

MobiGuider2010   
Standards and Practices

Table of content

[Table of content 2](#_Toc279663697)

[1. Prepare your Visual Studio 4](#_Toc279663698)

[Visual Studio Templates 4](#_Toc279663699)

[2. MobiGuider 2010 Codebase Structure 5](#_Toc279663700)

[Agent Folder 5](#_Toc279663701)

[Business Folder 5](#_Toc279663702)

[Choreography Folder 5](#_Toc279663703)

[Components Folder 5](#_Toc279663704)

[Dao Folder 6](#_Toc279663705)

[Domain Folder 6](#_Toc279663706)

[Framework Folder 6](#_Toc279663707)

[Library Folder 6](#_Toc279663708)

[3. Hosting/Debugging WCF Services in IIS 7](#_Toc279663709)

[Preparation 7](#_Toc279663710)

[WCF Service Development Deployment 7](#_Toc279663711)

[4. Practice: MobiGuider Default IIS Deployment 8](#_Toc279663712)

[Introduction 8](#_Toc279663713)

[Http Structure 8](#_Toc279663714)

[*Some Examples* 8](#_Toc279663715)

[Net.Pipe Structure 8](#_Toc279663716)

[Deployment 9](#_Toc279663717)

[5. Practice: Signing and Versioning of MobiGuider Assemblies 10](#_Toc279663718)

[Purpose 10](#_Toc279663719)

[Steps 10](#_Toc279663720)

[6. How to: create a MobiGuider Application Service Skeleton Solution 13](#_Toc279663721)

[Creating the Initial Application Service Solution 13](#_Toc279663722)

[Extending theApplication Service Solution 18](#_Toc279663723)

[7. How to: create a MobiGuider Application Service Agent 19](#_Toc279663724)

[Purpose 19](#_Toc279663725)

[Creating the Agent Assembly 19](#_Toc279663726)

[8. Adding Service References to an Agent Assembly 20](#_Toc279663727)

[9. How to create a MobiGuider User Interface Solution Skeleton 22](#_Toc279663728)

[Creating the Initial Solution 22](#_Toc279663729)

[Extending the UI Solution 23](#_Toc279663730)

[The Application Agent Assembly 23](#_Toc279663731)

[10. Creating an Application Domain Assembly 24](#_Toc279663732)

[11. How to: create a MobiGuider Database 26](#_Toc279663733)

[Purpose 26](#_Toc279663734)

[Database conventions 26](#_Toc279663735)

[Creating the database project 28](#_Toc279663736)

[Import Database schema from an existing DB 28](#_Toc279663737)

[Adding Database References 30](#_Toc279663738)

[Pre/Post Build scripts 32](#_Toc279663739)

[Update, patch or release a new version of the Database 33](#_Toc279663740)

[12. Domain Generation 36](#_Toc279663741)

[Purpose 36](#_Toc279663742)

[Location and Naming Convention 36](#_Toc279663743)

[Usage 36](#_Toc279663744)

[Possible Pitfalls 36](#_Toc279663745)

[13. Service Discovery 37](#_Toc279663746)

[Purpose 37](#_Toc279663747)

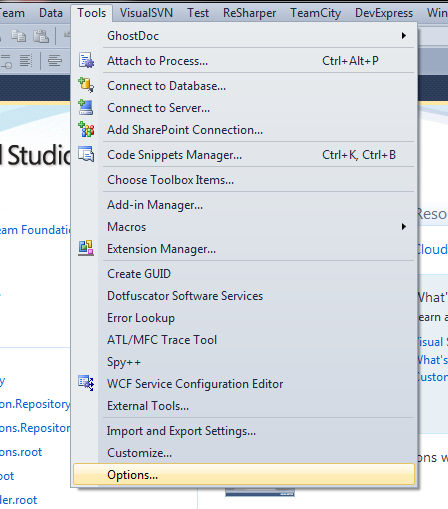
[Mechanics 37](#_Toc279663748)

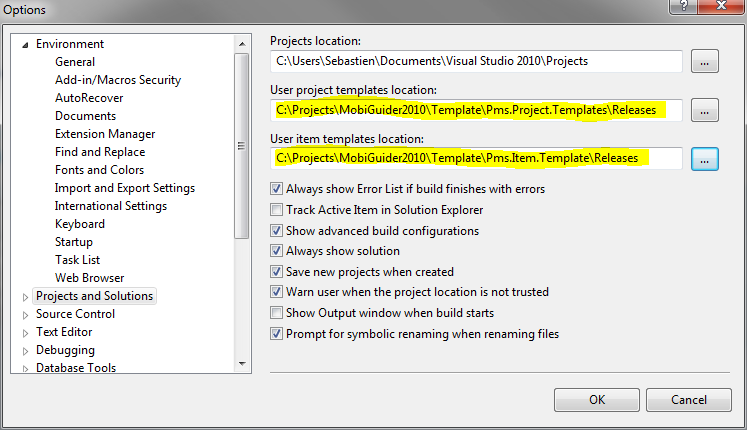
[Enabling services to be discovered 37](#_Toc279663749)

# Prepare your Visual Studio

## Visual Studio Templates

First of all, you’ll need to add all Prodata visual studio templates.

* Tools 🡪 Options  
  
* Click on Projects and Solutions



* Change "User project templates location:" to path *<MobiGuider2010>*\Template\Pms.Item.Templates\Releases
* Change "User item templates location:" to path *<MobiGuider2010>*\Template\Pms.Item.Templates\Releases

Template will be updated every time you will update Mobiguider svn folder.

# MobiGuider 2010 Codebase Structure

## Contract Folder

Purpose: This folder is the parent for all folders that contain Contract Projects  
Naming:

1. Projects that contain the Contract-layer components that are proxies to   
   ***Business Domain Services*** are named : **Pms.<BusinessDomainName>.Contract***e.g. Pms.Obstructions.Contract*
2. Projects that contain the Contract-layer components that are proxies to   
   ***Repository Services*** are named : **Pms.<RepositoryName>.Repository.Agent***e.g. Pms.Transmodel.Repository.Agent*
3. Projects that contain the Contract-layer components that are proxies to   
   ***Applet Services*** are named : **Pms.<AppletName>.Contract***e.g. Pms.ManageObstructions.Contract*

***NAMING CONVENTION NOTE: Applet Services must always have a Verb-Noun structure,***

***Mapping onto the name of the Use Case for which the applet covers the functionality.***

## Business Folder

Purpose: This Folder is the parent folder for all folders that contain Business Projects

Naming: **Pms.<BusinessDomainName>.Business or Pms.<AppletName>.Business**

Example: *Pms.Obstructions.Business or Pms.ManageObstructions.Business*

## Components Folder

Purpose: This folder is the parent folder for all folders that contain the wrapper around 3rd Party components.

Naming: **Pms.Components.<ComponentName>**

Example: *Pms.Components.ActiPro*

## Dao Folder

Purpose: This folder is the parent folder for all folders that contain the Data Access Object Projects.

Naming: **Pms.<SubDomainName>.Dao**

Example: *Pms.TransModel.Dao*

## Framework Folder

Contains the Framework Core Assemblies

## Library Folder

Contains the Framework Core Assemblies

# Hosting/Debugging WCF Services in IIS

## Preparation

Every MobiGuider developer should do the following in preparation

1. Install “**Internet Information Server 7**” (IIS7)
2. Create a **local folder** (i.e. C:\PMSroot)
3. Give Read/Write access to this folder for the” **Network Service**” user
4. Run **C:\windows\microsoft.net\framework\v3.0\Windows Communication Foundation\ServiceModelReg.exe –i**
5. Define a website in IIS and point it to the <local folder>.
6. Run this website on **port 80 (HTTP)**

## WCF Service Development Deployment

In order to deploy MobiGuider WCF Services:

1. Start Visual Studio as **“Administrator”**
2. **Rebuild** your Service Solution (for instance: Pms.Obstructions.Repository.sln)
3. **Publish**  your service solution to its designated directory. Please respect the structures and nomenclature as described in “MobiGuider Default IIS Deployment”

# Practice: MobiGuider Default IIS Deployment

## Introduction

The **MobiGuider Application** is to be deployed as a combination of IIS-hosted WCF Services and ClickOnce Client Applications. In order to obtain consistency in the development arena, the following web structure **MUST** be respected by **ALL** developers:

## Http Structure

[**http://localhost**](http://localhost)

**/services  
 /infrastructure  
 */<function name>*/*<function name>*.svc  
 /application  
 /*<application service name>*/*<application service name>*.svc  
 /business  
 /*<business service name>*/*<business service name>*.svc  
 /repository  
 /*<repository groupname>*/*<repository name>*.svc  
 /choreography  
 /*<choreography name>*/*<choreography name>*.svc**

**/clickonce**

**/Pms.<Applet Name>.UI**

Please note: **all service URL’s are in lower case**

## *Some Examples*

[**http://localhost/services/infrastructure/filetransfer/filetransfer.svc**](http://localhost/services/infrastructure/filetransfer/filetransfer.svc)[**http://localhost/services/infrastructure/discovery/discovery.svc**](http://localhost/services/infrastructure/discovery/discovery.svc)[**http://localhost/services/infrastructure/authentication/authentication.svc**](http://localhost/services/infrastructure/authentication/authentication.svc)[**http://localhost/services/application/manageobstructions/manageobstructions.svc**](http://localhost/services/application/manageobstructions/manageobstructions.svc)[**http://localhost/services/business/obstructions/obstructions.svc**](http://localhost/services/application/obstructions/obstructions.svc)[**http://localhost/services/repository/eod/tofheaderrepository.svc**](http://localhost/services/repository/eod/tofheaderrepository.svc)[**http://localhost/services/choreography/declarecost/declarecost.svc**](http://localhost/services/choreography/declarecost/declarecost.svc)[**http://localhost/clickonce/Pms.Obstructions.UI/**](http://localhost/clickonce/Pms.Obstructions.UI/)[**http://localhost/clickonce/Pms.AuthenticationManager.UI/**](http://localhost/clickonce/Pms.AuthenticationManager.UI/)

## Net.Pipe Structure

[**net.pipe://localhost**](http://localhost)

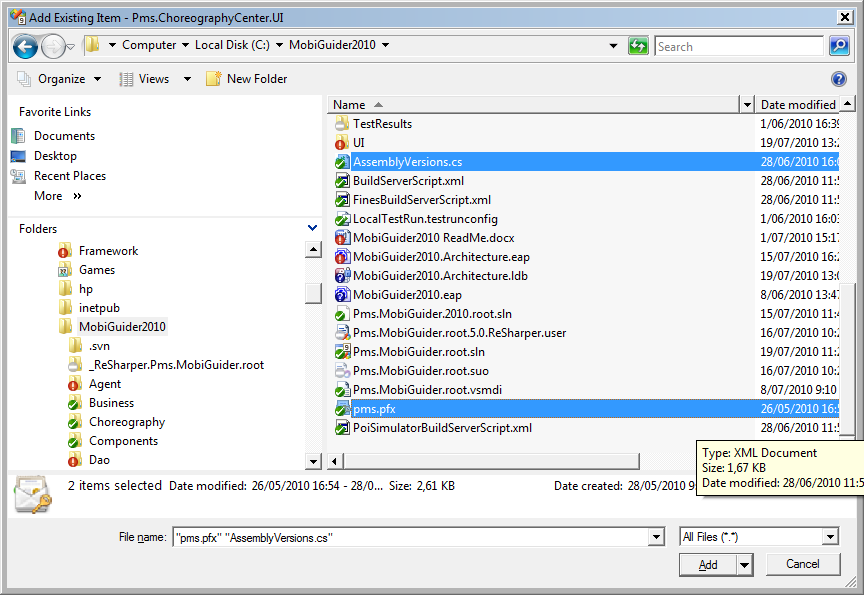
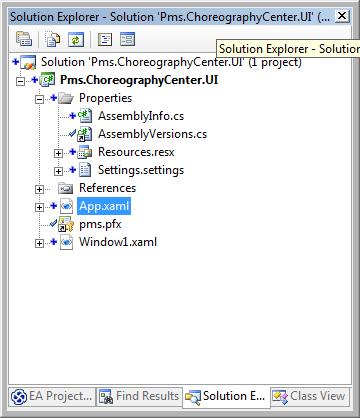
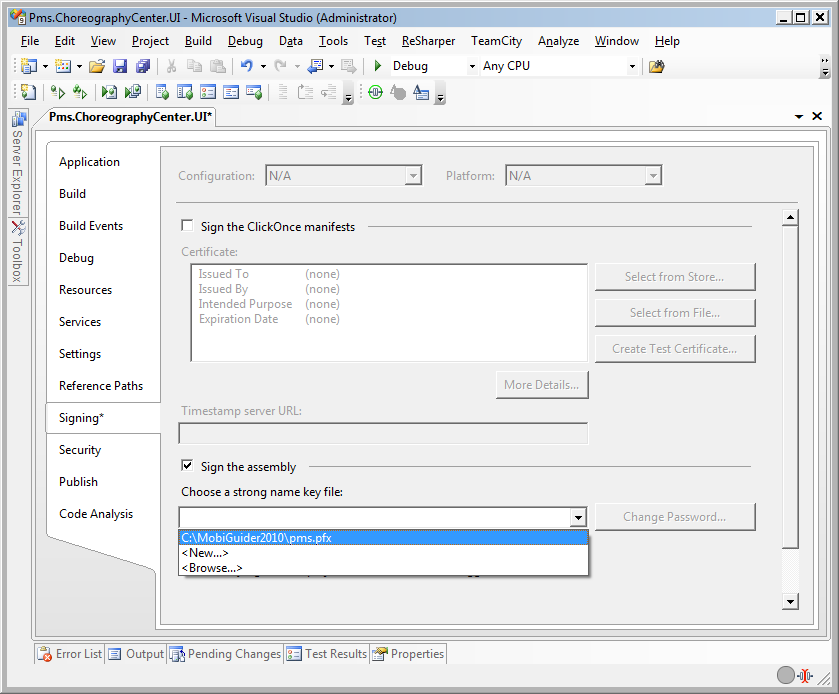
**/services  
 /infrastructure  
 */<function name>*/*<function name>*.svc  
 /application  
 /*<application service name>*/*<application service name>*.svc  
 /business  
 /*<business service name>*/*<business service name>*.svc  
 /repository  
 /*<repository groupname>*/*<repository name>*.svc  
 /choreography  
 /*<choreography name>*/*<choreography name>*.svc**

# Practice: Signing and Versioning of MobiGuider Assemblies

## Purpose

All MobiGuider Assemblies (Visual Studio Projects) MUST be signed and versioned in a similar manner:

## Steps

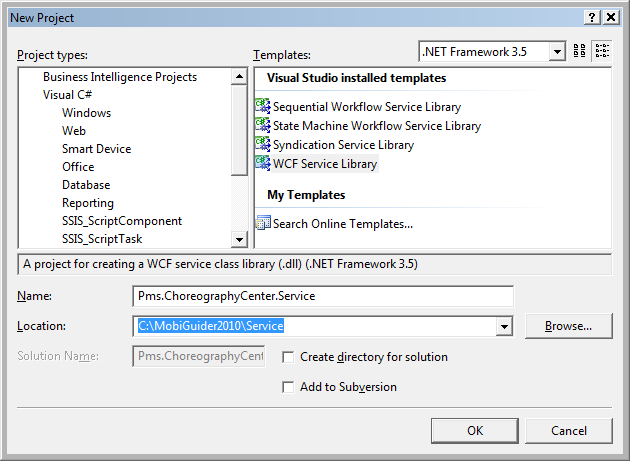
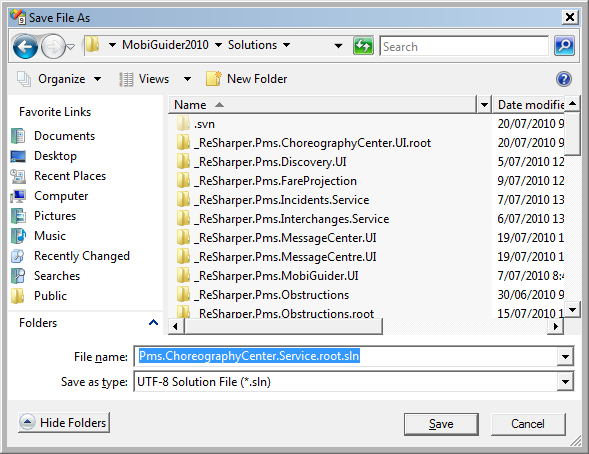
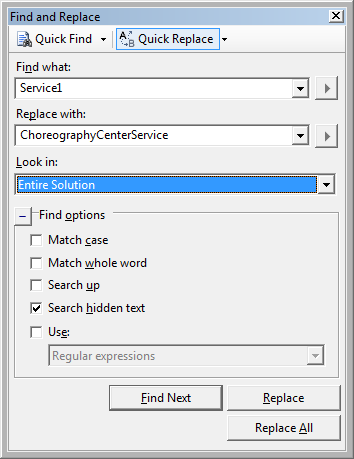
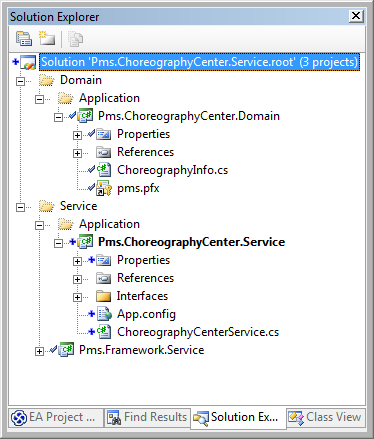
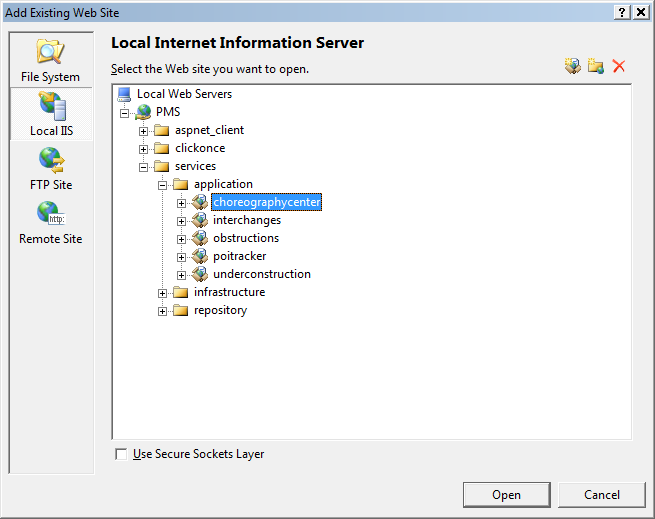
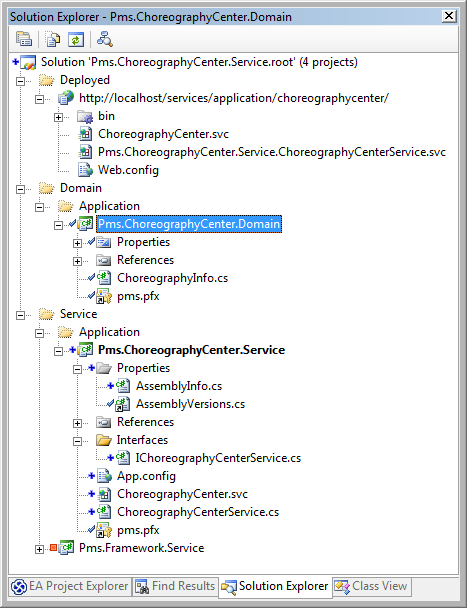
1. Select the Project that you want to Sign/Version in the VS Solution Explorer
2. Right-click on the project in the Solution Explorer and select “Add Existing Item” in the context menu
3. A file open dialog will appear.
4. Select the **\MobiGuider2010\AssemblyVersions.cs** and **\MobiGuider2010\pms.pfx** files and select **“Add as Link”**
5. This will result in shortcuts to both files being added to the project.
6. Drag the shortcut to the “**AssemblyVersions.cs**” file into the “**Properties**” folder of the project  
   
7. Open the **AssemblyInfo.cs** file that is present in the project and delete following sections:  
     
   [assembly: AssemblyCompany("\*\*\*\*\*\*\*\*")]  
   [assembly: AssemblyProduct("\*\*\*\*\*\*\*\*\*\*\*")]  
   [assembly: AssemblyCopyright("\*\*\*\*\*\*\*\*")]  
   [assembly: AssemblyTrademark("")]  
   [assembly: AssemblyCulture("")]  
   [assembly: AssemblyVersion("\*\*\*\*\*")]  
   [assembly: AssemblyFileVersion("\*\*\*\*\*\*\*\*")]
8. Right-Click on the Project File again, select “**Properties**” then the tab “**Signing**”  
   
9. Select the **\MobiGuider2010\pms.pfx** file for signing the assembly
10. Save the project.

# How to: create a MobiGuider Application Service Skeleton Solution

## Creating the Initial Application Service Solution

In MobiGuider 2010, Applets need a dedicated service. This so-called Application Service consists of a WCF service library, that serves as the façade for underlying business functionality and data access.

In order to create a Application Service from scratch, follow these steps

1. Create a new **WCF Service Library** project and call it **Pms.<AppletName>.Service.**
2. This project needs to be created in the **\MobiGuider2010\Service** directory.
3. Uncheck the “Create Directory for Solution Checkbox” and press OK to create the project  
   
4. Sign the Service assembly as described in section “Signing and Versioning of MobiGuider Assemblies”
5. Save the Project’s Solution file as **Pms.<AppletName>.Service.root.sln,** in the **\MobiGuider2010\Solutions** folder.  
   
6. Create the **Pms.<AppletName>.Service\Interfaces** project folder.
7. **Rename IService1** to **I<AppletName>Service**, and drag it into the **\Interfaces** folder
8. Move the **I<AppletName>Service** into the **Pms.<AppletName>.Service.Interfaces** Namespace
9. Rename **Service1** to **<AppletName>Service**.
10. Delete all auto-generated code from both interface and service.
11. Perform a Solution-wide **Search-and-Replace,**  replacing **Service1** with **<AppletName>Service** (7 occurences should be replaced) ****
12. Open the project’s **App.Config** file
13. Edit the Endpoint address of the service by replacing the line:  
      
     <endpoint address="" binding="wsHttpBinding" contract="Pms.ChoreographyCenter.Service.IChoreographyCenterService">  
      
    **by** <endpoint address="" binding="wsHttpBinding" contract="Pms.ChoreographyCenter.Service.**Interfaces**.IChoreographyCenterService">  
      
    This is necessary because the interface of this service was moved to that namespace.
14. Edit the line   
      
     <add baseAddress="http://localhost:8731/Design\_Time\_Addresses/Pms.ChoreographyCenter.Service/ChoreographyCenterService/" />  
      
    **and change it to**  
      
     <add baseAddress="http://localhost/services/application/choreographycenter/" />
15. Optionally change the names of some elements to create more readable identifiers.   
    (BE CAREFUL, only do this if you know what you are doing!)
16. If the project **Pms.<AppletName> .Domain** already exists, add it to the solution. If not; please follow the steps in the section “**Creating a MobiGuider Application Domain Assembly**”
17. Add the **\MobiGuider2010\Service\Pms.Framework.Service** project to the solution.
18. Example: (Solution folder as per convention)  
      
    
19. Build the solution.
20. If successful, publish the application to its standard deployment location (see section  
    ”MobiGuider Default IIS Deployment”)
21. Add a Solution folder “**Deployed**” to your solution
22. In this folder, perform an “**Add Existing Website**” action, adding the web site you have just published for the service  
      
    
23. From this Website,select the **Pms.<AppletName>.Service.<AppletName>Service.svc** file and copy it to **Pms.<AppletName>.Service** project**. Rename** this file to **<AppletName>.svc**
24. **Publish** the service library again. Your solution should look like this:  
      
    

## Extending theApplication Service Solution

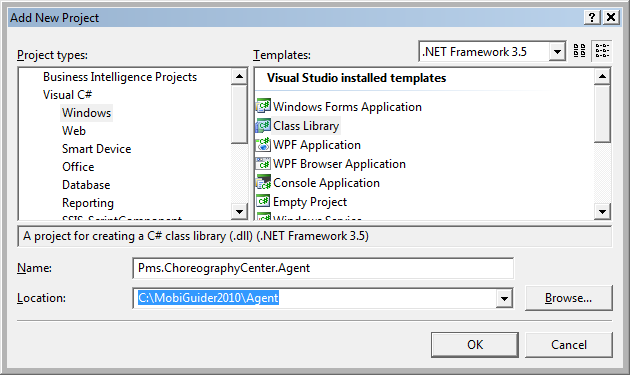
TODO

# How to: create a MobiGuider Application Service Agent

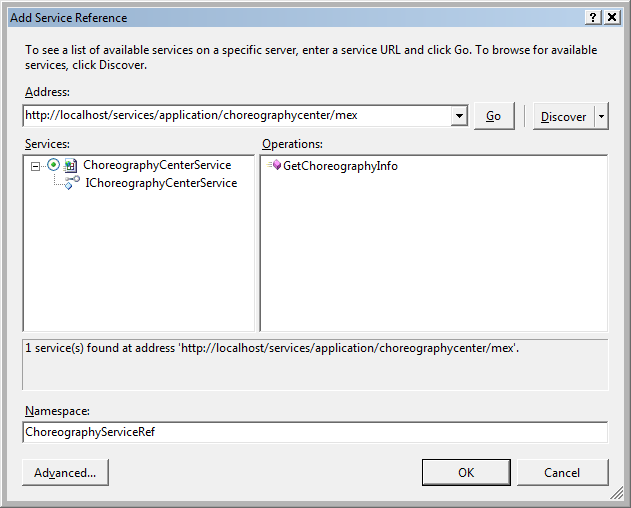
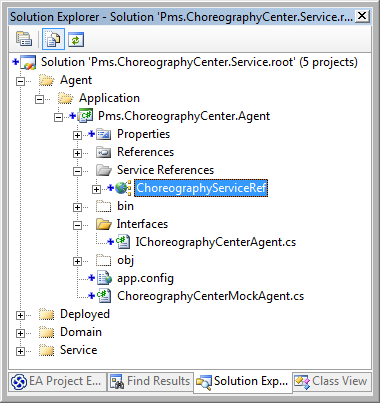
## Purpose

“MobiGuider Services may only be accessed (from other MobiGuider code) through their Agents” This statement implies, that there is an Agent for every Service created. It is the Service developer’s responsibitlity to provide this Agent. Agent classes are wrappers around the actual service clients that are generated using **svcutil.exe**, this way Agent classes hide the nuts and bolts of WCF communication and provide an opportunity to disconnect Service and UI development teams. A workable scenario in common practice is the creation of a MockAgent class, against which the major part of the UI development cycle can occur.

## Creating the Agent Assembly

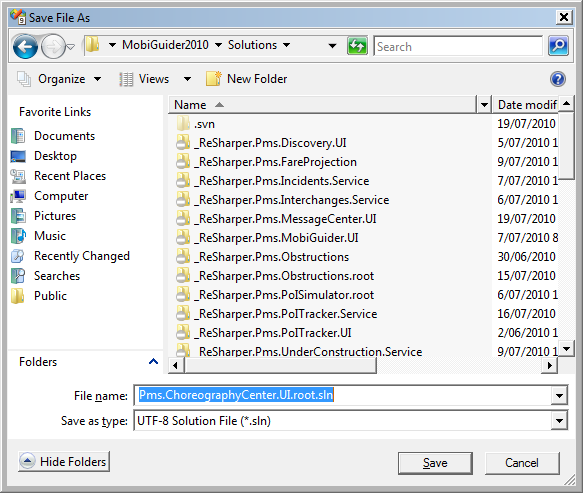
1. Open the solution containing the Application Service for which you want to create the Agent.
2. Create the **.root\Agent\Application** solution folders
3. Under this folder, create a new “Class Library” Project and name it **Pms.<AppletName>.Agent  
   **
4. Sign and Version the **Pms.<AppletName>.Agent** assembly according to the section “Versioning and Signing Mobiguider Assemblies”
5. Add a **$project$\Interfaces** folder to the Agent project.
6. Create a **I<AppletName>Agent.cs** interface and a **<AppletName>MockAgent.cs** class to the Agent Assembly.

# Adding Service References to an Agent Assembly

1. In the Agent Assembly, right-click on the project and click “**Add Service Reference**”.
2. In the Add Service Reference Dialog, type the address of the service you wish to reference.   
   For the MobiGuider2010 framework, this means you can find the service at following location: [**http://localhost/services/application/<appletname>/**](http://localhost/services/application/%3cappletname%3e/).   
   
3. Press “Go”
4. In the namespace box change the default to : **<AppletName>ServiceRef**
5. Press OK, the service reference will be added to the Agent project and the project will look like this:  
     
   

# How to create a MobiGuider User Interface Solution Skeleton

## Creating the Initial Solution

1. **Determine the name** of the Applet. This will result in an <AppletName>.   
   Example: “ChoreographyCenter”
2. Create a new Project of type **“WPF Application”**
   1. name it **Pms.<AppletName>.UI.**
   2. This project needs to be placed directly under the **\MobiGuider2010\UI** folder
   3. Uncheck the “Create Directory for Solution” checkbox
   4. Example screenshot  
        
      
   5. Execute the steps as described in section “Practice: Signing and Versioning of MobiGuider Asseblies”
   6. Save the created solution as **\MobiGuider2010\Solutions\Pms.<AppletName>.UI.root.sln**
   7. In our example: **Pms.ChoreographyCenter.UI.root.sln**
   8. Create a Solution Folder “**UI**” in the solution and drag the **Pms.<AppletName>.UI** project in it.
   9. Optional: Create a “**Solution Items**” folder and add the documents **MobiGuider2010.ReadMe.docx**  and **MobiGuider2010.Architecture.eap**

## Extending the UI Solution

### The Application Agent Assembly

#### Purpose

Every MobiGuider Applet (Web or Windows) has a corresponding Application Service, running in the backend. Sincethe MobiGuider framework imposes that all access to services MUST ONLY occur by their agents. This means that every Application Service MUST HAVE a corresponding Application Agent, which will addressed by the UI. In normal operations, the Application Agent will be an existing project, as it is the responsibility of the Application Service Implementer to provide and maintain this assembly. HOWEVER, daily practice will impose the need for the UI development team to “have an interface to talk to”. This means that in practice, the Application Agent Assembly will be

#### Creation Steps

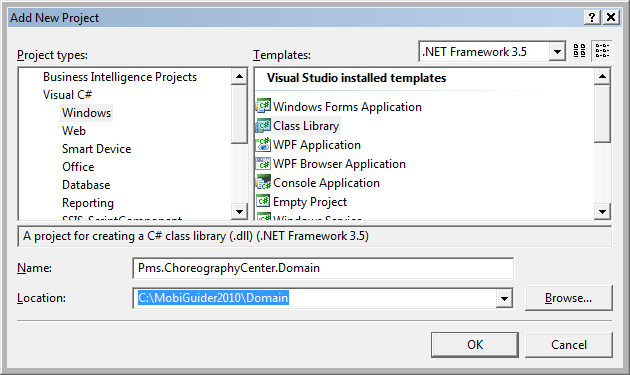
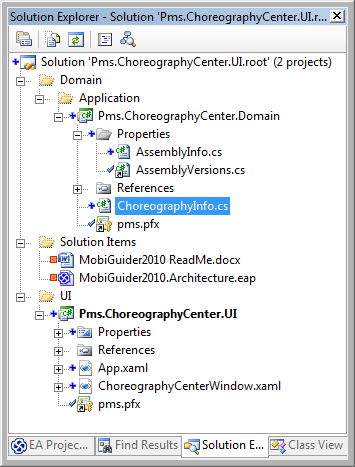
# Creating an Application Domain Assembly

#### Purpose

Every MobiGuider Solution will eventually need some data to work with. The classes that represent this data are called (application) domain classes and are contained in an assembly called **Pms.<AppletName>.Domain**. Classes in this domain are POCO’s (Plain Old C# Objects) and are best defined without any attributes.

In order to create an Application Domain Assembly follow these

#### Creation Steps

1. Add a new Solution Folder called **\Domain\Application**
2. Add to this folder a new “Class Library” Project
3. Call this project “**Pms.<AppletName>.Domain**” and store it in the **\MobiGuider2010\Domain** directory  
   
4. Sign and Version the Assembly using the practice as described in section Signing and Versioning of MobiGuider Assemblies
5. In our example: **Pms.ChoreographyCenter:Domain**
6. Final result:  
   

# How to: create a MobiGuider Database

## Purpose

Every MobiGuider Solution will eventually need a Database to work with. The project will keep the database schema versioned. The database project can be created from scratch or from an existing Db in MSSQL.

## Database conventions

Pascal case means that the first letter in each word in an [identifier](http://cplus.about.com/od/glossar1/g/identifier.htm) is capitalized.  
Camel case is similar except the first letter is in lower case.  
In the database all names are Pascal case.

**Schema**

* Name: PascalCase; Is the same as the repository name
* Filename and location: *<DatabaseProjectLocation>*\Schema Objects\Database Level Objects\Security\Schemas\***<SchemaName>***.schema.sql
* Example:

CREATE SCHEMA [Fines]

**Table**

All tables should be inserted in the repository schema (see schema).

* Table Name: PascalCase
* Filename and location: *<DatabaseProjectLocation>*\Schema Objects\Schemas\*<SchemaName>*\Tables*\<tablename>*.table.sql
* Primary key columns:
  + Primary Key Name: PascalCase; PK\_*<TableName>*
  + Column name: Id (not ID or Id*<TableName>*)
  + Filename and location: *<DatabaseProjectLocation>*\Schema Objects\ Schemas\*<SchemaName>*\Tables\Keys\***<TableName>*.*<PrimaryKeyName>***.pkey.sql
  + Column is NOT NULL.
  + Constraint will not be added in the table script file.
  + All Primary key column are from type varchar(50).
  + Example:  
    ALTER TABLE [Obsctruction].[Detour]  
     ADD CONSTRAINT [PK\_Detour]   
     PRIMARY KEY CLUSTERED ([Id] ASC)
* Foreign key columns:
  + Foreign Key Name: PascalCase; FK\_*<TableName>*\_*<ForeignkeyColumnName>*  
    *<ForeignkeyColumnName>* is the name of the column without Id at the end.
  + Column name: *<ReferencedTableName>*Id (not Id*<ReferencedTableName>*)
  + Filename and location: *<DatabaseProjectLocation>*\Schema Objects\ Schemas\*<SchemaName>* \Tables\Keys\***<TableName>*.*<ForeignKeyName>***.fkey.sql
  + Constraint will not be added in the table script file.
  + Example1:  
    ALTER TABLE [Obstruction].[Detour]  
     ADD CONSTRAINT [FK\_Detour\_Obstruction]   
     FOREIGN KEY ([ObstructionId])  
     REFERENCES [Obstruction].[Obstruction] ([Id])
  + Example 2:  
    ALTER TABLE [Obstruction].[Detour]   
     ADD CONSTRAINT [FK\_Detour\_ParentDetour]   
     FOREIGN KEY([ParentDetourId])  
     REFERENCES [Obstruction].[Detour] ([Id])
* Example:   
  CREATE TABLE [Obstruction].[Detour](  
   [Id] varchar(50) NOT NULL,  
   [Description] varchar(500) NULL,  
   [StatusId] varchar(50) NULL,  
   [VersionNumber] varchar(50) NULL,  
   [ObstructionId] varchar(50) NOT NULL,  
   [ParentDetourId] varchar(50) NULL,  
   [UserAssignedIdentity] int NULL)

**Stored Procedure**

All stored procedures should be inserted in the repository schema (see schema).

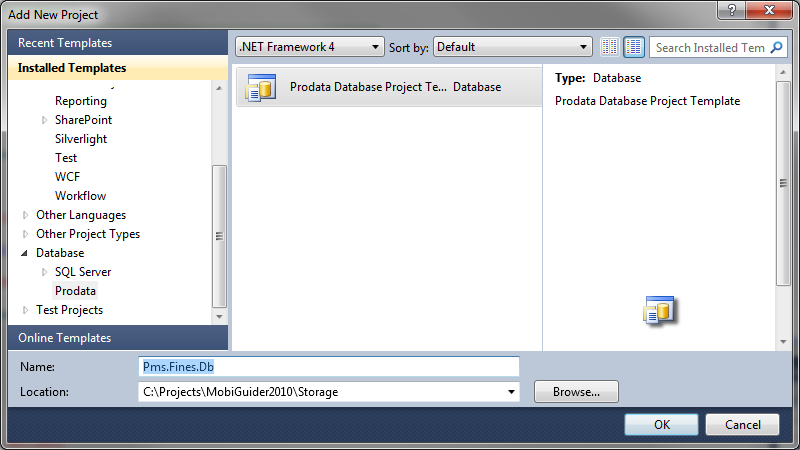
* Stored Procedure Name:
  + PascalCase
  + For multiple row result set: *<SchemaName>*.*<TableName>*\_*<OperationName>*  
    ex:[Fines].[FineWithPassenger\_SelectByDateRange]
  + For 1 row result set:

*<SchemaName>*.*<TableName>*\_SelectFirst*<OperationName>*  
ex:[Fines].[FineWithCredit\_SelectFirstByCardNumber]

* Filename and location: *<DatabaseProjectLocation>*\Schema Objects\Schemas\ *<SchemaName>*\Programmability\Stored Procedures\***<StoredProcedureName>***.proc.sql
* Parameters: PascalCase
* Local Variable: PascalCase

## Creating the database project

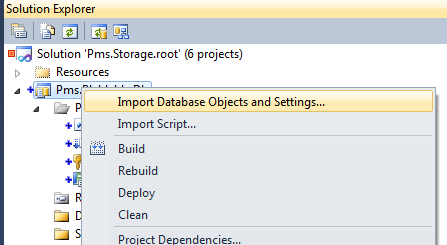
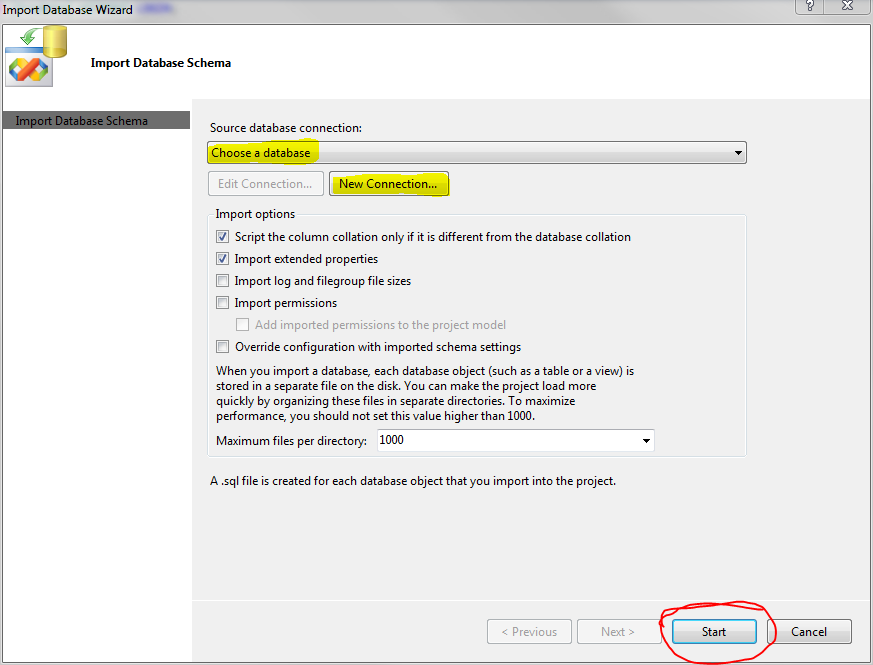
1. Open the solution “.root\Storage\Pms.Storage.root.sln”.
2. Add a new “Prodata Database project Template” Project and name it “**Pms.*<RepositoryName>*.Db”**   
   The Repository is a functional domain and can be used by several applets.

****

## Import Database schema from an existing DB

1. Create the database project like it is explained in “Creating the database project”

Open the properties of the following files:  
- Schema Objects\Database Level Objects\Security\Schemas\ Prodata.schema.sql  
- Schema Objects\Schemas\Prodata\Tables\Audit.table.sql  
- Schema Objects\Schemas\Prodata\Tables\Patch.table.sql  
- Schema Objects\Schemas\Prodata\Tables\Constraints\Audit.DF\_Audit\_Id.defconst.sql  
- Schema Objects\Schemas\Prodata\Tables\Keys\Audit.PK\_Audit.pkey.sql  
- Schema Objects\Schemas\Prodata\Tables\Keys\Patch.PK\_Patch.pkey.sql  
- Schema Objects\Schemas\Prodata\Programmability\Stored Procedures\Audit\_Insert.proc.sql  
- Schema Objects\Schemas\Prodata\Programmability\Stored Procedures\CompletePatch.proc.sql  
- Schema Objects\Schemas\Prodata\Programmability\Stored Procedures\StartPatch.proc.sql

1. Change Build Action from these files to “Not in Build”
2. Rename an existing database to“**Pms.*<RepositoryName>*.Db”**
3. Open “Import Database Objects and Settings…”
4. Choose the existing database where you want to import from. (If necessary, use “New Connection…”)
5. Click on “Start”
6. Reopen the properties of the same files:

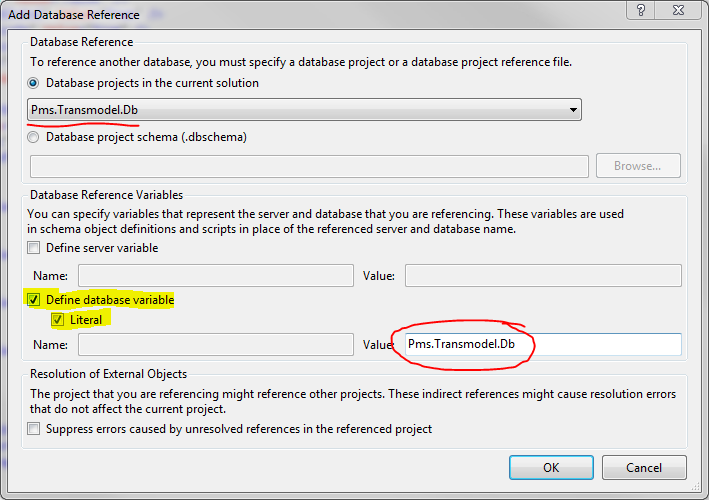
- Schema Objects\Database Level Objects\Security\Schemas\ Prodata.schema.sql  
- Schema Objects\Schemas\Prodata\Tables\Audit.table.sql  
- Schema Objects\Schemas\Prodata\Tables\Patch.table.sql  
- Schema Objects\Schemas\Prodata\Tables\Constraints\Audit.DF\_Audit\_Id.defconst.sql  
- Schema Objects\Schemas\Prodata\Tables\Keys\Audit.PK\_Audit.pkey.sql  
- Schema Objects\Schemas\Prodata\Tables\Keys\Patch.PK\_Patch.pkey.sql  
- Schema Objects\Schemas\Prodata\Programmability\Stored Procedures\Audit\_Insert.proc.sql  
- Schema Objects\Schemas\Prodata\Programmability\Stored Procedures\CompletePatch.proc.sql  
- Schema Objects\Schemas\Prodata\Programmability\Stored Procedures\StartPatch.proc.sql

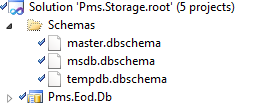
1. Change back the Build Action from to “Build”
2. Remove unused users in “Schema Objects\Database Level Objects\Security\Users”
3. Remove unused users in script   
   “Schema Objects\Pms.*<RepositoryName>*.Db.rolememberships.sql”
4. Edit users scripts in “Schema Objects\Database Level Objects\Security\Users”  
   Change “CREATE USER [<user>] FOR LOGIN [<user>]”   
   to “CREATE USER [<user>] WITHOUT LOGIN”

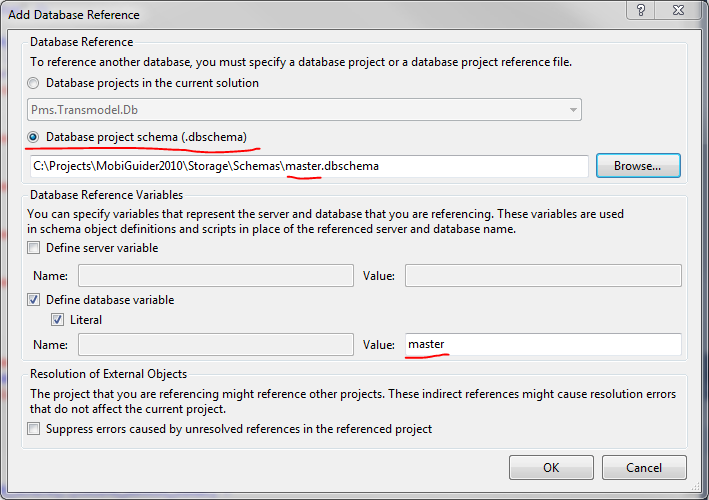
## Adding Database References

It is possible to references other databases from your database project. There are 2 different ways to reference another database from your database project.  
Referencing another database project or schema from your database project is by default a **bad practice,** except for whenyou are referencing the master, msdb or tempdb schemas.  
The good practice is to reference the repository service of the database in your application layer.

1. Referencing a database managed by us (database included in Storage solution).
2. Add Database Reference…
3. Choose the database you want to reference.
4. Check “Define database variable” and “Literal”.
5. Fill in the “Value” box the same name as the referenced database.  
   This name can be used in Stored Procedure, View, etc to reference the database.



1. Referencing a database managed by a 3rd party.
2. First you will need the database schema file (\*.dbschema) from the 3rd party database.
3. Add this file in “MobiGuider2010\Storage\Schemas “
4. Include this schema in the solution folder “Schemas” of the Storage solution.
5. Add Database Reference…
6. Check radio button “Database project schema (.dbschema)”  
   and browse to your schema file.
7. The literal database variable value will automatically be filled with the same name as your file. This name can be used in Stored Procedure, View, etc to reference the database.



## Pre/Post Build scripts

**Usage:** Changing database schema with data transformation.

When a change in the database needs to transform data, you will need to write some query’s in the “<Pre/Post >–Deployment/Releases/Sprint\_<sprint number>.sql” script.  
Write a complete description from your change in the “Pre-Deployment/Releases/Sprint\_<sprint number>.sql”. (See example)

Example:   
Two new columns are added to a table:

* The 5 first characters of the “Description” column need to be filled into a new column “Code”.
* The remaining characters of the “Description” column need to be filled into a new column “Name”.
* The “Description” column needs to be deleted.

Pre-Deployment/Releases/Sprint\_10.17.sql

-- ========================================================================================

-- || Number: 10.17.1 ||

-- || Date: 19/10/2010 ||

-- || Developer: Sébastien Van De Poel ||

-- || Altered items: Fines.OfficeType ||

-- || SprintPatch that depends on this sprintPatch: None ||

-- || SprintPatch on which this SprintPatch depends: 10.17.2 ||

-- || Depends on: None ||

-- || Description: Change "Name" column to "Code" and "Description" ||

-- ========================================================================================

-- ========================================================================================

-- || Number: 10.17.2 ||

-- || Date: 25/10/2010 ||

-- || Developer: Rafael Lefever ||

-- || Altered items: table1, view5, StoredProcedure5 ||

-- || Depends on this sprintPatch: 10.17.1 ||

-- || On which this SprintPatch depends: None ||

-- || Description: yadayadayada yada yada ||

-- ========================================================================================

-- ===================================Start 10.17.1============================================

-- save table into a tempory table

select Id, [Description], IsDeleted into #OfficeTypeTmpTable

from Fines.OfficeType

-- ===================================End 10.17.1============================================

Post-Deployment/Releases/Sprint\_10.17.sql

-- ==================================Start 10.17.1===========================================

set identity\_insert fines.OfficeType on

-- insert data in new table

insert into Fines.OfficeType(Id, Code, Name, IsDeleted)

select Id, substring([Description], 0, 3), substring([Description], 3, 255) , IsDeleted

from #OfficeTypeTmpTable

set identity\_insert fines.OfficeType off

-- delete temp table

drop table #OfficeTypeTmpTable

-- enable checks

ALTER TABLE [Fines].[Steward] WITH CHECK CHECK CONSTRAINT [FK\_Steward\_OfficeType];

-- ===================================End 10.17.1============================================

-- ==================================Start 10.17.2===========================================

-- some query's

-- ===================================End 10.17.2============================================

## Update, patch or release a new version of the Database

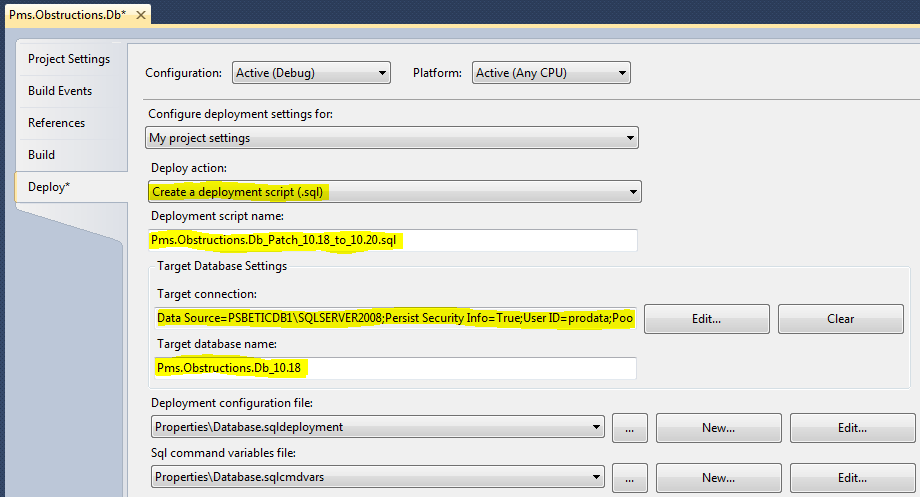
**Only your script may be committed to source control.  
All other changes must be reverted!!!**

**First**: you’ll need to update the project command variables.  
These variables can be found in “Database.sqlcmdvars” file under the “properties” folder.

* **$(ScriptType):** Possible values are “Dev”, “Initial” and “Patch”.  
  Every database will keep all executed scripts in the “Patch” table.
  + “Dev”: Use this option is you want to update or create your Database without logging it in the Patch table. (No rows will be inserted in the Patch table.)
  + “Initial”: Use this option to create a new database. The target database name should not exist in the database.
  + “Patch”: Use this option to update your database.
* **$(DatabaseType):** (Obsolete when ScriptType is “Dev”)  
  Type of Database on which it will be released.   
  Normally it will always be on “Release”.  
  **$(VersionName):** The name of the script. (Obsolete when ScriptType is “Dev”)
* **$(VersionDescription):** A description of the script. (Obsolete when ScriptType is “Dev”)
* **$(MajorVersionId):** The major version id is an integer. (Obsolete when ScriptType is “Dev”)

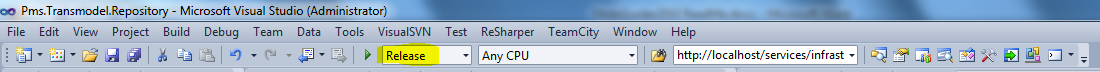
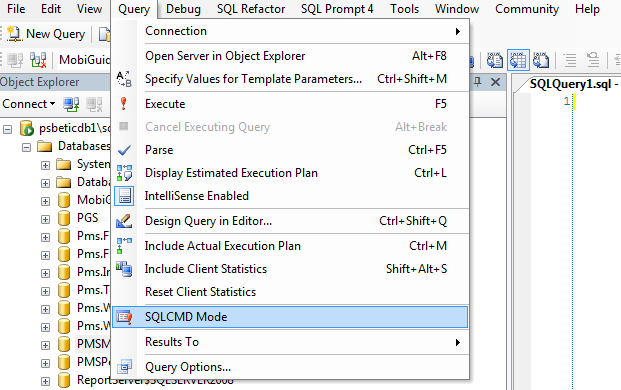
When ScriptType is “Sprint”, insert the 2 digit year.

* **$(MinorVersionId):** The minor version id is an integer. (Obsolete when ScriptType is “Dev”)  
  When ScriptType is “Sprint”, insert the 2 digit sprint number.

**Second**: you’ll need to update your target database.   
The database project will compare his schema with the target database schema. Every difference will be included in the generated script.  
  


Open the database project properties and change the following Deploy properties:

* Change the “Deploy action” to “Create a deployment script (.sql)”.
* When ScriptType is Initial:
  + Change the “Deployment script name” to  
     “Pms.*<RepositoryName>*.Db\_*<ScriptType>*\_*<MajorVersionId>*.*<MinorVersionId>*.sql”
  + clear the “Target connection”
  + “Target database name” has no importance
* When ScriptType is Patch:
  + Change the “Deployment script name” to “Pms.*<RepositoryName>*.Db\_*<ScriptType>*\_*<PreviousMajorVersionId>*.*<PreviousMinorVersionId>*\_To\_*<MajorVersionId>*.*<MinorVersionId>*.sql”
  + Edit the “Target connection” to the server where the previous database version is located.
  + Change “Target database name” to the name of the previous database version.

**Third**: Change your solution configuration to Release  
  
**Fourth**: Deploy and execute the generated script in management studio.  
The script location is “*<DatabaseProjectLocation>*\sql\release\*<Deployment script name>*.sql”.  
**To execute it in management studio you’ll need to enable “sqlcmd” (ALT Q M)**.

**Fifth**: Commit your script in source control.

* Revert all your deployment changes from source control.
* Include the script into your database project.
* Commit your deployment changes to source control.  
  (Only the project file and your script will be committed to source control)

Development  
ScripType = Dev

Development  
ScripType = Dev

Development  
ScripType = Dev

Development  
ScripType = Dev

Development  
ScripType = Dev

# Domain Generation

## Purpose

It is often required that your application will need Mock Data or require certain records to be created in the Database in order to have the application function propertly. For this purpose, we will always include a so called “DomainGenerator” class.

## Location and Naming Convention

The preferred place where we will create this class is in the **Pms.<AppletName>.Domain** (i.e. Pms.WorkspaceBrowser.Domain)and we will name it **<AppletName>DomainGenerator.** (i.e. WorkspaceBrowserDomainGenerator)

## Usage

A DomainGenerator class is a class that has a number of (static) methods that return structures of pre-filled Domain objects. These objects may contain mock data, in which case the method that returns this data will start like **GenerateMock…..()**, or they may contain valid data needed for the correct operation of your applet. In this case the name of the method could be something like **GenerateRequired….().** The generated data objects will then be inserted into the database using methods on the BUSINESS Service.

## Possible Pitfalls

It could be tempting to have the DomainGenerator return Repository Domain classes…of course, this should NEVER be the case, since we do not want to introduce a dependency between client-side code and server side objects.

# Service Discovery

## Purpose

Service Discovery is an essential part of any Service Oriented Architecture. Without it, services would have to know where all services reside, resulting in configuration hell. WCF supports the WS-Discovery standard. This section will elaborate on this topic and how it is implemented in the MobiGuider framework.

## Mechanics

WS Discovery in WCF comes in 2 flavors: **Ad Hoc** and **Managed** mode. Ad Hoc WS-Discovery is suitable for intranet applications, whereas Managed mode uses a Discovery Proxy, making it suitable for the internet. MobiGuider uses the Managed mode scenario in order to be open for both intranet- and internet-based solutions.

MobiGuider Applets are WPF desktop applications that rely on services in the backend. In order to minimize the configuration complexity, these applets will first query a dedicated Discovery Service, which in turn will forward the Find request to the Discovery Proxy.   
The Discovery Proxy service is an AlwaysRunning service that implements 2 endpoints:   
an **Announcement** interface and a **Find** (a.k.a. Probe) interface.

The Announcement interface implements the “Hello” and “Bye” methods. These methods are called by discoverable services when they enter or leave the network respectively.

The Find interface implements methods that enable the Discovery Service to query the Proxy Cache for known services.

## Enabling services to be discovered

Service discovery can be fully defined using configuration. Following is an example of a web.config file containing the specific entries enabling the service to be discovered.

## Default Discovery Configuration

### Setting up IIS 7.5 to enable WS-Discovery

Step 1: Open the IIS 7.5

