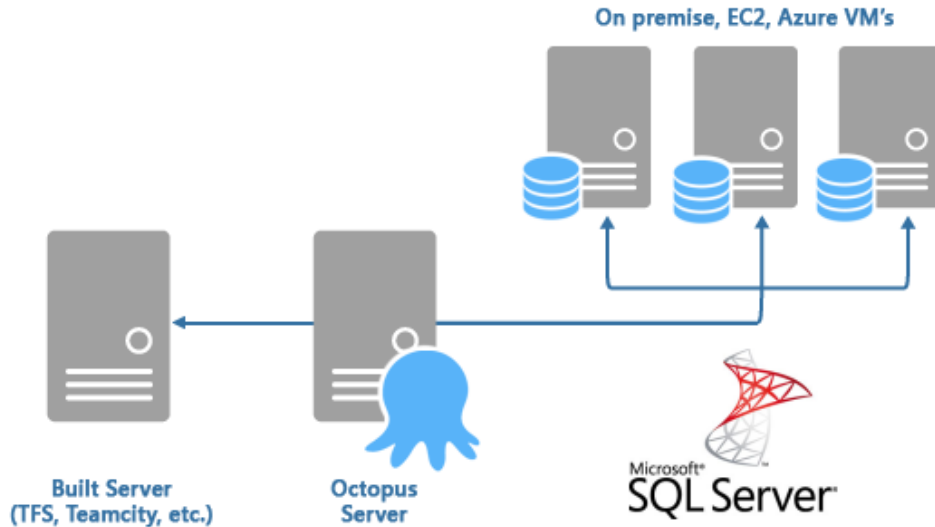


Octopus Deploy

When it comes to deploying SQL Server databases, [Octopus Deploy](#) and ReadyRoll make a great team.

Use ReadyRoll to carefully prepare your migrations – column additions, [stored procedure changes](#), [SQLCLR assemblies](#), or [static data](#) and then propagate those changes through your environments using Octopus with total consistency and minimal fuss.



ReadyRoll makes deploying with Octopus easy through its out-of-the-box support for OctoPack; a tool that produces standardized packages that can be consumed by Octopus Deploy.

During deployment of ReadyRoll database packages, Octopus can take care of:

- Creating the database if it doesn't exist
- Deploying of any pending migrations with [automatic transaction handling](#)
- Rolling-back to earlier versions of procs/views/functions (if an old package is redeployed)
- Running of any pre/post-deployment actions (eg. db backup or bulk data insertions)

In your SQL migration scripts, you can make use of any of system or user-defined variables from Octopus, allowing for [environment-specific settings](#) to be consumed during deployment.



Octopus Deploy Step Template

To get up and running with Octopus Deploy even more quickly we've created a ReadyRoll step template you can drop into your process. Instead of having to use variables, you can use our step template user interface to enter your database connection details. The template is available from the [Octopus Deploy Library](#).

See [Tutorial: Octopus Deploy Step Template](#) for more detail on getting set up.

Packaging your database project

After using Visual Studio to create your ReadyRoll database project and import your database (if pre-existing), open the project properties.

On the *Project Settings* tab, you'll notice an option to enable Octopus packaging for your project:

Project Settings

Configuration: N/A Platform: N/A

SQLCLR

SQLCLR Build

Build

SQLCMD Variables

Build Events

Debug

Reference Paths


Code Analysis

Target platform:

SQL Server 2012

Output types

☐ SQLCMD package (.sql file)

☒  Octopus-compatible NuGet package (.nupkg file) [Add/Edit NuSpec file](#) [Find out more about Octopus](#)

☒ Deploy to the default (local) instance on the Tentacle

Choose *Deploy to the default instance on the Tentacle* option if you intend to install a Tentacle on each of your target SQL Server hosts (more about local vs remote deployments later in this article).

In effect this option sets the *DatabaseServer* variable to (local), however if your SQL Server instance has a specific name, you can optionally add a value for the *DatabaseServer* variable on the [Octopus Deploy](#) tab:

AdventureWorks

Project Settings

SQLCLR

SQLCLR Build

Build

SQLCMD Variables

Build Events

Debug

Reference Paths

Code Analysis

Configuration: N/A Platform: N/A

System Defined:

\$(DefaultDataPath), \$(DefaultFilePrefix), \$(DatabaseName) and \$(DefaultLogPath) are system defined
Please click [here](#) for more information.

SQLCMD Variables:

	Variable	Default	Local
▶	\$(DatabaseServer)	localhost\SQLEXPRESS	
	\$(ReportServerPath)	\\bigdan01\outdir\$\	
*			

Switch back to the *Project Settings* tab and click *Add/Edit NuSpec File*. This will add a package metadata file to the project:

AdventureWorks.nuspec

```
<?xml version="1.0"?>
<package xmlns="http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd">
  <metadata>
    <id>AdventureWorks.Database</id>
    <title>AdventureWorks</title>
    <!-- The version number will be retrieved from the Assembly Information
         specified on the SQLCLR tab within project properties. -->
    <version>1.0.0</version>
    <authors>Your name</authors>
    <owners>Your name</owners>
    <licenseUrl>http://yourcompany.com</licenseUrl>
    <projectUrl>http://yourcompany.com</projectUrl>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>This database deployment package was generated by ReadyRoll using OctoPack.</description>
    <releaseNotes>This release contains the following changes...</releaseNotes>
  </metadata>
</package>
```

(Optional step) If you intend to use the [Assembly Info patcher in TeamCity](#), you will need to add assembly information to your project. To do so, switch to the the *SQLCLR* tab, and click *Assembly Information* to generate an **AssemblyInfo.cs** file:

AdventureWorks -> X

Project Settings

SQLCLR

SQLCLR Build

Build

SQLCMD Variables

Build Events

Debug

Reference Paths

Code Analysis

Configuration: N/A Platform: N/A

Assembly name: AdventureWorks

Default namespace: AdventureWorks

Target framework: .NET Framework 4.5

Assembly Information...

Permission level: SAFE

Assembly owner:

Language: C#

i If you intend to supply the *OctoPack Package Number* property as part of your Continuous Integration build (e.g. using the [Octopus plugin for TeamCity](#)), then adding the assembly information file is not necessary; OctoPack will simply ignore the assembly version if this property is specified.

Now switch to the *Release* configuration and **Build** the solution to test packaging of your database project:

Output

Show output from: Build

```

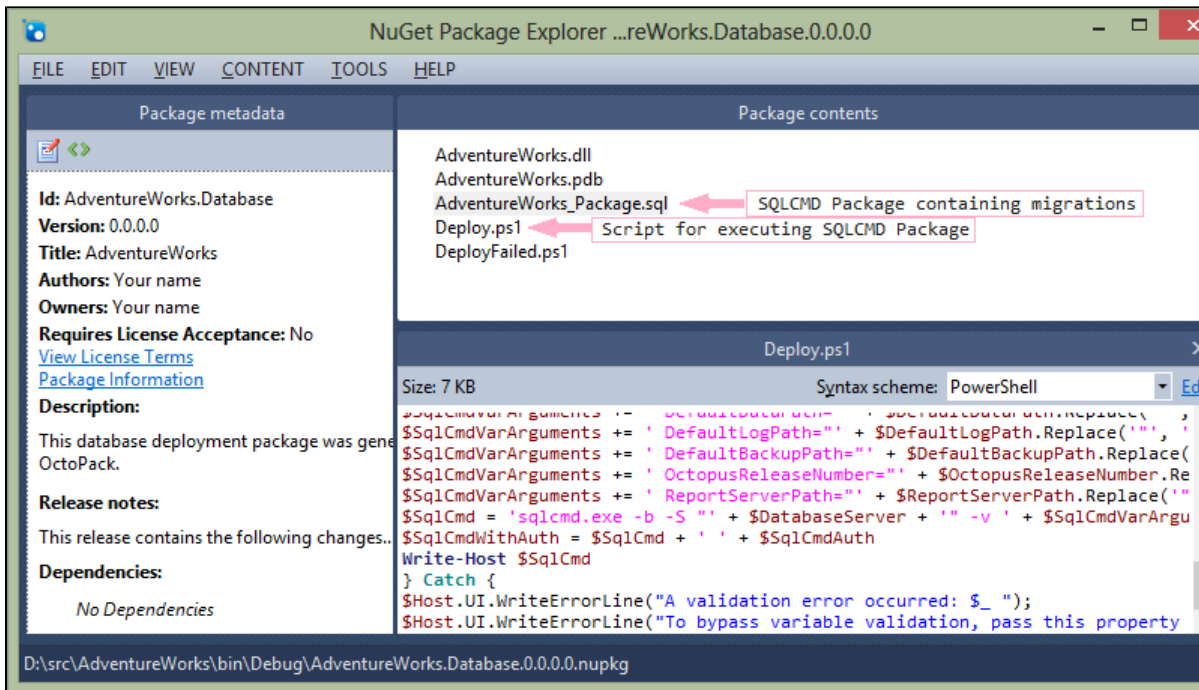
----- Build started: Project: AdventureWorks, Configuration: Debug Any CPU -----
Using ReadyRoll toolpath: C:\Program Files (x86)\MSBuild\ReadyRoll\ReadyRoll.Data.Schema.SSDT.targets
Building patch for the [AdventureWorks] database on [(local)]...

No migrations pending deployment

Generating "D:\src\AdventureWorks\obj\Debug\AdventureWorks.sql"...
AdventureWorks -> D:\src\AdventureWorks\bin\Debug\AdventureWorks.dll
AdventureWorks -> D:\src\AdventureWorks\bin\Debug\AdventureWorks.dacpac
OctoPack: Packaging a console or Window Service application
OctoPack: Attempting to build package from 'AdventureWorks.nuspec'.
OctoPack: Successfully created 'D:\src\AdventureWorks\obj\octopacked\AdventureWorks.Database.0.0.0.0.nupkg'.
OctoPack: OctoPack successful
===== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped =====
|

```

Opening up the generated package using [NuGet Package Explorer](#) will reveal a handful of deployment assets:



Performance tip: Once you have configured your database project for Octopus packaging, feel free to un-check the *Octopus Deploy-compatible NuGet package* output option in project settings. This will result in much faster Visual Studio builds (don't worry, the package will still be generated during Continuous Integration).

Automating your solution build

You can choose any Continuous Integration server to automate the build of your ReadyRoll database project, but the [Octopus plug-in for TeamCity](#) makes integrating the packaging and deployment of your application and database components incredibly easy.

Add the following build step to a new or existing build configuration/plan:

Runner type: MSBuild

Build file path: (solution or project file path)

MSBuild platform: x86 (32-bit)

MSBuild version: .NET Framework 4.0 (or later)

MSBuild tools version: 4.0 (or later)

System Properties:

RunOctoPack = True (*only needed if not using the TeamCity Octopus plugin*)

OctoPackPackageVersion = e.g. %build.number% (*only needed if not using the TeamCity Octopus plugin*)

OctoPackTargetsPath = path to the *OctoPack.targets* file, specify it only if you want to use a different version of the OctoPack than the one bundled into the ReadyRoll (*optional*)

TargetServer = [Prod SQL Server Instance] (*optional, see below*)

ShadowServer = [Test SQL Server Instance] (*optional, see below*)



Optional system properties

Specify the *TargetServer* and *ShadowServer* properties if you would like to include a [preview of your deployment within Octopus](#):

- **TargetServer:** Target instance of SQL Server to generate the preview against. Typically this will be the SQL Server instance in your Production environment. Read (db_datareader membership) and VIEW DEFINITION permission is required within the target database (s).
- **ShadowServer:** An instance of SQL Server where ReadyRoll may create a temporary copy of your database based on the project sources and produce a schema snapshot file to use as the basis of report generation. Typically this will be an instance of SQL Server in a Development environment.

If the ShadowServer property is not specified, the following message will be output during deployment within Octopus: *Skipping snapshot deployment as a snapshot file could not be found* (note: prior to ReadyRoll 1.16.18088, this message was logged as a warning). Until the property is added, report generation will not be available however the deployment of your scripts themselves will be unaffected. [More information](#)

Build-time dependencies

Before you can build ReadyRoll projects, you'll need to do one of the following:

- install ReadyRoll on your build agent (server), or
- install or include the **ReadyRoll.MSBuild** NuGet package in your solution

For more information, see [Installing on your build agents](#).

As part of your build configuration, you will need to ensure that the NuPkg artifacts (produced during build) are made available to Octopus via a NuGet feed. [Read more](#) about how to integrate TeamCity with Octopus.

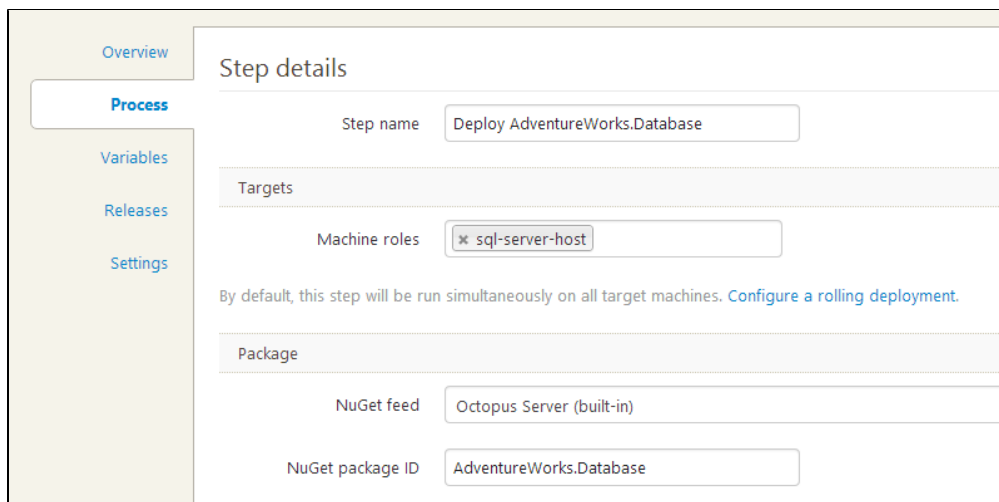
Packaging with TFS Build

Please see the Octopus documentation if you would like to [build and package your solution with Team Foundation Server](#).

Setting up package deployment

The packaging steps above should have resulted in your database being packaged and served up to your chosen NuGet feed.

Within Octopus Deploy, add a new step to your deployment process, specifying the NuGet package ID from your NuSpec file:



The screenshot shows the 'Step details' configuration page in Octopus Deploy. On the left is a sidebar with tabs: Overview, Process (selected), Variables, Releases, and Settings. The main area is titled 'Step details' and contains the following fields:

- Step name:** Deploy AdventureWorks.Database
- Targets:**
 - Machine roles:** sql-server-host
 - By default, this step will be run simultaneously on all target machines. [Configure a rolling deployment](#).
- Package:**
 - NuGet feed:** Octopus Server (built-in)
 - NuGet package ID:** AdventureWorks.Database

In this example, we're assuming that you've installed a Tentacle (deployment agent) onto each of your target SQL Server machines, enabling a local database deployment. More about local vs remote database deployments later.



Be sure to remove any deployment *Features* from the newly-added step; ReadyRoll does not require any features in order to work, and these may cause errors during deployment.

Specifying the Database Server/Name (optional)

By default, ReadyRoll will deploy to the default SQL Server instance on the Tentacle, using the database name specified in your .sqlproj file. However this can be overridden by **adding the following variables to your Octopus project**:

- **DatabaseServer:** The target SQL Server server/instance
- **DatabaseName:** The name of the target database

Scope the variables appropriately to provide a different server name or database name for each target environment. You may want to do this if, for example, you want to deploy from a central Tentacle ([more on this below](#)), or if you need to deploy multiple copies of a database to the same SQL instance.



By default, *Windows Authentication* is used to connect to your database server. This means that the account that the OctopusDeploy Tentacle service is configured to run-as needs to have access to your target SQL Server instances.

If you would prefer to use *SQL Server Authentication* instead, add the following variables within your Octopus project:

- **UseWindowsAuth:** False
- **DatabaseUserName:** (username)
- **DatabasePassword:** (password)

Creating a release

From the *Project Overview* page, click **Create Release** and select the appropriate package version.

If you enabled [deployment previews](#) in your build configuration, the release contents should give you a list of pending changes:

AdventureWorks

Releases

1.3.0.305

Overview

Steps

Variables


Releases

Settings

Release 1.3.0.305

No release notes.

NuGet packages

 AdventureWorks.Database version 1.3.0.305

Published: Thursday, April 18, 2013 4:00 PM

This database deployment package was generated by ReadyRoll using OctoPack.

Migrations pending

1. Deploy-Once\0025_Create-MyTable.sql

2. Deploy-Once\0026_Drop-ProductionCulture-Table.sql

3. Deploy-Once\0027_Drop-ProductionCulture-Refs.sql

4. Deploy-Once\0028_Expand-JobTitle-Column.sql

This list is based on [.\SQL11].[AdventureWorks_PRODUCTION] at 2013-04-18 16:00:01.

The actual list of migrations to be deployed may vary depending on the state of the target environment.

Table changes

>> dbo.MyTable

<> HumanResources.Employee

<< Production.Culture

<> Production.ProductModelProductDescriptionCulture

Other object changes

<< (StoredProcedure) dbo.uspGetWhereUsedProductID

<> (StoredProcedure) HumanResources.uspUpdateEmployeeHireInfo

<> (View) Production.vProductAndDescription

A side-by-side diff report ([AdventureWorks.html](#)) can be found in the project's build artifacts.

Click *Deploy this Release* and select an environment to deploy to. If the database doesn't exist on the target SQL Server instance, it will be created and then all migrations will be executed.

If you switch to the *Task Log*, you'll see the output from your database deployment:

Octopus Deploy

Deploy.ps1

```
If you require that all SqlCmd variable values be passed in explicitly, specify UseSqlCmdVariableDefaults=False.
Using default value for DatabaseName variable: AdventureWorks
Using default value for ForceDeployWithoutBaseline variable: False
**Deploying to (local) because OctoOptionDeployToLocalInstance=True
Using Windows Authentication
Starting 'AdventureWorks' Database Deployment to '(local)'
sqlcmd.exe -b -S "(local)" -v DatabaseName="AdventureWorks" ReleaseVersion="1.0.0.123" ForceDeployWithoutBaseline
DefaultDataPath="d:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\" DefaultLogPath="d:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\Backup\" OctopusReleaseNumber
"D:\Octopus\Applications\Dev\AdventureWorks.Database\1.0.0.123\AdventureWorks_Package.sql"
----- executing pre-deployment script "Pre-Deployment\01_Create_Database.sql" -----
Creating AdventureWorks...
# Beginning transaction
Changed database context to 'AdventureWorks'.
***** EXECUTING MIGRATION "Deploy-Once\1.0.0-Baseline\0001_20140301-0112_User.sql", ID: {f9d39b3b-d725-46d7-b7e3-
Create Schema [HumanResources]
Create Extended Property MS_Description on [HumanResources]
Create Schema [Production]
Create Extended Property MS_Description on [Production]
```

When you're ready to deploy onto other environments, including Production, click *Promote to...* [Environment]. The exact same set of migrations will execute against your upstream environment, giving you a predictable deployment outcome every time.

During subsequent deployments, the package will only deploy any new migrations that have been added to the project (if any). Redeploying the same package, or deploying the package with no new migrations added, has no effect.

Using Octopus variables in your database deployment

During deployment, ReadyRoll will automatically map your Octopus variables to [SQLCMD variables](#). This makes it easy to consume values from your Octopus project's variables, such as environment-specific settings, or to make use of the system variables [provided by Octopus itself](#).

For example, say you want to use the Octopus release number in a migration script. To do so, firstly add the `$(OctopusReleaseNumber)` variable to the ReadyRoll project and give it a Default value, eg. 1.0.0.0.

AdventureWorks

Project Settings

SQLCLR

SQLCLR Build

Build

SQLCMD Variables

Build Events

Debug

Reference Paths

Code Analysis

Configuration: N/A Platform: N/A

System Defined:
\$(DefaultDataPath), \$(DefaultFilePrefix), \$(DatabaseName) and \$(DefaultLogPath) are system defined
Please click [here](#) for more information.

SQLCMD Variables:

	Variable	Default	Local
▶	\$(OctopusReleaseNumber)	1.0.0.0	
*			

Then add a new migration script to the project and specify the variable name in [SQLCMD format](#):

```
-- <Migration ID="05a44c70-fd8b-4f1f-9d50-5441daa0db58" />
GO
PRINT 'We're deploying release $(OctopusReleaseNumber)';
GO
```

When you deploy on your own machine, it will use the *Default* value provided in the [SQLCMD variables](#) tab. However when you deploy via Octopus, the value will be substituted with the current release number:



The screenshot shows the Octopus Deploy interface with a console window titled 'Deploy.ps1'. The output text shows the execution of a SQLCMD script. Key parts of the output include: 'Using default value for DatabaseName variable: AdventureWorks', 'Using default value for ForceDeployWithoutBaseline variable: False', '**Deploying to (local) because OctoOptionDeployToLocalInstance=True', 'Using Windows Authentication', 'Starting 'AdventureWorks' Database Deployment to '(local)', 'sqlcmd.exe -b -S "(local)" -v DatabaseName="AdventureWorks" ReleaseVersion="1.0.0.124" ForceDeployWithoutBaseline', 'DefaultDataPath="d:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\" DefaultLogPath="d:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\Backup\" OctopusReleaseNumber="D:\Octopus\Applications\Dev\AdventureWorks.Database\1.0.0.124\AdventureWorks_Package.sql"', '---- executing pre-deployment script "Pre-Deployment\01_Create_Database.sql" ----', '# Beginning transaction', 'Changed database context to 'AdventureWorks'.', '**** EXECUTING MIGRATION "Deploy-Once\1.2.0-VariablesDemo\003_20140404-1257_User.sql", ID: {05a44c70-fd8b-4f14-8000-000000000000}', 'We're deploying release 1.0.0.124', '**** FINISHED EXECUTING MIGRATION "Deploy-Once\1.2.0-VariablesDemo\003_20140404-1257_User.sql", ID: {05a44c70-fd8b-4f14-8000-000000000000}', '# Committing transaction', '---- executing post-deployment script "Post-Deployment\01_Finalize_Deployment.sql" ----', and 'Deployment completed successfully.'.

System variable tip: Remove the "." character from the name of the variable to use Octopus system variables in your migrations. In the above example, *Octopus.Release.Number* became *\$(OctopusReleaseNumber)*.

Mapping isn't limited to just the built-in Octopus variables: to use any of your own project variables (or any variable libraries that have been shared with your Octopus project), simply add the variable names to the *SQLCMD variables* tab within the database project in Visual Studio.

The scope that you specify in Octopus for each of your variables (e.g. whether the value applies to Development & Staging, or just Production) will be applied in the same way as a website or Windows service deployment.

Local vs Remote Tentacle deployments

In this tutorial, we have configured the project to deploy to a local instance of SQL Server. This requires that a Tentacle is installed on each of your SQL Server host servers. For example, if you have 2 SQL Server machines in each your environments (Dev, Test & Prod), you would need to install the agent on 6 servers.

To reduce maintenance of your deployment server infrastructure, you may prefer to deploy from a single "central" Tentacle instead, assuming your network layout permits it. Follows these steps to perform a remote deployment from a single deployment agent:

1. In your project variables, add a variable called *DatabaseServer* and scope it appropriately for each of your target environments and/or deployment step(s) (see [Specifying the Database Server/Name](#), above)
2. Install the **SQL Native Client 2012 (x86/x64)** and **SQL Command Line Utilities 2012 (x86/x64)** on the Tentacle server.
3. Install the **OctopusDeploy Tentacle** on a server that can communicate with the target SQL Server instances.
4. Add the machine on the Octopus *Environments* page, and under *Deployment*, specify all environments. Assign a new role name to the machine, e.g. *remote-sql-deployment*.
5. On the project *Process* page, update the package step with the new role name.

Rollback support

Once a migration is executed against a given database instance, and the transaction has been committed, the operations performed within that migration cannot be undone (save for the restoration of a backup taken prior to its deployment). [Read more about handling rollbacks](#)



Programmable object rollbacks

Note that, if you enable the [programmable objects feature](#) within your project, then any stored procedures, functions and views that changed within the */Programmable-Objects* folder will be automatically rolled-back if you subsequently deploy an older package to the target environment. To perform a partial rollback of programmable objects, simply revert the appropriate programmable objects in source control and then build and deploy a new package to the target environment.