Pro Sync Framework

Copyright © 2009 by Rituraj Singh and Joydip Kanjilal

ISBN-13 (pbk): 978-1-4302-1005-4

ISBN-13 (electronic): 978-1-4302-1006-1

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Ewan Buckingham
Technical Reviewer: Todd Meister
Editorial Board: Clay Andres, Steve Anglin, Mark Beckner, Ewan Buckingham, Tony Campbell, Gary Cornell, Jonathan Gennick, Michelle Lowman, Matthew Moodie, Jeffrey Pepper, Frank Pohlmann, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Project Manager: Beth Christmas
Copy Editor: Nancy Sixsmith
Associate Production Director: Kari Brooks-Copony
Production Editor: Ellie Fountain
Compositor: Susan Glinert
Proofreader: Nancy Bell
Indexer: Brenda Miller
Artist: Kinetic Publishing Services, LLC
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at `http://www.apress.com/info/bulksales`.

The source code for this book is available to readers at `http://www.apress.com`.

CHAPTER 1

Introduction to the Microsoft Sync Framework

Microsoft's answer to a synchronization platform for offline data accessibility is here. The Microsoft Sync Framework (MSF) is a comprehensive framework for synchronizing offline data with its online counterpart. Using this framework, you can build applications that can synchronize data from any data store using any protocol over any type of network. It is independent of the protocol used and the data store that contains the data to be synchronized.

Microsoft says, "Microsoft Sync Framework is a comprehensive synchronization platform that enables collaboration and offline access for applications, services, and devices. It features technologies and tools that enable roaming, sharing, and taking data offline. Using Microsoft Sync Framework, developers can build sync ecosystems that integrate any application, with any data from any store using any protocol over any network."

In this chapter you will learn the following:

- The features and benefits of using the Microsoft Sync Framework and how it can resolve common problems of synchronization
- How to install the Sync Framework
- Different components of the Sync Framework

Benefits of Synchronization

Before we dig into the Sync Framework, it's important to understand what synchronization is and why it is required.

In its simple form, *synchronization* can be described as the process of bringing together two end points or data stores. When the contents of the two data stores are the same, they are known to be *in sync* with each other. For example, if you want to synchronize two databases manually, you do the following (this isn't the best way but it explains the concept in a simple way):

1. Determine changes in the source database.
2. Send the changes to the destination database.
3. Apply the source's changes to the destination database.
4. Repeat the previous steps by swapping the source and destination databases.

Why would you need to build synchronization for an application, store, or services? The following lists some of its benefits:

- *Takes the application, store, or services offline*: The biggest advantage of synchronization is that it enables you to take your application offline. If you build synchronization into your application, users of the application can interact with their local data stores until they need the items that are not contained in a local repository or until the application is back online.
- *Builds a faster and richer user interface*: Building synchronization into the application allows you to build a richer user interface without worrying about the performance of the application. Because data is usually fetched from a local store, your application can provide faster responses.
- *Reduces the network cost*: Sync-enabled applications or services upload and download only incremental changes, thereby reducing the amount of the data that needs to be sent over the network.

Now that you understand the need for synchronization, the next section examines some issues associated with synchronization and how Sync Framework can resolve them.

Life Before Sync Framework

The importance of Sync Framework can't be understood without discussing problems involved while implementing synchronization, including the following:

- *Storage and application errors and failover handling*: Imagine that you're synchronizing two databases, and an application error such as a connection timeout or a constraint violation occurs while changes to the destination database are applied. What happens to the record that needs to be synchronized? It is the responsibility of the Sync Framework to recover from such application and storage errors.

- *Network failure*: Imagine the same example of synchronizing two databases. What happens if one of the databases is downloading changes from another and suddenly the Internet connection goes down? Sync Framework can recover the network failures.

- *Conflict detection*: A conflict is said to occur if the same item was modified at both end points at the same time. The Sync Framework should be able to detect the conflicts and provide a mechanism to resolve or log the conflicts.

Obviously, you can write the code to implement the solution to the problems listed previously. But wouldn't it be nice to have a framework with built-in capabilities to handle all these problems so you can concentrate on implementing business rules instead? Well, allow us to introduce the Microsoft Sync Framework, which is the framework offered by Microsoft to implement synchronization between any data store over any protocol over any network.

Why the Microsoft Sync Framework?

The capability of the synchronization to support offline and online data is the greatest benefit that this framework offers. The major goal of the Sync Framework is to enable any data source to be integrated in the data synchronization, regardless of its type. Imagine a scenario in which the sales data entered in offline mode by a salesperson needs to be in sync with the data available online. The salesperson might be using a smart phone or a personal digital assistant (PDA) to connect to the remote corporate network to retrieve the latest data and information. It is critical for the company to ensure that the data and information are consistent and in sync, regardless of the modes in which they are looked at. This is where the Sync Framework comes into play.

The Sync Framework documentation states, "Microsoft Sync Services for ADO.NET lets you synchronize data from different sources over two-tier, N-tier, and service-based architectures. The Sync Services for ADO.NET API for client and server synchronization provides a set of components to synchronize data between data services and a local store, instead of only replicating a database and its schema."

Here are some of the salient features of the Microsoft Sync Framework:

- A powerful synchronization model independent of the underlying data store, data type, or protocol in use
- Extensible provider model
- Built-in support for filters and data conflict resolution
- Sync support for file systems, databases, and simple sharing extensions (SSEs) such as Really Simple Syndication (RSS*)* and Atom feeds
- Supports peer-to-peer and hub-and-spoke topologies
- Works with any number of end points over any configuration
- Can be used from managed as well as unmanaged code

Installing Microsoft Sync Framework

At the time of writing, Microsoft Sync Framework 1.0 was the latest release. You can get a copy of the Microsoft Sync Framework software development kit (SDK) in three ways:

- Sync Framework ships with Microsoft SQL Server 2008.
- Sync Framework ships with Microsoft Visual Studio 2008 Service Pack 1.
- You can download the Sync Framework from the Microsoft download page: `www.microsoft.com/downloads/details.aspx?FamilyId=C88BA2D1-CEF3-4149-➥ B301-9B056E7FB1E6&displaylang=en`.

The Microsoft Sync Framework SDK is available in 11 languages:

- Chinese (Hong Kong)
- Chinese (Simplified)
- English
- German
- French
- Italian

- Japanese
- Korean
- Portuguese
- Russian
- Spanish

The Microsoft Sync Framework SDK is available for three types of processors:

- AMD64
- IA64
- X86

You can download the appropriate version from the Microsoft download page. For example, if you want to download the Sync Framework for the X86 processor in English, select `SyncSetup_es.x86.zip`.

Before installing the Sync Framework SDK, you must uninstall all previous versions of Microsoft Sync Framework and sync services for ADO.NET 2.0.

The Sync Framework is free on Windows and Windows mobile devices. It supports the following versions of Windows platforms:

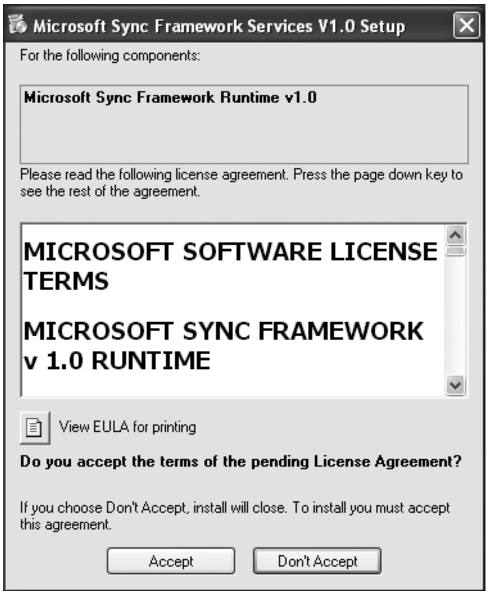
- Windows Server 2003
- Windows Vista
- Windows XP
- Windows 2000 Service Pack 3
- Windows Vista Business
- Windows Vista Business 64-bit edition
- Windows Vista Enterprise
- Windows Vista Enterprise 64-bit edition
- Windows Vista Home Basic
- Windows Vista Home Basic 64-bit edition

- Windows Vista Home Premium
- Windows Vista Home Premium 64-bit edition
- Windows Vista Starter
- Windows Vista Ultimate
- Windows Vista Ultimate 64-bit edition
- Windows XP Service Pack 2

Support for other frameworks can be obtained from other commercial licensing and portal kits. For developers who want to implement the Sync Framework on non-Windows platforms, Microsoft is licensing the specifications and source code porting kit. Visit the following MSDN page for more information on licensing: `http://msdn.microsoft.com/en-us/sync/bb887636.aspx`.

To install Sync Framework, follow these steps:

1. Double-click the setup file to start the installation process. Figure 1-1 shows what the screen looks like.



Microsoft Sync Framework Services V1.0 Setup

For the following components:

Microsoft Sync Framework Runtime v1.0

Please read the following license agreement. Press the page down key to see the rest of the agreement.

MICROSOFT SOFTWARE LICENSE TERMS

MICROSOFT SYNC FRAMEWORK v 1.0 RUNTIME

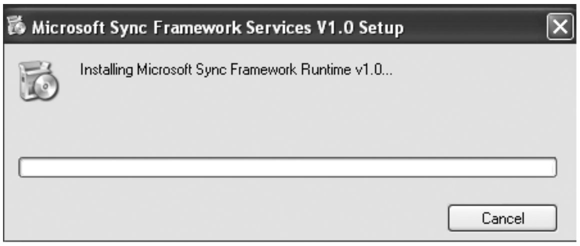
View EULA for printing

Do you accept the terms of the pending License Agreement?

If you choose Don't Accept, install will close. To install you must accept this agreement.

Figure 1-1. *Microsoft Sync Framework setup*

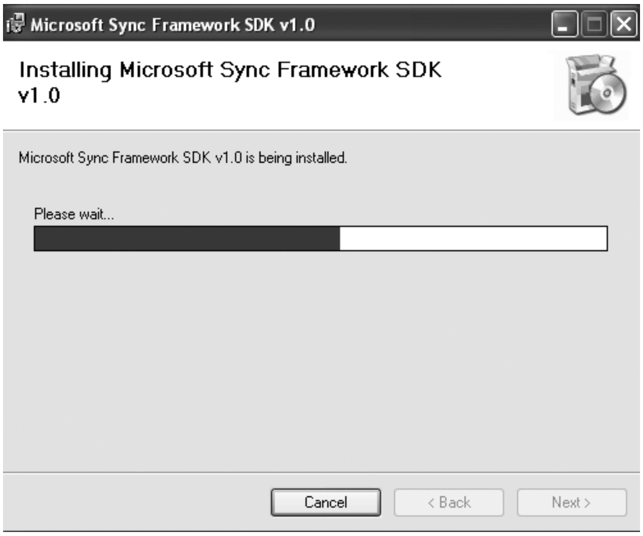
2. Click Accept to accept the license terms and conditions. The installation process for the Sync Framework runtime starts (see Figure 1-2).



Installing Microsoft Sync Framework Runtime v1.0...

Figure 1-2. *Installing the Microsoft Sync Framework runtime*

3. After the runtime is installed, the installation process continues with the installation of the Sync Framework SDK (see Figure 1-3).



Installing Microsoft Sync Framework SDK v1.0

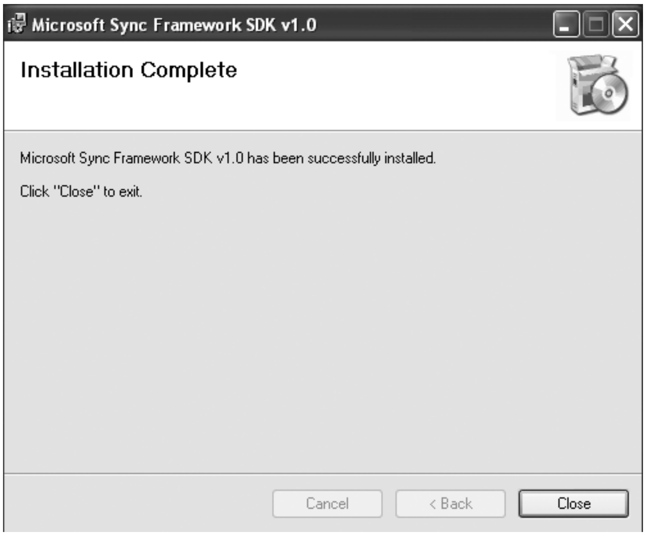
Microsoft Sync Framework SDK v1.0 is being installed.

Please wait...

Figure 1-3. *Installing the Sync Framework SDK*

4. After the installation process is complete, you see the message shown in Figure 1-4.

5. If the Sync Framework is already installed in your system and you want to repair it, follow the preceding steps. When prompted whether you want to repair or uninstall it from your system, select the repair option and then click Finish (see Figure 1-5).

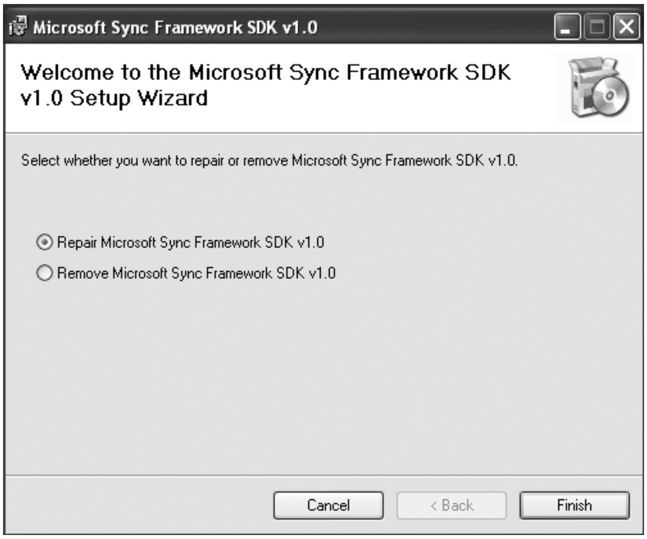


Installation Complete

Microsoft Sync Framework SDK v1.0 has been successfully installed.

Click "Close" to exit.

Figure 1-4. *Completing the Sync Framework installation*



Welcome to the Microsoft Sync Framework SDK v1.0 Setup Wizard

Select whether you want to repair or remove Microsoft Sync Framework SDK v1.0.

[✓ Repair Microsoft Sync Framework SDK v1.0]
Remove Microsoft Sync Framework SDK v1.0

Figure 1-5. *Repairing the Sync Framework*

Core Components

The Sync Framework comes with support for ADO.NET, file systems, RSS, Atom feeds, and even custom data stores. You can divide the components of the Sync Framework into the following three categories:

- *Microsoft Sync Framework runtime*: This SDK allows developers to use the built-in sync providers and create their own sync providers.
- *Metadata services*: Provide the necessary infrastructure to store the sync metadata. Sync metadata is used by the Sync Framework runtime during the synchronization. The Sync Framework ships with the built-in Microsoft SQL Server Compact Edition (CE) that can be used to store the metadata.
- *Built-in providers*: The Sync Framework ships with the following three built-in providers:
  - Synchronization services for ADO.NET provide offline and collaboration support for ADO.NET enabled data stores.
  - Synchronization services for SSEs are the built-in providers for synchronizing RSS and Atom feeds.
  - Synchronization services for file systems are the built-in providers for synchronizing files and folders on Win32-compatible file systems.

Synchronization Providers

You can create custom providers using the Sync Framework runtime to facilitate the support for data synchronization the way you want.

Sync Framework Participants

A discussion on the Sync Framework is incomplete without understanding the participants. A *participant* refers to the location in which the data to be synchronized is retrieved. Note that any one type of participant can synchronize data from any other type of participant. Participants can be one of the following types:

- *Full participants*: Devices that have the capability to create new data stores and execute applications on the device itself. Examples of full participants are laptops, tablet PCs, and smart phones. Full participants can also store the sync metadata required for synchronization. Full participants have the capability to synchronize with any other participants.
- *Partial participants*: Devices that can create new data store and store sync metadata like full participants, but can't create or execute new applications. Thumb drives are good examples of partial participants—they can store the data, but can't execute an application. Partial participants can synchronize with full participants, but they can't synchronize directly with other partial participants.

- *Simple participants*: Devices that can't store new data or execute any application; they just provide the information if requested. Some examples of simple participants include RSS feeds and third-party web services such as those offered by Amazon or Google. Simple participants can synchronize only with full participants because they require sync metadata to be stored on full participants.

Synchronization Flow

Synchronization providers enable you to synchronize data between different replicas. Replicas are also known as end points or data stores. The actual data is stored in the replica. You need to have one sync provider for each replica for the replica to synchronize its data with other replicas. A replica synchronizes its data with another replica by establishing a sync session.

As shown in Figure 1-6, synchronization providers communicate with each other using a sync session. The two sync providers are attached to the sync agent, and the sync application initiates the communication between the two providers using the sync agent. The sync agent is responsible for establishing and managing the sync session. Sync providers can receive and apply changes to the replicas. There are two sync providers on the top of the Sync Framework runtime: the source sync provider and destination sync provider, respectively.

After being invoked by a sync agent, the destination sync provider sends its knowledge to the source sync provider. The source provider uses this knowledge to determine the changes and sends its knowledge to the destination. The destination provider compares its knowledge with the source, resolves the conflicts, and then sends the request to the source provider for changed data. The source provider sends the changes to the destination provider, and the destination provider applies the changes to the destination replica.

Within a sync session, synchronization flow is always in one direction. What this means is that the source provider and the destination provider cannot work simultaneously. At any given point within a sync session, information flows between the source and destination replicas or between destination and source replicas, but doesn't flow simultaneously between both. In its simple form, a sync session contains a sync agent and two sync providers. One of the providers is a source provider that sends the changes; the other is a destination provider that receives and applies the changes. Of course, the sync agent controls this flow.

The sync providers shown in Figure 1-6 illustrate a scenario in which the metadata is stored in the built-in metadata store provided by the Sync Framework, which is very easy to use and is built on top of the SQL Server CE. You can also store the metadata inside your own custom store.

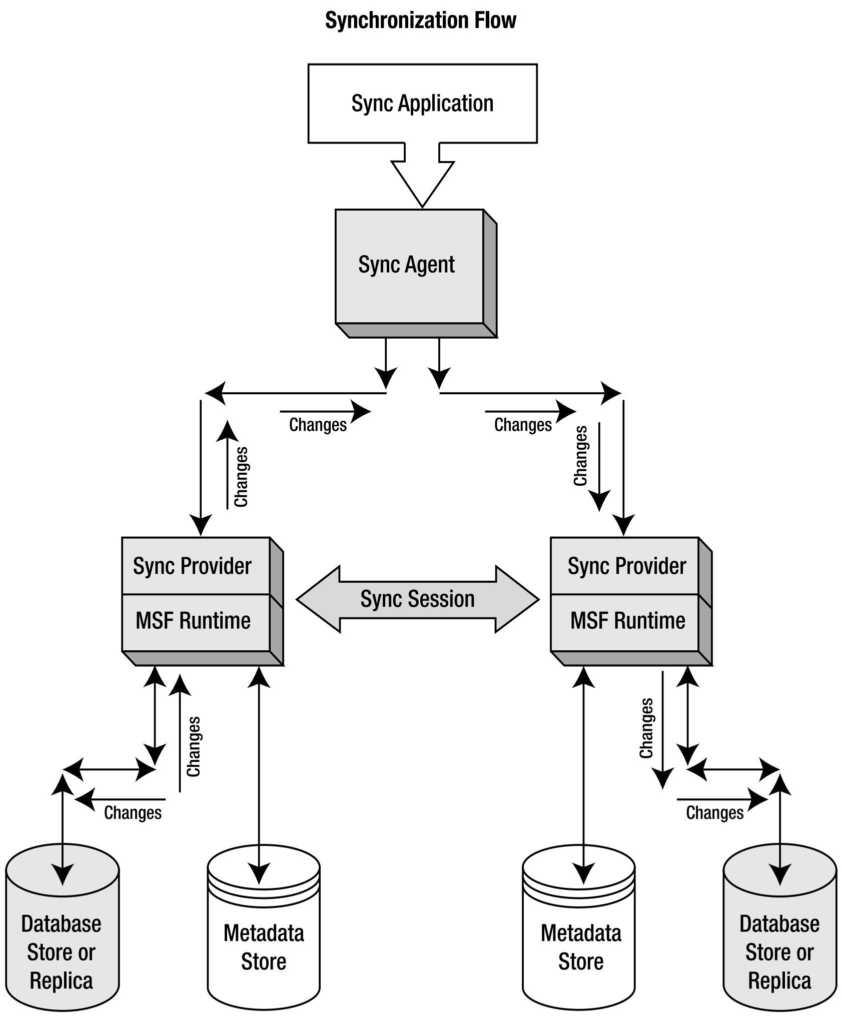


Figure 1-6. *Synchronization flow*

Synchronizing Two Replicas Using Built-in Providers

This section shows you how to synchronize two replicas using the built-in sync provider. The Sync Framework ships with the following three out-of-the-box synchronization providers:

- Synchronization provider for ADO.NET-enabled data sources
- Synchronization provider for files and folders
- Synchronization provider for RSS and Atom feeds

Using the built-in sync providers is very easy. The tasks can be summarized as follows:

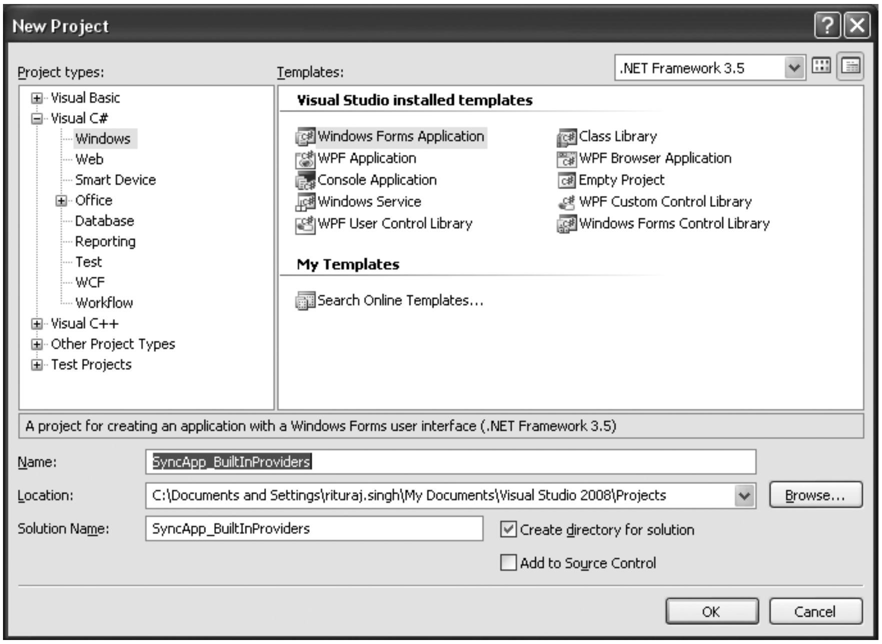
1. Create unique Globally Unique Identifiers (GUIDs) for the source replica and destination replica.
2. Create a source provider by creating a new instance of the built-in provider and attaching the source provider to the source replica.
3. Create a destination provider by creating a new instance of the built-in provider and attaching the destination provider to the destination replica.
4. Create a new instance of a sync agent and attach the source and destination provider to it.
5. Set the direction of the synchronization by using the sync agent. The sync application can now use the sync agent to start synchronization.

Let's now dig into some code. Recall that the file sync provider helps to synchronize the files, folders, and subfolders.

■**Warning** Make sure that you have installed the Sync Framework before trying the following steps!

---

1. Create a new Windows Forms application using Visual Studio 2008 and name it `SyncApp_BuiltInProviders` (see Figure 1-7).
2. Add a reference to `Microsoft.Synchronization` and `Microsoft.Synchronization.Files`. (The Add Reference dialog box shown in Figure 1-8 can be launched by right-clicking the project in Solution Explorer and clicking Add a Reference.)
3. Place a button on form1 and name it `btnSynchronize` with this text: **Synchronize**. Double-click the button to wire an event handler for the button.
4. Place a label on the form with its `Name` as **label1** and text as **Click on Synchronize button to start the synchronization**.
5. Add two folders in your C drive and name them `TestSync1` and `TestSync2`. Create a new text file in `TestSync1` and name it `TestSync1.txt`. Open the file and type some text into it.



A project for creating an application with a Windows Forms user interface (.NET Framework 3.5)

Name: SyncApp_BuiltInProviders

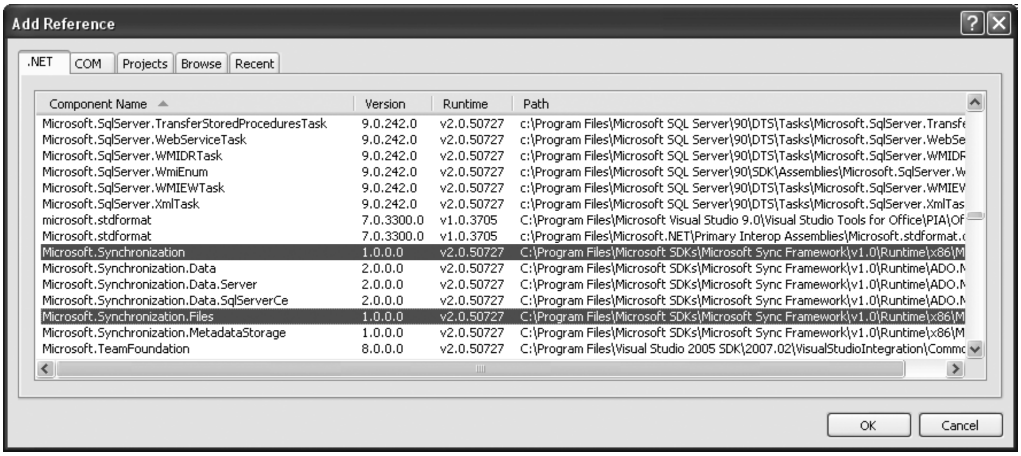
Location: C:\Documents and Settings\rituraj.singh\My Documents\Visual Studio 2008\Projects

Solution Name: SyncApp_BuiltInProviders

[✓] Create directory for solution

[ ] Add to Source Control

Figure 1-7. *Creating a Sync application*



Component Name	Version	Runtime	Path
Microsoft.SqlServer.TransferStoredProceduresTask	9.0.242.0	v2.0.50727	c:\Program Files\Microsoft SQL Server\90\DTS\Tasks\Microsoft.SqlServer.Transfe
Microsoft.SqlServer.WebServiceTask	9.0.242.0	v2.0.50727	c:\Program Files\Microsoft SQL Server\90\DTS\Tasks\Microsoft.SqlServer.WebSe
Microsoft.SqlServer.WMIDRTask	9.0.242.0	v2.0.50727	c:\Program Files\Microsoft SQL Server\90\DTS\Tasks\Microsoft.SqlServer.WMIDF
Microsoft.SqlServer.WmiEnum	9.0.242.0	v2.0.50727	c:\Program Files\Microsoft SQL Server\90\SDK\Assemblies\Microsoft.SqlServer.W
Microsoft.SqlServer.WMIEWTask	9.0.242.0	v2.0.50727	c:\Program Files\Microsoft SQL Server\90\DTS\Tasks\Microsoft.SqlServer.WMIEV
Microsoft.SqlServer.XmlTask	9.0.242.0	v2.0.50727	c:\Program Files\Microsoft SQL Server\90\DTS\Tasks\Microsoft.SqlServer.XmlTas
microsoft.stdformat	7.0.3300.0	v1.0.3705	C:\Program Files\Microsoft Visual Studio 9.0\Visual Studio Tools for Office\PIA\Of
Microsoft.stdformat	7.0.3300.0	v1.0.3705	c:\Program Files\Microsoft.NET\Primary Interop Assemblies\Microsoft.stdformat.c
Microsoft.Synchronization	1.0.0.0	v2.0.50727	C:\Program Files\Microsoft SDKs\Microsoft Sync Framework\v1.0\Runtime\x86\M
Microsoft.Synchronization.Data	2.0.0.0	v2.0.50727	C:\Program Files\Microsoft SDKs\Microsoft Sync Framework\v1.0\Runtime\ADO.N
Microsoft.Synchronization.Data.Server	2.0.0.0	v2.0.50727	C:\Program Files\Microsoft SDKs\Microsoft Sync Framework\v1.0\Runtime\ADO.N
Microsoft.Synchronization.Data.SqlServerCe	2.0.0.0	v2.0.50727	C:\Program Files\Microsoft SDKs\Microsoft Sync Framework\v1.0\Runtime\ADO.N
Microsoft.Synchronization.Files	1.0.0.0	v2.0.50727	C:\Program Files\Microsoft SDKs\Microsoft Sync Framework\v1.0\Runtime\x86\M
Microsoft.Synchronization.MetadataStorage	1.0.0.0	v2.0.50727	C:\Program Files\Microsoft SDKs\Microsoft Sync Framework\v1.0\Runtime\x86\M
Microsoft.TeamFoundation	8.0.0.0	v2.0.50727	C:\Program Files\Visual Studio 2005 SDK\2007.02\VisualStudioIntegration\Commc

Figure 1-8. *Adding a reference to the Sync Framework*

6. In the using block at the top of the `form1.cs`, add the following two namespaces:

```
using Microsoft.Synchronization;
using Microsoft.Synchronization.Files;
```

7. Create two new GUIDs for the source and destination replicas, as shown in the following code:

```
Guid sourceReplicaId;
Guid destReplicaId;
private void Form1_Load(object sender, EventArgs e)
{
    //Assign Unique Guid's to the Replicas
    sourceReplicaId = Guid.NewGuid();
    destReplicaId = Guid.NewGuid();
}
```

8. Write the following code in the `btnSynchronize_Click` event (the main logic of the code resides in this event):

```
private void btnSynchronize_Click(object sender, EventArgs e)
{

   btnSynchronize.Enabled = False;
    //Create the Source and destination Sync provider.
    //Attach the source sync provider to C:\TestSync1 folder and
    //assign it a unique replica guid.
    FileSyncProvider sourceProvider = new FileSyncProvider(sourceReplicaId,
      @"C:\TestSync1");
    //Attach the destination sync provider to C:\TestSync2 folder and
   //assign it a a unique replica guid.
    FileSyncProvider destProvider = new
          FileSyncProvider(destReplicaId, @"C:\TestSync2");

    //syncAgent is the Sync Controller and it co-ordinates the sync session
    SyncOrchestrator syncAgent = new SyncOrchestrator();
    syncAgent.LocalProvider = sourceProvider;
    syncAgent.RemoteProvider = destProvider;
    syncAgent.Synchronize();
    label1.Text = "Synchronizing Finished...";
   btnSynchronize.Enabled = True;
}
```

The code is incredibly simple. We created the source and destination sync providers by passing the replica IDs and the folder name as a parameter. Then we created a new instance of `SyncOrchestrator` (also known as the sync agent) and attached the two providers to it. Finally we called the `Synchronize` method on it.

That's it; we wrote the code to synchronize two replicas (`TestSync1` and `TestSync2`) using two instances of a built-in file sync provider.

Run the application and click the Synchronize button. After a few seconds, the message shown in Figure 1-9 displays.

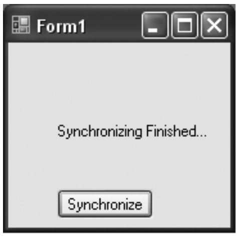


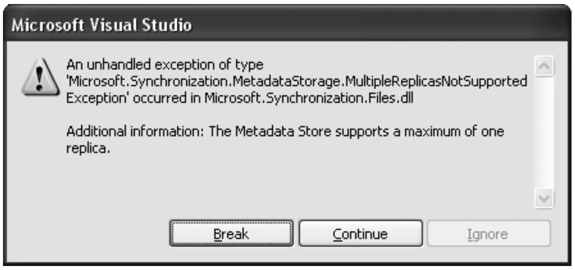
Figure 1- 9. *Synchronization is complete*

Now check the contents of the `C:\TestSync2` folder. The file `TestSync1.txt` is copied into the directory.

Try deleting and updating a file—it works, too! If that's not enough, the file sync provider also does conflict detection and resolution on your behalf. (A *conflict* is said to occur if the same item was modified in both replicas between synchronizations.) We will explore conflicts in detail in Chapters 2 and 3.

Try updating the `TestSync1.txt` with different text in the two folders and then press Synchronize. The `TestSync1.txt` file in both the folders is updated with the latest text (the text you updated last).We will cover the file sync provider in detail in Chapter 5.

However, there is a bug in this code. If you try to run the application a second time, you see the message shown in Figure 1-10.



```
An unhandled exception of type
'Microsoft.Synchronization.MetadataStorage.MultipleReplicasNotSupported
Exception' occurred in Microsoft.Synchronization.Files.dll

Additional information: The Metadata Store supports a maximum of one
replica.
```

Figure 1-10. *Multiple replicas not supported exception*

When you call the `Synchronization` method of the `SyncOrchestrator`, it creates a metadata file in each of the folders (replicas) .This file is called `filesync.metadata`. When you run the application for the second time, it tries to create a new replica with a different GUID.

We can fix this error by persisting the GUIDS in a data store such as a database or file system instead of creating a new GUID every time. Let's try and store the GUIDs into a file within the local file system.

To fix the bug, follow these steps:

1. In the using block at the top of `form1.cs`, add the following namespace:

   ```
   using System.IO;
   ```

2. Create a new `GetReplicaGuid` method in the `form1.cs` and paste the following code in it:

   ```
    private Guid GetReplicaGuid(string GuidPath)
    {
        if (!File.Exists(GuidPath))
        {
            Guid replicaId = Guid.NewGuid();
            using (FileStream fs = File.Open(GuidPath, FileMode.Create))
            using (StreamWriter sw = new StreamWriter(fs))
            {
                sw.WriteLine(replicaId.ToString());
            }
            return replicaId;
        }

        else
        {
            using (FileStream fs = File.Open(GuidPath, FileMode.Open))
            using (StreamReader sr = new StreamReader(fs))
            {
                return new Guid(sr.ReadLine());
            }
        }
    }
   ```

   In the preceding code, we created one more private method called `GetReplicaGuid`, which has an input string parameter that contains the path to store the IDs of the replica. The method checks whether the file containing a replica ID exists. If the file does not exist, it creates a new replica ID and stores it in the path provided by the input parameter. Finally it returns the replica ID. If the file exists, the method returns the replica ID stored in the file.

3. We also need to move the code for creating file sync providers in the `Form` load event instead of creating a new provider every time the Synchronize button is pressed, which will prevent a new instance of the sync provider being created every time you press the button. We need to declare the sync providers as a class-level variable so that we can initialize the sync providers in the `Form` load and use the same instances in the click event of the Synchronize button. To do this, replace the code in the `Form` load event with the following code listing:

```
Guid sourceReplicaId;
Guid destReplicaId;
FileSyncProvider sourceProvider;
FileSyncProvider destProvider;

private void Form1_Load(object sender, EventArgs e)
{
    //Assign Unique Guid's to the Replicas
    sourceReplicaId = GetReplicaGuid(@"C:\TestSync1\ReplicaID");
    destReplicaId = GetReplicaGuid(@"C:\TestSync2\ReplicaID");
    //Create the Source and destination Sync provider.
    //Attach the source sync provider to C:\TestSync1 folder
      and assign it a a unique replica guid.
    sourceProvider = new FileSyncProvider(sourceReplicaId, @"C:\TestSync1");
    //Attach the destination sync provider to C:\TestSync2 folder
      and assign it a a unique replica guid.
    destProvider = new FileSyncProvider(destReplicaId, @"C:\TestSync2");
}
```

4. The last step is to change the code in the click event of the Synchronize button to remove the instantiation of providers:

```
    private void btnSynchronize_Click(object sender, EventArgs e)
    {

        btnSynchronize.Enabled = false;
        //syncAgent is the Sync Controller and it co-ordinates the sync session
        SyncOrchestrator syncAgent = new SyncOrchestrator();
        syncAgent.LocalProvider = sourceProvider;
        syncAgent.RemoteProvider = destProvider;
        syncAgent.Synchronize();

        label1.Text = "Synchronizing Finished...";
        btnSynchronize.Enabled = true;
}
```

The complete code can be found at `Sync\ChapterI\Code\SyncApp_BuiltInProviders`.

Summary

The Microsoft Sync Framework (MSF) is Microsoft's answer for the increasing demand of a common synchronization platform for synchronizing two end points of any type over the network on any protocol. It guarantees an eventual convergence, regardless of common problems such as network failures, storage and application errors, and unhandled conflicts.

The Sync Framework allows developers to focus on their business requirements without paying too much attention to synchronization problems. The Sync Framework ships with built-in providers for synchronizing some very common end points such as files, SSEs such as RSSs and Atom feeds, and ADO.NET-enabled data sources. It also provides developers with the necessary infrastructure to quickly create their own sync providers. Developers can use the built-in SQL Server CE metadata store to store the sync metadata or they can choose their own metadata store.