ClickOnce, WPF, MSBuild и несколько окружений

Или сказ о том, как сделать публикацию приложений в один клик на разные окружения для тестирования разных версий приложений на WPF с помощью ClickOnce и TFS 2010.

На днях мне нужно было решить следующую, на мой взгляд, достаточно распространенную задачу, по размещению двух версий приложения. Одна версия QA – для тестирования текущих наработок, то, что реализуется каждый день: UI, логика, исправление мелких ошибок. Другая версия – Prod – для тестирования общих алгоритмов на реальных данных. В целом стандартная практика в мире разработки.

Решение этой задачи оказалось не столь стандартное и простое как я ожидал, для такой задачи. Но обо всем по порядку.

Диспозиция

В качестве исходных данных мы примем то, что у вас есть работоспособное приложение на WPF (для WinForm будет то же самое, но WPF более сложный случай), которое можно локально собрать в нескольких конфигурациях: Dev, QA(Cons), Prod. Не важно, что именно у вас зависит от конфигурации, в общем случае, скорее всего это строки соединения с базой данных, оптимизация и логирование ошибок/действий пользователя.

Так же, как дополнительное условие, все конфигурации должны собираться на билд-сервере, в данном случае на TFS и при использовании MSBuild. То, что дальше описывается можно собирать локально, с помощью командной строки. Так что TFS – усложнение задачи, которое будем принимать во внимание.

И еще у вас успешно настроена на публикацию из VisualStudio с помощью ClickOnce хотя бы одна конфигурация приложения.

Еще раз, у нас есть:

- WPF приложение
- У приложения несколько рабочих конфигураций
- Все конфигурации компилируются на билд-сервере (TFS)
- Хотя бы одна конфигурация публикуется с помощью ClickOnce из VisualStudio

Проблема

В описанной системе все хорошо, все собирается на TFS, в разных конфигурациях. Т.е. я могу в любой момент получить рабочее приложение с любой конфигурацией. Однако распространение для пользователей идет только по линии QA. До определенного этапа разработки этого хватало. Теперь же надо опубликовать Prod, при этом на одном компьютере должны одновременно стоять как QA версия, так и Prod.
Проблем с физической публикацией не возникло. Т.е. на сервер складываются разные версии приложения, **но при установке пользователю, одна версия затирает другую**.

Кроме этой проблемы вам придется решить еще ряд других, не менее важных задач, как например обеспечение версионности приложения.

Надеюсь, что проблема ясна, но я более подробно покажу еще раз, как приходят к этой ситуации и как ее решить во всех подробностях.

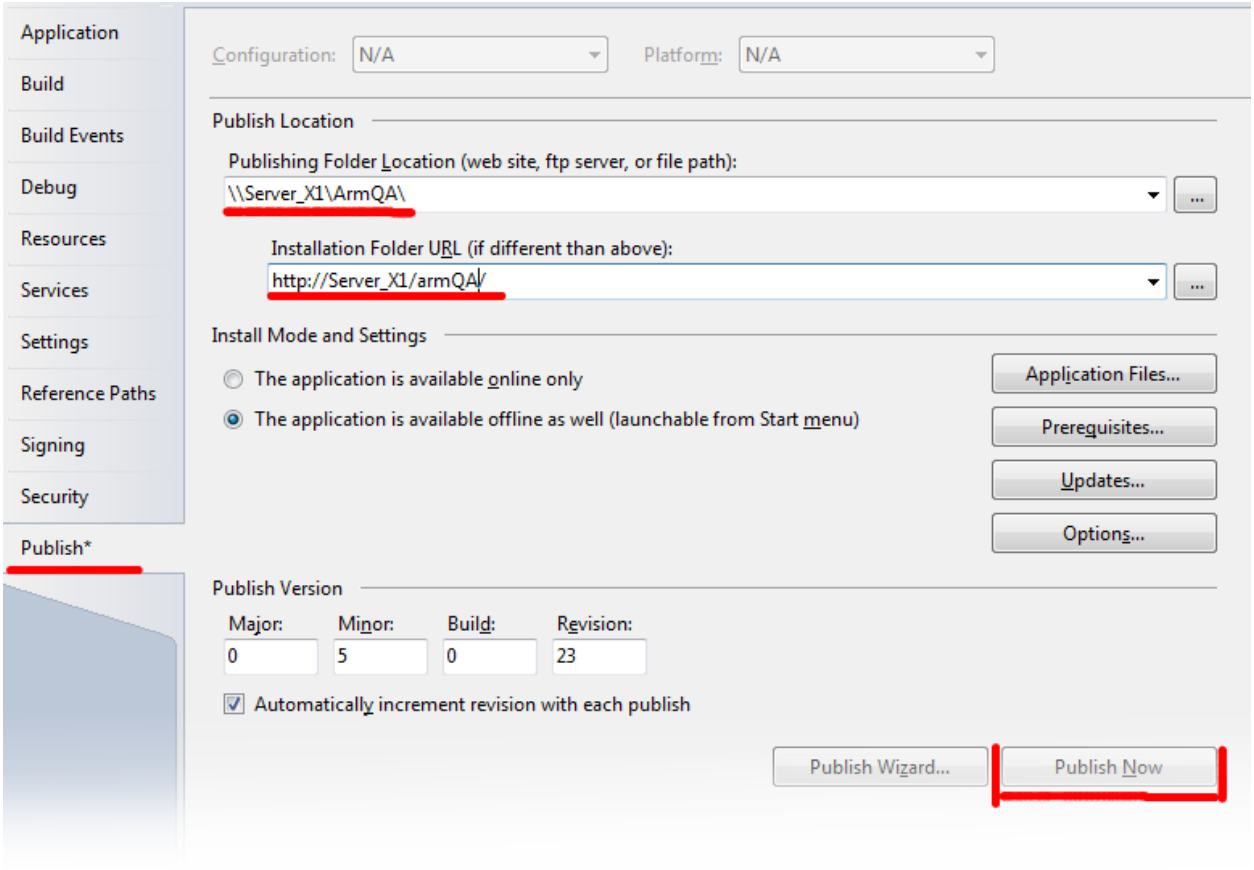
Осознание проблемы

Все примеры будут на реальном приложении и окружении, так что исходных кодов не будет, так как они бессмыслены в текущем примере, но будет много картинок, по которым вы сможете быстро узнать свой случай и настроить все легко и безболезненно.

Итак, как было уже ранее сказано у вас есть приложение, которое работоспособно и собирается на билд-сервере.

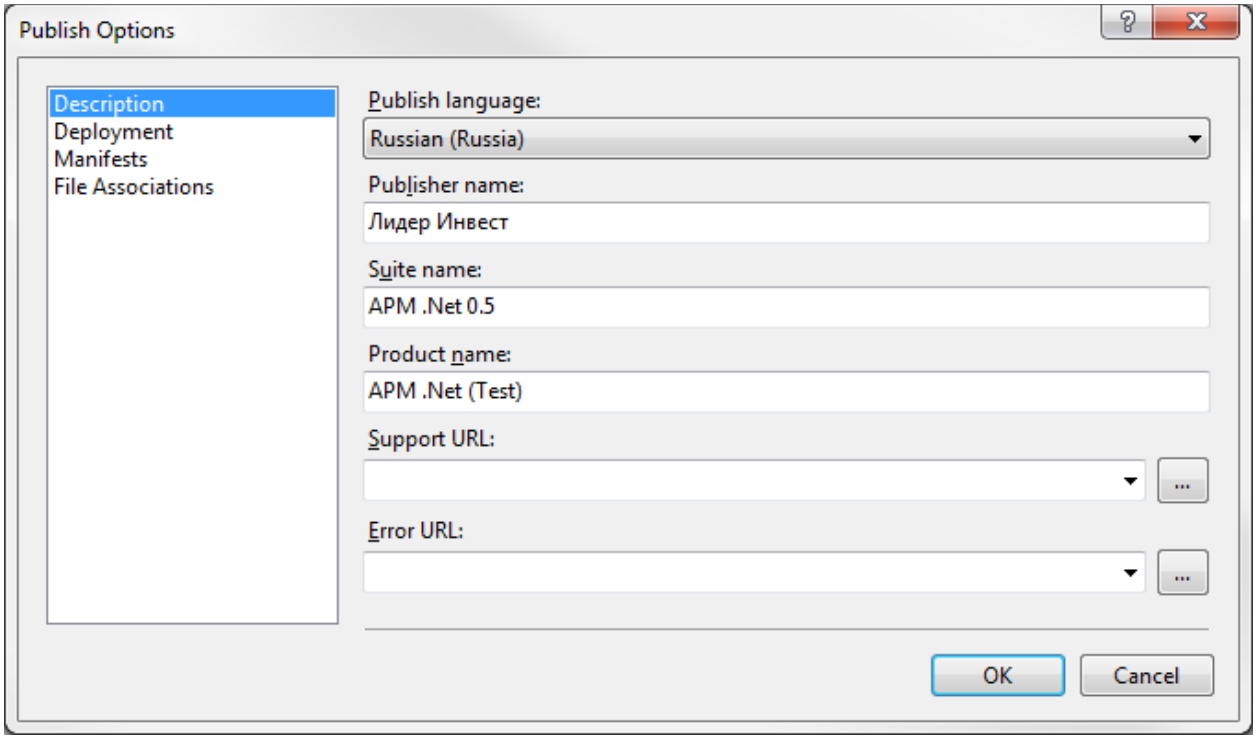
Стандартная публикация ClickOnce

Публикация ClickOnce приложения настраивается в свойствах головного (исполняемого) проекта.

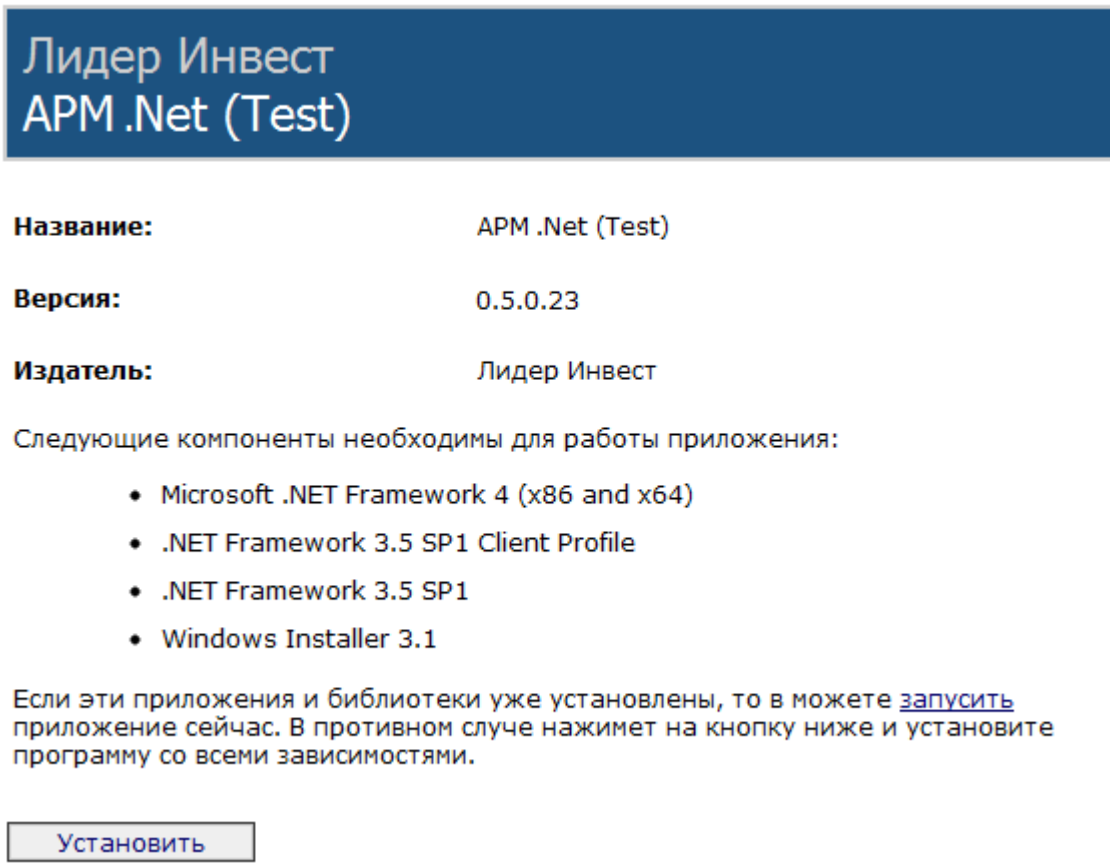


Вам надо настроить папку на сервере для выкладывания готового приложения, и задать адрес, где будет опубликован файл **publish.htm**

Опустим в данном случае Application Files, Prerequisites, Updates, а остановимся на **Options**.

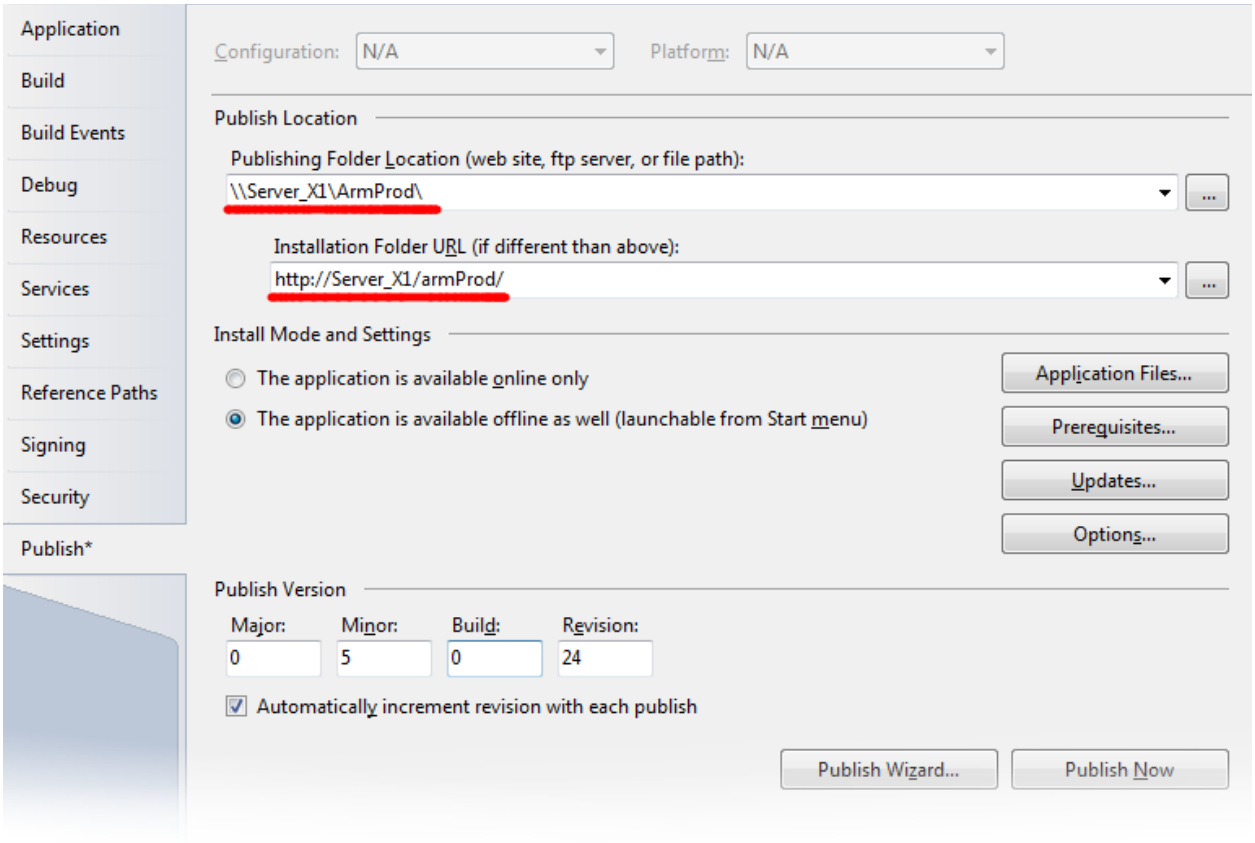


Здесь задаются основные данные, которые вы увидите на странице **publish.htm**, она, к слову, выглядит приблизительно так:



Здесь отображаются данные о названии продукта, компании, версии опубликованной. При этом прошу заметить, что **версия публикации никак не коррелирует с версией приложения**.

Для публикации версии Prod, придется менять адреса публикации, название продукта в меню Options, конфигурацию. Это все не очень удобно, и велика вероятность забыть поменять важные настройки.



Публиковать приложения из VS в целом является моветоном.

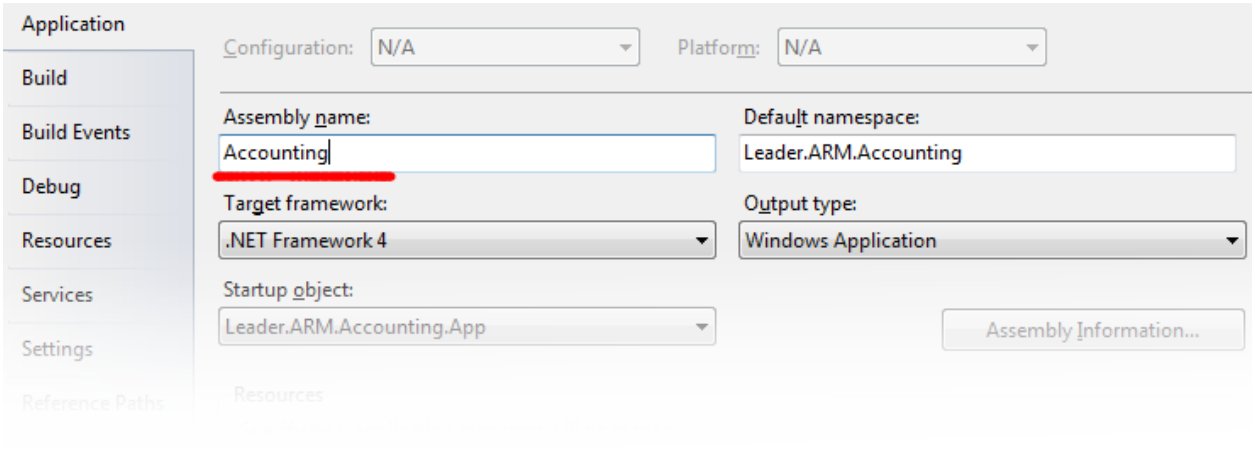
После того, как вы опубликовали приложения в две разные директории, это не значит, что они будут различными с точки зрения системы клиента. Они будут идентичны. Можете проверить на досуге.

Отличительные признаки

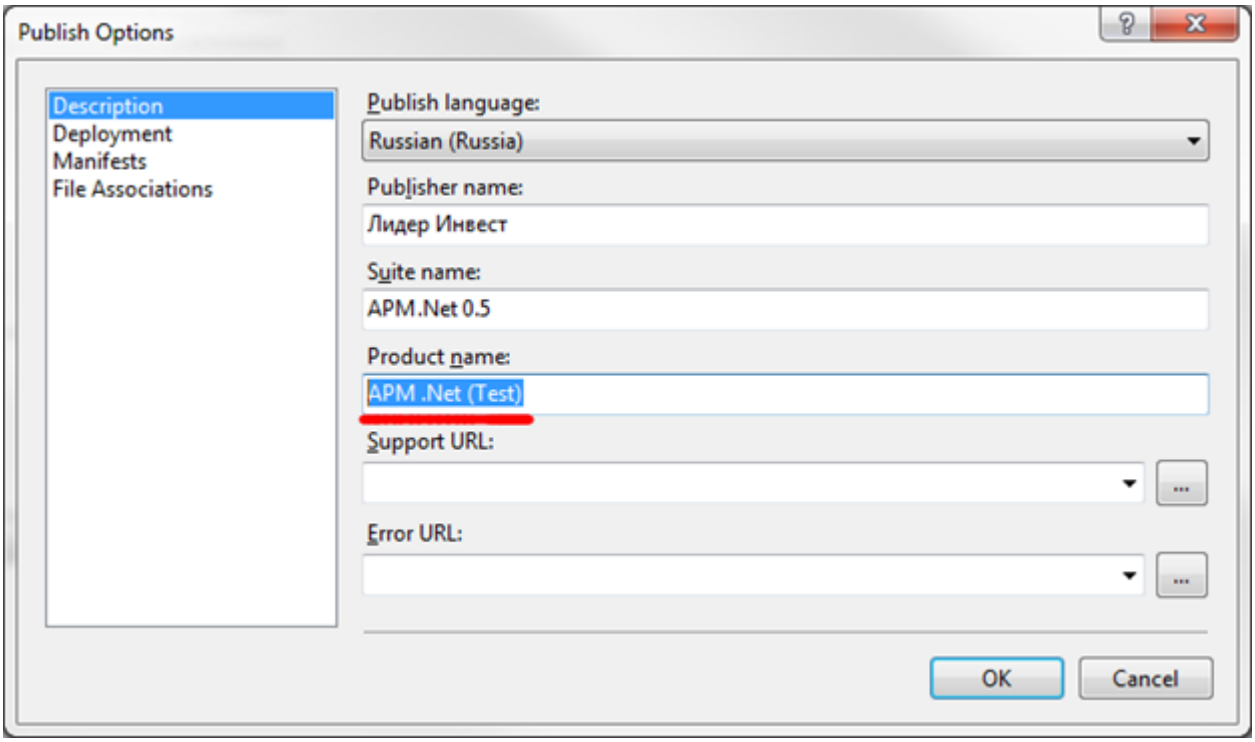
После копания в сети, оказалось, что отличительными признаками приложения с точки зрения клиентской системы являются:

- Имя сборки
- Имя приложения

Имя сборки:



Имя приложения:



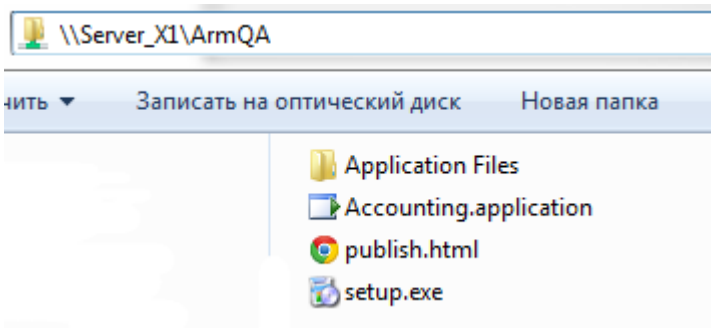
Казалось бы, вот оно счастье и решение всех проблем, но не тут-то было.

Я уже упомянул, что для пущей сложности у нас WPF приложение, которое завязано на имя сборки, и поменять его запросто не получится, потому что у нас есть достаточное количество пользовательских экранов (UserControl), объявления XAML которых начинается как:

```
<UserControl x:Class="Leader.ARM.Accounting.Views.AccountsAnalysisView"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             …
```

Не получается легкой жизни по всей видимости, т.е. если сменить названия продукта можно в зависимости от условий компиляции, то переписывать исходные коды… что-то мне не улыбается.

Дальнейший поиск и логика показали, что наверно можно поискать правды в файле **%Публикуемый_проект%.application**. Данный файл находится по адресу публикации приложения, и в директории сборки проекта.



В моем случае это **Accounting.application**. Это простой xml файл, который можно открыть в любом текстовом редакторе. Открываем и пытаемся найти там отличительные признаки.

```
<?xml version="1.0" encoding="utf-8"?>
<asmv1:assembly не_важно_тут_стандартные_схемы>
  <assemblyIdentity name="Accounting.application" version="0.5.0.23"
publicKeyToken="0000000000000000" language="ru-RU" processorArchitecture="x86"
xmlns="urn:schemas-microsoft-com:asm.v1" />
  <description asmv2:publisher="Лидер Инвест" co.v1:suiteName="ARM .Net 0.5"
asmv2:product="ARM .Net (Test)" xmlns="urn:schemas-microsoft-com:asm.v1" />
  <deployment install="true" mapFileExtensions="true" co.v1:createDesktopShortcut="true">
    <subscription>
      <update>
        <beforeApplicationStartup />
      </update>
    </subscription>
    <deploymentProvider codebase="http://Server_X1/armQA/Accounting.application" />
  </deployment>
  <compatibleFrameworks xmlns="urn:schemas-microsoft-com:clickonce.v2">
    <framework targetVersion="4.0" profile="Full" supportedRuntime="4.0.30319" />
  </compatibleFrameworks>
  <dependency>
    <dependentAssembly dependencyType="install" codebase="Application
Files\Accounting_0_5_0_23\Accounting.exe.manifest" size="47449">
      <assemblyIdentity name="Accounting.exe" version="0.5.0.23"
publicKeyToken="0000000000000000" language="ru-RU" processorArchitecture="x86"
type="win32" />
      <hash>
        <dsig:Transforms>
          <dsig:Transform Algorithm="urn:schemas-microsoft-com:HashTransforms.Identity"
/>
        </dsig:Transforms>
        <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
        <dsig:DigestValue>1dRMDHoKoUsB7bUWmC/GvUZllZc=</dsig:DigestValue>
      </hash>
    </dependentAssembly>
  </dependency>
</asmv1:assembly>
```

Методом проб и ошибок, а так же с учетом ранее полученной информации обнаруживаем, что самая важная часть, для различения приложений является:

- assemblyIdentity name="Accounting.application" (строка 3)
- asmv2:product="ARM .Net (Test)" (строка 4)

Все остальное по большому счету не важно. Номер версии важен, но он не будет являться отличительным признаком приложения.

Т.е. если вы в одной из папок (QA) с опубликованным приложением поменяете эти значения, так чтобы они отличались от другой программы (Prod) и попробуете установить приложения, то вы получите 2 разных программы. Это выразится в двух ярлыках на рабочем столе и в меню пуск. Если конечно в настройках публикации указано на создание ярлыков.

Получается, что на данный момент мы выяснили что, как и где, надо поменять для того чтобы получить два разных приложения.

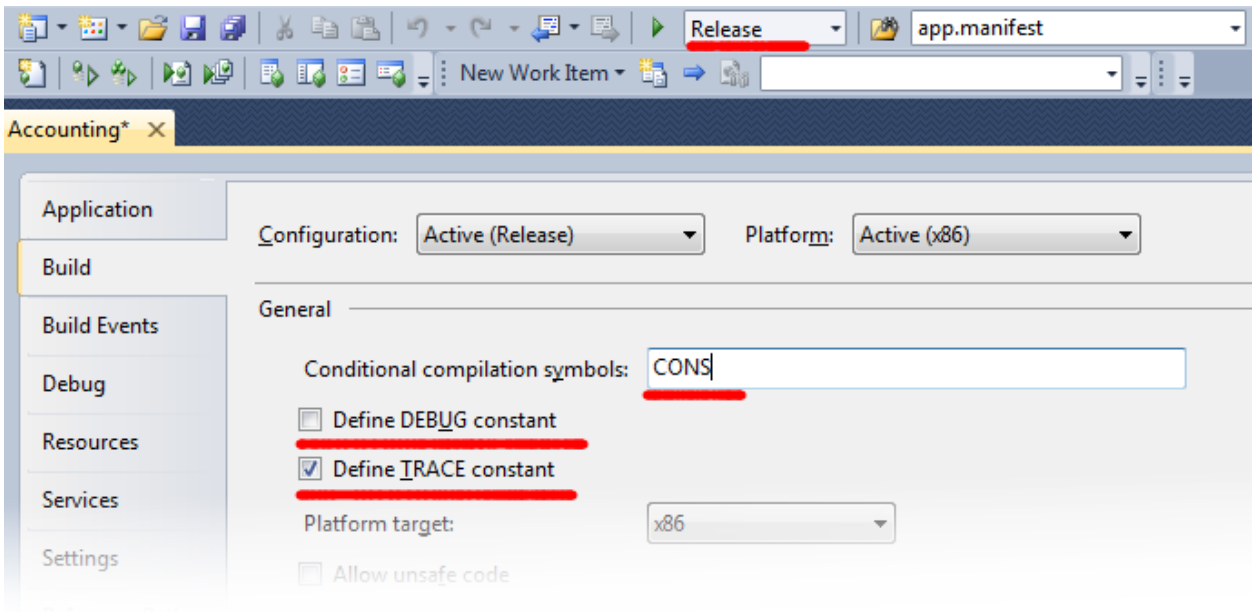
Хехе, большинство статей, чуть ли не на данном моменте останавливаются в повествовании, так как типа дальше дело техники поменять эти значения в процессе сборки приложения. Так вот посылаю лучи фертильности авторам таких постов.

Подготовка проекта

Прежде чем приступить к написанию скриптов по замене значений во время публикации приложений и всего такого прочего, необходимо должным образом подготовить проект и окружение к будущим манипуляциям.

Первым делом надо задать значения папки для публикации, и адрес страницы публикации в зависимости от конфигурации проекта.

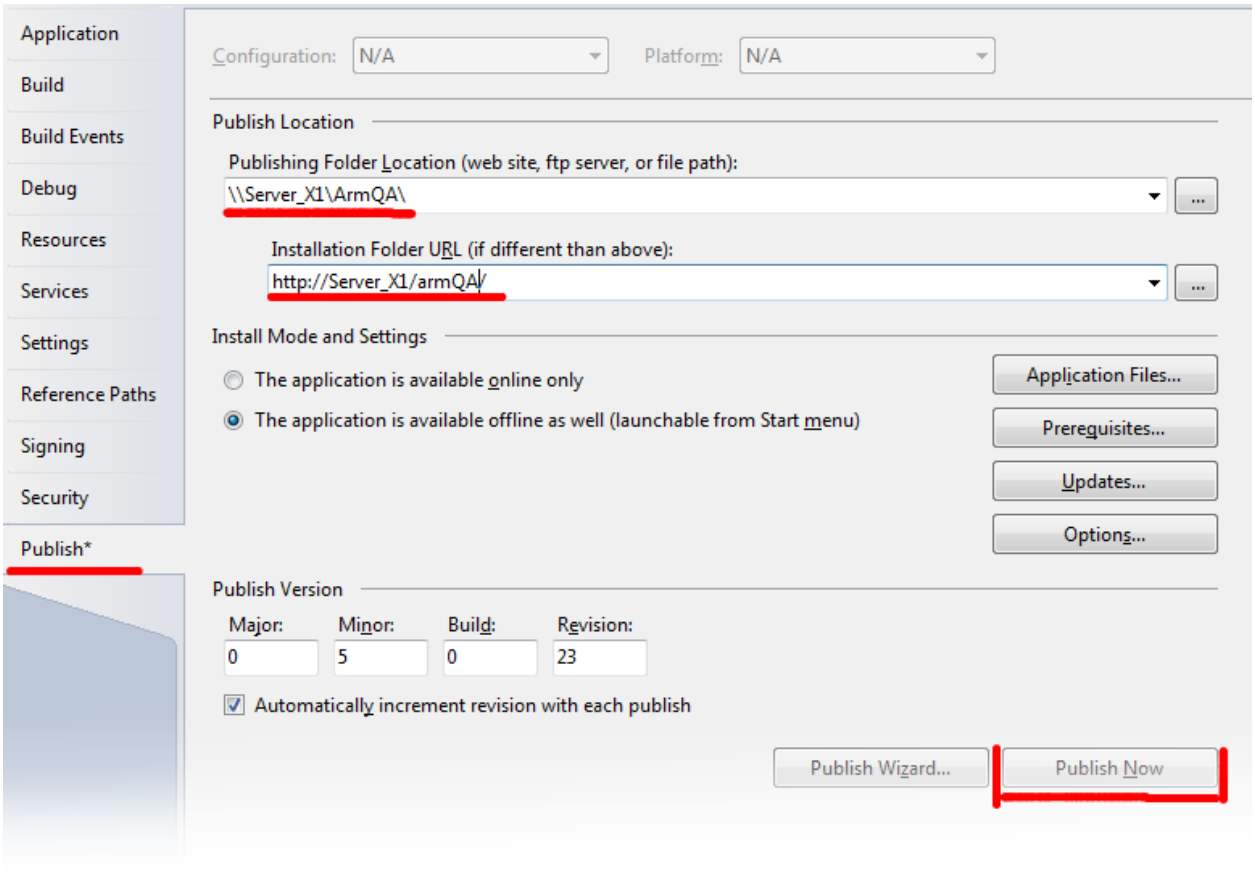
Для многих не новость, что настройки проекта меняются в зависимости от того, какая конфигурация выставлена. Например, в конфигурации Debug объявляются символы DEBUG, TRACE, можно указать определенные PreBuild и PostBuild действия. Если переключить конфигурацию в Release, то можно увидеть, как исчезнут галки с опций объявления символа DEBUG



Поэкспериментируйте с настройками, для кого это в новинку.

Однако настройки конфигурации не затрагивают закладку **Publish** с помощью, которой происходит публикация приложений, и значения:

- Publishing Folder
- Installation Folder URL



Не привязываются к конфигурации окружения. Это поначалу может показаться камнем преткновения, но так как определения всех проектов и способов их сборки обозначаются в XML нотации, то логично предположить, что все видимые настройки имеют свое отображение в схеме XML, и с помощью определенных свойств мы сможем задать эти значения.

Настройка адресов публикации

По большому счету VisualStudio больше нам не помощник и все дальнейшие действия будут совершаться с помощью Notepad++ и командной строки из окружения VisualStudio.

Открываем в Notepad++ файл публикуемого проекта, в моем случае, как вы заметили это Accounting.csproj. Находим секции, где определяются настройки, зависящие от конфигурации. Их легко найти по отличительным признакам:

- Находятся в начале файла, в районе 40-50 строки.
- Начинаются с <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == **'Debug|x86'** ">

В начале файла это практически всегда самые длинные строки. Итак, ищем свои конфигурации, в моем случае это:

- <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|x86' ">
- <PropertyGroup Condition="'$(Configuration)|$(Platform)' == 'Prod|x86'">

Первая для Cons, вторая для Prod. Вот так вот несогласовано, показываю на примере реального проекта, а создать конфигурацию легче и лучше, на мой взгляд, чем возиться с определениями символов.

После того, как вы нашли свою секцию с определением конфигурации, надо будет добавить следующие настройки:

```
    <InstallUrl>http://Server_X1/armQA/</InstallUrl>
    <PublishDir>\\Server_X1\armQA\</PublishDir>
```

Конечно, адреса сервера должны быть ваши. Не обращайте внимания, если VS будет ругаться на недопустимые тэги, все хорошо на самом деле. Полная секция в простейшем варианте выглядит так:

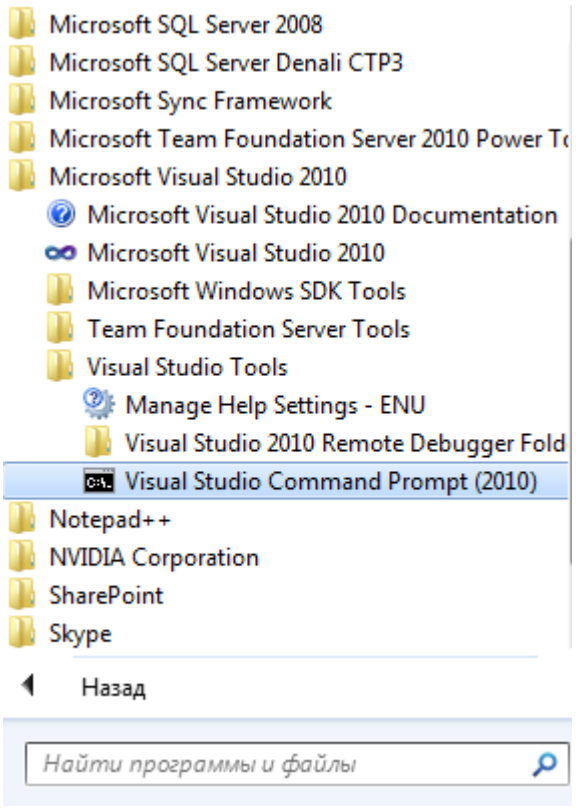
```
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|x86' ">
    <PlatformTarget>x86</PlatformTarget>
    <DebugType>pdbonly</DebugType>
    <Optimize>true</Optimize>
    <OutputPath>bin\Release\</OutputPath>
    <DefineConstants>TRACE;CONS</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
    <InstallUrl>http://Server_X1/armQA/</InstallUrl>
    <PublishDir>\\Server_X1\armQA\</PublishDir>
  </PropertyGroup>
```

То же самое надо проделать для других конфигураций, которые должны идентифицироваться клиентом как разные программы. Вставить теги InstallUrl и PublishDir, указав значения папок и адресов для публикации приложения.

Данные теги являются переменными, которыми оперирует MSBuild при сборке проекта.

Проверка публикации

На данном этапе можно проверить работу только что введенных переменных путем публикации приложения из командной строки. Однако имя приложения все еще не изменится. Это мы сделаем немного позже и в другом файле. Сначала опробуем публикацию приложения из командной строки. Открываем **Visual Studio Command Prompt**



Переходим в директорию проекта, или же надо будет указать полный путь до файла **.sln** при использовани MSBuild.

Далее набираем *msbuild "arm .net.sln" /**t:Publish** /p:Configuration=Release* и запускаем. Будет много-много текста, в конце концов напишут что ошибок нет и время публикации такое-то. В данном случае для нас важным параметром является указание цели сборки – **Publish**. Для этой цели происходит rebuild и публикация проекта. Проект будет опубликован с последней указанной версией на закладке Publish, и сколько бы раз вы не запускали эту команду.

Если все прошло без ошибок, то можно проверить, что опубликовалась версия для QA (удалите файлы из папки для публикации и запустите ее снова). В моем случае QA (Cons) так как она связана с конфигурацией Release.

Для публикации Prod версии надо будет выполнить *msbuild "arm .net.sln" /t:Publish /p:**Configuration=Prod***

Если все хорошо, а так и должно быть, то переходим к следующему пункту. Тут стоит упомянуть, что при такой публикации не будет создана страница publish.htm, однако эту проблему мы решим, но чуть позже.

Версионность

Публикация с одной и той же версией нас не устраивает совершенно, так как тогда клиент не сможет распознать, что появилась новая версия. Версию можно передать во время сборки приложения с помощью параметра **ApplicationVersion**

msbuild "arm .net.sln" /t:Publish /p:Configuration=Release;**ApplicationVersion=0.5.0.34**

Но это неудобно, как увеличивать номера при автоматической сборке?

Вообще вопрос составления версий не столь простой как может показаться. На первый взгляд кажется, что нумерация в духе 0.5.1.456 вполне хороша. Я тоже так подумал, нашел решение для увеличения номера ревизии (последний номер в версии продукта), однако это решение опиралось на внешний файл, в который и записывалась текущая версия. Такой подход не подходил для Continuous Deployment, так как после сборки я не смогу зачекинить файл с версией обратно в TFS. Честно сказать даже если это и возможно, то сама мысль мне показалась порочной и я не стал искать в этом направлении.

Следующей идеей, на которой я и остановился, представление версии в виде некоторой комбинации временных значений. Например, мне понравился вариант:

(Главная версия).(Версия и Год).(Порядковый номер дня в году).(Час и минута)

Примеры:

- 1.411.302.1001
- 0.012.12.2322

Данную версионность мы и реализуем в программе. Реализовать свою комбинацию будет не очень сложно, честное слово.

Версию приложения надо будет передать и для сборки, чтобы можно было в свойствах исполняемого файла увидеть версию приложения, и для публикации.

Подготовка к трансформации проекта

Чтобы у нас все получилось, потребуется скачать и установить на машине билд-сервера MSBuild Community Tasks Project, так же советую его поставить и на своей машине, для экспериментов и быстрого доступа к файлу справки. Данный пакет позволяет обращаться к реализации массы наиболее часто востребованных функций во время преобразований в процессе построения приложения с помощью MSBuild. Пакет устанавливается по адресу C:\Program Files (x86)\MSBuild\MSBuildCommunityTasks, это чтобы вы быстро нашли справку по новым доступным задачам.

Канон

Так уж получается, что мы вносим серьезные изменения в процесс сборки приложения, и как все хорошие разработчики будем отделять мух от котлет, т.е. почти все существенные изменения будут производиться в отдельном файле.

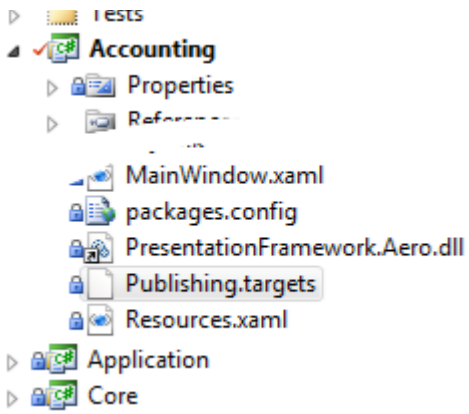
MSBuild при сборке проекта руководствуется сложным набором правил и указаний, которые мы можем модифицировать или дополнять. Основные указания о сборке проекта содержаться в файле Common.CSharp.targets – который менять каким-либо образом крайне не рекомендуется. По принятому соглашению, все дополнения и расширения, касающиеся построения приложений должны иметь расширение **.targets**, по сути это XML файл.

Итак, нам придется:

- немного модифицировать файл проекта .csproj
- создать новый файл .targets

Начнем с создания нового файла в корне исполняемого проекта, в котором будем изменять параметры построения и публикации нашего приложения. Файл можете назвать как угодно, я назвал **Publishing.targets**. В начало файла следует скопировать определение проекта из файла .csproj того проекта, который вы будете публиковать. Т.е. открываю файл Accounting.csproj в Notepad++, и копирую узел объявления проекта в новый файл. В итоге в файле Publishing.targets должно быть что-то в духе:

```
<Project ToolsVersion="4.0" DefaultTargets="Build"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
</Project>
```



Сохраняем и начинаем вносить изменения в исполняемый проект. Надо внести ссылки на MSBuild Community Task Project и на наш новый файл.

Открываем файл проекта, перемещаемся вниз файла, находим тэги **Imports** и дописываем там свои две строчки:

```
<Import Project="Publishing.targets" />
<Import
Project="$(MSBuildExtensionsPath)\MSBuildCommunityTasks\MSBuild.Community.Tasks.Targets"
/>
```

Указываем путь до нового файла, имя напишите свое, путь относительный от проекта, и указываем путь до расширений MSBuild – тут ничего менять не надо по идее. Пробуем собрать проект из командной строки, все должно быть хорошо для конфигураций Release и Prod. Для тех разработчиков, у которых не стоит MSBuild Community Task Project проект не соберется и будет просить установить недостающие библиотеки. Чтобы этого избежать сделаем включение задач из MSBuild CTP опциональным, для всех конфигураций кроме Debug.

```
<Import Project="Publishing.targets" Condition=" '$(Configuration)|$(Platform)' !=
'Debug|x86' "/>
<Import
Project="$(MSBuildExtensionsPath)\MSBuildCommunityTasks\MSBuild.Community.Tasks.Targets"
Condition=" '$(Configuration)|$(Platform)' != 'Debug|x86' "/>
```

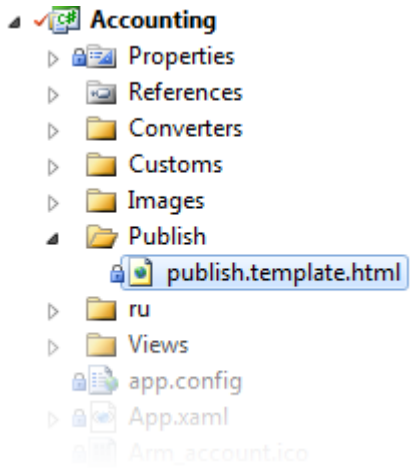
Создание страницы публикации

Как уже было упомянуто выше, при публикации из командной строки, страница publish.htm не будет создана автоматически, так что нам придется подготовить шаблон страницы, включить в

проект и затем в процессе публикации копировать ее в папку, где будет опубликовано итоговое приложение.

Не буду тут приводить полный текст файла publish.htm, лучше его скачать здесь. Затем можете отредактировать его на свой вкус: перевести, дополнить информацией или удалить лишние пункты. Главное оставить/добавить уникальные идентификаторы для последующей замены во время публикации.

Отредактированный файл надо включить в проект, я создал папку Publish и поместил туда publish.htm



О том, как заменять значения в этом файле речь пойдет чуть ниже.

Трансформация данных публикации

При публикации приложения хочется, чтобы данные были актуальными и заменялись на то, что нам надо в зависимости от конфигурации сборки. Имя продукта, версия, еще какие-либо данные. В данном случае я хочу менять название приложения, версию, идентификационное имя приложения.

Замена значений по условию

Наверно, это самый важный момент для того, чтобы приложение для разных окружений воспринималось клиентом как две разных программы. Сейчас будем устанавливать значения для окружения QA и PROD. Данные значения будут размещены в файле **Publishing.targets**.

Открываем файл и сразу за открытием тега Project вставляем код, все вместе должно выглядеть так:

```
<Project ToolsVersion="4.0" DefaultTargets="Build"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
 <Choose>
    <When Condition=" '$(ConfigurationName)' == 'Prod' ">
      <PropertyGroup>
        <ArmApplicationId>&quot;Accounting.application&quot;</ArmApplicationId>
        <ArmApplicationName>APM .Net</ArmApplicationName>
        <ProductName>APM .Net</ProductName>
      </PropertyGroup>
    </When>
    <When Condition=" '$(ConfigurationName)' == 'Release' ">
      <PropertyGroup>
```

```
        <ArmApplicationId>&quot;Accounting.application.test&quot;</ArmApplicationId>
        <ArmApplicationName>APM .Net (Test)</ArmApplicationName>
        <ProductName>APM .Net (Test)</ProductName>
      </PropertyGroup>
    </When>
  </Choose>
</Project>
```

Данная секция описывает условный переход и присвоение значений переменным, которые будем использовать далее. Choose – контейнер для описания условий, далее используется тег When в котором описывается условие для входа. В данном случае условие накладывается на значение конфигурации. Вам надо выставить свои значения в условии.

Далее идет тег **PropertyGroup** –он открывает секцию с объявлением переменных, так же в этой секции можно переписывать значения переменных. В этой секции ключевыми переменными являются:

- ArmApplicationId – можете придумать свое имя, мы специально вводим эту переменную для модификации файла **Accounting.application** который является конфигурацией опубликованного приложения.
- ProductName – это существующее имя для имени продукта.

Описанные переменные должны иметь различные значения для каждой конфигурации.

Прошу заметить, что значение переменной ArmApplicationId заключено в кавычки.

Для безопасности и для того, чтобы можно было собрать приложение для любой другой конфигурации рекомендуется указать значения по умолчанию для новых переменных. Для этого можете просто продублировать одну из секций PropertyGroup вставив ее перед тегом Choose.

Замены для страницы опубликованного приложения

Теперь можно подготовиться к окончательному формированию страницы опубликованного приложения publish.htm. Нам надо провести замены «переменных» в файле publish.htm, для этого продолжаем дописывать файл Publishing.targets и вставляем следующий код после секции Choose.

```
  <ItemGroup>
    <Tokens Include="PublisherName">
      <ReplacementValue>$(PublisherName)</ReplacementValue>
      <Visible>false</Visible>
    </Tokens>
    <Tokens Include="ProductName">
      <ReplacementValue>$(ProductName)</ReplacementValue>
      <Visible>false</Visible>
    </Tokens>
    <Tokens Include="Prerequsites">
      <ReplacementValue>@(BootstrapperPackage-
>'&lt;li&gt;%(ProductName)&lt;/li&gt;','%0D%0A')</ReplacementValue>
      <Visible>false</Visible>
    </Tokens>
    <Tokens Include="Username">
      <ReplacementValue>$(Username)</ReplacementValue>
      <Visible>false</Visible>
    </Tokens>
  </ItemGroup>
```

В данных заменах пока что нет версии приложения. Это скоро будет исправлено.

В действии

Собственно сейчас осталось только указать время, когда производить указанные замены. Для этого можно переопределить несколько событий (в терминологии MSBuild - Targets), сейчас же нам понадобиться только **AfterPublish**.

В описании этого события мы внесем последние изменения в страницу приложения, опубликуем страницу, а так же сделаем так, чтобы для клиента опубликованные приложения CONS и PROD были разными программами.

Прежде чем указать действия для AfterPublish, надо создать переменную для публикации файла publish.htm

```
  <PropertyGroup>
    <PublishFilePath>$(PublishDir)publish.html</PublishFilePath>
  </PropertyGroup>
```

Разместите этот код в конце файла Accounting.targets перед закрывающим тегом </Project>.

Следующим шагом будет реализация AfterPublish:

```
  <Target Name="AfterPublish">
    <Time Format="dd/MM/yyyy HH:mm">
      <Output TaskParameter="FormattedTime" PropertyName="PublishTime" />
    </Time>
    <TemplateFile Template="Publish\publish.template.html" Tokens="@(Tokens)"
OutputFilename="$(PublishFilePath)" />
    <FileUpdate Files="$(PublishFilePath)" Regex="\${PublishTime}"
ReplacementText="$(PublishTime)" />
    <FileUpdate Files="$(PublishDir)Accounting.application" Encoding="ASCII"
Regex="&quot;Accounting.application&quot;" ReplacementText="$(ArmApplicationId)" />
  </Target>
```

Первой строкой идет указание того, когда это будет выполнено – AfterPublish – зарезервированное имя, доступное к переопределению. Далее идет использование задачи Time из пакета MSBuild Community Task, с помощью этой задачи будем указывать время публикации на странице publish.htm.

Далее идет TemplateFile, с помощью этой задачи шаблон страницы модифицируется путем замены внутренних переменных на значения указанные в файлах .targets.

Следующие две задачи опять из пакета MS Build Community Task и позволяют заменять значения в файлах. По структуре вы можете догадаться о работе данной задачи. В нашем случае мы выставляем время публикации, а так же делаем уникальным идентификатор приложения, по которому (совместно с ProductName) клиент будет думать, что у него разные приложения.

Можно снова попробовать опубликовать приложение с разными настройками конфигурации и установить их у себя. Вы увидите, что установилось два приложения: QA (Cons) и PROD. Они работают независимо и не мешают работе друг друга.

Все было бы хорошо, однако номер версии приложений остаются неизменны и клиентские версии не будут знать о новых версиях, которые вы опубликуете. Так что сейчас решим задачу формирования номеров версий для каждой публикации проекта.

Создание версий

Как вы могли прочитать или заметить на собственном опыте, версия приложения и опубликованная версия – это две совершенно разных версии, однако желательно привести их в соответствие. Версия приложения хранится в AssemblyInfo.cs, версия публикации - в файле *.publish (это по версии интернетов, однако я не смог найти этот файл, так что для меня загадка, где хранится номер опубликованной версии.)

С помощью задач из пакета MSBuild CTP можно создать свой файл AssemblyInfo.cs в котором писать номер версии до того, как проект начнет собираться.

По условию задачи становится понятно, что надо искать и модифицировать цель\задачу (targets) с назаванием **BeforeBuild**. Эту задачу можно найти в конце файла .csproj для публикуемого проекта. Задача закомментирована, так что вы можете удалить комментарий и внести следующий код в задачу:

```
<Target Name="BeforeBuild" Condition=" '$(Configuration)|$(Platform)' != 'Debug|x86' ">
  <Attrib Files="$(ProjectDir)Properties\AssemblyInfo.cs" ReadOnly="false" System="false"
/>
  <Time>
    <Output TaskParameter="Hour" PropertyName="Hour" />
    <Output TaskParameter="DayOfYear" PropertyName="Revision" />
  </Time>
  <AssemblyInfo CodeLanguage="CS" OutputFile="$(ProjectDir)Properties\AssemblyInfo.cs"
AssemblyVersion="0.711.$(Revision).$(Hour)$([System.DateTime]::Now.Minute.ToString(&quot;
00&quot;))"
AssemblyFileVersion="0.711.$(Revision).$(Hour)$([System.DateTime]::Now.Minute.ToString(&q
uot;00&quot;))"
AssemblyCompany="Leader Invest" AssemblyProduct="Application"
AssemblyCopyright="Copyright © Leader Invest 2011" AssemblyTrademark="Leader Invest"
AssemblyCulture="" />
  <PropertyGroup>

<ApplicationVersion>0.711.$(Revision).$(Hour)$([System.DateTime]::Now.Minute.ToString("00
"))</ApplicationVersion>

<ApplicationVersionPublishPage>0.711.$(Revision).$(Hour)$([System.DateTime]::Now.Minute.T
oString("00"))</ApplicationVersionPublishPage>
  </PropertyGroup>
</Target>
```

Первым делом надо снять метку ReadOnly с файла AssemblyInfo.cs. Это делается с помощью задачи **Attrib**. Далее используем задачу Time из пакета дополнений к MSBuild, и формируем новые переменные из значений, которые могут быть предоставлены задачей Time. Для нас актуальны: часы и порядковый день года. Тег Output заполняет переменные.

Далее идет формирование файла AssemblyInfo, где формируется версия приложения и сборки. Так как сборка идет на MSBuild 4.0, то можно использовать вызовы к стандартным библиотекам.

```
[System.DateTime]::Now.Minute.ToString(&quot;00&quot;)
```

Это сделано для того, чтобы получить минуты с ведущими нулями, можно тот же фокус провернуть и со значением часов.

После того, как создали новый файл AssemblyInfo.cs записываем значения новой версии в переменные:

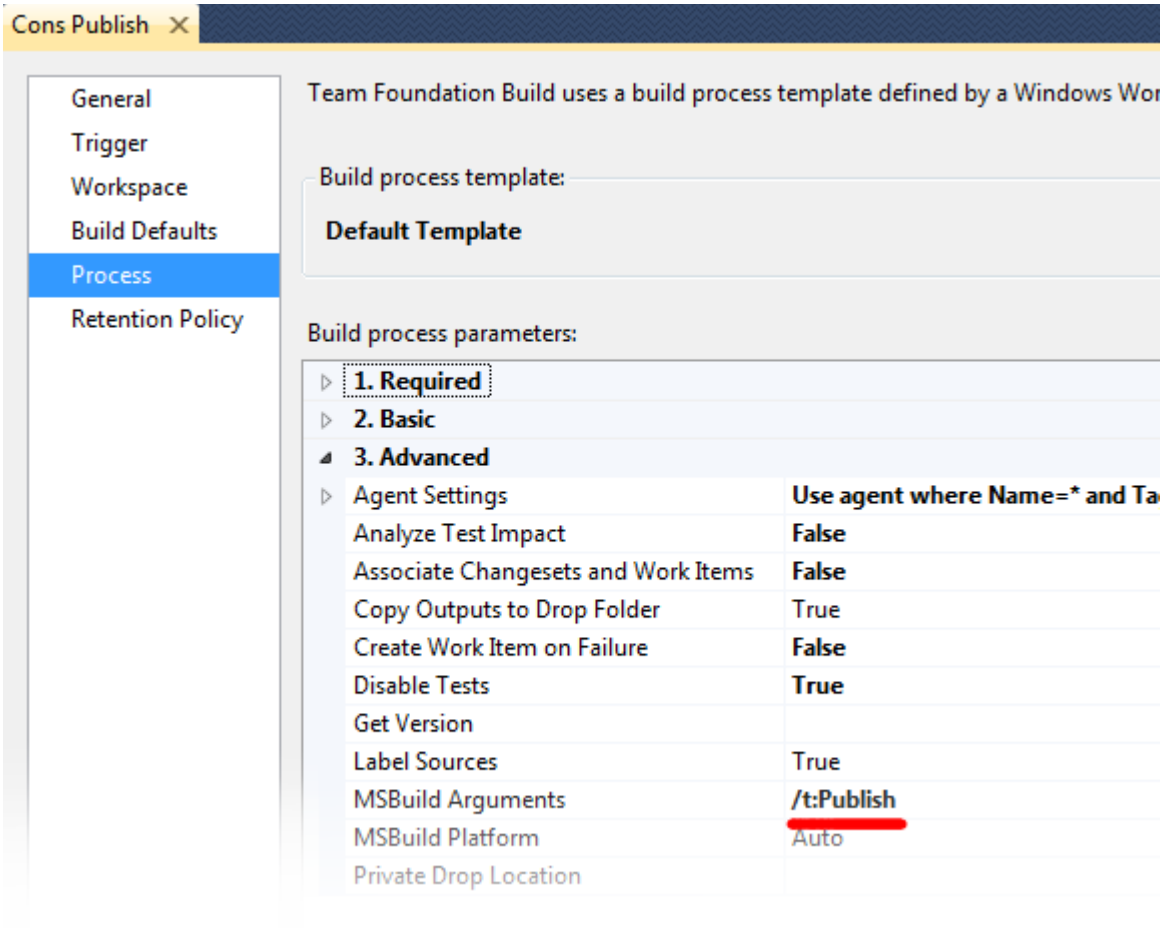
- ApplicationVersion –версия публикации
- ApplicationVersionPublishPage – для отображения на странице публикации

После того, как указали все действия на момент сборки, осталось внести финальные изменения в файл Publishing.targets. Надо внести изменения для использования ApplicationVersionPublishPage.

```
<Target Name="BeforePublish">
  <ItemGroup>
    <Tokens Include="ApplicationVersion">
      <ReplacementValue>$(ApplicationVersionPublishPage)</ReplacementValue>
      <Visible>True</Visible>
    </Tokens>
  </ItemGroup>
</Target>
```

Внесите этот кусок кода до определения задачи AfterPublish.

Все, теперь можете собрать приложение еще раз из командной строки и убедиться, что страница проекта обновляется, версия выставляется корректно, версия исполняемого файла меняется, при публикации установленное приложение рапортует о необходимости обновления. Осталось только создать новые определения сборок для билд-сервера и указать цель сборки Publish.



Заключение

Надеюсь, что у вас получится изменить проекты для публикации с первого раза и без привлечения сторонних источников. Буду рад любым отзывам и заметкам, по поводу неочевидных моментов в этой статье.

Hard’n’heavy!

Violet Tape