ReadyRoll 1.4 release notes

1.4.6 - March 14th, 2013

This is a minor release that improves SQL Server 2012 support, provides some new options for script generation and addresses some issues introduced by version 1.4.3.

Features

- Add script-generation support for Sequences in the ReadyRoll DBSync tool (Details)
- Provide project-level options to allow fine-grained control on schema import/script generation (Details)

Fixes

- Connection to target server fails if database name in design-time connection string doesn't exist (Details)
- Multiple versions of the ReadyRoll NuGet package can cause unpredictable build behaviour (Details)

1.4.3 - February 13th, 2013

Today's release brings some improvements to the way you build your ReadyRoll database projects.

These changes make it easier to do Continuous Integration with ReadyRoll, whether you're building your projects on self-hosted hardware (eg. with TeamCity or TFS Build), or in the cloud (eg. with AppHarbor or Bamboo OnDemand).

Previously you need to manually install a whole bunch of software on your build agent to deploy and test your database projects. This included Visual Studio 2012/Visual Studio 2010 with SP1, SQL Server Data Tools, ReadyRoll itself plus a handful of client tools.

Now, with the power of Chocolatey and NuGet, this process has been made a whole lot simpler. This coupled with the fact that ReadyRoll build agent licenses are free (as in beer). This gives you the flexibility to scale your Continuous Integration/Delivery environment without incurring additional cost. It also means no messy software activation steps to run on your build agents!

Installing Build Agent Pre-Requisites with Chocolatey

Before getting started, ReadyRoll's most basic requirement is that the host system is running one of the following operating systems:

- Windows Server 2008 R2 SP1
- Windows Server 2012
- Windows 7 SP1
- Windows 8

Once you've got an operating system up-and-running, you'll need to get the supporting software installed. This includes:

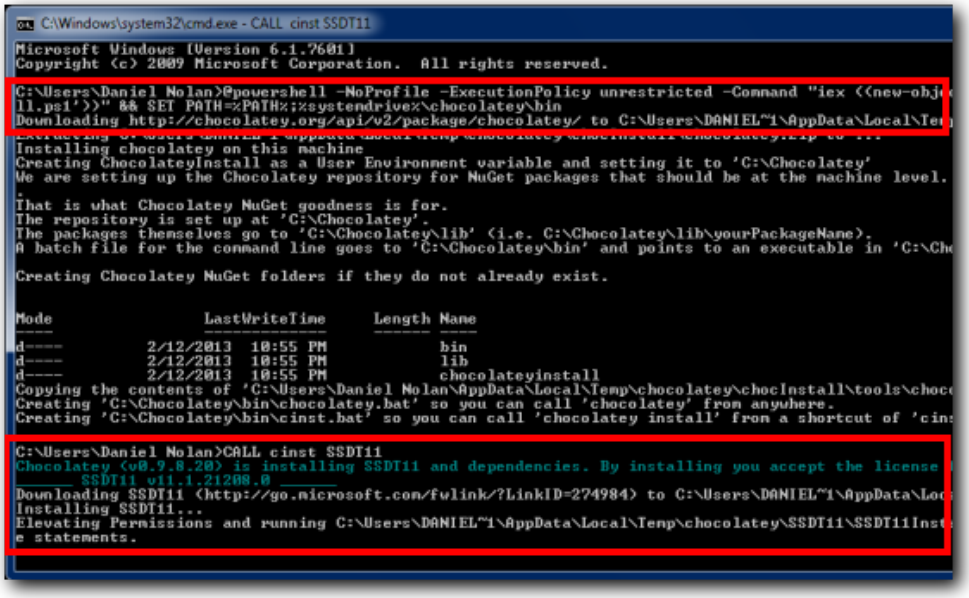
- SQL Server Data Tools
  Provides build-time support for SQL parsing and .NET CLR compilation.

- SQL Server Express 2012
  Allows ReadyRoll to test the deployment of your database on a stand-alone instance of SQL Server (referred to as a Shadow build).

Now if you have some time to burn you could go ahead and download these and manually click-through the dozens of dialog boxes needed to get them installed. However if you'd rather have a coffee break so you can think about how awesome it will be when your database builds are automated (as some of us do!), then Chocolatey could be your saviour today.

To install the pre-requisites above, including Chocolatey, open a command prompt in administrator mode and run the following:

```
@powershell -NoProfile -ExecutionPolicy unrestricted -Command "iex ((new-object net.webclient).DownloadString
('http://chocolatey.org/install.ps1'))" && SET PATH=%PATH%;%systemdrive%\chocolatey\bin
CALL cinst SSDT11
CALL cinst MsSqlServer2012Express @ECHO Adding [NT AUTHORITY\Authenticated Users] to sysadmin role on local SQL
instance
"%ProgramFiles%\Microsoft SQL Server\110\Tools\Binn\sqlcmd.exe" -S . -E -Q "ALTER SERVER ROLE [sysadmin] ADD
MEMBER [BUILTIN\Users];"
```



That's it! Well, it may take a little while to install but you won't be prompted during the installation (your acceptance of the Microsoft EULA's is implied; see more legal stuff here).

Oh and that last command (`ALTER SERVER ROLE...`) just ensures that your build agent can access the newly-installed SQLEXPRESS instance (which becomes the default SQL Server instance on the machine). You might choose to make this access more restrictive if you know which user your build agent will run-as.

Bootstrapping Build-Time Dependencies with NuGet

In an ideal world, all the software that is needed to produce a build would be checked into source control so we wouldn't need to install any software on the build server; a key tenet of Continuous Delivery is that builds should be reproducible, and changes in the software environment can often result in inconsistent build artifacts being produced.

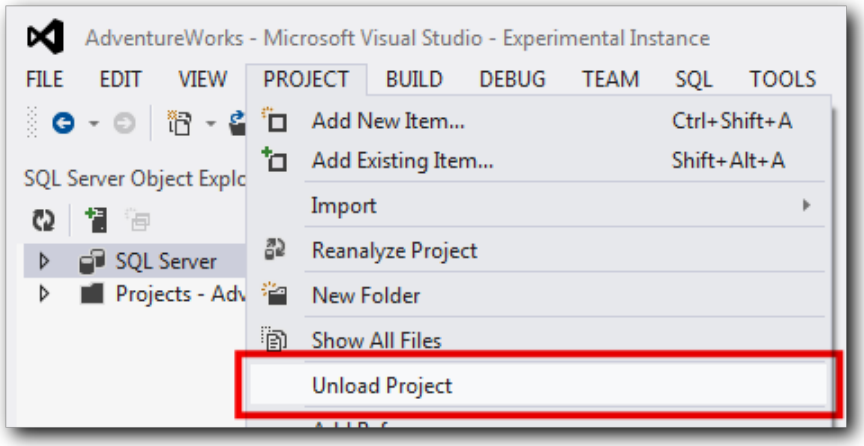
Though you can't yet bootstrap an instance of SQL Server Express or even the Data Tools to your project code, ReadyRoll 1.4.3 does at least allow you to commit its build-time dependencies to source control using the power of NuGet. This means that you can rely on the outcome of your database builds to be consistent between deployment environments (ie. TEST/QA/PROD). It also means that there's nothing extra to install on your build agents to get your ReadyRoll builds to work.

To integrate the build-time dependencies into your project, open the Package Manager Console tool-window and install the package from the NuGet gallery:

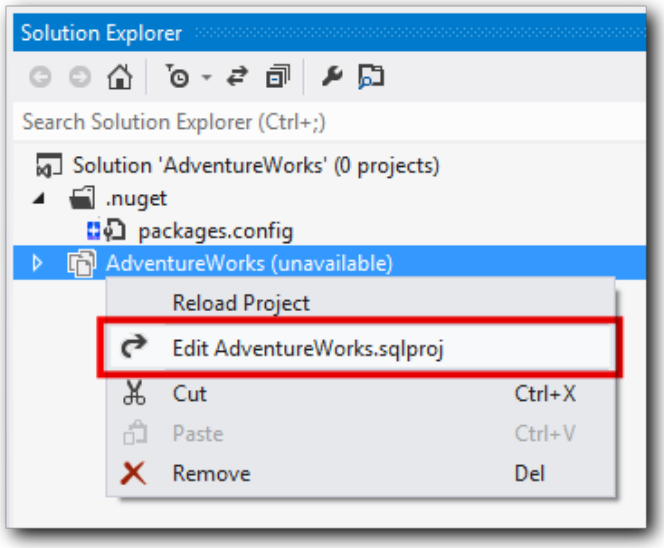
```
PM> Install-Package ReadyRoll.MSBuild
```

This will install the package into the current solution, rather than into a specific Database Project within the solution. This is a current limitation in NuGet, so to work around this we'll need to link the page to the database project manually.

To do this, firstly select the ReadyRoll Database Project in the Solution Explorer, then click *Project… Unload Project*.



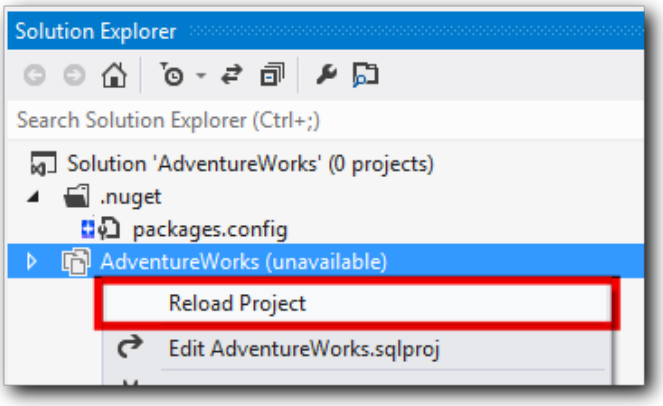
Then, right-click the project in the Solution Explorer and click *Edit Project*.



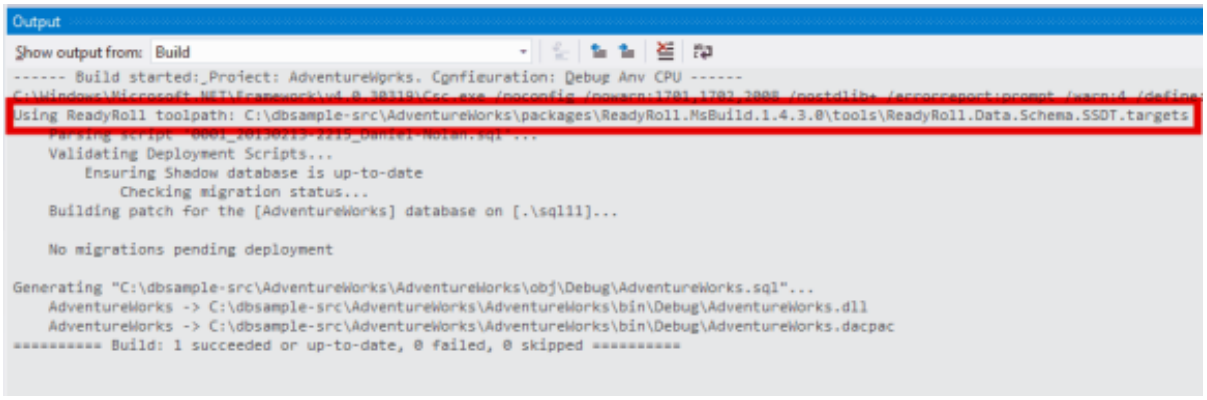
Find the `<ReadyRollTargetsPath>` element and replace the contents as follows:

```
<PropertyGroup>
  <ReadyRollTargetsPath>..\packages\ReadyRoll.MsBuild.1.4.3.0\tools\ReadyRoll.Data.Schema.SSDT.targets<
/ReadyRollTargetsPath></ReadyRollTargetsPath>
</PropertyGroup>
```

Save the project file, then right-click the project in the Solution Explorer and select *Reload Project*



Confirm that the build-time dependencies have been successfully linked to the project by building the solution:



Now commit the solution/project changes along with the package files to source control.

Congratulations, your build agents are now ready to roll your database projects!

1.4.0 - December 6th, 2012

The theme of today's release can be summed up in one word: variables, variables and more variables!

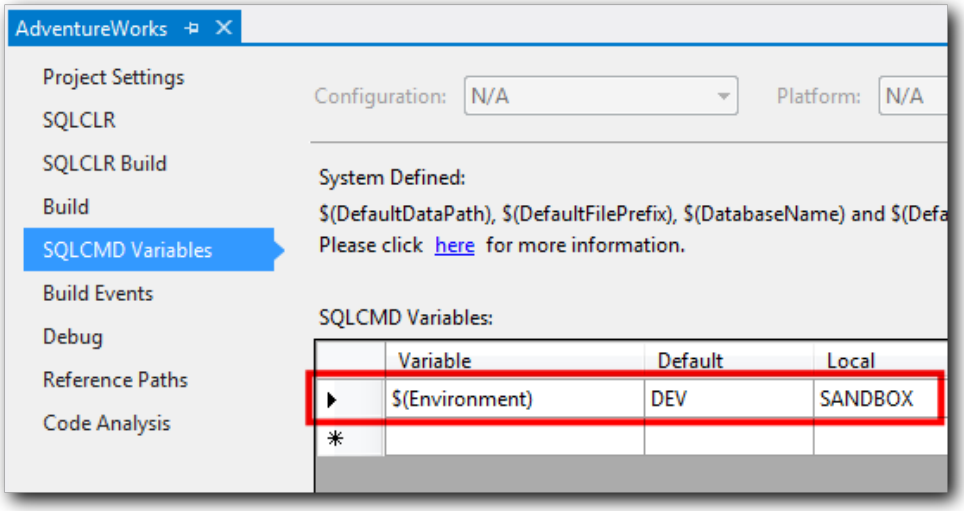
SQLCMD Variable Support Added

If in your organisation you need handle subtle differences in the setup of your environments (eg. Test/Staging/Production), or if you are deploying to customer sites with variations that you don't want to hardcode into your project scripts, it can be useful to provide certain values from an external source. ReadyRoll 1.4 allows you to do this by tapping into the SQLCMD variable support built-in to Visual Studio, providing you with a way to write scripts that use different values depending on the context.

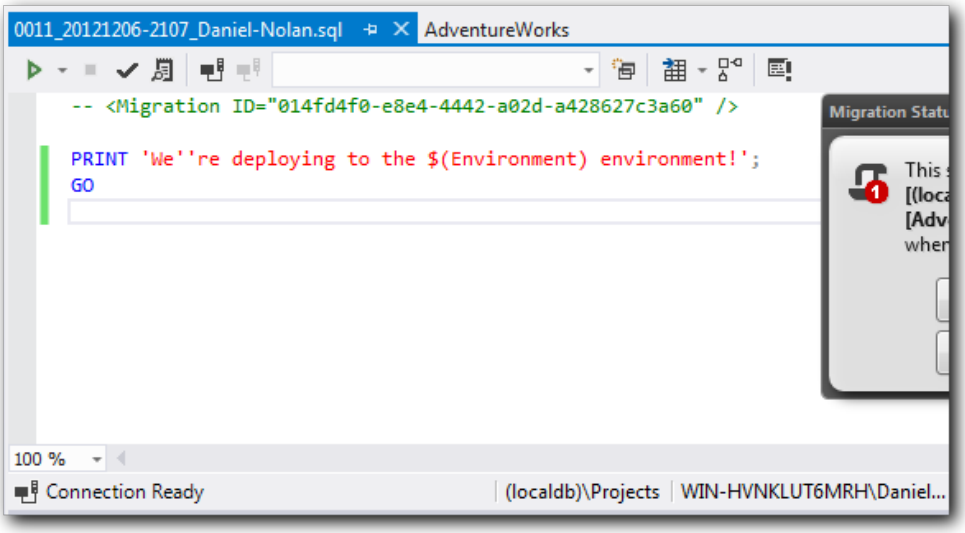
When working within Visual Studio, you can use sandbox or project-level, and outside the IDE you can provide environment-level values from an external source, like a Continuous Integration/Automated Deployment server (eg. Octopus).

Project and Sandbox-specific values

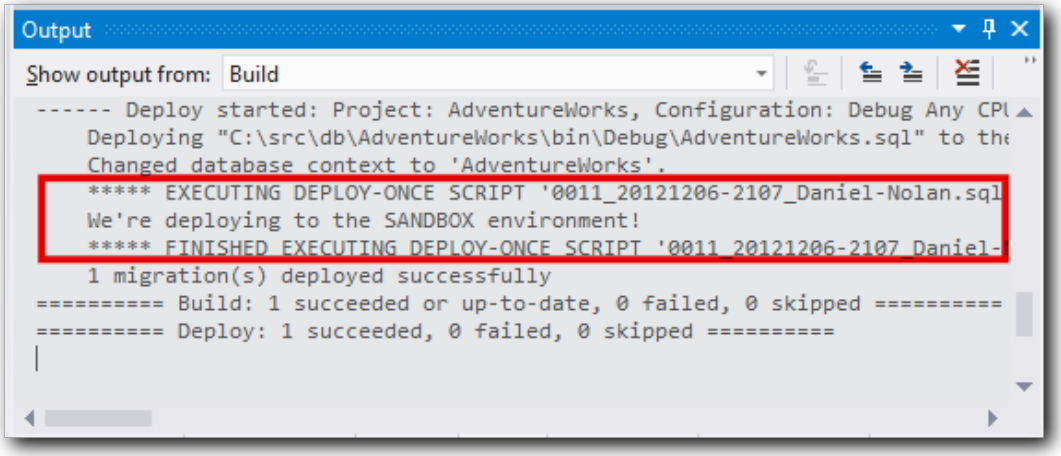
Start by adding a variable to the SQLCMD Variables tab in the Project Property pages. The value you provide in the Default column will be stored in the project file (and therefore shared with other team members) however the Local value is specific to your machine (stored in the non-source controlled *.user* fi le). If you leave the Local column blank, the Default will be used when deploying inside Visual Studio.



Using the *$(VariableName)* notation, reference the variable you just created in a new Deploy-Once script.



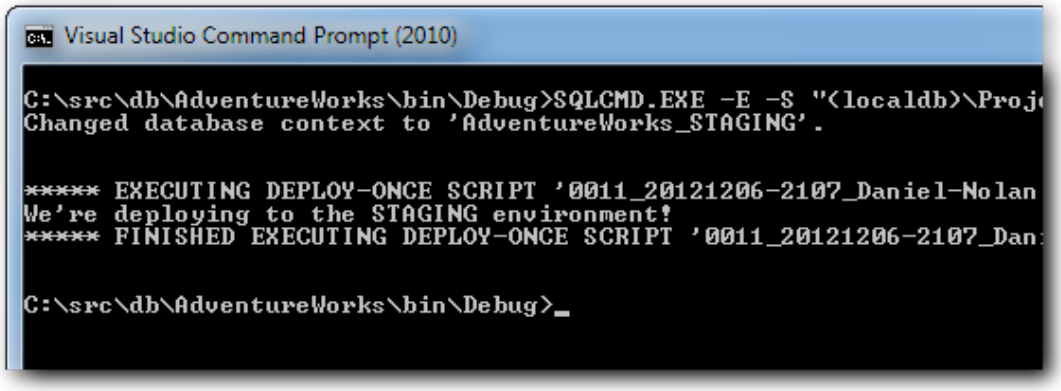
When you build the project, the variable will be substituted with the Default value (or Local value, if it was provided).



Environment-specific values

When deploying your database outside of Visual Studio, you can also provide a set of values for the variables defined in your project.To do this, first enable SQLCMD package scripts for output (or enable *.nupkg* output if you are using Octopus). Build your project to produce the package deployment script, eg. *MyDatabase_Package.sql*. Notice in the header of the file that the full list of project variables are included, but commented out along with their default values. You will need to provide values for each of the variables listed, as part of the next step to deploy your database (Note that *$(DatabaseName)* is a built-in SQLCMD variable that is required for all database projects). To deploy your database, open the command prompt and execute the following command:

```
SQLCMD.EXE -E -S "(localdb)\Projects"
           -i "AdventureWorks_Package.sql"
           -v DatabaseName=AdventureWorks_STAGING
           -v Environment=STAGING
```



Note: There is currently no straightforward way of passing SQLCMD variables into MSBuild; at present the Package deployment method is the best way to use SQLCMD variables at this time. If you'd prefer to use the Patch deployment script method instead (which provides a delta file of pending migrations and is only available via MSBuild), please get in touch.

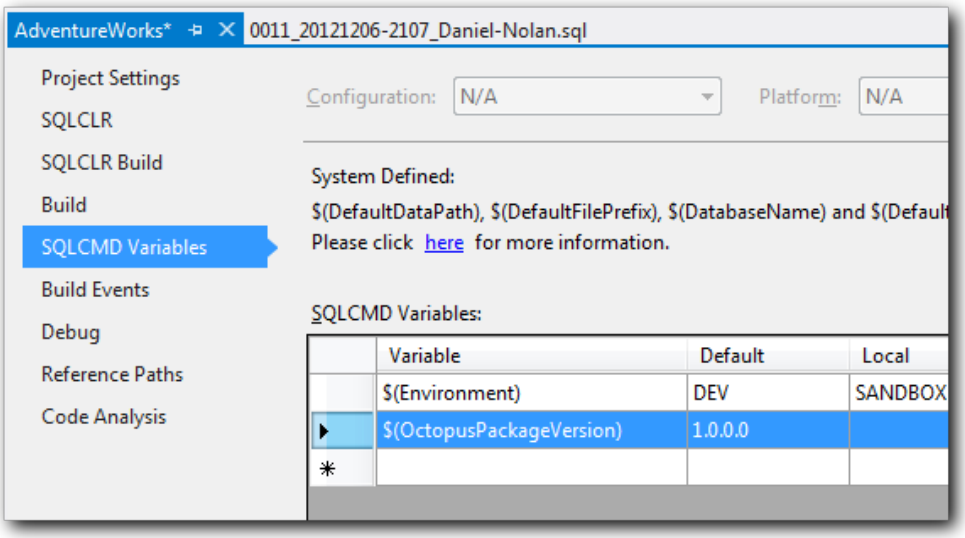
Next we'll look at a variation of this approach, which involves using the built-in Octopus support to provide SQLCMD variable values you can use in your database deployments.

Enhanced Octopus Support with Variable Mapping

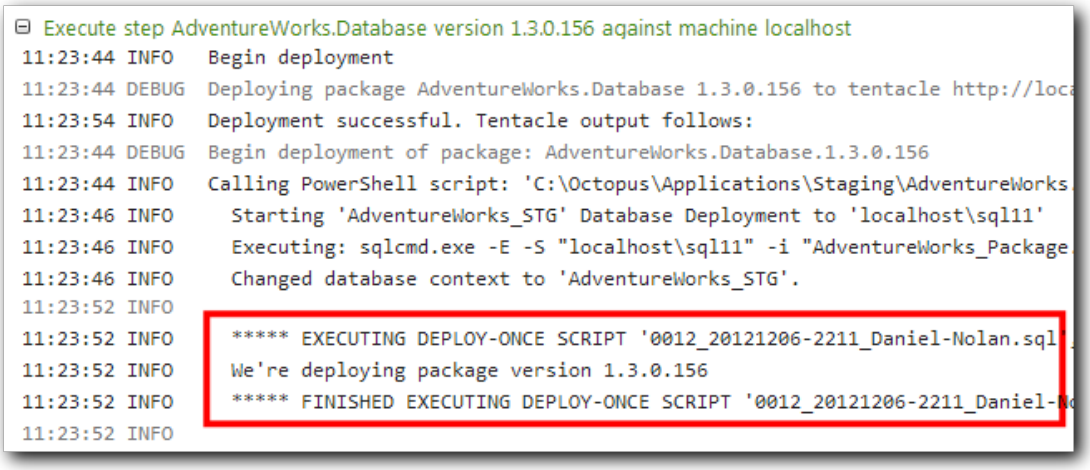
ReadyRoll 1.3 introduced support for Octopus, making it easy for database deployments to be coordinated alongside other application components using the NuGet-based package format.

ReadyRoll 1.4 builds on this support by providing automated mapping of Octopus variables to SQLCMD variables. This makes it incredibly easy to consume values from your existing pool of environment-specific variables, or to make use of the project and package variables provided by Octopus itself.

For example, say you wanted to store the version number of the deployment package in the target database. To do this, firstly add the *$(OctopusPackageVersion)* variable to the ReadyRoll project and give it a Default value, eg. 1.0.0.0.

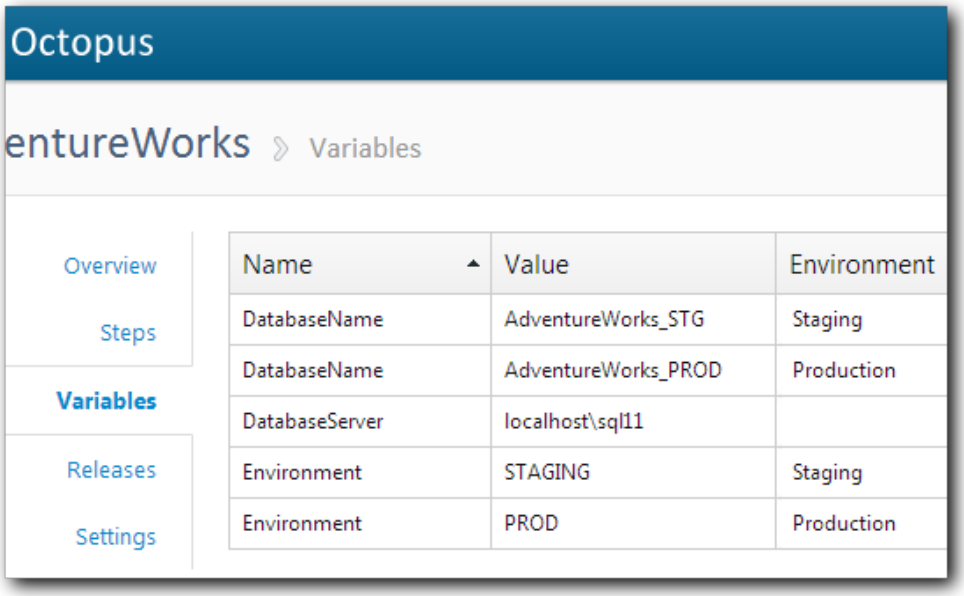


Then include a reference to the variable in a Deploy-Once script. When you deploy locally it will output the Default value, however when you deploy via Octopus the value will be substituted with the deployment package version:



To get a full list of built-in variables, check out the Octopus Variables documentation.

Mapping isn't just limited to built-in Octopus variables; simply add your variables to your ReadyRoll project to map your custom variables as well.



Note: If you're already deploying your database projects using Octopus, please note there is a code-breaking change in this release: previously, the *$(DatabaseName)* SQLCMD variable would to hardcoded to the name specified in your ReadyRoll project. From ReadyRoll 1.4 onwards, you will need to provide a value for the database name within the Octopus project variables.

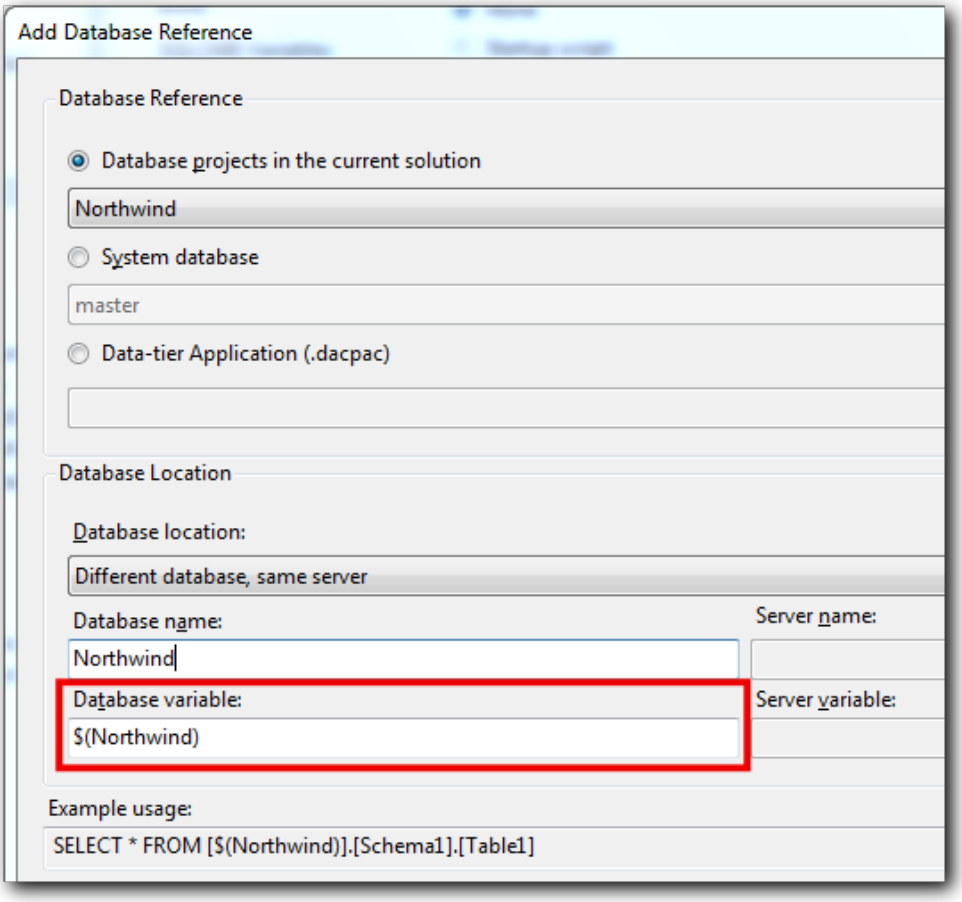
Multi-database Support

As ReadyRoll projects are based within the Visual Studio IDE, you've always had the ability to develop and deploy multiple databases. However up until now, it has been difficult to manage the deployment of databases with interdependent object references.

For example, say you have two projects: *DatabaseA* and *DatabaseB*. If DatabaseB contained a script that referred to an object within DatabaseA, you would first need to build and deploy DatabaseA before you could even build DatabaseB. This is because of the way that ReadyRoll uses a separate copy of your database called the *Shadow* that ensures your project scripts are parsed & validated before deploying to your *Sandbox* database. This process is called gated deployment, and it requires that all database references be dynamic (which is why you need to use the *$(DatabaseName)* variable whenever you reference your database).

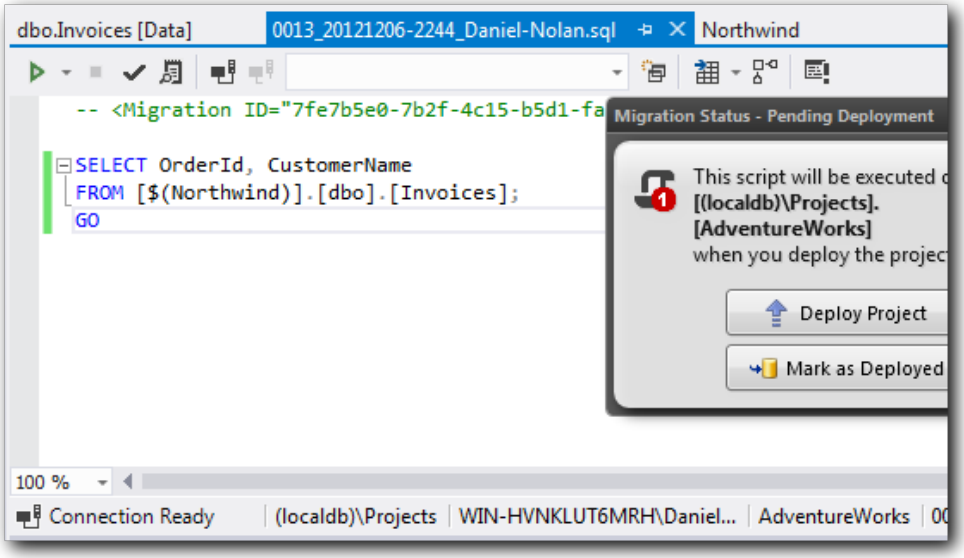
To make it easier to work with interdependent databases, ReadyRoll 1.4 introduces the ability to create dynamic database references within your projects.

Start by adding a Database Reference to your project (from the Solution Explorer context menu).



Don't worry about the *Database name* field; this will be automatically substituted at build time.

To refer to an object within the database, you can then use a three-part object name within a Deploy-Once script using – you guessed it – a SQLCMD variable:



When deploying outside Visual Studio, just make sure you specify a variable/value pair for each of the database names that are referenced from your project.

Object Exclusion Rules

For a variety of reasons, sometimes it's necessary to exclude certain objects from your database projects: maybe some objects are needed only in Production (such as log tables or reporting objects) or perhaps their deployment is managed by a completely separate deployment mechanism (like in a CRM system). This can be a problem when using the ReadyRoll *DBSync* tool, which will eagerly import any new objects from the source database that are added to your schemas.

To make it easier to prevent these types of objects from appearing in your project altogether, ReadyRoll 1.4 introduces regex-based exclude rules to the project system. For example, say you wanted to exclude the *[reporting].[log]* table object from the database. To do so, just add the following to the top of your *.sqlproj* file:

```
<PropertyGroup>
    <ExcludeObjectsFromImport>
        Table=\[reporting\]\.\[log\];
    </ExcludeObjectsFromImport>
  </PropertyGroup>
```

If you'd like to exclude all Tables & Stored Procedures in a particular Schema (along with the Schema itself) from your project, try this:

```
  <PropertyGroup>
    <ExcludeObjectsFromImport>
        Schema=\[reporting\];
        Table=\[reporting\]\.\[(.*?)\];
        StoredProcedure=\[reporting\]\.\[(.*?)\];
    </ExcludeObjectsFromImport>
  </PropertyGroup>
```

Don't worry… we won't be offended if you add lots of exclude rules to your project! Thanks to @DataChomp for the idea.