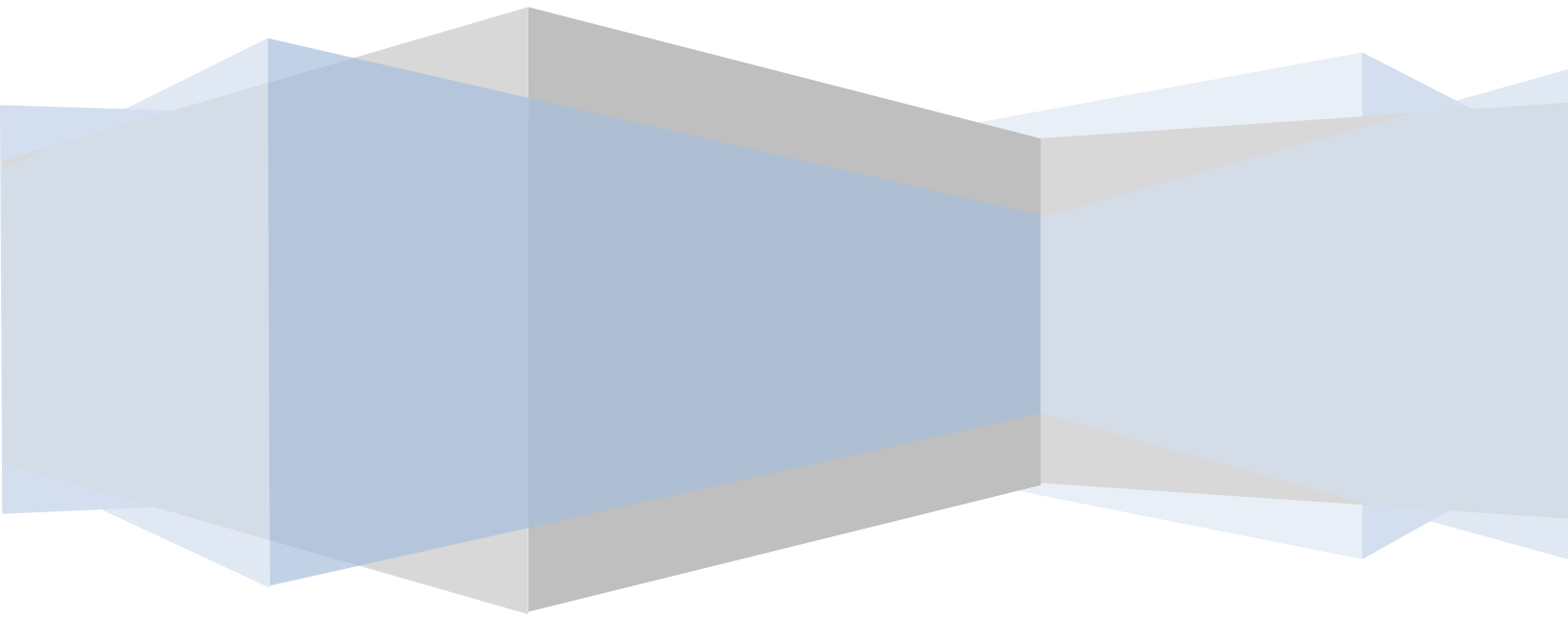
GGK Coding Standards



Contents

1. General Rules

1.1.General

1.1.1. Naming Conventions

- **Rule:**
  - Always use Camel or Upper Camel Case (also known as Pascal Case) names.
  - Avoid ALL CAPS and all lower case names.
  - Do not create declarations of the same type (namespace, class, method, property, field, or parameter) and access modifier (protected, public, private, internal) that vary only by capitalization.
  - Do not use names that begin with a numeric character.
  - Always choose meaningful and specific names.
  - Variables and Properties should describe an entity not the type or size.
  - Use uppercase for two-letter abbreviations, and UpperCamelCase (also known as Pascal Case) for longer abbreviations.
  - Do not use C# reserved words as names.
  - Avoid adding redundant or meaningless prefixes and suffixes to identifiers

    Example:

    ```
    // Bad!
    public enum ColorsEnum {…}
    public class CVehicle {…}
    public struct RectangleStruct {…}
    ```

  - Do not include the parent class name within a property name.

Example	
Legal	Customer.Name
Illegal	Customer.CustomerName

  - Try to prefix Boolean variables and properties with “Can”, “Is” or “Has”.
  - Append computational qualifiers to variable names like Average, Count, Sum, Min, and Max where appropriate.
  - When defining a root namespace, use a Product or Company as the root.

1.1.2. Abbreviations

- **Rule:** Partial abbreviations are not legal in any programing language, database design, file naming. Full words or acronyms are used.
- Full words lend themselves to clarity.
- Acronym abbreviations allow for many words to be grouped together while being concise.
  - 'Data Base' is two words and therefore 'db' or 'DB' are valid acronym abbreviations

- To reiterate, stay away from abbreviations where possible as they have many variations.
- **Exception:** Some words are very long and have acceptable abbreviations – see list below.

Abbreviation Exceptions	
Word	Valid Abbreviation
Application	App or app
Information	Info or info
Arguments / Argument	Args or args / Arg or arg
Procedure	Proc or proc
Exception	Ex or ex
Document	Doc or doc

Example	
Legal	`string submission`
Illegal	`string sub;` `string submiss;`

1.1.3. Commas

- **Rule:** Add a space character after each comma of each argument or parameter in all applicable languages specified in this document.

Example	

Legal	`public int MethodX(string firstName, string color, object someObject)`
	`{`
	`  MethodY(firstName, color, someObject);`
	`  return 0;`
	`}`

Illegal	`public int MethodX(string firstName,string color,object someObject)`
	`{`
	`  MethodY(firstName,color,someObject);`
	`  return 0;`
	`}`

1.1.4. Importing code From Internet

- **Rule:** When using code from the web (msdn, Google search, other) include the url to the page(s) that code was taken from.
  - Allows other developers to get more information on the implementation of the code in question.
  - Gives the true credit to original developer.

2. C#

2.1.Naming Conventions

2.1.1. Class

- **Rule:** All classes should be named using upper-lower (Pascal) type naming.
  - Exception: web pages are often created as all lower case directories and filenames with underscores and when these pages are created in a project they are given a namespace / class name to match, i.e. the aspx file /SomeWebProject/home/terms_of_use.aspx would be given the class name SomeWebProject.home.terms_of_use.
- Always match Class name and file name.

  Example:

  MyClass.cs => public class MyClass

  {…}
- Use a noun or noun phrase for class name.
- Add an appropriate class-suffix when sub-classing another type when possible.

  Examples:

```
private class MyClass
{…}

internal class SpecializedAttribute : Attribute
{…}

public class CustomerCollection : CollectionBase
{…}

public class CustomEventArgs : EventArgs
{…}

private struct ApplicationSettings
{…}
```

Example	

Legal	public class MyClass
	{
	public MyClass()
	{
	}
	}

Illegal	public class my_class
	{
	public my_class()
	{
	}
	}

2.1.2. Generic Class

- **Rule:** Generic Classes should be named using Upper Camel Case (Pascal Case). And use T or K as type identifier.
- Always use a single capital letter, such as T or K

Example:

```
public class Stack<T>
{
        public void Push(<T> obj)
        {…}
        public <T> Pop()
        {…}
}
```

2.1.3. Class Member Variables:

- **Rule:** All members that are not publicly exposed and Global variables to class should be prefixed with "m_".
- Try to avoid Public variables, instead you can use properties. If there is no option then use UpperCamelCase.
  - Exception: Some protected members must be named to match the database columns that they correspond to.

Example	
Legal	private string m_Name = String.Empty;
Illegal	private string Name = String.Empty;

2.1.4. Class Constants:

- **Rule:** Constants should be PascalCase.

Example	
Legal	private const int DaysPerWeek = 7;
Illegal	private string Name = String.Empty;

2.1.5. Interfaces:

- **Rule:** Interfaces should be Pascal Case and prefix with a capital I

Example	
Legal	Interface ICustomer {..}
Illegal	Interface Customer {..} Interface Icustomer {..}

2.1.6. Methods:

- **Rule:** Methods should be Pascal Case and use a verb.

  Example:

```
public void Execute() {…}
private string GetAssemblyVersion(Assembly target) {…}
```

Example	

Legal	public int GetInt()
	{
	return 0;
	}
	Public IList<Customer> GetCustomerDetails()
	{…}

Illegal	public int getInt()
	{
	return 0;
	}
	Public IList<Customer> CustomerDetails()
	{…}

2.1.7. Property:

- **Rule:** Properties should be Pascal Case. Property name should represent the entity it returns. Never prefix property names with “Get” or “Set”.

Example	

Legal	public string FirstName
	{
	get
	{..}
	set
	{..}
	}

Illegal	public string FirstNameOfCustomer
	{
	get
	{..}
	set
	{..}
	}
	public string GetFirstName

	{
	get
	{..}
	}

2.1.8. Local Variables and Method Arguments:

- **Rule:** Use "camelCase" style naming convention. All variables start with lower case for first words and use upper case to start additional words. Hungarian notation will not be considered legal.
- **Guideline:** camelCase cannot be applied to abbreviations, rv not rV.
- **Guideline:** use prefix is, do, or use for bool type variables where appropriate.
- **Guideline:** for cases where the same variable needs to be stored in multiple variables of different data types, the type name can be used as a suffix to the variable, this can also help when dealing with object types to clarify their usage.

Example	

Legal	string firstName = String.Empty;
	int count = 0;
	int i = 0;
	int rv = 0;
	NameValueCollection colors = new NameValueCollection();
	DataAccess headersConnection = new DataAccess(…);
	
	public int SomeMethod(string firstName, string color, object someObject)
	{
	return rv;
	}

Illegal	string sFirstName = String.Empty;
	int iCount = 0;
	int rV = 0;
	NameValueCollection nvcNames = new NameValueCollection();
	NameValueCollection oNames = new NameValueCollection();
	DataAccess da = new DataAccess(…);
	
	public int MethodX(string sFirstName, string sColor, object objSomeObject)
	{
	return rV;
	
	}

Example – Optional Naming	
	// same data stored in multiple datatypes

	string costString = String.Empty;
	double costDouble = 0;
	
	// optional naming for object types
	XmlDocument clientXmlDocument = new XmlDocument();
	XmlNode clientNode =
	clientXmlDocument.SelectSingleNode("/Clients/Client[@ID = '2']");

2.1.9. Enum, Delegate and Event

- **Rule:** Use UpperCamelCase (Pascal Case).

Example:

```
public event EventHandler LoadPlugin;
public enum CustomerTypes
{
    Consumer,
    Commercial
}
```

2.1.10. Linq, Lambda expressions

- **Rule:** Use Camel Case for parameters
- Don't write the entire query in one line
- If the entire query fit cleanly (user discretion) on one line then this is acceptable.

Example	
Legal	List<User> users = GetAllUsers(); List<User> activeUsers = users .Where(user => user.Active) .OrderBy(user => user.LastName) .ToList<User>(); var inActiveUsers = from user in users where user.Active orderby user.LastName select user; // In VS Studio this is all in one line List<User> activeUsers = users.Where(User => User.Active) .OrderBy(User => User.LastName).ToList<User>(); // In VS Studio this is all in one line var inActiveUsers = from u in users where u.Active orderby u.LastName select u;

2.2.Syntax Formatting

2.2.1. Code Blocks

- **Rule:** All code blocks will have starting and ending brackets that line up in the same vertical plane (path) – see examples below.

Example	

Legal	if(SomeTest())
	{
	// execute conditional processing here
	}
	else
	{
	// execute conditional processing here
	}

Illegal	if(SomeTest()) {
	// execute conditional processing here
	}
	else {
	// execute conditional processing here
	}

Example	

Legal	public string FirstName
	{
	get
	{
	return this.m_FirstName;
	}
	set
	{
	this.m_FirstName = value;
	}
	}

Illegal	public string FirstName
	{
	get { return this.m_FirstName; }
	set { this.m_FirstName = value; }
	}

2.2.2. Tabbing – variable initialization

- **Rule:** When initializing variables tabbing of the equal sign is considered illegal.

Example	

Legal	`string firstName = String.Empty;`
	`SqlDataReader dr = null;`
	`int i = 0;`
	
	`firstName = "Joe";`
	`dr = new SqlDataReader();`

Illegal	`string firstName = String.Empty;`
	`SqlDataReader dr = null;`
	`int i = 0;`
	
	`firstName = "Joe";`
	`dr = new SqlDataReader();`

2.2.3. Tabbing - Region Blocks

- **Rule:** region blocks must line up in the same vertical plane as the code that the region contains.

Example	

Legal	`#region Method X`
	`public void MethodX()`
	`{`
	`int i = 0;`
	`string sFirstName = String.Empty;`
	`}`
	`#endregion`

Illegal	`#region Method X`
	`public void MethodX()`
	`{`
	`#region init vars`
	`int i = 0;`
	`string sFirstName = String.Empty;`
	`#endregion`
	`}`
	`#endregion`

2.2.4. Nested Region Blocks

- **Rule:** region blocks cannot be nested.

Example	

Legal	#region SampleMethod
	private void SampleMethod()
	{
	  // do stuff here
	}
	#endregion

Illegal	#region SampleMethod
	private void SampleMethod()
	{
	  // do stuff here stuff
	
	  #region NestedRegion
	  // this is illegal
	  #endregion
	}
	#endregion

2.2.5. Region Block Usage

- **Rule:** Region blocks can only be used for functions, constructors and properties.
- **Guideline:** do not use region blocks for constants.
- **Guideline:** do not use region blocks for member variables.
- **Guideline:** avoid use of regions at all times if you can

Example	

Legal	// Member Variables
	private long m_MyMember;
	
	// Constants
	private const long DaysPerWeek = 7;

Illegal	#region Member Variables
	// Member Variables
	private long m_MyMember;
	#endregion
	
	#region Constants
	// Constants
	private const long DaysPerWeek = 7;
	#endregion

2.2.6. Parenthesis surrounding white spaces

- **Rule:** no white space should precede open parenthesis and no white space should be included directly inside open or closed parenthesis.
  - Exception: if method parameters are on separate lines.

Example	

Legal	`string s = GetString();`
	`string s = GetString(argument);`
	`if(isValid)`
	`{`
	`    // do stuff here ...`
	`}`

Illegal	`string s = GetString (); // Preceding white space`
	`string s = GetString( argument ); // internal white space`
	`string s = GetString ( argument ); // both`
	`if (isValid)`
	`{`
	`    // do stuff here ...`
	`}`
	`if( isValid )`
	`{`
	`    // do stuff here ...`
	`}`

2.2.7. Conditional Grouping

- **Rule:** always group individual expressions in their own parentheses.
  - Exception: if the expression has one element then parentheses are not needed.

Example	

Legal	`if((x == 1) && (i == 0))`
	`{`
	`    // do stuff here ...`
	`}`

Illegal	`if(x == 1 && i == 0)`
	`{`
	`    // do stuff here ...`
	`}`

Exception	`if(isValid && (i == 0))`
	`{`
	`    // do stuff here ...`
	`}`

2.2.8. Method Arguments

- **Rule:** if there are many arguments to a method and they do not all fit onto the visible screen then put each one on a separate line.
- Try to avoid declaring methods with more than 5 parameters. Consider refactoring this code.
- Try to replace large parameter-sets (> than 5 parameters) with one or more class or struct parameters – especially when used in multiple method signatures.

Example	

Legal	// method implementation
	private void SomeMethod
	(
	string firstName,
	string lastName,
	string ssn,
	string clientUrl
	)
	{
	// do stuff
	}
	
	// method call
	this.SomeMethod
	(
	"Joe",
	"Blow",
	this.m_Ssn,
	this.m_ClientUrl
	);
	
	// new constructor option, specify properties at initialization time
	DBUser user = new DBUser{ FirstName = "Joe", LastName = "Smith" };
	
	// new constructor option, multiline and nested
	DBUser user = new DBUser
	{
	FirstName = "Joe",
	LastName = "Smith",
	ContactType = new
	{
	Title = "Primary",
	UpdatedBy = GetLoggedOnUser()
	},

	Role = roleApi.GetByID(Enum.Role.HiringManager), UpdatedBy = GetLoggedOnUser() };

Illegal	// in studio this is all on 1 line
	private void SomeMethodWithLotsOfArgs(string firstName, string lastName, string ssn, string clientUrl)
	{
	// do stuff
	}
	
	// in studio this is all on 1 line
	this.SomeMethodWithLotsOfArgs("Joseph", "AReallyLongLastName", this.m_Ssn, "http://www.google.com/search?hl=en&lr=&q=c%23+interface+site%3Amsdn.microsoft.com");

2.2.9. For loops

- **Rule:** a space character between each statement in the "for" statement.

Example	

Legal	for(int i = 0; i < args.Length; i++)
	{
	// loop on the array
	}

Illegal	for(int i=0;i<args.Length;i++)
	{
	// loop on the array
	}
	
	int i=0;
	for(i=0;i<args.Length;i++)
	{
	// loop on the array
	}

2.2.10. Ternary operator

- **Rule:** ternary statements usually include a single line of code, if they get too long for a single line of code use the following format rule.

Example	
Legal	bool isNull = (GetDropDownControl() == null) ? true : false; GetDropDownControl().Items[i].Value = ((i < 12) ? String.Format("{0} {1}", (i % 12), times[0]) : String.Format("{0} {1}", (i % 12), times[1]));
Illegal	GetDropDownControl().Items[i].Value = (i < 12) ? String.Format("{0} {1}", (i % 12), times[0]) : String.Format("{0} {1}", (i % 12), times[1]); GetDropDownControl().Items[i].Value = (i < 12) ? String.Format("{0} {1}", (i % 12), times[0]) : String.Format("{0} {1}", (i % 12), times[1]);

2.3.Best Practices

2.3.1. If else code blocks

- **Rule:** In C# it is legal to leave curly brackets out of an if conditional in certain situations, this is considered illegal for GGK coding standards.

Example	

Legal	string s = String.Empty;
	
	if(this.SomeTest())
	{
	s = "yes";
	}
	else
	{
	s = "no";
	}

Illegal	string s = String.Empty;
	
	if(this.SomeTest())
	s = "yes";
	else
	s = "no";

2.3.2. Multi Line Statements

- **Rule:** If entire expression or string doesn't fit into one line(user discretion), then break into multiple lines in an understandable manner.

Example

Legal

```
string js = string.Format(@"var CharLeftText = '{0}';
            function UpdateCount(labelID, textboxID, characters)
            {
              try
              {
                var textBox = $('#' + textboxID);
                var length = characters - textBox.val().replace("\n)/g, ""\r\n"").length;
                if(length < 0)
                {
                  textBox.val(textBox.val().substr(0, characters));
                  length = 0;
                }
                $('#' + labelID).html(CharLeftText.replace('{CHAR}', length));
                $('#' + labelID).css('color', ((length == 0) ? 'red' : ''));
               }
              catch (Error)
              { /* eat it */ }
            } ", base.GetLabelText("CHARACTERS_REMAINING"));

string myString1 = "This is the first line of my string.\n" +
                   "This is the second line of my string.\n" +
                   "This is the third line of the string.\n";

string myString2 = @"This is the first line of my string.
                   This is the second line of my string.
                    This is the third line of the string.";
```

2.3.3. Initialize strings

- **Rule:** Initialize string variables to String.Empty.

Example	

Legal	`string s = String.Empty; // use the base class`
	`string s = string.Empty; // last resort use c# string`
	`string s = "joe";`

Illegal	`string s = "";`

2.3.4. Checking string length

- **Rule:** when checking strings for no value or Null use “String.IsNullOrEmpty”

Example	

Legal	if(String.IsNullOrEmpty(r.Title))
	{
	// string is ""
	}

Illegal	if(s == "")
	{
	// string is 0 length
	}

2.3.5. Try, catch, finally

- **Rule:** try to catch specific types of exceptions and handle those, let higher level error handling catch non-specific exceptions.
  - **Exception:** guideline only.
- **Rule:** whenever a try catch block is added by a developer they must thoroughly test for all possible paths of execution, with the application in both Development and Staging modes to determine if the appropriate actions are taking place.
  - If any database activity is taking place, force a raise error into the stored procedure executing.
  - Set an object to null so that it throws a NullReferenceException.
  - Make sure in both cases that all database connections are closed if the exception is going to stop all execution of current thread.
  - **Exception:** none – this must be done.
- **Rule:** use finally blocks without catch blocks where appropriate.
  - **Exception:** guideline only, use where it makes sense, usually when a connection or data reader must be closed.

Example	

Example	SqlDataReader dr = null;
	try
	{
	dr = cmd.ExecuteReader();
	
	while(dr.Read())
	{
	sSomeValue = dr["SomeDbColumn"].ToString();
	}
	}
	finally

```
{
        if(dr != null)
        {
                dr.Close();
        }
}
```

2.4.Language Usage

2.4.1. General

- Do not omit access modifiers. Explicitly declare all identifiers with the appropriate access modifier instead of allowing the default.

  Example:

  ```
  // Bad!
  Void WriteEvent(string message)
  {…}
  // Good!
  private Void WriteEvent(string message)
  {…}
  ```

- Do not use the default ("1.0.*") versioning scheme. Increment the AssemblyVersionAttribute value manually.
- Set the ComVisibleAttribute to false for all assemblies.
- Only selectively enable the ComVisibleAttribute for individual classes when needed.

  Example:

  ```
  [assembly: ComVisible(false)]
  [ComVisible(true)]
  public MyClass
  {…}
  ```

- Consider factoring classes containing unsafe code blocks into a separate assembly.
- Avoid mutual references between assemblies.

2.4.2. Variable and Types

- Try to initialize variables where you declare them.
- Always choose the simplest data type, list, or object required.
- Use built-in C# data type and not .NET CTS types.

  Example:

  short NOT System.Int16
  int NOT System.Int32
  long NOT System.Int64
  string NOT System.String

- Only declare member variables as private. Use properties to provide access to them with public, protected, or internal access modifiers.
- Try to use int for any non-fractional numeric values that will fit the int datatype - even variables for nonnegative numbers.
- Only use long for variables potentially containing values too large for an int.
- Try to use double for fractional numbers to ensure decimal precision in calculations.
- Only use float for fractional numbers that will not fit double or decimal.

- Avoid using float unless you fully understand the implications upon any calculations.
- Try to use decimal when fractional numbers must be rounded to a fixed precision for calculations. Typically this will involve money.
- Avoid using sbyte, short, uint, and ulong unless it is for interop (P/Invoke) with native libraries.
- Avoid specifying the type for an enum - use the default of int unless you have an explicit need for long (very uncommon).
- Avoid using inline numeric literals (magic numbers). Instead, use a Constant or Enum.

Example	

Legal	public const double Pi = 3.14159;
	double radius = 5.3;
	double area = Pi * (radius * radius);

Illegal	double radius = 5.3; double area = 3.14159 * (radius * radius);

-
- Avoid declaring string literals inline. Instead use Resources, Constants, Configuration Files, Registry or other data sources.
- Declare readonly or static readonly variables instead of constants for complex types.
- Only declare constants for simple types.
- Avoid direct casts. Instead, use the “as” operator and check for null.

  Example:

```
object dataObject = LoadData();
DataSet ds = dataObject as DataSet;
if(ds != null)
{…}
```

- Always prefer C# Generic collection types over standard or strong-typed collections.
- Always explicitly initialize arrays of reference types using a for loop.
- Avoid boxing and unboxing value types.

  Example:

```
int count = 1;
object refCount = count; // Implicitly boxed.
int newCount = (int)refCount; // Explicitly unboxed.
```

- Floating point values should include at least one digit before the decimal place and one after.

  Example: totalPercent = 0.05;
- Try to use the “@” prefix for string literals instead of escaped strings.
- Prefer String.Format() or StringBuilder over string concatenation.
- Never concatenate strings inside a loop.
- Do not compare strings to “String.Empty” or “” to check for empty strings. Instead, compare by using “String.IsNullOrEmpty”.
- Avoid hidden string allocations within a loop. Use String.Compare() for case-sensitive

  Example: (ToLower() creates a temp string)

```
// Bad!
int id = -1;
```

```
string name = “myname”;
for(int i=0; i < customerList.Count; i++)
{
if(customerList[i].Name.ToLower() == name)
{
id = customerList[i].ID;
}
}
// Good!
int id = -1;
string name = “myname”;
for(int i=0; i < customerList.Count; i++)
{
// The “ignoreCase = true” argument performs a
// case-insensitive compare without new allocation.
if(String.Compare(customerList[i].Name, name, true)== 0)
{
id = customerList[i].ID;
}
}
```

2.4.3. Flow Control

- Avoid invoking methods within a conditional expression.
- Avoid creating recursive methods. Use loops or nested loops instead.
- Avoid using foreach to iterate over immutable value-type collections. E.g. String arrays.
- Do not modify enumerated items within a foreach statement.
- Use the ternary conditional operator only for trivial conditions. Avoid complex or
- compound ternary operations.

  Example: int result = isValid ? 9 : 4;
- Avoid evaluating Boolean conditions against true or false.

  Example:

```
// Bad!
if (isValid == true)
{…}
// Good!
if (isValid)
{…}
```

- Avoid assignment within conditional statements.

  Example: if((i=2)==2) {…}
- Avoid compound conditional expressions – use Boolean variables to split parts into multiple manageable expressions.

  Example:

```
// Bad!
if (((value > _highScore) && (value != _highScore)) && (value <
_maxScore))
{…}
// Good!
```

```
isHighScore = (value >= _highScore);
isTiedHigh = (value == _highScore);
isValid = (value < _maxValue);
if ((isHighScore && ! isTiedHigh) && isValid)
{…}
```

- Avoid explicit Boolean tests in conditionals.

  Example:

  ```
  // Bad!
  if(IsValid == true)
  {…};
  // Good!
  if(IsValid)
  {…}
  ```

- Only use switch/case statements for simple operations with parallel conditional logic.
- Prefer nested if/else over switch/case for short conditional sequences and complex conditions.

2.4.4. Exceptions

- Do not use try/catch blocks for flow-control.
- Only catch exceptions that you can handle.
- Never declare an empty catch block.
- Avoid nesting a try/catch within a catch block.
- Always catch the most derived exception via exception filters.
- Avoid re-throwing an exception. Allow it to bubble-up instead.
- If re-throwing an exception, preserve the original call stack by omitting the exception argument from the throw statement.

  Example:

  ```
  // Bad!
  catch(Exception ex)
  {
  Log(ex);
  throw ex;
  }

  // Good!
  catch(Exception)
  {
  Log(ex);
  throw;
  }
  ```

- Only use the finally block to release resources from a try statement.
- Always use validation to avoid exceptions.

  Example:

  ```
  // Bad!
  try
  {
  conn.Close();
  ```

```
}
Catch(Exception ex)
{
// handle exception if already closed!
}
// Good!
if(conn.State != ConnectionState.Closed)
{
conn.Close();
}
```

2.4.5. Object Composition

- Always declare types explicitly within a namespace. Do not use the default "{global}" namespace.
- Avoid overuse of the public access modifier. Typically fewer than 10% of your types and members will be part of a public API, unless you are writing a class library.
- Consider using internal or private access modifiers for types and members unless you intend to support them as part of a public API.
- Never use the protected access modifier within sealed classes unless overriding a protected member of an inherited type.
- Avoid declaring methods with more than 5 parameters. Consider refactoring this code.
- Try to replace large parameter-sets (> than 5 parameters) with one or more class or struct parameters – especially when used in multiple method signatures.
- Do not use the "new" keyword on method and property declarations to hide members of a derived type.
- Only use the "base" keyword when invoking a base class constructor or base implementation within an override.
- Consider using method overloading instead of the params attribute (but be careful not to break CLS Compliance of your API's).
- Always validate an enumeration variable or parameter value before consuming it. They may contain any value that the underlying Enum type (default int) supports.

  Example:

  ```
  public void Test(BookCategory cat)
  {
  if (Enum.IsDefined(typeof(BookCategory), cat))
  {…}
  }
  ```

- Consider overriding Equals() on a struct.
- Always override the Equality Operator (==) when overriding the Equals() meth
- Always override the String Implicit Operator when overriding the ToString() m
- Always call Close() or Dispose() on classes that offer it.
- Wrap instantiation of IDisposable objects with a "using" statement to ensure

  Example:

  ```
  using(SqlConnection cn = new SqlConnection(_connectionString))
  {…}
  ```

3. Database

3.1.Naming Conventions

3.1.1. Table and Columns

- **Rule:** All standard tables and columns should be named using upper-lower case. Most junction tables will use the same upper-lower case but with under scores to separate the table names. Tables names rarely end in a plural i.e. Product would not be Products.
- **Rule:** The primary key for every table should be the name of the table followed by ID, i.e. Season would have the primary key of SeasonID (note ID is capitalized for both letters).

Example	
Legal	Season.Title Season_Race.RaceID
Illegal	season.title SEASON.TITLE Seasons.Title SeasonRace.RaceID

3.1.2. SQL statements – Upper Case

- **Rule:** The following SQL statements / key words will be in all upper case
  - SELECT
  - FROM
  - INSERT
  - UPDATE
  - DELETE
  - WHERE
  - JOIN (FULL, OUTER, LEFT, RIGHT and INNER)
  - AS
  - GROUP BY
  - HAVING
  - ORDER BY
  - SET
  - EXEC
  - CASE
  - WHEN
  - BEGIN
  - END
  - COUNT()
  - MAX()
  - GETDATE()
  - SCOPE_IDENTITY()
  - DECLARE
  - IF

- ELSE
- RETURN

Example	
Legal	SELECT Season.Title FROM Season WHERE Season.SeasonID = @SeasonID
Illegal	Select Season.Title from Season Where Season.SeasonID = @SeasonID

3.1.3. SQL statements – Mixed Case

➢ **Rule:** The following SQL statements / key words will be in mixed upper-lower case; these include only global variables and local variables.

- @RequisitionID
- @AssignmentID
- @@Identity
- @@Error

Example	
Legal	SET @Var = @@Identity
Illegal	SET @Var = @@IDENTITY

3.1.4. SQL statements – lower case

➢ **Rule:** The following SQL statements / key words will be in lower case; these include only data type declarations

- int
- varchar(150)
- money
- etc…

Example	
Legal	DECLARE @Var int DECLARE @VarMoney money
Illegal	DECLARE @Var INT DECLARE @VarMoney Money

3.1.5. Stored Procedures

➢ **Rule:** Upper-lower case naming with optional underscores. When appropriate stored procedures should include the name of the database tables that are being affected and the type of action taking place.

- **Example:** CustomerSubmission_StepTwo_Update; a front-end proc that deals with data mostly contained in the CustomerSubmission which does primarily update and insert operations and pertains explicitly to step two of the process.

3.1.6. User Defined functions

- **Rule:** Upper-lower case naming

Example	
Legal	CREATE FUNCTION dbo.[ActiveDisplay]
Illegal	CREATE FUNCTION dbo.[get_active_display]

3.1.7. Capitalize Aliases

- **Rule:** If using aliases, they should be capitalized.
- **Rule:** The alias should be built up from the first letter of each word in the table name.
  - Exception: multiple tables are referenced that have the same alias. A table called 'State' and one called 'Status', for example.
- **Note:** the use of AS between the alias and the table name itself is not required. Be consistent with usage, if using the AS keyword use it throughout; or vice-versa.

Example	
Legal	SELECT R.Title AS Role, U.Login AS Login, U.Email AS Email, U.Active AS Active, dbo.ApplicationCrypt(Password, 1) AS Password FROM User AS U INNER JOIN User_Role AS UR ON U.UserID = UR.UserID INNER JOIN Role AS R ON R.RoleID = UR.RoleID
Illegal	SELECT r.Title AS Role, ur.Login AS Login, ur.Email AS Email, ur.Active AS Active, dbo.ApplicationCrypt(Password, 1) AS Password FROM User AS ur INNER JOIN User_Role AS u_r ON ur.UserID = u_r.UserID

	INNER JOIN Role AS r ON r.RoleID = ur.RoleID

3.2. Syntax Formatting

3.2.1. Tabbing – If Else Blocks

- **Rule:** When IF ELSE statements use BEGIN and END blocks the tabbing will follow the example below; note that this is not ideal but the BEGIN END blocks do not lend themselves to easily readable code no matter how they are used. It is highly advisable to use comments after every END block to clarify the code.

Example	

Legal	IF @VarX = 3
	BEGIN
	SELECT X
	FROM Y
	WHERE Z = @VarX
	END -- End If @VarX = 3
	ELSE
	BEGIN
	SET @Foo = @Bar
	END -- End ELSE of If @VarX = 3

Illegal	IF @VarX = 3
	BEGIN
	SELECT X
	FROM Y
	WHERE Z = @VarX
	END
	ELSE
	BEGIN
	SET @Foo = @Bar
	END

3.2.2. Tabbing - Variable Initialization

- **Rule:** when initializing variables tabbing of the equal sign is considered illegal.

Example	
Legal	DECLARE @VarInteger = 1 DECLARE @VarChar = 'test'
Illegal	DECLARE @VarInteger = 1 DECLARE @VarChar = 'test'

3.2.3. SELECT statement data columns

- **Rule:** Each column will appear on its own line.

Example	
Legal	SELECT A.RequisitionID, A.StartDate, A.EndDate FROM Assignment A -- lends itself to FOR XML style SELECTS SELECT 1 AS TAG, NULL AS Parent, Title AS [Item!1!Title!element], ValueInteger AS [Item!1!ValueInteger!element]
Illegal	SELECT A.RequisitionID, A.StartDate, A.EndDate FROM Assignment A

3.2.4. SELECT statement table joins

- **Rule:** All keywords FROM, JOIN (INNER, OUTER, etc…) and ON statements that are used in the joining of data tables will line up in a vertical plane (path).

Example	
Legal	SELECT R.RequisitionID, R.JobTitle, A.StartDate, A.EndDate FROM Assignment A INNER JOIN Requisition R ON A.RequistionID = R.RequisitionID INNER JOIN Location_Department LD ON (A.DepartmentID = LD.DepartmentID AND A.LocationID = LD.LocationID) WHERE A.AssignmentID = @AssignmentID
Illegal	-- ON statement inline SELECT R.RequisitionID, R.JobTitle, A.StartDate, A.EndDate FROM Assignment A

```
INNER JOIN Requisition R ON A.RequistionID = R.RequisitionID
WHERE A.AssignmentID = @AssignmentID

-- ON statement tabbed
SELECT R.RequisitionID,
                R.JobTitle,
                A.StartDate,
                A.EndDate
FROM Assignment A
INNER JOIN Requisition R
        ON A.RequistionID = R.RequisitionID
WHERE A.AssignmentID = @AssignmentID
```

3.2.5. WHERE statement

- **Rule:** See complex WHERE clause in the below example use this style for indenting and determining when to move statements to the next line.

Example

Legal

```
WHERE Requisition.Active = 1
AND Requisition.Deleted = 0
AND Requisition.StatusID IN
(
        @REQUISITION_STATUS___CANCELLED,
        @REQUISITION_STATUS___CLOSED,
        @REQUISITION_STATUS___COMPLETE,
        @REQUISITION_STATUS___REJECTED
)
AND
(
        EXISTS
        (
                SELECT D.DelegatorID
                FROM Delegate D
                INNER JOIN Requisition R
                ON
                (
                        R.HiringManagerID = D.DelegatorID
                        OR
                        R.CreatedByID = D.DelegatorID
                )
                WHERE D.DelegateeID = @DBUserID
                AND DelegateTypeID = @DelegateTypeID
```

	AND R.RequisitionID = Requisition.RequisitionID)-- END EXISTS)
Illegal	WHERE Requisition.StatusID IN (@STATUS___CANCELLED, @STATUS___CLOSED) WHERE Requisition.Active = 1 AND Requisition.Deleted = 0 AND (R.HiringManagerID = D.DelegatorID OR R.CreatedByID = D.DelegatorID)

3.2.6. INSERT statements

- **Rule:** All columns in an insert statement will be on separate lines; see example below.

Example	
Legal	INSERT INTO Status (Title, DisplayTitle, UpdatedByID) VALUES (@Title, @DisplayTitle, @UpdatedByID)
Illegal	INSERT INTO Status(Title, DisplayTitle, UpdatedByID) VALUES(@Title, @DisplayTitle, @UpdatedByID)

3.2.7. UPDATE statements

- **Rule:** updates must have each column on a separate line with no tabbing of the equal sign.

Example	
Legal	UPDATE Status SET Title = @Title, DisplayTitle = @DisplayTitle, UpdatedByID = @UpdatedByID WHERE StatusID = @StatusID
Illegal	UPDATE Status SET Title = @Title,

	DisplayTitle = @DisplayTitle, UpdatedByID = @UpdatedByID WHERE StatusID = @StatusID

3.3.Best Practices

3.3.1. SELECT vs SET when setting variables

- **Rule:** If variables are being set from the result of an SQL query then SELECT should be used, otherwise SET should be used.

Example	
Legal	Declare @Var int SET @Var = 0
Illegal	Declare @Var int SELECT @Var = 0

3.3.2. Check for Active = 1

- **Rule:** Many queries do not want to include inactive records, but some do, i.e. most admin pages will return both active and inactive records. So when determining whether to check for active or not it should be based on the context of what data is being returned and what purpose it will be used.

3.3.3. Revision Comments

- **Rule:** Add revision comments at the beginning of every stored procedure, see example below. Comment should include the date in format MM/DD/YYYY, developer initials and a brief comment on what changes were applied.
  - If no previous comment exists then add in an initial comment with an unknown date, followed by the current revision; see below.

Example	
Legal	/*** Revision History ============= Ver Date Who Comment 1.0 ??/??/2002 RV init 1.1 05/01/2004 MM External identifier formatting SSN 1.2 02/15/2005 JH Coding Standard sample ***/

3.3.4. Use SCOPE_IDENTITY() instead of @@Identity

- **Rule:** Use SCOPE_IDENTITY() instead of @@Identity as it avoids issues with tables that have complex triggers that may update other tables resulting in the @@Identity variable having the wrong value.

Example	
Legal	INSERT INTO SomeTable (Title, UpdatedByID) VALUES (@Title, @UpdatedByID) SET @SomeTableID = SCOPE_IDENTITY()
Illegal	INSERT INTO SomeTable (Title, UpdatedByID) VALUES (@Title, @UpdatedByID) **SET @SomeTableID = @@Identity**

3.3.5. LEFT OUTER JOIN vs IN

- **Rule:** Use LEFT OUTER JOIN for active / available drop downs. With large amounts of data the sub-query (Illegal below) will run slowly
  - **Exception:** if legal doesn't work use illegal

Example	
Legal	SELECT DISTINCT 1 AS TAG, NULL AS Parent, Department.DepartmentID AS [Item!1!DepartmentID!element], Department.Title AS [Item!1!Title!element] FROM Department LEFT OUTER JOIN Location_Department ON Location_Department.DepartmentID = Department.DepartmentID WHERE Department.Active = 1 AND Department.Deleted = 0 AND Department.ClientID = @ItemID AND Location_Department.LocationID = @ExtraID

	FOR XML EXPLICIT
Illegal	SELECT 1 AS TAG, NULL AS Parent, Department.DepartmentID AS [Item!1!DepartmentID!element], Department.Title AS [Item!1!Title!element] FROM Department WHERE Department.Active = 1 AND Department.Deleted = 0 AND Department.ClientID = @ItemID AND Department.DepartmentID IN (SELECT DepartmentID FROM Location_Department WHERE LocationID = @ExtraID AND Location_Department.Deleted = 0) FOR XML EXPLICIT

4. ASPX Forms

4.1. Naming Conventions

4.1.1. ASPX Controls

- **Rule:** Use Pascal case, prefixed by the item in question suffixed by the control type, note that no Hungarian notation should be used, use {Item}{Control Type}. Although aspx controls are actually protected members of the Page class we treat them as if they are public properties, at least as far as naming conventions go.
- **Exceptions:** There are many exceptions to this rule, these are mainly to do with control names being too long and when used as the suffix for a variable the name can become even longer, see table below for acceptable exceptions. The majority of these are validator controls, not all are listed, follow the same convention as the examples.

Control Naming Non Standard	
Control Type	**Valid Suffix**
DropDownList	DropDown - LocationDropDown
Label	Label or LabelData – FirstNameLabel or FirstNameLabelData Labels that end in Data are for binding transactional data
ValidationSummary	ValidationSummary Summaries tend to encompase multiple

	controls, so no prefix is usually needed.
RequiredFieldValidator	FirstNameRequired
RangeValidator	QuantityRange
RegularExpressionValidator	EmailRegEx NOTE: this naming convention breaks our most important rule of not abbreviating varibables, this is an exception and is normally considered bad practice.

Example	
Legal	<asp:Label ID="FirstNameLabel" Text="First Name" runat="server" > </asp:Label> <asp:Label ID="FirstNameLabelData" Text="Joe" runat="server" > </asp:Label> <asp:TextBox ID="LastNameTextBox" runat="server" > </asp:TextBox> <asp:DropDownList ID="LocationDropDown" runat="server" > </asp:DropDownList>
Illegal	<asp:Label ID="lblFirstName" RunAt="server" > </asp:Label>

	<asp:Label ID="lblFirstNameData" runat="server" > </asp:Label> <asp:TextBox ID="txtLastName" RunAt="server" > </asp:TextBox> <asp:DropDownList ID="ddlLocation" RunAt="server" > </asp:DropDownList>

4.2.Syntax Formatting

4.2.1. Attributes Placement

- **Rule:** Attributes should be on individual lines entirely or on separate lines entirely.
  - If several attributes exist for a tag they should be placed on separate lines. One attribute per line.
  - If the entire tag can fit cleanly (user discretion) on one line then this is acceptable.
  - Register tags at the top of markup files. Register tags at the top of files should be on a single line.

Example	
Legal	<asp:Label ID="FirstNameLabel" Enabled="true" runat="server" > </asp:Label> <asp:Label ID="FirstNameLabel" runat="server"></asp:Label>
Illegal	<asp:Label ID="FirstNameLabel" runat="server" Enabled="true" > </asp:Label> <asp:Label ID="FirstNameLabel" runat="server"

	Enabled="true" > </asp:Label>

Example – exception to the rule	
Exception	<%@ Page Language="C#" AutoEventWireup="true" CodeFile="test_01.aspx.cs" Inherits="test_01" %>

4.2.2. Attribute Location

- **Rule:** The 'ID' property is always the first of the properties.
- **Rule:** If 'runat' is needed, it is always placed at the end of the attribute list.

Example	
Legal	<asp:Label ID="FirstNameLabel" Enabled="true" runat="server" > </asp:Label>
Illegal	<asp:Label RunAt="server" ID="FirstNameLabel" Enabled="true" > </asp:Label>

4.2.3. Attribute Casing

- **Rule:** Properties are 'PascalCase'
  - Exception: runat, .NET's default is lower case, let's not fight it.

Example	
Legal	<asp:Label ID="FirstNameLabel" Enabled="true" runat="server" > </asp:Label>
Illegal	<asp:Label runat="server" id="FirstNameLabel" enabled="true" >

	</asp:Label>

4.2.4. Div tags

- **Rule:** In cases where a label and a text box (or drop down, other) are on the same row use a div tag to encapsulate both items. This allows for both controls to be enabled/disabled together using the div tag.

Example	
Legal	 <asp:Label ID="ControlNameLabel" SkinID="ControlLabel" Text="[Control Name]:" runat="server" /> <asp:TextBox ID="ControlNameTextBox" MaxLength="150" Width="200" runat="server"></asp:TextBox> <asp:RequiredTextValidator ID="ControlNameTextBoxRequired" ControlIDToEvaluate="ControlNameTextBox" SummaryErrorMessage="[todo]localize" runat="server" />
Illegal	<asp:Label ID="Label1" runat="server" SkinID="ControlLabel" Text="[Control Name]:" /> <asp:TextBox ID="ControlNameTextBox" MaxLength="150" Width="200" runat="server"></vam:TextBox> <asp:RequiredTextValidator ID="ControlNameTextBoxRequired" runat="server" ControlIDToEvaluate="ControlNameTextBox" SummaryErrorMessage="[todo]localize"

	/>

4.2.5. Tag Prefix

- **Rule:** Custom controls should have tag prefix with project Name

4.3.Best practices

4.3.1. More than one Validation Summary

- **Rule:** if there are more than one Validation Summary controls i.e to prefix the section name to ValidationSummary.

Example	
Legal	 <asp:TextBox ID="EmailTextbox" runat="server"></asp:TextBox> <asp:RegularExpressionValidator ValidationGroup="Email" ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*" runat="server"> </asp:RegularExpressionValidator> <asp:Button ID="EmailVerifyButton" OnClick="OnClickEmailVerifyButton" ValidationGroup="Email" runat="server" /> <asp:ValidationSummary ID="EmailValidationSummary" ValidationGroup="Email" runat="server" /> <asp:TextBox ID="NameTextBox" runat="server"></asp:TextBox> <asp:RequiredFieldValidator ControlToValidate="NameTextBox" ValidationGroup="Name" runat="server"> </asp:RequiredFieldValidator> <asp:Button ID="NameVerifyButton" OnClick="OnClickNameVerifyButton" ValidationGroup="Name"

```
            runat="server"
        />
        <asp:ValidationSummary
            ID="NameValidationSummary"
            ValidationGroup="Name"
            runat="server"
        />
</div>
```

4.4.Code Behind method naming guidelines

4.4.1. Guidelines

- **Rule:** Note the following guidelines for method naming in code behind aspx files
- Button event: On{Action}{Item / Control}
  - For example the OnClick event of a button with id SubmitButton would be OnClickSubmitButton, normally the VS auto generated would create a method like SubmitButton_Click
  - Prefix event handler implementations with On where appropriate.
- Data loading: Load{Item / Control}
  - For example LoadLocationGrid, this is where data binding to the LocationGrid would occur
- Initialize: Initialize{Object}, methods that usually construct and set needed properties
  - i.e. InitializeLog4Net
- Invoke: Invoke{Object}, method that instantiate objects usually via reflection
  - i.e. InvokeGLStrategy
- Set: Set{Member}, called from within an object, used to set its own members
  - i.e. SetItemID, would set the m_ItemID member
- Handle: Handle{Item / Control}, to identify that a controls properties are being manipulated, usually visibility or formatting, not data binding
  - i.e. HandleFirstNameLabel

Example	

Legal	Protected void OnClickSubmitButton(object sender, EventArgs e)
	{
	// code goes here
	}

Illegal	Protected void SubmitButton_Click(object sender, EventArgs e)
	{
	// code goes here
	}

4.4.2. Method Coding

- **Rule:** Specific code should not exist in a code behind method. A helper method should be called to perform the desired actions. This will allow for code reuse.

- Validation can be performed in the code behind method before performing the desired action.
- If needed code several action methods. For example, UpdateRequisition() followed by UpdatedRequisitionActivityLog().
- Note, the Page.IsValid call below, this checks the page validity (validation controls) before continuing the desired action.

Example	

Legal	protected void OnClickSubmitButton(object sender, EventArgs e)
	{
	  if(Page.IsValid)
	  {
	  UpdateRequisition(sender, e);
	  }
	}

Illegal	protected void OnClickSubmitButton(object sender, EventArgs e)
	{
	  if(Page.IsValid)
	  {
	  m_Requisition.HiringManager = HiringManagerDropDown.GetDbUser();
	  m_Requisition.ReportsTo = ReportsToDropDown.GetDbUser();
	  m_Requisition.Location = LocationDropDown.GetLocation();
	  m_Requisition.Department = DepartmentDropDown.GetDepartment();
	  m_Requisition.RequisitionType = RequisitionTypeDropDown.GetRequisitionType();
	
	  m_Api.StepOneUpdate(m_Requisition);
	  }
	}

5. HTML

5.1.Naming Conventions

5.1.1. Html Markup

➢ **Rule:** All html markup (element tags and attributes) should be in lower case and all attribute values should be quoted using double quotes.

- Exception: calls to java script events are often in upper-lower case. Also auto generated html from Microsoft products will usually destroy html, so unfortunately we currently have no control over this.

Example	
Legal	<select name="DropDown" onChange="Execute();">

Illegal	<SELECT name=DropDown onChange=Execute();>

5.1.2. Html form input naming

- **Rule:** Html form input controls should be named using upper-lower case naming; in addition these will usually match the name of a data base column that they correspond to.
  - Exception: may vary depending on the situation.

Example	
Legal	<input type="text" name="FirstName" />
Illegal	<input type="text" name="first_name" />

5.2.Syntax Formating

5.2.1. Tabbing

- **Rule:** All nested element sections should be tabbed in – see example.
  - Exception: may vary depending on the situation – often the head section is not tabbed in so that the body tag starts at the left most side.

Example	
Legal	<table> <tr> <td>This is some content</td> </tr> </table>
Illegal	<table> <tr> <td>This is some content</td> </tr> </table>

6. JavaScript

6.1.Naming conventions

Rule: Use all applicable rules from C# section such as for, switch, while, if, etc.

6.1.1. Function

Rule: Use Camel Case for function names and for its parameters. Place the braces in separate line

Example

Legal	function action(tip, list)
	{
	// your code
	}

Illegal	function Action(tip,list) {
	// your code
	}

6.1.2. Jquery

Rule: Don't write entire jquery code in a line. If the entire tag can fit cleanly (user discretion) on one line then this is acceptable.

Example	

Legal	$("ul.tierList li").bind
	({
	"click": function (e)
	{
	if (!e.ctrlKey)
	{
	$(this).parent().children().removeClass("selectedItem");
	}
	
	if (e.which != 3)
	{
	$(this).addClass("selectedItem");
	}
	},
	"mousedown": function (e)
	{
	$(this).parent().removeClass("selectedItem");
	}
	});

Illegal	// In VS Studio it is in one line
	$("ul.tierList li").bind({ "click": function (e) { if (!e.ctrlKey)
	{$(this).parent().children().removeClass("selectedItem"); } if (e.which != 3) {
	$(this).addClass("selectedItem"); }},"mousedown": function (e)
	{$(this).parent().removeClass("selectedItem"); } });

6.2.Syntax Formating

6.2.1. Syntax

Rule: Use all applicable rules from C# section.

6.3.Best Practices

6.3.1. Variable Declaration

Rule: All variables should be declared using the “var” keyword to minimize the overheads in traversing the scope chain. Do not use global variables unless absolutely necessary as they are always the slowest to lookup. Variables should be initialized to a default value or explicitly set to null.

Example	
Legal	var count = 0; var empty = null;
Illegal	var count, p; var emptyl;

6.3.2. Semi Colons

Rule: JavaScript allows any expression to be used as a statement and uses semi-colons to mark the end of a statement. However, it attempts to make this optional with "semi-colon insertion", which can mask some errors and will also cause JS aggregation to fail. All statements should be followed by “;”. Except for the following for, function, if, switch, try, while.

Example	

Legal	var count = 0;
	
	function action(tip, list)
	{
	  count = count + 1;
	}

Illegal	var count = 0
	
	function action(tip, list) {
	  count = count + 1
	}

7. CSS

7.1. Naming Conventions

7.1.1. ID and Class selectors

- **Rule:** Use Camel Case. Don’t appending control name to css class.
- If you don't have to reference specific object, it is strongly recommended to use class instead of id selectors especially if they share the same formatting.
- Declare the hacks as the last property.

Example	
Legal	.infoSet { margin: 5px 0; font-size: 12px; *font-size: 15px; /*IE7 Hack*/ } #mainBanner { float: left; }

Illegal	.emailLabel { float: left; }
	.emailTextbox{ float: left; }
	.InfoSet
	{
	*font-size: 15px; /*IE7 Hack*/
	margin: 5px 0;
	font-size: 12px;
	}
	#mainBanner {
	float: left;
	}

7.2.Syntax Formatting

7.2.1. Syntax

- Use one line to declare css. If it is big then try to break into many classes.
- Observe the spacing for css for below example.

Example	
Legal	.infoSet { margin: 5px 0; font-size: 12px; *font-size: 15px; /*IE7 Hack*/ } #mainBanner { float: left; }

Illegal	.InfoSet
	{
	margin: 5px 0;
	font-size: 12px;
	}
	#mainBanner {
	float: left;
	}

7.3.Best Practices

7.3.1. Structural Naming Convention

- **Rule:** Structural naming convention, in essence just means that you name (by assigning a class and/or id attribute to them) elements by describing what they are, and not where they are or how the look. Its counterpart is called presentational naming which describes the location and/or appearance of web page elements.
- **Exception:** “#header”, “#footer”, “#sidebar” can be declared as name because even though they describe “where they are”, as it is universally accepted.
- **When assigning a class or id, ask yourself “What is this element for?”**
  - For example, if you have a span class that’s meant for captions, you might call it .caption or .figure-caption instead of .smaller-text.

- If a particular group of images are banner advertisements, you might want to call the class .banner or .ad-banner instead of .sidebar-images (since they may change locations).

➢ **Avoid using names that rely on locational or visual aspects of the particular element**

- For example, don't use ".redLink" for links that are red. Instead use ".externalLink". The practical reason for doing this is that it keeps your HTML markup accurate even if we change the style of the .red-link class to green or brown.

Example	
Legal	.externalLink { color: Red; }
Illegal	.redLink { color: Red; }

7.3.2. Expressions

➢ **Rule:** Never use expressions unless you absolutely have to. It is better and more compatible to use javascript if you need feature like that.

- **Example:** .someclass { background-color: expression((new Date()).getHours()%2 ? "#B8D4FF" : "#F08A00"); }

7.3.3. Multiple CSS files

➢ **Rule:** It is better to have one large css file rather than multiple small files because that way you reduce http requests and since browsers cache css, you will see the performance benefits right after the first page load.

7.3.4. External CSS file

➢ **Rule:** Always use external CSS. Never use internal or in-line CSS.