

یک سری ویدیوی رایگان ایجاد و همچنین برنامه نویسی workflow های SharePoint 2007 در سایت آقای [shelton](#) موجود است که متأسفانه این سایت با اینترنت داتک باز نمی‌شود (یا ازون طرف یا از این طرف فیلتر شده! یا مشکل DNS اینترنت داتک است، نمی‌دونم).

برای راحتی دریافت آن‌ها یک mirror از این ویدیوها ایجاد شد:

Enabling incoming/outgoing email for MOSS 2007,

[download](#)

Extracting Document Details with SharePoint Workflow's,

[download](#)

Extracting email addresses & Sending emails with SharePoint Workflow,

[download](#)

Programmatically creating user tasks with SharePoint Workflow,

[download](#)

Programmatically escalating Overdue Tasks ,

[download](#)

Building Basic Approval Workflow,

[download](#)

Building a Multilevel Approval Workflow,

[download](#)

Using Active Directory Searching/Lookup in a SharePoint/MOSS 2007 Workflow,

[download](#)

همچنین یک سری ویدیوی آموزشی رایگان دیگر نیز در مورد workflow foundation در [این آدرس](#) قابل دریافت است.

### [\(Intro to Windows Workflow Foundation \(Part 1 of 7\): Workflow in Windows Applications \(Level 100](#)

This webcast is a code-focused introduction to developing workflow-enabled Microsoft Windows platform applications. We cover the basics of developing, designing, and debugging workflow solutions. Gain the knowledge and insight you need to be confident choosing workflow for everyday applications.

[Intro to Windows Workflow Foundation \(Part 2 of 7\): Simple Human Workflow Using E-mail \(Level 200\)](#) Have you thought about how you might apply the workflow concept to e-mail? In this webcast New Zealand based regional director, Chris Auld, leads attendees through a simple worked example of the use of SMTP e-mail as part of a workflow solution. Chris demonstrates how to create custom activities to query Active Directory to retrieve user data, send e-mail, and wait for e-mail responses to continue the workflow process. This code-intensive session gives users taking their first steps with workflow a good grounding in some of the key extensibility concepts.

### [Intro to Windows Workflow Foundation \(Part 3 of 7\): Hosting and Communications Options in Workflow Scenarios \(\(Level 300](#)

The session looks at options for hosting workflow applications. We cover managing events, instance tracking, and persistence, and provide a close look at the simple communications mechanisms that are available for you to use in your workflow applications.

[Intro to Windows Workflow Foundation \(Part 4 of 7\): Workflow, Messaging, and Services: Developing Distributed Applications with Workflows \(Level 300\)](#) Web service technologies have typically taken a "do-it-yourself" approach to maintaining the interoperation state of services. Using workflow, developers now have tools that allow them to describe the long-running state of their services and delegate much of the state management to the underlying platform. Managing this state correctly becomes even more challenging in applications that coordinate work across multiple services either within an organization or at an Internet scale. This session looks at how developers who use either Microsoft ASMX or Microsoft's framework for building service-oriented applications, code-named "Indigo", can create workflow-oriented applications that are both faster to write and more manageable and flexible once deployed.

### [\(Intro to Windows Workflow Foundation \(Part 5 of 7\): Developing Event Driven State Machine Workflows \(Level 300](#)

State machines used to be something that you had to first draw on paper and then implement in code. This session shows how to use technologies to create event-driven workflows and how to apply this to a typical programming problem. We introduce the concept of a flexible process and show how this can help with modeling real-world processes using state and sequential workflow. Plenty of coding is included to illustrate how you can seamlessly merge state machine design and your code.

[Intro to Windows Workflow Foundation \(Part 6 of 7\): Extending Workflow Capabilities with Custom Activities \(Level 300\)](#) It is helpful to think of activities as controls within a workflow, similar to controls used with Microsoft ASP.NET Pages or Microsoft Windows Forms. You can use activities to encapsulate execution logic, communicate with the host and decompose a workflow into reusable components. This session examines the simple

process of creating custom activities. If you want to expose activities to other developers designing workflows, you are likely to find this session valuable

[Intro to Windows Workflow Foundation \(Part 7 of 7\): Developing Rules Driven Workflows \(Level 300\)](#) Rules can be a powerful business tool when combined with workflow. In this session, learn how to develop more advanced activities that support the modeling of rich business behavior such as human workflow. Understand when to use rules for business logic, and see how rule policies allow for the description of sophisticated behavior in an integrated and flexible way. This session gives you an interesting insight into the power of using workflow at the core of a line of business application

## نظرات خوانندگان

نویسنده: Mohsen  
تاریخ: ۱۳:۵۷:۴۴ ۱۳۸۹/۰۵/۰۴

با سلام  
ببخشید از لینک های بالا نمی توانم Video ها را Download نمایم  
لطفا راهنمایی نمایید  
با تشکر فراوان  
nezafat.mohsen@yahoo.com

نویسنده: وحید نصیری  
تاریخ: ۱۸:۱۰:۳۴ ۱۳۸۹/۰۵/۰۴

تست کردم مشکلی نبود و فایل را پس از وارد کردن یک آدرس ایمیل دلخواه و نام شرکت دلخواه در صفحه بعد می شود دانلود کرد.  
البته فقط با IE کار می کند.  
با سایر مرورگرها تست نکنید.

## چرا از WorkFlow در پروژه‌های نرم افزاری استفاده می‌شود ؟

زمانیکه در حال انجام یک پروژه نرم افزاری هستید که این پروژه دارای پیچیدگی خاصی از لحاظ فرآیند و قوانین کاری می‌باشد بهترین راه حل Workflow Engine یا BPMS Engine می‌باشد. البته شایان ذکر می‌باشد که میان این دو Engine تفاوت‌های بسیاری وجود دارد. شاید خیلی از برنامه نویسی‌ها از خود این سوال را بپرسند که تمام قوانین کاری و فرآیندهای یک سازمان را می‌توان با کد نویسی انجام داد، چه نیازی به این Engine‌ها برای مکانیزه کردن فرایندهای یک سازمان است؟

جواب این سوال را با یک مثال ساده آغاز می‌کنم :

فرض کنید یک فرآیند خیلی ساده داریم که کار آن دریافت اطلاعات از بانک اطلاعاتی و ارسال آن به مدیر بخش و دریافت تایید از طرف مدیر می‌باشد. این کار توسط دو کاربر انجام می‌شود که در سازمان نقش و سطح دسترسی مختلفی را دارا می‌باشند و به این نکته توجه کنید و آن اینکه فرض کنید زمانیکه نرم افزار شما در سازمانی در حال انجام کار می‌باشد به شما خبر داده می‌شود که کاربر x به مرخصی رفته و نقش آن به کسی دیگر سپرده شده است و این کار باید از طریق سیستم و با تایید مدیر انجام شود و یا سطح دسترسی افراد در سازمان عوض شود. این ساده‌ترین فرآیندی است که در زمان انجام پروژه با آن رو به رو می‌شویم . اگر این فرآیندهای ساده را بخواهیم با 100% کد نویسی انجام دهیم، تعداد خط کدها بسیار زیاد، زمان بر و انرژی زیادی از گروه گرفته می‌شود و مشکل به تعداد خط کد زیاد نیست، مشکل اصلی آن جایی است که برای پروژه بعدی قصد استفاده از این سیستم را داشته باشیم و نیاز به تغییر در بعضی از قسمت‌های سیستم باشد در این قسمت است که بیشترین زمان و انرژی از گروه گرفته می‌شود ولی در صورت استفاده از Workflow می‌توان در کمترین زمان و هزینه، پیچیده‌ترین Business Logic‌ها را پیاده سازی کرد.

نکته دیگری که در مورد اینگونه Engine‌ها باید گفته شود این است که در معماری SOA نقش فراوانی را دارا می‌باشند .

## نظرات خوانندگان

نویسنده: Petek

تاریخ: ۱۱:۱۱ ۱۳۹۱/۰۹/۰۲

با سلام

آیا کتاب فارسی در زمینه WF در بازار موجود می‌باشد و ممنون می‌شوم دوستان در این زمینه دست به قلم شده و مطالبی در زمینه یادگیری آن منتشر کنند. با تشکر

نویسنده: نصیری

تاریخ: ۱۱:۱۵ ۱۳۹۱/۰۹/۰۲

بدون شک، استفاده از Workflow Engine یا BPMS میتواند تاثیر زیادی در رویه‌های سازمانی بگذارد. متأسفانه در ایران کمتر به آنها توجه شده است. امیدوارم مطالب خوبی در آینده در این زمینه بخوانیم

نویسنده: مهدی پایروند

تاریخ: ۱۲:۳ ۱۳۹۱/۰۹/۰۲

مشکل جایی شروع میشه که خروجی هایی که هر سازمان از یه مفهوم دارند متفاوت هست، بطور مثال در حوزه نرم افزارهای مربوط به حضور و غیاب که به ظاهر شاید فقط ورود و خروج و اضافه کار و کسر کار است ولی در ریز کار و در اعمال قوانین مربوط به سازمان استفاده کننده کاملاً متفاوت و شاید تجمیع آنها در قالب هارد کد دست نیافتی باشد: برای نمونه میتوان روی نوع برخورد با اضافه کاری صحبت کرد که در یک سازمان طریق تجمیع و محاسبه این مورد در روز و ماه و سال تفاوت است. جایی حتی ورود قبل از ساعت کاری اضافه کار محسوب میشود و ضرایب و نوع ضرایب آن متغیر و متفاوت است، شاید با گروه بندی و راهکارهایی از این دست بتوان از این سفارشی سازی‌ها تا حدودی در امان ماند ولی در نهایت کار پروژه به اجبار باید به تولید یه موتور تعریف قوانین برای سپردن این مسئولیت (ساخت و ویرایش، به‌همراه پیچیدگیهای سازمانی آن) برود که در بعضی از اوقات WF جواب دلخواه هیچ کدام از دو سمت را نمیدهد.

نویسنده: محمد جواد تواضعی

تاریخ: ۱۵:۵۲ ۱۳۹۱/۰۹/۰۲

اینجا است که BPMS Engineها خودشان را نشان می‌دهند. تمام Bpmsها دارای ماژول BRE می‌باشند که شما می‌توانید قوانین کاری را در آن تعریف کنید. بعضی از این BPMSها مانند Intalio Bpms این امکان را به شما می‌دهند که قوانین کاری را در زمان اجرای برنامه تغییر دهید و دیگر سازمان لازم به تغییر فرایند و انتظار برای ایجاد برنامه جدید نیست. خودشان می‌توانند چارت سازمانی و قوانین کاری خود را در این Engine بدون اینکه لازم به توقف برنامه باشد تعریف و به روز رسانی کنند.

نویسنده: محسن

تاریخ: ۱۵:۵۸ ۱۳۹۱/۰۹/۰۲

مطلب بسیار جالبیه. از دوستان باتجربه ممنون میشم که BPMS را آموزش دهند. به شخصه با مواردی که ذکر شد کاملاً درگیر هستم و به دنبال راه حل.

نویسنده: محمد جواد تواضعی

تاریخ: ۱۷:۱۱ ۱۳۹۱/۰۹/۰۲

آقا محسن اگر عمر یاری کند بعد از آموزش Workflow ابتدا به آموزش زبان BPM می‌پردازم که یک رویکرد تحلیل می‌باشد سپس به آموزش یک BPMS

نویسنده: hrh

تاریخ: ۱۸:۲۹ ۱۳۹۱/۰۹/۰۲

بی صبرانه منتظر آموزش‌های مفیدتان هستم. بسیار ممنون

نویسنده: پویا امینی  
تاریخ: ۱۹:۲۴ ۱۳۹۱/۰۹/۰۵

با سلام. دوست عزیز من روی به برنامه‌ی کار می‌کنم که به سری سلسله مراتب داره مثلاً یک درخواست ثبت می‌شود و باید امضا مجاز آن را تایید کند و بعد از آن به سمت سرپرست واحد ارسال می‌شود و سرپرست واحد بعد از آن می‌تواند آن را به سمت کارشناس ارسال کند و هم می‌تواند به سمت انباردار ارسال کند. من این سلسله مراتب را با DB پیاده کردم حال برای سازمان‌های مختلف مجبورم این سلسله مراتب را عوض کنم مثلاً امضا مجاز از این سلسله مراتب حذف می‌شود برای این کار من خیلی باید کدهای خودم را تغییر دهم آیا می‌توانم این روند را با استفاده از WF پیاده کنم؟

ممنونم

نویسنده: محمد جواد تواضعی  
تاریخ: ۲۰:۲ ۱۳۹۱/۰۹/۰۵

سلام  
شما گردش کاری را به راحتی می‌توانید با WF انجام دهید.

نویسنده: پویا امینی  
تاریخ: ۲۰:۳۵ ۱۳۹۱/۰۹/۰۵

آیا منبع خاصی برای کار سراغ دارید؟ ممنون

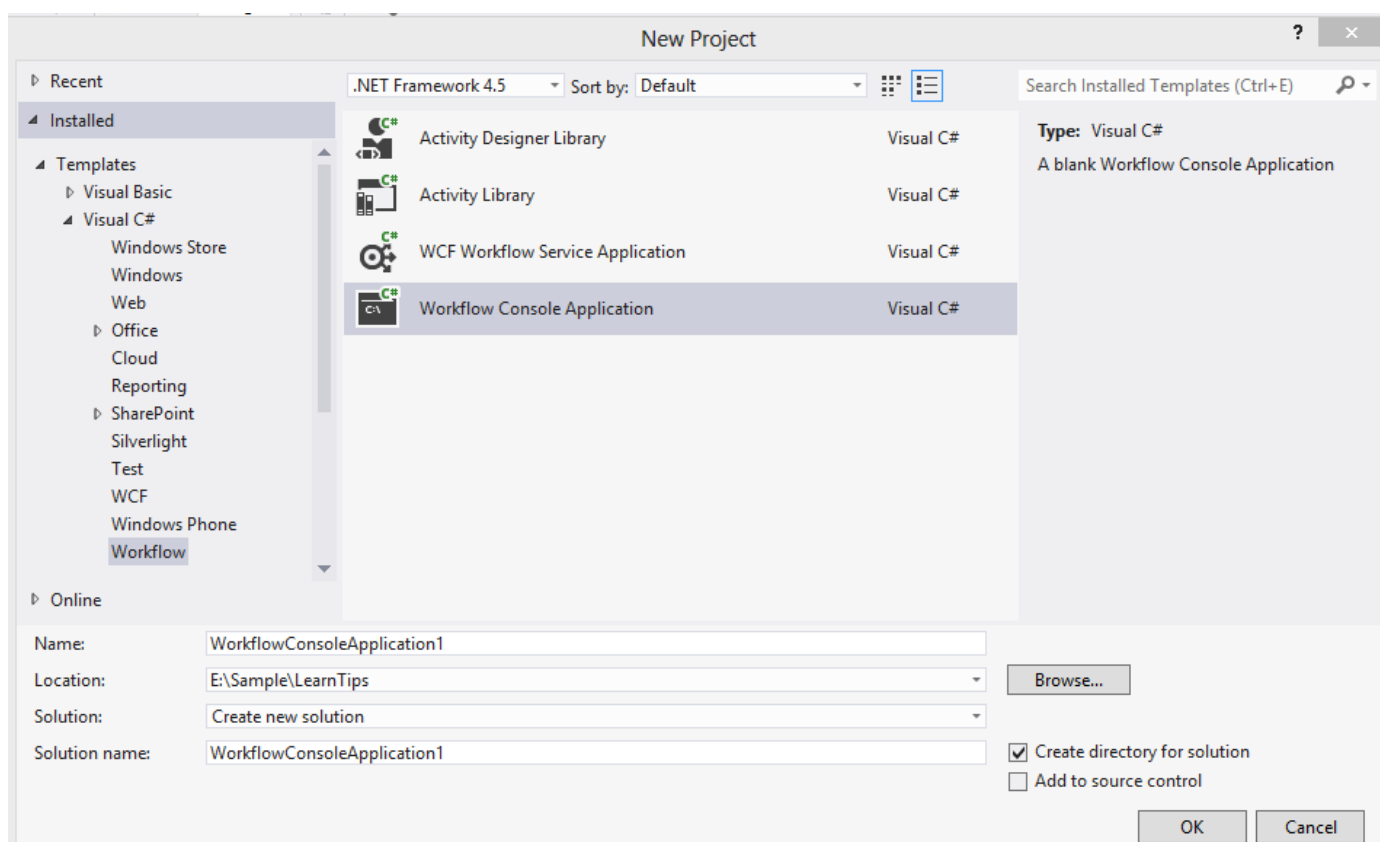
نویسنده: محمد جواد تواضعی  
تاریخ: ۲۳:۴۰ ۱۳۹۱/۰۹/۰۵

شما می‌توانید از طریق این سایت پیگیر باشید به زودی مطالب آموزشی در این زمینه گذاشته می‌شود. من سعی می‌کنم هر چه سریع‌تر این کار را انجام دهم، و لینک به سری Sample که در زمینه کاری شما است، در سایت قرار می‌دهم.

## ایجاد یک گردش ساده

در این دوره آموزشی قصد آموزش WF4 را داریم. برای ایجاد یک پروژه از نوع WF4 نیاز به VS2010 یا VS2012 است.

زمانیکه ویژوال استودیو را باز می‌کنید و بر روی گزینه ایجاد پروژه جدید کلیک می‌نمائید، در قسمت Workflow، چندین نوع پروژه وجود دارد که هر کدام از آن‌ها را به نوبت بررسی خواهیم کرد. ابتدا یک پروژه از نوع Workflow Console Application را ایجاد کنید:

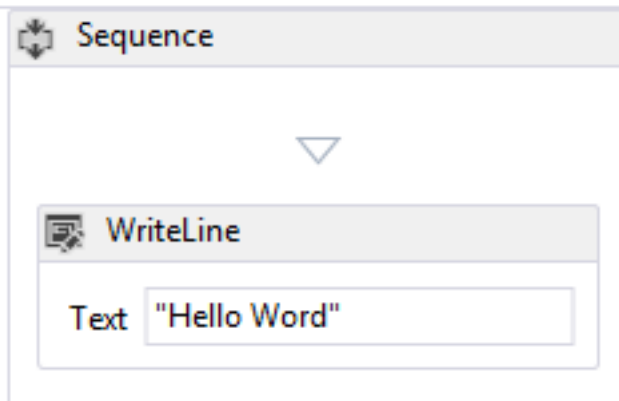


پس ایجاد پروژه، اگر دقت داشته باشید، فایل ایجاد شده با پسوند XAML می‌باشد و یک کلاس Program.cs هم در پروژه ایجاد گردیده که حاوی کلاس Main است و کار آن دقیقاً مثل برنامه‌های کنسول معمولی می‌باشد. اگر به قسمت پایین ویژوال استودیو دقت کنید، سه گزینه به نام‌های Imports, Arguments, Variables قابل مشاهده هستند: Variables: همین طور که از نام این گزینه بر می‌آید، برای تعریف متغیرها در طول یک فرآیند می‌باشد. Arguments: این گزینه هم مانند قسمت قبل عمل کرده، ولی تفاوت‌هایی باهم دارند. به عنوان مثال در گزینه ۱ می‌توان برای متغیر، محدوده ایجاد کرد. ولی در گزینه ۲ هیچگونه محدودیتی وجود ندارد و در تمام Workflowها می‌توان از آن استفاده کرد و دیگر اینکه در گزینه ۲ می‌توان به متغیر گفت که از نوع ورودی In یا از نوع خروجی Out و یا هر دو In/Out و یا اینکه از نوع Property باشد. Imports: این قسمت فضاهای نامی که در برنامه استفاده می‌شوند، نشان داده می‌شوند.

برای ایجاد یک Flow ابتدا باید از Toolbox، قسمت Control Flow، کنترل Sequence را به داخل صفحه کشید و پس از آن



می‌توانیم از سایر کنترل‌ها استفاده نمائیم. پس از این کار، از قسمت Primitives کنترل WriteLine را به درون Sequence انتقال می‌دهیم.



این کنترل برای چاپ مقادیر در خروجی می‌باشد. مانند کدی که در زیر نوشته شده است عمل کرده است و این کار را از طریق خاصیتی به نام Text انجام می‌دهد.

```
Console.WriteLine("Hello Word");
```

**نکته:** توجه کنید برای تغییر نام اجزایی که به درون صفحه کشیده می‌شوند، می‌توان بر روی نام آن دوبار کلیک کرده و نام آن را تغییر دهید و یا با انتخاب آن، در پنجره Properties نام آن را عوض کنیم. برای این کار کافی است مقدار گزینه DisplayName را با مقدار مورد نظر عوض نمائیم. این گزینه در تمام اجزای Workflow موجود می‌باشد. حال باید برنامه را اجرا کنیم تا خروجی را که چاپ رشته "Hello Word" است، مشاهده نمائیم. ولی قبل از آن باید تغییراتی در فایل Program.cs داده شود که به شرح زیر می‌باشد:

```
WorkflowInvoker.Invoke( new Workflow1());  
  
Console.WriteLine("Press ENTER to exit");  
Console.ReadLine();
```

از کلاس WorkflowInvoker برای اجرای Flow مورد نظر استفاده شده و این کار از طریق متد Invoke انجام داده می‌شود.

## اضافه کردن عناصر رویه ای

در این قسمت به بررسی عناصر رویه‌ای مانند دستورات IF,While,Assign می‌پردازیم .  
در این بخش توضیحات را با یک مثال آغاز می‌کنیم. در این مثال می‌خواهیم به بررسی کار با زمان بپردازیم.  
قبل از هر کاری، ابتدا نیاز به دو متغیر داریم؛ یکی کار شمارنده را بر عهده داشته و دیگری وظیفه ذخیره کردن ساعت را بر عهده دارد. برای انجام این کار، ابتدا مانند شکل زیر عمل می‌کنیم :

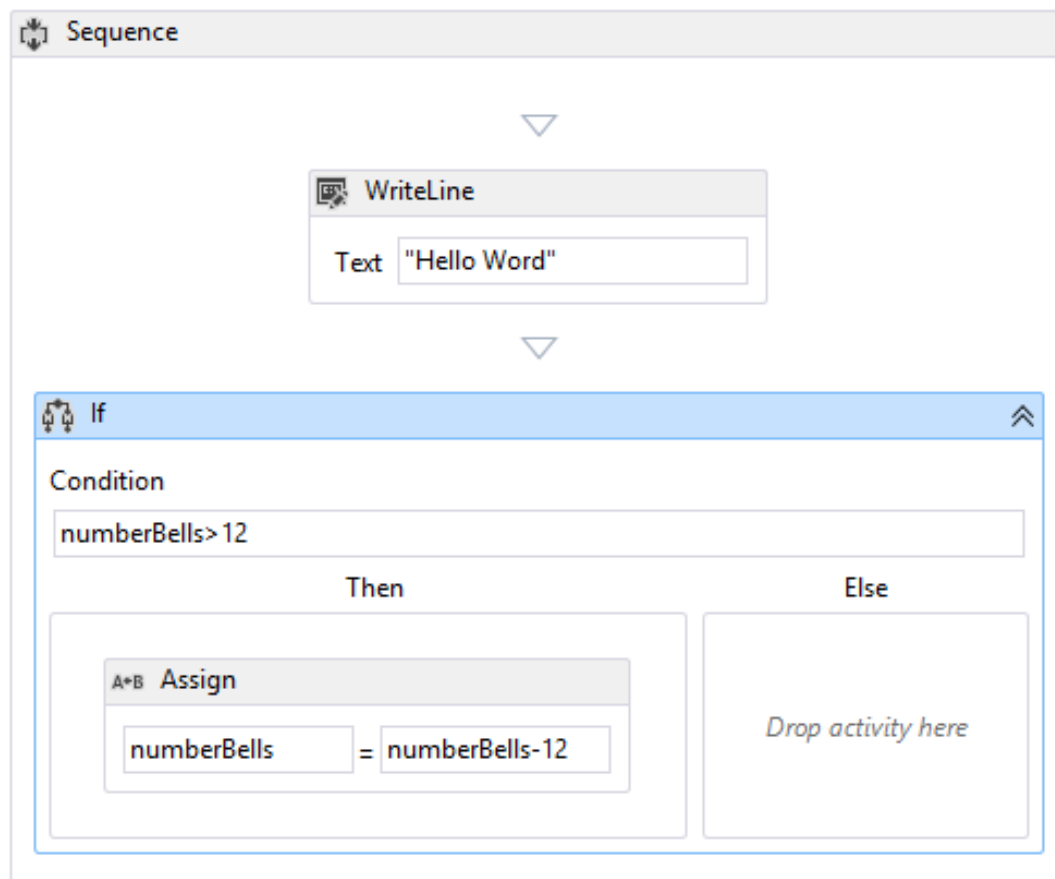
Name	Variable type	Scope	Default
Counter	Int32	Sequence	1
numberBells	Int32	Sequence	DateTime.Now.Hour

Create Variable

همانطور که در شکل مشاهده می‌کنید دو متغیر به نام‌های Counter و numberBells تعریف شده است و نوع هر دو، از جنس Int32 می‌باشد و در محدوده Sequence قرار گرفته‌اند. در قسمت پیش فرض، مقدار مورد نظر را تعیین کرده‌ایم.

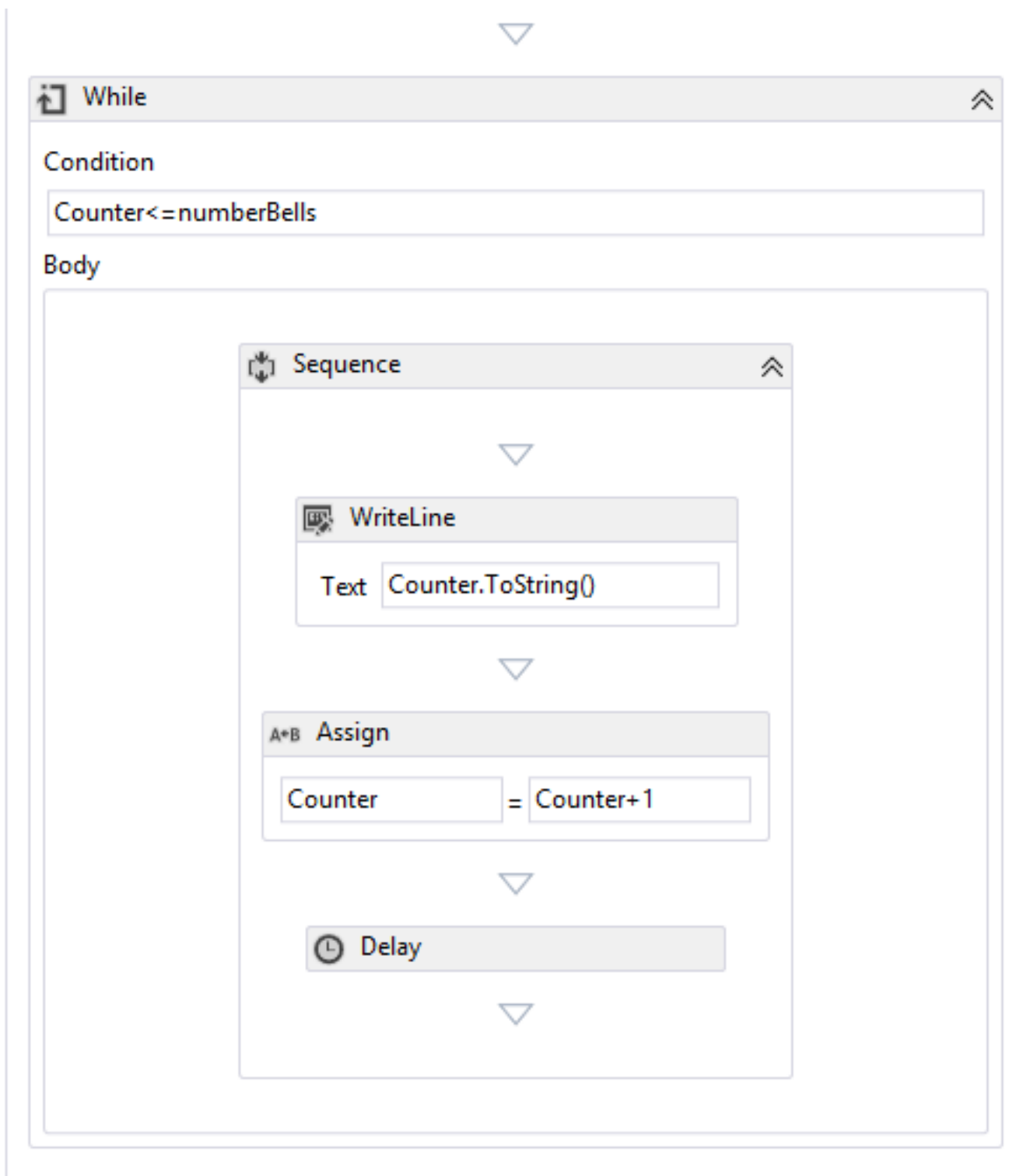
**نکته :** این مورد را در نظر داشته باشید که قبل از تعریف Variables باید حتماً یک Sequence در صفحه داشته باشیم تا بتوانیم محدوده متغیر مورد نظر را مشخص کنیم.

حال مانند قسمت پیش، ابتدا باید از Toolbox، قسمت Control Flow، کنترل Sequence را به داخل صفحه کشید و پس از آن می‌توانیم از سایر کنترل‌ها استفاده نمائیم. پس از این کار، از قسمت Primitives کنترل WriteLine را به درون Sequence انتقال می‌دهیم، سپس مانند شکل زیر یک کنترل IF که در قسمت Control Flow موجود می‌باشد را انتخاب کرده و به زیر کنترل WriteLine انتقال می‌دهیم. مانند شکل زیر:



همانطور که در شکل مشاهده می‌کنید، در کنترل IF در قسمت Condition شرط مورد نظر را مشخص می‌کنیم. در قسمت THEN از کنترل Assign استفاده شده است. از این کنترل وقتی استفاده می‌شود که قصد انتساب یک مقدار را به متغیری داریم. این کنترل در قسمت Primitives موجود است.

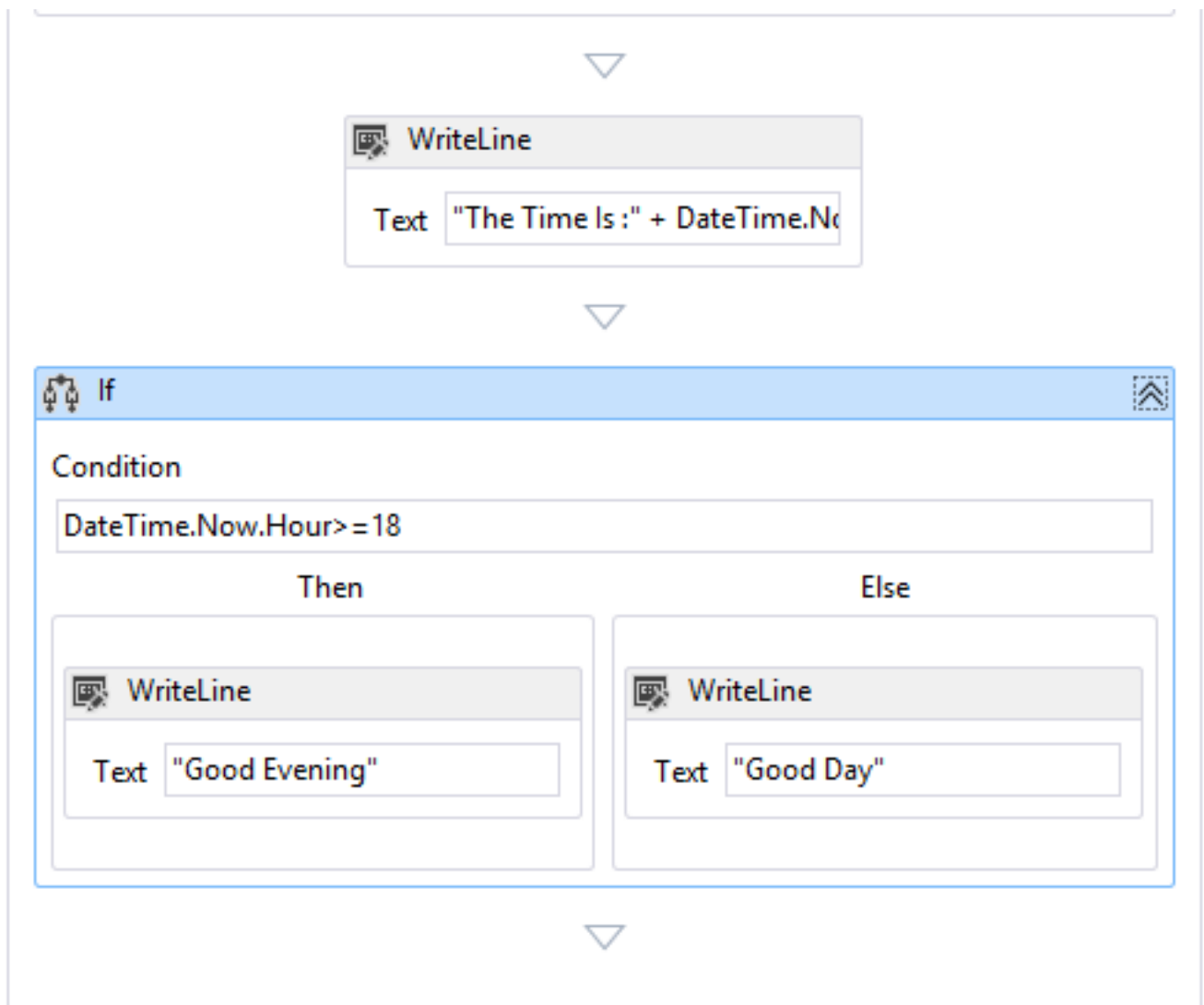
حال مانند شکل زیر عمل کرده و یک کنترل While را به زیر IF اضافه می‌کنیم؛ مانند شکل زیر :



کنترل While از دو قسمت تشکیل شده، شرط حلقه و بدنه آن. در قسمت شرط، مانند دستور IF عمل کرده و در قسمت بدنه، دستورات مورد نظر را مشخص می‌کنیم. در حلقه موجود تا زمانی که متغیر Counter از numberBells کوچکتر مساوی باشد، این حلقه اجرا می‌شود و در طی این جریان، ابتدا مقدار متغیر counter چاپ می‌شود، سپس یکی به مقدار آن اضافه می‌شود؛ البته با یک وقفه مشخص. برای اینکه بتوانیم در هر قسمت Workflow وقفه ایجاد کنیم، از کنترل Delay استفاده می‌شود. این کنترل دارای خاصیتی است به نام Duration. در این قسمت می‌توان میزان وقفه را مشخص نمود. برای مقدار دادن به این خاصیت، کنترل Delay را انتخاب کرده، سپس از قسمت Properties در VS2010 یا VS2012 می‌توان به خاصیت Duration مقدار داد.

```
TimeSpan.FromSeconds(1)
```

از این طریق وقفه‌ای که در بر نامه ایجاد می‌شود، یک ثانیه می‌باشد.



در این قسمت هم چک می‌شود که اگر ساعت جاری سیستم بیشتر از ۱۸ بود مقدار «عصر بخیر» چاپ شود، در غیر اینصورت مقدار «روز خوب» چاپ می‌شود.

**نکته :** در قسمت بدنه حلقه While حتما باید از کنترل Sequence استفاده شود، در غیر این صورت امکان تعریف بدنه حلقه While وجود ندارد.

## نظرات خوانندگان

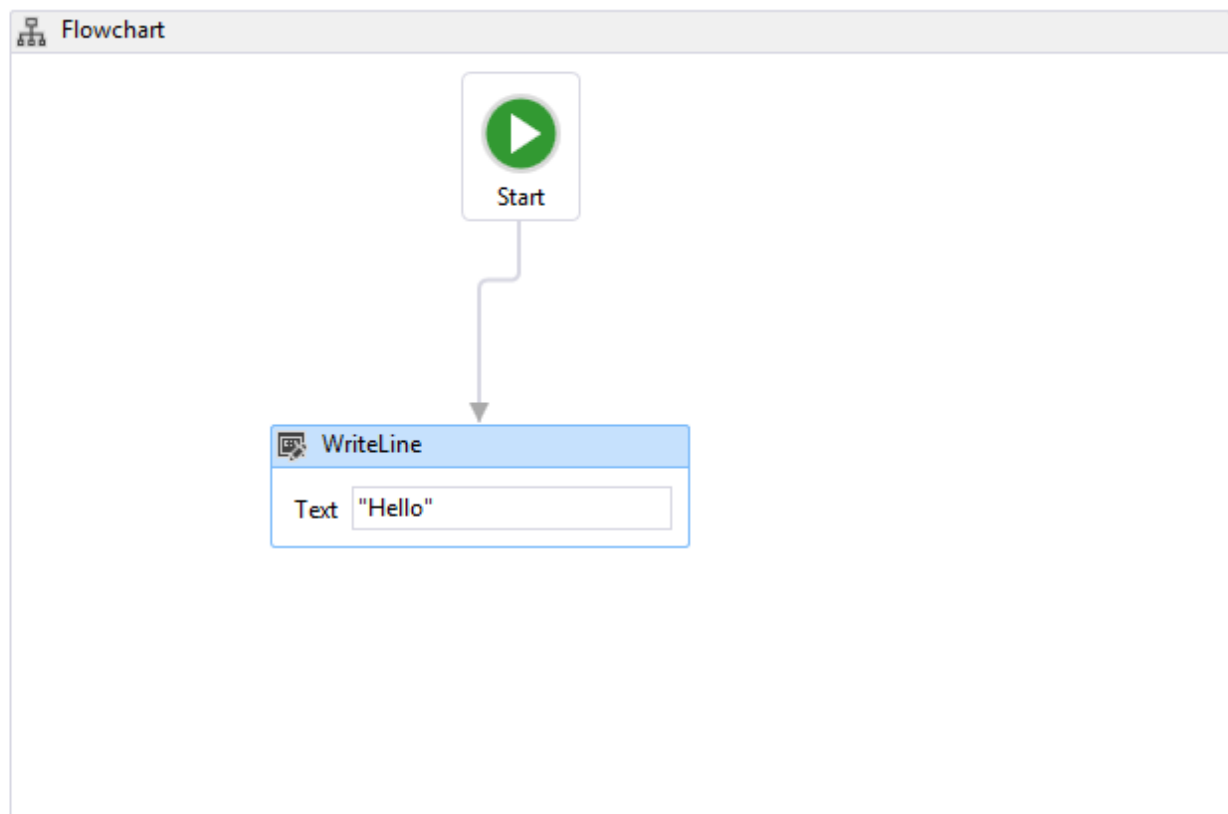
نویسنده: دهقان پیر  
تاریخ: ۱۱:۴۲ ۱۳۹۱/۰۹/۰۷

این مطلب Workflow بسیار خوب و ارزشمند است. سپاسگزارم

## Flowchart Workflow

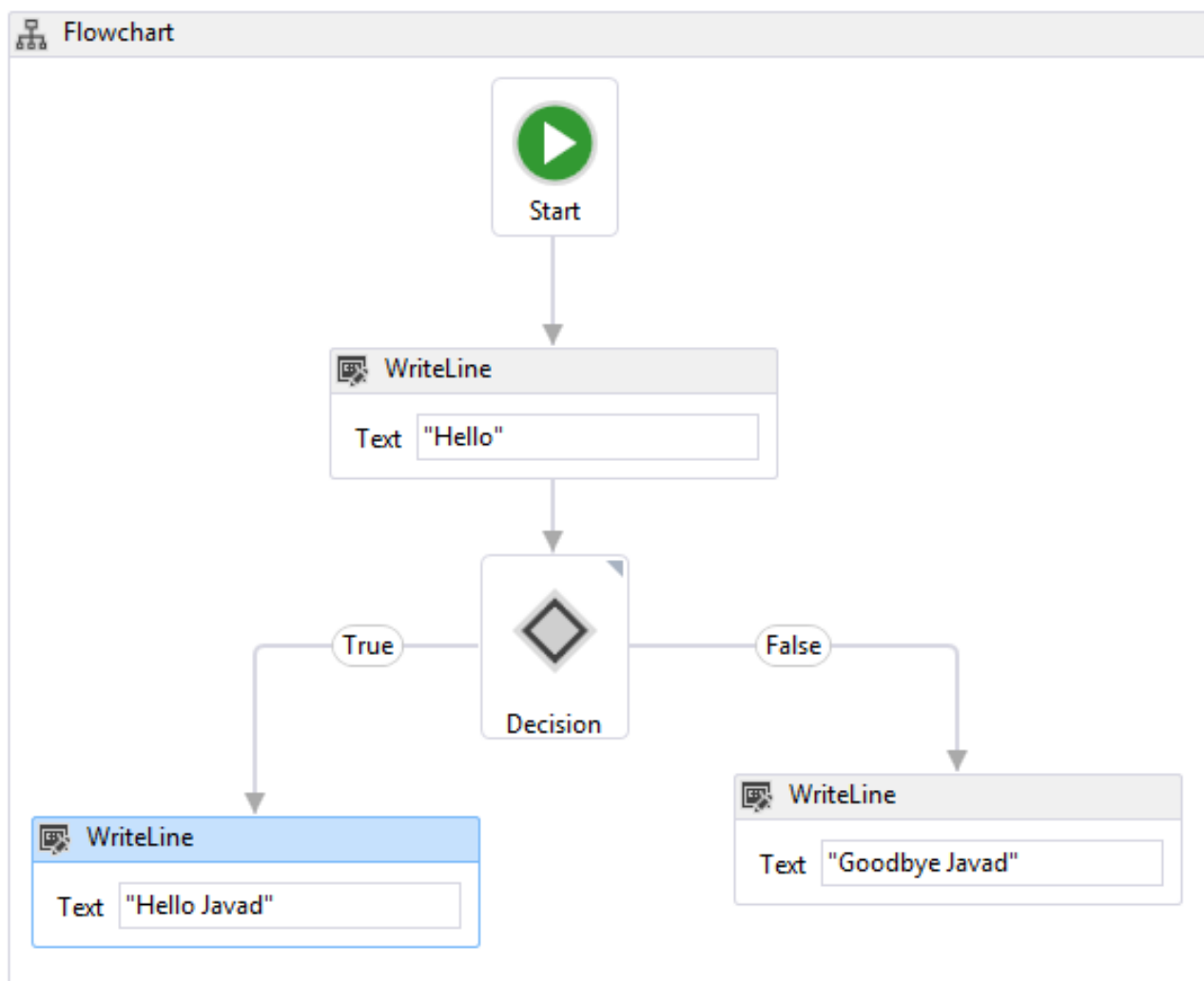
در این قسمت به ساخت یک Workflow از طریق Flowchart می‌پردازیم. در Workflow هایی که از طریق Flowchart تعریف می‌شوند، فعالیت‌ها به صورت درختی تعریف شده و با هم در ارتباط بوده و ارتباط آنها از طریق درخت‌های تصمیم‌گیری می‌باشد. برای استفاده از آن، از قسمت Toolbox، قسمت Flowchart، کنترل Flowchart را به داخل صفحه کشیده، حال می‌توانیم از دیگر کنترل‌ها در درون Flowchart استفاده کنیم.

**نکته:** تفاوتی که در Sequence و Flowchart است، در نحوه اجرای فعالیت‌ها می‌باشد. در Sequence فعالیت‌ها همانطور که از بالا به پایین چیده شده است اجرا می‌شوند، ولی در Flowchart می‌توان در خواست‌هایی را که جهت اجرای فعالیت‌ها فرستاده می‌شوند، کنترل کرد.



## FlowDecision

این کنترل برای مشخص کردن یک شرط می‌باشد و بر اساس این شرط، این کنترل می‌تواند دو خروجی داشته باشد. یکی در صورت درست بودن شرط و دیگری در صورت غلط بودن. این شرط هم مانند دستور IF یک خصوصیت شرط دارد که در قسمت Properties آن مشخص می‌شود.



`DateTime.Now.Hour==1`

اگر به برچسب‌های True و False دقت کنید، خروجی‌های کنترل را نشان می‌دهند. این نوشته‌ها ثابت نیستند و می‌توانید با انتخاب کنترل، از قسمت Properties آنها را عوض کنید.

**نکته :** به عکس بالا دقت کنید. زمانیکه از کنترل خروجی گرفته شده است، سمت پیکان‌ها به سمت کنترل WriteLine می‌باشد و این به این معنا می‌باشد که این پیکان از کنترل Decision به سمت پایین کشیده شده است. اگر این کار را بر عکس انجام دهیم دیگر کنترل Decision از کنترل WriteLine را نمی‌شناسد و در صورت درست بودن شرط در خروجی مقداری چاپ نمی‌شود.

در کنترل Flowchart امکان استفاده از کنترل Sequence وجود دارد و همین‌طور بالعکس. این امر بستگی به نوع فرآیند شما دارد که چه موقع باید از ساختار ترتیبی استفاده شود و کجا درختی و شاید این امکان وجود داشته باشد که فرآیند مورد نظر ترکیبی از دو باشد.

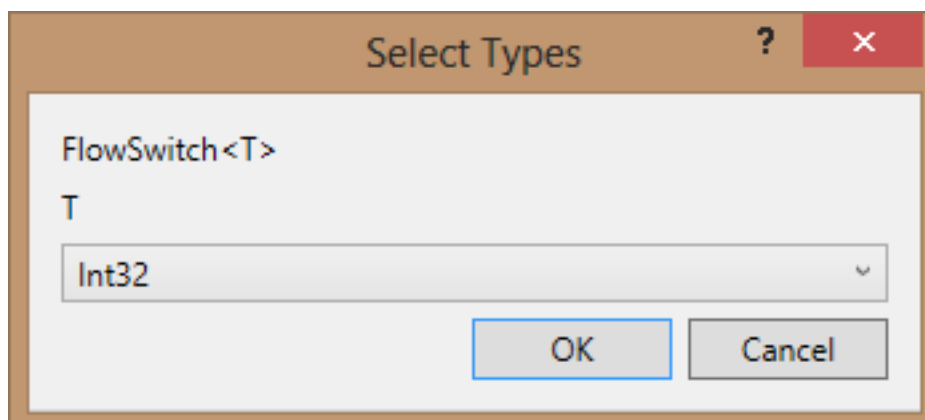
### Flow Switch

زمانیکه از کنترل Decision استفاده می‌کنید، محدود می‌شوید به دو مقدار True و False. ولی زمانیکه از کنترل Flow Switch یا Switch استفاده می‌کنید، این محدودیت از بین می‌رود و شما می‌توانید n دستور عمل را اجرا کنید. این کنترل دقیقاً شبیه دستور Switch Case در C# عمل می‌کند.

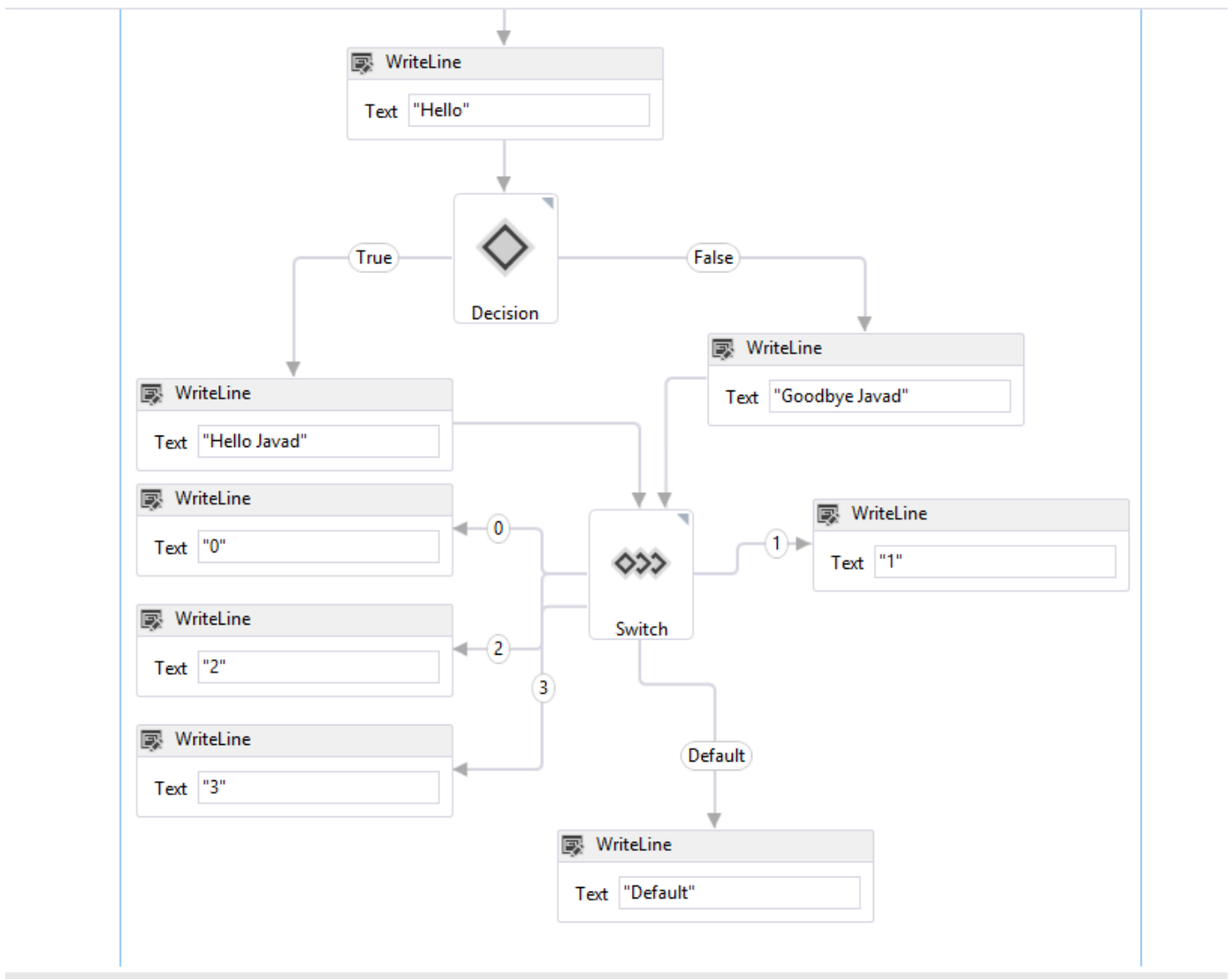
نحوه کار کردن آن به این صورت می‌باشد که ابتدا کنترل را انتخاب کرده، سپس از قسمت Properties، در قسمت Expression



آن، مقدار مورد نظر خود را وارد می‌کنیم. اینجا این سؤال پیش می‌آید این کنترل از کجا متوجه می‌شود که خروجی Expression آن باید از چه نوعی باشد؟ اگر به شکل کنترل Flow Switch، در قسمت Toolbox دقت کرده باشید می‌بینید که نحوه نوشتن آن مانند این است که یک Generic Class تعریف کرده باشیم و دقیقاً به همین صورت می‌باشد. زمانیکه این کنترل به درون کنترل Flowchart کشیده می‌شود، شکل زیر نمایان خواهد شد.



همانطور که مشاهده می‌کنید نوع T، از نوع عددی صحیح تعیین شده و این نوع، مقدار پیش فرض است. البته می‌توان این نوع را عوض کرد. T می‌تواند از نوع Object، مثلاً کلاس دانش آموزان باشد و یا از نوع یک Workflow دیگر، که در جلسات بعدی توضیح داده خواهد شد.



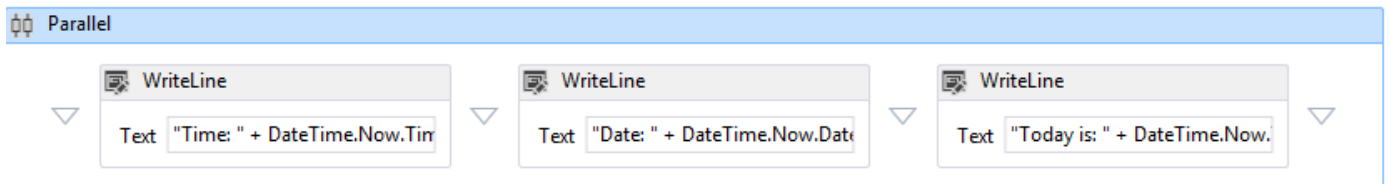
همانطور که در شکل بالا مشاهده می‌کنید از کنترل Decision دو خروجی گرفته شده است و هر دو به کنترل Switch اشاره دارند. این امر به این معنا می‌باشد که شرط در هر حالتی که باشد، چه درست چه غلط، پس از چاپ رشته مورد نظر، به کنترل Switch اشاره می‌کند.

```
Convert.ToInt32(((DateTime.Now.Month % 12) + 1) / 4)
```

شرط کنترل Switch در بالا مشخص شده است و بر اساس عددی که بر می‌گرداند، رشته مورد نظر را چاپ می‌کند. همچنین یک مقدار Default وجود دارد که حتما باید مشخص شود.

### Parallel

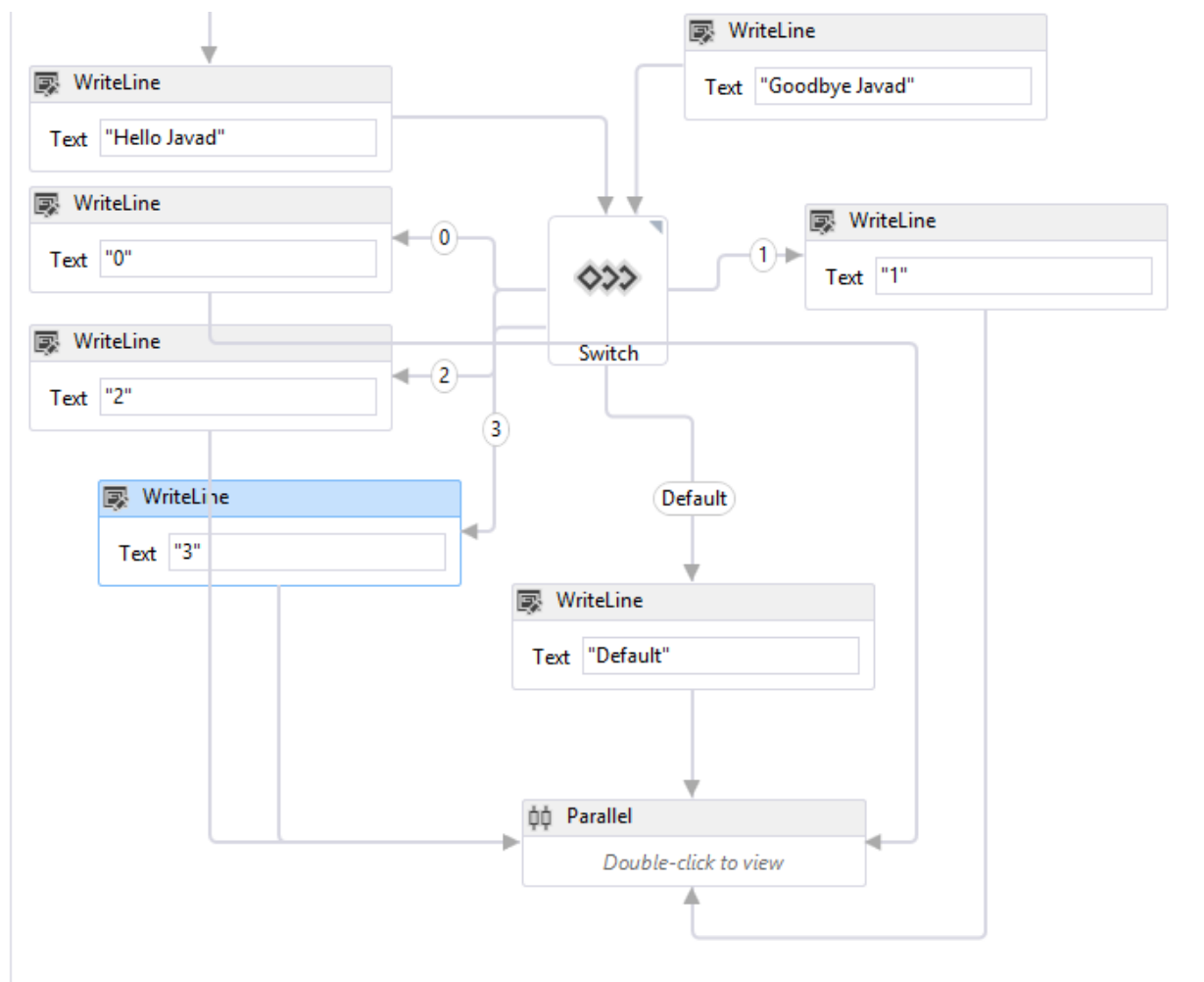
از این کنترل برای اجرای هم زمان چندین فعالیت استفاده می‌شود .  
به مثال زیر توجه کنید :



در کنترل‌های WriteLine کد زیر نوشته شده است :

```
"Time: " + DateTime.Now.TimeOfDay.ToString()
"Date: " + DateTime.Now.Date.ToShortDateString()
"Today is: " + DateTime.Now.ToString("dddd")
```

کنترل Parallel باعث می‌شود ، که سه رشته هم زمان در خروجی چاپ شود .



اگر به عکس بالا دقت کنید، از بین حالت‌های مختلفی که کنترل Switch می‌تواند داشته باشد، یک اتصال به کنترل Parallel صورت گرفته و این به آن معنا است که در هر حالتی کنترل Parallel انجام می‌گیرد. البته شایان ذکر است که این روش را می‌توان محدود کرد.

## نظرات خوانندگان

نویسنده: Mehrsa

تاریخ: ۱۳۹۱/۰۹/۰۶ ۱۲:۸

من اینو تو محیط کنسول نوشتم این ارور رو بهم داد

workflow1': the private implementation of activity '1: workflow1' has the following validation error: condition 'must be set before the flowdecision in flowchart 'flowchart' can be used

نویسنده: محمد جواد تواضعی

تاریخ: ۱۳۹۱/۰۹/۰۶ ۱۴:۴۲

سلام دوست عزیز

به نظر می‌آید که مشکل از شرطی است که برای Flowdecision باید ست شود، شما قبل از اینکه عناصر را به هم متصل کنید ابتدا باید تکلیف Flowdecision و شرط مربوطه را مشخص کنید و سپس خروجی مورد نظر را از آن دریافت کنید .

نویسنده: Mehrsa

تاریخ: ۱۳۹۱/۰۹/۰۶ ۱۴:۵۸

بله مشکل از همینجا بود ممنون

و یک سوال : فرض بر اینکه ما یک برنامه حسابداری داریم که به چند شرکت مختلف فروختیم هر شرکت روش کاره خودش رو برای کار داره مثلاً به شرکت یک سند حسابداری مستقیم به دست حسابدار می‌رسونه ولی به شرکت دیگه اول به دست رییس مربوط می‌رسونه و در صورت تایید به دست حسابدار امکان نوشتن workflow وجود داره که خود کاربر نهایی بتونه به همچین چیزو تنظیم کنه بنویسه و به راحتی ازش استفاده کنه و اینکه از workflow میشه تو معماری چند لایه استفاده کرد؟ و این ترجمه کتاب wf beginning از apress نیست ؟ ولی در کل از این مطالبتون بسیار ممنون مرجع فارسی از workflow میشه گفت اصلاً وجود نداره

نویسنده: محمد جواد تواضعی

تاریخ: ۱۳۹۱/۰۹/۰۶ ۱۶:۳۳

بله دوست عزیز من زمانی که شروع به یاد گیری WF کردم از همان PDF که مربوط به نشریات Apress است شروع کردم و اینکه از WF در هر نوع معماری که بشود در .NET پیاده سازی کرد قابل استفاده می‌باشد و هیچ گونه محدودیتی برای توسعه دهنده ایجاد نمی‌کند . با استفاده از Activity code می‌توان این جور مشکل‌ها را با در WF پیاده سازی کنیم .

نویسنده: حمید

تاریخ: ۱۳۹۱/۰۹/۰۷ ۱۲:۱۸

آقا خیلی ممنون. عالیه. اگه میشه سریعتر پیش برین.

نویسنده: محمد جواد تواضعی

تاریخ: ۱۳۹۱/۰۹/۰۷ ۱۷:۱۶

سلام

چشم آقا حمید :

نویسنده: Dezireh

تاریخ: ۱۱:۳۶ ۱۳۹۱/۱۲/۲۳

ممنون از مطالب مفید و عالی که در سایت قرار میدید.

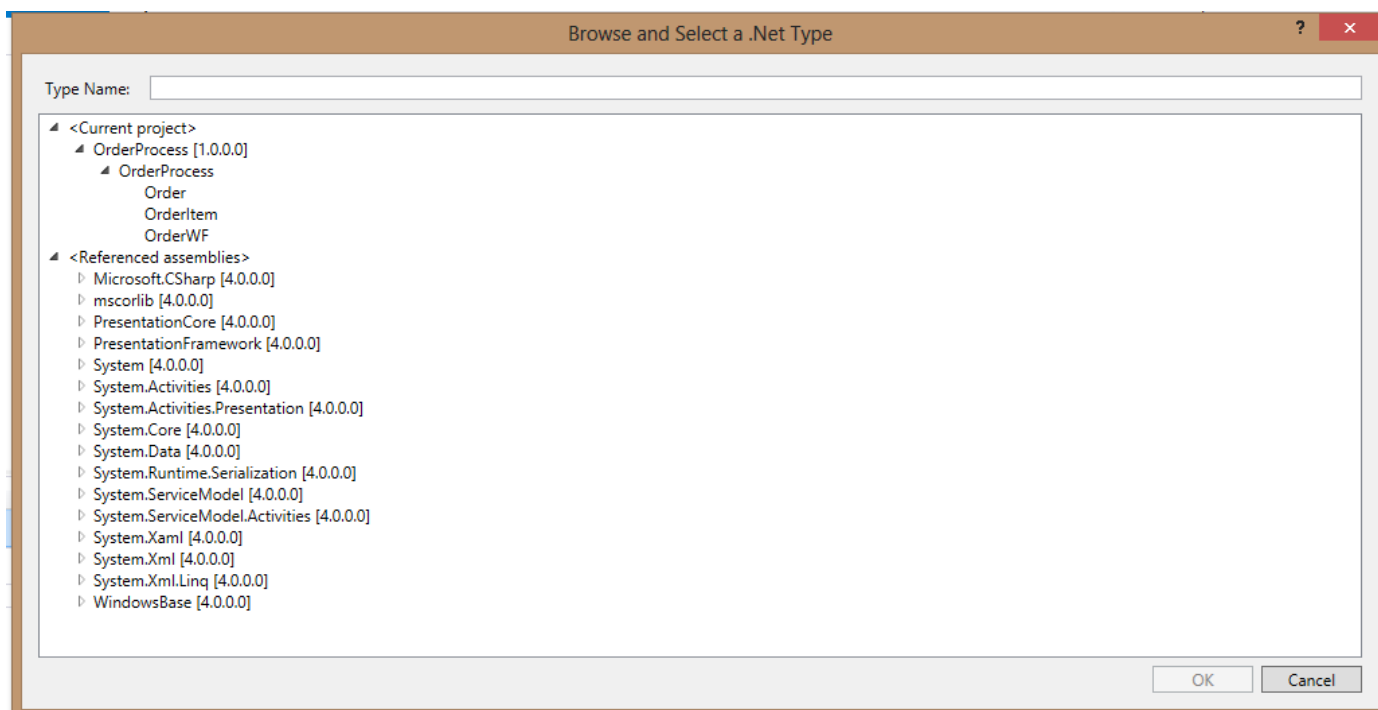
در این قسمت به پیاده سازی یک فرآیند سفارش ساده می‌پردازیم. ابتدا یک پروژه از نوع Workflow Console Application را ایجاد کرده و نام آن را Order Process می‌گذاریم و سپس کلاس‌های زیر را به آن اضافه می‌کنیم:

```
public class OrderItem
{
    public int OrderItemID { get; set; }
    public int Quantity { get; set; }
    public string ItemCode { get; set; }
    public string Description { get; set; }
}

public class Order
{
    public Order()
    {
        Items = new List<OrderItem>();
    }
    public int OrderID { get; set; }
    public string Description { get; set; }
    public decimal TotalWeight { get; set; }
    public string ShippingMethod { get; set; }
    public List<OrderItem> Items { get; set; }
}
```

در اینجا دو کلاس تعریف شده است؛ یکی به نام OrderItem می‌باشد که شامل اطلاعات مربوط به میزان سفارش بوده و دیگری کلاس Order می‌باشد که شامل مشخصات سفارش است. سپس فایل OrderWF.xaml را باز کرده و شروع به ساخت فرآیند مورد نظر می‌کنیم. ابتدا یک Sequence را به درون صفحه کشیده و پس از آن در قسمت Arguments دو متغیر را تعریف می‌کنیم. یکی به نام TotalAmount و از نوع Decimal و Out می‌باشد و دیگری به نام OrderInfo که از نوع کلاس Order و In می‌باشد. سپس یک کنترل WriteLine را به آن اضافه می‌کنیم و در خاصیت Text آن رشته "Order Received" را قرار می‌دهیم. در ادامه یک کنترل Assign را در زیر آن قرار داده و مقدار متغیر TotalAmount را مساوی صفر وارد می‌کنیم.

**نکته :** برای اینکه نوع متغیر OrderInfo را از نوع کلاس Order قرار دهیم، ابتدا DropDown مربوطه را انتخاب کرده و گزینه Browse For Type را انتخاب می‌کنیم تا پنجره مورد نظر باز شود و از طریق آن، کلاس مورد نظر را انتخاب می‌کنیم. اگر در این قسمت کلاس مورد نظر یافت نشد، نیاز است ابتدا عمل Build Project را یک بار انجام دهیم.



Name	Direction	Argument type	Default value
OrderInfo	In	Order	Enter a C# expression
TotalAmount	Out	Decimal	Default value not supported

Create Argument

بعضی از کنترل‌های Workflow در قسمت Toolbox موجود نمی‌باشند. از جمله این کنترل‌ها می‌توان به کنترل Add اشاره کرد. برای استفاده از این کنترل، ابتدا باید آن را به لیست کنترل‌ها اضافه نمود. جهت این امر، ابتدا در قسمت Toolbox یک Tab جدید را با نام دلخواه ایجاد کرده و سپس بر روی Tab کلیک راست نموده و گزینه Choose Items را انتخاب می‌کنیم. سپس از قسمت System.Activities.Components کنترل Add را انتخاب کرده و سپس بر روی دکمه OK کلیک می‌نمائیم. حال کنترل Add به لیست کنترل‌ها در Tab مورد نظر اضافه شده است.

در ادامه یک کنترل Switch را به فرایند خود اضافه کرده و مقدار آن را برابر String قرار می‌دهیم؛ زیرا نوع داده‌ای که در قسمت Expression کنترل Switch قرار می‌گیرد، از نوع رشته می‌باشد. پس از اضافه کردن کنترل مورد نظر، کد زیر را به قسمت Expression کنترل اضافه خواهیم کرد:

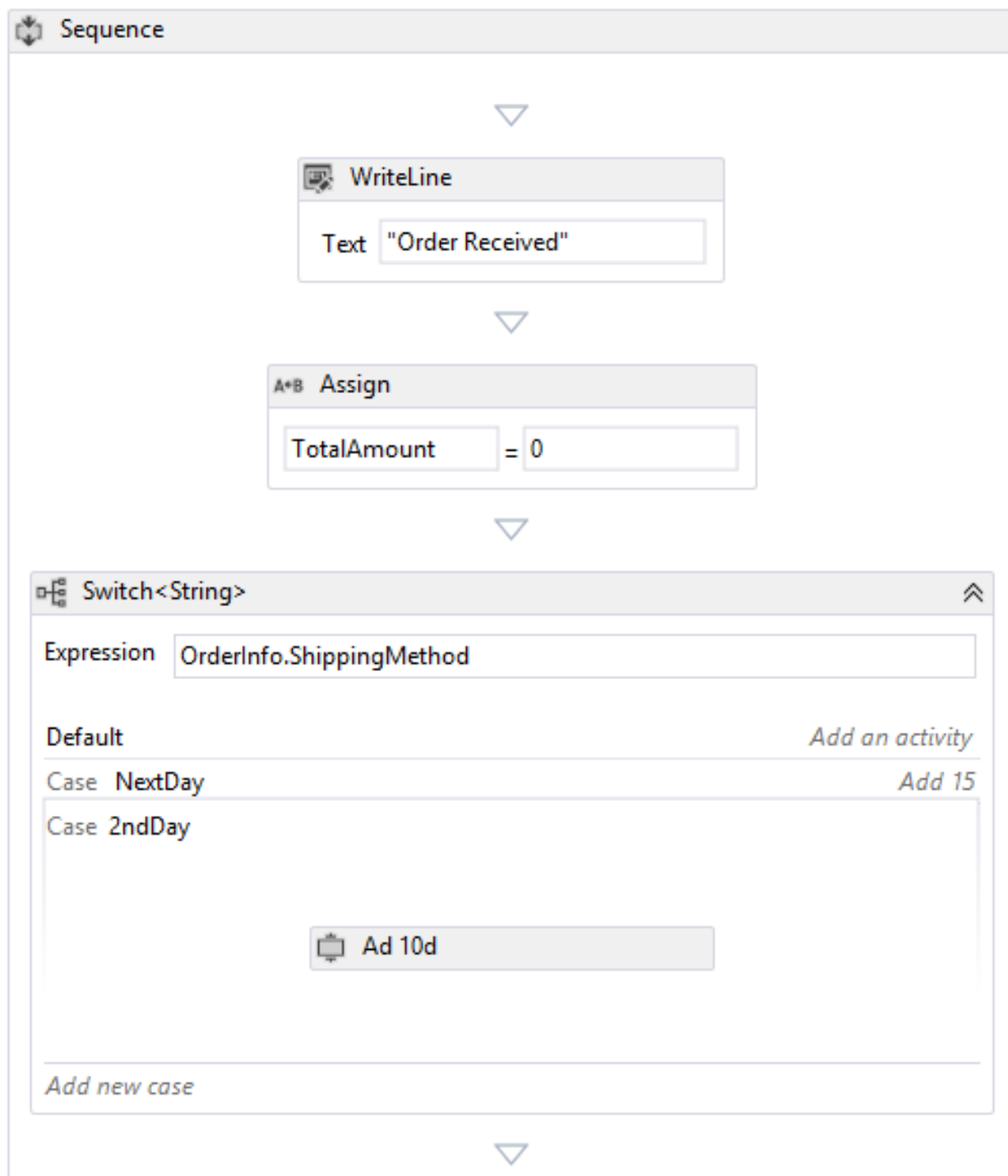
OrderInfo.ShippingMethod

سپس در کنترل Switch، بر روی قسمت Add new case کلیک کرده و رشته‌های مورد نظر را اضافه می‌کنیم که شامل "" و ""NextDay و ""2ndDay می‌باشند. اکنون در بدنه هر دو Case، کنترل Add را اضافه می‌کنیم. در هنگام اضافه کردن باید برای سه خصوصیت، نوع مشخص شود و نوع هر سه را برابر Decimal قرار می‌دهیم.

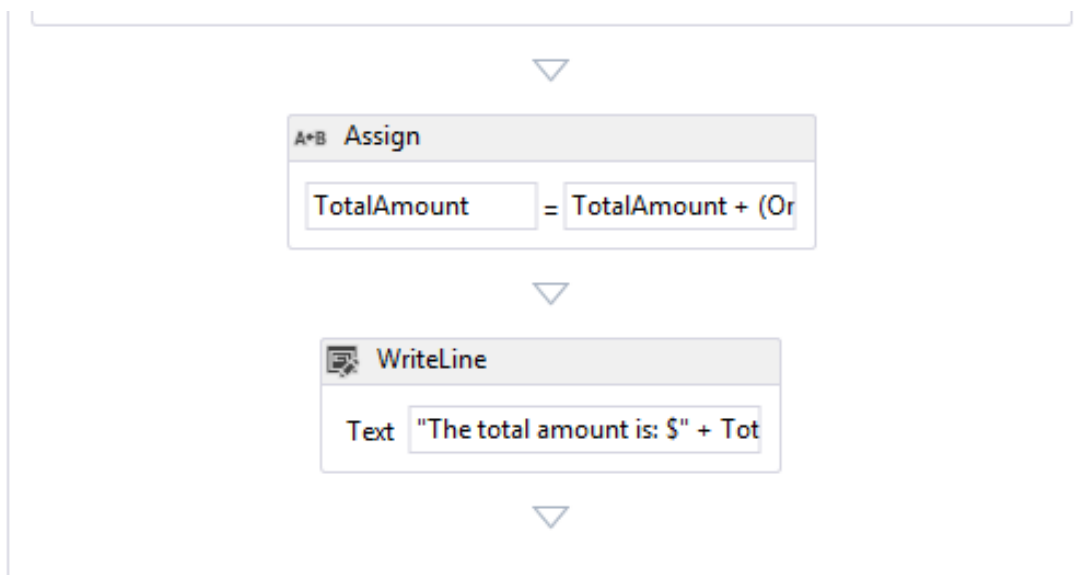
در ادامه کنترل Add را انتخاب کرده و به خاصیت Right آن‌ها به ترتیب مقدارهای 10.0m و 15.0m را اضافه می‌کنیم و برای خصوصیت Result هر دو کنترل، متغیر TotalAmount را انتخاب می‌کنیم. سپس یک کنترل Assign را به صفحه اضافه کرده و در قسمت To، متغیر TotalAmount را قرار می‌دهیم و در قسمت Value کد زیر را:

TotalAmount + (OrderInfo.TotalWeight \* 0.50m)

و در آخر با استفاده از کنترل WriteLine به چاپ محتوای متغیر TotalAmount می‌پردازیم.







اکنون برای اینکه بتوانیم برنامه را اجرا کنیم، کد زیر را به کلاس Program.cs اضافه می‌کنیم:

```
static void Main(string[] args)
{
    Order myOrder = new Order
    {
        OrderID = 1,
        Description = "Need some stuff",
        ShippingMethod = "2ndDay",
        TotalWeight = 100
    };
    IDictionary<String, object> input = new Dictionary<String, Object>
    {
        { "OrderInfo", myOrder }
    };
    IDictionary<String, Object> output = WorkflowInvoker.Invoke(new OrderWF(), input);
    Decimal total = (Decimal)output["TotalAmount"];
    Console.WriteLine("Workflow returned ${0} for my order total", total);
    Console.WriteLine("Press ENTER to exit");
    Console.ReadLine();

    //Activity workflow1 = new OrderWF();
    //WorkflowInvoker.Invoke(workflow1);
}
```

در اینجا علت استفاده از `IDictionary`، نوع خروجی متد `Invoke` می‌باشد. در ادامه به کامل کردن این مثال پرداخته می‌شود.

## نظرات خوانندگان

نویسنده: علیرضا جهانشاهلو  
تاریخ: ۱۳۹۱/۰۹/۰۹ ۱۲:۲۵

خیلی ممنون از آموزش مفیدتون لطفا ادامه بدید.

نویسنده: محسن موسوی  
تاریخ: ۱۳۹۱/۰۹/۰۹ ۱۲:۵۲

تشکر از این سری آموزش ها.  
لطفا به مقوله چگونگی کاربرد آنها در وب نیز بپردازید.  
منتظر ادامه مطلبتون هستیم.

نویسنده: محمد جواد تواضعی  
تاریخ: ۱۳۹۱/۰۹/۰۹ ۲۳:۴

آقا محسن حتما مثالی که چگونگی کاربرد Workflow در برنامه های وب و دسکتاپ به چه نحو است حتما گفته می شود .

در این قسمت به تکمیل مثالی که در [قسمت قبل زده](#) شد پرداخته می‌شود و همچنین کنترل‌های Try Catch , Foreach نیز بررسی خواهند شد.

در ابتدا دو کلاس به نام‌های ItemInfo و OutOfStockException را به برنامه اضافه می‌کنیم. کلاس اول برای ذخیره سازی مشخصات هر سفارش و کلاس دیگر برای مدیریت خطاها می‌باشد.

```
public class ItemInfo
{
    public string ItemCode { get; set; }
    public string Description { get; set; }
    public decimal Price { get; set; }
}

public class OutOfStockException : Exception
{
    public OutOfStockException()
        : base()
    {
    }

    public OutOfStockException(string message)
        : base(message)
    {
    }
}
```

در Workflow مورد نظر که به نام OrderWF.xaml می‌باشد، پس از کنترل Assign که برای صفر کردن مقدار متغیر TotalAmount از آن استفاده می‌شود، یک کنترل ForEach را به Flow جاری اضافه می‌کنیم. این کنترل دارای دو خاصیت به نام‌های Type Arguments و Values می‌باشد. اولین خاصیت که مقدار پیش فرض آن، مقدار عددی Int32 است، برای مشخص کردن نوع متغیر حلقه و دیگری برای مشخص کردن نوع منبع داده حلقه تعریف شده‌اند.

A+B Assign

TotalAmount = 0



ForEach<OrderItem>

ForEach item in OrderInfo.Items

Body

Sequence

LookupItem

A+B Assign

TotalAmount = TotalAmount + (ite

همانطور که در شکل بالا مشخص می‌باشد، Type Arguments حلقه برابر با کلاس OrderItem می‌باشد. Values هم برابر با OrderInfo.Items است. از این جهت نوع حلقه را از جنس کلاس OrderItem مشخص کرده‌ایم تا کنترل بر روی مقادیر Items اجرا شود (لیستی از کلاس مورد نظر).

حال همانند شکل بالا، در قسمت Body کنترل ForEach، یک کنترل Sequence را ایجاد کرده و سپس برای اینکه کنترل LookupItem را ایجاد کنیم، ابتدا باید یک Code Activity را به پروژه اضافه کنیم. به همین منظور پروژه جاری را انتخاب کرده و یک Code Activity به آن اضافه و نام آن را LookupItem می‌گذاریم. سپس کد زیر را به آن اضافه می‌کنیم:

```

public sealed class LookupItem : CodeActivity
{
    // Define an activity input argument of type string
    public InArgument<string> ItemCode { get; set; }
    public OutArgument<ItemInfo> Item { get; set; }

    // If your activity returns a value, derive from CodeActivity<TResult>
    // and return the value from the Execute method.
    protected override void Execute(CodeActivityContext context)
    {
        // Obtain the runtime value of the Text input argument
        ItemInfo i = new ItemInfo();
        i.ItemCode = context.GetValue<string>(ItemCode);
        switch (i.ItemCode)
        {
            case "12345":
                i.Description = "Widget";
                i.Price = (decimal)10.0;
                break;
            case "12346":
                i.Description = "Gadget";
                i.Price = (decimal)15.0;
                break;
            case "12347":
                i.Description = "Super Gadget";
                i.Price = (decimal)25.0; break;
        }

        context.SetValue(this.Item, i);
    }
}

```

در این کد، دو متغیر تعریف شده‌اند؛ یکی از نوع رشته بوده و از طریق آن، دستور Switch تصمیم می‌گیرد که کلاس ItemInfo را با چه مقادیری پر کند. متغیر دیگر از نوع کلاس ItemInfo می‌باشد و برای گرفتن مقدار کلاس از دستور Switch تعریف شده است. حال برای اینکه بتوانیم از Code Activity مورد نظر استفاده کنیم، ابتدا باید پروژه را یکبار Build کنیم. اکنون در قسمت Toolbox یک Tab ای به نام پروژه ایجاد شده است و در آن یک کنترل به نام LookupItem موجود می‌باشد. آن را گرفته و به درون Sequence انتقال می‌دهیم.

سپس برای مقدار دادن به متغیرهای تعریف شده در Code Activity، کنترل LookupItem را انتخاب کرده و در قسمت Properties به خصوصیت ItemCode، کد زیر را اضافه می‌کنیم:

```
item.ItemCode
```

**نکته :** از کلاس Code Activity برای ارسال و دریافت مقادیر به درون Workflow استفاده می‌شود.

### Try Catch

از این کنترل برای مدیریت خطاها استفاده می‌شود.

ابتدا یک کنترل Try Catch را به Workflow اضافه کرده، مانند شکل زیر:

**TryCatch**

**Try**

**ForEach<OrderItem>**

ForEach  in

**Body**

**If**

Condition

**Then** **Else**

**Then**:

**Else**: *Drop activity here*

**Catches**

OutOfStockException	WriteLine
---------------------	-----------

*Add new catch*

**Finally** *Add an activity*

در بدنه Try می‌توان از کنترل‌های مورد نظر استفاده کنیم و همانطور که در شکل بالا مشخص است، از کنترل Throw برای ایجاد خطا استفاده شده است. کنترل جاری را انتخاب کرده و از قسمت Properties در خاصیت Exception کد زیر را اضافه می‌کنیم:

```
new OutOfStockException("Item Code"+item.ItemCode)
```

این دستور باعث ایجاد یک خطا از نوع کلاس OutOfStockException می‌شود. برای کنترل خطای مورد نظر در قسمت Catches مانند شکل زیر عمل می‌کنیم.

TryCatch

Try

ForEach<OrderItem>

Catches

OutOfStockException

exception

WriteLine

Text "Item is out of stock - " + exce

Add new catch

Finally

Add an activity

## نظرات خوانندگان

نویسنده: حامد

تاریخ: ۱۳۹۱/۰۹/۱۵ ۷:۲۶

با سلام

مطالب بسیار آموزنده ایه و خوبی اون اینه که همه‌ی برنامه نویسان از اون استفاده می‌برند.  
لطفاً ادامه بدید.

نویسنده: محمد جواد تواضعی

تاریخ: ۱۳۹۱/۰۹/۱۸ ۳:۵۳

سلام آقا حامد

حتما , من اینجا از همه دوستان عذر خواهی میکنم که در ارسال مطالب وقفه ایجاد شد .

نویسنده: ترابی

تاریخ: ۱۳۹۲/۰۳/۲۲ ۱۱:۱۲

باتشکر از مطالب مفیدتون

منتظر ادامه‌ی آموزش‌ها هستیم

موفق باشید

نویسنده: سروش شیرالی

تاریخ: ۱۳۹۴/۰۳/۲۲ ۲۰:۲۰

سلام و تشکر به خاطر مطالب مفیدتان

شما پروژه ای از نوع workflow console application را به پروژه اضافه نموده اید. اگر من بخواهم در یک پروژه وب از wf استفاده کنم نیز باید پروژه ای از همین نوع را به solution برنامه اضافه نمایم؟  
در یکسری ویدیوهای آموزشی من پروژه هایی از نوع sequential و غیره دیده ام اما من که از vs 2013 استفاده می‌کنم این گزینه‌ها را ندارم . ممکن است راهنمایی بفرمایید؟

نویسنده: محمد جواد تواضعی

تاریخ: ۱۳۹۴/۰۳/۲۳ ۱:۰

می توانید پروژه wf را به صورت WCF WorkFlow Service Application در Solution مورد نظر اضافه کنید پس از ان سرویس را بر روی ویندوز سرور هاست کنید به کمک برنامه AppFabric که می‌توانید ان را از لینک زیر دانلود کنید .

<http://msdn.microsoft.com/appfabric>

روش دیگر این است که شما مستقیما از کلاس‌های WF در پروژه خود استفاده کنید و Activity های خود را تولید کنید بدون اینکه احتیاج به Model Designer داشته باشید مانند کد زیر:

```
namespace LeadGenerator
{
    public sealed class CreateLead : CodeActivity
    {
        public InArgument<string> ContactName { get; set; }
        public InArgument<string> ContactPhone { get; set; }
        public InArgument<string> Interests { get; set; }
        public InArgument<string> Notes { get; set; }
    }
}
```



```
public InArgument<string> ConnectionString { get; set; }
public OutArgument<Lead> Lead { get; set; }
protected override void Execute(CodeActivityContext context)
{
    // Create a Lead class and populate it with the input arguments
    Lead l = new Lead();
    l.ContactName = ContactName.Get(context);
    l.ContactPhone = ContactPhone.Get(context);
    l.Interests = Interests.Get(context);
    l.Comments = Notes.Get(context);
    l.WorkflowID = context.WorkflowInstanceId;
    l.Status = "Open";
    // Insert a record into the Lead table
    LeadDataDataContext dc =
    new LeadDataDataContext(ConnectionString.Get(context));
    dc.Leads.InsertOnSubmit(l);
    dc.SubmitChanges();
    // Store the request in the OutArgument
    Lead.Set(context, l);
}
}
```

## معرفی کتابخانه stateless به عنوان جایگزین سبک وزنی برای windows workflow foundation

کتابخانه سورس باز [Stateless](#) ، برای طراحی و پیاده سازی «ماشین‌های حالت گردش کاری مانند» تهیه شده و مزایای زیر را نسبت به windows workflow foundation دارا است:

- جمعا 30 کیلوبایت است!
- تمام اجزای آن سورس باز است.
- دارای API روان و ساده‌ای است.
- امکان تبدیل UML state diagrams، به نمونه معادل Stateless بسیار ساده و سریع است.
- به دلیل code first بودن، کار کردن با آن برای برنامه نویس‌ها ساده‌تر بوده و افزودن یا تغییر اجزای آن با کدنویسی به سادگی میسر است.

دریافت کتابخانه Stateless از [Google code](#) و یا از [NuGet](#)

## پیاده سازی مثال کلید برق با Stateless

در ادامه همان مثال ساده کلید برق قسمت قبل را با Stateless پیاده سازی خواهیم کرد:

```
using System;
using Stateless;

namespace StatelessTests
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                string on = "On", off = "Off";
                var space = ' ';

                var onOffSwitch = new StateMachine<string, char>(initialState: off);

                onOffSwitch.Configure(state: off).Permit(trigger: space, destinationState: on);
                onOffSwitch.Configure(state: on).Permit(trigger: space, destinationState: off);

                Console.WriteLine("Press <space> to toggle the switch. Any other key will raise an
error.");

                while (true)
                {
                    Console.WriteLine("Switch is in state: " + onOffSwitch.State);
                    var pressed = Console.ReadKey(true).KeyChar;
                    onOffSwitch.Fire(trigger: pressed);
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine("Exception: " + ex.Message);
                Console.WriteLine("Press any key to continue...");
                Console.ReadKey(true);
            }
        }
    }
}
```

کار با ایجاد یک وهله از ماشین حالت (new StateMachine) آغاز می‌شود. حالت آغازین آن (initialState) مطابق مثال قسمت قبل، مساوی off است.

امضای کلاس StateMachine را در ذیل مشاهده می‌کنید؛ جهت توضیح آرگومان‌های جنریک string و char معرفی شده در مثال:

```
public class StateMachine<TState, TTrigger>
```

که اولی بیانگر نوع حالات قابل تعریف است و دومی نوع رویداد قابل دریافت را مشخص می‌کند. برای مثال در اینجا حالات روشن و خاموش، با رشته‌های on و off مشخص شده‌اند و رویداد قابل قبول دریافتی، کاراکتر فاصله است. سپس نیاز است این ماشین حالت را برای معرفی رویدادهایی (trigger در اینجا) که سبب تغییر حالت آن می‌شوند، تنظیم کنیم. اینکار توسط متدهای Configure و Permit انجام خواهد شد. متد Configure، یکی از حالات از پیش تعیین شده را جهت تنظیم، مشخص می‌کند و سپس در متد Permit تعیین خواهیم کرد که بر اساس رخدادی مشخص (برای مثال در اینجا فشرده شدن کلید space) وضعیت حالت جاری، به وضعیت جدیدی (destinationState) منتقل شود. نهایتاً این ماشین حالت در یک حلقه بی‌نهایت مشغول به کار خواهد شد. برای نمونه یک Thread پس زمینه (BackgroundWorker) نیز می‌تواند همین کار را در برنامه‌های ویندوزی انجام دهد.

#### یک نکته

علاوه بر روش‌های یاد شده تشخیص الگوی ماشین حالت که [در قسمت قبل](#) بررسی شدند، مورد refactoring انبوهی از if و else ها و یا switch‌های بسیار طولانی را نیز می‌توان به این لیست افزود.

#### استفاده از Stateless Designer برای تولید کدهای ماشین حالت

کتابخانه Stateless دارای یک طراح و Code generator بصری سورس باز است که آن‌را به شکل افزونه‌ای برای VS.NET می‌توانید [در سایت Codeplex دریافت کنید](#). این طراح از [کتابخانه GLEE](#) برای رسم گراف استفاده می‌کند.

کار مقدماتی با آن به نحو زیر است:

الف) فایل StatelessDesignerPackage.vsix را از سایت کدپلکس دریافت و نصب کنید. البته نگارش فعلی آن فقط با VS 2012 سازگار است.

ب) ارجاعی را به اسمبلی stateless به پروژه خود اضافه نمائید (به یک پروژه جدید یا از پیش موجود).

ج) از منوی پروژه، گزینه Add new item را انتخاب کرده و سپس در صفحه ظاهر شده، گزینه جدید Stateless state machine را انتخاب و به پروژه اضافه نمائید.

کار با این طراح، با ادیت XML آن شروع می‌شود. برای مثال گردش کاری ارسال و تأیید یک مطلب جدید را در بلاگی فرضی، به نحو زیر وارد نمائید:

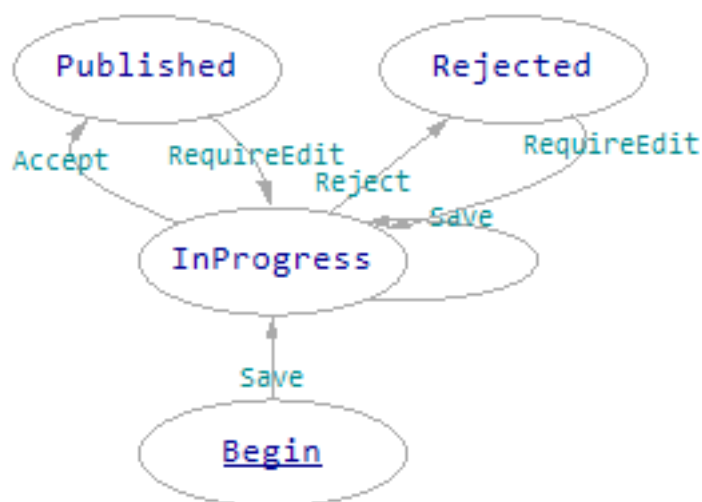
```
<statemachine xmlns="http://statelessdesigner.codeplex.com/Schema">
  <settings>
    <itemname>BlogPostStateMachine</itemname>
    <namespace>StatelessTests</namespace>
    <class>public</class>
  </settings>
  <triggers>
    <trigger>Save</trigger>
    <trigger>RequireEdit</trigger>
    <trigger>Accept</trigger>
    <trigger>Reject</trigger>
  </triggers>
  <states>
    <state start="yes">Begin</state>
    <state>InProgress</state>
    <state>Published</state>
    <state>Rejected</state>
  </states>
  <transitions>
    <transition trigger="Save" from="Begin" to="InProgress" />

    <transition trigger="Accept" from="InProgress" to="Published" />
    <transition trigger="Reject" from="InProgress" to="Rejected" />

    <transition trigger="Save" from="InProgress" to="InProgress" />
  </transitions>
</statemachine>
```

```
<transition trigger="RequireEdit" from="Published" to="InProgress" />
<transition trigger="RequireEdit" from="Rejected" to="InProgress" />
</transitions>
</statemachine>
```

حاصل آن گراف زیر خواهد بود:



به علاوه کدهای زیر که به صورت خودکار تولید شده‌اند:

```
using Stateless;

namespace StatelessTests
{
    public class BlogPostStateMachine
    {
        public delegate void UnhandledTriggerDelegate(State state, Trigger trigger);
        public delegate void EntryExitDelegate();
        public delegate bool GuardClauseDelegate();

        public enum Trigger
        {
            Save,
            RequireEdit,
            Accept,
            Reject,
        }

        public enum State
        {
            Begin,
            InProgress,
            Published,
            Rejected,
        }

        private readonly StateMachine<State, Trigger> stateMachine = null;

        public EntryExitDelegate OnBeginEntry = null;
        public EntryExitDelegate OnBeginExit = null;
        public EntryExitDelegate OnInProgressEntry = null;
        public EntryExitDelegate OnInProgressExit = null;
        public EntryExitDelegate OnPublishedEntry = null;
        public EntryExitDelegate OnPublishedExit = null;
        public EntryExitDelegate OnRejectedEntry = null;
        public EntryExitDelegate OnRejectedExit = null;
        public GuardClauseDelegate GuardClauseFromBeginToInProgressUsingTriggerSave = null;
    }
}
```

```

public GuardClauseDelegate GuardClauseFromInProgressToPublishedUsingTriggerAccept = null;
public GuardClauseDelegate GuardClauseFromInProgressToRejectedUsingTriggerReject = null;
public GuardClauseDelegate GuardClauseFromInProgressToInProgressUsingTriggerSave = null;
public GuardClauseDelegate GuardClauseFromPublishedToInProgressUsingTriggerRequireEdit = null;
public GuardClauseDelegate GuardClauseFromRejectedToInProgressUsingTriggerRequireEdit = null;
public UnhandledTriggerDelegate OnUnhandledTrigger = null;

public BlogPost()
{
    stateMachine = new StateMachine<State, Trigger>(State.Begin);
    stateMachine.Configure(State.Begin)
        .OnEntry(() => { if (OnBeginEntry != null) OnBeginEntry(); })
        .OnExit(() => { if (OnBeginExit != null) OnBeginExit(); })
        .PermitIf(Trigger.Save, State.InProgress, () => { if
(GuardClauseFromBeginToInProgressUsingTriggerSave != null) return
GuardClauseFromBeginToInProgressUsingTriggerSave(); return true; } )
        ;
    stateMachine.Configure(State.InProgress)
        .OnEntry(() => { if (OnInProgressEntry != null) OnInProgressEntry(); })
        .OnExit(() => { if (OnInProgressExit != null) OnInProgressExit(); })
        .PermitIf(Trigger.Accept, State.Published, () => { if
(GuardClauseFromInProgressToPublishedUsingTriggerAccept != null) return
GuardClauseFromInProgressToPublishedUsingTriggerAccept(); return true; } )
        .PermitIf(Trigger.Reject, State.Rejected, () => { if
(GuardClauseFromInProgressToRejectedUsingTriggerReject != null) return
GuardClauseFromInProgressToRejectedUsingTriggerReject(); return true; } )
        .PermitReentryIf(Trigger.Save, () => { if
(GuardClauseFromInProgressToInProgressUsingTriggerSave != null) return
GuardClauseFromInProgressToInProgressUsingTriggerSave(); return true; } )
        ;
    stateMachine.Configure(State.Published)
        .OnEntry(() => { if (OnPublishedEntry != null) OnPublishedEntry(); })
        .OnExit(() => { if (OnPublishedExit != null) OnPublishedExit(); })
        .PermitIf(Trigger.RequireEdit, State.InProgress, () => { if
(GuardClauseFromPublishedToInProgressUsingTriggerRequireEdit != null) return
GuardClauseFromPublishedToInProgressUsingTriggerRequireEdit(); return true; } )
        ;
    stateMachine.Configure(State.Rejected)
        .OnEntry(() => { if (OnRejectedEntry != null) OnRejectedEntry(); })
        .OnExit(() => { if (OnRejectedExit != null) OnRejectedExit(); })
        .PermitIf(Trigger.RequireEdit, State.InProgress, () => { if
(GuardClauseFromRejectedToInProgressUsingTriggerRequireEdit != null) return
GuardClauseFromRejectedToInProgressUsingTriggerRequireEdit(); return true; } )
        ;
    stateMachine.OnUnhandledTrigger((state, trigger) => { if (OnUnhandledTrigger != null)
OnUnhandledTrigger(state, trigger); });
}

public bool TryFireTrigger(Trigger trigger)
{
    if (!stateMachine.CanFire(trigger))
    {
        return false;
    }
    stateMachine.Fire(trigger);
    return true;
}

public State GetState
{
    get
    {
        return stateMachine.State;
    }
}
}

```

## توضیحات:

ماشین حالت فوق دارای چهار حالت شروع، در حال بررسی، منتشر شده و رد شده است. معمول است که این چهار حالت را به شکل یک enum معرفی کنند که در کدهای تولیدی فوق نیز به همین نحو عمل گردیده و public enum State معرف چهار حالت ذکر شده است. همچنین رویدادهای ذخیره، نیاز به ویرایش، ویرایش، تأیید و رد نیز توسط public enum Trigger معرفی شده‌اند. در قسمت Transitions، بر اساس یک رویداد (Trigger در اینجا)، انتقال از یک حالت به حالتی دیگر را سبب خواهیم شد. تعاریف اصلی تنظیمات ماشین حالت، در سازنده کلاس BlogPostStateMachine انجام شده است. این تعاریف نیز بسیار ساده

هستند. به ازای هر حالت، یک Configure داریم. در متدهای OnEntry و OnExit هر حالت، یک سری callback function فراخوانی خواهند شد. برای مثال در حالت Rejected یا Approved می‌توان ایمیلی را به ارسال کننده مطلب جهت یادآوری وضعیت رخ داده، ارسال نمود.

متدهای PermitIf سبب انتقال شرطی، به حالتی دیگر خواهند شد. برای مثال رد یا تأیید یک مطلب نیاز به دسترسی مدیریتی خواهد داشت. این نوع موارد را توسط Guardهای Guard delegate می‌توان مدیریت شرطها ایجاد کرده است، می‌توان تنظیم کرد. PermitReentryIf سبب بازگشت مجدد به همان حالت می‌گردد. برای مثال ویرایش و ذخیره یک مطلب در حال انتشار، سبب تأیید یا رد آن نخواهد شد؛ صرفاً عملیات ذخیره صورت گرفته و ماشین حالت مجدداً در همان مرحله باقی خواهد ماند.

### نحوه استفاده از ماشین حالت تولیدی:

همانطور که عنوان شد، حداقل استفاده از ماشین‌های حالت، refactoring انبوهی از if و elseها است که در حالت مدیریت یک چنین گردش‌های کاری باید تدارک دید.

```
namespace StatelessTests
{
    public class BlogPostManager
    {
        private BlogPostStateMachine _stateMachine;
        public BlogPostManager()
        {
            configureWorkflow();
        }

        private void configureWorkflow()
        {
            _stateMachine = new BlogPostStateMachine();

            _stateMachine.GuardClauseFromBeginToInProgressUsingTriggerSave = () => { return
UserCanPost; };
            _stateMachine.OnBeginExit = () => { /* save data + save state + send an email to admin */
};

            _stateMachine.GuardClauseFromInProgressToPublishedUsingTriggerAccept = () => { return
UserIsAdmin; };
            _stateMachine.GuardClauseFromInProgressToRejectedUsingTriggerReject = () => { return
UserIsAdmin; };
            _stateMachine.GuardClauseFromInProgressToInProgressUsingTriggerSave = () => { return
UserHasEditRights; };
            _stateMachine.OnInProgressExit = () => { /* save data + save state + send an email to user
*/ };

            _stateMachine.OnPublishedExit = () => { /* save data + save state + send an email to admin
*/ };
            _stateMachine.GuardClauseFromPublishedToInProgressUsingTriggerRequireEdit = () => { return
UserHasEditRights; };

            _stateMachine.OnRejectedExit = () => { /* save data + save state + send an email to admin
*/ };
            _stateMachine.GuardClauseFromRejectedToInProgressUsingTriggerRequireEdit = () => { return
UserHasEditRights; };
        }

        public bool UserIsAdmin
        {
            get
            {
                return true; // TODO: Evaluate if user is an admin.
            }
        }

        public bool UserCanPost
        {
            get
            {
                return true; // TODO: Evaluate if user is authenticated
            }
        }

        public bool UserHasEditRights
        {
            get
            {
                return true; // TODO: Evaluate if user is owner or admin
            }
        }
    }
}
```

```

    }

    // User actions
    public void Save() { _stateMachine.TryFireTrigger(BlogPostStateMachine.Trigger.Save); }
    public void RequireEdit() {
        _stateMachine.TryFireTrigger(BlogPostStateMachine.Trigger.RequireEdit); }

    // Admin actions
    public void Accept() { _stateMachine.TryFireTrigger(BlogPostStateMachine.Trigger.Accept); }
    public void Reject() { _stateMachine.TryFireTrigger(BlogPostStateMachine.Trigger.Reject); }
}

```

در کلاس فوق، نحوه استفاده از ماشین حالت تولیدی را مشاهده می‌کنید. در `Guard`های `delegate`، سطوح دسترسی انجام عملیات بررسی خواهند شد. برای مثال، از بانک اطلاعاتی بر اساس اطلاعات کاربر جاری وارد شده به سیستم اخذ می‌گردند. در متدهای `Exit` هر مرحله، کارهای ذخیره سازی اطلاعات در بانک اطلاعاتی، ذخیره سازی حالت (مثلا در یک فیلد که بعدا قابل بازیابی باشد) صورت می‌گیرد و در صورت نیاز ایمیلی به اشخاص مختلف ارسال خواهد شد. برای به حرکت درآوردن این ماشین، نیاز به یک سری اکشن متد نیز می‌باشد. تعدادی از این موارد را در انتهای کلاس فوق ملاحظه می‌کنید. کد نویسی آن‌ها در حد فراخوانی متد `TryFireTrigger` ماشین حالت است.

#### یک نکته:

ماشین حالت تولیدی به صورت پیش فرض در حالت `State.Begin` قرار دارد. می‌توان این مورد را از بانک اطلاعاتی خواند و سپس مقدار دهی نمود تا با هر بار وهله سازی ماشین حالت دقیقاً مشخص باشد که در چه مرحله‌ای قرار داریم و `TryFireTrigger` بتواند بر این اساس تصمیم‌گیری کند که آیا مجاز است عملیاتی را انجام دهد یا خیر.

## نظرات خوانندگان

نویسنده: برنامه نویسی  
تاریخ: ۱۰:۲۵ ۱۳۹۱/۱۰/۱۴

استفاده از یک Dictionary از نوع string و Action، چه مشکلی نسبت به این روش دارد؟

نویسنده: وحید نصیری  
تاریخ: ۱۰:۴۶ ۱۳۹۱/۱۰/۱۴

مشکلی ندارد. شما در هر زمانی می‌تونید دست به [اختراع مجدد](#) چرخ بزنید. با یک Dictionary از نوع string و Action فقط قسمت حالات و رویدادها رو طراحی کردید. مابقی قسمت‌ها مانند انتقال‌ها رو هم که اضافه کنید می‌شود کتابخانه Stateless.

نویسنده: امیر هاشم زاده  
تاریخ: ۲۳:۲۶ ۱۳۹۳/۰۶/۱۶

طبق کد زیر:

```
_stateMachine.OnPublishedExit = () => { /* save data + save state + send an email to admin */ };  
_stateMachine.GuardClauseFromPublishedToInProgressUsingTriggerRequireEdit = () => { return  
UserHasEditRights };
```

آیا درست متوجه شدم که: باز هم ابتدا سطح دسترسی بررسی می‌شود و سپس عملیات ذخیره سازی صورت می‌پذیرد؟!

نویسنده: وحید نصیری  
تاریخ: ۲۳:۴۲ ۱۳۹۳/۰۶/۱۶

این‌ها فقط یک سری تنظیم اولیه سیستم هستند. مهم نیست ترتیب معرفی آن‌ها به چه صورتی است. اجرای آن‌ها اینجا انجام نمی‌شود.

نویسنده: امیر هاشم زاده  
تاریخ: ۰:۱۶ ۱۳۹۳/۰۶/۱۷

ظاهراً در کلاس BlogPostStateMachine، متد سازنده آن به اشتباه BlogPost درج شده است.  
برای تغییر حالت و مقدار دهی آن از بانک اطلاعاتی، باید کد کلاس BlogPostStateMachine را مانند زیر تغییر دهیم:

```
public BlogPostStateMachine(State state)  
{  
    stateMachine = new StateMachine<State, Trigger>(state);
```

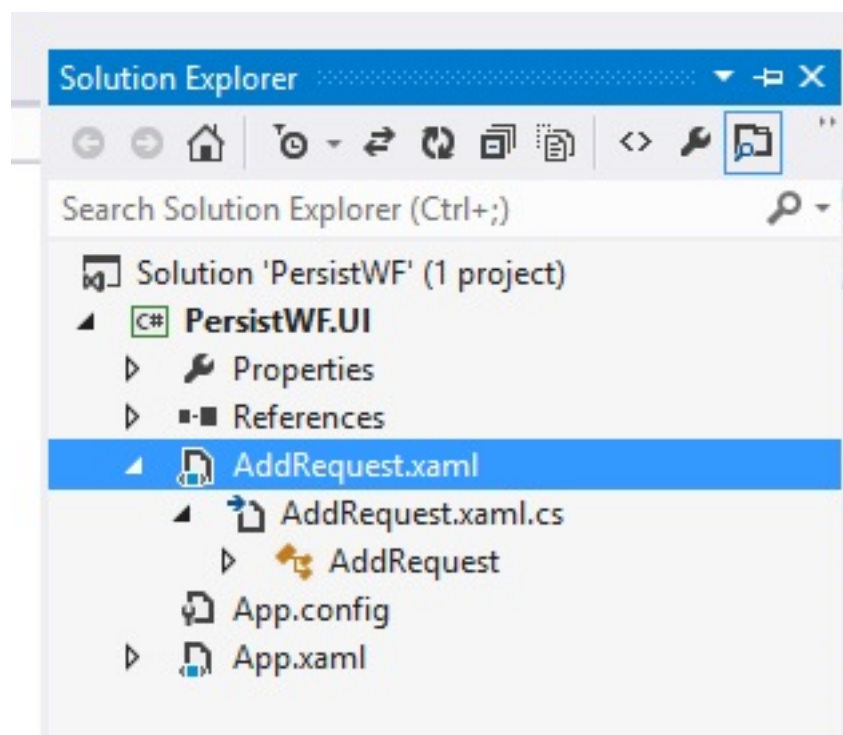


در خیلی از مواقع workflow ها به مرحله‌ای می‌رسند که احتیاج به دستوری از بیرون از فرآیند دارند. در هنگام انتظار، اگر به هر دلیلی workflow از حافظه حذف شود، امکان ادامه فرآیند وجود ندارد. اما می‌توان با Persist (ذخیره) کردن آن، در زمان انتظار و فراخوانی مجدد آن در هنگام نیاز، این ریسک را برطرف نمود.

قصد دارم با این مثال، طریقه persist شدن یک workflow در زمانیکه نیاز به انتظار برای تایید دارد و فراخوانی آن از همان نقطه پس از تایید مربوطه را توضیح دهم.

ساختار اینترفیس کاربری ما WPF می‌باشد. پس در ابتدا یک پروژه از نوع WPF ایجاد می‌کنیم. اسم solution را PersistWF و اسم Project را PersistWF.UI انتخاب می‌کنیم.

در پروژه UI نام فایل MainWindow.xaml را به AddRequest.xaml تغییر می‌دهیم. همچنین اسم کلاس مربوطه را در codebehind



همین طور مقدار StartupUri را هم در app.xaml اصلاح می‌کنیم

```
StartupUri="AddRequest.xaml"
```

Reference های زیر رو هم به پروژه اضافه می‌کنیم

```

•System.Activities
•System.Activities.DurableInstancing
•System.Configuration
•System.Data.Linq
•System.Runtime.DurableInstancing
•System.ServiceModel
•System.ServiceModel.Activities
•System.Workflow.ComponentModel
•System.Runtime.DurableInstancing
•System.Activities.DurableInstancing

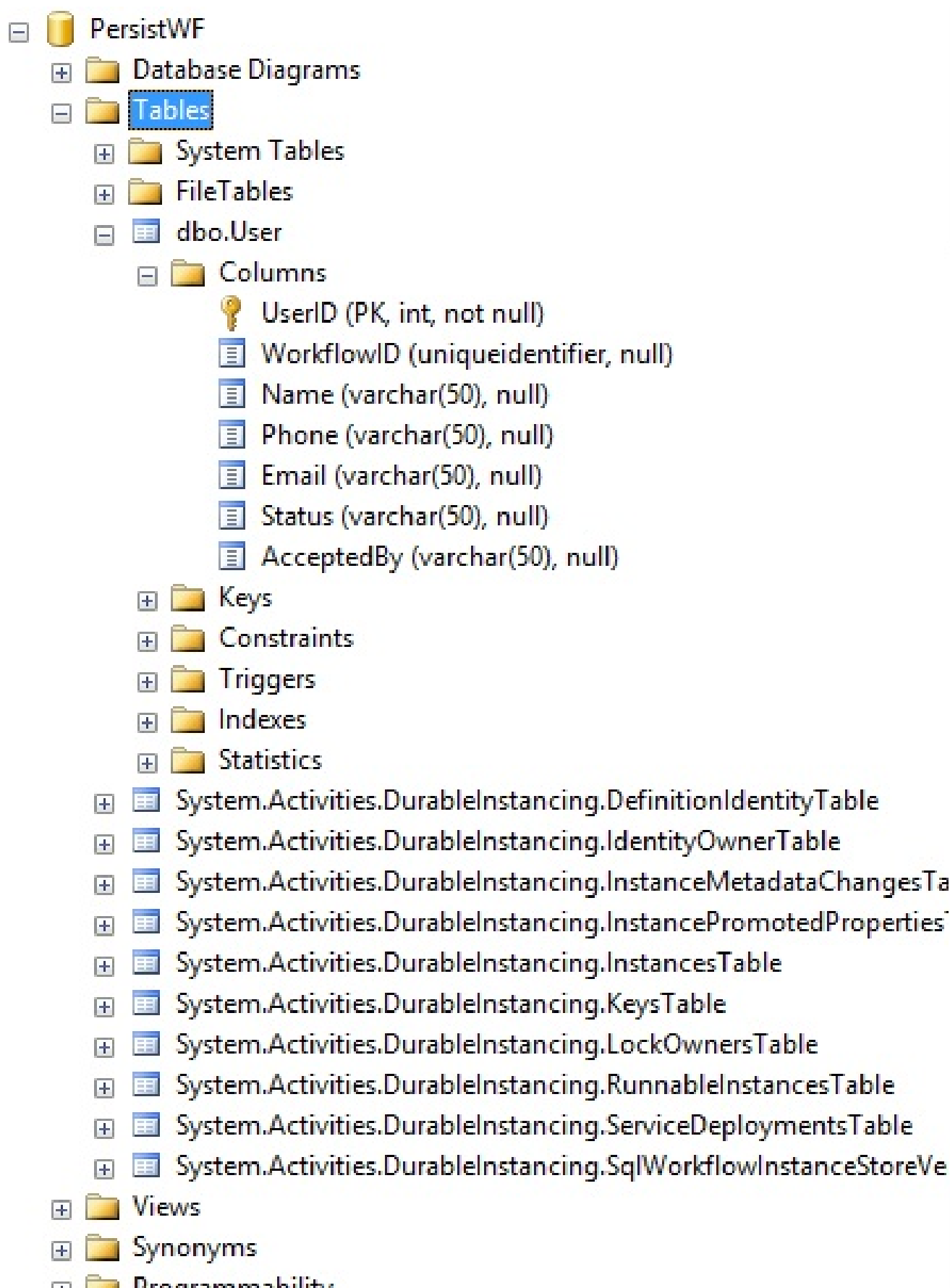
```

قرار است کاربری ثبت نام کند، در فرایند ثبت، منتظر تایید یکی از مدیران قرار می‌گیرد. مدیر، لیست کاربران جدید را می‌بینید، یک کاربر را انتخاب می‌کند؛ مقادیر لازم را وارد می‌کند و سپس پروسه تایید را انجام می‌دهد که فراخوانی فرآیند مربوطه از همان قسمتی است که منتظر تایید مانده است.

برای Persist کردن workflow از کلاس SqlWorkflowInstanceStore استفاده می‌کنم. این شی به connection ای به یک دیتابیس با یک ساختار معین احتیاج دارد. خوشبختانه اسکریپت‌های مورد نیاز این ساختار در پوشه  
 [en]\SQL\Microsoft.NET\Framework\v4.0.30319\Drive]:\Windows\Microsoft.NET\Framework\v4.0.30319\SQL\en]  
 وجود دارند. دو اسکریپت با نام‌های SqlWorkflowInstanceStoreSchema و SqlWorkflowInstanceStoreLogic باید به ترتیب در دیتابیس اجرا شوند.

من یک دیتابیس با نام PersistWF ایجاد می‌کنم و اسکریپت‌ها را بر روی آن اجرا می‌کنم. یک جدول هم برای نگهداری کاربران ثبت شده در همین دیتابیس ایجاد می‌کنم.

و شمایل دیتابیس ما پس از اجرا کردن اسکریپت‌ها و ساختن جدول User بدین شکل است:



XAML زیر، ساختار فرم AddRequest می‌باشد که قرار است نقش UI برنامه را ایفا کند. آن را با XAML های پیش فرض عوض کنید.

```

<Window x:Class="PersistWF.UI.AddRequest"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="520" Width="550" Loaded="Window_Loaded">
    <Grid MinWidth="300" MinHeight="100" Width="514">
        <Label Height="30" Margin="5,10,10,10" Name="lblName" VerticalAlignment="Top"
            HorizontalAlignment="Left" Width="90" HorizontalContentAlignment="Right">Name:</Label>
        <Label Height="30" Margin="270,10,10,10" Name="lblPhone" VerticalAlignment="Top"
            HorizontalAlignment="Left" Width="90" HorizontalContentAlignment="Right">Phone Number:</Label>
        <Label Height="30" Margin="5,40,10,10" Name="lblEmail" VerticalAlignment="Top"
            HorizontalAlignment="Left" Width="90" HorizontalContentAlignment="Right">Email:</Label>
        <TextBox Height="25" Margin="100,10,10,10" Name="txtName" VerticalAlignment="Top"
            HorizontalAlignment="Left" Width="170" />
        <TextBox Height="25" Margin="365,10,10,10" Name="txtPhone" VerticalAlignment="Top"
            HorizontalAlignment="Left" Width="100" />
        <TextBox Height="25" Margin="100,40,10,10" Name="txtEmail" VerticalAlignment="Top"
            HorizontalAlignment="Left" Width="300" />
        <Button Height="23" Margin="100,86,0,0" Name="brnRegister" VerticalAlignment="Top"
            HorizontalAlignment="Left" Width="70" Click="brnRegister_Click">Register</Button>
        <ListView x:Name="lstUsers" Margin="10,125,10,10" Height="145" VerticalAlignment="Top"
            ItemsSource="{Binding}" HorizontalContentAlignment="Center"
            SelectionChanged="lstUsers_SelectionChanged" >
            <ListView.View>
                <GridView>
                    <GridViewColumn Header="Current User" Width="480">
                        <GridViewColumn.CellTemplate>
                            <DataTemplate>
                                <StackPanel Orientation="Horizontal">
                                    <TextBlock Text="{Binding Name}" Width="110"/>
                                    <TextBlock Text="{Binding Phone}" Width="70"/>
                                    <TextBlock Text="{Binding Email}" Width="130"/>
                                    <TextBlock Text="{Binding Status}" Width="70"/>
                                    <TextBlock Text="{Binding AcceptedBy}" Width="100"/>
                                </StackPanel>
                            </DataTemplate>
                        </GridViewColumn.CellTemplate>
                    </GridViewColumn>
                </GridView>
            </ListView.View>
        </ListView>
        <Label Height="37" HorizontalAlignment="Stretch" Margin="10,272,5,10" Name="lblSelectedNotes"
            VerticalAlignment="Top" Visibility="Hidden" />
        <Label Height="30" Margin="10,0,0,140" Name="lblAgent" VerticalAlignment="Bottom"
            HorizontalAlignment="Left" Width="40" HorizontalContentAlignment="Left" Visibility="Hidden">Admin
            Name:</Label>
        <TextBox Height="25" Margin="60,0,0,140" Name="txtAcceptedBy" VerticalAlignment="Bottom"
            HorizontalAlignment="Left" Width="190" Visibility="Hidden" />
        <Button Height="25" Margin="270,0,0,140" Name="btnAccept" VerticalAlignment="Bottom"
            HorizontalAlignment="Left" Width="90" Click="btnAccept_Click" Visibility="Hidden">Accept</Button>
        <Label Height="27" HorizontalAlignment="Left" Margin="10,0,0,110" Name="lblEvent"
            VerticalAlignment="Bottom" Width="76">Event Log</Label>
        <ListBox Margin="12,0,5,12" Name="lstEvents" Height="100" VerticalAlignment="Bottom"
            FontStretch="Condensed" FontSize="10" />
    </Grid>
</Window>

```

اگر همه چیز مرتب باشد؛ ساختار فرم شما باید به این شکل باشد

The screenshot shows a WPF application window titled "MainWindow". Inside the window, there is a registration form. At the top, there are two input fields: "Name:" and "Phone Number". Below these are "Email:" and "Notes:" fields. To the right of the "Notes:" field is a "Register" button. Below the registration fields is a section titled "Current User" which is currently empty. At the bottom of the window is a section titled "Event Log", also currently empty.

اکثر workflow ها از activity معروف Wrteline استفاده می کنند که برای نمایش یک رشته به کار می رود. ما هم در workflow مثالمان از این Activity استفاده می کنیم. اما برای اینکه مقادیری که توسط این Activity ایجاد می شوند در کادر event log فرم خودمان نمایش داده شود؛ احتیاج داریم که یک TextWriter سفارشی برای خودمان ایجاد کنیم. اما قبل از آن یک کلاس static در پروژه ایجاد می کنیم که بتوانیم در هر قسمتی، به فرم دسترسی داشته باشیم.

کلاسی را با نام ApplicationInterface به پروژه اضافه کرده و یک Property استاتیک از جنس فرم AddRequest هم برای آن تعریف می کنیم:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PersistWF.UI
{
    public static class ApplicationInterface
    {
```

```

    public static AddRequest _app { get; set; }
}

```

به Constructor کلاس موجود در فایل AddRequest.xaml.cs این خط کد رو اضافه می‌کنم

```

public AddRequest()
{
    InitializeComponent();
    ApplicationInterface._app = this;
}

```

این دو متد را هم به این کلاس اضافه می‌کنیم

```

private void AddEvent(string szText)
{
    lstEvents.Items.Add(szText);
}
public ListBox GetEventListBox()
{
    return this.lstEvents;
}

```

متد اول برای اضافه کردن یک event Log و متد دوم هم که کنسول لاگ را در اختیار درخواست کننده‌اش قرار می‌دهد.

و حالا کلاس TextWriter سفارشی‌امان را می‌نویسیم. یک کلاس به نام ListBoxTextWriter به پروژه اضافه می‌کنیم که از TextWriter مشتق می‌شود و محتویات آنرا در زیر می‌بینید:

```

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Controls;

namespace PersistWF.UI
{
    public class ListBoxTextWriter : TextWriter
    {
        const string textClosed = "This TextWriter must be opened before use";
        private Encoding _encoding;
        private bool _isOpen = false;
        private ListBox _listBox;
        public ListBoxTextWriter()
        {
            // Get the static list box
            _listBox = ApplicationInterface._app.GetEventListBox();
            if (_listBox != null)
                _isOpen = true;
        }
        public ListBoxTextWriter(ListBox listBox)
        {
            this._listBox = listBox;
            this._isOpen = true;
        }
        public override Encoding Encoding
        {
            get
            {
                if (_encoding == null)
                {
                    _encoding = new UnicodeEncoding(false, false);
                }
                return _encoding;
            }
        }
    }
}

```

```

    public override void Close()
    {
        this.Dispose(true);
    }
    protected override void Dispose(bool disposing)
    {
        this._isOpen = false;
        base.Dispose(disposing);
    }
    public override void Write(char value)
    {
        if (!this._isOpen)
            throw new ApplicationException(textClosed); ;
        this._listBox.Dispatcher.BeginInvoke(new Action(() =>
this._listBox.Items.Add(value.ToString())));
    }
    public override void Write(string value)
    {
        if (!this._isOpen)
            throw new ApplicationException(textClosed);
        if (value != null)
            this._listBox.Dispatcher.BeginInvoke(new Action(() =>
this._listBox.Items.Add(value)));
    }
    public override void Write(char[] buffer, int index, int count)
    {
        String toAdd = "";
        if (!this._isOpen)
            throw new ApplicationException(textClosed); ;
        if (buffer == null || index < 0 || count < 0)
            throw new ArgumentOutOfRangeException("buffer");
        if ((buffer.Length - index) < count)
            throw new ArgumentException("The buffer is too small");
        for (int i = 0; i < count; i++)
            toAdd += buffer[i];
        this._listBox.Dispatcher.BeginInvoke(new Action(() => this._listBox.Items.Add(toAdd)));
    }
}
}
}

```

همان طور که می بینید کلاس ListBoxTextWriter از کلاس abstract TextWriter مشتق شده و پیاده سازی از متد Write را فراهم می کند تا یک رشته را به کنترل ListBox اضافه کند. (البته سه تا از این متدها را Override می کنیم تا بتوانیم یک رشته، یک کاراکتر و یا آرایه ای از کاراکترها را به ListBox اضافه کنیم) در constructor پیشفرض از کلاس ApplicationInterface استفاده کردیم تا بتوانیم کنترل lstEvents را از فرم اصلی برنامه به دست بیاوریم. برای Add کردن از Dispatcher و متد BeginInvoke مرتبط با آن استفاده کردیم. این کار، متد را قادر می سازد حتی وقتی که از یک thread متفاوت فراخوانی می شود، کار کند.

حالا می توانیم از این کلاس، به عنوان مقدار خاصیت TextWriter برای WriteLine استفاده کنیم.

به کلاس ApplicationInterface برگردیم تا متد زیر را هم به آن اضافه کنیم

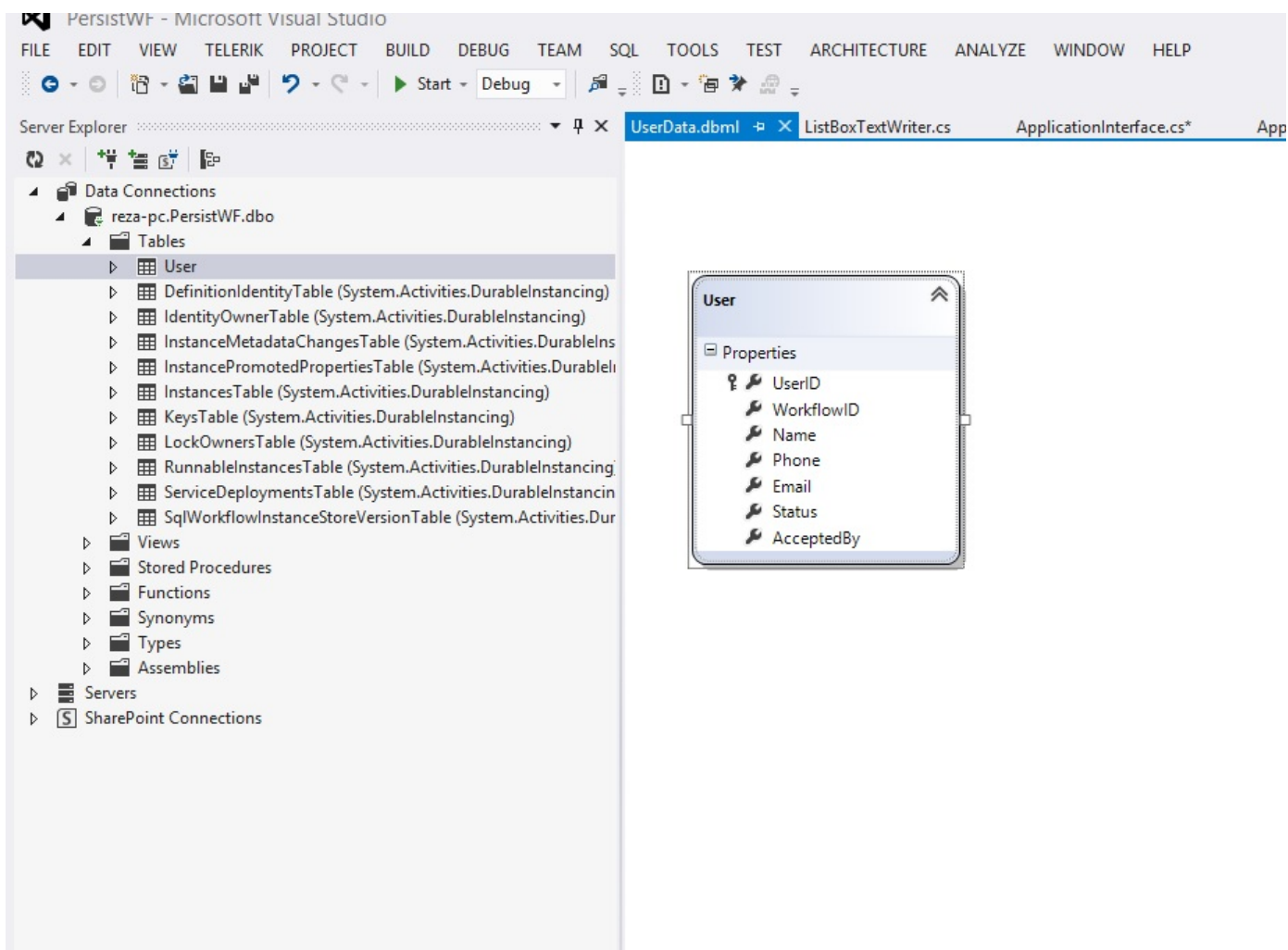
```

public static void AddEvent(String status)
{
    if (_app != null)
    {
        new ListBoxTextWriter(_app.GetEventListBox()).WriteLine(status);
    }
}

```

این هم از constructor دومی استفاده می کنه برای معرفی ListBox.

برای ارتباط با دیتابیس از LINQ to SQL استفاده می کنیم تا User رو ذخیره و بازیابی کنیم. به پروژه یک آیتم از نوع LINQ to SQL با نام UserData.dbml اضافه می کنیم. به دیتابیس متصل شده و جدول User رو به محیط Design می کشیم. در ادامه برای شی کلاس SQLWorkflowInstanceStore هم از همین Connectionstring استفاده می کنیم.

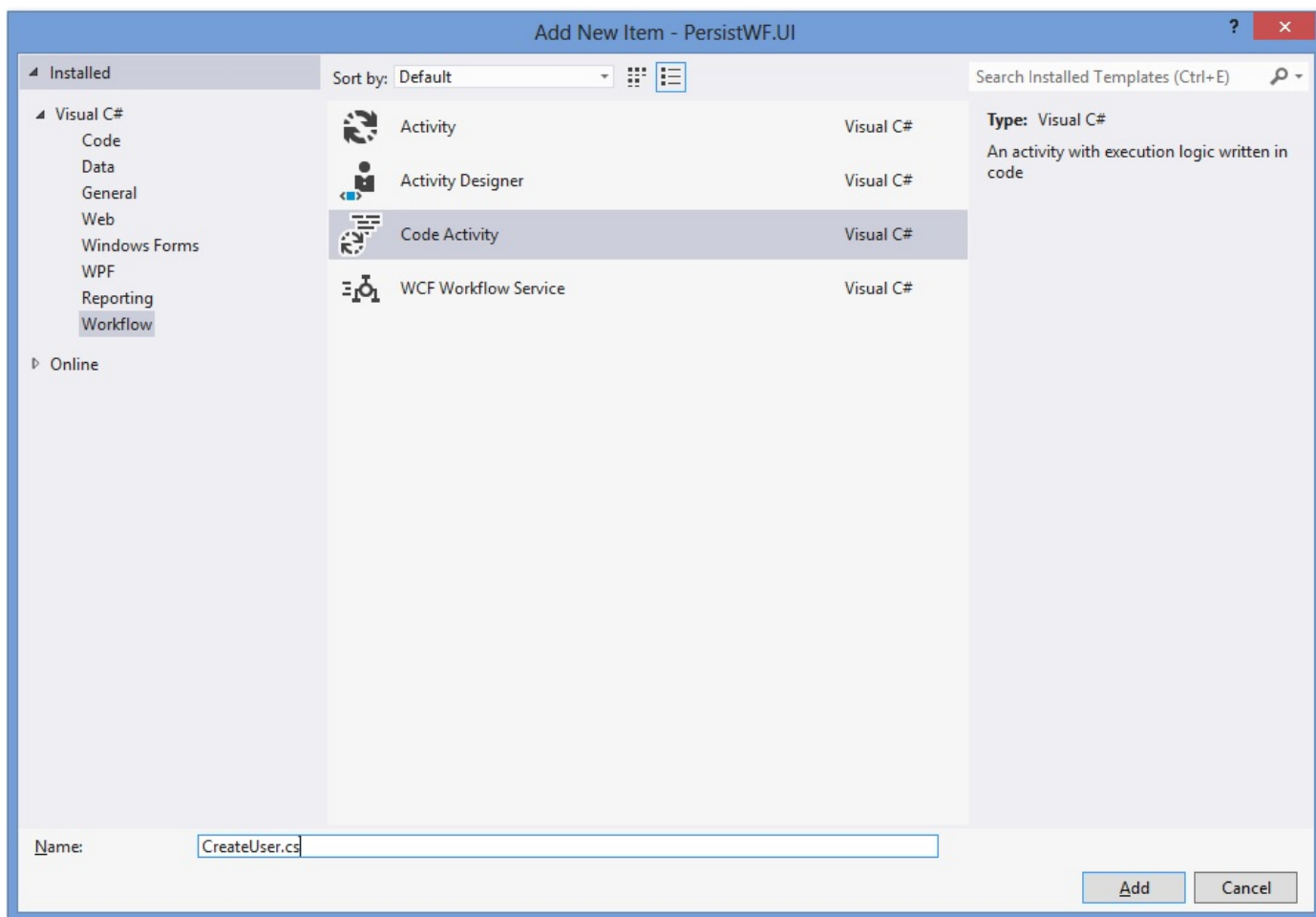


برای ایجاد workflow مورد نظر، به دو Activity سفارشی احتیاج داریم که باید خودمان ایجاد نماییم. یک پوشه با نام Activities به پروژه اضافه می‌کنم تا کلاس‌های مورد نظر را آنجا ایجاد کنیم.

#### 1. یک Activity برای ایجاد User

این Activity تعدادی پارامتر از نوع InArgument دارد که توسط آن‌ها یک Instance از کلاس User ایجاد می‌کند و در حقیقت آن را به دیتابیس می‌فرستد و ذخیره می‌کند. Connectionstring را هم می‌شود توسط یک آرگومان ورودی دیگر مقدار دهی کرد. یک آرگومان خروجی هم برای این Activity در نظر می‌گیریم تا User ایجاد شده را برگردانیم. روی پوشه‌ی Activities کلیک راست می‌کنیم و Add - NewItem را انتخاب می‌کنیم. از لیست workflow ها Template مربوط به CodeActivity را انتخاب کرده و یک CodeActivity با نام CreateUser ایجاد می‌کنیم





محتویات این کلاس را هم مانند زیر کامل می‌کنیم

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Activities;

namespace PersistWF.UI.Activities
{
    public sealed class CreateUser : CodeActivity
    {
        public InArgument<string> Name { get; set; }
        public InArgument<string> Email { get; set; }
        public InArgument<string> Phone { get; set; }
        public InArgument<string> ConnectionString { get; set; }

        public OutArgument<User> User { get; set; }

        protected override void Execute(CodeActivityContext context)
        {
            // ایجاد کاربر
            User user = new User();
            user.Email = Email.Get(context);
            user.Name = Name.Get(context);
            user.Phone = Phone.Get(context);
            user.Status = "New";
            user.WorkflowID = context.WorkflowInstanceId;
            UserDataDataContext db = new UserDataDataContext(ConnectionString.Get(context));
            db.Users.InsertOnSubmit(user);
            db.SubmitChanges();
        }
    }
}
```

```

        User.Set(context, user);
    }
}

```

متد Execute ، توسط مقادیری که به عنوان پارامتر دریافت شده، یک شی از کلاس User ایجاد می‌کند و به کمک DataContext آنرا در دیتابیس ذخیره کرده و در آخر User ذخیره شده را در اختیار پارامتر خروجی قرار می‌دهد.

### 1. یک Activity برای انتظار دریافت تایید

این Activity قرار است Workflow را Idle کند تا زمانیکه مدیر دستور تایید را با فراخوانی مجدد workflow از این همین قسمت صادر نماید.

این Activity باید از NativeActivity مشتق شده و برای اینکه workflow را وادار به معلق شدن کند کافی‌است خاصیت CanInduceIdle را با مقدار برگشتی override , true کنیم.

مثل قسمت قبل یک CodeActivity ایجاد می‌کنیم. اینبار با نام WaitForAccept که محتویاتش را هم مانند زیر تغییر می‌دهیم.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Activities;
using System.Workflow.ComponentModel;

namespace PersistWF.UI.Activities
{
    public sealed class WaitForAccept<T> : NativeActivity<T>
    {
        public WaitForAccept()
            :base()
        {
        }
        public string BookmarkName { get; set; }
        public OutArgument<T> Input { get; set; }

        protected override void Execute(NativeActivityContext context)
        {
            context.CreateBookmark(BookmarkName, new BookmarkCallback(this.Continue));
        }

        private void Continue(NativeActivityContext context, Bookmark bookmark, object value)
        {
            Input.Set(context, (T)value);
        }
        protected override bool CanInduceIdle
        {
            get
            {
                return true;
            }
        }
    }
}

```

این کلاس را generic نوشتم تا به جای User بشود هر پارامتر دیگه‌ای را به آن ارسال کرد. در واقع وقتی workflow به این Activity می‌رسد، Idle می‌شود. این activity یک bookmark هم ایجاد می‌کند. ما وقتی workflow را با این bookmark فراخوانی کنیم؛ workflow از همینجا ادامه می‌یابد. فراخوانی bookmark می‌تواند همراه با وارد کردن یک object باشد. متد Continue آن object را به آرگومان خروجی می‌دهد تا مسیر workflow را طی کند. ما User هایی را که به این نقطه رسیدن نمایش می‌دهیم. مدیر اونها را دیده و با مقدار دهی فیلد AcceptedBy، آن User را از اینجا

به workflow می‌فرستد و ما user وارد شده را در ادامه‌ی فرآیند Accept می‌کنیم.

برای ایجاد workflow هم می‌توانید از designer استفاده کنید و هم می‌توانید کد مربوط به workflow را پیاده سازی کنید.

برای پیاده سازی از طریق کد، یک کلاس با نام UserWF ایجاد می‌کنیم و محتویات workflow را مانند زیر پیاده سازی خواهیم کرد:



```
using PersistWF.UI.Activities;
using System;
using System.Activities;
using System.Activities.Statements;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


namespace PersistWF.UI
{
    public sealed class UserWF : Activity
    {
        public InArgument<string> Name { get; set; }
        public InArgument<string> Email { get; set; }
        public InArgument<string> Phone { get; set; }
        public InArgument<string> ConnectionString { get; set; }
        public InArgument<TextWriter> Writer { get; set; }


        public UserWF()
        {
            Variable<User> User = new Variable<User> { Name = "User" };
            this.Implementation = () => new Sequence
            {
                DisplayName = "EnterUser",
                Variables = { User },
                Activities = {
                    new CreateUser // 1. ایجاد کاربر با ورود پارامترهای ورودی
                    {
                        ConnectionString = new InArgument<string>(c => ConnectionString.Get(c)),
                        Email = new InArgument<string>(c => Email.Get(c)),
                        Name = new InArgument<string>(c => Name.Get(c)),
                        Phone = new InArgument<string>(c => Phone.Get(c)),
                        User = new OutArgument<User>(c => User.Get(c))
                    },
                    new WriteLine // 2. لاگ مربوط به ذخیره کاربر
                    {
                        TextWriter = new InArgument<TextWriter>(c => Writer.Get(c)),
                        Text = new InArgument<string>(c => string.Format("User {0} Registered and waiting for Accept", Name.Get(c) ) )
                    },
                    new InvokeMethod
                    {
                        TargetType = typeof(ApplicationInterface), // 3. برای به روزرسانی لیست کاربران ثبت شده در نمایش فرم
                        MethodName = "NewUser",
                        Parameters =
                        {
                            new InArgument<User>(env => User.Get(env))
                        }
                    },
                    new WaitForAccept<User> // 4. اینجا فرایند متوقف می‌شود و منتظر تایید مدیر می‌ماند
                    {
                        BookmarkName = "GetAcceptes",
                        Input = new OutArgument<User>(env => User.Get(env))
                    },
                    new WriteLine // 5. لاگ مربوط به تایید شدن کاربر
                    {
                        TextWriter = new InArgument<TextWriter>(c => Writer.Get(c)),
                        Text = new InArgument<string>(c => string.Format("User {0} Acceptor by {1}", Name.Get(c), User.Get(c).AcceptedBy))
                    }
                }
            };
        }
    }
}
```


```
}
```


اگر بخواهیم از Designer استفاده کنیم. فرایندمان چیزی شبیه شکل زیر خواهد بود

 Sequence 






 CreateUser



 WriteLine

Text





 InvokeMethod 


TargetType


TargetObject

MethodName




 WaitForAccept<User>



 WriteLine

Text



۵۳/۵۹

به Application بر می‌گردیم تا آن را پیاده سازی کنیم. ابتدا به app.config که اتوماتیک ایجاد شده رفته تا اسم ConnectionString رو به UserGenerator تغییر دهیم. محتویات درون app.config به شکل زیر است.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
  </configSections>
  <connectionStrings>
    <add name="UserGenerator"
      connectionString="Data Source=.;Initial Catalog=PersistWF;Integrated Security=True"
      providerName="System.Data.SqlClient" />
  </connectionStrings>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
</configuration>
```

در کلاس AddRequest کد زیر را اضافه می‌کنم. برای نگهداری مقدار connectionString

```
private string _connectionString = "";
```

همچنین کدهای زیر را به رویداد Load فرم اضافه می‌کنم تا مقدار connectionString را از Config بخوانم:

```
Configuration config = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);
ConnectionStringSection css =
(ConfigurationStringsSection)config.GetSection("connectionStrings");
_connectionString = css.ConnectionStrings["UserGenerator"].ConnectionString;
```

خط زیر را هم به کلاس AddRequest اضافه نمایید.

```
private InstanceStore _instanceStore;
```

این ارجاعیه به کلاس InstanceStore که برای Persist و Load کردن workflow از آن استفاده می‌کنیم و کدهای زیر را هم به رویداد Load فرم اضافه می‌کنیم.

```
_instanceStore = new SqlWorkflowInstanceStore(_connectionString);
InstanceView view = _instanceStore.Execute(_instanceStore.CreateInstanceHandle(), new
CreateWorkflowOwnerCommand(), TimeSpan.FromSeconds(30));
_instanceStore.DefaultInstanceOwner = view.InstanceOwner;
```

InstanceStore یک کلاس abstract می باشد که همه‌ی Provider های مربوط به persistence از آن مشتق می‌شوند. در این پروژه من از کلاس SqlWorkflowInstanceStore استفاده کردم تا workflow ها را در دیتابیس SQL Server ذخیره کنم.

برای ایجاد یک Request مقادیر را از فرم دریافت کرده، یک User ایجاد می‌کنیم و آن را در فرآیند به جریان می‌اندازیم. این کار را در رویداد کلیک دکمه Register انجام می‌دهیم

```
private void btnRegister_Click(object sender, RoutedEventArgs e)
{
    Dictionary<string, object> parameters = new Dictionary<string, object>();
    parameters.Add("Name", txtName.Text);
    parameters.Add("Phone", txtPhone.Text);
    parameters.Add("Email", txtEmail.Text);
    parameters.Add("ConnectionString", _connectionString);
    parameters.Add("Writer", new ListBoxTextWriter(lstEvents));
    WorkflowApplication i = new WorkflowApplication
    (new UserWF(), parameters);
    // Setup persistence
    i.InstanceStore = _instanceStore;
    i.PersistableIdle = (waiea) => PersistableIdleAction.Unload;
    i.Run();
}
```

پارامترهای ورودی را از روی فرم مقدار دهی می‌کنیم. یک شی از کلاس WorkflowApplication ایجاد می‌کنیم. خاصیت InstanceStore آن را با Store ای که ایجاد کردیم مقدار دهی می‌کنیم. توسط رویداد PersistableIdle فرآیند رو مجبور می‌کنیم به Persist شدن و Unload شدن.

و سپس فرایند را اجرا می‌کنم.

اگر یادتان باشد، در فرآیند، از یک InvoiceMethod استفاده کردیم. متد مورد نظر را هم در کلاس ApplicationInterface.cs ایجاد می‌کنیم.

```
public static void NewUser(User l)
{
    if (_app != null)
        _app.AddNewUser(l);
}
```

همین طور که می‌بینید، یک متد هم در کلاس AddRequest ایجاد می‌شود؛ با این محتوا

```
public void AddNewUser(User l)
{
    this.lstUsers.Dispatcher.BeginInvoke(new Action(() => this.lstUsers.Items.Add(l)));
}
```

این متد فقط یک کاربر را به لیست کاربران اضافه می‌کند. این لیست همه کاربران را نشان می‌دهد. توسط رویداد SelectionChanged این کنترل، کاربر انتخاب شده را بررسی کرده در صورتی که کاربر جدید باشد، امکان تایید شدن را برایش فراهم می‌کنیم؛ که نمایش دکمه تایید است.

```
private void lstUsers_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    if (lstUsers.SelectedIndex >= 0)
    {
        User l = (User)lstUsers.Items[lstUsers.SelectedIndex];
        lblSelectedNotes.Visibility = Visibility.Visible;
        if (l.Status == "New")
        {
            lblAgent.Visibility = Visibility.Visible;
            txtAcceptedBy.Visibility = Visibility.Visible;
            btnAccept.Visibility = Visibility.Visible;
        }
    }
}
```

```

        else
        {
            lblAgent.Visibility = Visibility.Hidden;
            txtAcceptedBy.Visibility = Visibility.Hidden;
            btnAccept.Visibility = Visibility.Hidden;
        }
    }
    else
    {
        lblSelectedNotes.Content = "";
        lblSelectedNotes.Visibility = Visibility.Hidden;
        lblAgent.Visibility = Visibility.Hidden;
        txtAcceptedBy.Visibility = Visibility.Hidden;
        btnAccept.Visibility = Visibility.Hidden;
    }
}
}

```

و برای رویداد کلیک دکمه تایید کاربر :

```

private void btnAccept_Click(object sender, RoutedEventArgs e)
{
    if (lstUsers.SelectedIndex >= 0)
    {
        User u = (User)lstUsers.Items[lstUsers.SelectedIndex];
        Guid id = u.WorkflowID.Value;
        UserDataDataContext dc = new UserDataDataContext(_connectionString);
        dc.Refresh(RefreshMode.OverwriteCurrentValues, dc.Users);
        u = dc.Users.SingleOrDefault<User>(x => x.WorkflowID == id);
        if (u != null)
        {
            u.AcceptedBy = txtAcceptedBy.Text;
            u.Status = "Assigned";
            dc.SubmitChanges();
            // Clear the input
            txtAcceptedBy.Text = "";
        }
        // Update the grid
        lstUsers.Items[lstUsers.SelectedIndex] = u;
        lstUsers.Items.Refresh();
        WorkflowApplication i = new WorkflowApplication(new UserWF());
        i.InstanceStore = _instanceStore;
        i.PersistableIdle = (waiea) => PersistableIdleAction.Unload;
        i.Load(id);
        try
        {
            i.ResumeBookmark("GetAcceptes", u);
        }
        catch (Exception e2)
        {
            AddEvent(e2.Message);
        }
    }
}

```

کاربر را انتخاب می‌کنم مقادیرش را تنظیم می‌کنیم. آن را ذخیره کرده و workflow را از روی guid مربوط به آن که قبلاً در فرآیند به Entity دادیم، Load می‌کنیم و همانطور که می‌بینید توسط متد ResumeBookmark فرآیند رو از جایی که می‌خواهیم ادامه می‌دهیم. البته می‌توان تایید کاربر را هم در خود فرآیند انجام داد و چون نوشتن Activity مرتبط با آن تقریباً تکراری است با اجازه‌ی شما من اون رو نوشتم و زحمتش با خودتونه.

حالا فقط مانده‌است که همه کاربران را در ابتدای نمایش فرم از دیتابیس فراخوانی کنیم و در لیست نمایش دهیم:

```

private void LoadExistingLeads()
{
    UserDataDataContext dc = new UserDataDataContext(_connectionString);
    dc.Refresh(RefreshMode.OverwriteCurrentValues, dc.Users);
    IEnumerable<User> q = dc.Users;
    foreach (User u in q)
    {

```



```

    AddNewUser(u);
}
}

```

و فراخوانی این متد را به انتهای رویداد Load صفحه واگذار می‌کنیم.

پروژه رو اجرا کرده و یک کاربر را اضافه می‌کنم. همانطور که می‌دانید این کاربر در فرآیند ایجاد و در دیتابیس ذخیره می‌شود

The screenshot shows a Windows application window titled "MainWindow". Inside the window, there is a registration form with three input fields: "Name:" containing "Reza1", "Phone Number" containing "2231", and "Email:" containing "reza\_bazargan@msn.com". Below these fields is a "Register" button. Under the button is a table titled "Current User" with three columns. The first row of the table contains the values "Reza1", "2231", and "reza\_bazargan@msn.comNew". At the bottom of the window is an "Event Log" section containing a single log entry: "User Reza1 Registered and waiting for Accept".

Current User		
Reza1	2231	reza_bazargan@msn.comNew

Event Log

User Reza1 Registered and waiting for Accept

برنامه را می‌بندم و دوباره اجرا می‌کنم. کاربر را انتخاب می‌کنم و یک نام برای admin انتخاب و آن را تایید می‌کنم. فرآیند را از bookmark مورد نظر اجرا کرده و به پایان می‌رسد. با بسته شدن برنامه، فرایند Idle و Unload می‌شود و ذخیره آن در sqlserver صورت می‌گیرد.



## نظرات خوانندگان

نویسنده: وهاب زاده  
تاریخ: ۱۳۹۲/۰۴/۲۱ ۰:۳۵

آقا رضا عالی بود، استفاده کردم.

موفق باشی

نویسنده: بهار  
تاریخ: ۱۳۹۲/۰۶/۰۶ ۱۷:۴۹

سلام. ممنون از زحماتی که کشیدید. یک سوال کلید WorkflowID در جدول User کلید خارجی از کدوم یک از جداول WF است؟  
توی هیچ کدوم از جدولها نیست!

نویسنده: رضا بازرگان  
تاریخ: ۱۳۹۲/۰۷/۲۱ ۲۱:۵۰

ببخشید که دیر شد

شما در هر Activity که ایجاد می‌کنید با کمک خاصیت Context به خصوصیات workflow جاری دسترسی دارید.

اگر دیده باشید ما یک Activity به نام CreateUser ایجاد کردیم که Entity مربوط به User رو ایجاد می‌کند.

درست قبل از اینکه user رو ذخیره کنیم مقدار فیلد WorkflowId اون رو که در واقع شناسه‌ی workflow جاری برای این user هست رو به اون اختصاص می‌دیم.

این فیلد در حقیقت شناسه‌ی workflow ایه که این entity در اون جریان داره

شناسه workflow در جدول instancesTable فیلدیه به نام WorkflowInstanceId