.NET CODING STANDARDS

V1.1

PROJECT
RELEASE/Version

Documentation Location:

Repository	
Location	
Document	

Version History/Approval Details:

Contents

1. General Rules

1.1. General

1.1.1. Naming Conventions

- **Rule:**

- Always use either Camel or 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 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.

- Do not use abbreviations or contractions as parts of identifier names. For example, use GetWindow instead of GetWin.Do not use acronyms that are not generally accepted in the computing field.
- Where appropriate, use well-known acronyms to replace lengthy phrase names. For example, use UI for User Interface and OLAP for On-line Analytical Processing.
- When using acronyms, use Pascal case or camel case for acronyms more than two characters long. For example, use HtmlButton or htmlButton. However, you should capitalize acronyms that consist of only two characters, such as System.IO instead of System.Io.
- Do not use abbreviations in identifiers or parameter names. If you must use abbreviations, use camel case for abbreviations that consist of more than two characters, even if this contradicts the standard abbreviation of the word.

- **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, search url, 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 in Pascal case.
- **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:

  Customer.cs => public class Customer

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

  Examples:

```
private class Customer
{…}

internal class SpecializedAttribute : Attribute {…}

public class CustomerCollection : CollectionBase
{…}

public class CustomEventArgs : EventArgs
{…}

private struct ApplicationSettings
{…}
```

Example	

Legal	public class LineItem
	{
	public LineItem()
	{
	}
	}

Illegal	public class line_item
	{
	public line_item()
	{
	}
	}

2.1.2. Generic Class

- **Rule:** Generic Classes should be named using Pascal Case.
- Always use T or K or V as type identifiers.

Example: public class

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

2.1.3. Class Member Variables:

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

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

2.1.4. Class Constants:

- **Rule:** Constants should be in Uppercase with Underscore seperated.

Example	
Legal	private const int DAYS_PER_WEEK = 7;
Illegal	private const int DaysPerWeek = 7;

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 GetCustomerID()
	{
	return 0;
	}
	Public IList<Customer> GetCustomers()
	{…}

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

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 for all variables.
- **Guideline:** camelCase cannot be applied to abbreviations, 'HtmlText' not 'htmlText'.
- **Guideline:** use prefix is, has or can 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 ID = 0;
	NameValueCollection colors = new NameValueCollection(); DataAccess headersConnection = new DataAccess(…);
	
	public int GetCustomer(string firstName, string color, Employee activeEmployee)
	{
	return ID;
	}

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 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) in 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. General guidelines

- Layout Conventions
- Use the default Code Editor Settings (smart indenting, four-character indents, tabs saved as spaces).
- Write only one statement per line.
- Write only one declaration per line.
- If continuation lines are not indented automatically, indent them one tab stop (four spaces).
- Add at least one blank line between method definitions and property definitions.

- Commenting Conventions
- Place the comment on a separate line, not at the end of a line of code.
- Begin comment text with an uppercase letter.
- End comment text with a period.
- Insert one space between the comment delimiter (//) and the comment text, as shown in the following example.

Example	
Legal	// The following declaration creates a query. It does not // run the query.
Illegal	//The following declaration creates a query. It does not //run the query.

- Do not create formatted blocks of asterisks around comments.

2.2.2. String Data Type

- Use StringBuilder object when you are working with large amount of text

Example	
Legal	`var phrase = "lalalalalalal"; var manyPhrases = new StringBuilder(); for (var i = 0; i < 10000; i++) { manyPhrases.Append(phrase); }`
Illegal	`var phrase = "lalalalalalal"; String manyPhrases = String.Empty; for (var i = 0; i < 10000; i++) { manyPhrases += phrase; }`

- For strong string concatenation use + operator

Example

Legal	`// For short strings operations`
	`string displayName = nameList[n].LastName + ", " +`
	`nameList[n].FirstName;`

Illegal	`var displayName = new StringBuilder();`
	`displayName.Append(nameList[n].LastName);`
	`displayName.Append(", ");`
	`displayName.Append(nameList[n].FirstName);`

2.2.3. Implicitly Typed Local Variables

- Use implicit typing for local variables when the type of the variable is obvious from the right side of the assignment, or when the precise type is not important.
- Avoid the use of var in place of dynamic.

Example	

Legal	`// When the type of a variable is clear from`
	`the context, use var   // in the declaration.`
	`var description = "This is clearly a string.";`
	`var id = 27;`
	`var input = Convert.ToInt32(Console.ReadLine());`

Illegal	`var input = Console.ReadLine();`

2.2.4. 'using' statement

- Simplify your code by using the C# using statement. If you have a try-finally statement in which the only code in the finally block is a call to the Dispose method, use a using statement instead.

Example	

Legal	`using (Font buttonFont = new Font("Arial", 10.0f))`
	`{`
	`    byte charset = buttonFont.GdiCharSet;`
	`}`

Illegal	// This try-finally statement only calls Dispose in the finally block.
	Font buttonFont = new Font("Arial", 10.0f);
	try { byte charset = buttonFont.GdiCharSet;
	} finally { if (buttonFont != null)
	{
	((IDisposable)buttonFont).Dispose();
	}

	}

2.2.5. New Operator

- Use object initializers to simplify object creation.

Example	

Legal	var instance = new ExampleClass
	{
	Name = "Desktop",
	ID = 37414,
	Location = "Redmond",
	Age = 2.3
	};

Illegal	var instance = new ExampleClass();
	instance4.Name = "Desktop";
	instance4.ID = 37414;
	instance4.Location = "Redmond";
	instance4.Age = 2.3;

2.2.6. 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._firstName;
	}
	set
	{
	this._firstName = value;
	}
	}

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

2.2.7. 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
	rowCount = 0;
	
	firstName = "Joe";
	dr = new SqlDataReader();

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

2.2.8. 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 Default Values
	public void DefaultValues()
	{
	int index = 0;
	string firstName = String.Empty;
	}
	#endregion

Illegal	#region Method Default Values
	public void DefaultValues()
	{
	#region init vars
	int index = 0;
	string firstName = String.Empty;
	#endregion
	}
	#endregion

2.2.9. 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

2.2.10. 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
- **Guideline:** Avoid using regions inside method

Example	

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

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

2.2.11. 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 name = GetName(); string
	address = GetAddress(argument);
	if(isValid)
	{
	// do stuff here ...
	}

Illegal	string name = GetName (); // Preceding white space string
	address = GetAddress(argument); // internal white space
	string name = GetName(argument); // both if
	(isValid)
	{
	// do stuff here ...
	}
	if(isValid)
	{
	// do stuff here ...
	}

2.2.12. 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((id == 1) && (index == 0))
	{
	// do stuff here ...
	}

Illegal	if(id == 1 && index == 0)
	{
	// do stuff here ...
	}

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

2.2.13. 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._ssn,
	this._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.ssn, "http://www.google.com/search?hl=en&lr=&q=c%23+interface+site%3Amsdn.microsoft.com");

2.2.14. For loops

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

Example	

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

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

2.2.15. 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[index].Value = ((index < 12) ? String.Format("{0} {1}", (i % 12), times[0]) : String.Format("{0} {1}", (i % 12), times[1]));
Illegal	GetDropDownControl().Items[index].Value = (index < 12) ? String.Format("{0} {1}", (index % 12), times[0]) : String.Format("{0} {1}", (index % 12), times[1]); GetDropDownControl().Items[index].Value = (index < 12) ? String.Format("{0} {1}", (index % 12), times[0]) : String.Format("{0} {1}", (index % 12), times[1]);

2.3. **Best Practices**

2.3.1. if else code blocks

- **Rule:** Code related to conditional/loop statements must be enclosed in curly brackets even it is a single statement.

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 quisition.Deleted = 0 AND quisition.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"
        />
    </div>
<div id="NameDiv">
        <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. ASP.NET MVC

5.1. Best Practices

5.1.1.Controllers:

- Delete the demo code files like AccountController.cs file

- Reduce the coupling between controllers and other dependencies like data access component, exception and logging blocks, etc.
- Controllers should be as slim as possible and contain much less code. Ideally, it should just delegate the control flow to some business logic component inside controller class. The controller in an ASP.Net MVC application should be isolated from the data access layer
- The controller is responsible to render the appropriate view at runtime based on certain action.

5.1.2.Views:

- Use strongly-typed views wherever possible
- The views should be lean and should not contain any business logic code
- Use HtmlHelpers only when you need conditional decisions to be taken on the data through the views.
- The HtmlHelpers should never contain code that invokes the data access layer, i.e., it should refrain from writing data access logic inside the HtmlHelpers.
- Do not put JavaScript code inside the view, separate them into distinct script files.

5.1.3.Bundling and minifying the script and CSS files

- Too many requests from a single HTML page may cause significant delays and affect the overall time-to-last-byte metrics for a site. Bundling is therefore the process of grouping distinct resources such as CSS or JS files into a single downloadable resource. In this way, multiple and logically distinct CSS or JS files can be downloaded through a single HTTP request

5.1.4. Cache your data

- Caching is a technique that enables you to store relatively stale data in the memory so as to reduce the network bandwidth consumption

```
public class IDGController : Controller

{

   [OutputCache(Duration=3600, VaryByParam="none")]
```

```
    public ActionResult Index()

    {

    }

}
```

5.1.5. Beware of mixing up await/async with Task.Wait. await will not block the current thread but simply instruct to compiler to generate a state-machine. However, Task.Wait will block the thread and may even cause dead-locks

6. Entity Framework

6.1. Best Practices

6.1.1. **Only use var when the type is very obvious.**

Example	
Legal	var q = from order in orders where order.Items > 10 and order.TotalValue > 1000; var repository = new RepositoryFactory.Get<IOrderRepository>(); var list = new ReadOnlyCollection<string>(); In all of three above examples it is clear what type to expect. For a more detailed rationale about the advantages and disadvantages of using var, read Eric Lippert's Uses and misuses of implicit typing. https://blogs.msdn.microsoft.com/ericlippert/2011/04/20/uses-and-misuses-ofimplicit-typing/

Illegal	var i = 3; // what type? int? uint? float?
	var myfoo = MyFactoryMethod.Create("arg"); // Not obvious what base-class or
	// interface to expect. Also difficult
	// to refactor if you can't search for
	// the class

6.1.2. **Use Lambda expressions instead of delegates**

- Lambda expressions provide a much more elegant alternative for anonymous delegates

Example	
Legal	Customer c = Array.Find(customers, c => c.Name == “Tom”); Or even better var customer = customers.Where(c => c.Name == “Tom”);
Illegal	Customer c = Array.Find(customers, delegate(Customer c) { return c.Name == “Tom”; });

6.1.3. **Being too greedy with Rows**

- It would be far more efficient to let SQL Server (which is designed for exactly this kind of operation and may even be able to use indexes if available) do the filtering instead, and transfer a lot less data.

Example	
Legal	`List<School> newYorkSchools = db.Schools.Where(s => s.City == city).ToList();`

Illegal	`string city = "New York";`
	`List<School> schools = db.Schools.ToList();`
	`List<School> newYorkSchools = schools.Where(s => s.City ==`
	`city).ToList();`

6.1.4. **Being too greedy with Columns**

Let’s say we want to print the name of every pupil at a certain **SchoolId**. We can do:

```
int schoolId = 1;

List<Pupil> pupils = db.Pupils
    .Where(p => p.SchoolId == schoolId)
    .ToList();

foreach(var pupil in pupils)
{
```

```
    textBox_Output.Text += pupil.FirstName + " " + pupil.LastName;
textBox_Output.Text += Environment.NewLine; }
```

The problem here is that, at the point when the query is run, EF has no idea what properties you might want to read, so its only option is to retrieve all of an entity's properties, i.e. every column in the table. That causes two problems

- **We're** transferring more data than necessary. This impacts everything from SQL Server I/O and network performance, through to memory usage in our client application.
- By selecting every column (effectively running a "Select * From…" query), we make it almost impossible to index the database usefully. A good indexing strategy involves considering what columns you frequently match against and what columns are returned when searching against them, along with judgements about disk space requirements and the additional performance penalty indexes incur on writing. If you always select all columns, this becomes very difficult

```
var pupils = db.Pupils
    .Where(p => p.SchoolId == schoolId)
    .Select(x => new { x.FirstName, x.LastName })
    .ToList();
```

6.1.5. **Minimizing the trips to the database**

Example	

Legal	`List<School> schools = db.Schools`
	`    .Where(s => s.City == city)`
	`    .Include(x => x.Pupils)`
	`    .ToList();`

Illegal	`string city = "New York";`
	`List<School> schools = db.Schools.Where(s => s.City == city).ToList();`
	`var sb = new StringBuilder(); foreach(var school in schools)`
	`{`
	`    sb.Append(school.Name);`
	`sb.Append(": ");`
	`    sb.Append(school.Pupils.Count);`
	`sb.Append(Environment.NewLine); }`

6.1.6. If you know you only want to read data from a database (for example in an MVC Controller which is just fetching data to pass to a View) you can explicitly tell Entity Framework not to do this tracking:

Example	

Legal	`string city = "New York";`
	`List<School> schools = db.Schools`
	`.AsNoTracking()`
	`.Where(s => s.City == city)`
	`.Take(100)`
	`.ToList();`

Illegal	`string city = "New York";`
	`List<School> schools = db.Schools`
	`.Where(s => s.City == city)`

	`.Take(100)` `.ToList();`

6.1.7. It's essential to dispose contexts once you're done with them. It's best to do this by only creating contexts in a "using" block, but if there are good reasons, you can manually call Dispose() on the context instead when you're done with it.If contexts aren't disposed they cause performancedamaging work for the Garbage Collector, and can also hog database connections which can eventually lead to problems opening new connections to SQL Server.

7. HTML

7.1. Naming Conventions

7.1.1. 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.HTML5 allows mixing uppercase and lowercase letters in element names.
- Mixing uppercase and lowercase names is bad
- Lowercase look cleaner
- Lowercase are easier to write
- 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();>

7.1.2. 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" />

7.2. **Syntax Formatting**

7.2.1. 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>

7.2.2. Close All HTML Elements

- In HTML5, you don't have to close all elements (for example the <p> element). We recommend closing all HTML elements.

Example	

Legal	`<section>`
	`  <p>This is a paragraph.</p>`
	`  <p>This is a paragraph.</p>`
	`</section>`

Illegal	`<section>`
	`  <p>This is a paragraph.`
	`  <p>This is a paragraph.`
	`</section>`

7.2.3. Avoid Long Code Lines

- When using an HTML editor, it is inconvenient to scroll right and left to read the HTML code. Try to avoid code lines longer than 80 characters.

7.3. Best practices

7.3.1. Write Standards-Compliant Markup

- We have to pay close attention when writing HTML and be sure to nest and close all elements correctly, to use IDs and classes appropriately, and to always validate our code.

Example

Legal	New items on the menu today include caramel apple cider and breakfast crepes. The caramel apple cider is delicious.
Illegal	New items on the menu today include caramel apple cider and breakfast crepes . The caramel apple cider is delicious.

7.3.2. Make Use of Semantic Elements

- We need to research and double-check our code to ensure we are using the proper semantic elements. Users will thank us in the long run for building a more accessible website, and your HTML will arguably be easier to style.

Example	
Legal	Welcome Back It has been a while. What have you been up to lately?
Illegal	Welcome Back It has been a while. What have you been up to lately?

7.3.3. Use the Proper Document Structure

- Always be sure to the use proper document structure, including the <!DOCTYPE html> doctype, and the <html>, <head>, and <body> elements.

Example	
Legal	<!DOCTYPE html> <html> <head> <title>Hello World</title> </head> <body> Hello World This is a web page. </body> </html>
Illegal	<html> Hello World This is a web page. </html>

7.3.4.Use the Alternative Text Attribute on Images

Example	
Legal	`<img src="GGK.jpg" alt="GGK Technologies">`
Illegal	`<img src="GGK.jpg">`

7.3.5.Separate Content from Style

- Never, ever, use inline styles within HTML. Doing so creates pages that take longer to load, are difficult to maintain, and cause headaches for designers and developers. Instead, use external style sheets, using classes to target elements and apply styles as necessary.

Example	
Legal	`<p class="alert-success">Thank you!</p>`
Illegal	`<p style="color: #393; font-size: 24px;">Thank you!</p>`

7.3.6. Avoid a Case of “Divitis”

- **Rule:** When writing HTML, it is easy to get carried away adding a element here and a element there to build out any necessary styles. While this works, it can add quite a bit of bloat to a page, and before too long we’re not sure what each element does. We should use the HTML5 structural elements where suitable.

Example	

Legal	`<div class="container">`
	`  <article>`
	`  <h1>Headlines Across the World</h1>`
	`  </article>`
	`</div>`

Illegal	`<div class="container">`
	`  <div class="article">`
	`  <div class="headline">Headlines Across the World</div>`
	`</div>`
	`</div>`

7.3.7.General Markup Guidelines

- Use actual P elements for paragraph delimiters as opposed to multiple BR tags.
- Make use of DL (definition lists) and BLOCKQUOTE, when appropriate.
- Items in list form should always be housed in a UL, OL, or DL, never a set of DIVs or Ps. Make use of THEAD, TBODY, and TH tags (and Scope attribute) when appropriate.
- Headings should show a hierarchy indicating different levels of importance from the top down starting with h1 having the largest font size.
- Make use of these tags (h1 to h6) wherever appropriate rather than putting all headings in P or DIV tags

7.3.8.Do not add blank lines without a reason

- For readability, add blank lines to separate large or logical code blocks.
- Do not use unnecessary blank lines and indentation. It is not necessary to use blank lines between short and related items. It is not necessary to indent every element

7.3.9.Client side data storage

- Use local storage or session storage introduced with HTML5 rather than cookies
  Reference: http://technobytz.com/cookies-vs-html-5-web-storage-comparison.html
- Always check applicability of storage type for requirement in hand

7.3.10. Avoid indentation with non-breaking space

- Avoid indentation with non-breaking space or a series of non-breaking space like
- Prefer using CSS instead

  Caution: Each browser has its own interpretation for spaces and characters

7.3.11. Use of figurecaption Element to describe image

- Before HTML5, people relied on other HTML tags to add figure caption
- HTML5 rectifies this with the introduction of the <figure> element. When combined with the <figcaption> element, we can now semantically associate captions with their image counterparts. Note that standard does not say that always use figcaption, in some of the cases it may not be needed, but when there is such a requirement, go for this one rather that other sub-optimal approaches

Example

Legal	<figure> <figcaption> This is an image of something interesting. </figcaption> </figure>
Illegal	 Image of Mars.

7.3.12. No need of type for scripts and links

Example	
Legal	<link rel="stylesheet" href="path/to/stylesheet.css" /> <script src="path/to/script.js"></script>
Illegal	<link rel="stylesheet" href="path/to/stylesheet.css" type="text/css" /> <script type="text/javascript" src="path/to/script.js"></script>

7.3.13. Use HTML5 input types and regular expressions instead of JavaScript ➢ Use HTML5 input types wherever applicable.

➢ To validate the data which users are submitting, use pattern attribute of HTML5 to validate rather than relying on JavaScript function. ➢ Following is a good example

```
<form action="" method="post">
  <label for="username">Create a Username: </label>
  <input type="text"
name="username"
id="username"      placeholder="4
<> 10"      pattern="[A-Za-
z]{4,10}"
    autofocus
    required>
  <button type="submit">Go </button>
```

```
</form>
```

Check for browser versions support before using this attribute

Reference: http://www.w3schools.com/tags/att_input_pattern.asp

7.3.14. Use of Async for JavaScript file whenever possible

- Consider using async attribute of script tag

  Reference: http://www.w3schools.com/tags/att_script_async.asp
  https://msdn.microsoft.com/en-us/library/windows/apps/hh700330.aspx

7.3.15. Use of contenteditable attribute

- Consider using contenteditable attribute if you want users to be able to edit the content of attribute. No need of using JavaScript/JQuery for the same Reference: http://www.w3schools.com/tags/att_global_contenteditable.asp

8. JavaScript

8.1. Naming conventions

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

8.1.1. 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
	}

8.1.2. 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"); } });

8.2. Syntax Formating

8.2.1. Syntax

Rule: Use all applicable rules from C# section.

8.3. Best Practices

8.3.1. 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;

8.3.2. Semi Colons

Rule: JavaScript allows any expression to be used as a statement and uses semicolons 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
	}

8.3.3.Avoid Global Variables

- Minimize the use of global variables.
- This includes all data types, objects, and functions.
- Global variables and functions can be overwritten by other scripts.

8.3.4.Avoid Global Lookups

- Global variables and functions are always more expensive to use than local ones because they involve a scope chain lookup

Example	

Legal	function updateUI(){ var
	doc = document;
	var imgs = doc.getElementsByTagName(“img”); for
	(var i=0, len=imgs.length; i < len; i++){ imgs[i].title
	= doc.title + ” image ” + i;
	}
	var msg = doc.getElementById(”msg”);
	msg.innerHTML = ”Update complete.”;
	}

Illegal	function updateUI(){
	var imgs = document.getElementsByTagName(“img”);
	for (var i=0, len=imgs.length; i < len; i++){ imgs[i].title
	= document.title + ” image ” + i;
	}
	var msg = document.getElementById(“msg”); msg.innerHTML
	= ”Update complete.”;
	}

This function may look perfectly fi ne, but it has three references to the global document object. If there are multiple images on the page, the document reference in the for loop could get executed dozens or hundreds of times, each time requiring a scope chain lookup. By creating a local variable that points to the document object, you can increase the performance of this function by limiting the number of global lookups to just one

8.3.5.Avoid the with Statement

- The with statement should be avoided where performance is important. Similar to functions, the with statement creates its own scope and therefore increases the length of the scope chain for code executed within it. Code executed within a with statement is guaranteed to run slower than code executing outside, because of the extra steps in the scope chain lookup

Example	

Legal	function updateBody(){ var body
	= document.body;
	alert(body.tagName);
	body.innerHTML = “Hello world!”;
	}

Illegal	function updateBody(){
	with(document.body){ alert(tagName);
	innerHTML = “Hello world!”;
	}
	}

8.3.6.Always Declare Local Variables

- All variables used in a function should be declared as local variables.
- Local variables must be declared with the var keyword, otherwise they will become global variables.

Example	
Legal	var variable1 = ‘Hello World’;

Illegal	variable1 = ‘Hello World’;

8.3.7.Use Namespaces for Global variables:

Always prefix identifiers in the global scope with a unique pseudo namespace related to the project or library. If you are working on "Project XYZ", a reasonable pseudo namespace would be XYZ.*.

```
var XYZ = {};

XYZ.sVariable1 = "";

XYZ.helloWorld = function() {
... };
```

8.3.8.Always put spaces around operators (= + - * /), and after commas:

Example	
Legal	var x = y + z; var values = ["Volvo", "Saab", "Fiat"];
Illegal	var x=y+z; var values = ["Volvo","Saab","Fiat"];

8.3.9.It is a good coding practice to initialize variables when you declare them.

This will:

- Give cleaner code
- Provide a single place to initialize variables
- Avoid undefined values

```
// Declare and initiate at the beginning
var firstName = "",    lastName = "",
price = 0,    discount = 0,    fullPrice =
0,    myArray = [],
   myObject = {};
```

8.3.10. Beware of Automatic Type Conversions

- Beware that numbers can accidentally be converted to strings or NaN (Not a Number).
- JavaScript is loosely typed. A variable can contain different data types, and a variable can change its data type:

```
var x = 5 + 7;      // x.valueOf() is 12,  typeof x is a number
var x = 5 + "7";    // x.valueOf() is 57,  typeof x is a string var
x = "5" + 7;    // x.valueOf() is 57,  typeof x is a string var x =
5 - 7;      // x.valueOf() is -2,  typeof x is a number var x = 5 -
"7";    // x.valueOf() is -2,  typeof x is a number var x = "5" -
7;    // x.valueOf() is -2,  typeof x is a number var x = 5 - "x";
// x.valueOf() is NaN, typeof x is a number
```

8.3.11. Prefer single quotes over double quotes

- JS allows both single quotes and double quotes
- But for consistency single-quotes (') are preferred to double-quotes ("). This is helpful when creating strings that include HTML.
- var sMsg = 'This is some "HTML"';

8.3.12. Always use semicolons to terminate JS statements:

Example:

Example	

Legal	var sVariable1 = 'Hello World'; function
	doSomething() {
	  return 'something';
	}

Illegal	var sVariable1 = 'Hello World' function
	doSomething() {
	  return 'something'
	}

8.3.13. Optimize Loops

- Loops are one of the most common constructs in programming and, as such, are found frequently in JavaScript. Optimizing these loops is an important part of the performance optimization process
- Cache the length in the loops
- In some of the cases, it's more efficient to start the iterator at the maximum number and decrement each time through the loop
- Posttest loops, such as do-while, avoid the initial evaluation of the terminal condition and tend to run faster.

Example	
Legal	var oObj = document.getElementsByTagName('p'); for(var i=0, len= oObj.length; i<len; i++) {};

	var i=values.length-1;
	if (i > -1){ do {
	process(values[i]);
	}while(--i >= 0);
	}

Illegal	var oObj = document.getElementsByTagName('p'); for(var i=0; i<oObj.length; i++) {};
	
	for (var i=0; i < values.length; i++){ process(values[i]);
	}

8.3.14. Declare variables outside of for statement.

- When executing lengthy "for" statements, don't make the engine work any harder than it must.

Example	

Legal	var oContainer = document.getElementById('container'); for(var i = 0, len = someArray.length; i < len; i++) {
	oContainer.innerHtml += 'my number: ' + i;
	console.log(i);}

Illegal	for(var i = 0; i < someArray.length; i++) {
	var container = document.getElementById('container');
	container.innerHtml += 'my number: ' + i;
	console.log(i);
	}

8.3.15. Use {} Instead of new Object()

Example:

Example	

Legal	var oObj = {
	name: 'Jeffrey',
	lastName = 'Way',
	someFunction : function() {
	console.log(this.name);
	}};

Illegal	var oObj = new Object();

	oObj.name = 'Jeffrey'; oObj.lastName = 'Way';
	oObj.someFunction = function() {
	console.log(this.name);
	}

8.3.16. Use [] Instead of new Array()

Example:

Example

Legal	var oArr = ['Joe','Plumber'];
Illegal	var a = new Array(); a[0] = "Joe"; a[1] = 'Plumber';

8.3.17. Always use strict comparison (=== Instead of ==).

- JavaScript utilizes two different kinds of equality operators: === | !== and == | != It is considered best practice to always use the former set when comparing. If two operands are of the same type and value, then === produces true and !== produces false. However, when working with == and !=, you'll run into issues when working with different types. In these cases, they'll try to coerce the values, unsuccessfully.

8.3.18. Avoid Magic Strings And Magic Numbers: Use symbolic constants for numeric literals and string literals.

Example	

Legal	var iDeckSize = 52;
	var j = 0;
	for (var i=0, len=iDeckSize, i<len; i++){ j =
	i + getRandomInt(iDeckSize + 1 - i) - 1;
	swapDeck(i, j);
	}

Illegal	var j = 0;
	for(var i=0, len=52, i<len; i++){ j
	= i + getRandomInt(53 - i) - 1;
	swapDeck(i, j);
	}

8.3.19. Using try/catch (Note: Exceptions Are For Exceptional Cases)

- Try/catch are expensive

- Do not use nested try/catch, instead use try/catch at the topmost level
- Do not ignore exceptions

Example	

Legal	try { doStuff();
	} catch(ignore) {
	log(ignore);
	}

Illegal	try { doStuff();
	} catch(ignore) {
	// Do nothing, just ignore.
	}

8.3.20. Do not use try/catch within loops

Example	

Legal	try {
	while(condition) {
	stuff();
	}
	} catch(e) {
	log(e);
	}

Illegal	while(condition) {
	try { stuff(); }
	catch(e) {
	log(e);
	}
	}

8.3.21. Use curly braces it even if it is not necessary

Example	

Legal	var bCheck = true; if(bCheck)
	{
	alert(bCheck);
	}
	else { alert(bCheck);

	}

Illegal	var bCheck = true;
	if(bCheck)
	alert(bCheck); else
	alert(bCheck);

8.3.22. Minimize DOM access: Accessing DOM elements with JavaScript is slow so in order to have a more responsive page, you should:

Cache references to accessed elements,

Example	

Legal	var oMsg = document.getElementById('someDiv') //store(cache) the object in a variable if (oMsg)
	{
	msg.style.display = 'none'
	}

Illegal	var sMsg = document.getElementById('someDiv') //store(cache) the object in a variable
	if (document.getElementById('someDiv'))
	{
	msg.style.display = 'none'
	}

8.3.23. Perform null checks for the object before accessing or updating any of its properties. Refer below the sample for same,

Example	

Legal	var oMsg = document.getElementById('someWrongDivName') if (oMsg)
	{
	oMsg.style.display = 'none'
	}
	else

	{
	alert('No error shown')
	}

Illegal	var msg = document.getElementById('someWrongDivName') msg.style.display = 'none' //msg is null here

8.3.24. Multiline string literals:

Example

Legal	var sMyString = 'We should use concatenation' + 'operator like this to wrap a long text' + 'to be shown to the user';
Illegal	var myString = 'We should use concatenation \ operator like this to wrap a long text \ to be shown to the user';

8.3.25. Use Array Joins Instead Of String Concatenation

Example	
Legal	var result = ['a','b','c','d'].join('');
Illegal	var result = 'a' + 'b' + 'c' + 'd';

8.3.26. Always put an operator on the preceding line for statement that can't fit in one line. ➢
For e.g. for Ternary operator,

```
var x = a ? b : c;  // All on one line if it will fit.

var y = a ?
longButSimpleOperandB : longButSimpleOperandC;

// Indenting to the line position of the first operand is also OK.
var z = a ?  moreComplicatedB
          :
          moreComplicatedC;
```

8.3.27. True and False Boolean Expressions:

➢ The following are all false in Boolean expressions,

i. null
ii. undefined

iii. '' the empty string iv. 0 the number

This means you can do this,

Example	
Legal	var y = 'Hi'; if (y) { }

Illegal	var y = 'Hi';
	if (y != null && y != '') {
	}

- But be careful, because these are all true,
  i. '0' the string
  ii. [] the empty array
  iii. {} the empty object

8.3.28. && and ||

These binary boolean operators are short-circuited, and evaluate to the last evaluated term. "||" has been called the 'default' operator

Example	
Legal	var win = opt_win || window;

Illegal	var win; if
	(opt_win) {
	win = opt_win;
	} else {
	win = window;
	}

Example	

Legal	if (node && node.kids && node.kids[index]) { alert(node.kids[index]);
	}
	
	OR

	var kid = node && node.kids && node.kids[index]; if
	(kid) {
	alert(kid);
	}

Illegal	if (node) { if
	(node.kids) { if
	(node.kids[index]) {
	alert(node.kids[index]);
	}
	}
	}

8.3.29. Group Related Statements Together Using Parentheses

Example	
Legal	return (obj !== undefined) && (obj !== null) && (klass === type);

Illegal	return obj !== undefined && obj !== null && klass === type;

8.3.30. Default Fallbacks

- All switch-case's should have a default: exit point. That last fallback should at least have a log statement.
- All if-else chains should have an else in the end. That last else should at least have a log

statement.

```
if (answer == 'no') {
        alert('You said no');
} else if (answer == 'ves') {
        alert('You said yes');
} else {
// This block should be here, even if we do not care about any outcome other than 'yes' or 'no
        alert('I should not be here'); //Log it
}
```

- A single if statement may not be regarded as an if-else "chain", so it's okay to leave single if's without an else.

8.3.31. Commenting a code:

- Use only line comments ("//"). Do not use group comments (/*…*/) in the functions, use it for adding comments only.
- Put your inline comments on top of the part that the comment is explaining:

```
// Cache the global function.  var
fnDo = doStuff;
```

Use full sentences for your comments

```
// If the request is sync, then process the response immediately.  processCallbacks(xhr, callbacks);
```

8.3.32. JavaScript Common Mistakes

- Accidentally Using the Assignment Operator

i. JavaScript programs may generate unexpected results if a programmer accidentally uses an assignment operator (=), instead of a comparison operator (==) in an if statement.

ii. This if statement returns true (maybe not as expected), because 10 is true:

```
var x = 0;
if (x = 10)
```

- Misunderstanding Floats
  i. All numbers in JavaScript are stored as 64-bits Floating point numbers (Floats).
  ii. All programming languages, including JavaScript, have difficulties with precise floating point values: var x = 0.1; var y = 0.2;

```
var z = x + y          // the result in z will not be 0.3
if (z == 0.3)          // this if test will fail
```

- Misplacing Semicolon: Because of a misplaced semicolon, this code block will execute regardless of the value of x:

```
if (x == 19);
{
  // code block
}
```

- Accessing Arrays with Named Indexes
  i. Many programming languages support arrays with named indexes. ii. Arrays with named indexes are called associative arrays (or hashes). iii. JavaScript does not support arrays with named indexes. iv. In JavaScript, arrays use numbered indexes:

Example	
Legal	var person = [];

	person[0] = "John";
	person[1] = "Doe"; person[2]
	= 46;
	var x = person.length; // person.length will return 3 var
	y = person[0];

Illegal	var person = [];
	person["firstName"] = "John";
	person["lastName"] = "Doe"; person["age"]
	= 46;
	var x = person.length; // person.length will return 0 var
	y = person[0]; // person[0] will return undefined

- Ending an Array Definition with a Comma

Example	
Legal	points = [40, 100, 1, 5, 25, 10];
Illegal	points = [40, 100, 1, 5, 25, 10,];

- Undefined is Not Null
  i. With JavaScript, null is for objects, undefined is for variables, properties, and methods.
  ii. To be null, an object has to be defined, otherwise it will be undefined.
  iii. If you want to test if an object exists, this will throw an error if the object is undefined

Example	
Legal	if (typeof myObj !== "undefined" && myObj !== null)
Illegal	if (myObj !== null && typeof myObj !== "undefined")

- Expecting Block Level Scope
  i. JavaScript does not create a new scope for each code block.
  ii. It is true in many programming languages, but not true in JavaScript. iii. It is a common mistake, among new JavaScript developers, to believe that this code returns undefined

```
for (var i = 0; i < 10; i++) {
  // some code
                              }

return i;
```

9. CSS

9.1. Naming Conventions

9.1.1. 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;
	}

9.2. Syntax Formatting

9.2.1. 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;
	 }

9.3. **Best Practices**

9.3.1. 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; }

9.3.2. 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"); }

9.3.3. 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.

9.3.4. External CSS file

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

9.3.5. Organize Code with Comments

- Keep styles organized in logical groups and provide a comment noting what the following styles pertain to.

Example

Legal	/* Primary header */ header { ... } /* Featured article */ article { ... } /* Buttons */ .btn { ... }
Illegal	header { ... } article { ... } .btn { ... }

9.3.6. Write CSS Using Multiple Lines & Spaces

- Makes the code easy to read as well as edit. When all of the code is piled into a single line without spaces, it's hard to scan and to make changes.

Example	

Legal	a,
	.btn {
	  background: #aaa;
	color: #f60; font-
	size: 18px;
	  padding: 6px;
	}

Illegal	a,.btn{background:#aaa;color:#f60;font-size:18px;padding:6px;}

9.3.7. Build Proficient Selectors

- CSS selectors can get out of control if they are not carefully maintained. They can easily become too long and too location specific.

Example

Legal	`.news a { ... }` `.news .special { ... }`
Illegal	`#aside #featured ul.news li a { ... }` `#aside #featured ul.news li a em.special { ... }`

9.3.8. Use Shorthand Properties & Values

- One feature of CSS is the ability to use shorthand properties and values. Most properties and values have acceptable shorthand alternatives.

Example	

Legal	`img {`
	`  margin: 5px 10px;`
	`}`
	`button`
	`{`
	`  padding-left: 20px;`
	`}`

Illegal	`img {`
	`  margin-top: 5px;  margin-right: 10px;  margin-bottom: 5px;  margin-left: 10px;`
	`}`
	`button`
	`{`
	`  padding: 0 0 0 20px;`
	`}`

9.3.9. Use Shorthand Hexadecimal Color Values

- When available, use the three-character shorthand hexadecimal color value, and always use lowercase characters within any hexadecimal color value.

Example	

Legal	`.module {`
	`background: #ddd;`
	`  color: #f60;`
	`}`

Illegal	.module {
	  background: #DDDDDD;
	  color: #FF6600;
	}

9.3.10. Drop Units from Zero Values

- One way to easily cut down on the amount of CSS we write is to remove the unit from any zero value. No matter which length unit is being used—pixels, percentages, em, and so forth—zero is always zero.

Example	
Legal	div { margin: 20px 0; letter- spacing: 0; padding: 0 5px; }
Illegal	div { margin: 20px 0px; letter- spacing: 0%; padding: 0px 5px;
	}

9.3.11. Modularize Styles for Reuse

- CSS is built to allow styles to be reused, specifically with the use of classes. For this reason, styles assigned to a class should be modular and available to share across elements as necessary.

Example	

Legal	.feat-box {
	background: #eee;
	border: 1px solid #ccc;
	  border-radius: 6px;
	}

Illegal	`.news { background: #eee; border: 1px solid #ccc; border-radius: 6px; } .events { background: #eee; border: 1px solid #ccc; border-radius: 6px; }`

10. JQuery

10.1. **Best Practices** 10.1.1.

Stay up to date!

- Always use the latest version of the library from http://jquery.com
- (Performance improvements and bug fixes are usually made between each version)

10.1.2. Know your selectors

- All selectors are not equally efficient ➢ Fastest to slowest selectors are:
- ID selectors
- Element selectors (form, input, etc.)
- Class selectors
- Pseudo and Attribute selectors (:visible, :hidden, [attribute=value], etc.)
- ID and element are fastest selectors as they are backed by native DOM operations

10.1.3. DOM insertion

- Every DOM insertion is costly
- You can minimize it by building HTML strings and using single append as late as possible
- Use detach() if doing heavy interaction with a node then re-insert it when done

10.1.4. Avoid Loops. Nested DOM Selectors can perform better

- Avoid unnecessary loops
- If possible, use the selector engine to address the elements that are needed

- There are places where loops can't be substituted but try your best to optimize

Example	
Legal	$('#menu a.submenu'). doSomething().doSomethingElse();
Illegal	if (myObj !== null && typeof myObj $('#menu a.submenu').each(function(){ $(this).doSomething().doSomethingElse(); });

10.1.5. Caching

- One of the most used features of jQuery is its ability to retrieve a DOM element by simply passing a reference to the $() function, be it an ID or class or whatever.
- Every time we use $() function jQuery searches for the DOM element, and if the element is found it creates an object representing a clone of that element, with added capabilities. This is true even for elements we already found using this function, so calling this function for the same elements more than once is redundant.

Example	
Legal	var $foo = $('#foo'); $foo.hide(); $foo.css('color', 'red'); $foo.show();
Illegal	$('#foo').hide(); $('#foo').css('color', 'red'); $('#foo').show(); Instead of calling the $() function multiple times for the same element, we should always keep a reference of the queried elements in a variable, and use it every time we need to access the same element.

10.1.6. Chaining

- In the previous example, we called for three consecutive times a method on the object $foo
- But instead of repeating each time the $foo variable, we should use the chaining functionality jQuery offers
- Most jQuery methods returns the same object on which we called the method
- Advantage - Less code, easier to write and it runs faster

Example	
Legal	$foo.hide().css('color', 'red').show();
Illegal	$('#foo').hide(); $('#foo').css('color', 'red'); $('#foo').show();

10.1.7. Passing object literals as parameters

- Passing an object literal as a parameter for jQuery methods avoid the need of calling the same method more than once.

Example	

Legal	// Passing object literals as parameters is good practice
	
	var $foo = $('#foo');
	$foo.css({
	'color': 'red',
	'width': '200px',
	'height': '200px'
	});

Illegal	// CSS method is being called three different times on the same object
	
	var $foo = $('#foo');
	$foo.css('color', 'red');
	$foo.css('width', '200px');
	$foo.css('height', '200px');

10.1.8. Use ID selector whenever possible

- Finding a DOM element by its ID is the fastest way, both in JavaScript and in jQuery. Whenever possible, you should always use the ID selector instead of using classes or tag names, or other ways.
- Also, when finding an element by its ID with the $() function, there's no need to specify a context, since it's slower than without.

Example	
Legal	// Faster $('#foo');
Illegal	// Slower $('#foo', this);

10.1.9. Don't mix CSS with jQuery

- Instead of changing CSS values with jQuery, you should use classes to change the appearance of elements. This way you can keep all your CSS in your CSS files, specify different appearances for different classes, and use jQuery to add or remove classes to apply your CSS.

Example

Legal	// add 'blue' class to each 'p' element
	$('p').addClass('blue');
	
	// if we want to change the text color to red, we remove the blue class and add the red one
	$('p').removeClass('blue').addClass('red');

Illegal	$('#foo').css('color', 'blue'); $('#foo').css('color', 'red');

10.1.10. Avoid multiple $(document).ready() calls

- Executing code only after the DOM has been fully created, or the page content being fully loaded, is a good way of executing scripts. But it's unnecessary to repeat $(document).ready() or $(window).load() for each script you want to run.

10.1.11. this vs. $(this)

- Use $(this) only when you need the jQuery methods otherwise use this when you only need a DOM element

Example	

Legal	$('button').click(function() {
	  alert('Button clicked: ' + this.id);
	});

Illegal	$('button').click(function() {
	  alert('Button clicked: ' + $(this).attr('id'));
	});

- Use this.checked instead of $(this).is(':checked')
- Use $.data(this, 'thing') instead of $(this).data('thing')

10.1.12. Use Sub-queries

- jQuery allows us to run additional selector operations on a wrapped set. This reduces performance overhead on subsequent selections since we already grabbed and stored the parent object in a local variable

```
<ul id="traffic_light">
<li><input type="radio" class="on" name="light" value="red" /> Red</li>
<li><input type="radio" class="off" name="light" value="yellow" /> Yellow</li>
<li><input type="radio" class="off" name="light" value="green" /> Green</li>
</ul>
```

For example, we can leverage sub-queries to grab the active and inactive lights and cache them for later manipulation.

```
var $traffic_light = $('#traffic_light'), $active_light
= $traffic_light.find('input.on'),
$inactive_lights = $traffic_light.find('input.off');
```

Tip: You can declare multiple local variables by separating them with commas – save those bytes!

10.1.13. Leverage Event Delegation (Bubbling)

- Every event (e.g. click, mouseover, etc.) in JavaScript "bubbles" up the DOM tree to parent elements. This is incredibly useful when we want many elements (nodes) to call the same function. Instead of binding an event listener function too many nodes — very inefficient — you can bind it once to their parent, and have it figure out which node triggered the event. For example, say we are developing a large form with many inputs, and want to toggle a class name when selected

Example	

Legal	$('#myList).bind('click', function(e){ var target = e.target, // e.target grabs the node that triggered the event.
	$target = $(target); // wraps the node in a jQuery object if
	(target.nodeName === 'LI') {
	$target.addClass('clicked');
	// do stuff
	}
	});

Illegal	$('#myList li).bind('click', function(){
	$(this).addClass('clicked');
	// do stuff

	});

10.1.14. Use jQuery.noConflict()

- Many JavaScript libraries use $ as a function or variable name, just as jQuery does. In jQuery's case, $ is just an alias for jQuery, so all functionality is available without using $. If we need to use another JavaScript library alongside jQuery, we can return control of $ back to the other library with a call to $.noConflict()

```
<script type="text/javascript" src="other_lib.js"></script>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
 $.noConflict();
 // Code that uses other library's $ can follow here. </script>
```

10.1.15. Adding and removing multiple classes from an element

- If we have an element with id “refiner” and we want to add two classes “addColor” and “addBackground”, we can do it by putting the two class names in addClass method separated by a space.

Example	
Legal	$(“#refiner”).addClass(“addColor addBackground”) $(“#refiner”).removeClass(“addColor addBackground”)

Illegal	$(“#refiner”).addClass(“addColor”) $(“#refiner”).addClass(“addBackground”) $(“#refiner”).removeClass(“addColor”) $(“#refiner”).removeClass(“addBackground”)