

پیش از ادامه‌ی نوشتار بهتر است توضیحاتی درباره‌ی قالب‌های T4 داده شود. این قالب‌های مصنوعی حاوی کدهایی که است که هدف آن صرفه‌جویی در نوشتن کد توسط برنامه نویسی است. مثلاً در MVC شما یکبار قالبی برای صفحه Index خود تهیه می‌کنید که برای نمونه بجای ساخت جدول ساده، از گرید Kendo استفاده کند و همچنین دارای دکمه ویرایش و جزئیات باشد. از این پس هر بار که نیاز به ساخت یک نمای نوع لیست برای یک ActionResult داشته باشید فرم ساز MVC از قالب شما استفاده خواهد کرد. روشن است که خود Visual Studio نیز از T4 در ساخت بسیاری از فرم‌ها و کلاس‌ها بهره می‌برد. خبر خوب این‌که برای ساخت کلاس‌های هر موجودیت در Entity Framework نیز از قالب‌های T4 استفاده می‌شود و این‌که این قالب‌ها در دسترس توسعه‌دهندگان برای ویرایش یا افزودن است.

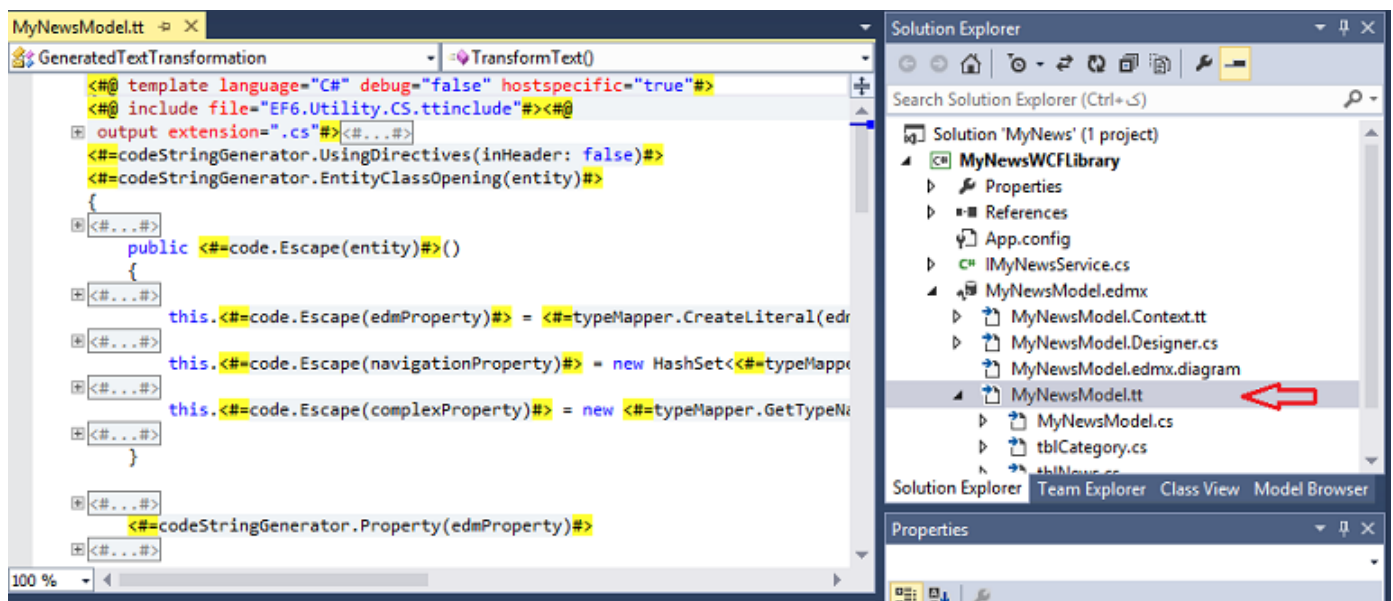
افزونه‌ی [Tangible](#) را دریافت کنید و سپس نصب کنید. این افزونه ظاهر نامفهوم قالب‌های T4 را ساده و روشن می‌کند. ما نیاز داریم که خود Visual Studio زحمت این سه کار را بکشد:

1- بالای هر کلاس موجودیت عبارت using System.Runtime.Serialization; را بنویسید.

2- صفت [DataContract] را پیش از تعریف کلاس بیفزاید.

3- صفت [DataMember] را پیش از تعریف هر ویژگی بیفزاید.

همانند شکل زیر روی فایل MyNewsModel.tt دوکلیک کنید تا محتوای آن در سمت چپ نشان داده شود. این محتوا باید ظاهری همانند شکل پیدا کرده باشد:



کد زیر را در محتوای فایل جست‌وجو کنید:

```
public string Property(EdmProperty edmProperty)
{
    return string.Format(
        CultureInfo.InvariantCulture,
        "{0} {1} {2} {{ {3}get; {4}set; }}",
        Accessibility.ForProperty(edmProperty),
        _typeMapper.GetTypeName(edmProperty.TypeUsage),
        _code.Escape(edmProperty),
        _code.SpaceAfter(Accessibility.ForGetter(edmProperty)),
        _code.SpaceAfter(Accessibility.ForSetter(edmProperty)));
}
```

```
}
```

متن آن‌را به این صورت تغییر دهید:

```
public string Property(EdmProperty edmProperty)
{
    return string.Format(
        CultureInfo.InvariantCulture,
        "[DataMember]" + Environment.NewLine +
        "{0} {1} {2} {{ {3}get; {4}set; }}",
        Accessibility.ForProperty(edmProperty),
        _typeMapper.GetTypeName(edmProperty.TypeUsage),
        _code.Escape(edmProperty),
        _code.SpaceAfter(Accessibility.ForGetter(edmProperty)),
        _code.SpaceAfter(Accessibility.ForSetter(edmProperty)));
}
```

بار دیگر به دنبال این کد بگردید:

```
public string EntityClassOpening(EntityType entity)
{
    return string.Format(
        CultureInfo.InvariantCulture,
        "{0} {1}partial class {2}{3}",
        Accessibility.ForType(entity),
        _code.SpaceAfter(_code.AbstractOption(entity)),
        _code.Escape(entity),
        _code.StringBefore(" : ", _typeMapper.GetTypeName(entity.BaseType)));
}
```

این کد را نیز به این صورت تغییر دهید:

```
public string EntityClassOpening(EntityType entity)
{
    return string.Format(
        CultureInfo.InvariantCulture,
        "[DataContract]" + Environment.NewLine +
        "{0} {1}partial class {2}{3}",
        Accessibility.ForType(entity),
        _code.SpaceAfter(_code.AbstractOption(entity)),
        _code.Escape(entity),
        _code.StringBefore(" : ", _typeMapper.GetTypeName(entity.BaseType)));
}
```

برای واپسین تغییر به دنبال کد زیر بگردید:

```
public string UsingDirectives(bool inHeader, bool includeCollections = true)
{
    return inHeader == string.IsNullOrEmpty(_code.VsNamespaceSuggestion())
        ? string.Format(
            CultureInfo.InvariantCulture,
            "{0}using System;{1}" +
            "{2}",
            inHeader ? Environment.NewLine : "",
            includeCollections ? (Environment.NewLine + "using System.Collections.Generic;") : "",
            inHeader ? "" : Environment.NewLine)
        : "";
}
```

سپس کد زیر را جاگزین آن کنید:

```
public string UsingDirectives(bool inHeader, bool includeCollections = true)
{
    return inHeader == string.IsNullOrEmpty(_code.VsNamespaceSuggestion())
        ? string.Format(
            CultureInfo.InvariantCulture,
            "using System.Runtime.Serialization;" + Environment.NewLine +
```

```
{0}using System;{1}" +
"{2}",
inHeader ? Environment.NewLine : "",
includeCollections ? (Environment.NewLine + "using System.Collections.Generic;") : "",
inHeader ? "" : Environment.NewLine)
: "";
}
```

فایل MyNewsModel.tt را ذخیره کنید و از آن خارج شوید. بار دیگر هر کدام از کلاس‌های tblNews و tblCategory را باز کنید. خواهید دید که به صورت خودکار تغییرات مد نظر ما به آن افزوده شده است. از این پس بدون هیچ دلوپسی بابت حذف صفات، می‌توانید هرچند بار که خواستید مدل خود را به‌هنگام کنید. در بخش پسین دوباره به WCF بازخواهیم گشت و به تعریف روال‌های مورد نیاز خواهیم پرداخت.

## نظرات خوانندگان

نویسنده: محسن خان  
تاریخ: ۱۴:۸ ۱۳۹۲/۱۰/۲۶

با تشکر از شما. روش دیگری برای حل مساله استفاده از AOP است:

[استفاده از IL Code Weaving برای تولید ویژگی‌های تکراری مورد نیاز در WCF](#)

نویسنده: حمید  
تاریخ: ۴:۱ ۱۳۹۲/۱۰/۲۷

هرچند که به نکته خوبی، اشاره کردین اما این کار از اساس غلط است چون شما دارید کلاسهای لایه داده خود را expose می‌کنید. سرویس‌ها بادی DTOها را به بیرون EXPOSE کنند و تبدیل کلاسهای لایه BUSINESS به dtoها از طریق ابزاری مثل AUTOMAPPER انجام می‌شود. متشکرم

نویسنده: محسن خان  
تاریخ: ۹:۲۸ ۱۳۹۲/۱۰/۲۷

بایدی وجود ندارد در این حالت و بهتر است که اینگونه باشد یا حتی مخلوطی از این دو در عمل:

[Pros and Cons of Data Transfer Objects](#)

In large projects with so many entities, DTOs add a remarkable level of (extra) complexity and work to do. In short, a pure, 100% DTO solution is often just a 100 percent painful solution