

# Embedded Resource

ReadyRoll's SQLCMD package format enables you to embed the deployment of your database within your application code. This can be useful if you don't have direct access to your customer's database environment, as with shrink-wrapped software or distributed application systems.

The db deployment can be performed within your application's "start" event or as part of the application installation process (e.g. using a [custom action in WiX](#)). ReadyRoll SQLCMD packages are smart enough to figure out whether any migration scripts need to be applied and run them, or simply exit if the database is already up-to-date. If any errors occur during deployment, then the deployment will be immediately halted and the [transaction will be rolled back](#). Additionally, the database will be automatically created on the SQL Server on-premise or Azure SQL instance if it does not exist yet.

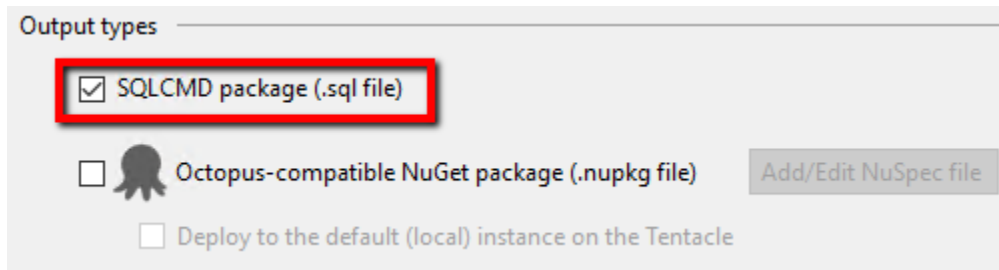
To embed the database deployment within your application, you will need to reference your ReadyRoll database project from your application project (e.g. from an ASP.NET website, Windows service or console application) and add ReadyRoll's output package artifact as an embedded resource. This includes the package file as a resource within your application assembly file, so you don't need to ship the package artifact separately. You can then include some code to execute the database deployment in a convenient part of your application.

Download the sample solution (C#): [MyConsoleAppWithDb.zip](#)

## Database project setup

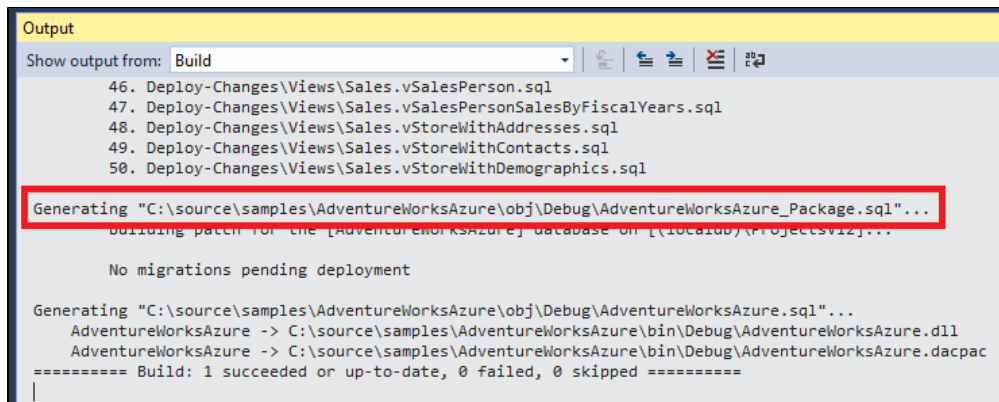
First the ReadyRoll database project needs to be configured to output a package file during build. In Visual Studio, right-click the database project in the *Solution Explorer* and select **Project Properties**.

On the *Project Settings* tab under Outputs, check the **SQLCMD package (.sql file)** option.



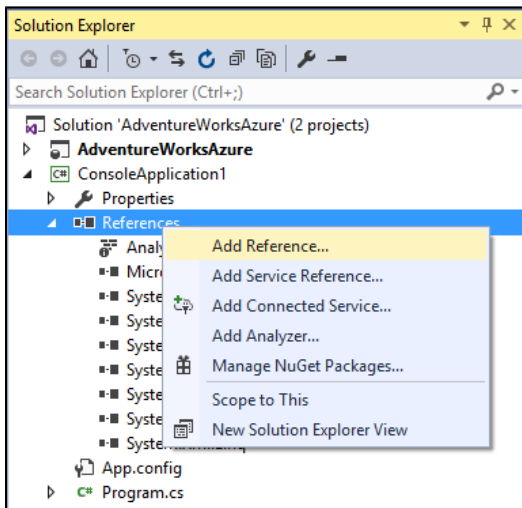
Perform a build of the solution.

In the output window, you'll notice that a package file is now being created as part of the build (in this case, a file called *AdventureWorksAzure\_Package.sql* is being created):

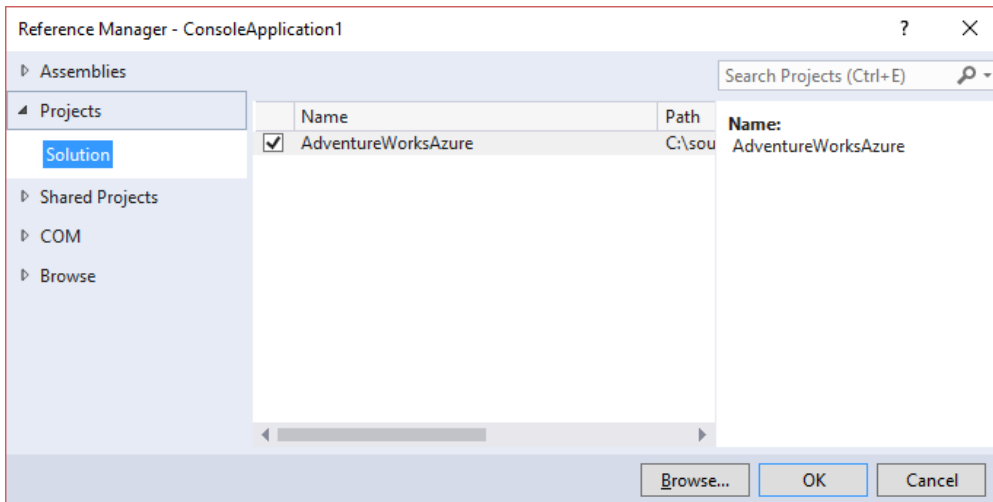


## Application project setup

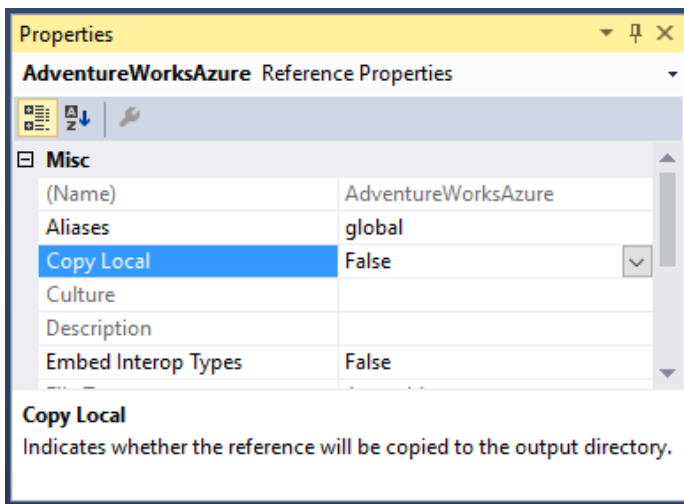
These steps should be performed in the project that you wish to embed the database package and perform the deployment from.



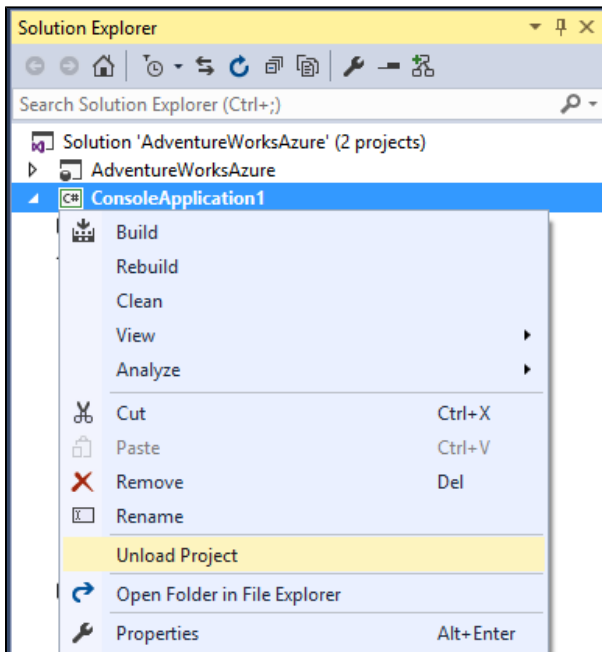
Within your application project, add a reference to the ReadyRoll database project.



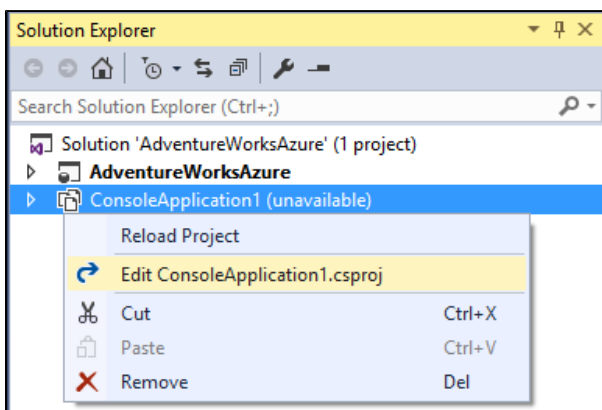
As no additional artifacts are needed from the database project, within the reference properties set **Copy Local** to *False*.



Next add a link to the package resource file by firstly unloading the application project.



And open the project file for editing.

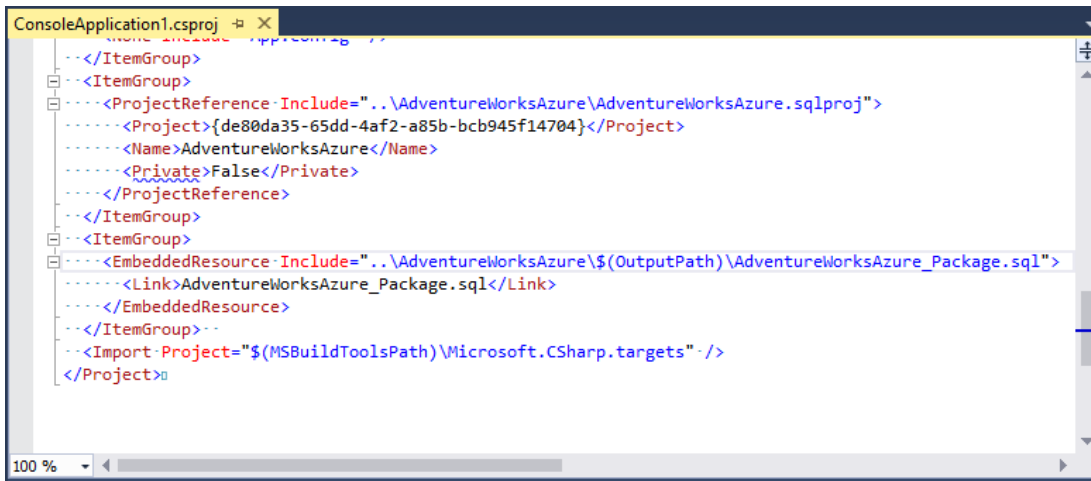


Insert an embedded link at the end of the project file in the following format:

#### Csproj/Vbproj file addition

```
<ItemGroup>
  <EmbeddedResource Include="..\DatabaseProjectName\$(OutputPath)\DatabaseProjectName_Package.sql">
    <Link>DatabaseProjectName_Package.sql</Link>
  </EmbeddedResource>
</ItemGroup>
```

For example:



```

<!-->
</ItemGroup>
<ItemGroup>
  <ProjectReference Include="..\AdventureWorksAzure\AdventureWorksAzure.sqlproj">
    <Project>{de80da35-65dd-4af2-a85b-bcb945f14704}</Project>
    <Name>AdventureWorksAzure</Name>
    <Private>False</Private>
  </ProjectReference>
</ItemGroup>
<ItemGroup>
  <EmbeddedResource Include="..\AdventureWorksAzure\$(OutputPath)\AdventureWorksAzure_Package.sql">
    <Link>AdventureWorksAzure_Package.sql</Link>
  </EmbeddedResource>
</ItemGroup>
<Import Project="$(MSBuildToolsPath)\Microsoft.CSharp.targets" />
</Project>

```

Save the file and reload the project.

## Adding the SQLCMD runner

To run the package deployment script, we'll use the [DBUtil open source library by Remus Rusanu](#).

Within the package manager console window, run the following to add the SQLCMD runner dependency to the project:

```
Install-Package com.rusanu.DBUtil
```



### DbUtil Limitations

Please note that DbUtil does not currently support the following SQLCMD syntax:

- :list
- :reset
- :error
- :ed
- :out
- :perftrace
- :help
- :serverlist
- :xml
- :listvar

If the full set of SQLCMD functionality is required, then we recommend using the [PowerShell method](#) to deploy your database. Unfortunately embedded deployment is not currently supported with this approach.



### .NET Framework only

The *com.rusanu.DBUtil* tool requires that your embedding project target .NET Framework 2.0 or later; unfortunately .NET Core is not currently supported.

However, other than this library, which currently includes a single assembly about 30kb in size, there are no other dependencies to ship with your application. This means you can deploy your application to a customer without needing SqlCmd.exe or ReadyRoll itself to be installed. You don't even need to ship the above SQL file, as it will be embedded within your application assembly at build time.

Add the method to migrate the database (no changes required):

### C# migrate db method

```
private static void MigrateDatabase(string connectionString, string sqlPackageResourceName)
{
    string databaseName = (new System.Data.SqlClient.SqlConnectionStringBuilder(connectionString)).
InitialCatalog;
    using (var sqlConnection = new System.Data.SqlClient.SqlConnection(connectionString))
    {
        sqlConnection.Open();
        using (var sqlRunner = new com.rusanu.DBUtil.SqlCmd(sqlConnection))
        {
            sqlRunner.Environment.Variables.Add("DatabaseName", databaseName);
            sqlRunner.Executing += (object sender, com.rusanu.DBUtil.SqlCmdExecutingEventArgs e) => Console.
Write(".");
            var assembly = System.Reflection.Assembly.GetExecutingAssembly();
            using (var sqlPackageStream = assembly.GetManifestResourceStream(assembly.EntryPoint.DeclaringType.
Namespace + "." + sqlPackageResourceName))
            {
                Console.WriteLine("Bringing the \"" + databaseName + "\" database up-to-date by running any pending
migrations");
                if (!sqlRunner.ExecuteStream(sqlPackageStream))
                    throw new Exception("Database deployment failed. SQL Server error details: " + sqlRunner.
LastException.Message + ", Batch that raised error: " + sqlRunner.LastBatch);
                Console.WriteLine("done");
            }
        }
    }
}
```

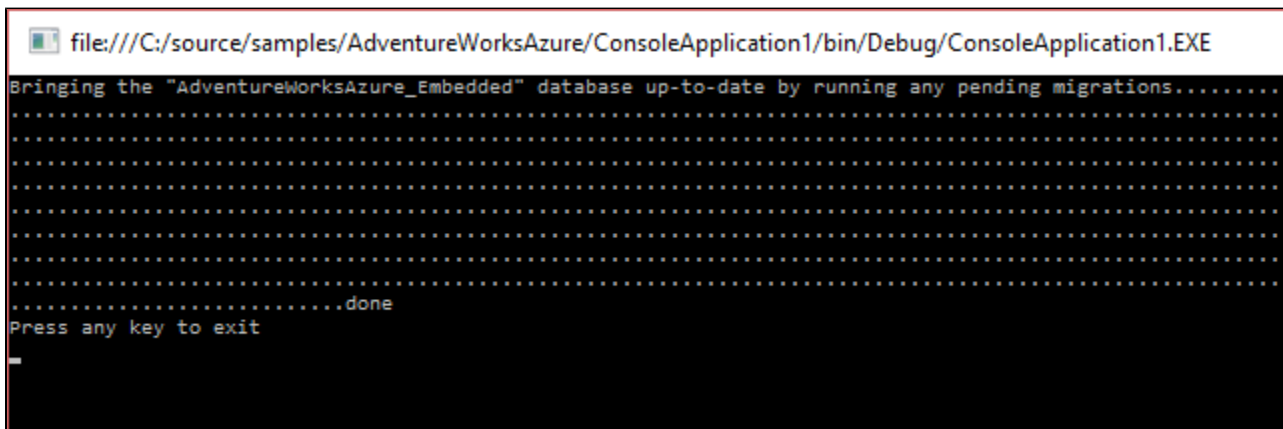
Add a call to the migrate method within the desired part of the application, substituting your own connection string and the name of the package artifact:

### C# call to migrate method

```
const string ConnectionString = @"Data Source=(localdb)\ProjectsV13;Integrated Security=True;Pooling=False;
Connect Timeout=30;Initial catalog=AdventureWorksAzure_Embedded";
const string SqlPackageResourceName = @"AdventureWorksAzure_Package.sql";

static void Main(string[] args)
{
    // Deploy the database from the embedded resource to the specified connection or incrementally migrating it
to the latest state
    MigrateDatabase(connectionString: ConnectionString, sqlPackageResourceName: SqlPackageResourceName);
    Console.WriteLine("Press any key to exit");
    Console.ReadKey();
}
```

Now try starting the solution. The database should be deployed as the application starts.



The screenshot shows a console window titled "file:///C:/source/samples/AdventureWorksAzure/ConsoleApplication1/bin/Debug/ConsoleApplication1.EXE". The output text is as follows:

```
Bringing the "AdventureWorksAzure_Embedded" database up-to-date by running any pending migrations.....
.....
.....done
Press any key to exit
```

A cursor is visible at the bottom of the window, indicating it is ready for user input.

## What happens if the database doesn't exist?

If you attempt to use the above code to deploy to a SQL Server where the database doesn't exist, you will receive the follow error message:

```
Login failed for user '<user_name>'. (Microsoft SQL Server, Error: 18456)
```

This error occurs due to the inclusion of the *Initial Catalog* property within the connection string, which refers to the as-yet non-existent database.

The solution is to simply omit the *Initial Catalog* property from the connection string. This will allow the deployment script to connect to the server, create the database and then run the migrations against the newly created database. However it may not be desirable to store the connection string in your app. config / web.config file without this property in place, as this may affect the ability of your application to connect to the database at runtime.

Therefore, you may prefer to add the following code to your application to handle this at deployment time:

### C# Methods to trim connection string

```
private static string GetConnectionStringWithoutInitCatalog(string connectionString, out string databaseName)
{
    var connStringBuilder = new SqlConnectionStringBuilder(connectionString);
    databaseName = connStringBuilder.InitialCatalog;
    if (string.IsNullOrEmpty(databaseName))
        throw new Exception("Could not find initial catalog attribute in connection string.");
    connStringBuilder.InitialCatalog = ""; // We remove the "Initial Catalog" attribute because the database
    may not exist yet
    return connStringBuilder.ToString();
}
private static bool DoesDatabaseExist(string connectionStringNoInitCatalog, string databaseName)
{
    using (var conn = new SqlConnection(connectionStringNoInitCatalog))
    {
        conn.Open();
        using (var cmd = new SqlCommand("select count(*) from sys.databases where name = @p1", conn))
        {
            cmd.Parameters.Add("@p1", SqlDbType.NVarChar).Value = databaseName;
            return Convert.ToInt32(cmd.ExecuteScalar()) != 0;
        }
    }
}
```

Then modify the beginning of the *MigrateDatabase* method as follows:

## C# Trim database name from connection string

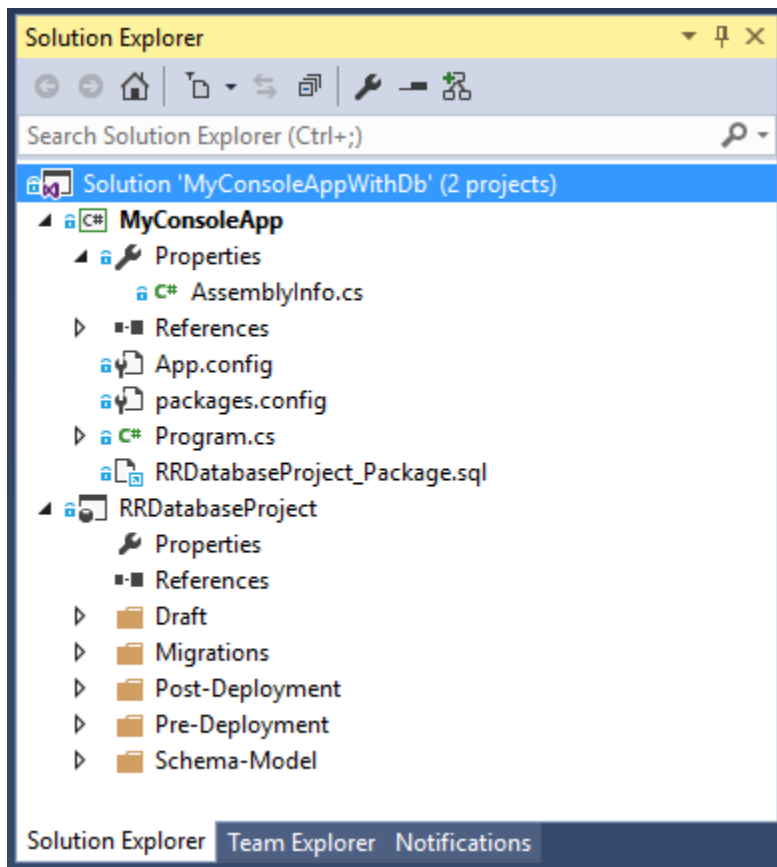
```
private static void MigrateDatabase(string connectionString, string sqlPackageResourceName)
{
    string databaseName;
    string connStringNoDatabaseName = GetConnectionStringWithoutInitCatalog(connectionString, out databaseName);
    // If database doesn't exist yet, then connect to the default db (usually [master]) instead of the
    specified db,
    // in order to allow the ReadyRoll migration package to create the db first.
    // NOTE: For Azure SQL deployments, the first deployment will fail after the db is created. Re-run the
    deployment to migrate the db
    if (!DoesDatabaseExist(connStringNoDatabaseName, databaseName))
        connectionString = connStringNoDatabaseName;
    using (var sqlConnection = new SqlConnection(connectionString))
    {
        sqlConnection.Open();
        using (var sqlRunner = new com.rusanu.DBUtil.SqlCmd(sqlConnection))
        {
            sqlRunner.Environment.Variables.Add("DatabaseName", databaseName);
            sqlRunner.Executing += (object sender, com.rusanu.DBUtil.SqlCmdExecutingEventArgs e) => Console.
Write(".");
            var assembly = System.Reflection.Assembly.GetExecutingAssembly();
            using (var sqlPackageStream = assembly.GetManifestResourceStream(assembly.EntryPoint.DeclaringType.
Namespace + "." + sqlPackageResourceName))
            {
                Console.WriteLine("Bringing the \"" + databaseName + "\" database up-to-date by running any pending
migrations");
                if (!sqlRunner.ExecuteStream(sqlPackageStream))
                    throw new Exception("Database deployment failed. SQL Server error details: " + sqlRunner.
LastException.Message + ", Batch that raised error: " + sqlRunner.LastBatch);
                Console.WriteLine("done");
            }
        }
    }
}
```

The above code will allow the *Initial Catalog* property to be left in your config file, and only removed from the connection string for the purposes of performing the initial database deployment.

To see this code in full, download the below sample project.

## Sample Project

[This sample project](#) contains a console application that deploys the package artifact from a database project as an embedded resource.



Download the sample solution/project: [MyConsoleAppWithDb.zip](#)