



PViMS Programmers' Manual

July 2018



USAID
FROM THE AMERICAN PEOPLE

SIAPS 
Systems for Improved Access
to Pharmaceuticals and Services

This report is made possible by the generous support of the American people through the US Agency for International Development (USAID), under the terms of cooperative agreement number AID-OAA-A-11-00021. The contents are the responsibility of Management Sciences for Health and do not necessarily reflect the views of USAID or the United States Government.

About SIAPS

The goal of the Systems for Improved Access to Pharmaceuticals and Services (SIAPS) Program is to assure the availability of quality pharmaceutical products and effective pharmaceutical services to achieve desired health outcomes. Toward this end, the SIAPS result areas include improving governance, building capacity for pharmaceutical management and services, addressing information needed for decision-making in the pharmaceutical sector, strengthening financing strategies and mechanisms to improve access to medicines, and increasing quality pharmaceutical services.

Recommended Citation

This report may be reproduced if credit is given to SIAPS. Please use the following citation.

SIAPS Program. 2018. *PViMS Programmers' Manual*. Submitted to the US Agency for International Development by the Systems for Improved Access to Pharmaceuticals and Services (SIAPS) Program. Arlington, VA: Management Sciences for Health.

Systems for Improved Access to Pharmaceuticals and Services
Pharmaceuticals and Health Technologies Group
Management Sciences for Health
4301 North Fairfax Drive, Suite 400
Arlington, VA 22203 USA
Telephone: 703.524.6575
Fax: 703.524.7898
E-mail: phtmis@msh.org
Website: www.siapsprogram.org

Contents

1	Program Model and Design	4
1.1	Data Access Methodology	4
1.1.1	Object-Relational Mapping	4
1.1.2	Entity Framework	4
1.1.3	LINQ to SQL	5
1.1.4	Code First design	6
1.2	Design Patterns	6
2	Development Environment	9
2.1	Integrated Development Environment (IDE)	9
2.2	Source Control	10
2.2.1	PViMS GitHub Repository	10
2.2.2	GitHub Extension for Visual Studio	11
2.3	NuGet Packages	12
2.4	Automated Code Formatting	15
3	Deployment	16
3.1	Preparing a package for deployment	16

1 Program Model and Design

This section will give an overview of the data access method as well as patterns used within the development of the system

1.1 Data Access Methodology

Database access within PViMS is facilitated through Entity Framework v6.1.1.

1.1.1 Object-Relational Mapping

Object-relational mapping is a programming technique for converting data between incompatible type systems in object-oriented programming languages. This creates, in effect, a virtual object database that can be used from within the programming language.

1.1.2 Entity Framework

Entity Framework (EF) is an open source ORM framework for ADO.NET and is part of the DotNet Framework. It is a set of technologies that supports the development of data-oriented software applications. EF enables developers to work with data in the form of domain-specific objects and properties, such as customers and customer addresses, without having to concern themselves with the underlying database tables and columns where this data is stored. With EF, developers can work at a higher level of abstraction when they deal with data and can create and maintain data-oriented applications with less overhead.

Architecture is as follows:

- **Data source specific providers**, which abstract the ADO.NET interfaces to connect to the database when programming against the conceptual schema
- **Map provider**, a database-specific provider that translates the Entity SQL command tree into a query in the native SQL language of the database. It includes the Store-specific bridge, which is the component responsible for translating the generic command tree into a store-specific command tree
- **EDM parser and view mapping**, which takes the SDL specification of the data model and how it maps onto the underlying relational model and enables programming against the conceptual model. From the relational schema, it creates views of the data corresponding to the conceptual model. It aggregates information from multiple tables in

order to aggregate them into an entity, and splits an update to an entity into multiple updates to whichever table(s) contributed to that entity

- **Query and update pipeline**, processes queries, filters and updates requests to convert them into canonical command trees which are then converted into store-specific queries by the map provider
- **Metadata services**, which handle all metadata related to entities, relationships and mappings
- **Transactions**, to integrate with transactional capabilities of the underlying store. If the underlying store does not support transactions, support for it needs to be implemented at this layer
- **Conceptual layer API**, the runtime that exposes the programming model for coding against the conceptual schema. It follows the ADO.NET pattern of using Connection objects to refer to the map provider, using Command objects to send the query, and returning *EntityResultSets* or *EntitySets* containing the result
- **Disconnected components**, which locally cache datasets and entity sets for using the ADO.NET Entity Framework in an occasionally connected environment
- **Embedded database**: ADO.NET Entity Framework includes a lightweight embedded database for client-side caching and querying of relational data.
- **Design tools**, such as Mapping Designer, are also included which simplifies the job of mapping a conceptual schema to the relational schema and specifying which properties of an entity type correspond to which table in the database
- **Programming layer**, which exposes the EDM as programming constructs which can be consumed by programming languages
- **Object services**, automatically generate code for CLR classes that expose the same properties as an entity, thus enabling instantiation of entities as .NET objects
- **Web services**, which expose entities as web services
- **High-level services**, such as reporting services which work on entities rather than relational data

1.1.3 LINQ to SQL

LINQ to SQL allows LINQ to be used to query the PViMS database. Since SQL Server data may reside on a remote server, and because SQL Server has its own query engine, it does not use the query engine of LINQ. Instead, it converts a LINQ query to an SQL query that is then sent to SQL Server for processing. However, since SQL Server stores the data as relational data and LINQ works with data encapsulated in objects, the two representations must be mapped to one

another. For this reason, LINQ to SQL also defines a mapping framework. The mapping is done by defining classes that correspond to the tables in the database and containing all or a certain subset of the columns in the table as data members.

1.1.4 **Code First design**

PViMS is developed using a Domain Driven Design which is effectively a collection of principles and patterns that lead to software abstractions called domain models. These models encapsulate all complex PViMS business logic, closing the gap between reality and code. Domain driven design is about mapping business domain concepts into software artefacts, using the same common ubiquitous language.

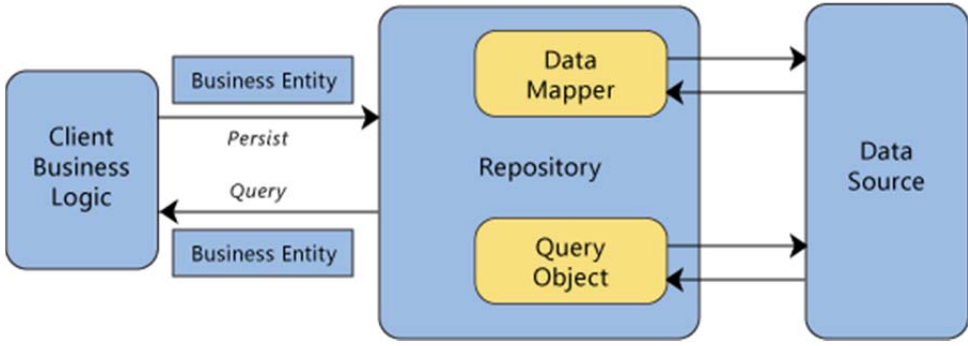
Code-First is mainly used in Domain Driven Design and means that PViMS is developed through a Code-First approach. The PViMS data model is defined using C# classes and additional configuration can be performed using attributes within these classes. Model changes are committed to the PViMS database using Code-First Migrations.

Migrations allows us to have an ordered set of steps that describe how to upgrade (and downgrade) our PViMS database schema

1.2 **Design Patterns**

A software design pattern is a general reusable solution to a common occurring problem within a given context. Design patterns are formalized best practices that describe the problem, the solution, when to apply the solution and its consequences, intended or unintended. In the case of PViMS, the following design patterns have been implemented:

Pattern	Description
MVC	The Model-View-Controller (MVC) architectural pattern separates an application into three main components: the model, the view, and the controller. The ASP.NET MVC framework provides an alternative to the ASP.NET Web Forms pattern for creating Web applications. The ASP.NET MVC framework is a lightweight, highly testable presentation framework that (as with Web Forms-based applications) is integrated with existing ASP.NET features, such as master pages and membership-based authentication. The MVC framework is defined in the System.Web.Mvc assembly.

<p>Repository and Unit of Work</p>	<p>A repository pattern separates the logic that retrieves the data and maps it to the entity model from the business logic that acts on the model. The business logic should be agnostic to the type of data that comprises the data source layer. The repository mediates between the data source layer and the business layers of the application. It queries the data source for the data, maps the data from the data source to a business entity, and persists changes in the business entity to the data source. A repository separates the business logic from the interactions with the underlying data source or Web service. There are two ways that the repository can query business entities. It can submit a query object to the client's business logic or it can use methods that specify the business criteria. In the latter case, the repository forms the query on the client's behalf. The repository returns a matching set of entities that satisfy the query.</p> <p>The following diagram shows the interactions of the repository with the client and the data source. The separation between the data and business tiers has three benefits:</p> <ul style="list-style-type: none"> • It centralizes the data logic or Web service access logic • It provides a substitution point for the unit tests • It provides a flexible architecture that can be adapted as the overall design of the application evolves  <pre> graph LR C[Client Business Logic] -- Persist --> R[Repository] R -- Query --> C subgraph Repository DM[Data Mapper] QO[Query Object] end R <--> DS[Data Source] DM <--> DS QO <--> DS </pre>
<p>Dependency Injection</p>	<p>Dependency Injection (DI) is a design pattern that demonstrates how to create loosely coupled classes. DI is where one object supplies the dependencies of another object. A dependency is an object that can be used (a service). An injection is the passing of a dependency to a dependent object (a client) that would use it. The intent behind dependency injection is to decouple objects to the extent that no client code has to be changed simply because an object it depends on needs to be changed to a</p>

	different one.
--	----------------

2 Development Environment

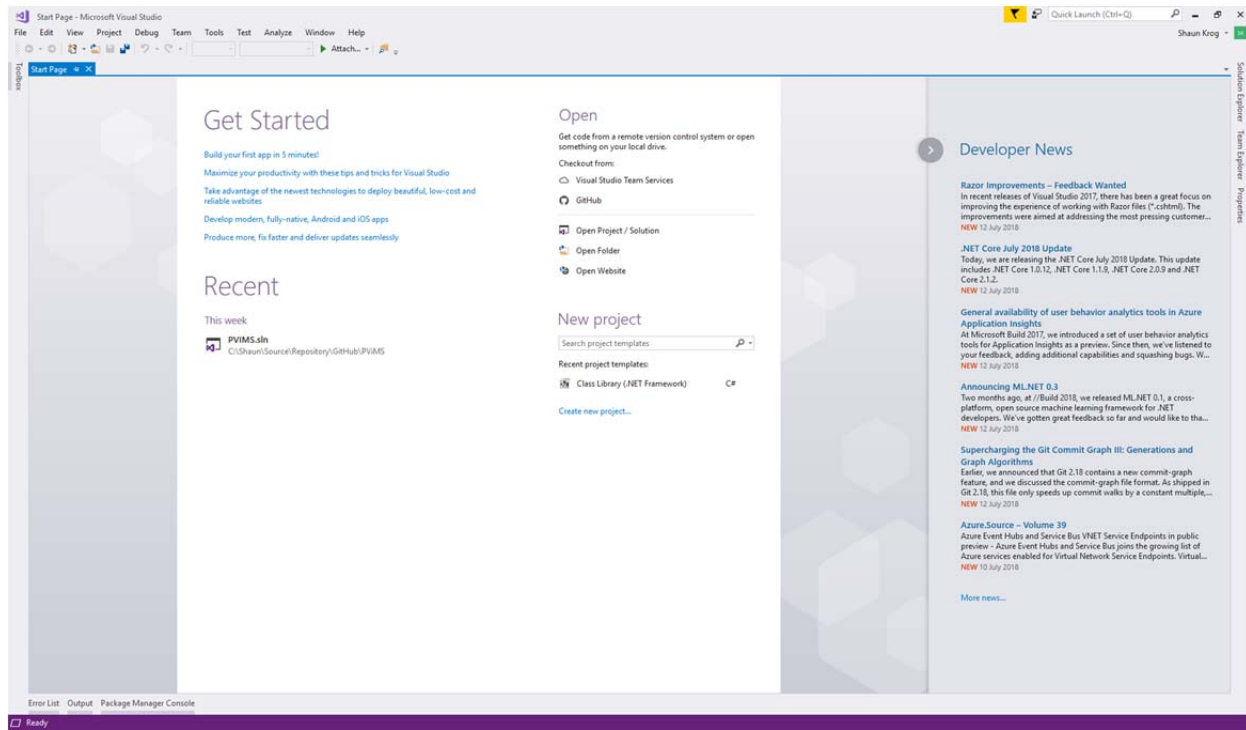
This section will give an overview of the development environment, including the method for source control and code sharing

2.1 Integrated Development Environment (IDE)

Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs for Microsoft Windows, as well as web sites, web apps, web services and mobile apps.

As of Visual Studio 2010, the Professional edition is the entry level commercial edition of Visual Studio, while the Community edition is a free, fully-featured IDE for open source developers. Visual Studio provides an IDE for all supported development languages. MSDN support is available as MSDN Essentials or the full MSDN library depending on licensing. It supports XML and XSLT editing, and can create deployment packages that only use ClickOnce and MSI. It includes tools like Server Explorer and integration with Microsoft SQL Server.

It is recommended that Visual Studio 2017 Professional edition be used for development and support of PViMS, although the Community edition is also suitable.



2.2 Source Control

Git is a version control system for tracking changes in source code files and coordinating work on those files among multiple software developers. PViMS utilizes GitHub, a Git repository hosting service, for management of PViMS source code. GitHub provides access control and several collaboration features such as a wiki and basic task management tools.

2.2.1 PViMS GitHub Repository

A GitHub repository is a storage space hosted within GitHub where you can access the PViMS project, source code and manuals.

The PViMS repository is hosted on the following URL:

<https://github.com/MSH/PViMS>

You will need to be registered on GitHub.com to be able to access the PViMS codebase.

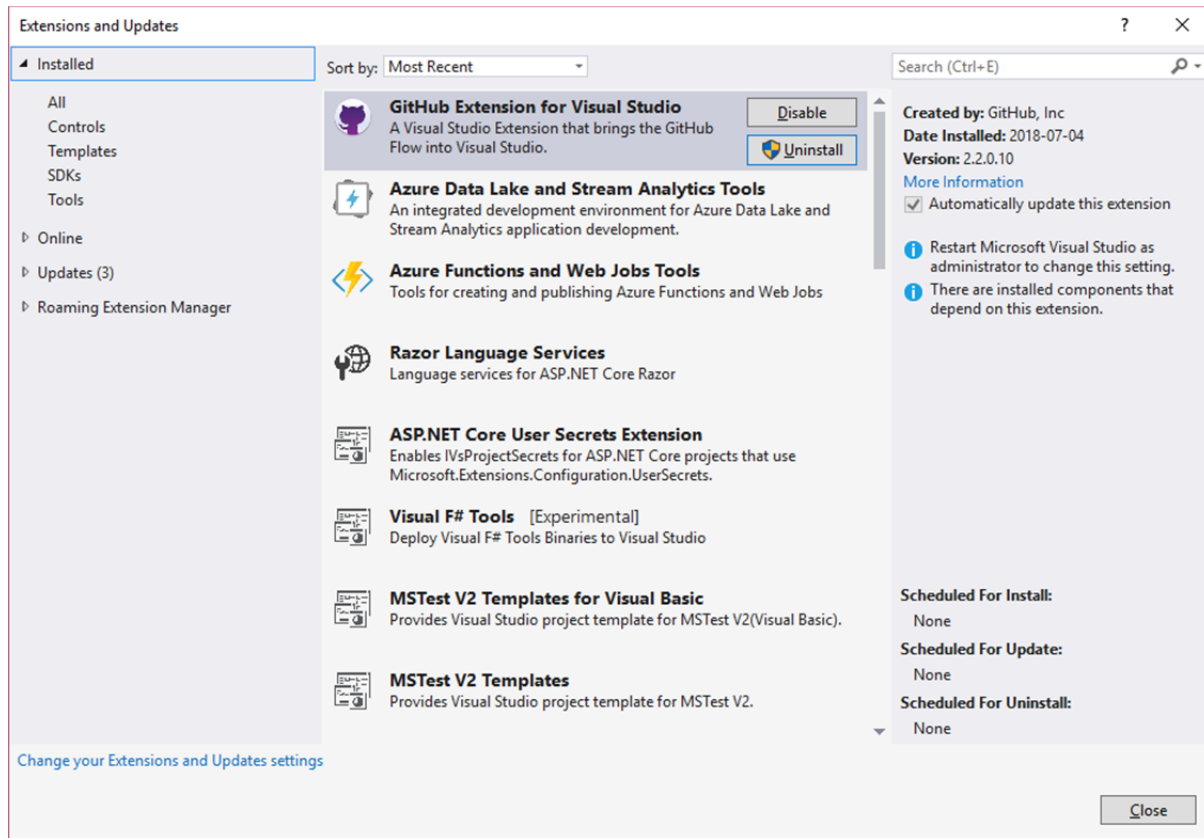
The following Git-related commands are useful to know:

Term	Description
------	-------------

Command Line	The command line is a non-native program that you can use to type text-based commands, known as prompts, to make changes within the repository
Version Control	In terms of software development, version control defines the ability to keep snap shots of source code files at various points in time, so that you never lose or overwrite code changes
Commit	A GIT commit effectively tracks changes to your local repository by creating a snap shot of your repository at the point of the commit. The commit itself stores what modifications have been made within the repository since the last commit.
Push	A GIT push command effectively publishes changes to the GitHub repository, thereby allowing your changes to be merged into the PViMS development branch.
Push	A GIT pull command effectively pulls latest changes from the GitHub repository and merges them into your local repository, thereby allowing you to consume other developer changes to PViMS into your local repository
Branch	Branching Is the mechanism that fundamentally allows multiple software developers to work on the same PViMS code at the same time. Branching allows each developer to branch out from the original code base and isolate their work from others. Different branches can ultimately be merged into any single branch, as long as it is part of the same repository.

2.2.2 GitHub Extension for Visual Studio

The easiest and most convenient way to connect to GitHub.com using Visual Studio is through the GitHub Extension, which can be added through the Visual Studio Extension Manager.



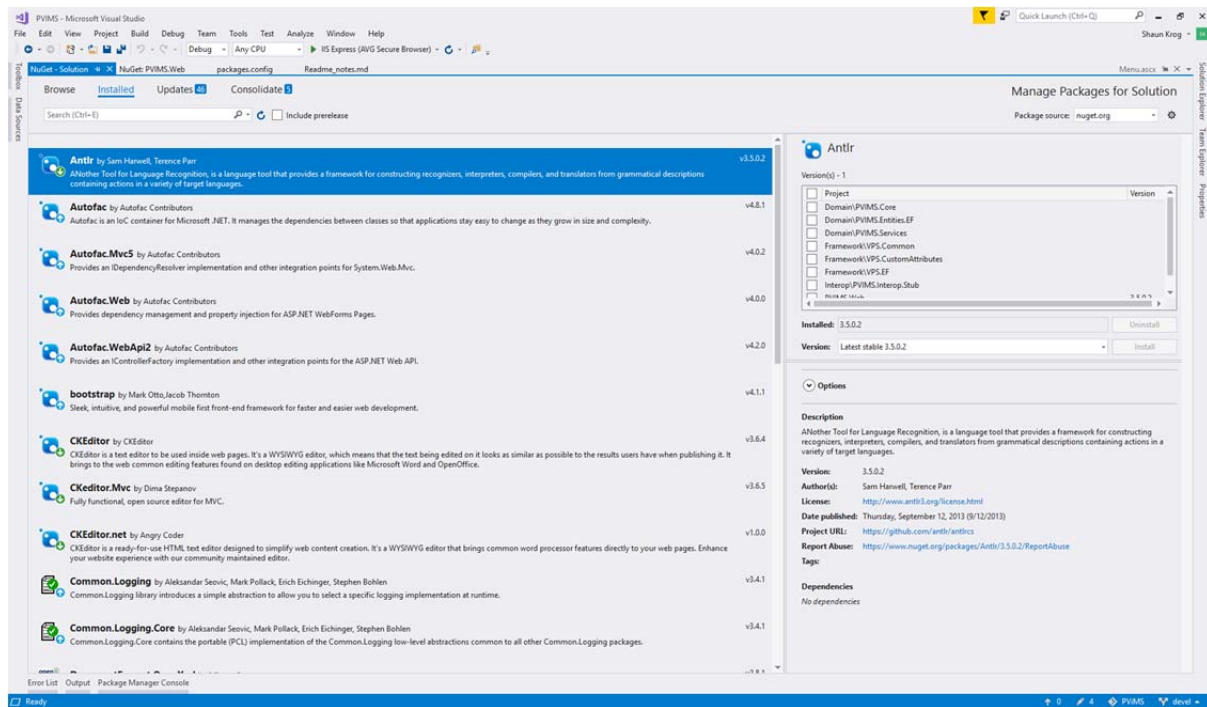
2.3 NuGet Packages

For Dotnet, the Microsoft-supported mechanism for sharing code is NuGet, which defines how packages are created, hosted and consumed and provided the tools for each of these roles.

A NuGet package, in simplified terms, is a single compressed file with a nupkg extension that contains a set of DLLs', additional context related files and a manifest that contains details for the package, including the version number.

The system integrates with a number of open source components as NuGet packages which are available on the online NuGet repository. These packages are owned, maintained and licensed by third-parties promote software reuse and increase productivity. Key NuGet packages used

include Entity Framework (for database integration) and Newtonsoft.Json (runtime serialization of objects).



A full set of NuGet packages used within PViMS can be described as follows:

Package	Details	Version
Antlr	ANother Tool for Language Recognition, is a language tool that provides a framework for constructing recognizers, interpreters, compilers, and translators from grammatical descriptions containing actions in a variety of target languages.	3.5.0.2
AutoFac	Autofac is an IoC container for Microsoft .NET. It manages the dependencies between classes so that applications stay easy to change as they grow in size and complexity.	3.5.2
Bootstrap	The most popular front-end framework for developing responsive, mobile first projects on the web	3.3.4
CKEditor	CKEditor is a text editor to be used inside web pages.	3.6.4
DotNetZip	DotNetZip is a FAST, FREE class library and toolset for manipulating zip files. Use VB, C# or any .NET language to easily create, extract, or update zip files.	1.9.3
EntityFramework	Entity Framework is Microsoft's recommended data access technology for new applications.	6.1.3
EPPlus	Create advanced Excel spreadsheets using .NET	4.0.5
JQuery	jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. jQuery is designed to change the way that you write JavaScript.	2.1.4
Knockout	A JavaScript MVVM library to help you create rich, dynamic user interfaces with clean maintainable code	3.3.0
LINQKit	LinqKit.EntityFramework is a free set of extensions for LINQ to SQL and Entity Framework power users.	1.1.2
NewtonSoft.Json	Json.NET is a popular high-performance JSON framework for .NET	7.0.1

RequireJS	A JavaScript module loader.	2.1.19
WebGrease	Web Grease is a suite of tools for optimizing javascript, css files and images.	1.6.0

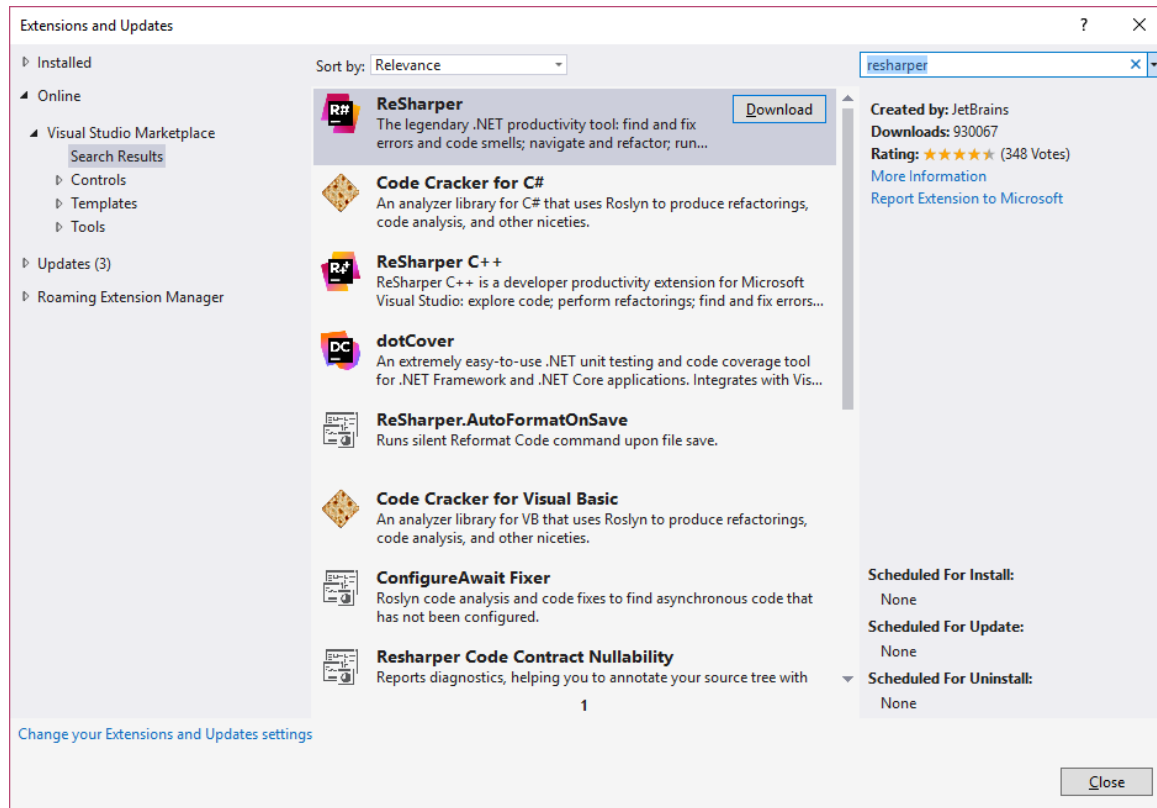
Packages can be installed through NuGet by using the package console manager within Visual Studio using the following syntax:

```
Install-Package Antlr -Version 3.5.0.2
```

2.4 Automated Code Formatting

ReSharper is a popular developer productivity extension for Microsoft Visual Studio. It automates most of what can be automated in your coding routines. It finds compiler errors, runtime errors, redundancies, and code smells right as you type, suggesting intelligent corrections for them.

- Code analysis
- Refactoring
- Navigation and Search
- Code formatting and clean up
- Code generation



3 Deployment

This section will give an overview of the process needed to prepare a PViMS deployment package. The deployment package prepared can be used for deployment as per the deployment manual.

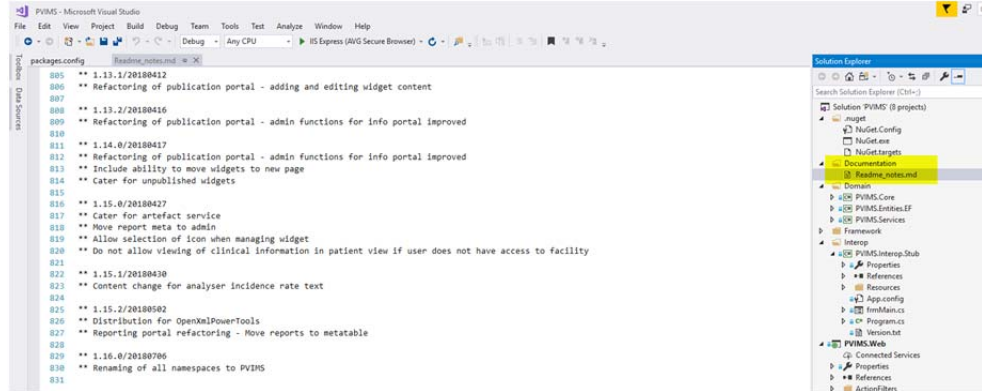
3.1 Preparing a package for deployment

The following steps should be followed when preparing the deployment package for a PViMS release.

Step	Description
Step 1.	❖ Ensure the Readme_notes.md file under the documentation folder in the

Update
Release notes
and
increment
version
number

solution is updated with details of the latest modifications.



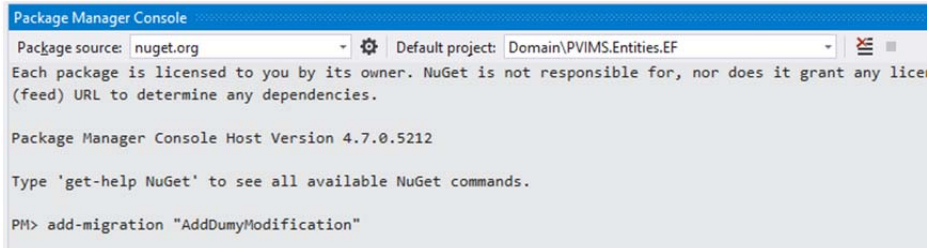
❖ Update all occurrences of the PVIMS version number

```
<!-- END MAIN PANEL -->
<!-- PAGE FOOTER -->
<div class="page-footer">
  <div class="row">
    <div class="text-center">
      <span class="text-color-white" style="color:white">Management Sciences For Health © 2016 | Logged in as <asp:Placeholder id="pnlLogin" runat="server"></asp:Placeholder> <span style="background-color:yellow">version 3.16.0</span>
    </div>
  </div>
</div>
<!-- END PAGE FOOTER -->
```

Step 2.
Create
migration
configuration
for any
model
modifications
and update
database

If you have made any modifications to the PVIMS database context, please ensure you have created a migration for these modifications.

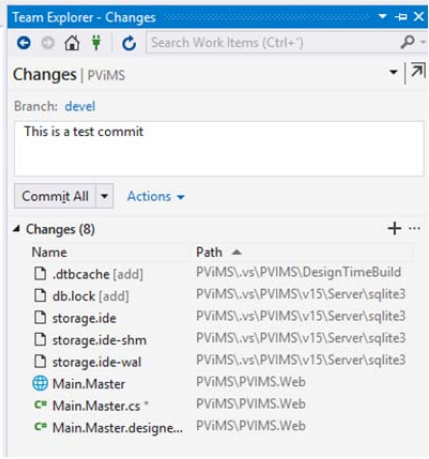
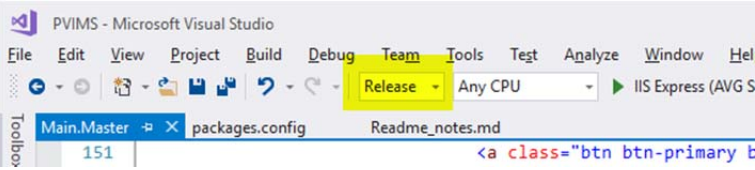
- ❖ Select the Package manager Console
- ❖ Ensure the PVIMS.Entities.EF project is select
- ❖ Migrate the configuration

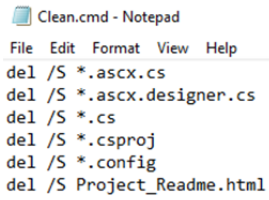


Step 3.
Commit
changes to
local
repository

Commit your changes to your localized GIT repository.

- ❖ Select Team Explorer → Changes
- ❖ Enter your commit message
- ❖ Select commit all

	
Step 4. Push changes to GitHub MSH\PViMS Repository	<p>Synchronize your changes with the GitHub repository</p> <ul style="list-style-type: none"> ❖ Select Team Explorer → Sync ❖ Under outgoing commits, select the push option <p>Please note, if any modifications have been made on the main PViMS development branch, you will be required to merge these changes into your local repository before synchronizing.</p>
Step 5. Rebuild the solution in release mode	<p>Create a release build of the PViMS application.</p>  <ul style="list-style-type: none"> ❖ Rebuild the solution
Step 6. Prepare the deployment package	<ul style="list-style-type: none"> ❖ Copy the contents of the \Repository\GitHub\PViMS\PVIMS.Web folder into your release folder. ❖ Cleanse the release folder.



```
File Edit Format View Help
del /S *.ascx.cs
del /S *.ascx.designer.cs
del /S *.cs
del /S *.csproj
del /S *.config
del /S Project_Readme.html
```

```
rmdir WWW\ActionFilters /s /q
rmdir WWW\ActionResults /s /q
rmdir WWW\App_Code /s /q
rmdir WWW\App_Data /s /q
rmdir WWW\App_Start /s /q
rmdir WWW\Obj /s /q
rmdir WWW\Controllers /s /q
rmdir WWW\Models /s /q
rmdir WWW\Properties /s /q
```

- ❖ Create a ZXIP file containing the contents of the
Repository\GitHub\PViMS\PVIMS.Web folder