

Chapter 1

Haystack Overview

Table of Contents

Chapter 1	1-1
Haystack Overview	1-1
Haystack Overview	1-2
Philosophy	1-2
Goals of Haystack	1-3
What Haystack can Create	1-4
Data Access Classes	1-4
Business Rule Classes	1-4
Entity Classes	1-4
Stored Procedures	1-5
XML Data Classes	1-5
XAML CRUD User Control	1-5
ASP.NET CRUD User Control / Page	1-5
View Model Generation	1-5
WCF Services Generation	1-5
Why use Data Access Classes	1-6
A Haystack Project	1-6
Chapter Index	1-7

Haystack Overview

Haystack Code Generator for .NET is a utility that will help you build N-Tier Data Access Classes, basic select, search, add, edit, and delete forms and XML data classes. A Data Access Class (DAC) is a wrapper around a single table, view, or stored procedure in a database. This wrapper class contains all of the CRUD (Create/Read/Update/Delete) logic for a table. Or this wrapper class will be a read-only class that can call a view. Or it can be a wrapper class that allows you to fill in parameters and then call a stored procedure and retrieve the output parameters from that stored procedure.

Haystack includes tools, templates, samples, and documentation that simplify and accelerate the delivery of WPF, Silverlight, Windows Phone and ASP.NET applications for C# and VB.NET developers. Haystack will build n-tier classes for you in a fraction of the time of other products. By utilizing Haystack tool you can standardize your company's development efforts, thereby increasing productivity and maintainability.

Haystack will provide a significant productivity boost to your development team in developing data access routines and in maintaining them. Most developers do not have the time or the expertise to use all of the latest component development techniques available today. By using Haystack you are taking a component architecture that is already designed and applying it to your application. This will significantly increase your reliability, speed of development and will reduce the costs of producing the application.

This chapter will give you an overview of the philosophy and the goals of Haystack.

Philosophy

To get the most out of the PDSA method of application development you should understand some key concepts:

Coding standards are vital to the success of a project. All programmers must follow the standards set forth in these documents. If you adhere to these standards, your code will be more consistent and all programmers will be able to read all other programmers code. **NOTE:** The PDSA standards documents are contained in the \Documentation\Standards folder in a Microsoft Word format in the installation folder of this product.

When developing WPF, Silverlight or ASP.NET application, the pages, Windows or user controls should NOT know anything about table structures, data access methods, or where the data is coming from. Instead all the logic

to retrieve, add, edit and/or delete from any table, view and/or stored procedure in your database should be encapsulated into its own Data Access Class (DAC). This makes sure that you are not duplicating code throughout your application. Once you create a class/method for accessing a specific object in your database, you ensure that class/method is used consistently throughout your application.

All Data Classes are further removed from the pages, windows or user controls through the use of a View Model class. This View Model class is directly used by the page, window or user control.

In addition to the DACs you generated, a Business Rule class and an Entity class will also be created. Each of these classes will be separated into two files and designated as “partial” classes. One file of the Business Rule class is where we generate rules that we infer from the schema. The other file is for where you can add your own business rules. The same holds true for the Entity class. One is for us to regenerate any new or deleted fields from your table. The other file is where you can freely add any additional properties that you require.

This separation allows you to regenerate the underlying DACs at any time via the Haystack while preserving your additions.

Goals of Haystack

When PDSA, Inc. set out to build a data access class code generator we had several goals in mind. Obviously ease-of-use and rapid application development were topmost in our minds. But there were some others that also came about:

- Supply templates that the code generator uses in combination with information from a repository (database) to create complete data access classes, generate stored procedures and generate wrappers around calls to XML files.
- Allow the programmer to modify these templates for each project. At the same time maintain a base set of templates that can be copied from project to project.
- Be able to quickly generate all of the data access methods for a particular object in a database.
- Create a repository whereby the programmer can add code to be generated into the template. This leads to the ability to regenerate class modules as changes need to be made.

- Be able to generate 100% of the data access code an application can use to SELECT, SEARCH, INSERT, UPDATE and DELETE data.
- Generate standard business rules inferred from the schema of a table.
- Supply several reusable classes and DLLs that can be of immediate use in your programs.
- Supply several examples of how to use the code generated from Haystack.

What Haystack can Create

Haystack helps you generate code that would otherwise be very time-consuming and boring to write. The following items can be generated by the Haystack.

Data Access Classes

Classes that are inherited from a PDSA base class to perform retrieval and modification of data in a table in a database. This retrieval and modification can be performed by submitting SQL statements, or by submitting the name of a stored procedure (complete with parameters) from within this class.

Business Rule Classes

These classes contain rules that are inferred from your schema and rules that you add explicitly. Like with Data Classes you should be able to regenerate these rule classes if your schema changes while preserving any custom code you have written.

Entity Classes

An “Entity” class is simple a list of properties that match to the fields of your table or view, or to the parameters of a stored procedure. This entity class is designed to be transferred across a service boundary through web services or WCF services.

Stored Procedures

The PDSA DACGen can generate stored procedures that read and modify data in a database table. These stored procedures can then be used in the above mentioned data access classes.

XML Data Classes

If you have an XML file that is just like a single table, you can generate a class wrapper to allow you to select, insert, update and delete xml elements from that file. These XML files can be either element and/or attribute based xml files. For samples, see the .XML files located in the [HaystackInstallFolder]\Samples folder.

XAML CRUD User Control

When generating for a single table a standard add, edit, delete user control for either Silverlight or WPF can be generated. This user control will give you a good starting point for developing your application's screens.

ASP.NET CRUD User Control / Page

When generating for a single table a standard add, edit, delete user control and page for ASP.NET can be generated. The user control and/or page will give you a good starting point for developing your application's screens.

View Model Generation

PDSA has standardized on the Model-View-View-Model (MVVM) approach for all of our application development. A view model class will be generated that all user controls (the View in MVVM) will communicate with. The view model class communicates with the data classes (the Model in MVVM).

WCF Services Generation

Haystack is designed with a Service Oriented Architecture (SOA) in mind. Our Entity Classes, Data Classes and Validation Classes all know how to work with WCF service applications. We generate Interfaces, Services and Response classes that can be used in Silverlight, WPF or ASP.NET.

Why use Data Access Classes

There are many reasons to use data access classes in your database programming.

- Eliminates SQL from front end
- Allows entity classes to be used across a service like a web or a WCF service.
- Hides database implementation
- Hides data access method from the front-end
- Less changes to front end code when column is changed
- Programmers like the IntelliSense listing of properties
- Faster development, less bugs
- Reusable components that can be used from C#, VB.NET, or any other .NET language.
- Code generation leads to more bullet proof and tested code. This leads to less bugs and faster development.

A Haystack Project

Haystack uses a “Haystack Project” as a pointer to a single database/catalog within a server. Each project has certain attributes about it. For example, a project will be able to generate classes for tables, views and stored procedures. Table classes can use either Dynamic SQL and/or Stored Procedures. Any project can point to a set of templates to be used to generate code. These are just a few of the attributes that you will setup to help you generate your data access classes.

Summary

Haystack helps your company develop your .NET applications much faster than coding by hand. In addition to data classes you can also generate Silverlight user controls, WPF windows and user controls, and ASP.NET pages.

Chapter Index

A

ASP.NET CRUD User Controls, 1-5
ASP.NET User Controls, 1-5

B

Business Rule Classes, 1-4

C

Code Generation, 1-4

D

Data Access Classes, 1-4, 1-6

E

Entity Classes, 1-4

G

Goals of Haystack, 1-3

H

Haystack
 Overview, 1-2
Haystack Project, 1-6

M

MVVM Generation, 1-5

O

Overview, 1-2

P

Philosophy, 1-2

S

Stored Procedure, 1-5

V

View Model Generation, 1-5

W

WCF Services Generation, 1-5
What Haystack can Create, 1-4
Why use Data Access Classes, 1-6

X

XAML CRUD User Controls, 1-5
XAML User Controls, 1-5
XML Data Classes, 1-5