# ProgressTen Coding Standards and Guidelines

## Document Intent

The purpose of this document is to outline the coding standards that will be used in the writing of the ProgressTen web site as well as to layout guidelines for the use of certain technologies. This is a living document and subject to change upon full agreement of all project owners.

## Coding Standards

### Language

#### Backend

C# will be the language of choice for the backend. All code shall be written in C# unless otherwise noted or directed.

#### UI

The UI will be written in HTML and Spark. The default view engine will be replaced with Spark. Layout will use the BluePrint CSS model. HTML tables will only be used to contain tabular data in cases where a dedicated grid control is not being used.

#### AJAX

JavaScript (jQuery 1.4.2 specifically) will be used to perform any AJAX interactions from the UI to the Backend.

### Naming Conventions

#### General

Please be sure to make your variable names descriptive to what that variable will hold. It would be preferable to have a longer name that is descriptive, than a short name used to save on typing.

**Examples**  
private int myCount; versus private int i;

Public string UserName; versus public string a;

#### Private variables

Private variables will be named using camel case, with all private variables starting with an underscore.

**Example**  
private string \_variable;

#### Public variables

Public variables will be named using Pascal casing.

**Example**  
public string Variable;

#### Method names

Method names will always be in Pascal case, regardless if they are public, protected, internal, or private. All parameters of the method will be in camel case.

**Example**  
public void MyMethod(int parameterOne) { }

#### Class names

All classes will be in Pascal case.

**Example**  
public class MyClass {}

#### Interface names

All interfaces will start with a capital “I” and be named in Pascal casing.

**Example**  
public interface IMyInterface{}

#### Enum names

All enums will be in Pascal case and will NOT contain “enum” in the name. The values of the enum will be named in Pascal case.

**Example**  
public enum MyStatus { Here, There, Everywhere }

#### View names

All views will be named in Pascal case.

**Example**  
MyView.spark

#### Partial view names

Partial views will be named in Pascal case and start with an underscore.

**Example**  
\_MyPartialView.spark

## Guidelines

### Technology used

The following technologies will be used to create the ProgressTen web site:

* ASP.NET MVC 2
* S#arp Architecture 1.5.x (latest version)(S# for short)
  + NHibernate
  + Fluent NHibernate
  + NHibernate Validators
  + Castle Components
    - Core
    - Dynamic Proxy 2
    - MicroKernel
    - Windsor
  + MVCContrib
* Spark View Engine
* Telerik MVC Controls
* JQuery
* MS SQL

### Domain Models, View Models, and Form Models

In this project, we will use **Domain models** that inherit from S# Entity class. These will be our core object and will be persisted to the database. **View models** will be used in the UI layer (.Web and .Web.Controllers projects) to flatten one or more **Domain models** for display in a view. **Form models** will be used to deliver data from the view to the controller, unless AJAX is being used. In the case of AJAX use, individual variables will be passed back to the controller.

When mapping **Domain models** to **View models**, Automapper will be used. This will cut down on the use of magic strings and thus eliminate some human error. Automapper will also be used to map **Form models** into **Domain models**. The mapping is done in the presentation layer, never below that point. The Application Services layer should always return **Domain** **models**, not **View models**. The reason for this is that the presentation layer should always implement its own **View** and **Form models**. While this may lead to duplication, this should be minimal.

### Model validation

MVC 2 has introduced many great new features; among them is the use of Client Validation out of the box. In the context of the ProgressTen there are two types of validation, UI and Domain. The **Domain models** and **Form models** will be decorated with NHibernate Validation attributes so that each may be validated. The reason for the separate validation is because the backend should never make the assumption that the frontend has validated the objects. This also allows for many different types of frontends to be used and to implement their own validation.