WPF Dev Camp - Module 2

Expenses App Scenario Overview

Lab version: 1.0

Last updated: 1/13/2015

Estimated demo delivery time: **10 minutes**.



Contents

[Overview 3](#_Toc408466720)

[Prerequisites 3](#_Toc408466721)

[Setup 3](#_Toc408466722)

[Exercise 1: Expenses App Scenario Overview 4](#_Toc408466723)

[Task 1: Expenses WPF Application Demo 4](#_Toc408466724)

[Task 2: Expenses WCF Code Overview 10](#_Toc408466725)

[Task 3: Expenses WPF Code Overview 13](#_Toc408466726)

# Overview

In this demo, we will introduce the Expenses application scenario and implementation to provide a concrete context for additional demos to follow.

# Prerequisites

The following are required to complete this demo:

* [Microsoft Visual Studio 2013](http://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx) (tested with Update 4)
* Expenses codebase

# Setup

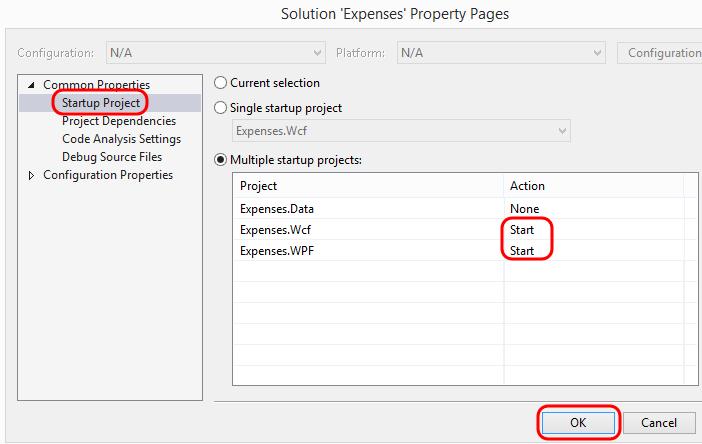
* Load and build the Expenses solution to ensure that it builds correctly. Note that the first build requires Internet connectivity in order to download needed NuGet packages.

# Exercise 1: Expenses App Scenario Overview

In this exercise, we will start by introducing the Expenses WPF application scenario and implementation.

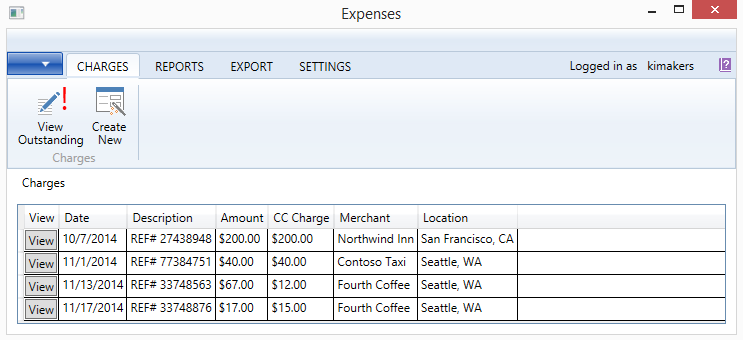
## Task 1: Expenses WPF Application Demo

1. Load the Expenses.sln solution in Visual Studio.
2. Right-click the Expenses solution node and select Properties.
3. In the Startup Project node, select the “Multiple startup projects” option and then change the Action property for both the Expenses.Wcf and Expenses.WPF projects to be “Start”.
4. Click the OK button.

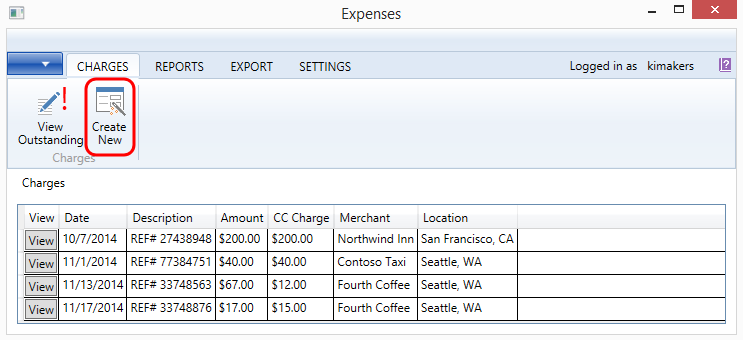


1. Press F5 to start both the WCF and WPF projects. The WCF service will be hosted locally in IIS Express.
2. In our demo scenario, the Charges tab shows the outstanding charges for the logged in user that are not already part of an expense report.

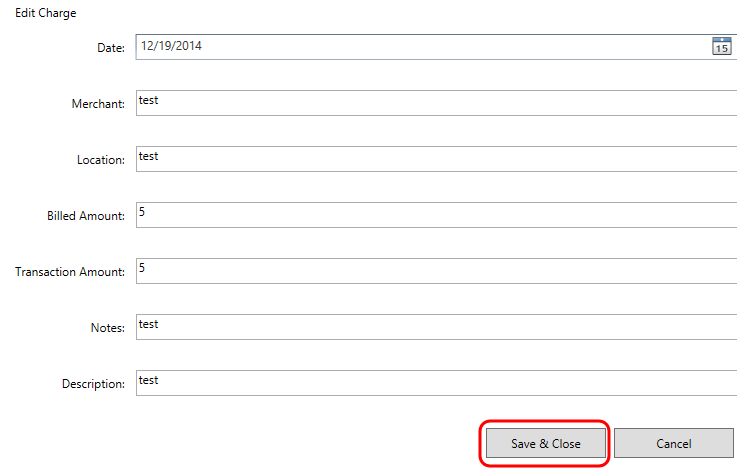
**Note:** This demonstration application does not actually provide login support, and thus the logged in user that is shown in the upper-right corner is simulated.



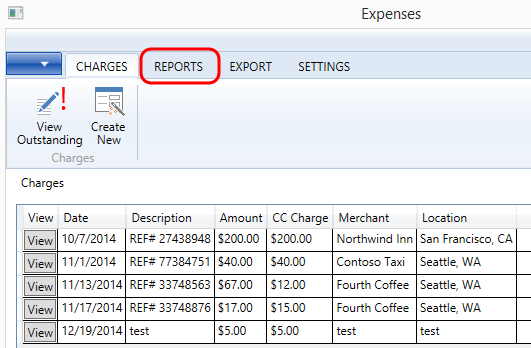
1. Click the Create New button.



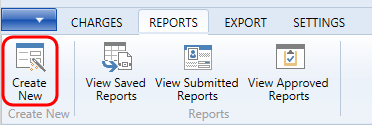
1. Provide the required fields for the new charge and then click Save & Close.



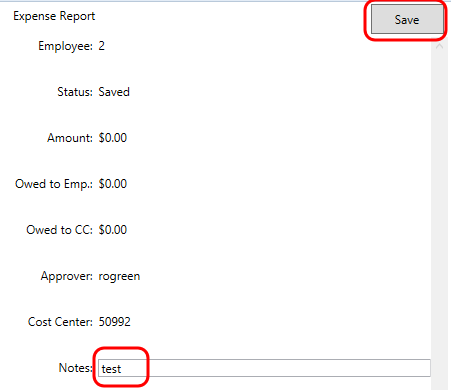
1. Click the Reports tab.



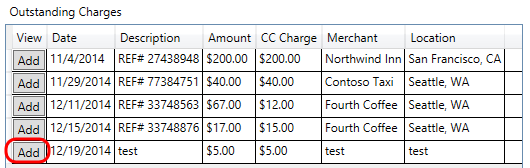
1. Click the Create New button.



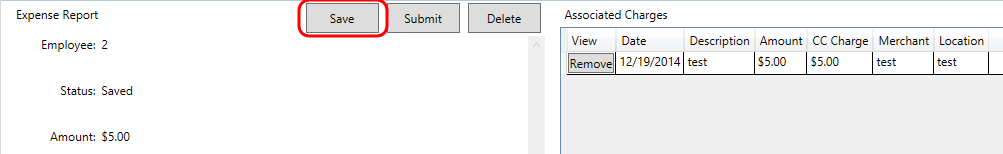
1. Provide some text in the Notes field and then click Save to create the new report.



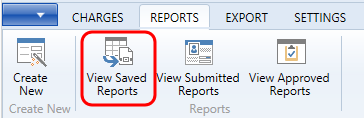
1. Click the Add button for the new charge in the Outstanding Charges pane to associate it with the expense report.



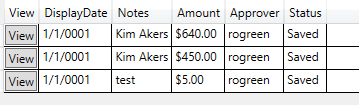
1. Click the Save button.



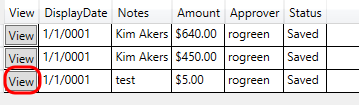
1. Click the View Saved Reports button.



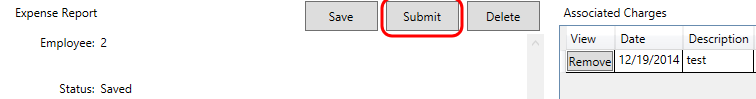
1. Our new expense report now shows up in this view, along with a couple of others that were saved previously.



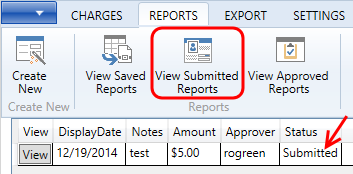
1. In this demo scenario, the next step in the workflow would be to submit the report to a manager for approval. Click the View button next to the new report to load it once again.



1. Select the Submit button.



1. The Submitted Reports view shows that our new expense report was submitted.



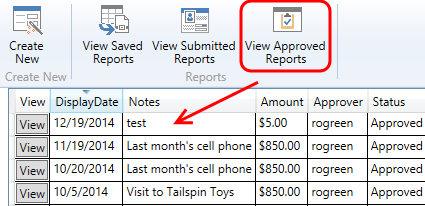
1. The final step in the expense report workflow is to get manager approval. Click the View button for the submitted expense report.



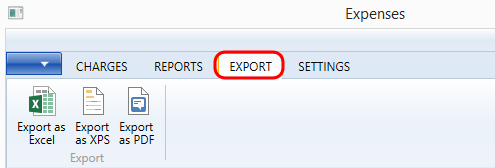
1. Click the Approve button.



1. The new expense report should now be the most recently approved report.



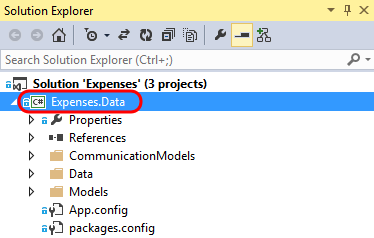
1. For the purposes of setting up our demo scenario, imagine that this application had many additional features that you might expect in a LOB application such as this, such as import and export. Click the Export tab and note that some export options are shown here (albeit non-functional in this demo app).



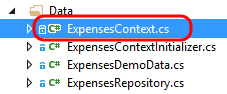
1. Close the Expenses application.

## Task 2: Expenses WCF Code Overview

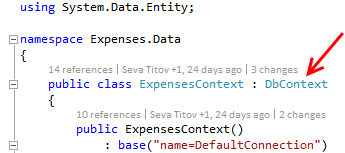
1. Return to Visual Studio and then press Shift+F5 to stop debugging.
2. In Solution Explorer, expand the Expenses.Data project. The Data project defines a data layer for the server-side WCF code.



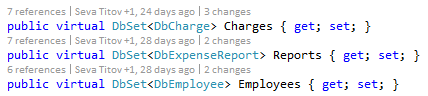
1. Open the ExpensesContext.cs file from the Data folder.



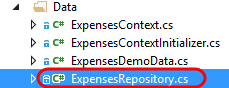
1. The ExpensesContext class uses Entity Framework Code First to define a data model that is persisted to SQL Server. Here you can see that the class derives from DbContext.



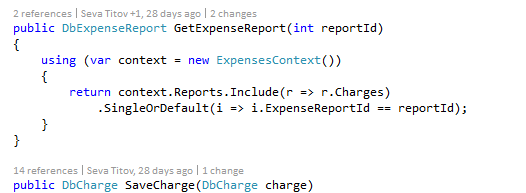
1. Scroll down and locate the DbSet definitions for Charges, Reports, and Employees.



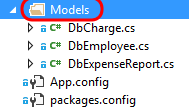
1. Open the ExpensesRepository.cs file from the Data folder.



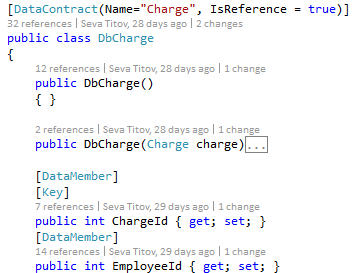
1. The ExpenseRepository class defines APIs for accessing the data layer, in order to provide typical CRUD operations against the database.



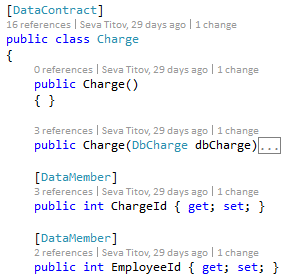
1. Expand and view the files located in the Models folder.



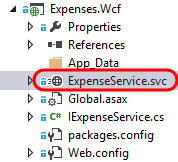
1. The Models folder contains the model objects that define the data model using Entity Framework. Open the DbCharge.cs file and take a look at the definition.



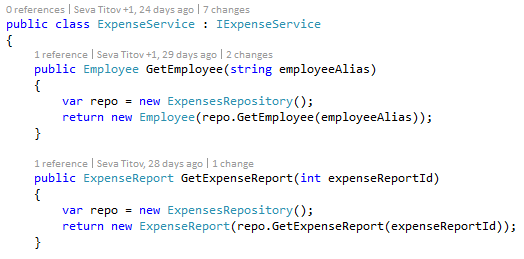
1. The CommunicationModels folder contains the model objects that are actually serialized and transferred over the wire back to client of the WCF service. These types of model objects are commonly referred to as Data Transfer Objects (DTOs). Open the Charge.cs file and take a look at the definition.



1. Expand the Expenses.Wcf project node and then double-click on the ExpenseService.svc file to load the service definition.

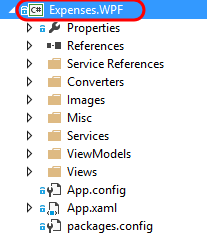


1. The ExpenseService class is similar to any other WCF service implementation in that it implements the interface that defines the service contract (IExpenseService in this case).

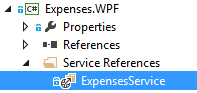


## Task 3: Expenses WPF Code Overview

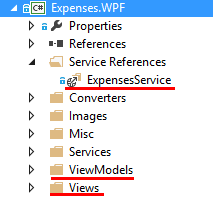
1. Expand the Expenses.WPF project node.



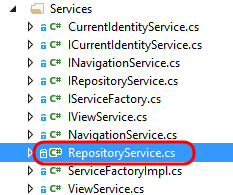
1. Expand the Service References folder and note that a service reference named ExpensesService was already added to the project. This generated the needed client-side proxy classes that are used by the WPF client application to communicate with the WCF service.



1. The WPF application is implemented using the often used Model View ViewModel (MVVM) architectural pattern. In this case, the model objects used are those that were auto-generated when adding the service reference, the views are contained in the Views folder, and the view-models are contained in the ViewModels folder.



1. Open the RepositoryService.cs file from the Services folder.



1. The RepositoryService class provides data to the Expenses app by making use of the proxy that was generated when the service reference was added.

