Programmable Objects

By default, any time you make a change to a stored procedure, function or view, SQL Change Automation will generate a new incremental migration when you import that change into your SQL Change Automation project.

SQL Change Automation also supports scripting these Programmable Objects to individual T-SQL scripts. This allows you to source control each of your database objects. Changes to these objects can be branched, merged and annotated.

How do Programmable Object scripts work?

When you enable Programmable Objects, SQL Change Automation will initialize your project by extracting all of the supported objects (shown below) from your schema into individual script files. These files are re-runnable (idempotent) because they contain a header to replace the entire programmable object – "IF EXISTS THEN DROP / CREATE PROCEDURE".

Supported objects

The following types of programmable objects are supported for automated script generation:

- DDL Triggers
- User Defined Functions
- Stored procedures
- Views

In addition to scripting the objects themselves, SQL Change Automation will also include the following statements in your programmable object files:

- Permissions (GRANT/REVOKE)
- Extended Properties (sp_addextendedproperty)

If you'd like to deploy other types of objects not listed here, you can add your own idempotent (re-runnable) scripts by including an Additional Scripts folder in your project.

Table Trigger limitation

Please note that table triggers are not currently supported for automatic script generation: after enabling the Programmable Objects feature, table trigger objects will continue to be scripted as numerically-ordered Migration scripts, rather than as individual object script files.

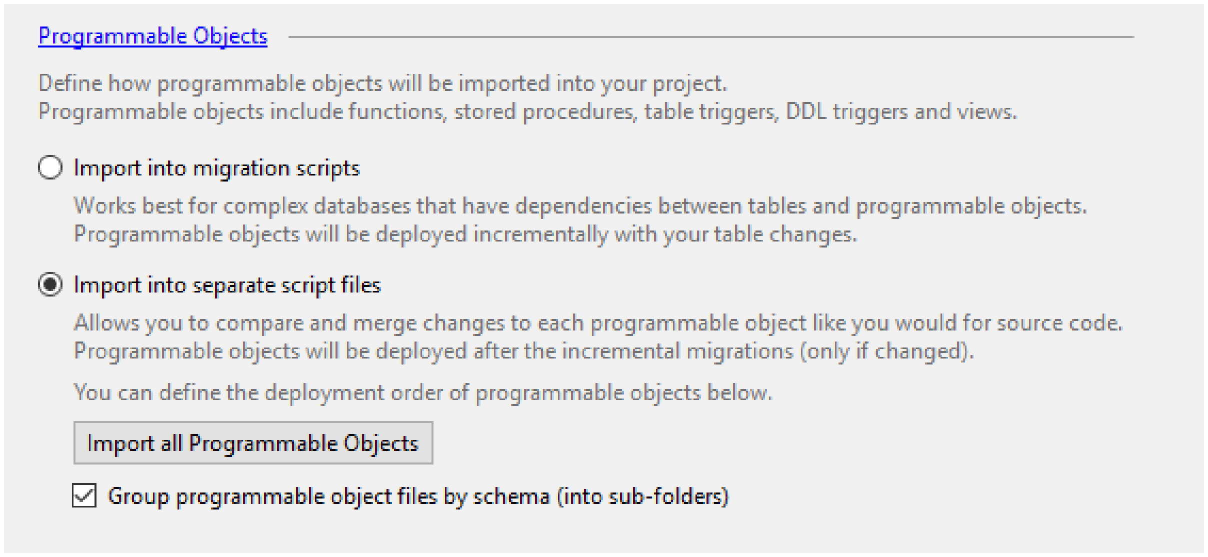
SSDT Users

If you've used SQL Server Data Tools (SSDT) database projects before, the file layout that SQL Change Automation creates for programmable objects may look familiar (see the below section for a screenshot of the SQL Change Automation layout). The primary difference that the files within the Programmable Objects folder of a SQL Change Automation project contain imperative logic, as opposed to the declarative logic of SSDT. This means that whatever T-SQL you put into these files will be executed in an unaltered way each time you make a change to the file.

Extracting existing Programmable Objects

If you are working with an existing database, you will first need to import existing programmable objects to individual script files within your SQL Change Automation project.

To perform this one-off step, start by enabling *Import into separate script files* and select Import all Programmable Objects in Project Settings.


Programmable Objects

Define how programmable objects will be imported into your project.
Programmable objects include functions, stored procedures, table triggers, DDL triggers and views.

○ Import into migration scripts

Works best for complex databases that have dependencies between tables and programmable objects.
Programmable objects will be deployed incrementally with your table changes.

● Import into separate script files

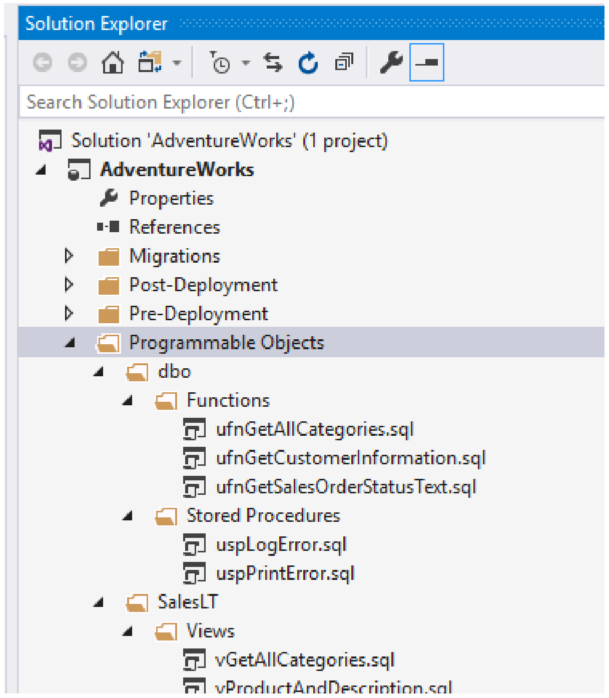
Allows you to compare and merge changes to each programmable object like you would for source code.
Programmable objects will be deployed after the incremental migrations (only if changed).

You can define the deployment order of programmable objects below.

☑ Group programmable object files by schema (into sub-folders)

Then confirm “**Yes**” to proceed with importing existing database objects into the project for offline editing.

When the import is complete, the SQL Change Automation tool-window will create a script for each of the programmable objects within your database:

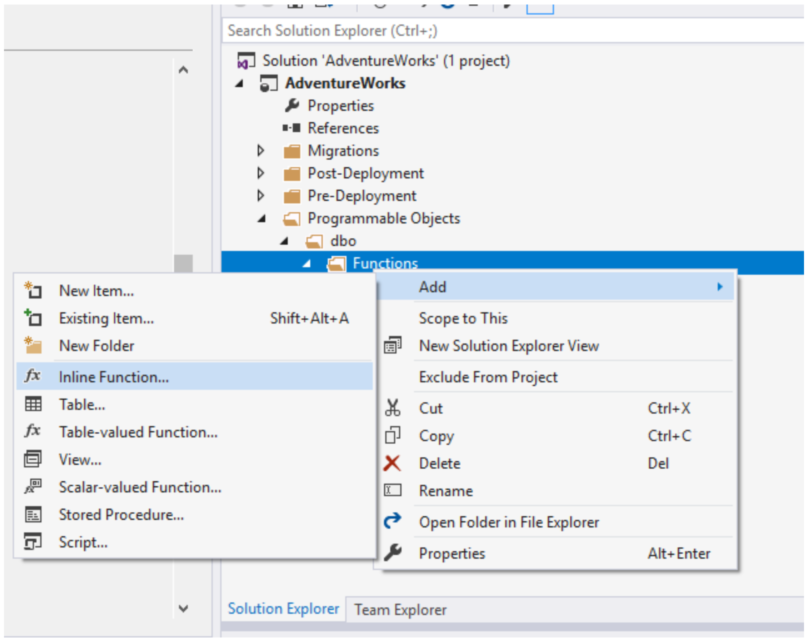


```
Solution 'AdventureWorks' (1 project)
  AdventureWorks
    Properties
    References
    Migrations
    Post-Deployment
    Pre-Deployment
    Programmable Objects
      dbo
        Functions
          ufnGetAllCategories.sql
          ufnGetCustomerInformation.sql
          ufnGetSalesOrderStatusText.sql
        Stored Procedures
          uspLogError.sql
          uspPrintError.sql
      SalesLT
        Views
          vGetAllCategories.sql
          vProductAndDescription.sql
```

Adding new Programmable Objects

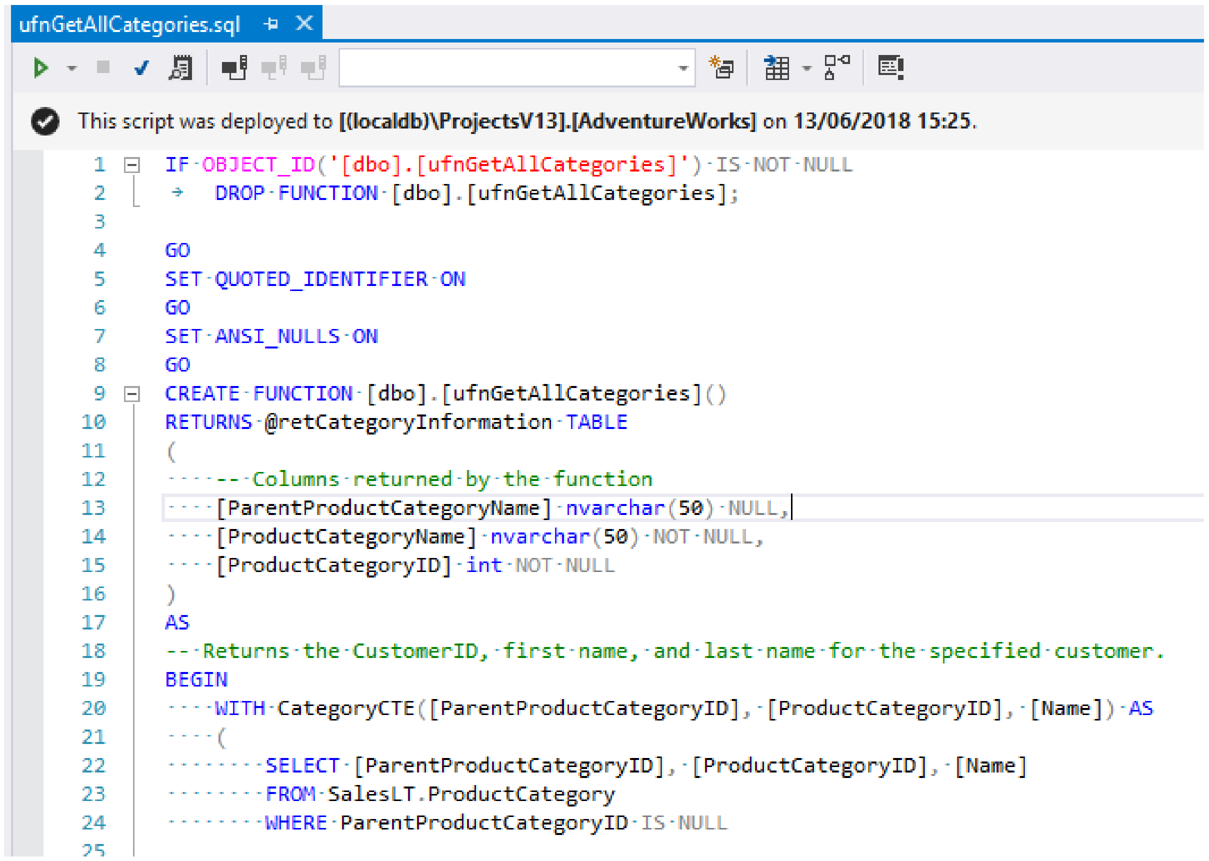
We recommend allowing SQL Change Automation to add Programmable Objects automatically using the **Import and Generate Script** command (which can be found within the SQL Change Automation tool-window).

If you want to add a programmable object manually, you can add new programmable objects to your project directly within *Solution Explorer*:



Editing Programmable Objects (Offline)

In each of the imported migrations, you'll notice that the script appears as "Deployed", just like an incremental migration.

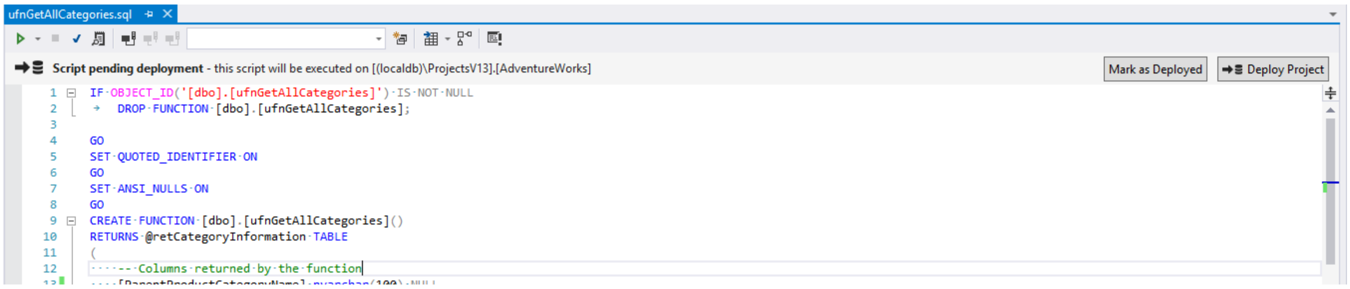


This script was deployed to **[(localdb)\ProjectsV13].[AdventureWorks]** on **13/06/2018 15:25**.

```sql
IF OBJECT_ID('[dbo].[ufnGetAllCategories]') IS NOT NULL
    DROP FUNCTION [dbo].[ufnGetAllCategories];

GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS ON
GO
CREATE FUNCTION [dbo].[ufnGetAllCategories]()
RETURNS @retCategoryInformation TABLE
(
    -- Columns returned by the function
    [ParentProductCategoryName] nvarchar(50) NULL,
    [ProductCategoryName] nvarchar(50) NOT NULL,
    [ProductCategoryID] int NOT NULL
)
AS
-- Returns the CustomerID, first name, and last name for the specified customer.
BEGIN
    WITH CategoryCTE([ParentProductCategoryID], [ProductCategoryID], [Name]) AS
    (
        SELECT [ParentProductCategoryID], [ProductCategoryID], [Name]
        FROM SalesLT.ProductCategory
        WHERE ParentProductCategoryID IS NULL
```

The difference being that, if we edit a programmable object, we are notified that the script will be executed again:



ufnGetAllCategories.sql

Script pending deployment - this script will be executed on [(localdb)\ProjectsV13].[AdventureWorks]

Mark as Deployed | Deploy Project

```sql
IF OBJECT_ID('[dbo].[ufnGetAllCategories]') IS NOT NULL
    DROP FUNCTION [dbo].[ufnGetAllCategories];

GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS ON
GO
CREATE FUNCTION [dbo].[ufnGetAllCategories]()
RETURNS @retCategoryInformation TABLE
(
    -- Columns returned by the function
```

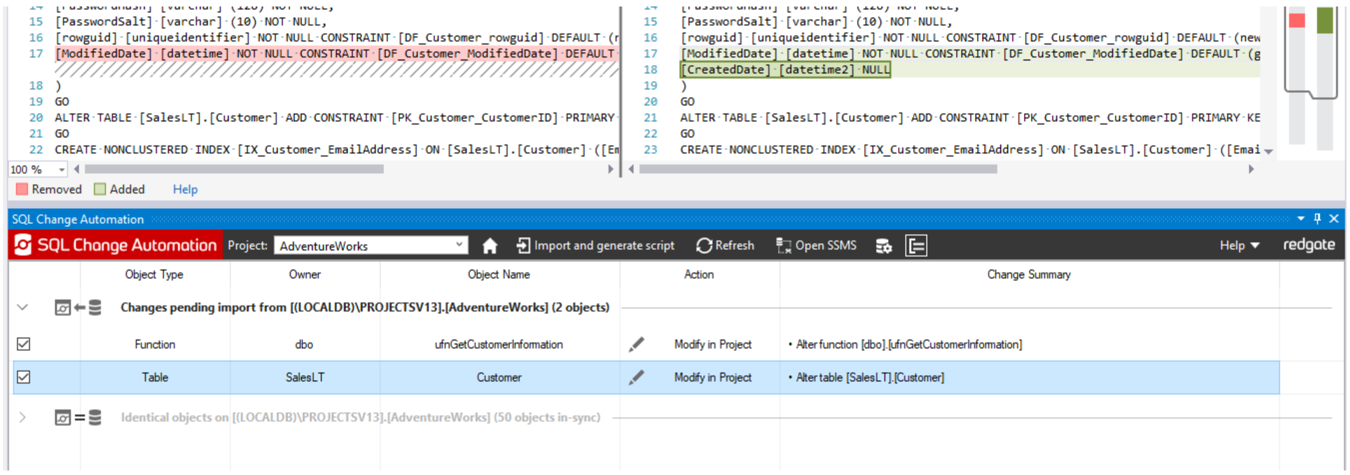
Deploying the project within Visual Studio, or indeed deploying from a third-party tool like Octopus Deploy or TeamCity, will apply the changes.

Editing Programmable Objects (Online)

So far we've looked at how you can edit your programmable objects in an *offline* manner using Visual Studio. However, SQL Change Automation also allows you to edit your objects in an *online* manner – i.e. directly on the database itself – and then import those changes back into your programmable object scripts.

For example, say you've just used *SQL Server Management Studio* to add a new column `[CreatedDate]` to the `[SalesLT].[Customer]` table. In addition, you've also altered a user-defined function to reference this new column.

After making these changes, you switch back to Visual Studio and click **Refresh** within the SQL Change Automation tool-window. You are then presented with both the table change and function change as *pending import*:



```sql
[PasswordSalt] [varchar] (10) NOT NULL,
[rowguid] [uniqueidentifier] NOT NULL CONSTRAINT [DF_Customer_rowguid] DEFAULT (r
[ModifiedDate] [datetime] NOT NULL CONSTRAINT [DF_Customer_ModifiedDate] DEFAULT

)
GO
ALTER TABLE [SalesLT].[Customer] ADD CONSTRAINT [PK_Customer_CustomerID] PRIMARY
GO
CREATE NONCLUSTERED INDEX [IX_Customer_EmailAddress] ON [SalesLT].[Customer] ([Em
```

```sql
[PasswordSalt] [varchar] (10) NOT NULL,
[rowguid] [uniqueidentifier] NOT NULL CONSTRAINT [DF_Customer_rowguid] DEFAULT (new
[ModifiedDate] [datetime] NOT NULL CONSTRAINT [DF_Customer_ModifiedDate] DEFAULT (g
[CreatedDate] [datetime2] NULL
)
GO
ALTER TABLE [SalesLT].[Customer] ADD CONSTRAINT [PK_Customer_CustomerID] PRIMARY KE
GO
CREATE NONCLUSTERED INDEX [IX_Customer_EmailAddress] ON [SalesLT].[Customer] ([Emai
```

🟥 Removed 🟩 Added Help

SQL Change Automation

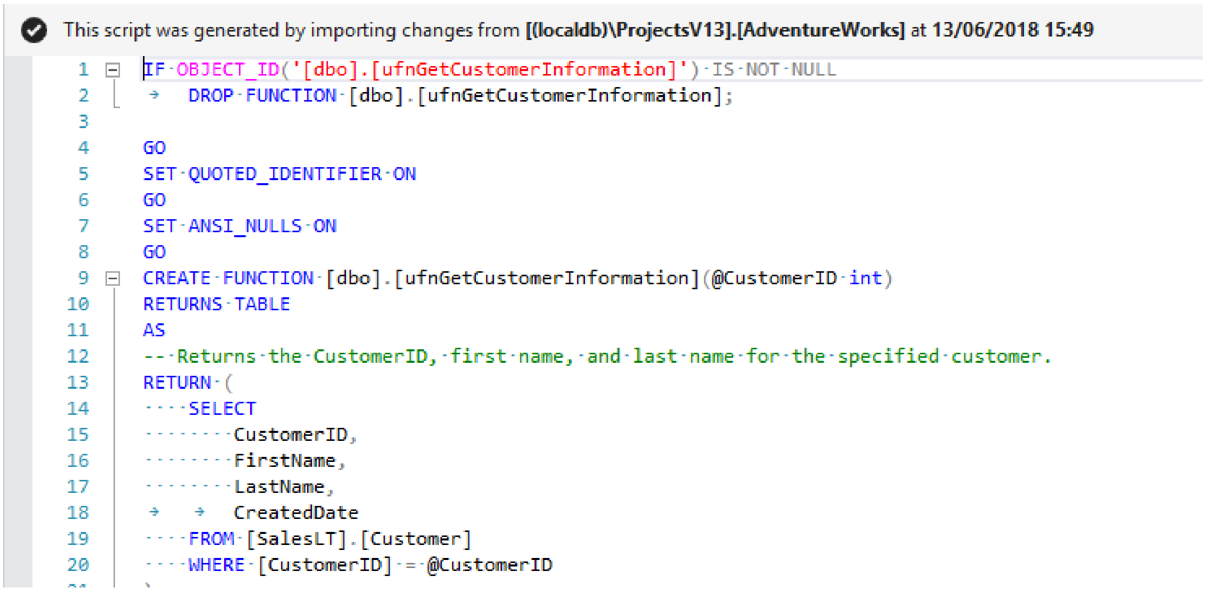
Changes pending import from [(LOCALDB)\PROJECTSV13].[AdventureWorks] (2 objects)

	Object Type	Owner	Object Name	Action	Change Summary
✓	Function	dbo	ufnGetCustomerInformation	Modify in Project	• Alter function [dbo].[ufnGetCustomerInformation]
✓	Table	SalesLT	Customer	Modify in Project	• Alter table [SalesLT].[Customer]

Identical objects on [(LOCALDB)\PROJECTSV13].[AdventureWorks] (50 objects in-sync)

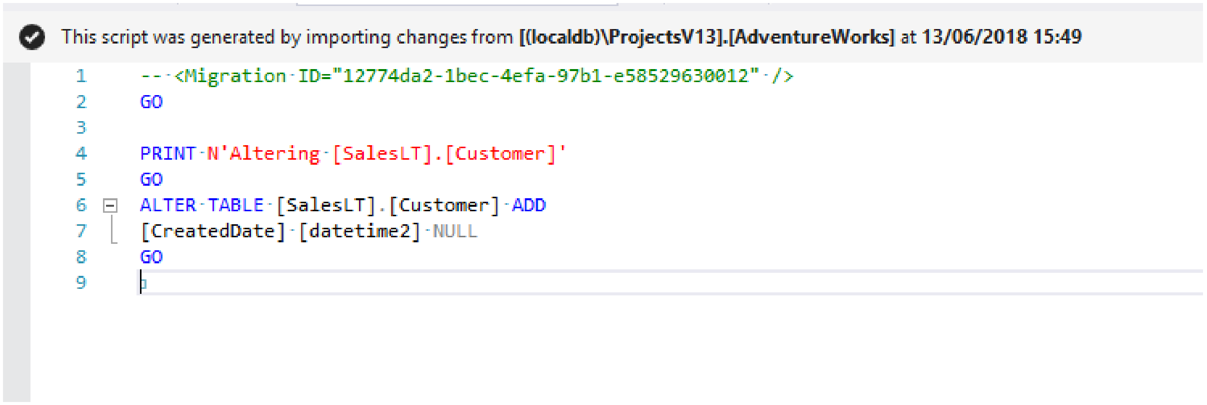
If you import both objects, the following changes will be made to your project**:**

- `Programmable Objects\Functions\dbo.ufnGetContactInformation.sql`
  Existing file updated to reflect function change
- `Migrations\1.1.0-AddCreateDate\001_20180613-1538_A.Developer.sql`
  New migration added, containing the add-column operation


This script was generated by importing changes from **[(localdb)\ProjectsV13].[AdventureWorks]** at **13/06/2018 15:49**

```
IF OBJECT_ID('[dbo].[ufnGetCustomerInformation]') IS NOT NULL
	DROP FUNCTION [dbo].[ufnGetCustomerInformation];

GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS ON
GO
CREATE FUNCTION [dbo].[ufnGetCustomerInformation](@CustomerID int)
RETURNS TABLE
AS
-- Returns the CustomerID, first name, and last name for the specified customer.
RETURN (
    SELECT
        CustomerID,
        FirstName,
        LastName,
		CreatedDate
    FROM [SalesLT].[Customer]
    WHERE [CustomerID] = @CustomerID
```


This script was generated by importing changes from **[(localdb)\ProjectsV13].[AdventureWorks]** at **13/06/2018 15:49**

```
-- <Migration ID="12774da2-1bec-4efa-97b1-e58529630012" />
GO

PRINT N'Altering [SalesLT].[Customer]'
GO
ALTER TABLE [SalesLT].[Customer] ADD
[CreatedDate] [datetime2] NULL
GO

```

After importing, click *Refresh (Verify Scripts)* to execute the migration scripts against the *Shadow* database. If successful, the tool-window will update, indicating that the project is now in-sync with the target database.

For more information about the shadow database, see Development and shadow databases.

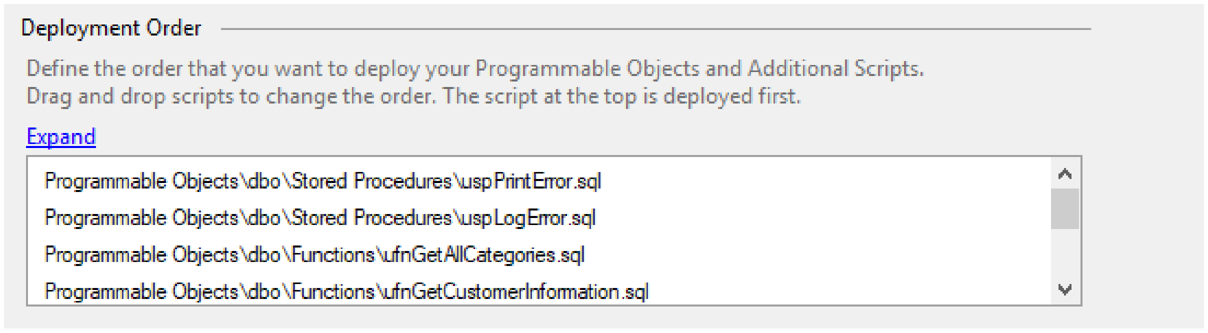
We recommend importing your changes prior to updating from source control, in case merge conflicts need to be resolved in any of your programmable objects.

If you do happen to update prior to synchronizing with your project, SQL Change Automation will force you to deploy your project in Visual Studio first to protect the integrity of your source controlled code. **This may mean the loss of work-in-progress changes in your target database.**

Controlling the order of execution

Programmable Objects are updated after all incremental migrations have been applied. This means you can couple a table change (eg. adding of a column) and the associated procedure change in the one deployment.

If you need to control the specific order that your programmable objects are deployed, this can be set within the Project Properties by dragging and dropping items into the required order:


Deployment Order

Define the order that you want to deploy your Programmable Objects and Additional Scripts.
Drag and drop scripts to change the order. The script at the top is deployed first.

Expand

```
Programmable Objects\dbo\Stored Procedures\uspPrintError.sql
Programmable Objects\dbo\Stored Procedures\uspLogError.sql
Programmable Objects\dbo\Functions\ufnGetAllCategories.sql
Programmable Objects\dbo\Functions\ufnGetCustomerInformation.sql
```

Group by schema (into sub-folders)

By default, SQL Change Automation groups the objects in the Programmable Objects folder by object type, and prefixes each file with the appropriate schema name (e.g. *Stored Procedures\dbo.MyProc.sql*). If, however, you would like to also group your object files into schema sub-folders (e.g. *Stored Procedures\dbo\MyProc.sql*), you can do so with a bit of editing to your SQL Change Automation project file.

To group your programmable object scripts by schema, edit your `.sqlproj` file (e.g. in Notepad) and add the following under the root node of the file:

```
<PropertyGroup>
    <DeployChangesImportSchemaFolders>True</DeployChangesImportSchemaFolders>
</PropertyGroup>
```

Note for existing database projects

The new folder structure will only apply to new objects that you import using the SQL Change Automation tool-window, so unfortunately you will need to move any existing files into the appropriate sub-folders yourself.

Permission Handling

After importing your programmable objects, you'll notice that SQL Change Automation scripts each of your objects using DROP and CREATE statements. This means that, whenever you deploy your object, the permissions assigned to that object will be reset.

To avoid losing the permissions assigned to your objects in each deployment, SQL Change Automation appends GRANT/REVOKE statements to each programmable object script, based on the database environment that you imported the scripts from. This works fine for permissions that are stored in source control, however if you need to control permissions at the object level directly in the target environments, we recommend assigning the permissions via role objects, rather than assigning them at the individual user/group level.

If the assigning of user permissions directly in your databases is unavoidable, then you may prefer to follow a "pure" migrations approach, instead of using programmable objects. This approach results in most stored procedure/view/function/etc changes being made using the ALTER statement, which preserves the permission structures between deployments.

The reason for the difference in scripting behavior between migrations (ALTER) and programmable objects (DROP/CREATE) is to ensure that the programmable object scripts are idempotent; since those scripts may be executed many times, using alter might leave behind some state in the database, so it's safest to recreate the object with each deployment.