

CSLA Templates

Archive of the CodeSmith Tools documentation for CSLA Templates, as published on the CodeSmith Confluence site. It is kept for reference and is no longer updated. Current documentation is at www.codesmithtools.com/docs.

23 pages.

Contents

Home

Installing and Upgrading

 Downloading the CSLA templates

 Upgrading the CSLA templates

Introduction and Tutorials

 Main Features

 What's New

 Tutorials

 Getting Started

 Using the QuickStart template

 Inspecting the generated solution

 Configuring Business Object Types

Generating from Views and Stored Procedures

Where to go from here

Advanced Topics

 Renaming classes and properties

 Using partial methods

 Extending Stored Procedures

 Exploring the sample applications

 Using Extended Properties

Using the Templates

 Entities Template

 Quick Start Template

Home

Generate a CSLA framework for your application in minutes.



Rockford Lhotka's CSLA .NET framework is an **Application Development Framework**.

These templates further reduce the cost of building and maintaining applications.

CSLA .NET Templates Documentation

All of our documentation is provided in a friendly wiki format.

Please use the navigation tree on the left to locate the subject that you are interested in.

If you can not find what you are looking for, please feel free to [contact us](#).

CSLA .NET Documentation

We try our best to stay true to the CSLA methodologies and best practices in our templates. This allows you to use previous CSLA knowledge with the generated code. As such, we highly recommend reading the [CSLA E-Books](#) while using our templates for a deeper understanding of CSLA.

Additional Resources

[Official Website](#)

[Issue Tracker](#)

Installing and Upgrading

The following pages will go over how to download, install and upgrade the CSLA templates.

- [Downloading the CSLA templates.](#)
- Installing the CSLA Templates.
- [Upgrading the CSLA templates.](#)

Installing the CSLA Templates

This section will guide you through installing the latest version of the CSLA Templates.

Installing the downloaded templates from the website

1. Please [follow these steps](#) outlined in the "Downloading the latest Templates" section after downloading the CSLA templates from the internet.
2. Install the CSLA Templates by extracting the downloaded templates to:
 1. Any non networked drive.
 2. A previous version of the templates. You can do this by overwriting your existing templates and extracting the new templates over them.
3. Run the [QuickStart template](#) to get up and running. Please follow the [upgrade guide](#), if upgrading from a previous version.

Installing the templates from NuGet

1. Install the [CodeSmith.Generator.CSLA NuGet package](#) to a new or existing CSharp or Visual Basic project.
 1. Open the [NuGet Package Manager Console](#).
 2. Type **Install-Package CodeSmith.Generator.CSLA** into the console and press enter.
2. Configure the newly created Entities.csp CodeSmith Generator Project file.
 1. You will first need to open the [Manage Outputs](#) dialog. You can do this by right clicking on the Entities.csp file and selecting [Manage Outputs](#).
 2. Next, double click on the [Entities.cst](#) template output or right click and select Edit Output.
 3. Next, configure what gets generated by choosing a Source Database. You can accomplish this by selecting the (...) button next to the SourceDatabase property.
 4. Finally, save your configured properties by clicking the OK button.
3. Right click the Entities.csp and select Generate.

Downloading the CSLA templates

This section will walk you through downloading the CSLA Templates.

CodeSmith Generator automatically ships with the latest version of templates. If you just downloaded the latest version of CodeSmith Generator, chances are you are already using the latest release.

Please **follow these steps** outlined in the "Downloading the latest Templates" section after downloading any template from the internet.

Latest Releases

The latest version of the CLSA templates can always be found on **GitHub** or **NuGet**.

The CLSA templates require the latest version of **CodeSmith Generator**.

Other Downloads

You can also download sample projects and the source code to the templates. The sections below will guide you through those processes.

Sample Projects

You can download the latest sample applications via **GitHub releases**. The downloaded sample contains examples for both C# and Visual Basic.

Please see the following **document** for more information on the Sample Applications.

Source Code

If you like to get into the code and see how things work, the CSLA Templates source is available to view online or can be checked out using git from **here**.

Upgrading the CSLA templates

We strive to ensure that the CSLA templates are backwards compatible with previous versions. However, in some circumstances we break backwards compatibility to progress the platform. When we do break backwards compatibility, we ensure that the benefits of breaking compatibility greatly outway the benefits of not breaking compatibility. Please refer to this document when upgrading the CSLA Templates to ensure that you have the smoothest experience possible.

Upgrading from previous versions of the CSLA Templates

The following steps will guide you through updating your existing project to consume the latest version of the templates.

Updating Visual Studio Solution dependencies

The first step is to ensure that your solution meets the requirements of the version you are wanting to upgrade to.

Lets assume your Visual Studio solution targets Visual Studio 2008 and your projects target .NET 3.5 / CSLA 3.x. You are wanting to upgrade to CSLA 4.x. You must first upgrade to a version of Visual Studio that supports .NET 4.0 and upgrade your projects to .NET 4.0.

After you have verified that your projects are targeting the correct version of the .NET Framework, the next step is to update your assembly dependencies. You can do this by right clicking the CSLA assembly references in Solution Explorer and selecting remove. You must do this for all of your projects. This is required because CodeSmith Generator will not update existing assembly references, it will only add them if they do not exist.

Updating existing CodeSmith Generator Project files

The next step is to update all existing **Generator Project files**. The Generator Project files will continue to point to the previous version of the templates if you don't update each template output's paths. The fastest way to update a Generator Project file is to open it with notepad. Once you have opened the Generator Project file, you will want to **update each propertySet's template node path** to point to the current location as shown below.

change:

```
<propertySet template="C:\PREVIOUS.PATH.TO.OLD.TEMPLATES\Csla\Entities.cst" ....>
```

to:

```
<propertySet template="C:\NEW.PATH.TO.LATEST.TEMPLATES\Csla\Entities.cst" ....>
```

Once, this has been done, save the file.

Next, open the updated Project file with **Manage Outputs** and ensure everything compiles. If it doesn't, please ensure that you followed the current step, Please contact support if you continue to run into issues.

You will want to ensure that all properties are set correctly.

Regenerate

The next step is to regenerate the solution. You can do this by right clicking the Generator Project and selecting build.

Next, check for template compile and generation errors in the **output window**.

- If you see any errors about template file not found, then you will need to go to the previous step and ensure that the template output mentioned has the correct propertySet template node path.
- If you are seeing an error that deals with a template property not being able to be set. Please check the release notes to see if a property has been removed. If it has or you are getting an error, you can follow the previous step, but remove the property XML node in question and repopulate it via Manage Outputs. This is very rare if you do encounter this issue.
- If you are unable to compile or generate the template, please **contact support** with the full stack trace / output window and we will take a look into the issue.

If you receive any project compile time bugs and are:

- upgrading from a minor version of CSLA (EG., 4.1 to 4.2), then chances are you are running into a bug with the templates. Please **contact support** and we will get this taken care of.
- upgrading to a major versions of CSLA (EG., 3.x to 4.x), then please check the release notes or upgrade documentation below.

Upgrading from CSLA 3.x to 4.x

It is highly recommended that you read over the **official CSLA Documentation when upgrading between versions of CSLA**. This ensures that you are aware of all the changes that will occur in your application.

In addition to following the steps above for upgrading your templates, you will also need to complete a few steps to ensure everything compiles when upgrading from 3.x to 4.x. Please follow the sections below after upgrading.

Updating existing CodeSmith Generator Project files

You will need to open **Manage Outputs** for every project file you updated and change the FrameworkVersion property from v35 to v40. This tells the templates to generate CSLA 4.x compliant code.

Update existing non generated partial classes

Before you continue this step, please ensure that you have followed the previous upgrade guide and have regenerated. You will notice that there are some build errors. This is due to the entity partial classes not being regenerated and there are some required changes for Rules or for Silverlight when migrating to CSLA 4.x.

Update Imported Assemblies

There were major changes to Rules in CSLA 4.0. You will need to update **every business object's** non generated partial class (E.G., Account.cs or Account.vb).

C#	Visual Basic
Change all of your Imports Statements from:	
using Csla.Validation;	Imports Csla.Validation
To	
using Csla.Rules;	Imports Csla.Rules

C#	Visual Basic
Update existing method signatures from (as well as existing rule logic inside of the method):	
protected override void AddAuthorizationRules()	Protected Overrides Sub AddAuthorizationRules()
To:	

```
public static void  
AddObjectAuthorizationRules()
```

```
Protected Shared Sub  
AddObjectAuthorizationRules()
```

Introduction and Tutorials

For our Introduction and Tutorials of the CSLA templates we offer the below sections.

Main Features

There are many features available in the CSLA templates. In our Main Features page you will see the features that really set our CSLA templates apart from the rest and how powerful they can be.

What's New

Check out our What's New section to see the features that we added for each new version.

Tutorials

Tutorials is for any CodeSmith Generator user wanting to know more about our CSLA templates. It was designed to help get users up and running in no time flat.

Main Features

The templates have the following features.

- Generation against CSLA 3.8 or CSLA 4.3.
- Object Factory support.
- Silverlight 4 Support.
- Supports generating all object types (E.G., EditableRoot, EditableChild, DynamicRoot, Lists, Criteria...)!
- Generate business objects from **Tables, Views and Stored [procedures](#)**.
- **Quick start** - get up and running quickly
- Detects relationships and generates child objects and lists.
- Generates **partial classes and methods** making customizing the generated code easy.
- Generates complete Parameterized SQL or SQL Stored Procedure driven Data Access Layer (DAL).
- Overridable rules.
- **Column and Table renaming functionality**.

What's New

For a full change log of all the new features and bug fixes for each version of CSLA be sure to look at the [CSLA Release Blog](#).

CSLA 4.0

- Added support for [Generator 6.5](#) (**The CSLA 4.0 Templates require Generator 6.5**).
- Added support for [generating from Stored Procedures and Views](#).
- Added support for CSLA 4.3.10.
- Updated the templates to use the latest version of SchemaHelper (many improvements to association detection, naming conventions and much more).
- Added various List Business Object enhancements. The generated list Business Objects now contain the following methods Get, GetByCriteria, Remove, Contains and ContainsDeleted.
- Added support for generating off of key less Data Sources.
- Added the ability to exclude associations from being generated.
- Templates now generating Async methods by default.
- Added Beta LINQ-to-SQL Support (contrib: Joe McPherson). This requires generating PLINQO for LINQ to SQL and is not documented.
- Improved support for Silverlight, NameValueLists, ObjectFactory, Composite Keys and Child relationships.
- Added support for IncludeExpressions, you can now specify a Regex for only the entities you wish to include.
- Added support for better Friendly Names in regards to generated properties. *To override the default Generated Friendly Name, simply add an ExtendedProperty to the database column called CS_FriendlyName.*
- Improved workflow for Criteria generation. They will now be automatically generated based on the entities you select.
- When the DataAccessImplementation is set to none, the data access partial classes will not be regenerated.

CSLA 3.0.2

- Fixed a bug where a FileLoadException would be thrown when selecting a table on any master template deriving from EntityCodeTemplate.

CSLA 3.0.1

- Added support for CSLA 4.0.1
- Added Silverlight support for the Command Object.
- Added a ChildLoader to facilitate Silverlight lazy loading.
- Added a partial method override for OnChildLoading.
- Added the ability to specify a child class name. This is so you can create a list and specify what the child type should be.

CSLA 3.0

- Added support for CSLA 4.0.
 - Updated templates to support the new Rules Engine.

- Updated support for the latest CSLA 4.0 Business base classes.
- Added Silverlight 4.0 support (Requires CSLA 4.0).
- Updated Framework Samples with a Silverlight project.
 - Supports Lazy Loading of Child properties.
 - Preprocessor directives.
- Added Support for Visual Studio 2010 and the .NET Framework 4.0.
 - Added EditableRootListBase or DynamicListBase Business Object Type Template.
 - Added New Partial Overrides to make customization even easier.
 - Increased business objects naming detection support.
 - Added Namespace support for the Visual Basic Templates *Breaking Change.
 - Added better support for composite keys and guid columns.
 - Added better support for grand -children and great grand-children.
 - Updated Collections to not throw an exception when no records are found *Breaking Change.
 - Added Collection Unit Tests to the sample applications and updated NUnit.
 - Added beta unit tests templates for the CSharp Version (These were contributed by the community and will be built and expanded on in future releases).
 - Added support for passing down parent information to children when creating new children.

CSLA 2.0

- Added SQL Stored Procedure support
- Added Object Factory support.
- Added merge region support as well as support to actively generate stored procedures.
- Added ExistsCommand, this does a database lookup to see if an item exists in the database.
- Added support for generating ReadOnly business objects along side of an Editable business objects.
- Added Column and Table renaming support.
- Moved all repetitive template logic into sub templates (common folder)
- Added support for the new ByPassPropertyCheck in CSLA 3.8.
- Updated the templates to CSLA 3.8.2.
- Fixed detection logic for GetBy Criteria.
- Fixed many bugs dealing with Child objects and how they are populated.
- Added partial method hooks for the DataPortal methods.
- Updated the templates to throw an exception when a record doesn't exist.
- Added support to rename a non identity primary key column.
- Fixed many other [template bugs](#).

CSLA 1.1

- Visual Basic support
- Added generated authorization rules based on work described [here](#).
- Fixed a bug where a Column that is a primary key and a foreign key was not handled correctly.
- Fixed a bug where many to many tables would be excluded from the Entities collection.
- The CleanExpression and IgnoreExpression are now case insensitive.
- Fixed many small template bugs.

CSLA 1.0

- Working data access layer with concurrency support.
- Support for completely overriding the generated rules.
- Generates three partial classes per business object.
- Generates a common business, data and UI layer.
- Quick start.
- Implements all 14 **object types**.
- Detects and generates properties and child objects for the following database relationships (One to Zero or One, One to One, One to Many, Many to One, Many to Many).
- All DateTime data types are generated as Csla.SmartDate
- Correctly handles Binary Database fields (E.G Images, TimeStamp)

Tutorials

The CodeSmith Generator CSLA template tutorial includes the various sections which will quickly help you become a master of Code Generation with the CSLA templates. Although there's a lot of depth to the CSLA templates, you can get started with it quickly. In this section of the documentation, we'll walk you through a few common code generation scenarios.

Getting Started

The best way to understand the power of the CSLA template is to try it out. Although the CSLA templates have many advanced features, you can begin using it to help produce code without mastering all of those features. In this section, you'll learn how to use the CSLA QuickStart to quickly get up to speed.

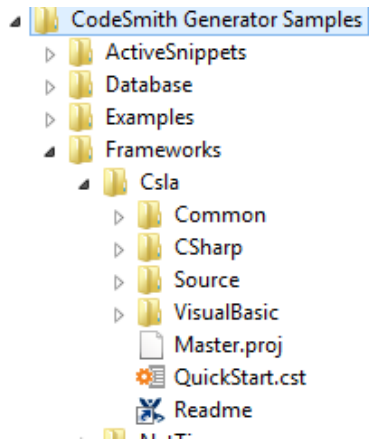
Getting Started

You can get up and running with our CSLA Templates in a matter of seconds. The following guide will walk you through generating your first CSLA application.

Next: [Using the QuickStart template](#)

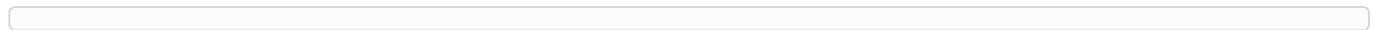
Using the QuickStart template

You will first need to [open Template Explorer](#) to display a list of templates that you can execute. Once opened, you will need to expand the CodeSmith Generator Samples\Frameworks\CSLA folder (by double clicking on each closed folder).

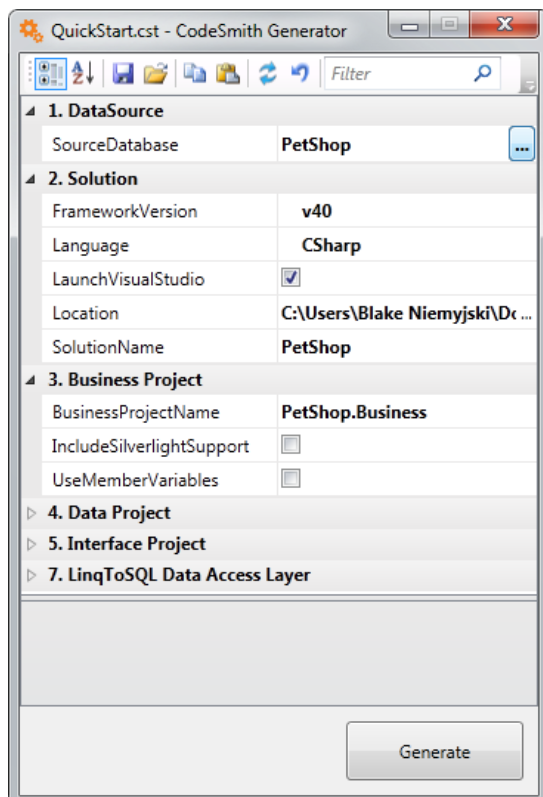


Once you see the QuickStart.cst Template, double click it or right click it and select Execute.

Now it's time to configure the QuickStart's [template properties](#). Choose a Source Database by clicking the select "..." button for the Source Database property.



The QuickStart will now populate the property grid with the the default settings.



Finally, configure the settings to meet your requirements and click the Generate button.



In the next step, we will learn about the generated Visual Studio solution that is created by the QuickStart.

Next: [Inspecting the generated solution](#)

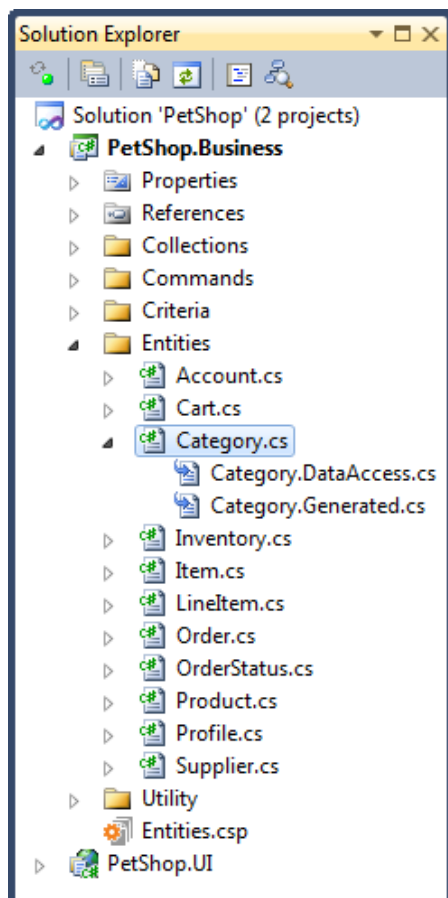
Inspecting the generated solution

By default, the QuickStart will automatically open the generated Visual Studio solution. If this does not happen, you will need to open the generated solution. The solution will be located in the directory you specified in the [QuickStart's Location](#) property.

Once the generated Visual Studio solution is opened, you will need to build the solution to see the generated code as shown in the image below.

CodeSmith Generator uses the Entities.csp [Generator Project File](#) (CSP) to generate code for your project. This file is created by the QuickStart with all of your information pre-populated from the previous step. Generator is smart enough to figure out if you have already configured the Entities CSP file. If you haven't, Generator will automatically try to resolve the correct business object types (E.G., Criteria, EditableRoot, EditableRootList...) to generate based off of your database relationships. The generated code will be separated into multiple folders.

Folder Name	Description
Collections	Contains all non-collection business object types.
Commands	Contains the data access implementation.
Criteria	Contains properties, factory methods and required rules.
Entities	Contains all non-collection business object types.
Utility	Contains any utilities classes that may be used by the generated code.



If you are using Visual Basic, then you will have to click the Show All files button in the Solution Explorer to see the **partial classes**.

For the simplicity of this walk through, I'll choose to talk about the Category business object. The Category business object is shown in the image above. As you might have noticed, there are multiple files for the Category business object. These files are called **partial classes**, which allows us to separate the different responsibilities of a business object.

Partial Class File Name	Editable	Description
Category.cs	Yes	Contains all of your custom code.
Category. DataAccess .cs	No	Contains the data access implementation.
Category. Generated .cs	No	Contains properties, factory methods and required rules.

If you are generating Object Factory support, then the DataAccess partial class will not be generated.

Partial classes allow for quick Silverlight Integration. All you need to do is add a project item link to the business object partial classes while executing the DataAccess partial class.

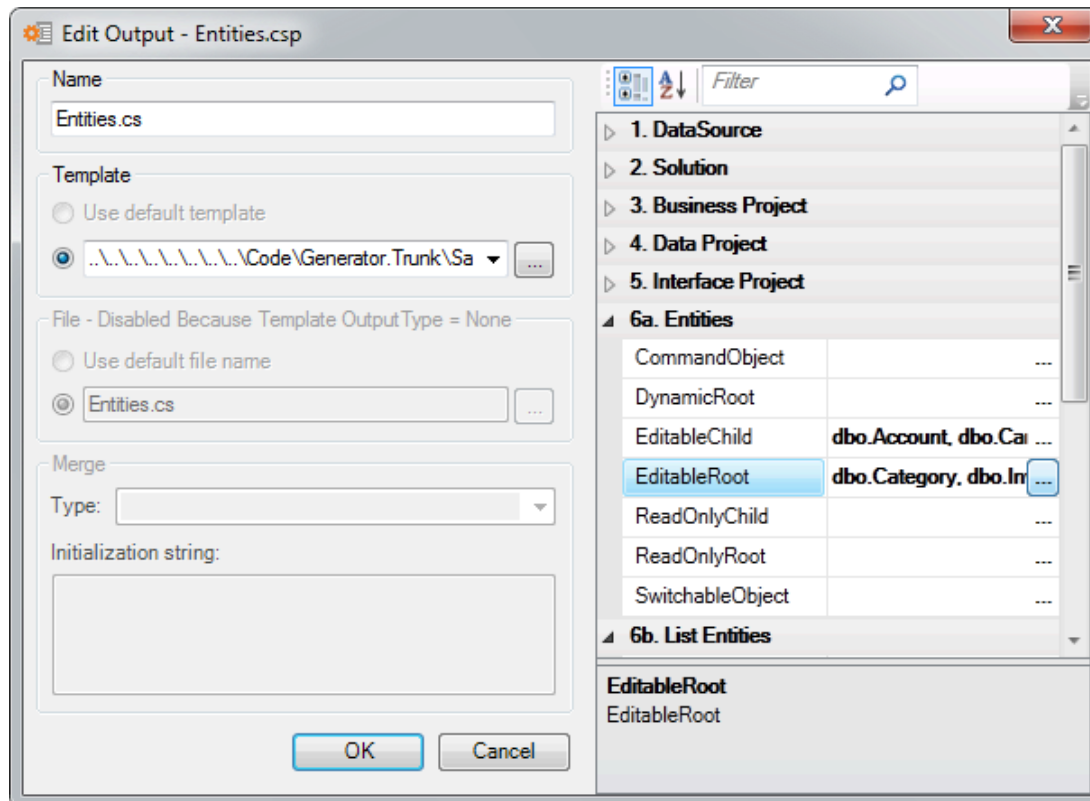
In the next step, we will cover how to configure Category's business object type.

Next: **Configuring Business Object Types**

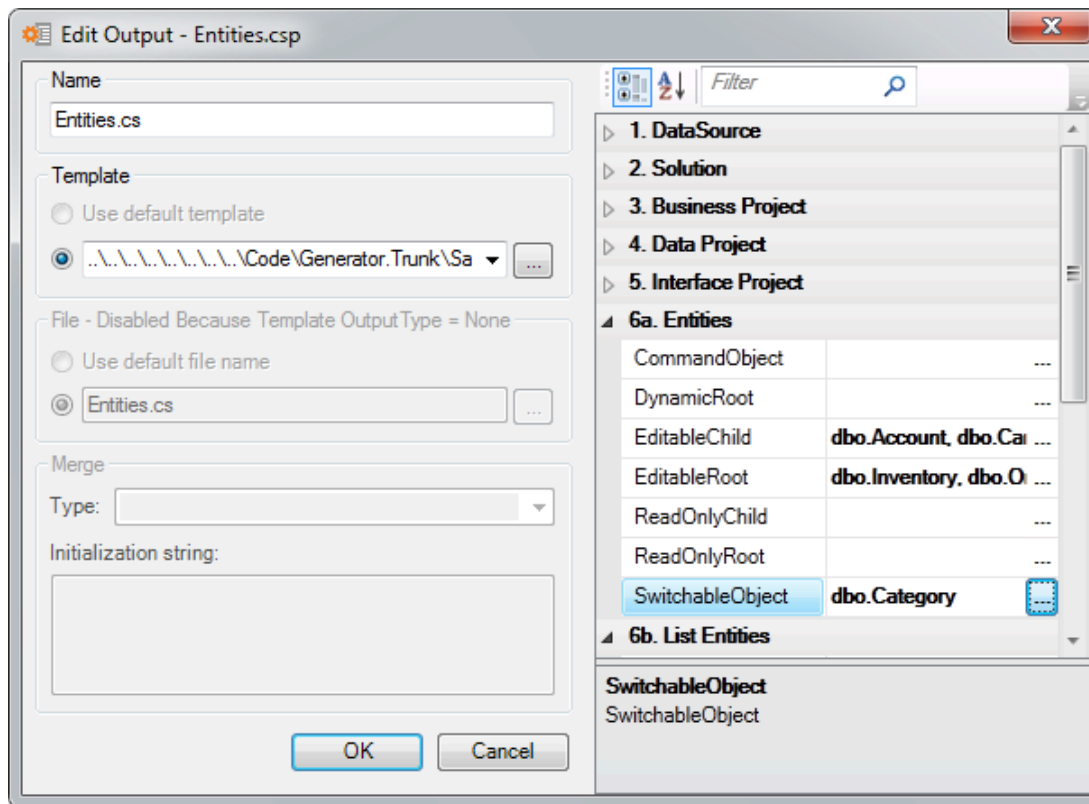
Configuring Business Object Types

In this section we are going to cover how to change Category's business object type. You may want to change how the business object behaves by changing its type. In this example, we want our Category business object to act as both a parent and a child business object. To do this, we will need to change the type from EditableRoot to Switchable Object.

You can configure each entities business object type quickly and easily through **Manage Outputs**. Lets start by right clicking the Entities.csp template and selecting Manage Outputs. After the manage outputs dialog comes up double click on the Entities.cst item. You can control exactly what gets generated right inside of Edit Outputs.



Now lets change Category from an EditableRoot to SwitchableObject. We can do this by selecting the "..." button for the SwitchableObject Property. Next, we would hold the control key to allow selection of multiple items and select Category.



As you can see, the Category database table was removed from EditableRoot and placed into Switchable Object!

The Entities template is smart enough to realize that you may want both an Editable Object as well as a Read-Only object. When the template detects this, it will not remove the previous choice and allow you to generate two business objects.

Now we can save our settings by clicking the OK button on the Edit Output dialog. When we regenerate the project, the criteria business object will be regenerated as an SwitchableObject.

Any time you need to change a setting, you can just go back into the Edit Output dialog and change business object types, Data Access Implementation and much more!

In the next step we are going to cover generating business objects from a stored procedure or view.

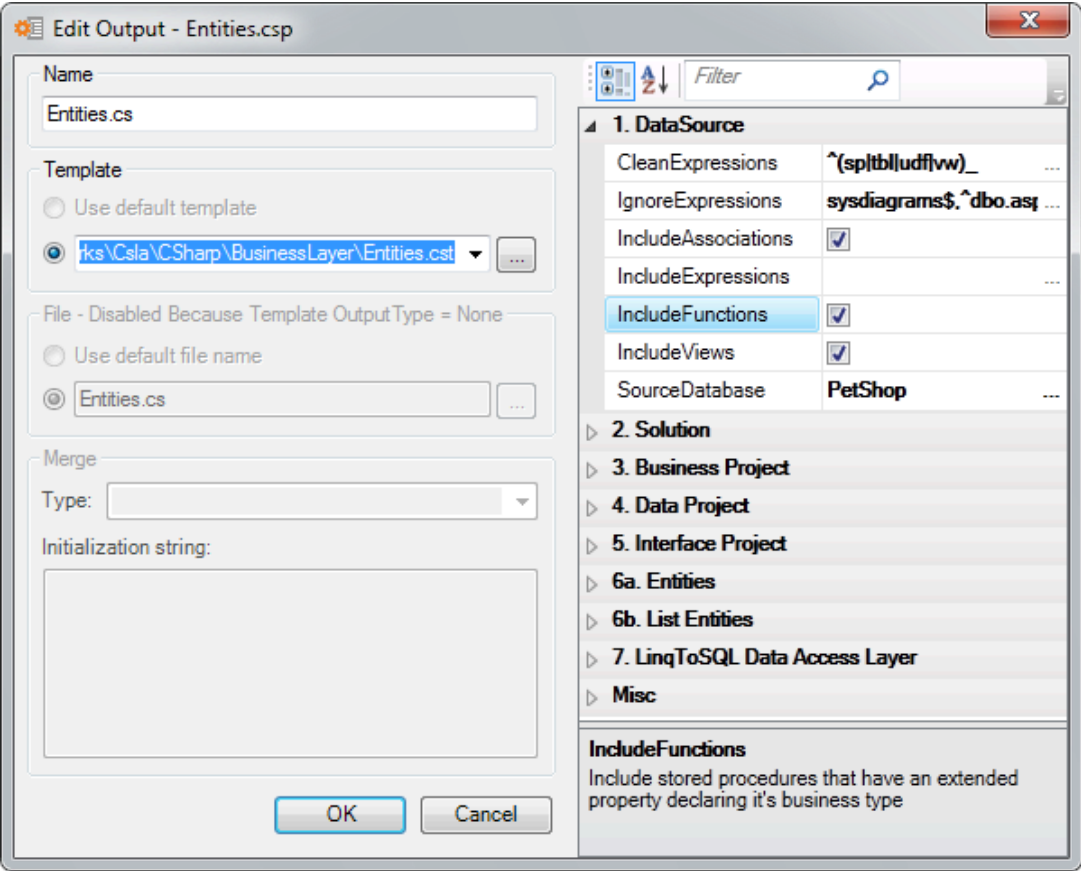
Next: [Generating from Views and Stored Procedures](#)

Generating from Views and Stored Procedures

The following section will show you how you can take advantage of generating against stored procedures and views.

We will start by going back into our Entities.csp Edit Output dialog as shown in the [previous step](#). You can open this dialog by right clicking the Entities.csp template and selecting Manage Outputs. After the manage outputs dialog comes up, double click on the Entities.cst item.

Once the Entities Edit Output dialog is opened, you will need to check the **IncludeFunctions** and **IncludeViews** properties as shown below. When we set these two properties to **true**, the templates will enable stored procedure and view support.



Since earlier versions of the templates didn't support views and stored procedures, we decided against breaking backwards compatibility with the Entities output. This saves you from having to reconfigure all of your outputs when [upgrading](#). But this also prevents you from selecting what business object type a view and stored procedure will be generated as. Since you can't choose a view or stored procedure inside of the Entities.csp Edit Output screen, you might be wondering "How do I configure my business object types then?" The solution is to control this by [adding an Extended Property](#) to your stored procedure or view!

Here is a list of the valid Extended Property names that can be added to an existing view or stored procedure.

Extended Property Name	Extended Property Value	Business Object Type
------------------------	-------------------------	----------------------

DynamicRoot		Dynamic Root
EditableChild		Editable Child
EditableRoot		Editable Root
ReadOnlyChild		Read Only Child
ReadOnlyRoot		Read Only Root
SwitchableObject		Switchable Object
DynamicRootList		Dynamic Root List
EditableRootList		Editable Root List
DynamicListBase		Dynamic List Base
EditableChildList		Editable Child List
ReadOnlyList		Read Only List
ReadOnlyChildList		Read Only Child List
NameValueList		Name Value List

After you have added an extended property to your view or stored procedure. All you need to do is regenerate and watch your new business objects be added to the project!

Next: [Where to go from here](#)

Where to go from here

That's all you need to know to generate valid CSLA code using the CSLA templates that are included with CodeSmith Generator:

1. [Launch Template Explorer](#)
2. Supply values for the [QuickStart's properties](#)
3. Generate the code

But you can do much more than just use the QuickStart and Entities templates. Here are some places to continue your exploration:

- [Advanced Topics](#)
- [Using the Templates](#)

Advanced Topics

The following section will cover some advanced topics that deal with customizing and extending the generated code.

Renaming classes and properties

Many of you might have a legacy database with those ugly prefixes, or things you wish you could refactor but just can't. Today, you're in luck, as CodeSmith Generator is going to save the day once again! By default we will strip any invalid characters from your property names like prefixed digits.

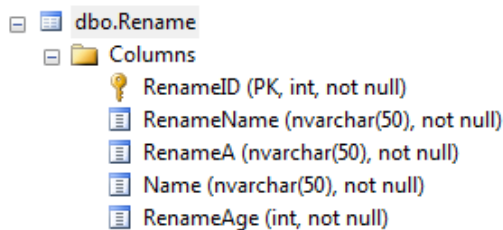
Renaming a class or property name will not affect the generated stored procedures or parameterized SQL column names.

The methods below also work for renaming class names.

In this example, we are going to be renaming column names in the following database table:

```
CREATE TABLE [dbo].[Rename](
    [RenameID] [int] NOT NULL,
    [RenameName] [nvarchar](50) NOT NULL,
    [RenameA] [nvarchar](50) NOT NULL,
    [Name] [nvarchar](50) NOT NULL,
    [RenameAge] [int] NOT NULL,
    CONSTRAINT [PK_Rename] PRIMARY KEY CLUSTERED
(
    [RenameID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
```

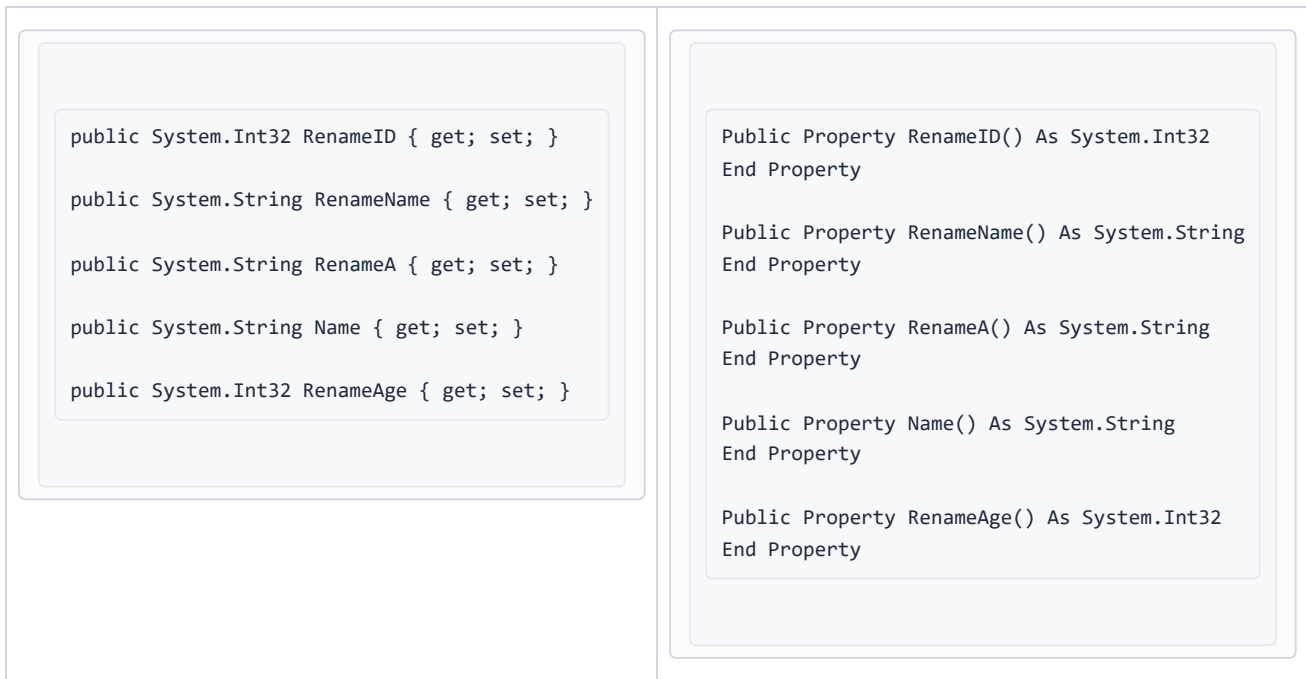


As you can see this would cover a few use cases when it comes to legacy database tables. By default, when this is generated, the code will look like this.

Please note that all code has been simplified to show the results only.

C#

Visual Basic



As you can see the name's haven't changed at all. The CSLA templates offer you three methods of overriding a property or class name.

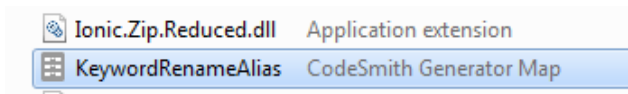
Method 1

This method can be a lot of work, but for renaming a few columns, it is the quickest and easiest way to rename property names. Inside of the CSLA Template folder, there will be a folder called **Common**.

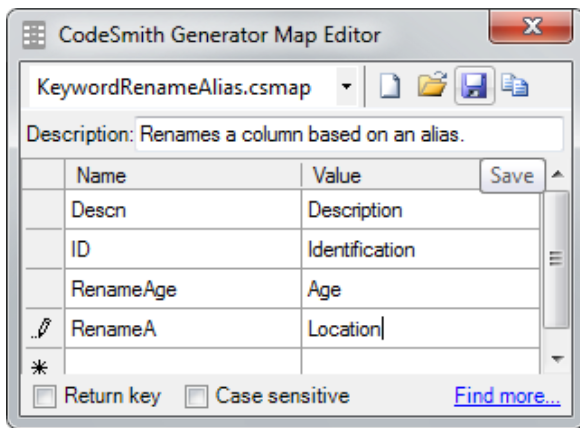
Name	Type	Size
Common	File folder	
CSharp	File folder	
Source	File folder	
VisualBasic	File folder	
Master.proj	PROJ File	5 KB
QuickStart	CodeSmith Generator Template	7 KB
Readme	Internet Shortcut	1 KB

Open this folder and double click on the **KeywordRenameAlias** mapping file.

In the future, we would like to make this a per project setting as this method renames everything generated by the templates. To create a per project setting, use a different method or create a copy of the templates for a specific project.



Next, we will quickly rename the **RenameAge** column to **Age** and the **RenameA** column to **Location**. As you can see there are already a few existing entries. We'll add the two entries and then hit the save icon. Then we will go back into our solution and regenerate.



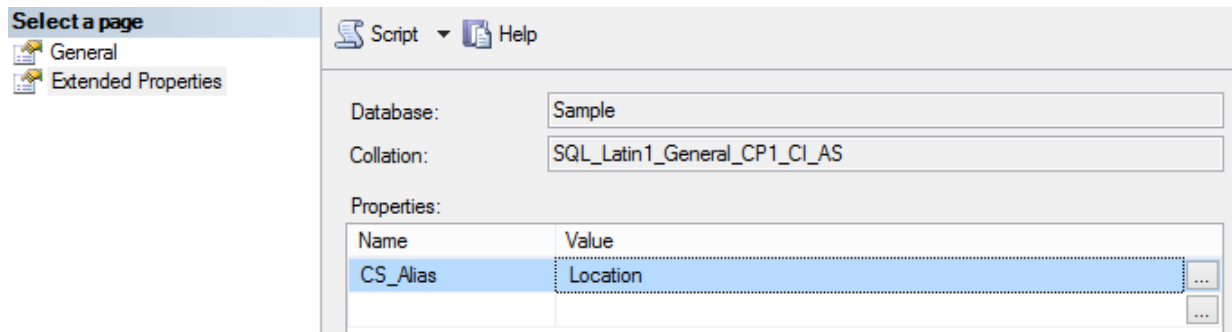
The generated code will be updated with our new names as shown below!

C#	Visual Basic
<pre> public System.Int32 RenameID { get; set; } public System.String RenameName { get; set; } public System.String Location{ get; set; } public System.String Name { get; set; } public System.Int32 Age { get; set; } </pre>	<pre> Public Property RenameID() As System.Int32 End Property Public Property RenameName() As System.String End Property Public Property Location() As System.String End Property Public Property Name() As System.String End Property Public Property Age() As System.Int32 End Property </pre>

We will now revert the KeywordRenameAlias mapping file back to it's default values for the next section.

Method 2

This method works much like Method one, but instead your settings are stored as **extended properties** inside of SQL Server. We will open SQL Management Studio and right click on the column in question and select properties. Then we will add an extended property called **CS_Alias** with the value being the desired name.



You can also add an extended property with the follow SQL. This was created by selecting the Script button shown above.

```
EXEC sys.sp_addextendedproperty @name=N'CS_Alias', @value=N'Location' ,
@level0type=N'SHEMA',@level0name=N'dbo', @level1type=N'TABLE',@level1name=N'Rename',
@level2type=N'COLUMN',@level2name=N'RenameA'
GO
```

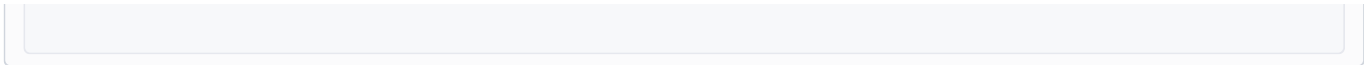
Method 3

This method is very powerful and could save you a bunch of time! However, it currently requires you to open up the source solution located in the **Source** folder, make one change and recompile the solution. Inside of the Generator.CSLA Project, open up the CodeTemplates\CSLABaseTemplate.cs class. Then navigate to the constructor and update the following line:

```
Configuration.Instance.NamingProperty.ColumnNaming = ColumnNaming.Preserve;
```

To:

```
Configuration.Instance.NamingProperty.ColumnNaming = ColumnNaming.RemoveTablePrefix;
```



When ColumnNaming is set to **RemoveTablePrefix**, sit back and be amazed at how much cleaner your properties become automatically! When **RemoveTablePrefix** is set, **Method 1** and **Method 2** will run and take preference. After **Method 1** or **Method 2** run, if the property name matches the cleaned table name and the renaming character length is greater than one character (for uniqueness). We will then strip the table name from the property name and rerun **Method 1** and **Method 2**.

Association names will not be renamed via this method.

After regenerating, your properties will look like this:

C#	Visual Basic
<pre>public System.Int32 Identification { get; set; } public System.String Name1 { get; set; } public System.String RenameA { get; set; } public System.String Name2 { get; set; } public System.Int32 Age { get; set; }</pre>	<pre>Public Property Identification() As System.Int32 End Property Public Property Name1() As System.String End Property Public Property RenameA() As System.String End Property Public Property Name2 () As System.String End Property Public Property Age() As System.Int32 End Property</pre>

You might be wondering why RenameID was changed into Identification. This is because the table prefix was removed leaving just ID. The underlying engine then runs both **Method 1** and **Method 2**. ID is renamed to Identification in **Method 1** since ID is defined in the **KeywordRenameAlias** mapping file. You might also be wondering why RenameA wasn't renamed to A. This is because the final result would be one character long, which can lead to many naming conflicts.

The following generated code shows that even though you renamed a property, the associated database column was not renamed!

C#	Visual Basic
<pre>const string commandText = "INSERT INTO [dbo]. [Rename] ([RenameID], [RenameName], [RenameA], [Name], [RenameAge]) VALUES (@p_RenameID, @p_RenameName, @p_RenameA, @p_Name,</pre>	<pre>Const commandText As String = "INSERT INTO [dbo].[Rename] ([RenameID], [RenameName], [RenameA], [Name], [RenameAge]) VALUES (@p_RenameID, @p_RenameName, @p_RenameA,</pre>

```

@p_RenameAge)";
using (SqlConnection connection = new
SqlConnection(ADOHelper.ConnectionString))
{
    connection.Open();
    using (SqlCommand command = new
SqlCommand(commandText, connection))
    {

command.Parameters.AddWithValue("@p_RenameID",
this.Identification);

command.Parameters.AddWithValue("@p_RenameName"
, this.Name1);

command.Parameters.AddWithValue("@p_RenameA",
this.RenameA);

command.Parameters.AddWithValue("@p_Name",
this.Name2);

command.Parameters.AddWithValue("@p_RenameAge",
this.Age);

        //result: The number of rows changed,
inserted, or deleted. -1 for select statements;
0 if no rows were affected, or the statement
failed.
        int result = command.ExecuteNonQuery();
        if (result == 0)
            throw new
DBConcurrencyException("The entity is out of
date on the client. Please update the entity
and try again. This could also be thrown if the
sql statement failed to execute.");

LoadProperty(_originalIdentificationProperty,
this.Identification);
    }
}

```

```

@p_Name, @p_RenameAge)"
Using connection As New
SqlConnection(ADOHelper.ConnectionString)
    connection.Open()
    Using command As New
SqlCommand(commandText, connection)

command.Parameters.AddWithValue("@p_RenameID",
Me.Identification)

command.Parameters.AddWithValue("@p_RenameName"
, Me.Name1)

command.Parameters.AddWithValue("@p_RenameA",
Me.RenameA)
    command.Parameters.AddWithValue("@p_Name",
Me.Name2)

command.Parameters.AddWithValue("@p_RenameAge",
Me.Age)

        'result: The number of rows changed,
inserted, or deleted. -1 for select statements;
0 if no rows were affected, or the statement
failed.
        Dim result As Integer =
command.ExecuteNonQuery()
        If (result = 0) Then
            throw new
DBConcurrencyException("The entity is out of
date on the client. Please update the entity
and try again. This could also be thrown if the
sql statement failed to execute.")
        End If
    End Using

LoadProperty(_originalIdentificationProperty,
Me.Identification)

End Using

```

Using partial methods

One of the best features of code generation is the reality of being able to actively generate your code while preserving custom changes. This allows you to implement custom logic while constantly making changes to your metadata and getting the latest template changes. The CSLA templates offer many **partial method overrides** to make your life easier. It is recommended that you place any partial methods that you implement in the non-generated **partial class**; please see this for more [information](#).

A list of partial methods can be found at the bottom of the every **generated** partial class. Here is an example of the generated partial methods.

C#

```
#region DataPortal partial methods

partial void OnCreating(ref bool cancel);
partial void OnCreated();
partial void OnFetching(CalendarCriteria criteria, ref bool cancel);
partial void OnFetched();
partial void OnMapping(SafeDataReader reader, ref bool cancel);
partial void OnMapped();
partial void OnUpdating(ref bool cancel);
partial void OnUpdated();
partial void OnAddNewCore(ref Calendar item, ref bool cancel);

#endregion
```

Visual Basic

```
#Region "DataPortal partial methods"

Partial Private Sub OnCreating(ByRef cancel As Boolean)
End Sub
Partial Private Sub OnCreated()
End Sub
Partial Private Sub OnFetching(ByVal criteria As CalendarCriteria, ByRef cancel As Boolean)
End Sub
```

```

Partial Private Sub OnFetched()
End Sub
Partial Private Sub OnMapping(ByVal reader As SafeDataReader, ByRef cancel As Boolean)
End Sub
Partial Private Sub OnMapped()
End Sub
Partial Private Sub OnUpdating(ByRef cancel As Boolean)
End Sub
Partial Private Sub OnUpdated()
End Sub
Partial Private Sub OnAddNewCore(ByVal item As Calendar, ByRef cancel As Boolean)
End Sub

#End Region

```

In this example we will be generating a DynamicListBase business object called CalendarList, whose purpose is to hold a list of EditableRoot Calendar business objects. The default generated code will select all calendar entries within a specific criterion. We will be using the Calendar table with the following definition:

```

CREATE TABLE [dbo].[Calendar](
    [ID] [nvarchar](50) NOT NULL,
    [CalendarName] [nvarchar](50) NOT NULL,
    [Name] [nvarchar](50) NOT NULL,
    [EventStart] [datetime] NOT NULL,
    [EventEnd] [datetime] NOT NULL,
    [Resource] [nvarchar](150) NOT NULL,
    CONSTRAINT [PK_Calendar] PRIMARY KEY CLUSTERED
(
    [ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

```

The code below is the code that is generated in the CalendarList.DataAccess partial class:

C#

```

private void DataPortal_Fetch(CalendarCriteria criteria)
{
    bool cancel = false;
    OnFetching(criteria, ref cancel);
    if (cancel) return;
}

```

```

        RaiseListChangedEvents = false;
        // Fetch Child objects.
        string commandText = String.Format("SELECT [ID], [CalendarName], [Name], [EventStart], [EventEnd],
[Resource] FROM [dbo].[Calendar] {0}", ADOHelper.BuildWhereStatement(criteria.StateBag));
        using (var connection = new SqlConnection(ADOHelper.ConnectionString))
        {
            connection.Open();
            using (var command = new SqlCommand(commandText, connection))
            {
                command.Parameters.AddRange(ADOHelper.SqlParameters(criteria.StateBag));
                using (var reader = new SafeDataReader(command.ExecuteReader()))
                {
                    if (reader.Read())
                    {
                        do
                        {
                            this.Add(Calendar.GetCalendar(reader));
                        } while (reader.Read());
                    }
                }
            }
        }
        RaiseListChangedEvents = true;
        OnFetched();
    }
}

```

Visual Basic

```

Private Shadows Sub DataPortal_Fetch(ByVal criteria As CalendarCriteria)
    Dim cancel As Boolean = False
    OnFetching(criteria, cancel)
    If (cancel) Then
        Return
    End If

    RaiseListChangedEvents = False

    ' Fetch Child objects.
    Dim commandText As String = String.Format("SELECT [ID], [CalendarName], [Name], [EventStart],
[EventEnd], [Resource] FROM [dbo].[Calendar] {0}", ADOHelper.BuildWhereStatement(criteria.StateBag))
    Using connection As New SqlConnection(ADOHelper.ConnectionString)
        connection.Open()
        Using command As New SqlCommand(commandText, connection)
            command.Parameters.AddRange(ADOHelper.SqlParameters(criteria.StateBag))
            Using reader As SafeDataReader = New SafeDataReader(command.ExecuteReader())
                If reader.Read() Then
                    Do

```

```

                Me.Add(Calendar.GetCalendar(reader))
            Loop While reader.Read()
        End If
    End Using
End Using

RaiseListChangedEvents = True

OnFetched()
End Sub

```

As you can see, this isn't optimal because we only want to select all Calendar entries from the first of the current month. *In the real world we would only want to select maybe a month or two of data...* This can easily be done by implementing the partial method for OnFetching and modifying the SQL statement. We will now go to the partial non generated class (CalendarList.cs or CalendarList.vb) and update the code.

C#

```

#region Custom Data Access

partial void OnFetching(CalendarCriteria criteria, ref bool cancel)
{
    RaiseListChangedEvents = false;
    // Fetch Child objects.
    string commandText = string.Format("SELECT [ID], [CalendarName], [Name], [EventStart], [EventEnd], [Resource] FROM [dbo].[Calendar] {0} AND [EventStart] >= CONVERT(varchar,DATEADD(MONTH,DATEDIFF(MONTH,0,GETDATE()),0),101)", ADOHelper.BuildWhereStatement(criteria.StateBag));
    using (var connection = new SqlConnection(ADOHelper.ConnectionString))
    {
        connection.Open();
        using (var command = new SqlCommand(commandText, connection))
        {
            command.Parameters.AddRange(ADOHelper.SqlParameters(criteria.StateBag));
            using (var reader = new SafeDataReader(command.ExecuteReader()))
            {
                if (reader.Read())
                {
                    do
                    {
                        this.Add(Calendar.GetCalendar(reader));
                    } while (reader.Read());
                }
            }
        }
    }
    RaiseListChangedEvents = true;
    // Cancel the existing Dal method.
}

```

```

        cancel = true;
    }

    #endregion

```

Visual Basic

```

Private Sub OnFetching(ByVal criteria As CategoryCriteria, ByRef cancel As Boolean)
    RaiseListChangedEvents = False

    ' Fetch Child objects.
    Dim commandText As String = String.Format("SELECT [ID], [CalendarName], [Name], [EventStart],
[EventEnd], [Resource] FROM [dbo].[Calendar] {0} AND [EventStart] >=
CONVERT(varchar,DATEADD(MONTH,DATEDIFF(MONTH,0,GETDATE()),0),101)",
ADOHelper.BuildWhereStatement(criteria.StateBag))
    Using connection As New SqlConnection(ADOHelper.ConnectionString)
        connection.Open()
        Using command As New SqlCommand(commandText, connection)
            command.Parameters.AddRange(ADOHelper.SqlParameters(criteria.StateBag))
            Using reader As SafeDataReader = New SafeDataReader(command.ExecuteReader())
                If reader.Read() Then
                    Do
                        Me.Add(Calendar.GetCalendar(reader))
                    Loop While reader.Read()
                End If
            End Using
        End Using
    End Using

    RaiseListChangedEvents = True
    'Cancel the existing Dal method.

    cancel = True
End Sub
#End Region

```

As you can see all we needed to do was implement the OnFetching partial method and override the sql query that was then sent to the database server. You can do anything you want in this partial method. The last thing to do is set the reference cancel variable to true if you wish to have everything after the partial method call ignored. If you set this to false, then the existing generated code will also run. In this case, we want to cancel out. A scenario where you would want to return false, is if you wanted to do some processing before accessing the database.

If you come across something that you wish had a partial method, please log it on the [CodeSmith Generator template issue tracker](#).

Extending Stored Procedures

This section will show you how to preserve your custom stored procedure logic while regenerating. The CSLA templates by default will not automatically execute the generated stored procedures but it can! To do this you need to set the **AutoExecuteStoredProcedures** property to **True**. Now, when your database related metadata changes so will your stored procedures.

This is a great feature until you need to implement custom logic into your stored procedure. By default we will generate logic that will prevent a stored procedure from being overwritten if you have AutoExecuteStoredProcedures turned to true or you try to drop the stored procedure. Here is the example that we are going to be modifying that will result in the same exact output as our [previous article](#).

```
SET NUMERIC_ROUNDABORT OFF
GO
SET ANSI_PADDING, ANSI_WARNINGS, CONCAT_NULL_YIELDS_NULL, ARITHABORT, QUOTED_IDENTIFIER, ANSI_NULLS ON
GO
IF EXISTS (SELECT * FROM tempdb..sysobjects WHERE id=OBJECT_ID('tempdb..#tmpErrors')) DROPTABLE #tmpErrors
GO
CREATE TABLE #tmpErrors (Error int)
GO
SET XACT_ABORT ON
GO
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
GO
BEGIN TRANSACTION
GO
PRINT N'Dropping CSLA_Calendar_Select'
GO
IF EXISTS(SELECT 1 FROM fn_listextendedproperty (NULL, 'SCHEMA', 'dbo', 'PROCEDURE','CSLA_Calendar_Select',
default, default) WHERE name = 'CustomProcedure' and value = '1')
BEGIN
    RAISERROR ('The procedure CSLA_Calendar_Select has an Extended Property "CustomProcedure" which means is
has been customized. Please review and remove the property if you wish to drop the procedure.',16,1)
    INSERT INTO #tmpErrors (Error) SELECT 1
END
GO

IF OBJECT_ID(N'CSLA_Calendar_Select') IS NOT NULL
    DROP PROCEDURE CSLA_Calendar_Select

GO
IF @@ERROR!=0 AND @@TRANCOUNT>0 ROLLBACK TRANSACTION
GO

IF @@TRANCOUNT=0 BEGIN INSERT INTO #tmpErrors (Error) SELECT 1 BEGIN TRANSACTION END
GO

IF EXISTS (SELECT * FROM #tmpErrors) ROLLBACK TRANSACTION
GO
```

```

IF @@TRANCOUNT>0 BEGIN
PRINT 'The stored procedure drop has succeeded'
COMMIT TRANSACTION
END
ELSE PRINT 'The stored procedure drop has failed'
GO

DROP TABLE #tmpErrors
GO

--region [dbo].[CSLA_Calendar_Select]

-----
-- Procedure Name: [dbo].[CSLA_Calendar_Select]
-----

SET NUMERIC_ROUNDABORT OFF
GO
SET ANSI_PADDING, ANSI_WARNINGS, CONCAT_NULL_YIELDS_NULL, ARITHABORT, QUOTED_IDENTIFIER, ANSI_NULLS ON
GO
IF EXISTS (SELECT * FROM tempdb..sysobjects WHERE id=OBJECT_ID('tempdb..#tmpErrors')) DROPTABLE #tmpErrors
GO
CREATE TABLE #tmpErrors (Error int)
GO
SET XACT_ABORT ON
GO
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
GO
BEGIN TRANSACTION
GO

PRINT N'Creating [dbo].[CSLA_Calendar_Select]'
GO

IF EXISTS(SELECT 1 FROM fn_listextendedproperty (NULL, 'SCHEMA', 'dbo', 'PROCEDURE','CSLA_Calendar_Select',
default, default) WHERE name = 'CustomProcedure' and value = '1')
    BEGIN
        RAISERROR ('The procedure [dbo].[CSLA_Calendar_Select] has an Extended Property "CustomProcedure"
which means is has been customized. Please review and remove the property if you wish to create the stored
procedure.',16,1)
        INSERT INTO #tmpErrors (Error) SELECT 1
    END
GO

CREATE PROCEDURE [dbo].[CSLA_Calendar_Select]
    @p_ID nvarchar(50) = NULL,
    @p_CalendarName nvarchar(50) = NULL,
    @p_Name nvarchar(50) = NULL,
    @p_EventStart datetime = NULL,
    @p_EventEnd datetime = NULL,
    @p_Resource nvarchar(150) = NULL
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

```

```

SELECT
    [ID],
    [CalendarName],
    [Name],
    [EventStart],
    [EventEnd],
    [Resource]
FROM
    [dbo].[Calendar]
WHERE
    ([ID] = @p_ID OR @p_ID IS NULL)
    AND ([CalendarName] = @p_CalendarName OR @p_CalendarName IS NULL)
    AND ([Name] = @p_Name OR @p_Name IS NULL)
    AND ([EventStart] = @p_EventStart OR @p_EventStart IS NULL)
    AND ([EventEnd] = @p_EventEnd OR @p_EventEnd IS NULL)
    AND ([Resource] = @p_Resource OR @p_Resource IS NULL)

GO
IF @@ERROR!=0 AND @@TRANCOUNT>0 ROLLBACK TRANSACTION
GO

IF @@TRANCOUNT=0 BEGIN INSERT INTO #tmpErrors (Error) SELECT 1 BEGIN TRANSACTION END
GO

IF EXISTS (SELECT * FROM #tmpErrors) ROLLBACK TRANSACTION
GO
IF @@TRANCOUNT>0 BEGIN
PRINT 'Stored procedure creation succeeded.'
COMMIT TRANSACTION
END
ELSE PRINT 'Stored procedure creation failed.'
GO
DROP TABLE #tmpErrors
GO

--endregion

```

Let's open up SQL Management Studio and run the script above. Once you run the query above, you can see that the table was dropped and recreated.

```

Dropping CSLA_Calendar_Select
The stored procedure drop has succeeded
Creating [dbo].[CSLA_Calendar_Select]
Stored procedure creation succeeded.

```

Now that you can see that the stored procedure is being overwritten, let's modify the stored procedure to return the same results as our previous example.

```

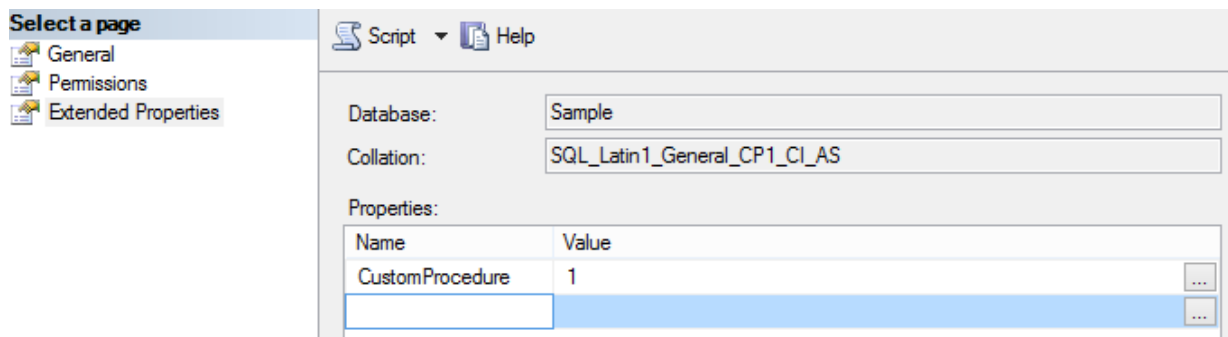
ALTER PROCEDURE [dbo].[CSLA_Calendar_Select]
    @p_ID nvarchar(50) = NULL,
    @p_CalendarName nvarchar(50) = NULL,
    @p_Name nvarchar(50) = NULL,
    @p_EventStart datetime = NULL,
    @p_EventEnd datetime = NULL,
    @p_Resource nvarchar(150) = NULL
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

SELECT
    [ID],
    [CalendarName],
    [Name],
    [EventStart],
    [EventEnd],
    [Resource]
FROM
    [dbo].[Calendar]
WHERE
    ([ID] = @p_ID OR @p_ID IS NULL)
    AND ([CalendarName] = @p_CalendarName OR @p_CalendarName IS NULL)
    AND ([Name] = @p_Name OR @p_Name IS NULL)
    AND ([EventStart] = @p_EventStart OR @p_EventStart IS NULL)
    AND ([EventEnd] = @p_EventEnd OR @p_EventEnd IS NULL)
    AND ([Resource] = @p_Resource OR @p_Resource IS NULL)
    AND [EventStart] >= CONVERT(varchar,DATEADD(MONTH,DATEDIFF(MONTH,0,GETDATE()),0),101)

```

Since you modified the stored procedure, you want to mark it as a custom Stored Procedure. If you choose not to do this, then you will need to modify this stored procedure any time you wish to regenerate or drop and recreate this stored procedure. This can be done by right clicking on the stored procedure, selecting properties and then selecting the Extended Properties node. You will want to create a new extended property with a name of **CustomProcedure** and a value of **1**.



The following SQL Script will also create the extended property.

```
EXEC sys.sp_addextendedproperty @name=N'CustomProcedure', @value=N'1' ,
@level0type=N'SHEMA',@level0name=N'dbo', @level1type=N'PROCEDURE',@level1name=N'CSLA_Calendar_Select'
GO
```

Now when you execute the SQL script to drop and create the stored procedure, it will error out and preserve your changes!

```
Dropping CSLA_Calendar_Select
Msg 50000, Level 16, State 1, Line 3
The procedure CSLA_Calendar_Select has an Extended Property "CustomProcedure" which means is has been
customized. Please review and remove the property if you wish to drop the procedure.

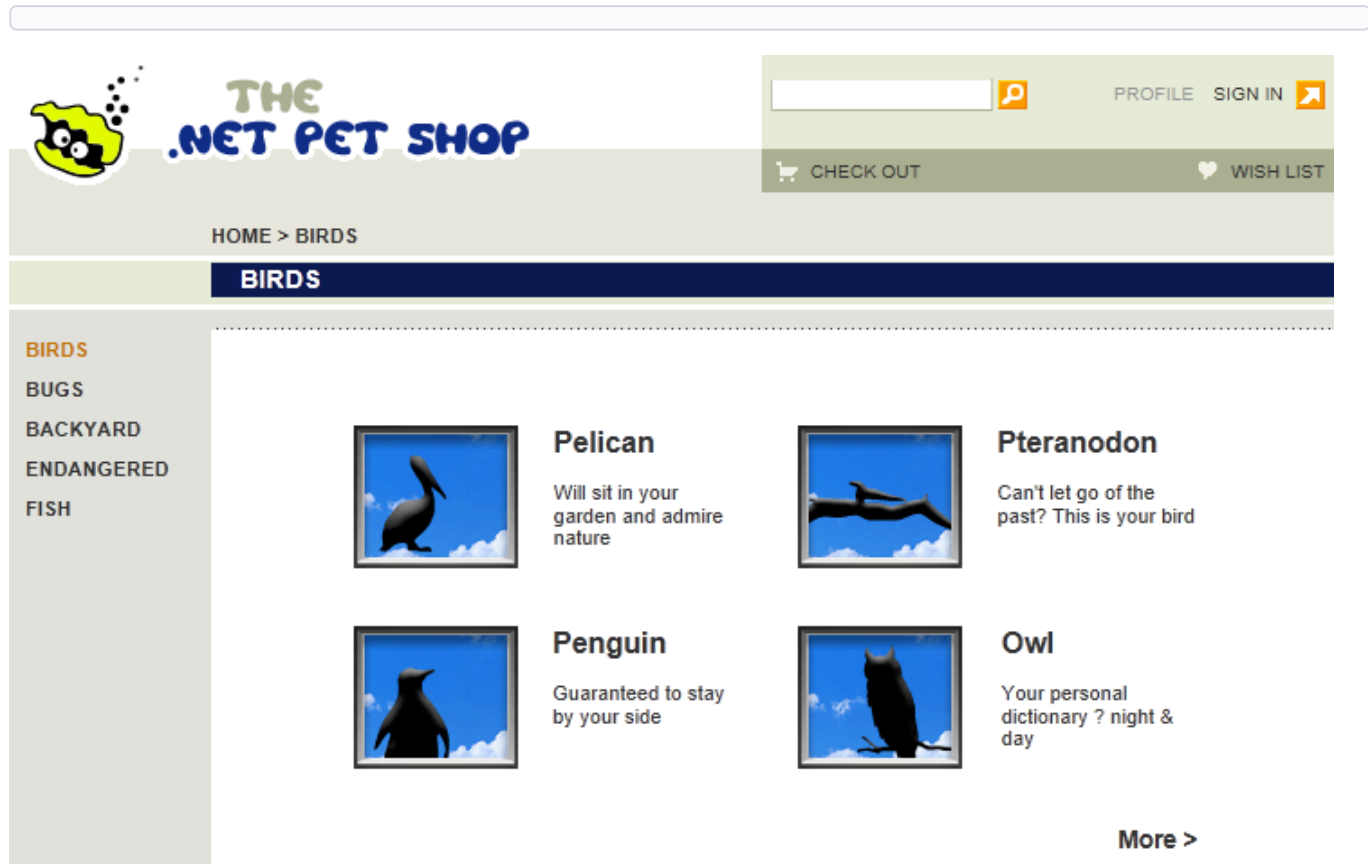
(1 row(s) affected)
The stored procedure drop has failed
Creating [dbo].[CSLA_Calendar_Select]
Msg 50000, Level 16, State 1, Line 4
The procedure [dbo].[CSLA_Calendar_Select] has an Extended Property "CustomProcedure" which means is has
been customized. Please review and remove the property if you wish to create the stored procedure.

(1 row(s) affected)
Msg 2714, Level 16, State 3, Procedure CSLA_Calendar_Select, Line 13
There is already an object named 'CSLA_Calendar_Select' in the database.

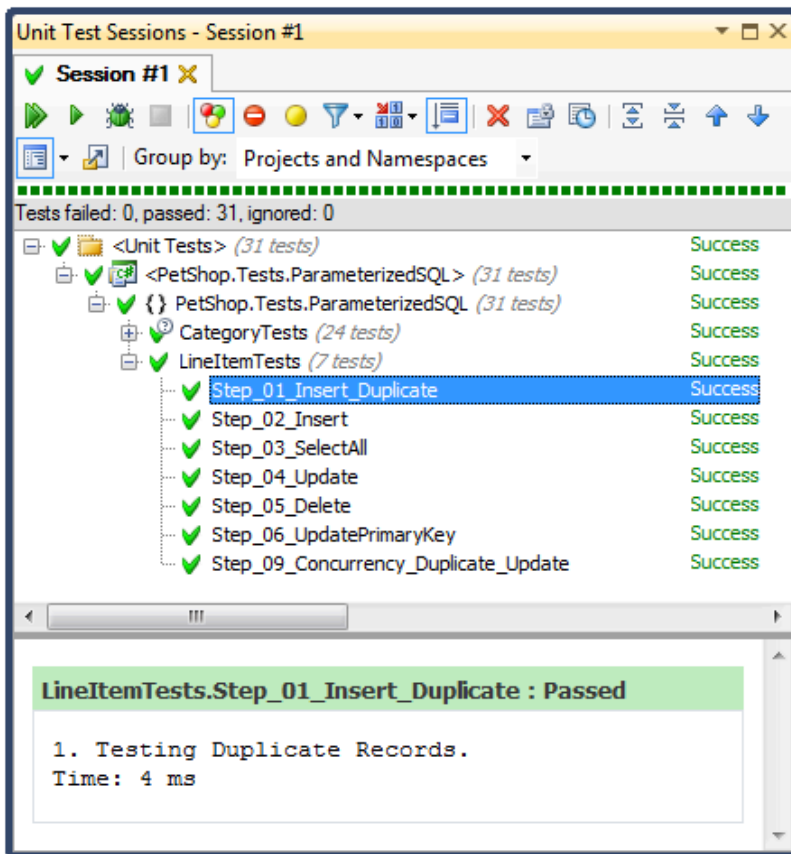
(1 row(s) affected)
Stored procedure creation failed.
```

Exploring the sample applications

During the development of the CSLA templates, we thought what better way to test the templates than to write a sample application. So we wrote a working PetShop Sample application in C# and Visual Basic that can be downloaded [here](#).



We took the PetShop sample applications a step further by writing unit tests against the generated code. You can find unit tests for all the business object types and Data Access Layer implementations. This is a great starting point for learning more about the generated code.



These **NUnit tests** will give you a good understanding of how different operations are performed in CSLA. Feel free to [contact support](#) or our [forums](#) with any unit tests you think we don't have. If you find a bug in the CSLA templates, its a huge help to the community and us if you submit a failing unit test for the PetShop sample application that can reproduce your bug. This ensures we have more test ability and no regressions in future versions. However, any unit test will be appreciated along with the respected sample schema.

When contributing new unit tests, please try to follow our common testing pattern for consistency that outputs a description and how long it took to run the test. Below is an example of one of our unit tests.

```
public void Step_01_Insert_Duplicate()
{
    Console.WriteLine("1. Testing Duplicate Records.");
    Stopwatch watch = Stopwatch.StartNew();
    //Insert should fail as there should be a duplicate key.
    Category category = Category.NewCategory();
    category.CategoryId = TestCategoryId;
    category.Name = TestUtility.Instance.RandomString(80, false);
    category.Description = TestUtility.Instance.RandomString(255, false);
    try
    {
        Assert.IsTrue(category.IsValid, category.BrokenRulesCollection.ToString());
        category = category.Save();
        // Fail as a duplicate record was entered.
    }
}
```

```
        Assert.Fail("Fail as a duplicate record was entered and an exception was not thrown.");
    }
    catch (Exception)
    {
        Assert.IsTrue(true);
    }
    Console.WriteLine("Time: {0} ms", watch.ElapsedMilliseconds);
}
```

Using Extended Properties

One can use [Extended Properties](#) to extend the generated code. These extended properties can do anything from [business object types](#), [naming conventions](#), descriptions and much more. Below is a list of all of the extended property names and valid values that the CSLA templates use.

Please see the following [document](#) for a full list of all of the extended properties used to specify the business object type of a view or stored procedure.

Extended Property Name	Data Type	Description
CS_Description	String	Controls the description of a property or class.
CS_Alias	String	Controls the property or class name .
CS_FriendlyName	String	Controls the friendly name that is generated for the Business Objects PropertyInfo's.
CS_Default	String	Controls the default value of a generated property.
CS_IsGeneric	Boolean	If true, then an entity will be generated with generic brackets.
CS_IsIdentity	Boolean	Marks a property as an identity column.
CS_IsComputed	Boolean	Marks a property as a computed column.
CS_ManyToMany	Boolean	Treats a relationship as a many to many if the table has at-least two FK columns.
CS_IsScalarFunction	Boolean	If set, a Command Entity will be treated as a function.
CS_IsTableValuedFunction	Boolean	If set, a Command Entity will be treated as a function.
CS_IsInlineTableValuedFunction	Boolean	If set, a Command Entity will be treated as a function.
CS_IsMultiStatementTableValuedFunction	Boolean	If set, a Command Entity will be treated as a function.

Using the Templates

The following section will cover how to use the various templates that ship with CSLA.

Please see the [Getting Started section](#) for steps on how to configure and use the templates within seconds.

Manual Generation

Use the following steps to get started using the CSLA templates if you are not planning on using the QuickStart template.

This is useful when you are wanting to add CSLA to an existing or new project.

1. Create a new Class Library project in Visual Studio.
2. Add a new [CodeSmith Generator Project file](#) to the Visual Studio project.
3. Add a new Output to the project file for the [Entities.cst](#) template.
 1. In the Visual Studio Solution Explorer, right click the [Generator Project File you added to the project in step 2](#).
 2. Click on the [Add Output](#) menu item.
 3. [Choose the Entities.cst template](#) in the Add Output dialog and click the OK button.
4. Required only If you are using the Object Factory Data Access Implementation.
 1. Follow steps 1-2.
 2. Add a new Output to the project file for the DataAccessLayer\DataAccessLayer.cst template.
5. Finally, Generate the Outputs
 1. In the Visual Studio Solution Explorer, right click the [Generator Project File you added to the project in step 2](#).
 2. Click on the Generate Outputs menu item.

Entities Template

The entities template can generate multiple CSLA business objects at once. The business objects are generated from a Database. Please see the following [tutorial](#) for more information on the generated Business Objects.

The Entities.cst template is located within the CSLA\CSharp\BusinessLayer or CSLA\Visual Basic\BusinessLayer folder.

You can control the names for classes and properties through various methods described [here](#).

Template Properties

The **CleanExpressions**, **IgnoreExpressions** and **IncludeExpressions** properties use a list of [Regular Expressions](#) that match the full name of a Table, View, or Command.

Here is a common Regex pattern that will match a table by name (E.G., ^owner.TableName\$). For Example, the following regex (^**dbo.Account**\$) will match the Account table.

We recommend using [Expresso](#) to build your regular expression lists.

Property	Data Type	Description
1. DataSource		
CleanExpressions	Regular Expression	List of Regular Expressions to use when cleaning property and class names.
IgnoreExpressions	Regular Expression	List of Regular Expressions that defines the schema objects to ignore.
IncludeExpressions	Regular Expression	List of Regular Expressions that defines the schema objects to include. This takes precedence over IgnoreExpressions.
IncludeAssociations	Boolean	Controls whether Entity associations are included in the generated output.
IncludeFunctions	Boolean	Controls whether Entities are generated from Stored Procedures.
IncludeViews	Boolean	Controls whether Entities are generated from Views.
SourceDatabase	DatabaseSchema	The Database that should be generated against.
2. Solution		
FrameworkVersion	Enumeration	The .NET Framework version you wish to target. If the value is set to: • v35, then a Visual Studio solution that targets CSLA 3.8.x will be created. • v40, then a Visual Studio solution that targets CSLA 4.3.x will be created.
Location	String	The root directory of the generated solution.
SolutionName	String	The name of the Visual Studio solution that should be created.
3. Business Project		

BusinessProjectName	String	The name of the business project directory and project.																																		
IncludeSilverlightSupport	Boolean	Adds Silverlight features to the business project.																																		
UseMemberVariables	Boolean	Uses private property backing fields for properties.																																		
4. Data Project																																				
AutoExecuteStoredProcedures	Boolean	Automatically executes the generated stored procedures against the selected SourceDatabase.																																		
DataAccessImplementation	Enumeration	Changes how the Business Data Access Methods and Data Access Layer are implemented. Please use the following chart for a break down of usages. <table><thead><tr><th>Value</th><th>Location of Data Access</th><th>Data Access Implementation</th><th>Editable</th></tr></thead><tbody><tr><td>None</td><td>Business Project</td><td>None, you can implement your own.</td><td>Yes</td></tr><tr><td>ParameterizedSQL</td><td>Business Project</td><td>Parameterized SQL</td><td>No</td></tr><tr><td>StoredProcedures</td><td>Business Project</td><td>Stored Procedures</td><td>No</td></tr><tr><td>LinqToSQL</td><td>Business Project</td><td>Linq To SQL (Only available for C#)</td><td>No</td></tr><tr><td>ObjectFactoryNone</td><td>Data Project</td><td>None, you can implement your own.</td><td>Yes</td></tr><tr><td>ObjectFactoryParameterizedSQL</td><td>Data Project</td><td>Parameterized SQL</td><td>No</td></tr><tr><td>ObjectFactoryStoredProcedures</td><td>Data Project</td><td>Stored Procedures</td><td>No</td></tr></tbody></table> Please see this for more information on data access implementations.			Value	Location of Data Access	Data Access Implementation	Editable	None	Business Project	None, you can implement your own.	Yes	ParameterizedSQL	Business Project	Parameterized SQL	No	StoredProcedures	Business Project	Stored Procedures	No	LinqToSQL	Business Project	Linq To SQL (Only available for C#)	No	ObjectFactoryNone	Data Project	None, you can implement your own.	Yes	ObjectFactoryParameterizedSQL	Data Project	Parameterized SQL	No	ObjectFactoryStoredProcedures	Data Project	Stored Procedures	No
Value	Location of Data Access	Data Access Implementation	Editable																																	
None	Business Project	None, you can implement your own.	Yes																																	
ParameterizedSQL	Business Project	Parameterized SQL	No																																	
StoredProcedures	Business Project	Stored Procedures	No																																	
LinqToSQL	Business Project	Linq To SQL (Only available for C#)	No																																	
ObjectFactoryNone	Data Project	None, you can implement your own.	Yes																																	
ObjectFactoryParameterizedSQL	Data Project	Parameterized SQL	No																																	
ObjectFactoryStoredProcedures	Data Project	Stored Procedures	No																																	
DataProjectName	String	The name of the data project directory and project.																																		
IsolationLevel	Enumeration	Isolation level to use in the generated stored procedures.																																		
ParameterPrefix	String	Prefix to use for all generated stored procedure parameters.																																		
ProcedurePrefix	String	Prefix to use for all generated stored procedures.																																		
UseLazyloading	Boolean	Properties will load data only when it is requested.																																		
5. Interface Project																																				
InterfaceProjectName	Boolean	The name of the interface project directory and project.																																		
6a. Entities (Please see this for more information)																																				
CommandObject	TableSchemaCollection	A collection of tables that should be generated as Command Objects.																																		
DynamicRoot	TableSchemaCollection	A collection of tables that should be generated as Dynamic Root Objects.																																		
EditableChild	TableSchemaCollection	A collection of tables that should be generated as Editable Child Objects.																																		

EditableRoot	TableSchemaCollection	A collection of tables that should be generated as Editable Root Objects.
ReadOnlyChild	TableSchemaCollection	A collection of tables that should be generated as Read Only Child Objects.
ReadOnlyRoot	TableSchemaCollection	A collection of tables that should be generated as Read Only Root Objects.
SwitchableObject	TableSchemaCollection	A collection of tables that should be generated as Switchable Objects.
6b. List Entities (Please see this for more information)		
DynamicListBase	TableSchemaCollection	A collection of tables that should be generated as Dynamic List Base Objects.
DynamicRootList	TableSchemaCollection	A collection of tables that should be generated as Dynamic Root List Objects.
EditableChildList	TableSchemaCollection	A collection of tables that should be generated as Editable Child List Objects.
EditableRootList	TableSchemaCollection	A collection of tables that should be generated as Editable Root List Objects.
NameValueList	TableSchemaCollection	A collection of tables that should be generated as Name Value List Objects.
ReadOnlyChildList	TableSchemaCollection	A collection of tables that should be generated as Read Only Child List Objects.
ReadOnlyList	TableSchemaCollection	A collection of tables that should be generated as Read Only List Objects.
7. LinqToSQL Data Access Layer		
LinqToSQLContextNamespace	String	The namespace of the generated PLINQO DataContext. Please see this for more information.
LinqToSQLDataContextName	String	The name of the generated PLINQO DataContext. Please see this for more information.

Quick Start Template

The QuickStart template is used to create a basic project using CSLA. The template will create a Visual Studio solution with up to three included projects: a Business Project, Data Project, and an Interface Project.

Please see the [Getting Started section](#) for steps on how to use the QuickStart template.

Template Properties

Property	Data Type	Description											
1. Database													
SourceDatabase	DatabaseSchema	The Database that should be generated against.											
2. Solution													
FrameworkVersion	Enumeration	The .NET Framework version you wish to target. If the value is set to: • v35, then a Visual Studio solution that targets CSLA 3.8.x will be created. • v40, then a Visual Studio solution that targets CSLA 4.3.x will be created.											
Language	Enumeration	The language that thee project will be generated in.											
LaunchVisualStudio	Boolean	Launches Visual Studio when the project finishes generating.											
Location	String	The root directory of the generated solution.											
SolutionName	String	The name of the Visual Studio Solution that should be created.											
3. Business Project													
BusinessProjectName	String	The name of the business project directory and project.											
IncludeSilverlightSupport	Boolean	Adds Silverlight features to the business project.											
UseMemberVariables	Boolean	Uses private property backing fields for properties.											
4. Data Project													
AutoExecuteStoredProcedures	Boolean	Automatically executes the generated stored procedures against the selected SourceDatabase.											
DataAccessImplementation	Enumeration	Changes how the Business Data Access Methods and Data Access Layer are implemented. Please use the following chart for a break down of usages. <table><tr><th>Value</th><th>Location of Data Access</th><th>Data Access Implementation</th><th>Editable</th></tr><tr><td>None</td><td>Business Project</td><td>None, you can implement your own.</td><td>Yes</td></tr></table>				Value	Location of Data Access	Data Access Implementation	Editable	None	Business Project	None, you can implement your own.	Yes
Value	Location of Data Access	Data Access Implementation	Editable										
None	Business Project	None, you can implement your own.	Yes										

		ParameterizedSQL	Business Project	Parameterized SQL	No
		StoredProcedures	Business Project	Stored Procedures	No
		LinqToSQL	Business Project	Linq To SQL (Only available for C#)	No
		ObjectFactoryNone	Data Project	None, you can implement your own.	Yes
		ObjectFactoryParameterizedSQL	Data Project	Parameterized SQL	No
		ObjectFactoryStoredProcedures	Data Project	Stored Procedures	No
		Please see this for more information on data access implementations.			
DataProjectName	String	The name of the data project directory and project.			
IsolationLevel	Enumeration	Isolation level to use in the generated stored procedures.			
ParameterPrefix	String	Prefix to use for all generated stored procedure parameters.			
ProcedurePrefix	String	Prefix to use for all generated stored procedures.			
UseLazyloading	Boolean	Properties will load data only when it is requested.			
5. Interface Project					
IncludeInterfaceProject	Boolean	Adds a website to the generated solution.			
InterfaceProjectName	Boolean	The name of the interface project directory and project.			
7. LinqToSQL Data Access Layer					
LinqToSQLContextNamespace	String	The namespace of the generated PLINQO DataContext. Please see this for more information.			
LinqToSQLDataContextName	String	The name of the generated PLINQO DataContext. Please see this for more information.			