Caching Application Block Hands-On Lab for Enterprise Library

This walkthrough should act as your guide for learning about the Enterprise Library Caching Application Block and for practicing how to leverage its capabilities in various application contexts.

This lab requires a (local)\SQLEXPRESS instance of Microsoft® SQL Server® or SQL Server Express.

The following three labs are included in this set:

- Lab 1: Using the Caching Block for Performance
- Lab 2: Persistent Caching
- Lab 3: Implementing Background Caching

After completing this lab, you will be able to do the following:

- Improve application performance with the use of in-memory cache.
- Add persistent caching to a Microsoft Windows® application.
- Use background loading to populate a cache.

The estimated completion time for all of the caching labs is **30 minutes**.

Authors

These Hands-On Labs were produced by the following individuals:

- Product/Program Management: Grigori Melnik (Microsoft Corporation)
- Development: Chris Tavares (Microsoft Corporation), Nicolas Botto (Digit Factory), Olaf Conijn (Olaf Conijn BV), Fernando Simonazzi (Clarius Consulting), Erik Renaud (nVentive Inc.)
- Testing: Rick Carr (DCB Software Testing, Inc) plus everyone above
- Documentation: Alex Homer and RoAnn Corbisier (Microsoft Corporation) and Dennis DeWitt (Linda Werner & Associates Inc)

All of the Hands-On Labs use a simplified approach to object generation through Unity and the Enterprise Library container. The recommended approach when developing applications is to generate instances of Enterprise Library objects using dependency injection to inject instances of the required objects into your application classes, thereby realizing all of the advantages that this technique offers.

However, to simplify the examples and make it easier to see the code that uses the features of each of the Enterprise Library Application Blocks, examples in the Hands-On Labs use the simpler approach to resolve Enterprise Library objects from the container by using the **GetInstance** method of the container service locator. You will see this demonstrated in each of the examples.

To learn more about using dependency injection to create instances of Enterprise Library objects, see the documentation installed with the Enterprise Library, or available on MSDN® at http://msdn.microsoft.com/entlib/.

Lab 1: Using the Caching Block for Performance

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

To begin this exercise, open the EmployeeBrowser.sln file located in the ex01\begin folder.

To populate the QuickStarts database with employee data

1. Run the batch file **SetCachingHOL.bat**, which can be found in the lab directory: labs\cs\Caching\setup.

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

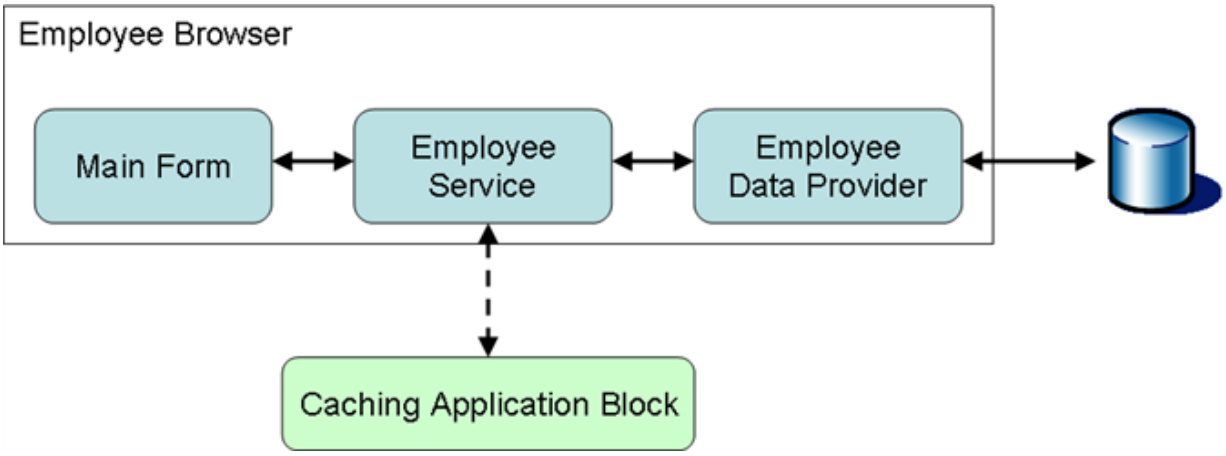
> The database will be installed into the **(local)\SQLEXPRESS** instance.

To review the application

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

   Although the application loads all employee records at startup, it does not 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 impede startup performance.

2. Select the **MainForm.cs** file in the **EmployeeBrowser** project. On the Visual Studio® **View** menu select **Code** 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 figure below.



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

3. Select the **EmployeeDataProvider.cs** file in Solution Explorer. On the Visual Studio **View** menu select **Code** and locate the **GetEmployeePhotoData** method.

   The **GetEmployeePhotoData** method adds a one-second delay to the call to simulate slow database access.

4. On the Visual Studio **Debug** menu, select **Start Without Debugging** or press Ctrl-F5 to run the application.

   Browse through the employees.

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

To implement caching in the EmployeeService class

1. Select the **EmployeeBrowser** project. On the Visual Studio **Project** menu select **Add Reference…**. Select the **Browse** tab and select the following assemblies, located in the Enterprise Library **bin** folder (typically C:\Program Files\Microsoft Enterprise Library 5.0\Bin):
   - Microsoft.Practices.EnterpriseLibrary.Caching.dll
   - Microsoft.Practices.EnterpriseLibrary.Common.dll
   - Microsoft.Practices.ServiceLocation.dll
2. Select the **EmployeeService.cs** file in the Solution Explorer. On the Visual Studio **View** menu select **Code**.
3. Add the following namespace inclusions to the list of namespaces at the beginning of the file:

```
using Microsoft.Practices.EnterpriseLibrary.Caching;
using Microsoft.Practices.EnterpriseLibrary.Common.Configuration;
```

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

```
public static Bitmap GetEmployeePhoto(Guid employeeId)
{
  byte[] photoData = null;

  // TODO: Add Caching of Photo

  // Attempt to retrieve from cache
  // Note that you should consider storing the reference to the cache
  // manager in a member variable and reusing it rather than resolving
  // it every time you use it.
  ICacheManager cache
      = EnterpriseLibraryContainer.Current.GetInstance<ICacheManager>() ;
  photoData = (byte[])cache[employeeId.ToString()];

  // Retrieve from dataProvider if not in Cache
```

```
  if (photoData == null)
  {
     EmployeeDataProvider dataProvider = new EmployeeDataProvider();
     photoData = dataProvider.GetEmployeePhotoData(employeeId);
     cache.Add(employeeId.ToString(), photoData);
  }

  // No data found.
  if (photoData == null)
     return null;

  // Convert bytes to Bitmap
  using (MemoryStream ms = new MemoryStream(photoData))
  {
     return new Bitmap(ms);
  }
}
```

This code uses the **GetInstance** method of the service locator, in conjunction with the Enterprise Library container implemented by Unity, to obtain an instance of the **CacheManager** class. This is automatically registered in the container as a singleton, so that each time you resolve an instance you will get the same instance.

> If you had configured more than one cache manager in your application, you would specify the name of the instance you require as a parameter to the **GetInstance** method.

This cache can be just in memory, or it can be backed up to a physical storage medium, depending on the 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 highlighted code to the **ClearCache** method, to allow the form to request the service to get new data.

```
public static void ClearCache()
{
  // TODO: Clear Cache
  ICacheManager cache
     = EnterpriseLibraryContainer.Current.GetInstance<ICacheManager>();
  cache.Flush();
}
```

This method will remove all items from the cache.

The next step is to use the Enterprise Library configuration editor to configure your application. You may either start the stand-alone version of the tool or the version integrated with the Visual Studio editor.

To configure the application

1. Open the application configuration in the configuration editor by using one of these approaches:
   - Start the stand-alone version of the configuration tool from the Windows Start menu (select **All Programs | Microsoft patterns & practices | Enterprise Library 5.0 | Enterprise Library Configuration**) and open the App.config file.

     There are four versions of the configuration tool available. Choose the one that is appropriate for the version of the Microsoft® .NET Framework you are using.

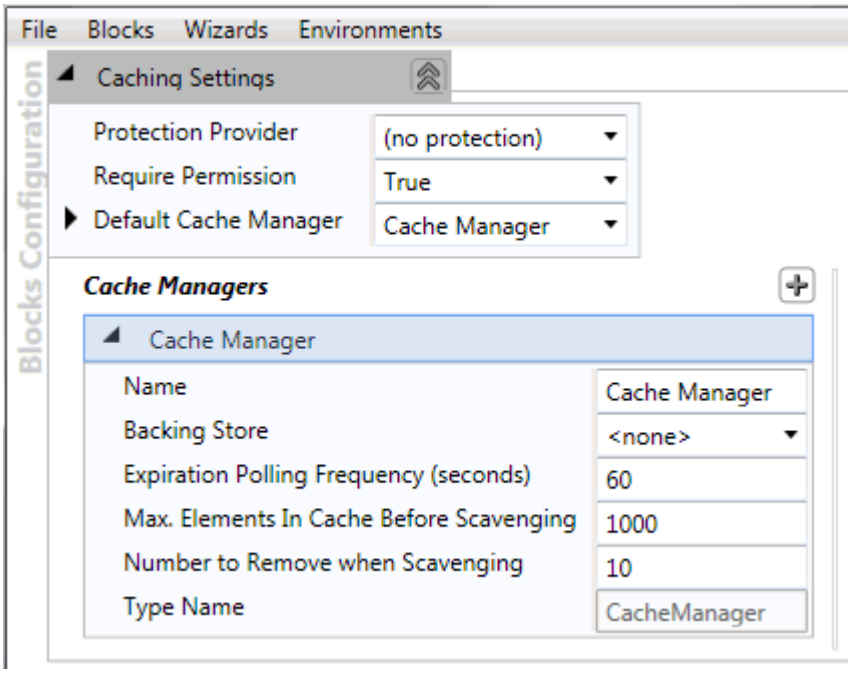
     EntLib Config .NET 3.5

     EntLib Config .NET 3.5 (x86)

     EntLib Config .NET 4

     EntLib Config .NET 4 (x86)

   - In Visual Studio, right-click on the App.config file in Solution Explorer and click **Edit Enterprise Library V5 Configuration**.
2. On the **Blocks** menu select **Add Caching Settings**. This adds a section that contains a default cache manager to the configuration file.
3. Click the chevron to the right of the **Caching Settings** title to view the settings for this section, as shown in the following figure. Click the expander arrow to the left of the **Cache Manager** node to show the settings for the cache manager. These are some of the settings that you can change to tune the performance of your cache. For now, leave the default settings.



4. On the **File** menu select **Save** to save the application configuration file.

   The App.config file now contains the caching configuration settings you added. The default cache manager is automatically configured to use the **Null Backing Store**, which means that the cache will be stored in memory.

To run the application

1. On the Visual Studio **Debug** menu select **Start Without Debugging** or press Ctrl-F5 to run the application.
2. Browse through the employees and notice the performance difference for cached photos.

To verify that you have completed the exercise correctly, you can use the solution provided in the ex01\end folder.

Lab 2: Persistent Caching

This lab demonstrates the use of persistent backing stores and expiration policies for an offline cache.

To begin this exercise, open the EmployeeBrowser.sln file located in the ex02\begin folder.

To implement offline caching

1. Select the **EmployeeService.cs** in the Visual Studio Solution Explorer. On the Visual Studio **View** menu select **Code**. Add the following **using** statement to the beginning of the file:

```
using Microsoft.Practices.EnterpriseLibrary.Caching.Expirations;
```

2. Locate the **GetContactDetails** method, and add the following code, (the code you must insert is in bold and highlighted here):

```
public static EmployeesDataSet GetContactDetails()
{
  EmployeesDataSet dsEmployees = null;

  // TODO: Add persistent caching with time-out

  // Attempt to retrieve from cache
  ICacheManager cache
      = EnterpriseLibraryContainer.Current.GetInstance<ICacheManager>();
  dsEmployees = (EmployeesDataSet)cache[CACHE_KEY];

  // Retrieve from dataProvider if not in Cache and Online
  if (dsEmployees == null && ConnectionManager.IsOnline)
  {
    EmployeeDataProvider dataProvider = new EmployeeDataProvider();
    dsEmployees = dataProvider.GetEmployees();

    // Expire in 2 days
    AbsoluteTime expiry = new AbsoluteTime(new TimeSpan(2, 0, 0, 0));
    cache.Add(CACHE_KEY, dsEmployees,
              CacheItemPriority.High, null,
              new ICacheItemExpiration[] { expiry });
  }

  return dsEmployees;
}
```

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

The contact details are loaded into the cache using an overload of the **Add** method which allows you to specify a cache item priority, a cache item removal callback (which must be serializable for persistent caches), and a set of expiration policies. In this case, your users are not allowed to keep the employee data on their machines for more than two 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 so that it does not attempt to retrieve information from the database when offline. Insert the highlighted code.

```
public static Bitmap GetEmployeePhoto(Guid employeeId)
{
  byte[] photoData = null;

  // Attempt to retrieve from cache
  ICacheManager cache
     = EnterpriseLibraryContainer.Current.GetInstance<ICacheManager>();
  photoData = (byte[])cache[employeeId.ToString()];

  // TODO: Retrieve from dataProvider if not in Cache and Online
  if (photoData == null && ConnectionManager.IsOnline)
  {
    EmployeeDataProvider dataProvider = new EmployeeDataProvider();
    photoData = dataProvider.GetEmployeePhotoData(employeeId);
    cache.Add(employeeId.ToString(), photoData);
  }

  // No data found.
  if (photoData == null)
      return null;

  // Convert bytes to Bitmap
  using (MemoryStream ms = new MemoryStream(photoData))
  {
    return new Bitmap(ms);
  }
}
```

To configure the persistent cache

1. Open the application configuration in the configuration editor using one of these approaches:
   - Start the stand-alone version of the tool from the Windows Start menu (select **All Programs | Microsoft patterns & practices | Enterprise Library 5.0 | Enterprise Library Configuration**) and open the App.config file.

     There are four versions of the configuration tool available. Choose the one that is appropriate for the version of the Microsoft® .NET Framework you are using.

     EntLib Config .NET 3.5

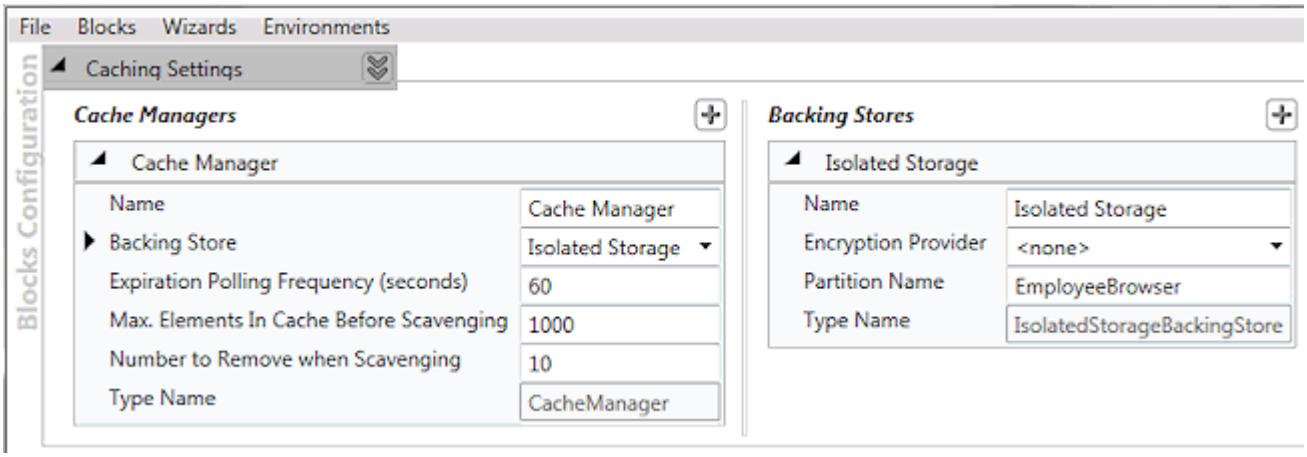
     EntLib Config .NET 3.5 (x86)

     EntLib Config .NET 4

EntLib Config .NET 4 (x86)

- In Visual Studio, right-click on the App.config file in Solution Explorer and click **Edit Enterprise Library V5 Configuration**.

2. Click the expander arrow at the left of the **Caching Settings** title to open the section containing the cache manager.
3. In the **Backing Stores** column, click the plus-sign icon, point to **Add Backing Stores**, and select **Add Isolated Storage Cache Store**. This adds an isolated storage backing store item to the configuration.
4. Set the **Partition Name** property of the isolated storage backing store item to **EmployeeBrowser**, as shown in the following figure.



The partition name allows multiple caches to use isolated storage without overwriting or interfering with each other's cache.

5. Set the **Backing Store** property of the **Cache Manager** by selecting the **Isolated Storage** backing store in the drop-down list.
6. On the **File** menu select **Save** to save the configuration file.

To run the application

1. On the Visual Studio **Debug** menu select **Start Without Debugging** to run the application.

   Browse to a few of the employees in order to load the cache with data, but do not browse all the employees. For example, do not 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.cs** file in Solution Explorer. On the Visual Studio **View** menu select **Code**. Modify the **IsOnline** property with the following highlighted code 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.

3. On the Visual Studio **Debug** menu select **Start Without Debugging** or press Ctrl-F5 to restart the application. The application is now offline and does not access the database. 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 (for example, Fuller, Dodsworth, and Callahan), no image is displayed.
4. Close the application and close Visual Studio.

To verify that you have completed the exercise correctly, you can use the solution provided in the ex02\end folder.

Lab 3: Implementing Background Caching

This lab demonstrates the background loading of an offline cache.

To begin this exercise, open the EmployeeBrowser.sln file located in the ex03\begin folder.

To implement background preloading of the cache when online

1. Select the **EmployeeService.cs** in the Visual Studio Solution Explorer. On the Visual Studio **View** menu select **Code**. Add the following two methods, which will load the cache in the background.

```
private static void PopulateCache()
{
  byte[] photoData = null;

  EmployeesDataSet dsEmployees = GetContactDetails();

  if (dsEmployees == null)
         return;

  ICacheManager cache
      = EnterpriseLibraryContainer.Current.GetInstance<ICacheManager>();

  foreach (EmployeesDataSet.EmployeesRow employee in dsEmployees.Employees)
  {
    if (!cache.Contains(employee.EmployeeID.ToString()))
    {
      EmployeeDataProvider dataProvider = new EmployeeDataProvider();
      photoData = dataProvider.GetEmployeePhotoData(employee.EmployeeID);
      cache.Add(employee.EmployeeID.ToString(), photoData);
    }
  }
}

private delegate void PopulateCacheDelegate();

public static void BeginBackgroundLoad()
{
  if (!ConnectionManager.IsOnline)
         return;

  PopulateCacheDelegate mi = new PopulateCacheDelegate(PopulateCache);
  mi.BeginInvoke(null, null);
}
```

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

The **PopulateCache** method iterates through all the employees to retrieve and cache their image. This is actually not a safe activity if the user can add or remove rows in the dataset on another thread. It would be safer to select the set of rows, and then iterate through them.

The Caching Application Block guarantees thread safety when using the cache, making it safe to access from multiple threads at the same time.

2. Select the **MainForm.cs** file in Solution Explorer. On the Visual Studio **View** menu select **Code**. Locate the **MainForm_Load** method and add the following highlighted code, which starts the background work at run time.

```
private void MainForm_Load(object sender, EventArgs e)
{
  this.ToolStripLabel1.Text = ConnectionManager.StatusText;

  // Load data into the 'EmployeesDataSet'.
  EmployeesDataSet tempDataset = EmployeeService.GetContactDetails();

  if (tempDataset != null)
      this.EmployeesDataSet.Merge(tempDataset);

  // TODO: Start loading cache in the background
  EmployeeService.BeginBackgroundLoad();
}
```

To run the application

1. On the Visual Studio **Debug** menu select **Start Without Debugging** to run the application.

   Do not browse to any other employees. 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, with its own **PartitionName** to differentiate it from the cache for the previous exercise.

2. Select the **ConnectionManager.cs** file in Solution Explorer. On the Visual Studio **View** menu select **Code**. Modify the **IsOnline** property as shown in the following highlighted code. This will simulate starting the application offline.

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

3. On the Visual Studio **Debug** menu select the **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.

To verify that you have completed the exercise correctly, you can use the solution provided in the ex03\end folder.

Copyright