Coding Standards

User Team Coding Conventions and Standards

1 File Organization

1.1 C# Source Files

Each C# source file contains a single class, interface, or enum. It should also be named to reflect the class within. When private classes and interfaces are associated with a public class, you can put them in the same source file as the public class. Inner classes can be moved into separate files. The public class should be the first class or interface in the file.

Also delegates can be above a public class, but we should tend to avoid those.

There should be no beginning comment on the source file as all copyright information is contained in the assembly information.

1.2 Namespace and Using statements

The first lines of a C# file should contain the using statements of the file. The using statements should be minimally sufficient for the file (i.e. don't have unused using statements). The usings should also be alphabetically ordered with framework namespace on top. Furthermore, named using statements (aliases) should be formatted with proper spacing and on the bottom of the list. After that the namespace is defined. See below for an example.

```
using System.Windows.Forms;
using ININ.InteractionClient.Contracts;
using ININ.Windows.Forms;
using IceLibSession = ININ.IceLib.Core.Session;

namespace ININ.InteractionClient.Common
{
```

1.3 Class Declaration Order

- *Class declaration comment (optional)*
- Any delegates that are not inner delegates
- class or interface statement.
- Constants (including static readonly)
- Static Fields
- Instance Fields
- Static Constructor
- Instance Constructors
- inner delegates
- Events
- Auto-Properties
- Properties
- Methods

2 Naming Conventions

The following are the naming standards enforced by the Resharper code styles.

1. Use PascalCasing for types and namespaces.

   ```
   namespace ININ.InteractionClient.Alerting
   ```

2. Prefix interface names with I.

   ```
   interface IServiceLocator
   ```

3. Use PascalCasing for Methods, properties, and events.

   ```
   public void GetService() {}
   ```

4. Use camelCasing for local variable names and method arguments.

```
private void GetService(Type serviceType)
{
    string labelName;
    ...
}
```

5. Use ALL_UPPER for all const including local.

```
private const int SOME_NUMBER = 3;
```

6. Use ALL_UPPER for static readonly variables.

```
private static readonly int SOME_NUMBER = 3;
```

7. Use PascalCasing for enum members

```
public enum SomeEnum
{
    MemberOne,
    ...
}
```

8. Prefix member fields with _. Use camelCasing for the rest of a member variable (e.x. private int _number).

```
private int _someNumber;
```

9. Prefix static members with _. Use PascalCasing for the rest of the variable (e.x private static int _Number).

```
private static int _SomeNumber;
```

10. Suffix custom attribute classes with Attribute.
11. Suffix custom exception classes with Exception.
12. Suffix custom EventArgs with EventArgs.
13. Do **not** use Hungarian notaion.

```
WRONG:
bool bRet;
```

14. Avoid abbreviations (e.x. num instead of number).
15. Acronyms should only capitalize first letter.

```
public class HtmlControl {}

public class SipEngine {}
```

16. Use Pascal casing for generic types and prefix with T. Avoid using just T for the type name.

```
public class LinkedList<TArgs>
```

17. Do **not** use fully qualified type names. Use a using statement instead. If you need to use them, consider an alias instead.
18. Namespaces should be representative of the folder structure appended to the default namespace. For example, if you are in *thinclient\alerting\contracts* any classes in that folder should have the namespace ININ.InteractionClient.**Alerting.Contracts**

```
public class SomeClass<T>
{
 private const int MY_NUMBER = 3;
 private static readonly MY_STRING = Strings.Get("A_STRING");

 private int _currentNumber;

 public int CurrentNumber
 {
            get { return _currentNumber + 1; }
 }

 public int GetCurrentNumber()
 {
            return CurrentNumber + 1;
 }

 public event CurrentNumberChanged;
}
```

3 Spacing

3.1 Line Length

Avoid lines longer than 150 character because they begin to extend past the edge of a screen.

For XML doc comments, a rule of thumb of no lines longer than 60 characters.

3.2 Wrapping Lines

When an expression will not fit on a single line, break it according to these general principles:

- Break after a comma
- Break before an operator
- Align the new line as a normal four indent, the beginning of the parameter list, or 8 spaces

3.3 Blank Lines

There should be a blank line before:

- A single line comment.

There should be a blank line after:

- A ending curly brace
- Between every class member.
- Anytime that separating lines of code would increase readability.

There should **never** be blank lines after:

- Another blank line.
- Single line comments
- The get closing brace on a property.
- Between a summary tag and a method header.

> ⚠ In general member fields should not have new lines between them. The only occasion to do so is to group similar fields.

3.4 Blank Spaces

1. Should appear after commas in argument lists.
2. All binary operators should be separated by spaces.
3. Casts should **not** be followed by a blank space
4. Do **not** use a blank space when performing a typeof operation.

4 Comments

Inline comments should be used to describe hard-to-read or unusual code.

Doc Comments are **optional** for non-API classes. For Contracts, which is a public-facing API, comments are required for all members. Comments when added should adhere to the following rules:

4.1 General

- Every doc comment should be a complete sentence. This means start with a capital letter and end with a period, as well as proper english/grammer and spelled correctly.
- When using a type or method in comments, use <see cref="type"/>. This makes navigating a documentation

file easier. Also make sure it is properly referenced (i.e. make sure dmake spits no warnings back to you).
- When using null, true, or other language words use <see langword="langword"/>. This is a sandcastle addition.
- Do **not** put doc comments on private members or fields.

4.2 Constructors

The doc comment should say:

```
/// <summary>
/// Initializes a new instance of the <see cref="DocumentationSample"/> class.
/// </summary>
```

If there are overloads then adding a suffix to the sentence such as "with the provided name" would be necessary.

4.3 Methods

Pretty much anything is ok. Just don't do something like this:

```
/// <summary>
/// Poll server
/// </summary>
public bool ServerPoll()
```

Rather do this:

```
/// <summary>
/// Sends a request to the server to verify a current connection.
/// </summary>
public bool ServerPoll()
```

4.4 Properties

The start of the tag should indicate what get or set are available. It is easier to see in the intellsence if you can set the property than to add it and see if the compiler breaks.

```
/// <summary>
  /// Gets or sets a value indicating if the server is active
  /// </summary>
  public bool ServerActive { get; set; }
```

4.5 Events

To be consistent with Microsoft and IceLib, events should begin with "Occurs When"

```
/// <summary>
  /// Occurs when the <see cref="ServerActive"/> property changes.
  /// </summary>
  public event EventHandler ServerActiveChanged;
```

4.6 Parameters

Parameters should give you a good idea what the parameter is used for in the method or constructor. This is especially important for booleans that switch functionality.

```
/// <param name="name">The name of the server to poll.</param>
  /// <param name="defaultsIfFail">
  /// Used to determine if the poll should use the default server
  /// if the <paramref name="name"/> server connection fails.
  /// </param>
  public bool ServerPoll(string name, bool defaultsIfFail)
```

Also, note if you document one parameter you should document them all just for consistency. You don't want people to get confused about what parameters exist and what ones don't.

4.7 Return Statements

Use returns tag for methods and value tag for properties. Try to add these because having the English description is important, but they aren't as important as summaries. One thing to note for booleans we should do the format below. This helps clarify what return values mean what.

```
/// <returns><see langword="true"/> if the poll succeeds; otherwise, <see
langword="false"/>.</returns>
public bool ServerPoll()
```

4.8 Exceptions

Exceptions should not be documented for the following reasons:

1. Our unit tests should be the contracts for implementations. So if we have a contract that can be implemented by multiple classes, we should have a base test class that uses a virtual to get back the specific concrete implementation.
2. Because we are strongly favoring composition over inheritance, the surface area for exceptions grows. So, I have an interface *IFoo* with a method *Bar()*. That's implemented by class *Foo* which takes in its constructor objects of type *IOne*, *ITwo*, *IThree*, etc. etc. depending on how those are used in the context of the method *B*

ar() the various exceptions that could be thrown are much harder to know, not to mention keep up in the documentation for *IFoo.Bar*

4.9 Overloads

When you overload a method or constructor you should add an overload tag which has a generic summary of all the methods. This is useful for the documentation file. Also put a break between the overload and summary to help when looking at the code.

```
/// <overloads>
/// Sends a request to the server to verify a current connection.
/// </overloads>
///
/// <summary>
/// Sends a request to the server to verify a current connection.
/// </summary>
/// <returns><see langword="true"/> if the poll succeeds; otherwise, <see
langword="false"/>.</returns>
public bool ServerPoll()
```

4.10 Inline Comments

Inline comments should be on a separate line and never trailing a line of code. It should have an empty line above it and code directly below it. It should be indented to the same level as the code it is in and should have one space before it begins. inline comments should never break up a control structure and its braces.

```
if (!active)
{
    // This is strange
    i++

    // This is to compensate
    j--;
}
else
{
    // Normal case
    ...
}
```

- Do **not** use block comments. If the comment is multiple lines use single line comments for the entire thing.
- In general, Do **not** include TODO comments in release code. TODO's should be used for individuals during development. A todo in release code never gets looked at and pollutes the task list.
- Avoid comments that explain the obvious.
- Document only operation assumptions, algorithm insights and so on.

5 Declarations

5.1 One declaration per line

```
int level;
int size;

WRONG:
int level, size;
```

Also never use tabs to align variable names. It is too difficult to maintain:

```
WRONG:
int  level;
int  size;
object currentEntry;
```

5.2 Initialization

- Initialize local variables when they are declared, unless they are out parameters or if the initial value is never used in a code path (e.g. when the variable is assigned in both an if and the else).
- Member fields and static fields should be initialized in the constructor and static constructor respectively.

6 Statements

1. Each line should contain at most one statement.
2. Return statements should not use parentheses unless it makes the return value more obvious.
3. Control statements that that use the next line should **always** include the braces.
4. Keywords should have a space between it and the opening paren.

6.1 If

if statements should have the following form:

```
if (condition)
{
    statement;
}
else if (condition2)
{
    statement;
}
else
{
    statement;
}
```

If you are doing a validation check, you can keep the if statement on one. This is the only situation where braces can be excluded.

```
if (_connection.Up) return;
```

6.2 For

for statements should have the following form:

```
for (init; condition; update)
{
    statement;
}
```

6.3 While

while statements should have the following form:

```
while (condition)
{
    statement;
}
```

Never use do while loops.

6.4 Switch

switch statements should have the following form:

```
switch (condition)
{
    case ABC:
 statements;
 break;
    case DEF:
 statements;
 // Falls Through
    default:
        statements;
}
```

There should always be a default and a marker if the case falls through.

6.5 Try-catch

try-catch statements should have the following form:

```
try
{
    statements;
}
catch (Exception ex)
{
    statements;
}
finally
{
    statements;
}
```

The exception variable should always be named ex even when multiple catch statements are used.

6.6 Using

using statements should have the following form:

```
using (var d = new Digit())
{
   statements;
}
```

7 Resharper Settings

UserSettings.xml file

The latest version of UserSettings.xml is attached to this page. Download it to C:\Users\YOUR_USERNAME_HERE\AppData\Roaming\JetBrains\ReSharper\v5.1\vs10.0 to use it with ReSharper 5.1 and Visual Studio 2010.

Plugins

The following plugins should be installed.

1. Agent Johnson - http://code.google.com/p/agentjohnsonplugin/
2. Agent Smith - http://code.google.com/p/agentsmithplugin/
3. StyleCop - http://stylecop.codeplex.com/

C# Settings

Images of the C# settings to use.

Using ThinClientResharperSettings.xml

See Using ThinClientResharperSettings.xml

Inspection Severities

Use the following settings for the Inspections. However, everything except for hints and do not shows should be changed, unless in direct violation with the coding standards mentioned in the rest of the coding standards.

Agent Johnson

- Annotate type members with Value Analysis attributes and assert statements. **Show as warning -*** *not actually sure what this one does
- Replace "" with string.Empty. **Do not show**
- Return values should be asserted. **Show as warning -** not actually sure what this one does
- Undocumented thrown exception. **Show as hint**

Agent Smith

- Declarations must conform to naming conventions. **Do not show**
- Members must have XML comment. **Do not show**
- Word can be surrounded with meta tags. **Show as suggestion**
- Word found in C# declaration doesn't exist in dictionary. **Show as suggestion**
- Word found in C# literal doesn't exist in dictionary. **Do not show**
- Word found in C# verbatim string doesn't exist in dictionary.**Show as suggestion**
- Word found in ResX file doesn't exist in dictionary. **Show as warning**
- Word found in XML comment doesn't exist in dictionary. **Show as suggestion**

ASP.NET

- Attribute with optional value problem. **Show as warning**
- Path error. **Show as warning**
- Unknown html entity. **Show as warning**
- Unknown symbol. **Show as error**
- Wrong image size. **Show as warning**

Common Practices and Code Improvements

- Access to a static member of a type via a derived type. **Show as warning**
- Base member has 'params' parameter, but overrider hasn't. **Show as warning**
- Class can be made sealed. **Do not show**
- Class can be made sealed (private accessibility). **Do not show**
- Class can be made sealed (non-private accessibility). **Do not show**
- Convert local variable or field to constant. **Show as suggestion**
- Convert local variable or field to constant (private accessibility). **Show as suggestion**
- Convert local variable or field to constant (non-private accessibility). **Show as suggestion**
- Field can be made readonly. **Show as suggestion**
- Field can be made readonly (private accessibility). **Show as suggestion**
- Field can be made readonly (non-private accessibility). **Show as suggestion**
- Iteration variable can be declared with a more specific type. **Show as suggestion**
- Join local variable declaration and assignment. **Show as suggestion**
- Local variable has too wide declaration scope. **Show as suggestion**
- Make constructor in abstract class protected. **Show as suggestion**
- Member can be made private. **Show as suggestion**

- Member can be made private (private accessibility). **Show as suggestion**
- Member can be made private (non-private accessibility). **Show as suggestion**
- Member can be made protected. **Show as suggestion**
- Member can be made protected (private accessibility). **Show as suggestion**
- Member can be made protected (non-private accessibility).**Show as suggestion**
- Member can be made static. **Show as suggestion**
- Member can be made static (private accessibility). **Show as suggestion**
- Member can be made static (non-private accessibility). **Show as suggestion**
- Member or type can be made internal. **Do not show**
- Parameter can be declared with base type. **Show as suggestion**
- Parameter type can be IEnumerable<T>. **Show as suggestion**
- Parameter type can be IEnumerable<T> (private accessibility). **Show as suggestion**
- Parameter type can be IEnumerable<T> (non-private accessibility). **Show as suggestion**
- Redundant using directive. **Show as error**
- Return type can be IEnumerable<T>. **Show as suggestion**
- Return type can be IEnumerable<T> (private accessibility). **Show as suggestion**
- Return type can be IEnumerable<T> (non-private accessibility). **Show as suggestion**
- Simplify negative equality expression. **Show as suggestion**
- Type parameter could be declared as covariant or contravariant. **Show as suggestion**
- Use indexed property. **Show as suggestion**
- Use 'String.IsNullOrEmpty'. **Show as error**

Constraints Violations

- Base type is required. **Show as warning**
- BaseTypeRequired attributes supports only classes and interfaces. **Show as warning**
- Compare with '==' types marked by 'CannotApplyEqualityOperatorAttribute'. **Show as warning**
- Inconsistent Naming. **Show as warning**
- Namespace does not correspond to file location. **Show as warning**
- Possible 'null' assignment to entity marked with 'Value cannot be null" attribute. **Show as warning**
- Required base type conflicting another type. **Show as warning**
- Type specified in BaseTypeRequired attribure conflicts another type. **Show as warning**

Language Usage Opportunities

- '?:' expression can be re-written as '??' expression. **Show as warning**
- Convert anonymous method to method group. **Show as suggestion**
- Convert 'if' statement to 'switch' statement. **Show as hint**
- Convert 'Nullable<T>' to 'T?'. **Show as suggestion**
- Convert property to auto-property. **Show as warning**
- Convert property to auto-property with private setter. **Show as warning**
- Convert static method invocation to extension method call. **Show as suggestion**
- Convert to lambda expression. **Show as suggestion**
- Convert to static class. **Show as suggestion**
- For-loop can be converted into foreach-loop **Show as warning**
- 'if' statement can be re-written as '?:' expression. **Show as suggestion**
- 'if' statement can be re-written as '??' expression. **Show as suggestion**
- 'if-return' statement can be re-written as 'return' statement. **Show as suggestion**
- Introduce optional parameters. **Show as suggestion**
- Introduce optional parameters (private accessibility). **Show as suggestion**
- Introduce optional parameters (non-private accessibility). **Show as suggestion**
- Invert 'if' statement to reduce nesting. **Show as hint**
- Loop can be converted into LINQ-expression. **Show as hint**
- Method group is constructed of interface method. **Show as suggestion**

- Method group is constructed of structure method. **Show as suggestion**
- Part of loop's body can be converted into LINQ-expression. **Show as hint**
- Use object or collection initializer when possible. **Show as suggestion**
- Use 'var' keyword when initializer explicitly declares type. **Show as warning**
- Use 'var' keyword when possible. **Do not show**

Potential Code Quality Issues

- '?:' expression has identical true and false branches. **Show as warning**
- Abstract or virtual event is never invoked. **Show as suggestion**
- Access to modified closure. **Show as warning**
- Assignment to a property of a readonly field can be useless. **Show as warning**
- Auto-implemented property accessor is never used. **Show as warning**
- Auto-implemented property accessor is never used (private accessibility). **Show as warning**
- Auto-implemented property accessor is never used (non-private accessibility). **Show as warning**
- Bitwise operation on enum which is not marked by [Flags] attribute. **Show as warning**
- Class is never instantiated. **Show as suggestion**
- Class is never instantiated (private accessibility). **Show as suggestion**
- Class is never instantiated (non-private accessibility). **Show as suggestion**
- Compare with float.NaN or double.NaN. **Show as warning**
- Duplicate resource name. **Show as warning**
- Element is localizable. **Show as warning**
- Empty general catch clause. **Do not show**
- Event is never subscribed to. **Show as suggestion**
- Event is never subscribed to (private accessibility). **Show as suggestion**
- Event is never subscribed to (non-private accessibility). **Show as suggestion**
- Event unsubscription via anonymous delegate. **Show as warning**
- Exception rethrow possibly intended. **Show as warning**
- 'for' loop control variable is never modifed. **Show as warning**
- Format string placeholders mismatch. **Show as warning**
- Function never returns. **Show as warning**
- Invocation of polymorphic field-like event. **Show as warning**
- Local variable hides member. **Show as warning**
- Member hides static member from outer class. **Show as warning**
- Mismatch optional parameter value in overridden method. **Show as warning**
- 'Object.ReferenceEquals' is always false because it is called with value type. **Show as warning**
- Parameter hides member. **Show as warning**
- Parameter name differs in partial method declaration. **Show as warning**
- Possible ambiguity while accessing member by interface. **Show as warning**
- Possible compare of value type with 'null'. **Show as warning**
- Possible cyclic constructor call. **Show as warning**
- Possible loss of fraction. **Show as warning**
- Possible 'System.InvalidCastException'. **Show as warning**
- Possible 'System.InvalidOperationException'. **Show as warning**
- Possible 'System.NullReferenceException'. **Show as warning**
- Problems in format string. **Show as warning**
- Resource is not declared in base culture. **Show as warning**
- Resource is not overridden in specific culture. **Show as warning**
- Resource overrides base resource with empty value. **Show as warning**
- Resource value type is invalid. **Show as warning**
- Right operand of dynamic shift operation should be convertible to 'int'. **Show as warning**
- Similar anonymous type detected nearby. **Show as warning**
- Similar expressions comparison. **Show as warning**

- Static field initializer refers to static field below or in other part. **Show as warning**
- Suspicious type conversion or check. **Show as suggestion**
- Unaccessed field
- Unaccessed field (private accessibility). **Show as warning**
- Unaccessed field (non-private accessibility). **Show as suggestion**
- Unassigned field. **Show as suggestion**
- Unknown item group in build script**. Show as warning**
- Unknown property in build script. **Show as warning**
- Unknown target in build script. **Show as warning**
- Unknown task element in build script. **Show as warning**
- Unknown task in build script**. Show as warning**
- 'value' parameter is not used. **Show as warning**
- Virtual member call in constructor. **Show as hint**

Redundancies in Code

- '??' condition is known to be null or not null. **Show as warning**
- Anonymous method signature is not necessary. **Show as warning**
- Assignment is not used. **Show as warning**
- Double negation operator. **Show as warning**
- Explicit delegate creation expression is redundant. **Show as warning**
- Expression is always 'true' or always 'false'. **Show as warning**
- Heuristically unreachable code. **Do not show**
- Parentheses are redundant if attribute has no arguments. **Show as warning**
- Redundant anonymous type property explicit name. **Show as warning**
- Redundant argument name specification. **Show as warning**
- Redundant argument name with default value. **Show as suggestion**
- Redundant 'base.' qualifier. **Show as warning**
- Redundant boolean comparison. **Show as warning**
- Redundant braces in collection initializer. **Show as warning**
- Redundant 'case' label. **Show as hint**
- Redundant cast. **Show as warning**
- Redundant catch clause. **Show as warning**
- Redundant condition check before assignments. **Show as warning**
- Redundant 'else' keyword. **Show as warning**
- Redundant empty argument list on object creation expression. **Show as warning**
- Redundant empty finally block. **Show as warning**
- Redundant empty object or collection initializer. **Show as warning**
- Redundant explicit nullable type creation. **Show as warning**
- Redundant explicit size specification in array creation. **Show as warning**
- Redundant explicit type in array creation. **Show as warning**
- Redundant 'IEnumerable.Cast<T>' call **Show as warning**
- Redundant lambda parameter explicit type specification. **Show as warning**
- Redundant lambda signature parentheses. **Show as warning**
- Redundant name qualifier. **Show as warning**
- Redundant 'object.ToString()' call. **Show as warning**
- Redundant operand in logical conditional expression. **Show as warning**
- Redundant string type. **Show as suggestion**
- Redundant 'string.ToCharArray()' call. **Show as warning**
- Redundant 'this.' qualifier. **Show as warning**
- Redundant type arguments of method. **Show as warning**
- Redundant XAML namespace alias. **Show as warning**
- Redundant XAML resource. **Show as warning**

- Resource is overridden with identical value. **Show as suggestion**
- 'true' is redundant as 'for'-statement condition. **Show as warning**
- Unsafe context declaration is redundant. **Show as warning**

Redundancies in Symbol Declarations

- Class with virtual members never inherited. **Show as suggestion**
- Class with virtual members never inherited (private accessibility). **Show as suggestion**
- Class with virtual members never inherited (non-private accessibility). **Show as suggestion**
- Empty constructor. **Show as warning**
- Empty destructor. **Show as warning**
- Empty namespace declaration. **Show as warning**
- Method return value is never used
- Method return value is never used (private accessibility). **Show as warning**
- Method return value is never used (non-private accessibility). **Show as suggestion**
- 'params' modifier is always ignored on overrides. **Show as warning**
- Redundant base constructor call. **Show as warning**
- Redundant class or interface specification in base types list. **Show as warning**
- Redundant field initializer. **Show as warning**
- Redundant member override. **Show as warning**
- Redundant method overload. **Show as suggestion**
- Redundant method overload (private accessibility). **Show as suggestion**
- Redundant method overload (non-private accessibility). **Show as suggestion**
- Redundant 'partial' modifier on method declaration. **Show as warning**
- Redundant 'partial' modifier on type declaration. **Show as warning**
- Sealed member in sealed class. **Show as warning**
- Underlying type of enum is 'int'. **Show as warning**
- Unused parameter
- Unused parameter (private accessibility). **Show as warning**
- Unused parameter (non-private accessibility). **Show as suggestion**
- Unused type parameter. **Show as warning**
- Virtual member is never overriden. **Show as suggestion**
- Virtual member is never overriden (private accessibility). **Show as suggestion**
- Virtual member is never overriden (non-private accessibility). **Show as suggestion**

StyleCop - Defaults

- Default Violation Severity.**Show as warning**

StyleCop - Documentation Rules

- SA1600: Elements Must Be Documented. **Show as hint**
- SA1601: Partial Elements Must Be Documented. **Show as hint**
- SA1602: Enumeration Items Must Be Documented. **Do not show**
- SA1603: Documentation Must Contain Valid Xml. **Show as warning**
- SA1604: Element Documentation Must Have Summary. **Do not show**
- SA1605: Partial Element Documentation Must Have Summary. **Show as warning**
- SA1606: Element Documentation Must Have Summary Text. **Show as warning**
- SA1607: Partial Element Documentation Must Have Summary Text. **Show as warning**
- SA1608: Element Documentation Must Not Have Default Summary. **Show as warning**
- SA1609: Property Documentation Must Have Value. **Show as warning**
- SA1610: Property Documentation Must Have Value Text. **Show as warning**
- SA1611: Element Parameters Must Be Documented. **Do not show**
- SA1612: Element Parameter Documentation Must Match Element Parameters. **Show as warning**

- SA1613: Element Parameter Documentation Must Declare Parameter Name. **Show as warning**
- SA1614: Element Parameter Documentation Must Have Text. **Show as warning**
- SA1615: Element Return Value Must Be Documented. **Do not show**
- SA1616: Element Return Value Documentation Must Have Text. **Show as warning**
- SA1617: Void Return Value Must Not Be Documented. **Show as warning**
- SA1618: Generic Type Parameters Must Be Documented. **Do not show**
- SA1619: Generic Type Parameters Must Be Documented Partial Class. **Show as warning**
- SA1620: Generic Type Parameter Documentation Must Match Type Parameters. **Show as warning**
- SA1621: Generic Type Parameter Documentation Must Declare Parameter Name. **Show as warning**
- SA1622: Generic Type Parameter Documentation Must Have Text. **Do not show**
- SA1623: Property Summary Documentation Must Match Accessors. **Show as warning**
- SA1624: Property Summary Documentation Must Omit Set Accessor With Restricted Access. **Show as warning**
- SA1625: Element Documentation Must Not Be Copied And Pasted. **Show as warning**
- SA1626: Single Line Comments Must Not Use Documentation Style Slashes. **Show as warning**
- SA1627: Documentation Text Must Not Be Empty. **Show as warning**
- SA1628: Documentation Text Must Begin With A Capital Letter. **Show as warning**
- SA1629: Documentation Text Must End With A Period. **Show as warning**
- SA1630: Documentation Text Must Contain Whitespace. **Do not show**
- SA1631: Documentation Must Meet Character Percentage. **Show as warning**
- SA1632: Documentation Text Must Meet Minimum Character Length. **Show as hint**
- SA1633: File Must Have Header. **Do not show**
- SA1634: File Header Must Show Copyright. **Show as warning**
- SA1635: File Header Must Have Copyright Text. **Show as warning**
- SA1636: File Header Copyright Text Must Match. **Show as warning**
- SA1637: File Header Must Contain File Name. **Show as warning**
- SA1638: File Header File Name Documentation Must Match File Name. **Show as warning**
- SA1639: File Header Must Have Summary. **Show as warning**
- SA1640: File Header Must Have Valid Company Text. **Show as warning**
- SA1641: File Header Company Name Text Must Match. **Show as warning**
- SA1642: Constructor Summary Documentation Must Begin With Standard Text. **Show as warning**
- SA1643: Destructor Summary Documentation Must Begin With Standard Text. **Show as warning**
- SA1644: Documentation Headers Must Not Contain Blank Lines. **Show as warning**
- SA1645: Included Documentation File Does Not Exist. **Show as warning**
- SA1646: Included Documentation X Path Does Not Exist. **Show as warning**
- SA1647: Include Node Does Not Contain Valid File And Path. **Show as warning**
- SA1648: Inherit doc must be used with Inheriting Class. **Do not show**
- SA1649: File Header File Name Documentation Must Match Type Name. **Show as warning**

StyleCop - Layout Rules

- SA1500: Curly Brackets For Multi Line Statements Must Not Share Line. **Show as warning**
- SA1501: Statement Must Not Be On Single Line. **Show as warning**
- SA1502: Element Must Not Be On Single Line. **Do not show**
- SA1503: Curly Brackets Must Not Be Omitted. **Do not show**
- SA1504: All Accessors Must Be Multi Line Or Single Line. **Do not show**
- SA1505: Opening Curly Brackets Must Not Be Followed By Blank Line. **Show as warning**
- SA1506: Element Documentation Headers Must Not Be Followed By Blank Line. **Show as warning**
- SA1507: Code Must Not Contain Multiple Blank Lines In A Row. **Show as warning**
- SA1508: Closing Curly Brackets Must Not Be Preceded By Blank Line. **Show as warning**
- SA1509: Opening Curly Brackets Must Not Be Preceded By Blank Line. **Show as warning**
- SA1510: Chained Statement Blocks Must Not Be Preceded By Blank Line. **Show as warning**
- SA1511: While Do Footer Must Not Be Preceded By Blank Line. **Show as warning**

- SA1512: Single Line Comments Must Not Be Followed By Blank Line. **Show as warning**
- SA1513: Closing Curly Bracket Must Be Followed By Blank Line. **Show as suggestion**
- SA1514: Element Documentation Header Must Be Preceded By Blank Line. **Show as warning**
- SA1515: Single Line Comment Must Be Preceded By Blank Line. **Show as warning**
- SA1516: Elements Must Be Separated By Blank Line. **Show as hint**
- SA1517: Code Must not Contain Blank Lines At Start Of File. **Show as warning**
- SA1518: Code Must not Contain Blank Lines At End Of File. **Show as warning**

StyleCop - Maintainability Rules

- SA1119: Statement Must Not Use Unnecessary Parenthesis. **Show as warning**
- SA1400: Access Modifier Must Be Declared. **Show as warning**
- SA1401: Fields Must Be Private. **Show as warning**
- SA1402: File May Only Contain A Single Class. **Show as warning**
- SA1403: File May Only Contain A Single Namespace. **Show as warning**
- SA1404: Code Analysis Suppression Must Have Justification. **Show as warning**
- SA1405: Debug Assert Must Provide Message Text. **Show as warning**
- SA1406: Debug Fail Must Provide Message Text. **Show as warning**
- SA1407: Arithmetic Expressions Must Declare Precedence. **Show as warning**
- SA1408: Conditional Expressions Must Declare Precedence. **Show as warning**
- SA1409: Remove Unnecessary Code. **Show as warning**
- SA1410: Remove Delegate Parenthesis When Possible. **Show as warning**
- SA1411: Attribute Constructor Must Not Use Unnecessary Parenthesis. **Show as warning**

StyleCop - Naming Rules

- SA1300: Element Must Begin With Upper Case Letter. **Show as warning**
- SA1301: Element Must Begin With Lower Case Letter. **Show as warning**
- SA1302: Interface Names Must Begin With I. **Show as warning**
- SA1303: Const Field Names Must Begin With Upper Case Letter. **Do not show**
- SA1304: Non Private Readonly Fields Must Begin With Upper Case Letter. **Show as warning**
- SA1305: Field Names Must Not Use Hungarian Notation. **Show as warning**
- SA1306: Field Names Must Begin With Lower Case Letter. **Do not show**
- SA1307: Accessible Fields Must Begin With Upper Case Letter. **Show as warning**
- SA1308: Variable Names Must Not Be Prefixed. **Show as warning**
- SA1309: Field Names Must Not Begin With Underscore. **Do not show**
- SA1310: Field Names Must Not Contain Underscore. **Do not show**

StyleCop - Ordering Rules

- SA1200: Using Directives Must Be Placed Within Namespace. **Do not show**
- SA1201: Elements Must Appear In The Correct Order. **Do not show**
- SA1202: Elements Must Be Ordered By Access. **Do not show**
- SA1203: Constants Must Appear Before Fields. **Show as warning**
- SA1204: Static Elements Must Appear Before Instance Elements. **Do not show**
- SA1205: Partial Elements Must Declare Access. **Show as warning**
- SA1206: Declaration Keywords Must Follow Order. **Show as warning**
- SA1207: Protected Must Come Before Internal. **Show as warning**
- SA1208: System Using Directives Must Be Placed Before Other Using Directives. **Show as warning**
- SA1209: Using Alias Directives Must Be Placed After Other Using Directives. **Show as warning**
- SA1210: Using Directives Must Be Ordered Alphabetically By Namespace. **Show as warning**
- SA1211: Using Alias Directives Must Be Ordered Alphabetically By Alias Name. **Show as warning**
- SA1212: Property Accessors Must Follow Order. **Show as warning**
- SA1213: Event Accessors Must Follow Order. **Show as warning**

- SA1214: Static Readonly Elements Must Appear Before Static Non Readonly Elements. **Do Not Show**
- SA1215: Instance Readonly Elements Must Appear Before Instance Non Readonly Elements. **Do Not Show**

StyleCop - Readability Rules

- SA1100: Do Not Prefix Calls With Base Unless Local Implementation Exists. **Show as warning**
- SA1101: Prefix Local Calls With This **Do not show**
- SA1102: Query Clause Must Follow Previous Clause. **Show as warning**
- SA1103: Query Clauses Must Be On Separate Lines Or All On One Line. **Show as warning**
- SA1104: Query Clause Must Begin On New Line When Previous Clause Spans Multiple Lines. **Show as warning**
- SA1105: Query Clauses Spanning Multiple Lines Must Begin On Own Line. **Show as warning**
- SA1106: Code Must Not Contain Empty Statements. **Show as warning**
- SA1107: Code Must Not Contain Multiple Statements On One Line. **Show as warning**
- SA1108: Block Statements Must Not Contain Embedded Comments. **Show as warning**
- SA1109: Block Statements Must Not Contain Embedded Regions. **Show as warning**
- SA1110: Opening Parenthesis Must Be On Declaration Line. **Show as warning**
- SA1111: Closing Parenthesis Must Be On Line Of Last Parameter. **Show as warning**
- SA1112: Closing Parenthesis Must Be On Line Of Opening Parenthesis. **Show as warning**
- SA1113: Comma Must Be On Same Line As Previous Parameter. **Show as warning**
- SA1114: Parameter List Must Follow Declaration. **Show as warning**
- SA1115: Parameter Must Follow Comma **Do not show**
- SA1116: Split Parameters Must Start On Line After Declaration **Do not show**
- SA1117: Parameters Must Be On Same Line Or Separate Lines. **Do not show**
- SA1118: Parameter Must Not Span Multiple Lines. **Show as suggestion**
- SA1120: Comments Must Contain Text. **Show as warning**
- SA1121: Use Built In Type Alias. **Do not show**
- SA1122: Use String Empty For Empty Strings **Do not show**
- SA1123: Do Not Place Regions Within Elements. **Show as warning**
- SA1124: Do Not Use Regions. **Show as warning**
- SA1125: Use Shorthand For Nullable Types. **Show as warning**
- SA1126: Prefix Calls Correctly. **Show as warning**

StyleCop - Spacing Rules

- SA1000: Keywords Must be Spaced Correctly. **Show as warning**
- SA1001: Commas Must be Spaced Correctly. **Show as warning**
- SA1002: Semicolons Must be Spaced Correctly. **Show as warning**
- SA1003: Symbols Must be Spaced Correctly. **Show as warning**
- SA1004: Documentation Lines Must begin With Single Space. **Show as warning**
- SA1005: Single Line Comments Must Begin With Single Space. **Show as warning**
- SA1006: Preprocessor Keywords Must Not Be Preceded By Space. **Show as warning**
- SA1007: Operator Keyword Must be Followed By Space. **Show as warning**
- SA1008: Opening Parenthesis Must Be Spaced Correctly **Show as warning**
- SA1009: Closing Parenthesis Must Be Spaced Correctly **Show as warning**
- SA1010: Opening Square Brackets Must Be Spaced Correctly **Show as warning**
- SA1011: Closing Square Brackets Must Be Spaced Correctly **Show as warning**
- SA1012: Opening Curly Brackets Must Be Spaced Correctly **Do not show**
- SA1013: Closing Curly Brackets Must Be Spaced Correctly **Do not show**
- SA1014: Opening Generic Brackets Must Be Spaced Correctly **Show as warning**
- SA1015: Closing Generic Brackets Must Be Spaced Correctly **Show as warning**
- SA1016: Opening Attribute Brackets Must Be Spaced Correctly **Show as warning**
- SA1017: Closing Attribute Brackets Must Be Spaced Correctly **Show as warning**
- SA1018: Nullable Type Symbols Must Not Be Preceded By Space **Show as warning**

- SA1019: Member Access Symbols Must Be Spaced Correctly **Show as warning**
- SA1020: Increment Decrement Symbols Must Be Spaced Correctly **Show as warning**
- SA1021: Negative Signs Must Be Spaced Correctly **Show as warning**
- SA1022: Positive Signs Must Be Spaced Correctly **Show as warning**
- SA1023: Dereference And Access Of Symbols Must Be Spaced Correctly **Show as warning**
- SA1024: Colons Must Be Spaced Correctly **Show as warning**
- SA1025: Code Must Not Contain Multiple Whitespace In A Row **Show as warning**
- SA1026: Code Must Not Contain Space After New Keyword In Implicitly Typed Array Allocation **Show as warning**
- SA1027: Tabs Must Not Be Used **Show as warning**

Unused Symbols

- Type is never used.
- Type is never used (private accessibility).**Show as warning**
- Type is never used (non-private accessibility).**Show as suggestion**
- Type member is never accessed via base type.
- Type member is never accessed via base type (private accessibility).**Show as warning**
- Type member is never accessed via base type (non-private accessibility).**Show as suggestion**
- Type member is never used.
- Type member is never used (private accessibility).**Show as warning**
- Type member is never used (non-private accessibility).**Show as suggestion**
- Type member is only used in overrides.
- Type member is only used in overrides (private accessibility).**Show as warning**
- Type member is only used in overrides (non-private accessibility).**Show as suggestion**

XAML

- XAML language level error. **Show as error**

8 makefile conventions

Order

The make file should be laid out in the following order

1. Makefile header
2. Assembly Target, Basename, and destination directory
3. USE statements (**USE_ION_CORE_I3TRACE_DOTNET_TRACING**, **USE_NINJECT**, **USE_ICELIB_CORE**, etc).
4. **USER_CSC_FLAGS** - Always including */nowarn:1591*, unless it is an assembly that requires comments (eg. contracts).
5. **ASSEMBLY_INFO -** which should point to *Properties\AssemblyInfo.cs*
6. **CSFILES**
7. Resources **-** such as **RESXFILES, RESOURCE_NAMESPACE**, etc.
8. String resources **- STR_RESXFILES**
9. **ASSEMBLIES,** followed by the assembly references statementsF
10. **DEPLOY_REF_FILES**
11. Assembly.dmake boilerplate code

Style

The following should be done to make the makefiles be standardized.

1. If there is more than one file being added to a variable then declare the variable first followed by += for the multiple files.

```
CSFILES =
CSFILES += Bar.cs
CSFILES += Foo.cs
```

2. The += should be in alphabetic order, with folders coming before the root

```
CSFILES =
CSFILES += Interactions\Call.cs
CSFILES += Interactions\WorkItem.cs
CSFILES += Form.cs
CSFILES += Server.cs
```

3. There should only be one space between operators
4. There should be a single line between variables except in the first section.

Example

```
################################################################################
##
## Copyright 2003 Interactive Intelligence, Inc.
## All Rights Reserved
##
## $File:
//depot/team/EIC/main/consultTransfer/products/eic/src/ThinClient/Interactions/makef
ile $
## $Revision: #11 $
## $Change: 381802 $
## $DateTime: 2010/03/19 16:38:10 $
##
################################################################################

ASSEMBLY_TARGET = LIB
BASENAME = ININ.InteractionClient.Interactions
DEST_DIR = $(EIC_BIN_DIR)

USE_ION_CORE_I3TRACE_DOTNET_TRACING = 1
USE_ICELIB_CORE = 1
USE_ICELIB_INTERACTIONS = 1
USE_ICELIB_PROCESSAUTOMATION = 1
USE_MSHTML = 1
USE_NINJECT = 1

USER_CSC_FLAGS = /nowarn:1591

ASSEMBLYINFO = Properties\AssemblyInfo.cs

CSFILES =
CSFILES += Addin\InteractionsAddinUtilities.cs
CSFILES += Assistance\SupervisorResponseComboBox.cs
CSFILES += Assistance\SupervisorResponseItem.cs
CSFILES += Commands\AddToConferenceInteractionCommand.cs
CSFILES += Commands\CoachInteractionCommand.cs
CSFILES += Commands\ConferenceInteractionCommand.cs
CSFILES += Commands\DisconnectInteractionCommand.cs
CSFILES += Commands\HoldInteractionCommand.cs
CSFILES += Commands\InteractionCommandBase.cs
CSFILES += Commands\InteractionCommandHelpers.cs
CSFILES += Commands\TransferInteractionCommand.cs
CSFILES += Commands\TransferToVoicemailInteractionCommand.cs
CSFILES += Commands\VoicemailInteractionCommand.cs
CSFILES += Extensions\ClientViewScrubbers.cs
CSFILES += Extensions\QueueConfigurationExtensions.cs
CSFILES += MessageHandlers\InteractionAutoAnsweredHandler.cs
CSFILES += MessageHandlers\InteractionCommandSendingHandler.cs
CSFILES += MessageHandlers\InteractionCommandSentHandler.cs
CSFILES += Services\Interactions\History\CallHistoryWatcher.cs
CSFILES += Services\Interactions\Images\ResourceService.cs
CSFILES += Services\Interactions\MyInteractions\ConnectedCallProvider.cs
CSFILES += Services\Interactions\MyInteractions\SelectedInteractionProvider.cs
CSFILES += Services\Interactions\Queues\MonitoredQueueAlerting.cs
CSFILES += Services\Interactions\Queues\MonitoredQueues.cs
CSFILES += Services\Interactions\Queues\Queue.cs
CSFILES += Services\Interactions\Queues\QueueService.cs
CSFILES += Services\Interactions\ScreenPop\DdeScreenPopWatchService.cs
CSFILES += Services\Interactions\ScreenPop\ScreenPopService.cs
CSFILES += Services\Interactions\ChatService.cs
CSFILES += Services\Interactions\ConsultTransferService.cs
CSFILES += Services\Server\PhoneNumberDetailService.cs
CSFILES += AclPickerModel.cs
CSFILES += ActionThumbBarService.cs
CSFILES += AssistanceInteractionAction.cs
CSFILES += AssociateProcessAction.cs
CSFILES += AutoCompleteTextBox.cs
CSFILES += ConsultTransferForm.Designer.cs
CSFILES += ConsultTransferPresenter.cs
CSFILES += ContactResolutionChangedEventArgs.cs
CSFILES += ContactResolutionManager.cs
CSFILES += ContactResolutionRenderer.cs
CSFILES += ContactResolutionToolstrip.cs
CSFILES += ContactTypesDropDownControl.cs
CSFILES += ContactTypesRes.Designer.cs
CSFILES += CustomButtons.cs
CSFILES += DdeEventMonitor.cs
CSFILES += DialPad.cs
CSFILES += DialPadButton.cs
CSFILES += DialPadButton.Designer.cs
CSFILES += DialPadClientView.cs
CSFILES += DialPadControl.cs
CSFILES += DialPadDockPage.cs
CSFILES += DialPadService.cs
CSFILES += DisassociateProcessAction.cs
CSFILES += DisconnectInteractionAction.cs
CSFILES += DistributionQueueClientView.cs
CSFILES += DoubleBufferedFlowLayoutPanel.cs
CSFILES += EmailAddressAutoCompleteProvider.cs
CSFILES += EmailAddressResolverForm.cs
CSFILES += EmailChangedForm.cs
CSFILES += EmailDigitalSignatureForm.cs
CSFILES += EmailDigitalSignatureForm.Designer.cs
CSFILES += EmailEncryptionInfoForm.cs
CSFILES += EmailEncryptionInfoForm.Designer.cs
CSFILES += EmailForm.cs
CSFILES += EmailMRUManager.cs
CSFILES += EmergencyCallAlert.cs
CSFILES += EmergencyCallAlertForm.cs

.IF "$(DEBUG)" != ""
 CSFILES += WorkgroupDetailsDisplay.cs
 CSFILES += SetAttributeForm.cs
 CSFILES += SetAttributeForm.Designer.cs
.ENDIF

RESOURCE_NAMESPACE = ININ.InteractionClient.Interactions

RESXFILES =
RESXFILES += CallbackForm.resx
RESXFILES += CallSecurityAlertForm.resx
RESXFILES += ChatForm.resx

.IF "$(DEBUG)" != ""
 RESXFILES += SetAttributeForm.resx
.ENDIF

STR_RESXFILES =
STR_RESXFILES += Properties\ActionsResources.resx
STR_RESXFILES += Properties\Resources.resx

ASSEMBLIES =
ASSEMBLIES += ININ.Client.Common
ASSEMBLIES += ININ.Common
ASSEMBLIES += ININ.Common.UI.CustomControls.DialPad
ASSEMBLIES += ININ.InteractionClient.Addin
ASSEMBLIES += ININ.InteractionClient.Alerting
ASSEMBLIES += ININ.Windows.Forms

ASSEMBLY_REFERENCES += {$(ASSEMBLIES)}$(OUTPUT_SUFFIX).dll
ASSEMBLY_REFERENCES += WindowsBase.dll

DEPLOY_REF_FILES += {$(I3_BUILDDIRS)}\common\bin\{ININ.Common}$(OUTPUT_SUFFIX).dll
DEPLOY_REF_FILES +=
{$(I3_BUILDDIRS)}\common\bin\{ININ.Windows.Forms}$(OUTPUT_SUFFIX).dll

$(PWD:B) : A AD

A  : ; $(MAKE) assembly all
AD  : ; $(MAKE) assembly DEBUG=1 all
```

```
.IMPORT: MAIN_MAKE_FILE
.INCLUDE: $(MAIN_MAKE_FILE:d)assembly.dmake
```

9 Programming Practices

1. Use String.Empty over ""
2. Avoid any unnecessary parentheses

```
AVOID:
((a)), a = (3 + 2)

RIGHT:
(a), a = (3 + 2) / 2
```

3. Use ?? whenever possible.

```
AVOID:
var x = expr1;
if (x = null) x = expr2;

USE:
var x = expr1 ?? expr2;
```

4. Structure return values to match the intent of the program

```
if (booleanExpression)
{
return true;
}
else
{
return false;
}
should instead be written as
return booleanExpression;

Similarly,

if (condition) {
return x;
}
return y;
should be written as

return (condition) ? x : y;
```

5. Expressions before ? in ternary operator should be parenthesized.

```
int a = (b == 5) ? 2 : 1;
```

Non-expressions however, needn't be parenthesized.

```
int a = !v ? 2 : 1;
```

6. If the member variable can be marked as readonly, mark it as readonly.
7. Use delegate inference instead of explicit delegate instantiation.

```
delegate void SomeDelegate();
public void SomeMethod()
{...}
SomeDelegate someAction = SomeMethod;
```

8. Maintain strict indentation. Do **not** use tabs. Use four spaces for the space.
9. Always place an open curly brace on a new line unless on a one-line property method

```
public int Height
{
get { return _height; }
}
```

10. Use lambda expressions instead of anonymous methods.
11. Omit parentheses on parameters in a lambda expression
12. With the exception of zero, one, or -1 never hard-code a numeric value. Always declare a const instead.
13. **Never** use error codes as method return values.
14. When defining custom exceptions follow Microsoft's rules for exceptions: custom serialization, proper constuctors, etc.
15. Avoid providing explicit values for enums unless they are powers of 2 (flags).
16. Name flags with a plural and normal enums as singular.
17. Do **not** provide public or protected or internal member variables. Use properties instead.
18. Use Automatic properties when possible
19. Avoid using the new inheritance qualifier. Use override instead.
20. Avoid defining event-handling delegates. Use EventHandler<T> instead, which is in ININ.Common
21. Access a static property through the class version (String.Format(...) or Int32.Convert(...)). If it is on the class you are in, do not access preface it with the class name.
22. Implement Dispose properly...See Microsoft's FxCop rules.
23. Use generic collections rather than non-generic (i.e. no ArrayList)
24. Return generic forms of an object on public methods.
25. Populate all fields in AssemblyInfo.cs such as company name, description, and version for major releases. Our build process does not affect them
26. Join the local variable declaration and assignment.
27. Minimize the scope of local variables.

28. Reduce the accessibility of methods and classes as much as possible (private, internal, protected, public).
29. Do not make a member static unless it has to be.
30. Prefer using the 'var' keyword when the variable type is explicitly defined in the assignment.
    a. Exception to this is for LINQ queries. Always use 'var' when defining a LINQ query, as it's required in most cases since the only entity to know the variable type is the compiler.

10 Coding Techniques

10.1 Locks

When constructing a lock, you should create a private class that represents the lock. This is useful for determining what deadlocked in a memory dump

```
public class SomeClass
{
    private class SomeClassLock {}

    private static SomeClassLock lockForClass = new SomeClassLock();

    public ThreadedAccess()
    {
        lock (lockForClass){
            access;
        }
    }
}
```

10.2 Events

10.2.1 Raising Events

You should favor raising events through the EventsHelper class in ININ.Common.

Event's Structure

We do **not** follow Microsoft's style for events. Our approach is to do the following:

- Create a raise method which is private and has the name RaiseEventName and takes in the event args.
    - Consider using the common library's EventsHelper static class in lieu of a specific private method to raise the event.
- Subscriptions should be added to an event with using implicit delegates and named with an appropriate name that follows our method name guidelines. The method name **can** match the event name, if that aids clarity.

10.2.2 Full Example

```
public class EventSample
{
    private int _something;

    public event EventHandler SomethingChanged;

    public int Something
    {
        get { return _something; }
        set
        {
            _something = value;
            RaiseSomethingChanged(EventArgs.Empty);
        }
    }

    private void RaiseSomethingChanged(EventArgs e){
        EventsHelper.Raise(SomethingChanged, this, e);
    }
}

public class EventConsumer
{
    public EventConsumer(EventSample eventSample)
    {
        eventSample.SomethingChanged += SomethingChanged;
    }

    private void SomethingChanged(object sender, EventArgs e)
    {
        do something;
    }
}
```

10.3 String comparisons

Hard-coded or internally used strings should be compared using Ordinal string comparisons (OrdinalIgnoreCase for case-insensitive). For strings used in the UI, or other localized strings you can use the Case insensitive version.

```
return String.Equals(str1, str2, StringComparison.Ordinal);

return String.Equals(str1, str2, StringComparison.OrdinalIgnoreCase);
```

11 Contracts

Contracts is a publicly supported API for writing client add-ins. There are a few coding practices we follow to make sure the use of the contracts is consistent.

1. Comment every publicly accessible member, their parameters, and return values.

2. Only have interfaces or data transfer objects in contracts namespace/project. No actual implementation should exist.
3. Throw exceptions only in extreme cases where the user would need to know of a failure. In general this rarely happens. Instead favor a null object pattern and return default values instead of nothing.
4. List returns should never be null. Instead make sure an empty list is returned.
5. When it is necessary to eat exceptions thrown by other components, still throw them in debug mode.

```
catch (Exception ex)
    {
        TraceTopics.SomeTopic.Exception(ex, "Some exception was encountered");


#if DEBUG
        throw;
#endif
    }
```

6. Interfaces should be around 5 member and no more than 10. We want to keep our interfaces small and be SOLID.
7. Only implement things that are used in the client. Adding future uses will bloat our code unnecessary and force us to support API's which may not be necessary.