

## Answer on Question #43977- Programming, C#

1. i finished my c# project with sql server dataBase and i want to make a setup for users to install in there computers now. i want to learn how to make setup like any program we install every day i made a simple program like peachtree accounting a little part of peachtree accounting program all i want to install my project to work in any computer like any program we install and i'm tired searching how to pack your project, how to make setup file, how to make your program work, nothing worked for me so please don't tell me go to file>new>setup wizard, or use inno, or any program if you made program using c# and sql server DB and you publish it for users tell me how step by step and if you didn't made any program and didn't publish it don't say any word please. \*they told me you will find the answer here and they will give you the solution. \*and i'm sorry cuz i really tired trying tutorial and nothing worked.

### Solution.

#### For the database:

1. Create database backup

#### For the application:

In the program code to restore the backup of the database and verify the status of the database. If your database is not installed, restore from backup.

```
// NOTE: Before running this application, run the database sample script that
is
// available in the documentation. The script drops and re-creates the tables
that
// are used in the code, and ensures that synchronization objects are dropped
so that
// Sync Framework can re-create them.
```

```
using System;
using System.IO;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using System.Data.SqlServerCe;
using Microsoft.Synchronization;
using Microsoft.Synchronization.Data;
using Microsoft.Synchronization.Data.SqlServer;
using Microsoft.Synchronization.Data.SqlServerCe;
```

```
namespace Microsoft.Samples.Synchronization
{
    class Program
    {
```

```

static void Main(string[] args)
{
    // Create the connections over which provisioning and
    synchronization
    // are performed. The Utility class handles all functionality
    that is not
    //directly related to synchronization, such as holding connection
    //string information and making changes to the server database.
    SqlConnection serverConn = new
SqlConnection(Utility.ConnStr_SqlSync_Server);
    SqlConnection clientSqlConn = new
SqlConnection(Utility.ConnStr_SqlSync_Client);
    SqlCeConnection clientSqlCelConn = new
SqlCeConnection(Utility.ConnStr_SqlCeSync1);

    // Create a scope named "customer", and add the Customer table to
    the scope.
    // GetDescriptionForTable gets the schema of the table, so that
    tracking
    // tables and triggers can be created for that table.
    DbSyncScopeDescription scopeDesc = new
DbSyncScopeDescription("customer");

    scopeDesc.Tables.Add(

SqlConnectionBuilder.GetDescriptionForTable("Sales.Customer",
serverConn));

    // Create a provisioning object for "customer" and specify that
    // base tables should not be created (They already exist in
    SyncSamplesDb_SqlPeer1).
    SqlSyncScopeProvisioning serverConfig = new
SqlSyncScopeProvisioning(scopeDesc);
    serverConfig.SetCreateTableDefault(DbSyncCreationOption.Skip);

    // Configure the scope and change-tracking infrastructure.
    serverConfig.Apply(serverConn);

    // Retrieve scope information from the server and use the schema
    that is retrieved
    // to provision the SQL Server and SQL Server Compact client
    databases.

    // This database already exists on the server.
    DbSyncScopeDescription clientSqlDesc =
SqlConnectionBuilder.GetDescriptionForScope("customer", serverConn);
    SqlSyncScopeProvisioning clientSqlConfig = new
SqlSyncScopeProvisioning(clientSqlDesc);
    clientSqlConfig.Apply(clientSqlConn);

    // This database does not yet exist.

Utility.DeleteAndRecreateCompactDatabase(Utility.ConnStr_SqlCeSync1, true);

```

```

        DbSyncScopeDescription clientSqlCeDesc =
SqlSyncDescriptionBuilder.GetDescriptionForScope("customer", serverConn);
        SqlCeSyncScopeProvisioning clientSqlCeConfig = new
SqlCeSyncScopeProvisioning(clientSqlCeDesc);
        clientSqlCeConfig.Apply(clientSqlCelConn);

// Initial synchronization sessions.
SampleSyncOrchestrator syncOrchestrator;
SyncOperationStatistics syncStats;

// Data is downloaded from the server to the SQL Server client.
syncOrchestrator = new SampleSyncOrchestrator(
    new SqlSyncProvider("customer", clientSqlConn),
    new SqlSyncProvider("customer", serverConn)
);
syncStats = syncOrchestrator.Synchronize();
syncOrchestrator.DisplayStats(syncStats, "initial");

// Data is downloaded from the SQL Server client to the
// SQL Server Compact client.
syncOrchestrator = new SampleSyncOrchestrator(
    new SqlCeSyncProvider("customer", clientSqlCelConn),
    new SqlSyncProvider("customer", clientSqlConn)
);
syncStats = syncOrchestrator.Synchronize();
syncOrchestrator.DisplayStats(syncStats, "initial");

// Remove the backup file because this application requires a
drop and recreation
// of the sample database, and the backup must be recreated each
time

// the application runs.
Utility.DeleteDatabaseBackup();

// Backup the server database.
Utility.CreateDatabaseBackup();

// Make changes on the server: 1 insert, 1 update, and 1 delete.
Utility.MakeDataChangesOnNode(Utility.ConnStr_SqlSync_Server,
"Customer");

// Synchronize the three changes.
syncOrchestrator = new SampleSyncOrchestrator(
    new SqlCeSyncProvider("customer", clientSqlCelConn),
    new SqlSyncProvider("customer", serverConn)
);
syncStats = syncOrchestrator.Synchronize();
syncOrchestrator.DisplayStats(syncStats, "subsequent");

syncOrchestrator = new SampleSyncOrchestrator(
    new SqlSyncProvider("customer", clientSqlConn),
    new SqlCeSyncProvider("customer", clientSqlCelConn)

```

```

        );
        syncStats = syncOrchestrator.Synchronize();
        syncOrchestrator.DisplayStats(syncStats, "subsequent");

        // Restore the server database from backup. The restored version
        // does not contain the three changes that were just
synchronized.
        Utility.RestoreDatabaseFromBackup();

        // Call the API to update synchronization metadata to reflect
that the database was
        // just restored. The restore stored procedure kills the
connection to the
        // server, so we must re-establish it.
        SqlConnection.ClearPool(serverConn);
        serverConn = new SqlConnection(Utility.ConnStr_SqlSync_Server);
        SqlSyncStoreRestore databaseRestore = new
SqlSyncStoreRestore(serverConn);
        databaseRestore.PerformPostRestoreFixup();

        // Synchronize a final time.
        syncOrchestrator = new SampleSyncOrchestrator(
            new SqlCeSyncProvider("customer", clientSqlCe1Conn),
            new SqlSyncProvider("customer", serverConn)
        );
        syncStats = syncOrchestrator.Synchronize();
        syncOrchestrator.DisplayStats(syncStats, "subsequent");

        syncOrchestrator = new SampleSyncOrchestrator(
            new SqlSyncProvider("customer", clientSqlConn),
            new SqlCeSyncProvider("customer", clientSqlCe1Conn)
        );
        syncStats = syncOrchestrator.Synchronize();
        syncOrchestrator.DisplayStats(syncStats, "subsequent");

        serverConn.Close();
        serverConn.Dispose();
        clientSqlConn.Close();
        clientSqlConn.Dispose();
        clientSqlCe1Conn.Close();
        clientSqlCe1Conn.Dispose();

        Console.WriteLine("\nPress any key to exit.");
        Console.Read();
    }

}

public class SampleSyncOrchestrator : SyncOrchestrator
{
    public SampleSyncOrchestrator(RelationalSyncProvider localProvider,
        RelationalSyncProvider remoteProvider)

```

```

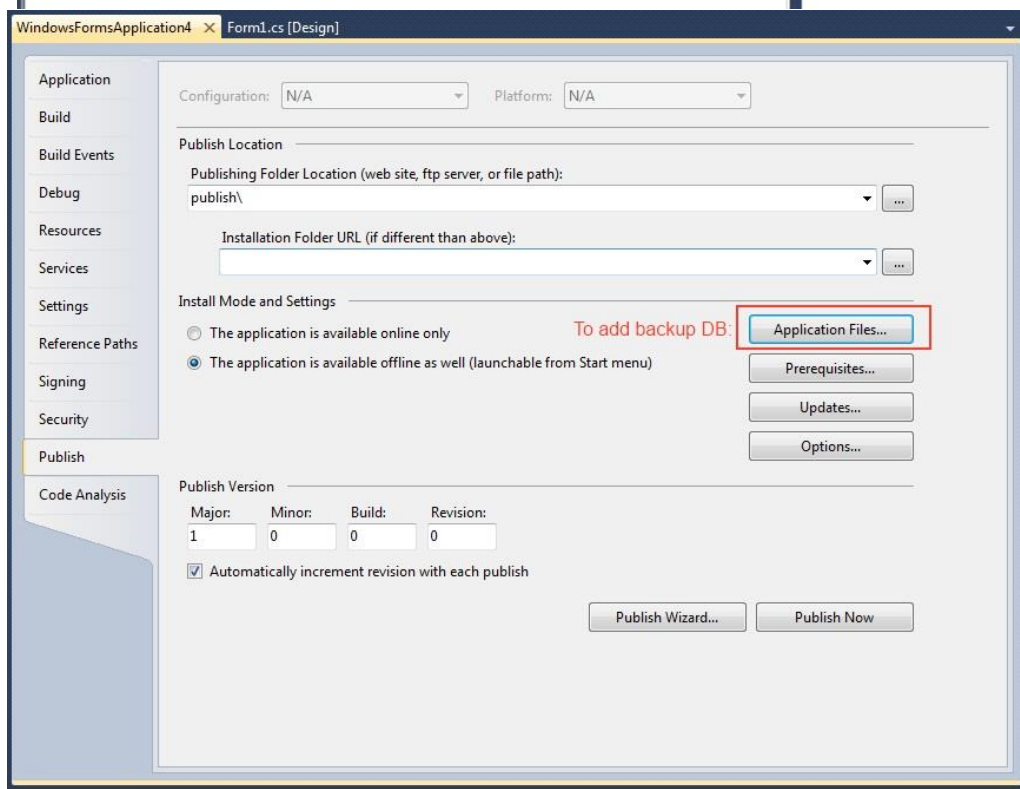
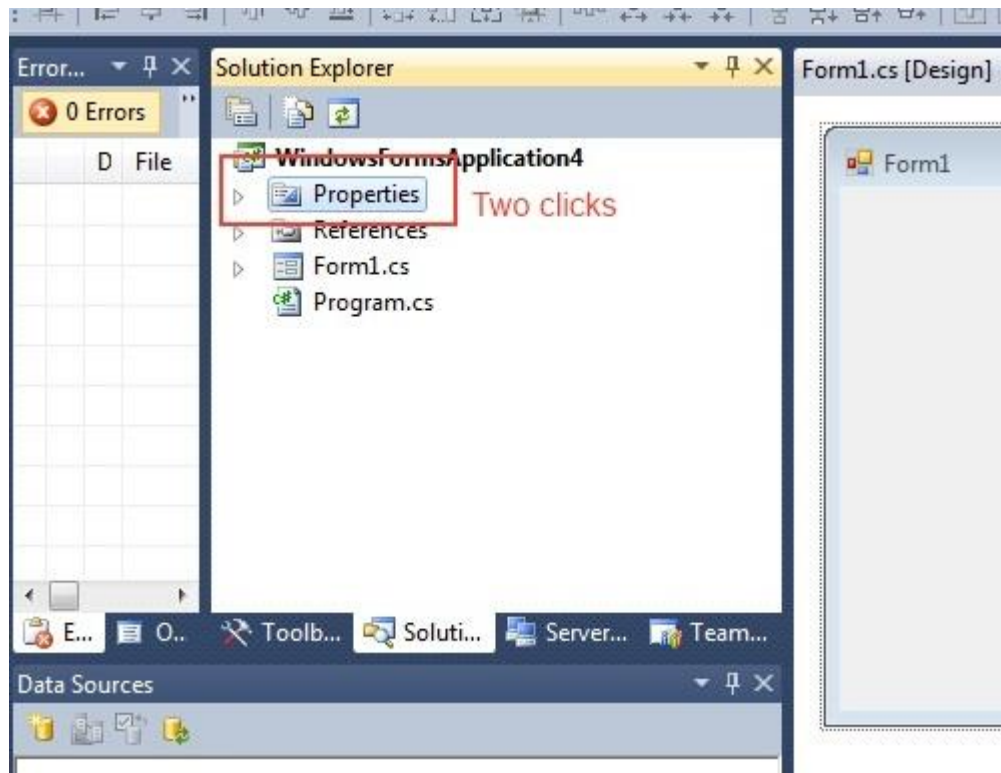
    {
        this.LocalProvider = localProvider;
        this.RemoteProvider = remoteProvider;
        this.Direction = SyncDirectionOrder.UploadAndDownload;
    }

    public void DisplayStats(SyncOperationStatistics syncStatistics,
string syncType)
    {
        Console.WriteLine(String.Empty);
        if (syncType == "initial")
        {
            Console.WriteLine("***** Initial Synchronization *****");
        }
        else if (syncType == "subsequent")
        {
            Console.WriteLine("***** Subsequent Synchronization *****");
        }

        Console.WriteLine("Start Time: " + syncStatistics.SyncStartTime);
        Console.WriteLine("Total Changes Uploaded: " +
syncStatistics.UploadChangesTotal);
        Console.WriteLine("Total Changes Downloaded: " +
syncStatistics.DownloadChangesTotal);
        Console.WriteLine("Complete Time: " +
syncStatistics.SyncEndTime);
        Console.WriteLine(String.Empty);
    }
}
}

```

Follow the steps shown below:



WindowsFormsApplication4 x Form1.cs [Design]

Application

Build

Build Events

Debug

Resources

Services

Settings

Reference Paths

Signing

Security

Publish

Code Analysis

Configuration: N/A Platform: N/A

Publish Location

Publishing Folder Location (web site, ftp server, or file path):  
publish\

Installation Folder URL (if different than above):

Install Mode and Settings

☐ The application is available online only

☒ The application is available offline as well (launchable from Start menu)

Application Files...

Prerequisites...

Updates...

Options...

Publish Version

Major: 1 Minor: 0 Build: 0 Revision: 0

☒ Automatically increment revision with each publish

Publish Wizard... Publish Now

Prerequisites

☒ Create setup program to install prerequisite components

Choose which prerequisites to install:

☐ Microsoft Visual Studio 2010 Report Viewer

☐ Microsoft Visual Studio 2010 Tools for Office Runtime (x86 and x64)

☐ SAP Crystal Reports Runtime Engine for .NET Framework 4.0

☐ SQL Server 2005 Express Edition SP2 (x86)

☐ SQL Server 2008 Express

☐ SQL Server Compact 3.5 SP2

☐ Visual C++ 2010 Runtime Libraries (IA64)

☐ Visual C++ 2010 Runtime Libraries (x64)

[Check Microsoft Update for more redistributable components](#)

Specify the install location for prerequisites

☒ Download prerequisites from the component vendor's web site

☐ Download prerequisites from the same location as my application

☐ Download prerequisites from the following location:

Browse...

OK Cancel

WindowsFormsApplication4 x Form1.cs [Design]

Application

Build

Build Events

Debug

Resources

Services

Settings

Reference Paths

Signing

Security

Publish

Code Analysis

Configuration: N/A Platform: N/A

Publish Location

Publishing Folder Location (web site, ftp server, or file path):  
publish\

Installation Folder URL (if different than above):

Install Mode and Settings

☐ The application is available online only

☒ The application is available offline as well (launchable from Start menu)

Application Files...

Prerequisites...

Updates...

Options...

Optionally review settings

Publish Version

Major: 1 Minor: 0 Build: 0 Revision: 0

☒ Automatically increment revision with each publish

Publish Wizard... Publish Now

WindowsFormsApplication4 x Form1.cs [Design]

Application

Build

Build Events

Debug

Resources

Services

Settings

Reference Paths

Signing

Security

Publish

Code Analysis

Configuration: N/A Platform: N/A

Publish Location

Publishing Folder Location (web site, ftp server, or file path):  
publish\

Installation Folder URL (if different than above):

Install Mode and Settings

☐ The application is available online only

☒ The application is available offline as well (launchable from Start menu)

Application Files...

Prerequisites...

Updates...

Options...




Publish Version

Major: 1 Minor: 0 Build: 0 Revision: 0

☒ Automatically increment revision with each publish

Publish Wizard... Publish Now



	Application Files	07.07.2014 15:51
	setup	07.07.2014 15:51
	WindowsFormsApplication4	07.07.2014 15:51

<http://www.AssignmentExpert.com/>