendy Wheeler", loan_amount: 325000, loan_duration: 30, total_income: 145000, fixed_expenses: 40000 },
      { name: "Ken Sanchez", loan_amount: 225000, loan_duration: 30, total_income: 162000, fixed_expenses: 40000 },
      { name: "Joe Healy", loan_amount: 625000, loan_duration: 30, total_income: 182000, fixed_expenses: 72000 },
      { name: "Mke Fitzmaurice", loan_amount: 725000, loan_duration: 8, total_income: 320000, fixed_expenses: 120000 },
      { name: "Chris Sells", loan_amount: 1225000, loan_duration: 15, total_income: 325000, fixed_expenses: 167000 }
    ];
    var currentApplicant = applicants[0];

    // The initialize function must be run each time a new page is loaded
    Office.initialize = function (reason) {
        $(document).ready(function () {
            app.initialize();
        });
    }

})();
```

14. Place your cursor under the code that assigns a function to **Office.initialize** and add five new functions named **updateAppUI**, **onInitializeUI**, **formatToCurrencyUSD**, **onRateChanged** and **onApplicantChanged**.

```
// The initialize function must be run each time a new page is loaded
Office.initialize = function (reason) {
    $(document).ready(function () {
        app.initialize();
    });
};

function updateAppUI() {
}

function onInitializeUI() {
}

```

```
function formatToCurrencyUSD(amount) {
}

function onRateChanged() {
}

function onApplicantChanged() {
}
```

15. Implement the **updateAppUI** function with the following code. If you prefer, you can copy and paste this code from the file named **updateAppUI.js.txt** in the **Starter Files** folder for this lab.

```
function updateAppUI() {
    $("#applicant_name").text(currentApplicant.name);
    $("#loan_amount").text(formatToCurrencyUSD(currentApplicant.loan_amount));
    $("#interest_rate").text((currentRate * 100) + "%");
    $("#loan_duration").text(currentApplicant.loan_duration + " years");
    $("#total_income").text(formatToCurrencyUSD(currentApplicant.total_income));

$("#fixed_expenses").text(formatToCurrencyUSD(currentApplicant.fixed_expenses));
}
```

16. Implement the **onInitializeUI** function with the following code. If you prefer, you can copy and paste this code from the file named **onInitializeUI.js.txt** in the **Starter Files** folder for this lab.

```
function onInitializeUI() {

    var divRates = $("#selectInterestRate");
    divRates.empty();

    for (var i = 0; i < interestRates.length; ++i) {
        var rate = interestRates[i];
        divRates.append($('<input>', { type: 'radio', name: 'rate', value: rate
}));
        var formatedRate = (rate * 100).toFixed(2) + "%";
        divRates.append($('<label>').text(formatedRate));
        divRates.append($("<br>"));
    }

    var divApplicants = $("#selectApplicant");
    divApplicants.empty();

    for (i = 0; i < applicants.length; ++i) {
        var name = applicants[i].name;
        divApplicants.append($('<input>', { type: 'radio', name: 'Applicant',
value: i }));
        divApplicants.append($('<label>').text(applicants[i].name));
        divApplicants.append($("<br>"));
```

```
    }

    $("#selectInterestRate :first-child").attr("checked", "checked");
    $("#selectApplicant :first-child").attr("checked", "checked");

    $("input[name='rate']").click(onRateChanged);
    $("input[name='Applicant']").click(onApplicantChanged);

    updateAppUI();
}
```

17. Implement the **formatToCurrencyUSD** function with the following code. If you prefer, you can copy and paste this code from the file named **formatToCurrencyUSD.js.txt** in the **Starter Files** folder for this lab.

```
function formatToCurrencyUSD(amount) {
    var sign; var cents; var i;
    amount = amount.toString().replace(/\$|\,/g, '');
    if (isNaN(amount)) { amount = "0"; }
    sign = (amount == (amount = Math.abs(amount)));
    amount = Math.floor(amount * 100 + 0.50000000001);
    cents = amount % 100;
    amount = Math.floor(amount / 100).toString();
    if (cents < 10) {
        cents = '0' + cents;
    }
    for (i = 0; i < Math.floor((amount.length - (1 + i)) / 3) ; i++) {
        amount = amount.substring(0, amount.length - (4 * i + 3)) + ',' +
amount.substring(amount.length - (4 * i + 3));
    }
    return (((sign) ? '' : '-') + '$' + amount + '.' + cents);
}
```

18. Implement **onRateChanged** and **onApplicantChanged** using the following code.

```
function onRateChanged() {
    var rate = parseFloat($(this).attr("value"));
    currentRate = rate;
    updateAppUI();
}

function onApplicantChanged() {
    var applicant = applicants[parseInt(this.value)];
    currentApplicant = applicant;
    updateAppUI();
}
```

19. Modify the app's initialization code to call the **onInitializeUI** function.

```
↪  // The initialize function must be run each time a new page is loaded
↪  Office.initialize = function (reason) {
↪      $(document).ready(function () {
↪          app.initialize();
↪          onInitializeUI();
↪      });
↪  }
```

20. Now it's again time to test the app in the Visual Studio. Press the **{F5}** key and wait for the debugging session and the app to initialize. Once the app has activated, you should be able to see it is displaying information about a load for the current applicant as shown in the following screenshot. Also note that the UI for the app will automatically update when you change the interest rate or the loan applicant.

21. Close Excel and return to Visual Studio.
22. Inside **Home.js** directly below the **onApplicantChanged** function, add six new functions named **createBindings**, **onAllBindingCreated**, **updateBindingsToDocument**, **onBindingUpdated**, **updateBindingsFromDocument** and **onBindingReadFromDocument**.

```
↪  function createBindings() {
↪  }
↪
↪  function onAllBindingCreated(asyncResult) {
↪  }
↪
↪
↪  function updateBindingsToDocument() {
```

```
↪  }
↪
↪  function onBindingUpdated(asncResult) {
↪  }
↪
↪  function updateBindingsFromDocument() {
↪  }
↪
↪  function onBindingReadFromDocument(asyncResult) {
↪  }
```

23. Implement the **createBindings** function using the following code. If you prefer, you can copy and paste this code from the file named **createBindings.js.txt** in the **Starter Files** folder for this lab.

```
↪  function createBindings() {
↪
↪      bindings.addFromNamedItemAsync("Sheet1!Applicant_Name", "text",
↪                                      { id: "applicant_name" }, function () { });
↪
↪      bindings.addFromNamedItemAsync("Sheet1!Loan_Amount", "text",
↪                                     { id: "loan_amount" }, function () { });
↪
↪      bindings.addFromNamedItemAsync("Sheet1!Interest_Rate", "text",
↪                                     { id: "interest_rate" }, function () { });
↪
↪      bindings.addFromNamedItemAsync("Sheet1!Loan_Duration", "text",
↪                                     { id: "loan_duration" }, function () { });
↪
↪      bindings.addFromNamedItemAsync("Sheet1!Monthly_Payment", "text",
↪                                     { id: "monthly_payment" }, function () { });
↪
↪      bindings.addFromNamedItemAsync("Sheet1!Total_Income", "text",
↪                                     { id: "total_income" }, function () { });
↪
↪      bindings.addFromNamedItemAsync("Sheet1!Yearly_Mortgage", "text",
↪                                     { id: "yearly_mortgage" }, function () { });
↪
↪      bindings.addFromNamedItemAsync("Sheet1!Fixed_Expenses", "text",
↪                                     { id: "fixed_expenses" }, function () { });
↪
↪      bindings.addFromNamedItemAsync("Sheet1!Available_Income", "text",
↪                                     { id: "available_income" },
   onAllBindingCreated);
↪
↪  }
```

24. Implement the **onAllBindingCreated** function using the following code.

```
function onAllBindingCreated(asyncResult) {
    updateBindingsToDocument();
}
```

25. Implement the **updateBindingsToDocument** function using the following code. If you prefer, you can copy and paste this code from the file named **updateBindingsToDocument.js.txt** in the **Starter Files** folder for this lab.

```
function updateBindingsToDocument() {

    Office.select("bindings#applicant_name")
            .setDataAsync(currentApplicant.name, function () { });

    Office.select("bindings#loan_amount")
            .setDataAsync(currentApplicant.loan_amount, function () { });

    Office.select("bindings#interest_rate")
            .setDataAsync(currentRate, function () { });

    Office.select("bindings#loan_duration")
            .setDataAsync(currentApplicant.loan_duration, function () { });

    Office.select("bindings#total_income")
            .setDataAsync(currentApplicant.total_income, function () { });

    Office.select("bindings#fixed_expenses")
            .setDataAsync(currentApplicant.fixed_expenses, onBindingUpdated);
    }
}
```

26. Implement the **onBindingUpdated** function using the following code.

```
function onBindingUpdated(asncResult) {
    updateBindingsFromDocument();
}
```

27. Implement the **updateBindingsFromDocument** function using the following code. If you prefer, you can copy and paste this code from the file named **updateBindingsFromDocument.js.txt** in the **Starter Files** folder for this lab.

```
function updateBindingsFromDocument() {

    Office.select("bindings#monthly_payment")
          .getDataAsync({
              asyncContext: "monthly_payment",
              valueFormat: Office.ValueFormat.Formatted
```

```
            }, onBindingReadFromDocument);

    Office.select("bindings#yearly_mortgage")
            .getDataAsync({
                asyncContext: "yearly_mortgage",
                valueFormat: Office.ValueFormat.Formatted
            }, onBindingReadFromDocument);

    Office.select("bindings#available_income")
            .getDataAsync({
                asyncContext: "available_income",
                valueFormat: Office.ValueFormat.Formatted
            }, onBindingReadFromDocument);
}
```

28. Implement the **onBindingReadFromDocument** function using the following code.

```
function onBindingReadFromDocument(asyncResult) {
    var value = asyncResult.value;
    var targetDiv = "#" + asyncResult.asyncContext;
    $(targetDiv).text(value);
}
```

29. Update both **onRateChanged** and **onApplicantChanged** so that each of these functions calls **updateBindingsToDocument**.

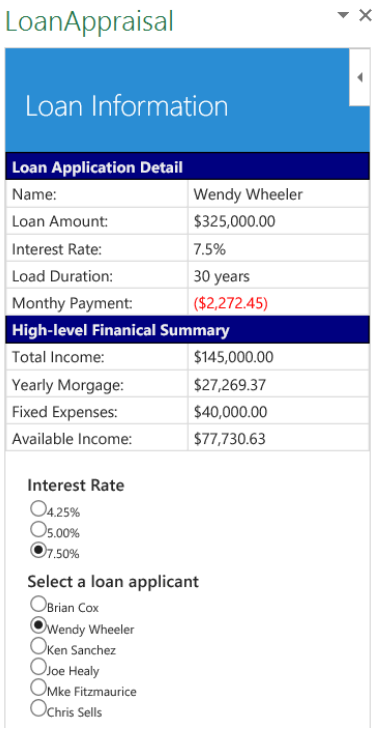
```
function onRateChanged() {
    var rate = parseFloat($(this).attr("value"));
    currentRate = rate;
    updateAppUI();
    updateBindingsToDocument();
}

function onApplicantChanged() {
    var applicant = applicants[parseInt(this.value)];
    currentApplicant = applicant;
    updateAppUI();
    updateBindingsToDocument();
}
```

30. Modify the app's initialization code to call the **createBindings** function just after calling **onInitializeUI**.

```
// The initialize function must be run each time a new page is loaded
Office.initialize = function (reason) {
    $(document).ready(function () {
        app.initialize();
        onInitializeUI();
        createBindings();
```

```
↪        });
↪    }
```

31. Now it's again time to test the app in the Visual Studio. Press the **{F5}** key and wait for the debugging session and the app to initialize. Once the app has activated, test how the app behaves when you change the Interest Rate or the Loan Applicant using the radio button at the bottom of the task pane. You should see that the app updates information in the workbook and then retrieves values from the workbook for Monthly Payment and Yearly Mortgage and updates the UI in the task pane.



32. You have now completed this exercise.