



enterprise library hands on labs

lab 2: caching block

After completing this lab, you will be able to:

- Add persistent caching to a windows application.
- Use background loading to populate a cache.

scenario

This lab demonstrates the use of the Enterprise Library Caching Block. It requires a (local)\SQLEXPRESS instance of SQL Server or SQL Server Express.

estimated time to complete this lab: 30 minutes.

exercise 1: using the caching block for performance

This exercise demonstrates how to implement caching using the Caching Block from the Enterprise Library. You will perform caching to speed up display of employee details, and then make the cache persistent, to support an offline scenario.

first step

1. Open the [EmployeeBrowser.sln](#) file.

populate the quickstarts database with employee data

1. Run the batch file **SetCachingHOL.bat**, which can be found in the lab directory: [labs\vb\Caching\setup](#).

This adds a set of employees which you will use to create an employee directory.

Note: the database will be installed into the **(local)\SQLEXPRESS** instance.

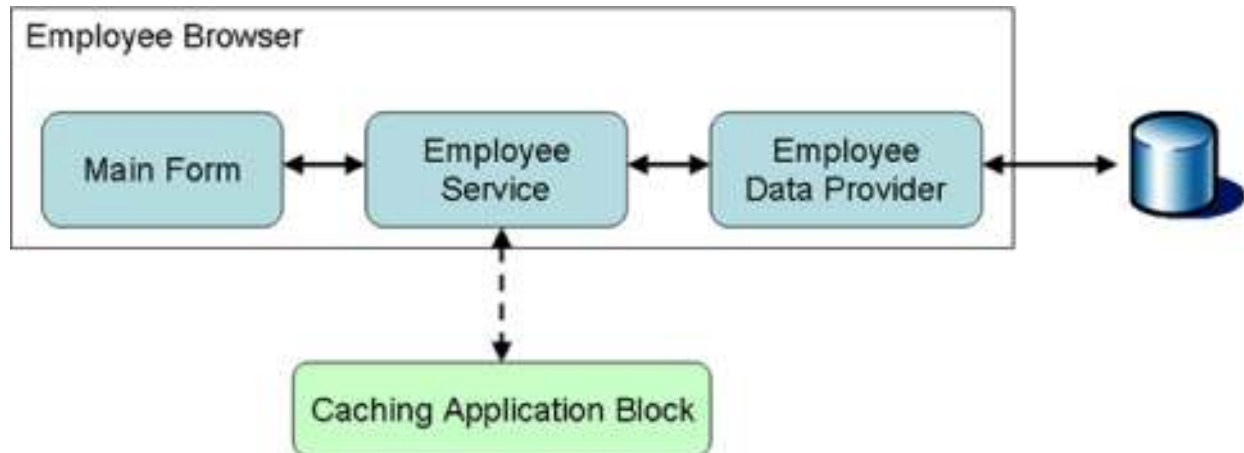
review the application

1. This application is a browser for the employee contact details stored in the database. As part of the browser the application displays a photograph of the employee.

Although the application loads all employee records at startup, it doesn't load the photographs until they are requested. Images can be large, and retrieving all images (if they are to be viewed or not) could seriously slow the startup performance.

2. Select the **MainForm.vb** file in the **EmployeeBrowser** project. Select the **View |**

Code menu command and locate the **MainForm_Load** method. The Form uses a class called **EmployeeService** to obtain the data to display. This in turn uses the **EmployeeDataProvider** class, as displayed in the picture below:



Currently, the **EmployeeService** just delegates directly to the **EmployeeDataProvider** class. You will enhance this class to use the Caching Block and to react appropriately when the application is offline.

3. Select the **EmployeeDataProvider.vb** file in the Solution Explorer. Select the **View | Code** menu command and locate the **GetEmployeePhotoData** method.

Note: The *GetEmployeePhotoData* method adds a 1 second delay to the call to simulate slow database access.

4. Select the **Debug | Start Without Debugging** menu command to run the application.

Browse through the employees.

Note: There is a significant delay while browsing through photos, even when you have viewed a photo previously.

implement caching in the employeeservice class

1. Select the **EmployeeBrowser** project. Select the **Project | Add Reference...** menu command. Select the **Browse** tab and select the following assembly located [here](#):
 - Microsoft.Practices.EnterpriseLibrary.Caching.dll.
2. Select the **EmployeeService.vb** file in the Solution Explorer. Select the **View | Code** menu command.
3. Add the following namespace inclusion to the list of namespaces at the top of the file:

```
Imports Microsoft.Practices.EnterpriseLibrary.Caching
```

4. Add the following code to the **GetEmployeePhoto** method.

```
Public Shared Function GetEmployeePhoto(ByVal employeeId As Guid) As
Bitmap
    Dim photoData As Byte() = Nothing
    ' TODO: Add Caching of Photo
    ' Attempt to retrieve from cache
    Dim cache As CacheManager = CacheFactory.GetCacheManager()
    photoData = CType(cache(employeeId.ToString()), Byte())
    ' Retrieve from dataProvider if not in Cache
    If photoData Is Nothing Then
        Dim dataProvider As New EmployeeDataProvider
        photoData = dataProvider.GetEmployeePhotoData(employeeId)
        cache.Add(employeeId.ToString(), photoData)
    End If
    ' No data found. Nothing to return
    If photoData Is Nothing Then
        Return Nothing
    End If
    ' Convert bytes to Bitmap
    Dim ms As New MemoryStream(photoData)
    Try
        Return New Bitmap(ms)
    Finally
        ms.Dispose()
    End Try
End Function 'GetEmployeePhoto
```

*This method uses the factory model, as with the rest of Enterprise Library, to create a new CacheManager instance. This can be purely in-memory, or can be backed by a physical storage medium, depending on configuration. Items can be retrieved from the cache by using an indexer, and can be added (or replaced) by using the **Add** method. The overload used in this method does not specify an expiration policy.*

5. Add the following code to the **ClearCache** method, to allow the form to request the service to get new data.

```
Public Shared Sub ClearCache()
    ' TODO: Clear Cache
    Dim cache As CacheManager = CacheFactory.GetCacheManager()
    cache.Flush()
End Sub 'ClearCache
```

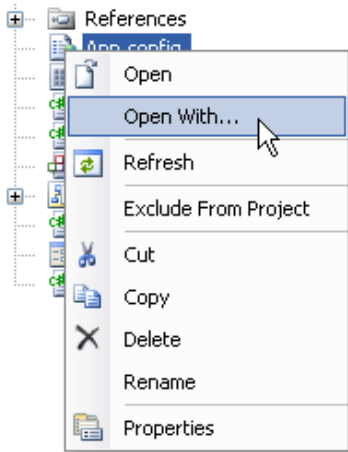
This method will remove all items from the cache.

"open with..." configuration console

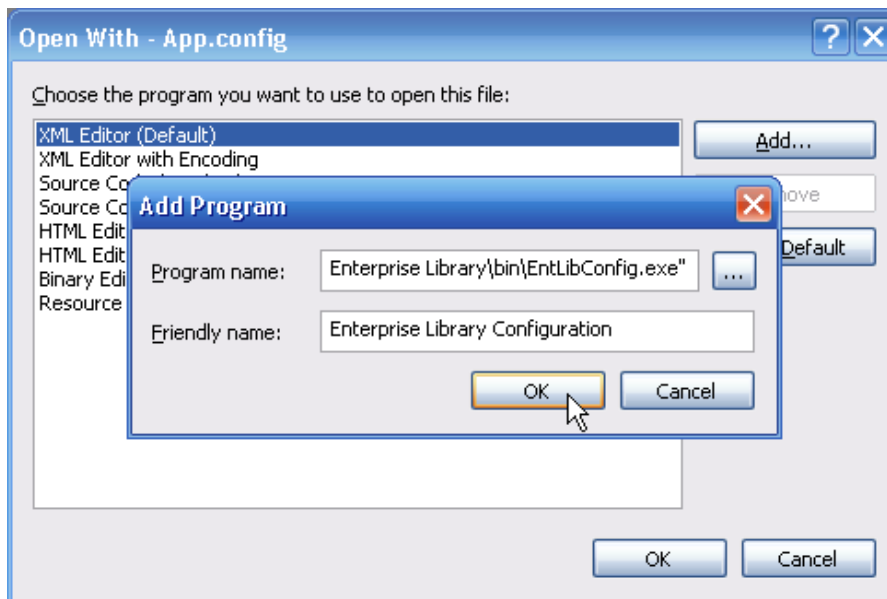
1. We will use the Enterprise Library Configuration tool to configure our application. You may either start this tool from the Windows Start menu (select **All Programs | Microsoft patterns and practices | Enterprise Library | Enterprise Library Configuration**) and open the App.config file located [here](#).

Alternatively you may configure Visual Studio to open the configuration file with the tool, as described below.

2. Select the **App.config** file in the Solution Explorer. Select the **View | Open With...** menu command. The **OpenWith** dialog is displayed. Click the **Add** button.

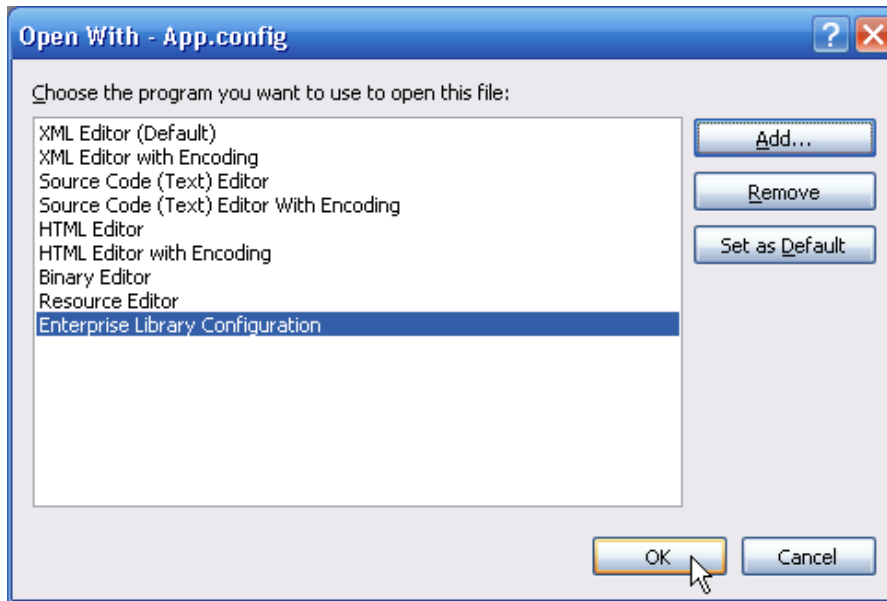


3. In the **Add Program** dialog, set the **Program name** to the **EntLibConfig.exe** file found [here](#). Set the **Friendly name** to **Enterprise Library Configuration**. Click the **OK** button.



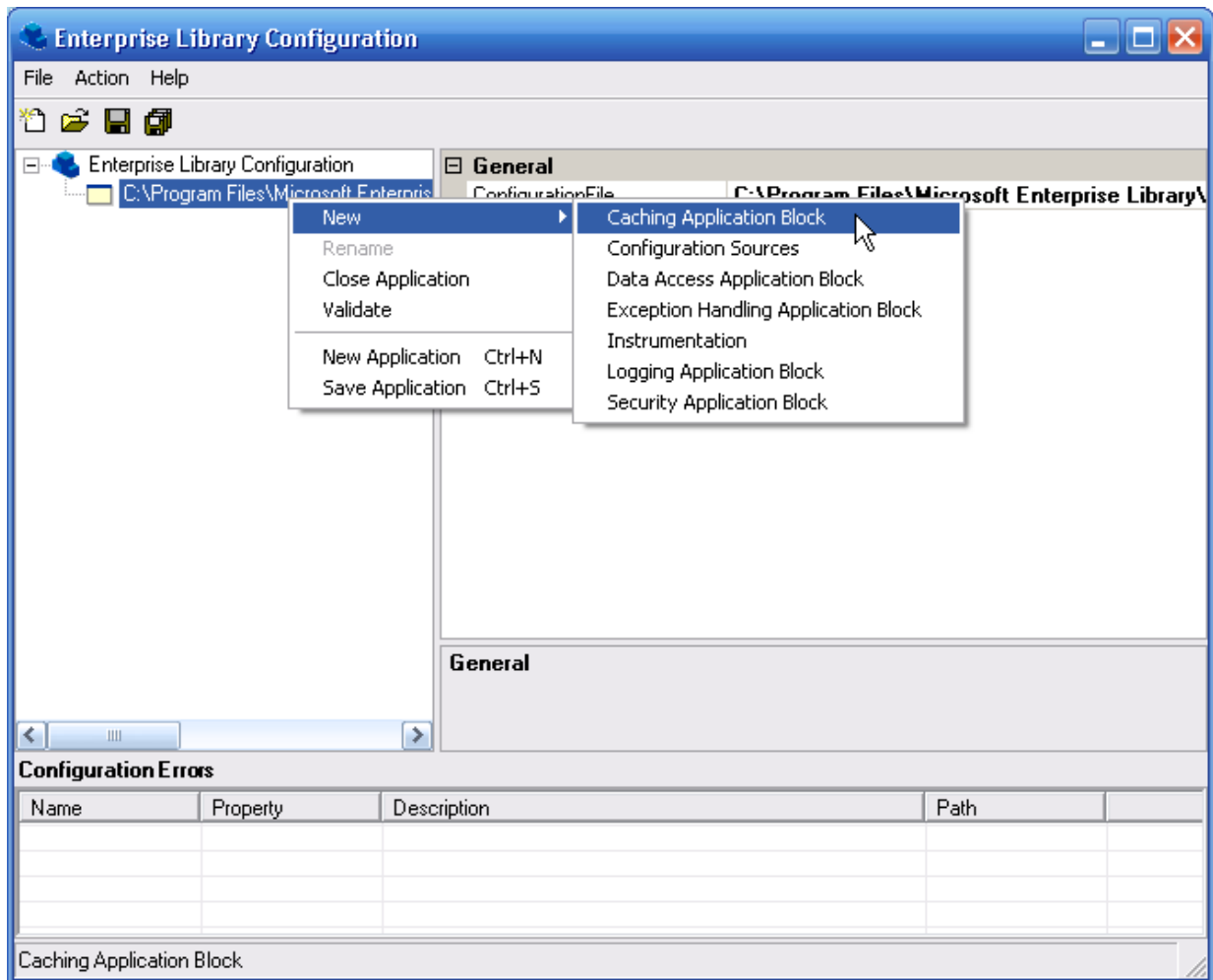
Visual Studio will pass the configuration file (App.config) to the EntLibConfig.exe program as a command line parameter.

4. In the **Open With** dialog, select the newly entered **Enterprise Library Configuration** program and click the **OK** button.

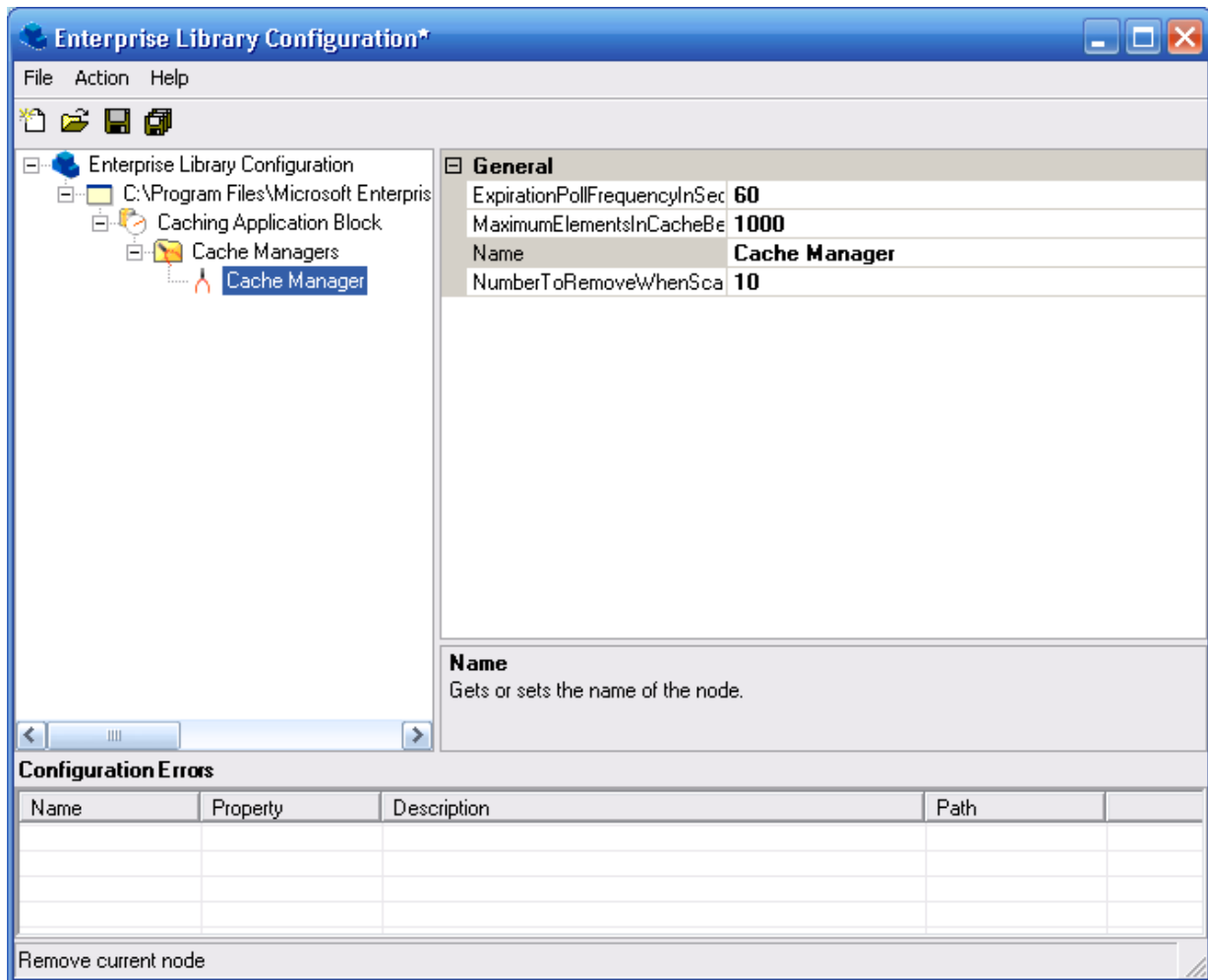


configure the application

1. Right click on the Application and select **New | Caching Application Block**.



2. Select the node **Caching Application Block | Cache Managers | Cache Manager**. Here some of the settings you can change to tune the performance of our cache. For now, leave the settings as they are.



3. Save the application configuration by choosing **File | Save All**. Close the Enterprise Library Configuration Console.
4. Select the **App.config** in the Visual Studio Solution Explorer. Select the **View | Open** menu command.

The App.config file now contains the caching configuration settings you added previously. Notice the backing store is **Null Storage**. That is, the cache will be stored in-memory.

run the application

1. Select the **Debug | Start Without Debugging** menu command to run the application.

Browse through the employees and notice the performance difference for cached photos.

2. Close the application and close Visual Studio .NET.

To check the finished solution, open the [EmployeeBrowser.sln](#) file.

exercise 2: persistent caching

This exercise demonstrates using persistent backing stores and expiration policies of an offline cache.

first step

1. Open the [EmployeeBrowser.sln](#) file.

implement offline caching

1. Select the **EmployeeServices.vb** in the Visual Studio Solution Explorer. Select the **View | Code** menu command. Add the following using statement to the top of the file:

```
Imports Microsoft.Practices.EnterpriseLibrary.Caching.Expirations
```

2. Locate the **GetContactDetails** method, and add the following code:

```
Public Shared Function GetContactDetails() As EmployeesDataSet
    Dim dsEmployees As EmployeesDataSet = Nothing
    ' TODO: Add persistent caching with time-out
    ' Attempt to retrieve from cache
    Dim cache As CacheManager = CacheFactory.GetCacheManager()
    dsEmployees = CType(cache(CACHE_KEY), EmployeesDataSet)
    ' Retrieve from dataProvider if not in cache and Online
    If dsEmployees Is Nothing AndAlso ConnectionManager.IsOnline Then
        Dim dataProvider As New EmployeeDataProvider
        dsEmployees = dataProvider.GetEmployees()
        ' Add to cache. Expire in 2 days
        Dim expiry As AbsoluteTime
        expiry = New AbsoluteTime(New TimeSpan(2, 0, 0, 0))
        cache.Add(CACHE_KEY, dsEmployees, _
            CacheItemPriority.High, Nothing, _
            New ICacheItemExpiration() {expiry})
    End If
    Return dsEmployees
End Function 'GetContactDetails
```

This code will only attempt to contact the database when the application is online.

This contact details are loaded into the cache using an overload of the Add method which allows the specifying of cache item priority, a cache item removal callback (which must be serializable for persistent caches), and a set of expiration policies. In this case, you do not want to allow our users to keep the employee data on their machines for more than 2 days without checking in. This helps to reduce the possibility of an employee taking the contact data to another company, as the

caching infrastructure will remove the contents once they have expired.

3. Modify the **GetEmployeePhoto** method to not attempt to retrieve information from the database when offline (modified code in bold):

```
Public Shared Function GetEmployeePhoto(ByVal employeeId As Guid) As
Bitmap

    Dim photoData As Byte() = Nothing

    ' Attempt to retrieve from cache
    Dim cache As CacheManager = CacheFactory.GetCacheManager()
    photoData = CType(cache(employeeId.ToString()), Byte())

    ' TODO: Retrieve from dataProvider if not in Cache and Online
    If photoData Is Nothing AndAlso ConnectionManager.IsOnline Then
        Dim dataProvider As New EmployeeDataProvider
        photoData = dataProvider.GetEmployeePhotoData(employeeId)
        cache.Add(employeeId.ToString(), photoData)
    End If

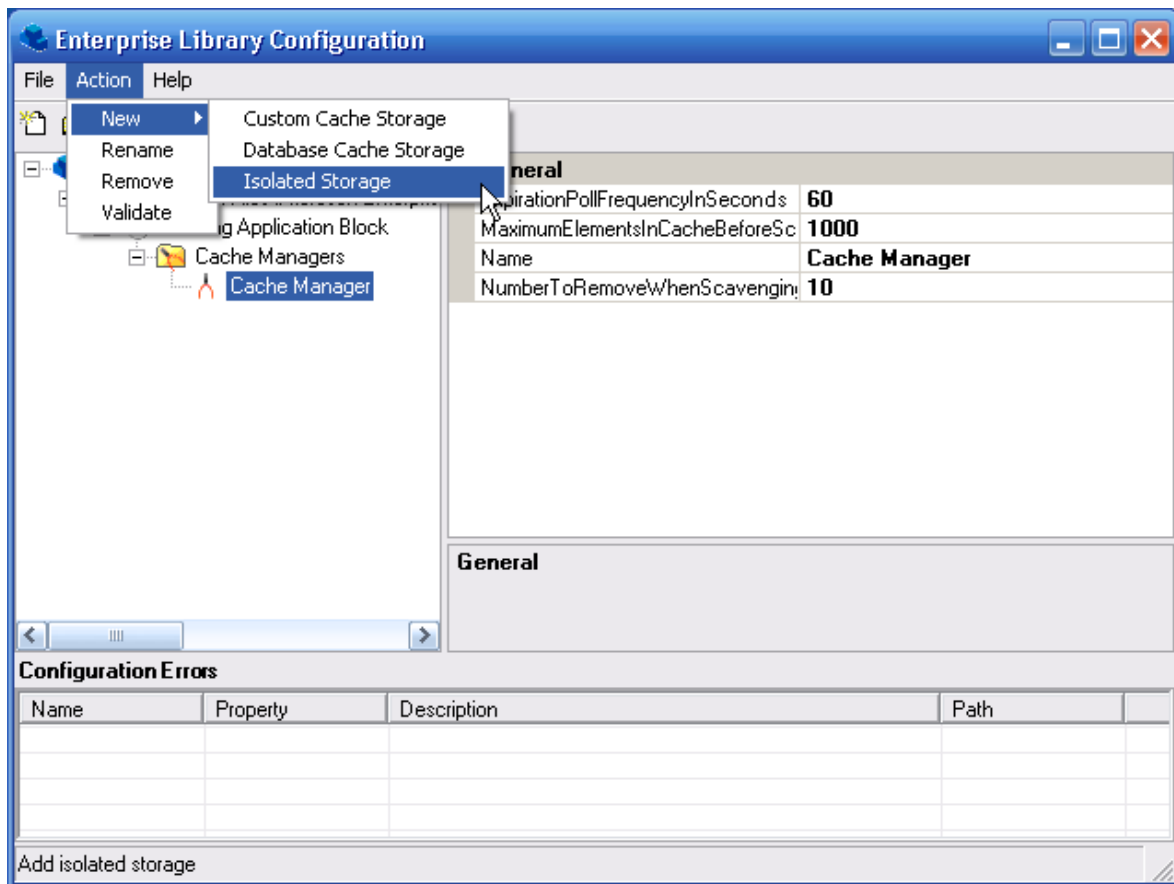
    ' No data found. Nothing to return
    If photoData Is Nothing Then
        Return Nothing
    End If

    ' Convert bytes to Bitmap
    Dim ms As New MemoryStream(photoData)
    Try
        Return New Bitmap(ms)
    Finally
        ms.Dispose()
    End Try

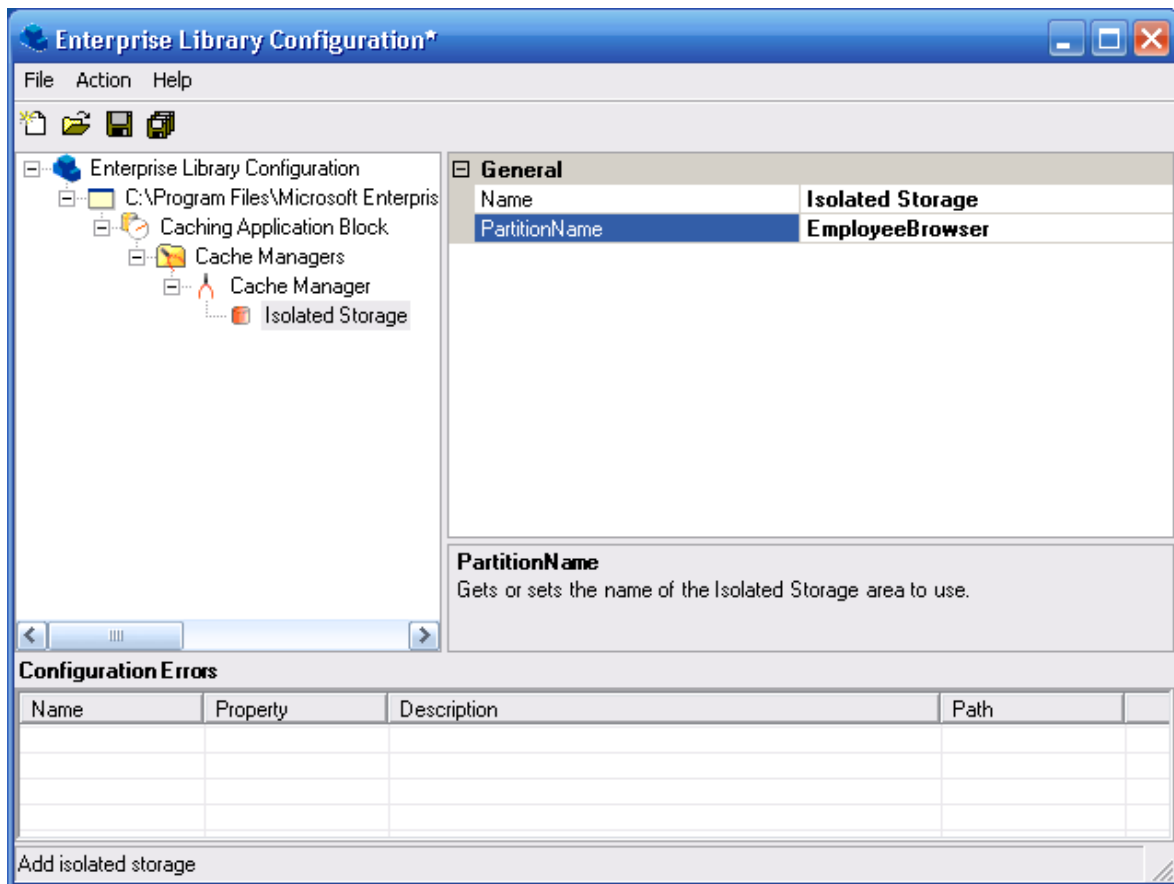
End Function 'GetEmployeePhoto
```

configure the persistent cache

1. Select the **App.config** file in the Solution Explorer. Select the **View | Open With...** menu command. Select **Enterprise Library Configuration** and click the **OK** button.
2. Select the **Caching Application Block | Cache Managers | Cache Manager** node. Select the **Action | New | Isolated Storage** menu command.



3. Set the **PartitionName** property to **EmployeeBrowser**.



The Partition Name allows multiple caches to share the same Isolated storage location.

4. Save the configuration by selecting the menu **File | Save All**. Close the configuration console.

run the application

1. Select the **Debug | Start Without Debugging** menu command to run the application.

Browse to a few of the employees, to load the cache with data, but don't browse all the employees (e.g. don't browse to Fuller, Dodsworth, or Callahan, so that their images are not cached). Close the application when you are finished browsing.

2. Select the **ConnectionManager.vb** file in Solution Explorer. Select the **View | Code** menu command. Modify the **IsOnline** property to simulate the application being started offline.

```
static public bool IsOnline
{
    get { return false; }
}
```

Normally this class would be responsible for testing server connectivity to

*determine whether the client is online and can access the database. See the **Offline Application Block** for ways to do this.*

3. Select the **Debug | Start Without Debugging** menu command to restart the application. The application is now offline and does not access the database. Rather the employee contact details are retrieved from the cache, as are the images of the employees you browsed to while online. For the employees you had not browsed to while online (e.g. Fuller, Dodsworth, and Callahan), no image is displayed.
4. Close the application and close Visual Studio .NET.

To check the finished solution, open the [EmployeeBrowser.sln](#) file.

exercise 3: implement background caching

This exercise demonstrates the background loading of an offline cache.

first step

1. Open the [EmployeeBrowser.sln](#) file.

implement background pre-loading of the cache when online

1. Select the **EmployeeServices.vb** in the Visual Studio Solution Explorer. Select the **View | Code** menu command. Add the following two methods, which will load the cache in the background.

```

Private Shared Sub PopulateCache()
    Dim photoData As Byte() = Nothing
    Dim dsEmployees As EmployeesDataSet

    dsEmployees = GetContactDetails()
    If (dsEmployees Is Nothing) Then
        Return
    End If

    Dim cache As CacheManager = CacheFactory.GetCacheManager()
    For Each employee As EmployeesDataSet.EmployeesRow In
dsEmployees.Employees
        If Not (cache.Contains(employee.EmployeeID.ToString())) Then
            Dim dataProvider As New EmployeeDataProvider
            photoData = dataProvider.GetEmployeePhotoData
(employee.EmployeeID)
            cache.Add(employee.EmployeeID.ToString(), photoData)
        End If
    Next
End Sub 'PopulateCache

Public Shared Sub BeginBackgroundLoad()
    If Not (ConnectionManager.IsOnline) Then
        Return
    End If

    Dim mi As MethodInvoker
    mi = New MethodInvoker(AddressOf PopulateCache)
    mi.BeginInvoke(Nothing, Nothing)
End Sub 'BeginBackgroundLoad

```

The **BeginBackgroundLoad** method uses a delegate to begin the **PopulateCache** method on a background thread, handled by the .NET worker thread implementation.

The **PopulateCache** method iterates through all the employees to retrieve and cache their image. Note that this is actually not a safe activity if the user can add or remove rows in the dataset on another thread (instead it would be better to Select the set of rows, and then iterate through them).

The Caching Block guarantees us thread safety when using the Cache, so it is safe to access from multiple threads at the same time.

2. Select the **MainForm.vb** file in Solution Explorer. Select the **View | Code** menu command. Locate the **MainForm_Load** method and the following code to start the background work.

```

Private Sub EmployeeForm_Load(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles MyBase.Load

    Me.ToolsStripLabel1.Text = ConnectionManager.StatusText

    ' Load data into the 'EmployeesDataSet'.
    Dim tempDataset As EmployeesDataSet =
EmployeeService.GetContactDetails()

    If Not (tempDataset Is Nothing) Then
        Me.EmployeesDataSet.Merge(tempDataset)
    End If

    ' TODO: PopulateCache & BeginBackgroundLoad
EmployeeService.BeginBackgroundLoad()

End Sub

```

run the application

1. Select the **Debug | Start Without Debugging** menu command to run the application.

Don't bother browsing to any other employees, but rather wait for at least 10 seconds then exit the application.

*While the application is online it will attempt to background cache the employee images. The cache is persisted to Isolated Storage, but uses a different **PartitionName** to the previous exercise.*

2. Select the **ConnectionManager.vb** file in Solution Explorer. Select the **View | Code** menu command. Modify the **IsOnline** property to simulate the application being started offline.

```

Public Shared ReadOnly Property IsOnline() As Boolean
    Get
        Return False
    End Get
End Property

```

3. Select the **Debug | Start Without Debugging** menu command to restart the application. The application is now offline and does not access the database, however all employee contact details and images should be cached.

more information

1. For more information on how to use caching in building occasionally connected solutions, see the Mobile Devices course which is part of the [Mastering Industrial Strength .NET](#) series. This uses Enterprise Library to support caching of reference data and Web service requests, using both intrusive application code changes and proxy interception based offline handling techniques.

To check the finished solution, open the [EmployeeBrowser.sln](#) file.

Copyright © 2005 Microsoft. All Rights Reserved.