Ameex

Ameex Coding Standards

May 15th, 2013

Table of Contents

1. Introduction

This document is prepared for the Ameex Web Developers, who develop Ektron and .Net web applications using ASP .Net. These conventions help the developers to code effectively and to use server resources efficiently and in a consistent style.

2. Usage of coding standards and best practices

To develop reliable and maintainable applications, you must follow coding standards and best practices.

The naming conventions, coding standards and best practices described in this document are compiled from our own experience and by referring to various Microsoft and non-Microsoft guidelines.

3. Naming Conventions and Standards

Note :

The terms Pascal Casing and Camel Casing are used throughout this document.

- **Pascal Casing** - First character of all words are Upper Case and other characters are lower case.
  Example: BackColor

- **Camel Casing -** First character of all words, except the first word are Upper Case and other characters are lower case.
  Example: backColor

3.1. Pascal casing

3.1.1. Class names

Use Pascal casing for Class names

```
Public class HelloWorld
{
        ...
}
```

3.1.2. Method names

Use Pascal casing for Method names

```
void SayHello(string name)
{
        ...

}
```

3.2. Camel casing

Use Camel casing for variables and method parameters

```
int totalCount = 0;
void SayHello(string name)
{
        string fullMessage = "Hello " + name;
        ...
}
```

3.3. Uppercase

All letters in the identifier are capitalized. Use this convention only for identifiers that consist of two or less letters; or abbreviations, such as BSU and UCS. In the example below, IO and UI are the uppercase identifiers, whereas System follows the Pascal capitalization style because the length is greater than two.

For example: System.IO; System.Web.UI

The following table summarizes the capitalization rules and provides examples for the different types of identifiers and items within your Web application/project.

Identifier	Case	Example
Class	Pascal	AppDomain
Enumeration type	Pascal	ErrorLevel
Enumeration value	Pascal	FatalError
Event	Pascal	ValueChange
Exception class	Pascal	WebException (always end with Exception)
Interface	Pascal	IDisposable (always begin with the prefix I)
Method	Pascal	ToString
Namespace	Pascal	BSU.UCS
Parameter	Camel	typeName (see list for parameter prefixes)
Property	Pascal	BackColor
Protected instance field	Camel	redValue (rarely used)
Public instance field	Pascal	RedValue
Read-only static field	Pascal	RedValue
Variable	Camel	typeName (see list for parameter prefixes)

Project item	Case	Example
Class library	Pascal	DatabaseConnection.cs
Folder	Pascal	MyLibrary
Page (Web form)	Pascal	DisplayInfo.aspx
Style sheet	Pascal	StyleSheet.css

3.4. Namespaces

Namespaces are a logical (not physical) way to group classes. A single namespace may contain one or more physical class files (ClassName.cs). When the ASP.NET framework compiles your application, all class files with the same namespace declaration are compiled into a single assembly (.dll file).

When creating namespaces, use the following guideline:

- BSU.UCS (Company name followed by the department name)
- You should use Pascal case for namespaces, and separate logical components with periods, as in BSU.UCS.Common.
- Use plural namespace names if it is semantically appropriate. For example, use System.Collections rather than System.Collection.

3.5. Variables

3.5.1. Case sensitivity

To avoid confusion and coding errors, do not use two names within the same context that differ only by case. C# is a case-sensitive language, and will see these two identifiers as separate instances.

The following is an example of an incorrectly named identifier:

- LastName
- lastname

As a developer, if you attempt to use the aforementioned identifiers to represent the Last Name, the C# complier will not understand that LastName = lastname, and will consider them as two different names and potentially cause compilation errors.

3.5.2. Hungarian Notation

Do not use Hungarian notation to name variables

In earlier days most of the programmers liked it - having the data type as a prefix for the variable name and using m_ as prefix for member variables.

Example:

- string m_sName;
- int nAge;

However, in .NET coding standards, this is not recommended. Usage of data type and m_ to represent member variables should not be used. All variables should use camel casing.

> Some programmers still prefer to use the prefix **m_** to represent member variables, since there is no other easy way to identify a member variable.

3.5.3. Abbreviations

Use Meaningful, descriptive words to name variables. Do not use abbreviations.

Good:

string address
int salary

Not Good:

string nam
string addr
int sal

3.5.4. Variable length

Do not use single character variable names like i, n, s etc. Use names like index, temp

3.5.5. Underscores (_)

- Do not use underscores (_) for local variable names.
- All member variables must be prefixed with underscore (_) so that they can be identified from other local variables.

3.5.6. Keywords

Do not use variable names that resemble keywords.

3.5.7. Prefix Boolean Variables

Prefix Boolean variables, properties and methods with “is” or similar prefixes.

Example: private isFinished

3.5.8. UI element Name

Use appropriate prefix for the UI elements so that you can identify them from the rest of the variables.

Follow the approaches recommended here.

Use appropriate prefix for each of the UI element. A brief list is given below. Since .NET has given several controls, you may have to arrive at a complete list of standard prefixes for each of the controls (including third party controls) you are using.

Control	Prefix
Label	lbl
TextBox	txt
DataGrid	dtg
Button	btn
ImageButton	imb
Hyperlink	hlk
DropDownList	ddl
ListBox	Lst
DataList	dtl
Repeater	rep

Control	Prefix
Checkbox	chk
CheckBoxList	cbl
RadioButton	rdo
RadioButtonList	rbl
Image	img
Panel	pnl
PlaceHolder	phd
Table	tbl
Validators	val

3.6. File Name

3.6.1. File Name should match with class name

For example, for the class HelloWorld, the file name should be HelloWorld.cs or HelloWorld.vb

3.6.2. Use Pascal Case for file names

For example, the file name should be HelloWorld.cs or HelloWorld.vb

4. Indentation and Spacing

4.1. Indent

Use TAB for indentation. Do not use SPACES. Define the Tab size as 4.

4.2. Blank Line

- Use one blank line to separate logical groups of code.

Good:

```
bool SayHello (string name)
{
        string fullMessage = "Hello " + name;
        DateTime currentTime = DateTime.Now;

        string message = fullMessage + ", the time is : " +
        currentTime.ToShortTimeString();

        MessageBox.Show ( message );

        if ( ... )
        {
                // Do something
                // ...

                return false;
        }

        return true;
}
```

Not Good:

```
bool SayHello (string name)
{
        string fullMessage = "Hello " + name;
        DateTime currentTime = DateTime.Now;
        string message = fullMessage + ", the time is : " +
        currentTime.ToShortTimeString();
        MessageBox.Show ( message );


        if ( ... )
        {
            // Do something
            // ...
            return false;
        }
        return true;
}
```

- There should be one and only one single blank line between each method inside the class.

4.3. Curly Braces

The curly braces should be on a separate line and not in the same line as if, for etc.

Good:

```
if ( ... )
{
    // Do something
}
```

Not Good:

```
if ( ... ) {
    // Do something
}
```

4.4. Single Space

Use a single space before and after each operator.

Good:

```
if ( showResult == true )
{
    for ( int i = 0; i < 10; i++ )
    {
        //
    }
}
```

Not Good:

```
if(showResult==true)
{
    for(int     i= 0;i<10;i++)
    {
        //
    }
}
```

4.5. Grouping the code

Use #region to group related pieces of code together. If you use proper grouping using #region, the page should look like this when all definitions are collapsed.

```
public class ContentUtilities
{

    Constructor

    GetContentDetails

    GetMetadataDetails

}
```

5. Good Programming practices

5.1. Method

5.1.1. Method Length

Avoid writing very long methods. A method should typically have 1~50 lines of code. If a method has more than 50 lines of code, you must consider re factoring into separate methods.

5.1.2. Method Name

Method name should tell what it does. Do not use misleading names. If the method name is obvious, there is no need of documentation explaining what the method does.

Good:

```
void SavePhoneNumber ( string phoneNumber )
{
        // Save the phone number.
}
```

Not Good:

```
// this method will save the phone number.
void SaveDetails ( string phoneNumber )
{
        // Save the phone number.
}
```

5.1.3. Method Functionality

A method should do only 'one job'. Do not combine more than one job in a single method, even if those jobs are very small.

Good:

```
// Save the address.
SaveAddress (  address );


// Send an email to the supervisor to inform that the address is updated.
SendEmail ( address, email );

void SaveAddress ( string address )
{
        // Save the address.
        // ...
}

void SendEmail ( string address, string email )
{
         // Send an email to inform the supervisor that the address is changed.
        // ...
}
```

Not Good:

```
// Save address and send an email to the supervisor to inform that
// the address is updated.
SaveAddress ( address, email );

void SaveAddress ( string address, string email )
{
        // Job 1.
        // Save the address.
        // ...

        // Job 2.
         // Send an email to inform the supervisor that the address is changed.
        // ...
}
```

5.2. Good Practices

5.2.1. Use the C# or VB.NET specific types (aliases), rather than the types defined in System namespace.

int age; (not **I**nt**16**)
string name; (not **S**tring)
object contactInfo; (not **O**bject)

Some developers prefer to use types in Common Type System than language specific aliases.

5.2.2. Always watch for unexpected values. For example, if you are using a parameter with 2 possible values, never assume that if one is not matching then the only possibility is the other value.

Good:

```
If ( memberType == eMemberTypes.Registered )
{
        // Registered user… do something…
}
else if ( memberType == eMemberTypes.Guest )
{
        // Guest user... do something…
}
else
{
                // Unexpected user type. Throw an exception
                    throw new Exception ("Unexpected value " +
memberType.ToString() + "'.")

                // If we introduce a new user type in future, we can easily find
    // the problem here.
}
```

Not Good:

```
If ( memberType == eMemberTypes.Registered)
{
                // Registered user… do something…
}
else
{
                // Guest user... do something…

// If we introduce another user type in future, this code will
// fail and will not be noticed.
}
```

5.2.3. Do not hardcode the numbers or strings. Use constants only if you are sure that this value will never change. Otherwise you should use the constants in the configuration (.config) file.

5.2.4. Convert strings to lowercase or upper case before comparing. This will ensure the string will match even if the string being compared has a different case.

```
if ( name.ToLower() == “john” )
{
        //…
}
```

Use String.Empty instead of “”

Good:

```
If ( name == String.Empty )
{
        // do something
}
```

Not Good:

```
If ( name == “” )
{
        // do something
}
```

5.2.5. Avoid using member variables. Declare local variables wherever necessary and pass it to other methods instead of sharing a member variable between methods. If you share a member variable between methods, it will be difficult to track which method changed the value and when.

5.2.6. Use enum wherever required. Do not use numbers or strings to indicate discrete values.

Good:

```
enum MailType
{
        Html,
        PlainText,
        Attachment
}

void SendMail (string message, MailType mailType)
{
        switch ( mailType )
        {
                case MailType.Html:
```

```
                    // Do something
                    break;
                case MailType.PlainText:


            // Do something
                    break;
                case MailType.Attachment:
                    // Do something
                    break;
                default:
                    // Do something
                    break;
            }
        }
```

Not Good:

```
        void SendMail (string message, string mailType)
        {
            switch ( mailType )
            {
                case "Html":
                    // Do something
                    break;
                case "PlainText":
                    // Do something
                    break;
                case "Attachment":
                    // Do something
                    break;
                default:
                    // Do something
                    break;
            }
        }
```

5.2.7. The event handler should not contain the code to perform the required action. Rather call another method from the event handler.

5.2.8. Never hardcode a path or drive name in code. Get the application path programmatically and use relative path.

5.2.9. Never assume that your code will run from drive "C:". You may never know, some users may run it from network or from a "Z:".

5.2.10. Do not have more than one class in a single file. If necessary we can use it.

5.2.11. Avoid having very large files. If a single file has more than 1000 lines of code, it is a good candidate for refactoring. Split them logically into two or more classes.

5.2.12. Avoid public methods and properties, unless they really need to be accessed from outside the class. Use “internal” if they are accessed only within the same assembly.

5.2.13. Avoid passing too many parameters to a method. If you have more than 4~5 parameters, it is a good candidate to define a class or structure.

5.2.14. If you have a method returning a collection, return an empty collection instead of null, if you have no data to return. For example, if you have a method returning an ArrayList, always return a valid ArrayList. If you have no items to return, then return a valid ArrayList with 0 items. This will make it easy for the calling application to just check for the “count” rather than doing an additional check for “null”.

5.2.15. Make sure you have a good logging class which can be configured to log errors, warning or traces. If you configure to log errors, it should only log errors. But if you configure to log traces, it should record all (errors, warnings and trace). Your log class should be written such a way that in future you can change it easily to log to Windows Event Log, SQL Server, or Email to administrator or to a File, etc. without any change in any other part of the application. Use the log class extensively throughout the code to record errors, warning and even trace messages that can help you trouble shoot a problem.

5.2.16. If you are opening database connections, sockets, file stream etc, always close them in the finally block. This will ensure that even if an exception occurs after opening the connection, it will be safely closed in the finally block.

5.2.17. Declare variables as close as possible to where it is first used. Use one variable declaration per line.

5.2.18. Use StringBuilder class instead of String when you have to manipulate string objects in a loop. The String object works in weird way in .NET. Each time you append a string, it is actually discarding the old string object and recreating a new object, which is a relatively expensive operations.

Consider the following example:

```
public string ComposeMessage (string[] lines)
{
  string message = String.Empty;
```

```
    for ( int index = 0; index < lines.Length; index++ )
    {
      message += lines [index];
    }
    return message;
  }
```

In the above example, it may look like we are just appending to the string object 'message'. But what is happening in reality is, the string object is discarded in each iteration and recreated and appending the line to it.

If your loop has several iterations, then it is a good idea to use StringBuilder class instead of String object.

See the example where the String object is replaced with StringBuilder.

```
public string ComposeMessage (string[] lines)
{
   StringBuilder message = new StringBuilder();

   for (int index = 0; index < lines.Length; index ++)
   {
     message.Append( lines[index] );
   }

   return message.ToString();
}
```

6. Coding Standards

6.1. JS Standards

6.1.1. All JS functions should be in an external file. There is a separate wsscript.js file for this. Do not make modifications to any of the existing js files.

6.1.2. Name the functions based on their purpose. The functions will be named with underscore and in Camel case, which means first letter of first word in lowercase, and first letter of subsequent words in uppercase.

Example

```
function _getNextObject(input_dt) {}
```

6.1.3. All information handled by scripts must have an accessible alternative when turned off

6.2. CSS Standards

6.2.1. Any style overrides should be done in wscss.css file. . Do not make modifications to any of the existing css files.

6.2.2. We do not use graphical text (e.g. graphical buttons containing text on the site)

6.2.3. The user must be able to increase the font size of any page for up to three times without “breaking” either the layout or readability

6.2.4. We aim to validate the SRA website at XHTML 1.0 transitional. The only validation error we accept is the use of a negative tabindex attribute in heading tags

6.2.5. We aim to adhere to the WCAG 1.0 and attain all three levels of compliance. If some aspect of the CMS means that certain WCAG checkpoints cannot be satisfied, this can be discussed as part of the design activities

6.2.6. “Skip navigation” functionality must be replicated, along with other accessibility measures such as ability to navigate site with keyboard only.

6.2.7. The SRA website must accessible and rendered (more or less) identically in Firefox (2 and 3), Internet Explorer (IE6-7), Safari (9) and Opera (9.1-5)

6.3. XSLT Standards

6.3.1. For any XSLT, The collection/content nodes presence should be tested before html is created.

6.3.2.In XSLT, Text should be wrapped with <xsl:text> elements.

7. Constructors

Do minimal work in the constructor. Constructors should not do much work other than to capture the constructor parameters and set main properties. The cost of any other processing should be delayed until required.

Do throw exceptions from instance constructors if appropriate.

Do explicitly declare the public default constructor in classes, if such a constructor is required. Even though some compilers automatically add a default constructor to your class, adding it explicitly makes code maintenance easier. It also ensures the default constructor

remains defined even if the compiler stops emitting it because you add a constructor that takes parameters.

Do not call virtual members on an object inside its constructors. Calling a virtual member causes the most-derived override to be called regardless of whether the constructor for the type that defines the most-derived override has been called.

8. Properties

Do create read-only properties if the caller should not be able to change the value of the property.

Do not provide set-only properties. If the property getter cannot be provided, use a method to implement the functionality instead. The method name should begin with Set followed by what would have been the property name.

Do provide sensible default values for all properties, ensuring that the defaults do not result in a security hole or an extremely inefficient design.

You should not throw exceptions from property getters. Property getters should be simple operations without any preconditions. If a getter might throw an exception, consider redesigning the property to be a method. This recommendation does not apply to indexers. Indexers can throw exceptions because of invalid arguments. It is valid and acceptable to throw exceptions from a property setter.

9. Global Variables

Do minimize global variables. To use global variables properly, always pass them to functions through parameter values. Never reference them inside of functions or classes directly because doing so creates a side effect that alters the state of the global without the caller knowing. The same goes for static variables. If you need to modify a global variable, you should do so either as an output parameter or return a copy of the global.

10. Comments

Good and meaningful comments make code more maintainable. However,

10.1. Do not write comments for every line of code and every variable declared.

10.2. Use **//** or **///** for comments. Avoid using **/* … */**

10.3. Write comments wherever required. But good readable code will require very less comments. If all variables and method names are meaningful, that would make the code very readable and will not need many comments.

10.4. Do not write comments if the code is easily understandable without comment. The drawback of having lot of comments is, if you change the code and forget to change the comment, it will lead to more confusion.

10.5. Fewer lines of comments will make the code more elegant. But if the code is not clean/readable and there are less comments, that is worse.

10.6. If you have to use some complex or weird logic for any reason, document it very well with sufficient comments.

10.7. If you initialize a numeric variable to a special number other than 0, -1 etc, document the reason for choosing that value.

10.8. The bottom line is, write clean, readable code such a way that it doesn't need any comments to understand.

10.9. Perform spelling check on comments and also make sure proper grammar and punctuation is used.

11. Exception Handling

11.1. Never do a 'catch exception and do nothing'. If you hide an exception, you will never know if the exception happened or not. Lot of developers uses this handy method to ignore non-significant errors. You should always try to avoid exceptions by checking all the error conditions programmatically. In any case, catching an exception and doing nothing is not allowed. In the worst case, you should log the exception and proceed.

11.2. In case of exceptions, give a friendly message to the user, but log the actual error with all possible details about the error, including the time it occurred, method and class name etc.

11.3. Always catch only the specific exception, not generic exception.

Good:

```
void ReadFromFile ( string fileName )
{
        try
        {
                // read from file.
        }
```

```
                catch (FileIOException ex)
                {
                        // log error.
                        //  re-throw exception depending on your case.
                        throw;
                }
        }
```

Not Good:

```
void ReadFromFile ( string fileName )
{
  try
  {
    // read from file.
  }
  catch (Exception ex)
  {
    // Catching general exception is bad... we will never know whether
    // it was a file error or some other error.
    // Here you are hiding an exception.
    // In this case no one will ever know that an exception happened.

    return "";
  }
}
```

11.4. No need to catch the general exception in all your methods. Leave it open and let the application crash. This will help you find most of the errors during development cycle. You can have an application level (thread level) error handler where you can handle all general exceptions. In case of an 'unexpected general error', this error handler should catch the exception and should log the error in addition to giving a friendly message to the user before closing the application, or allowing the user to 'ignore and proceed'.

11.5. When you re throw an exception, use the throw statement without specifying the original exception. This way, the original call stack is preserved.

Good:

```
catch
{
        // do whatever you want to handle the exception

        throw;
```

```
}
```

Not Good:

```
catch (Exception ex)
{
        // do whatever you want to handle the exception

        throw ex;
}
```

11.6. Do not write try-catch in all your methods. Use it only if there is a possibility that a specific exception may occur and it cannot be prevented by any other means. For example, if you want to insert a record if it does not already exists in database, you should try to select record using the key. Some developers try to insert a record without checking if it already exists. If an exception occurs, they will assume that the record already exists. This is strictly not allowed. You should always explicitly check for errors rather than waiting for exceptions to occur. On the other hand, you should always use exception handlers while you communicate with external systems like network, hardware devices etc. Such systems are subject to failure anytime and error checking is not usually reliable. In those cases, you should use exception handlers and try to recover from error.

12. Ektron

12.1. Object Elements

Use appropriate object name for the Ektron class so that you can identify them from the rest of the other objects.

Follow the approaches recommended here.

Use appropriate object for each of the Ektron class. A brief list is given below. Since Ektron has given several classes, you may have to arrive at a complete list of standard object names for each of the Ektron class you are using.

Class	Object	Namespace
Framework API		
Content		
AssetManager	assetManager	Ektron.Cms.Framework.Content
ContentAssetData	assetData	Ektron.Cms.Content
AssetCriteria	assetCriteria	Ektron.Cms.Content
ContentManager	contentManager	Ektron.Cms.Framework.Content
ContentData	contentData	Ektron.Cms
List<ContentData>	lstContentData	Ektron.Cms
ContentCriteria	contentCriteria	Ektron.Cms.Content
ContentMetadataCriteria	contentMetadataCriteria	Ektron.Cms.Content
ContentCollectionCriteria	contentCollectionCriteria	Ektron.Cms.Content
ContentTaxonomyCriteria	contentTaxonomyCriteria	Ektron.Cms.Content
ContentRatingManager	contentRatingManager	Ektron.Cms.Framework.Content
ContentRatingData	contentRatingData	Ektron.Cms.Framework.Content
ContentRatingCriteria	contentRatingCriteria	Ektron.Cms.Content
LibraryManager	libraryManager	Ektron.Cms.Framework.Content
LibraryData	libraryData	Ektron.Cms.Framework.Content
LibraryCriteria	libraryCriteria	Ektron.Cms.Content
MetadataTypeManager	metadataTypeManager	Ektron.Cms.Framework.Content
ContentMetaData	contentMetaData	Ektron.Cms
MetadataTypeCriteria	metadataTypeCriteria	Ektron.Cms.Content
TemplateManager	templateManager	Ektron.Cms.Framework.Content
TemplateData	templateData	Ektron.Cms
TemplateCriteria	templateCriteria	Ektron.Cms.Common

Class	Object	Namespace
Organization		
CollectionManager	collectionManager	Ektron.Cms.Framework.Organization
ContentCollectionData	contentCollectionData	Ektron.Cms.Organization
CollectionCriteria	collectionCriteria	Ektron.Cms
FolderManager	folderManager	Ektron.Cms.Framework.Organization
FolderData	folderData	Ektron.Cms
FolderCriteria	folderCriteria	Ektron.Cms
MenuManager	menuManager	Ektron.Cms.Framework.Organization

Class	Object	Namespace
Organization		
MenuData	menuData	Ektron.Cms.Organization
MenuItemData	menuItemData	Ektron.Cms.Organization
SubMenuData	subMenuData	Ektron.Cms.Organization
MenuCriteria	menuCriteria	Ektron.Cms.Organization
TaxonomyItemManager	taxonomyItemManager	Ektron.Cms.Framework.Organization
TaxonomyItemData	taxonomyItemData	Ektron.Cms
TaxonomyItemCriteria	taxonomyItemCriteria	Ektron.Cms.Organization

Class	Object	Namespace
Organization		
TaxonomyManager	taxonomyManager	Ektron.Cms.Framework.Organization
TaxonomyData	taxonomyData	Ektron.Cms
TaxonomyCriteria	taxonomyCriteria	Ektron.Cms.Organization
TaxonomyCustomPropertyCriteria	taxonomyCustomPropertyCriteria	Ektron.Cms.Organization

Class	Object	Namespace
Activity		
ActivityCommentManager	activityCommentManager	Ektron.Cms.Framework.Activity
ActivityCommentData	activityCommentData	Ektron.Cms.Activity
ActivityCommentCriteria	activityCommentCriteria	Ektron.Cms.Activity
ActivityManager	activityManager	Ektron.Cms.Framework.Activity
ActivityData	activityData	Ektron.Cms.Activity
ActivityCriteria	activityCriteria	Ektron.Cms.Activity
ActivityStreamManager	activityStreamManager	Ektron.Cms.Framework.Activity
ActivityTypeManager	activityTypeManager	Ektron.Cms.Framework.Activity
ActivityTypeData	activityTypeData	Ektron.Cms.Activity
ActivityTypeCriteria	activityTypeCriteria	Ektron.Cms.Activity

Class	Object	Namespace
Calendar		
WebCalendarManager	calendarManager	Ektron.Cms.Framework.Calendar
WebCalendarData	calendarData	Ektron.Cms.Common.Calendar
WebCalendarCriteria	calendarCriteria	Ektron.Cms.Common.Calendar
WebEventManager	eventManager	Ektron.Cms.Framework.Calendar
WebEventData	eventData	Ektron.Cms.Common.Calendar
WebEventCriteria	eventCriteria	Ektron.Cms.Common

Class	Object	Namespace
Community		
CommunityGroupManager	communityGroupManager	Ektron.Cms.Framework.Community
CommunityGroupData	communityGroupData	Ektron.Cms
CommunityGroupCriteria	communityGroupCriteria	Ektron.Cms.Community
FavoriteTaxonomyManager	favoriteTaxonomyManager	Ektron.Cms.Framework.Community

Class	Object	Namespace
Community		
FavoriteTaxonomyData	favoriteTaxonomyData	Ektron.Cms.Community
FavoriteManager	favoriteManager	Ektron.Cms.Framework.Community
FavoriteItemData	favoriteItemData	Ektron.Cms.Community
FlagManager	flagManager	Ektron.Cms.Framework.Community
ObjectFlagData	flagData	Ektron.Cms
FlagCriteria	flagCriteria	Ektron.Cms.Community
FriendsTaxonomyManager	friendsTaxonomyManager	Ektron.Cms.Framework.Community
FriendTaxonomyData	friendTaxonomyData	Ektron.Cms.Community

Class	Object	Namespace
Community		
FriendsManager	friendsManager	Ektron.Cms.Framework.Community
FriendsData	friendsData	Ektron.Cms.Community
FriendsCriteria	friendsCriteria	Ektron.Cms.Community
RatingManager	ratingManager	Ektron.Cms.Framework.Community
RatingData	ratingData	Ektron.Cms
RatingCriteria	ratingCriteria	Ektron.Cms.Community
MessageBoardManager	messageBoardManager	Ektron.Cms.Framework.Community
MessageBoardData	messageBoardData	Ektron.Cms
MessageBoardCriteria	messageBoardCriteria	Ektron.Cms.Community
MicromessageManager	micromessageManager	Ektron.Cms.Framework.Community
MicroMessageData	microMessageData	Ektron.Cms
MicroMessageCriteria	microMessageCriteria	Ektron.Cms
PrivateMessageManager	privateMessageManager	Ektron.Cms.Framework.Community
PrivateMessageData	privateMessageData	Ektron.Cms
PrivateMessageCriteria	privateMessageCriteria	Ektron.Cms.Community
TagManager	tagManager	Ektron.Cms.Framework.Community
TagData	tagData	Ektron.Cms
TagCriteria	tagCriteria	Ektron.Cms.Community

Class	Object	Namespace
Flag		
FlagDefinitionManager	flagDefinitionManager	Ektron.Cms.Framework.Flag
FlagDefData	flagDefData	Ektron.Cms
FlagDefinitionCriteria	flagDefinitionCriteria	Ektron.Cms.Flag

Class	Object	Namespace
Notifications		
NotificationManager	notificationManager	Ektron.Cms.Framework.Notifications
ActivityData	activityData	Ektron.Cms.Activity

Class	Object	Namespace
Notifications		
NotificationMessageData	notificationMessageData	Ektron.Cms.Notifications

Class	Object	Namespace
Search		
SearchManager	searchManager	Ektron.Cms.Framework.Search
AdvancedSearchCriteria	advancedSearchCriteria	Ektron.Cms.Search
KeywordSearchCriteria	keywordSearchCriteria	Ektron.Cms.Search

Class	Object	Namespace
Settings		
CmsMessageManager	cmsMessageManager	Ektron.Cms.Framework.Settings
CmsMessageData	cmsMessageData	Ektron.Cms.Messaging
CmsMessageCriteria	cmsMessageCriteria	Ektron.Cms.Messaging
CmsMessageTypeManager	cmsMessageTypeManager	Ektron.Cms.Framework.Settings
CmsMessageTypeData	cmsMessageTypeData	Ektron.Cms.Messaging

Class	Object	Namespace
Settings		
CmsMessageTypeCriteria	cmsMessageTypeCriteria	Ektron.Cms.Messaging
CustomPropertyManager	customPropertyManager	Ektron.Cms.Framework.Settings
UserCustomPropertyData	userCustomPropertyData	Ektron.Cms
CustomPropertyCriteria	customPropertyCriteria	Ektron.Cms.User
PermissionManager	permissionManager	Ektron.Cms.Framework.Settings
UserPermissionData	userPermissionData	Ektron.Cms
PermissionCriteria	permissionCriteria	Ektron.Cms.Settings
SmartFormConfigurationManager	smartFormConfigurationManager	Ektron.Cms.Framework.Settings
SmartFormConfigurationData	smartFormConfigurationData	Ektron.Cms
SmartFormConfigurationCriteria	smartFormConfigurationCriteria	Ektron.Cms.Content
TaskCategoryManager	taskCategoryManager	Ektron.Cms.Framework.Settings
TaskCategoryData	taskCategoryData	Ektron.Cms
TaskCategoryCriteria	taskCategoryCriteria	Ektron.Cms
TaskManager	taskManager	Ektron.Cms.Framework.Settings
TaskData	taskData	Ektron.Cms
TaskCriteria	taskCriteria	Ektron.Cms
TaskCommentManager	taskCommentManager	Ektron.Cms.Framework.Settings
TaxonomyCustomPropertyManager	taxonomyCustomPropertyManager	Ektron.Cms.Framework.Settings
CustomPropertyData	customPropertyData	Ektron.Cms.Common
NotificationAgentSettingManager	notificationAgentSettingManager	Ektron.Cms.Framework.Settings.Notifications
NotificationAgentData	notificationAgentData	Ektron.Cms.Notifications
NotificationAgentCriteria	notificationAgentCriteria	Ektron.Cms.Notifications
NotificationPreferenceManager	notificationPreferenceManager	Ektron.Cms.Framework.Settings.Notifications
NotificationPreferenceData	notificationPreferenceData	Ektron.Cms.Notifications
NotificationPreferenceCriteria	notificationPreferenceCriteria	Ektron.Cms.Notifications
NotificationPublishPreferenceManager	notificationPublishPreferenceManager	Ektron.Cms.Framework.Settings.Notifications
UserNotificationSettingManager	userNotificationSettingManager	Ektron.Cms.Framework.Settings.Notifications
UserNotificationSettingData	userNotificationSettingData	Ektron.Cms.Notifications
UserNotificationSettingCriteria	userNotificationSettingCriteria	Ektron.Cms.Notifications
DxHConnectionManager	dxHConnectionManager	Ektron.Cms.Framework.Settings.DxH
DxHConnectionData	dxHConnectionData	Ektron.Cms.Settings.DxH
DxHConnectionCriteria	dxHConnectionCriteria	Ektron.Cms.Settings.DxH
DXHUserConnectionManager	dXHUserConnectionManager	Ektron.Cms.Settings.DxH
DxHUserConnectionData	dxHUserConnectionData	Ektron.Cms.Settings.DxH
DxHMappingManager	dxHMappingManager	Ektron.Cms.Framework.Settings.DxH
DxHMappingData	dxHMappingData	Ektron.Cms.Settings.DxH
DxHMappingCriteria	dxHMappingCriteria	Ektron.Cms.Settings.DxH

DxHCmsMappingData	dxHCmsMappingData	Ektron.Cms.Settings.DxH
DxHCmsMappingCriteria	dxHCmsMappingCriteria	Ektron.Cms.Settings.DxH

Class	Object	Namespace
User		
UserGroupManager	userGroupManager	Ektron.Cms.Framework.User
UserGroupData	userGroupData	Ektron.Cms
UserGroupCriteria	userGroupCriteria	Ektron.Cms
UserManager	userManager	Ektron.Cms.Framework.User
UserData	userData	Ektron.Cms
UserCriteria	userCriteria	Ektron.Cms.User
UserCustomPropertyCriteria	userCustomPropertyCriteria	Ektron.Cms.User
UserTaxonomyCriteria	userTaxonomyCriteria	Ektron.Cms.User

Legacy API		
ContentAPI	contentApi	Ektron.Cms
API.Content.Content	contentApi	Ektron.Cms.API
API.Content.Taxonomy	taxonomyApi	Ektron.Cms.API
API.Content.Asset	assetApi	Ektron.Cms.API
API.Content.Blog	blogApi	Ektron.Cms.API
API.Content.ContentRating	contentRatingApi	Ektron.Cms.API
API.Content.Form	formApi	Ektron.Cms.API
API.Content.ThreadedDiscussion	threadedDiscussionApi	Ektron.Cms.API
API.Calendar.Calendar	calendarApi	Ektron.Cms.API
API.Calendar.CalendarEvent	calendarEventApi	Ektron.Cms.API
API.Calendar.CalendarEventType	calendarEventTypeApi	Ektron.Cms.API
API.Community.CommunityGroup	communityGroupApi	Ektron.Cms.API
API.Community.Favorites	favoritesApi	Ektron.Cms.API
API.Community.Flagging	flaggingApi	Ektron.Cms.API
API.Community.Friends	friendsApi	Ektron.Cms.API
API.Community.MessageBoard	messageBoardApi	Ektron.Cms.API
API.Community.Tags	tagsApi	Ektron.Cms.API
API.Search.SearchManager	searchManagerApi	Ektron.Cms.API
API.SiteMap	siteMapApi	Ektron.Cms.API
API.CustomFields	customFieldsApi	Ektron.Cms.API
API.Library	libraryApi	Ektron.Cms.API
API.Localization	localizationApi	Ektron.Cms.API
API.Messaging	messagingApi	Ektron.Cms.API
API.Metadata	metadataApi	Ektron.Cms.API
API.Permissions	permissionsApi	Ektron.Cms.API
API.UrlAliasing.UrlAliasAuto	urlAliasAutoApi	Ektron.Cms.API
API.User.User	userApi	Ektron.Cms.API

12.2. Controls

Use appropriate prefix for the UI elements so that you can identify them from the rest of the variables.

Follow the approaches recommended here.

Use appropriate prefix for each of the UI element. A brief list is given below. Since Ektron has given several controls, you may have to arrive at a complete list of standard prefixes for each of the controls you are using.

Control Name	Prefix
ActiveTopics	act
ActivityStream	avs
Analytics Tracker	ant
AssetControl	asst
Blogs	blg
Bookmarklet	bkm
BreadCrumb	bc
BusinessRules	bur
Cart	cart
Checkout	chkout
Collection (deprecated)	coll
CommunityDocuments	cmd
CommunityGroupBrowser	cmgb
CommunityGroupList	cmgl
CommunityGroubMembers	cmgm
CommunityGroupProfile	cmgp
ContentBlock (deprecated)	cb
ContentFlagging	cf
ContentList (deprecated)	cl
ContentReview	cr
CurrencySelect	crs
DesignTimeDiagnostic	dtd
Directory	dir
Favorites	fav
Flex Menu (deprecated)	fm
FolderBreadcrumb	fbc
FormBlock	fb
Forum	forum
Friends (colleagues)	frnds
ImageControl	imgc
Invite	invt
LanguageAPI	lng
LanguageSelect	lngs
ListSummary (deprecated)	ls
Login	login
Map	map
Membership	memb
Menu (deprecated)	menu
MessageBoard	mb
Messaging	msg

Metadata	meta
MetadataList (deprecated)	metalst
Micro-messaging	mm
MyAccount	ma
Control Name	**Prefix**
OrderList	ol
PhotoGallery	pg
Poll	poll
PostHistory	ph
Product	prdt
ProductList	prdl
Product Search	prds
Recommendation	recmd
RssAggregator	rss
SEO (search engine optimization)	seo
SiteMap	sm
Site Search	ssrch
SocialBar	sb
TagCloud	tc
UserProfile	up
User Search	us
WebCalendar	wc
Web Search (deprecated)	ws
XMLSearch	xmls

12.3. Project Base Setup

The WebAssets folder contains all the site files that we are developing, apart from the main files organized in the following folders.

- ✓ CSS
- ✓ Fonts
- ✓ Images
- ✓ JS
- ✓ MasterPages
- ✓ Templates
- ✓ UserControls
- ✓ Videos
- ✓ Xslts

The table explains the naming conventions to be followed for the files names.

CSS	• Do not use special characters • Do not use any spaces • The file name should start with a letter

	• Use all Pascal case • Keep the file name as short as possible • The Most Important Part of Your CSS File Name. The extension must be : .css
Fonts	This folder will contain the fonts used in the site which we are not going to name.
Images	{imageType}-{name}.{imageExtension}, where {imageType} is any of the following. • icon (e. g. question mark icon for help content) • img (e. g. a header image inserted via <img /> element) • button (e. g. a graphical submit button) • bg (image is used as a background image in css) • sprite (image is used as a background image in css and contains multiple "versions") Examples: icon-help.gif, img-logo.gif, sprite-main_headlines.jpg, bg-gradient.gif etc.
JS	script-{name}.js Examples: script-main.js, script-nav.js etc.
MasterPages	When starting a new ASP.NET website the very first thing I do is create a master page. Name it meaningful and avoid MasterPage.master Some Examples are Main.master, Site.Master for home page master file, and Inner.master, Secondary.master for content page master files.
Templates	Contains the aspx files. Should be a meaning name corresponding to the functionality of the page. Capitalize first letter, and first letter of every word.
UserControls	Again, a meaningful name prefixed by UX preferably. Example: UXUserControl.ascx
Videos	Contains any video files used in the site. Could be of any name.
Xslt	Use Pascal case for naming