

WeServ

C# Coding Standards

WS-QMS-GL-027

Fujitsu Philippines Inc. Group of Companies	C# Coding Standard		
Document No.: WS-QMS-GL-027	Effective Date: September 2, 2008	Version: 3.00	Page 2 of 19

This document is classified as ☐ Public ☒ Private ☐ Confidential ☐ Secret

Objective

- To provide guides on enforcing best practices and avoiding pitfalls in coding using C#.
-

Section 1

NAMING CONVENTIONS AND STYLE

	Description
1	Use Pascal casing for type and method names and constants: <pre>public class SomeClass { const int DefaultSize = 100; public SomeMethod() {} }</pre>
2	Use camel casing for local variable names and method arguments. <pre>int number; void MyMethod(int someNumber) { }</pre>
3	Prefix interface names with I <pre>interface IMyInterface { ... }</pre>
4	Prefix private member variables with m_ . Use Pascal casing for the rest of a member variable name following the m_ . <pre>public class SomeClass { private int m_Number; }</pre>
5	Suffix custom attribute classes with Attribute .
6	Suffix custom exception classes with Exception
7	Name methods using verb -object pair, such as ShowDialog() .
8	Methods with return values should have a name describing the value returned, such as GetObjectState() .
9	Use descriptive variable names. a) Avoid single character variable names, such as i or t . Use index or temp instead. b) Avoid using Hungarian notation for public or protected members. c) Do not abbreviate words (such as num instead of number).
10	Always use C# predefined types rather than the aliases in the System namespace. For example: <pre>object NOT Object string NOT String int NOT Int32</pre>

Fujitsu Philippines Inc. Group of Companies	C# Coding Standard		
Document No.: WS-QMS-GL-027	Effective Date: September 2, 2008	Version: 3.00	Page 3 of 19

This document is classified as ☐ Public ☒ Private ☐ Confidential ☐ Secret

11	With generics, use capital letters for types. Reserve suffixing Type when dealing with the .NET type Type . <pre>//Correct: public class LinkedList<K,T> {...} //Avoid: public class LinkedList<KeyType,DataType> {...}</pre>
12	Use meaningful namespaces such as the product name or the company name.
13	Avoid fully qualified type names. Use the using statement instead.
14	Avoid putting a using statement inside a namespace.
15	Group all framework namespaces together and put custom or third -party namespaces underneath . <pre>using System; using System.Collections.Generic; using System.ComponentModel; using System.Data; using MyCompany; using MyControls;</pre>
16	Use delegate inference instead of explicit delegate instantiation. <pre>delegate void SomeDelegate(); public void SomeMethod() {...} SomeDelegate someDelegate = SomeMethod;</pre>
17	Maintain strict indentation. Do not use tabs or nonstandard indentation, such as one space. Recommended values are three or four spaces, and the value should be uniformed across.
18	Indent comment at the same level of indentation as the code you are documenting.
19	All comments should pass spell checking. Misspelled comments indicate sloppy development.
20	All member variables should be declared at the top, with one line separating them from the properties or methods. <pre>public class MyClass { int m_Number; string m_Name; public void SomeMethod1() {} public void SomeMethod2() {} }</pre>
21	Declare a local variable as close as possible to its first use.
22	A file name should reflect the class it contains.
23	When using partial types and allocating a part per file, name each file after the logical part that part plays. For example: <pre>//In MyClass.cs public partial class MyClass {...} //In MyClass.Designer.cs public partial class MyClass {...}</pre>

Fujitsu Philippines Inc. Group of Companies	C# Coding Standard		
Document No.: WS-QMS-GL-027	Effective Date: September 2, 2008	Version: 3.00	Page 4 of 19

This document is classified as ☐ Public ☒ Private ☐ Confidential ☐ Secret

24	Always place an open curly brace ({) in a new line.
25	<p>With anonymous methods, mimic the code layout of a regular method, aligned with the anonymous delegate declaration. (complies with placing an open curly brace in a new line):</p> <pre> delegate void SomeDelegate(string someString); //Correct: public void InvokeMethod() { SomeDelegate someDelegate = delegate(string name) { MessageBox.Show(name); }; someDelegate("Juval"); } //Avoid public void InvokeMethod() { SomeDelegate someDelegate = delegate(string name) {MessageBox.Show(name);}; someDelegate("Juval"); } </pre>
26	<p>Use empty parentheses on parameter-less anonymous methods. Omit the parentheses only if the anonymous method could have been used on any delegate:</p> <pre> delegate void SomeDelegate(); //Correct SomeDelegate someDelegate1 = delegate() { MessageBox.Show("Hello"); }; //Avoid SomeDelegate someDelegate1 = delegate { MessageBox.Show("Hello"); }; </pre>

Section 2

CODING PRACTICES

	Description
1	Avoid putting multiple classes in a single file.
2	A single file should contribute types to only a single namespace. Avoid having multiple namespaces in the same file.
3	Avoid files with more than 500 lines (excluding machine-generated code).
4	Avoid methods with more than 25 lines.
5	Avoid methods with more than 5 arguments. Use structures for passing multiple arguments.
6	Lines should not exceed 80 characters.
7	<p>Do not manually edit any machine-generated code.</p> <p>a) If modifying machine generated code, modify the format and style to match this coding standard .</p> <p>b) Use partial classes whenever possible to factor out the maintained portions.</p>
8	Avoid comments that explain the obvious. Code should be self-explanatory. Good code with readable variable and method names should not require comments.
9	Document only operational assumptions, algorithm insights and so on.

This material is intended for FPIGC use only. It must not be reproduced in whole or in part, in any form, or by any means without a formal agreement or the written consent of the Process Management Unit (PMU). Any hard copy or unprotected soft copy of this document shall be regarded as an uncontrolled copy.

Filename: ws-qms-gl-027 c# coding standard

Fujitsu Philippines Inc. Group of Companies	C# Coding Standard		
Document No.: WS-QMS-GL-027	Effective Date: September 2, 2008	Version: 3.00	Page 5 of 19

This document is classified as ☐ Public ☒ Private ☐ Confidential ☐ Secret

10	Avoid method-level documentation. a) Use extensive external documentation for API documentation. b) Use method-level comments only as tool tips for other developers.
11	With the exception of zero and one, never hard -code a numeric value; always declare a constant instead.
12	Use the const directive only on natural constants such as the number of days of the week.
13	Avoid using const on read-only variables. For that, use the readonly directive. <pre>public class MyClass { public const int DaysInWeek = 7; public readonly int Number; public MyClass(int someValue) { Number = someValue; } }</pre>
14	Assert every assumption. On average, every fifth line is an assertion. <pre>using System.Diagnostics; object GetObject() {...} object someObject = GetObject(); Debug.Assert(someObject != null);</pre>
15	Every line of code should be walked through in a “white box” testing manner.
16	Catch only exceptions for which you have explicit handling.
17	In a catch statement that throws an exception, always throw the original exception (or another exception constructed from the original exception) to maintain the stack location of the original error: <pre>catch(Exception exception) { MessageBox.Show(exception.Message); throw; //Same as throw exception; }</pre>
18	Avoid error code as method return values.
19	Avoid defining custom exception classes.
20	When defining custom exceptions: a) Derive the custom exception from Exception . b) Provide custom serialization.
21	Avoid multiple Main() methods in a single assembly.
22	Make only the most necessary types public, mark others as internal .
23	Avoid friend assemblies, as they increase inter-assembly coupling.
24	Avoid code that relies on an assembly running from a particular location.
25	Minimize code in application assemblies (EXE client assemblies). Use class libraries instead to contain business logic.

Fujitsu Philippines Inc. Group of Companies	C# Coding Standard		
Document No.: WS-QMS-GL-027	Effective Date: September 2, 2008	Version: 3.00	Page 6 of 19

This document is classified as ☐ Public ☒ Private ☐ Confidential ☐ Secret

26	<p>Avoid providing explicit values for enums.</p> <pre>//Correct public enum Color { Red, Green, Blue } //Avoid public enum Color { Red = 1, Green = 2, Blue = 3 }</pre>
27	<p>Avoid specifying a type for an enum.</p> <pre>//Avoid public enum Color : long { Red, Green, Blue }</pre>
28	Always use a curly brace scope in an if statement, even if it conditions a single statement.
29	Avoid using the trinary conditional operator.
30	<p>Avoid function calls in Boolean conditional statements. Assign into local variables and check on them.</p> <pre>bool IsEverythingOK() {...} //Avoid: if(IsEverythingOK()) {...} //Correct: bool ok = IsEverythingOK(); if(ok) {...}</pre>
31	Always use zero -based arrays.
32	<p>Always explicitly initialize an array of reference types using a for loop.</p> <pre>public class MyClass { } const int ArraySize = 100; MyClass[] array = new MyClass[ArraySize]; for(int index = 0; index < array.Length; index++) { array[index] = new MyClass(); }</pre>
33	Do not provide public or protected member variables. Use properties instead.
34	Avoid using the new inheritance qualifier. Use override instead.
35	Always mark public and protected methods as virtual in a non-sealed class.
36	Never use unsafe code, except when using interop.
37	<p>Avoid explicit casting. Use the as operator to defensively cast to a type.</p> <pre>Dog dog = new GermanShepherd(); GermanShepherd shepherd = dog as GermanShepherd; if(shepherd != null) {...}</pre>
38	Always check a delegate for null before invoking it.

Fujitsu Philippines Inc. Group of Companies	C# Coding Standard		
Document No.: WS-QMS-GL-027	Effective Date: September 2, 2008	Version: 3.00	Page 7 of 19

This document is classified as ☐ Public ☒ Private ☐ Confidential ☐ Secret

39	Do not provide public event member variables. Use event accessors instead. <pre> public class MyPublisher { MyDelegate m_SomeEvent; public event MyDelegate SomeEvent { add { m_SomeEvent += value; } remove { m_SomeEvent -= value; } } } </pre>
40	Avoid defining event -handling delegates. Use <code>EventHandler<T></code> or <code>GenericEventHandler</code> instead.
41	Avoid raising events explicitly. Use <code>EventsHelper</code> to publish events defensively.
42	Always use interfaces.
43	Classes and interfaces should have at least 2:1 ratio of methods to properties.
44	Avoid interfaces with one member.
45	Strive to have three to five members per interface.
46	Do not have more than 20 members per interface. Twelve is probably the practical limit.
47	Avoid events as interface members.
48	When using abstract classes, offer an interface as well.
49	Expose interfaces on class hierarchies.
50	Prefer using explicit interface implementation.
51	Never assume a type supports an interface. Defensively query for that interface. <pre> SomeType obj1; IMyInterface obj2; /* Some code to initialize obj1, then: */ obj2 = obj1 as IMyInterface; if(obj2 != null) { obj2.Method1(); } else { //Handle error in expected interface } </pre>
52	Never hardcode strings that will be presented to end users. Use resources instead.
53	Never hardcode strings that might change based on deployment such as connection strings.
54	Use <code>String.Empty</code> instead of <code>""</code> : <pre> //Avoid string name = ""; //Correct string name = String.Empty; </pre>
55	When building a long string, use <code>StringBuilder</code> , not <code>string</code> .

Fujitsu Philippines Inc. Group of Companies	C# Coding Standard		
Document No.: WS-QMS-GL-027	Effective Date: September 2, 2008	Version: 3.00	Page 8 of 19

This document is classified as ☐ Public ☒ Private ☐ Confidential ☐ Secret

56	Avoid providing methods on structures. a) Parameterized constructors are encouraged. b) Can overload operators.
57	Always provide a static constructor when providing static member variables.
58	Do not use late-binding invocation when early-binding is possible.
59	Use application logging and tracing.
60	Never use <code>goto</code> unless in a <code>switch</code> statement fall-through.
61	Always have a <code>default</code> case in a <code>switch</code> statement that asserts. <pre>int number = SomeMethod(); switch (number) { case 1: Trace.WriteLine("Case1:"); break; case 2: Trace.WriteLine("Case 2:"); break; default: Debug.Assert(false); break; }</pre>
62	Do not use the <code>this</code> reference unless invoking another constructor from within a constructor. <pre>//Example of proper use of 'this' public class MyClass { public MyClass(string message) {} public MyClass() : this("Hello") {} }</pre>
63	Do not use the <code>base</code> word to access base class members unless you wish to resolve a conflict with a subclasses member of the same name or when invoking a base class constructor. <pre>//Example of proper use of 'base' public class Dog { public Dog(string name) {} virtual public void Bark(int howLong) {} } public class GermanShepherd : Dog { public GermanShepherd(string name) : base(name) {} override public void Bark(int howLong) { base.Bark(howLong); } }</pre>
64	Do not use <code>GC.AddMemoryPressure()</code> .
65	Do not rely on <code>HandleCollector</code> .

Fujitsu Philippines Inc. Group of Companies	C# Coding Standard		
Document No.: WS-QMS-GL-027	Effective Date: September 2, 2008	Version: 3.00	Page 9 of 19

This document is classified as ☐ Public ☒ Private ☐ Confidential ☐ Secret

66	<p>Always run code unchecked by default (for the sake of performance), but explicitly in checked mode for overflow- or underflow-prone operations:</p> <pre>int CalcPower(int number, int power) { int result = 1; for(int count = 1; count <= power; count++) { checked { result *= number; } } return result; }</pre>
67	<p>Avoid explicit code exclusion of method calls (#if...#endif). Use conditional methods instead:</p> <pre>public class MyClass { [Conditional("MySpecialCondition")] public void MyMethod() {} }</pre>
68	<p>Avoid casting to and from System.Object in code that uses generics. Use constraints or the as operator instead:</p> <pre>class SomeClass { } //Avoid: class MyClass<T> { void SomeMethod(T t) { object temp = t; SomeClass obj = (SomeClass)temp; } } //Correct: class MyClass<T> where T : SomeClass { void SomeMethod(T t) { SomeClass obj = t; } }</pre>
69	<p>Do not define constraints in generic interfaces. Interface level-constraint can often be replaced by strong-typing.</p> <pre>public class Customer {...} //Avoid: public interface IList<T> where T : Customer {...} //Correct: public interface ICustomerList : IList<Customer> {...}</pre>
70	Do not define method-specific constraints in interfaces.
71	Do not define constraints in delegates.
72	If a class or a method offers both generic and non generic flavors, always prefer using the generics flavor.

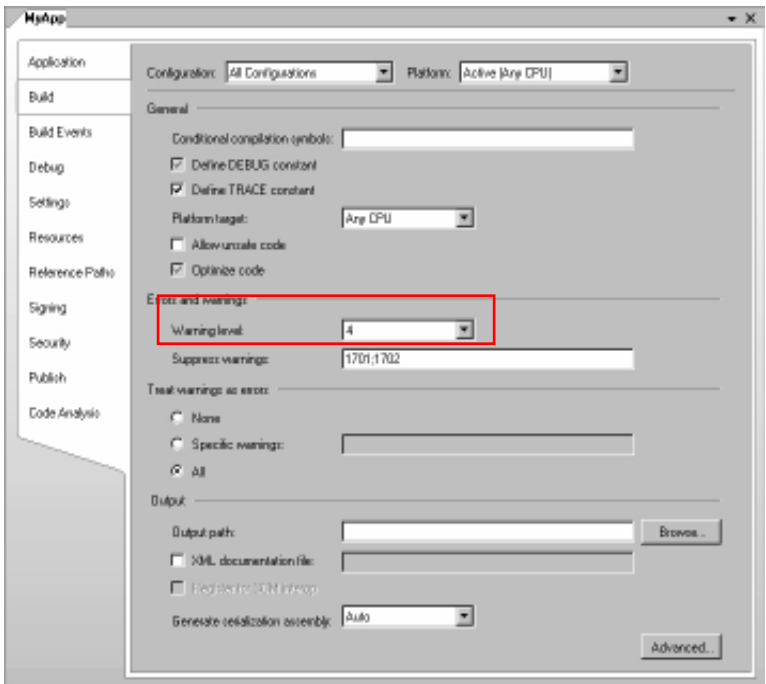
Fujitsu Philippines Inc. Group of Companies	C# Coding Standard		
Document No.: WS-QMS-GL-027	Effective Date: September 2, 2008	Version: 3.00	Page 10 of 19

This document is classified as ☐ Public ☒ Private ☐ Confidential ☐ Secret

73	<p>When implementing a generic interface that derived from an equivalent non-generic interface (such as <code>IEnumerable<T></code>), use explicit interface implementation on all</p> <pre> class MyCollection<T> : IEnumerable<T> { IEnumerator<T> IEnumerable<T>.GetEnumerator() {...} IEnumerator IEnumerable.GetEnumerator() { IEnumerable<T> enumerable = this; return enumerable.GetEnumerator(); } } </pre>	n
-----------	---	---

Section 3

PROJECT SETTINGS AND PROJECT STRUCTURE

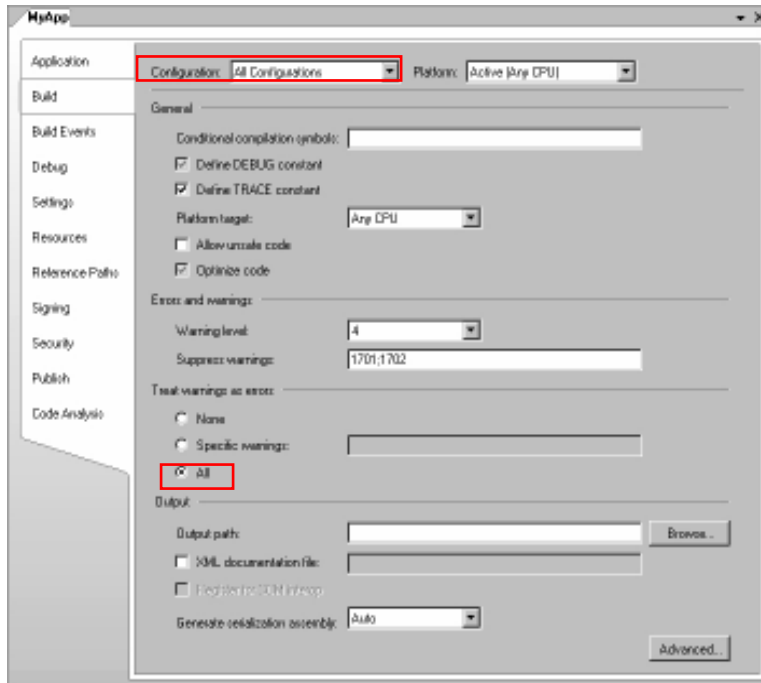
	Description
1	<p>Always build your project with warning level 4</p> 

Fujitsu Philippines Inc. Group of Companies	C# Coding Standard		
Document No.: WS-QMS-GL-027	Effective Date: September 2, 2008	Version: 3.00	Page 11 of 19

This document is classified as ☐ Public ☒ Private ☐ Confidential ☐ Secret

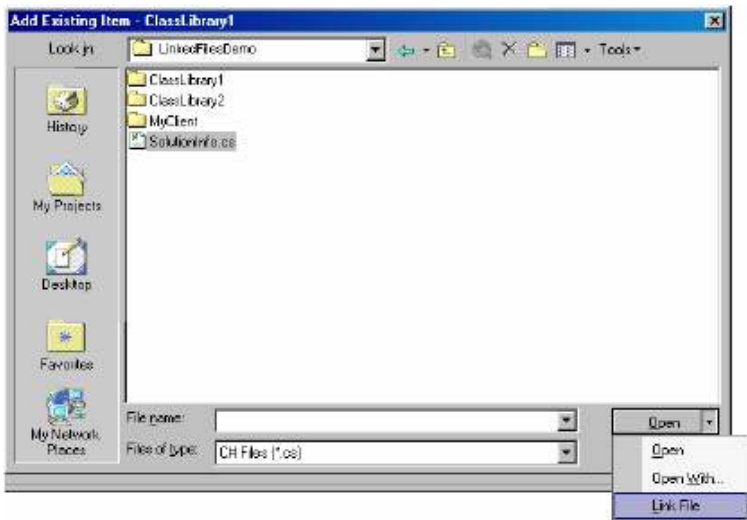
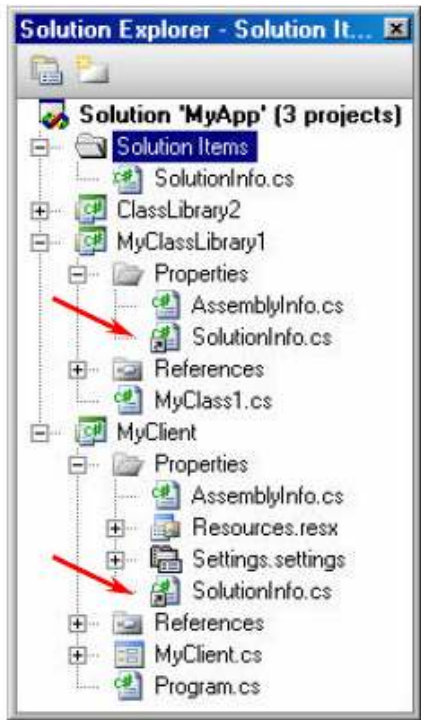
2

Treat warnings as errors in the Release build (note that this is not the default of Visual Studio). Although it is optional, this standard recommends treating warnings as errors in Debug builds as well.



Fujitsu Philippines Inc. Group of Companies	C# Coding Standard		
Document No.: WS-QMS-GL-027	Effective Date: September 2, 2008	Version: 3.00	Page 12 of 19

This document is classified as ☐ Public ☒ Private ☐ Confidential ☐ Secret

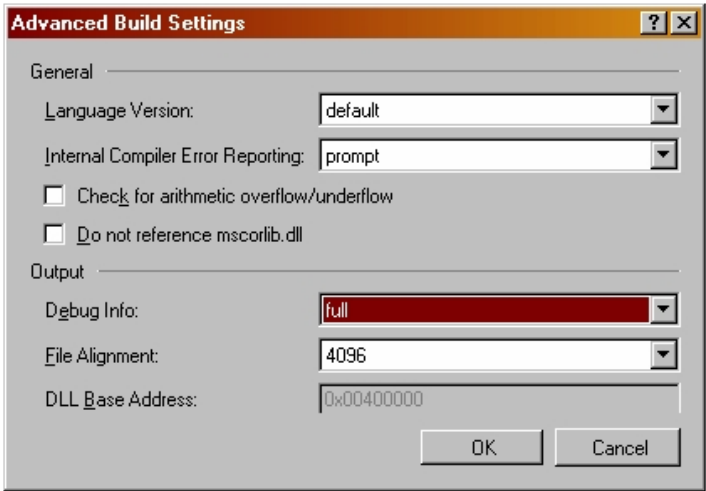

3	Avoid suppressing specific compiler warnings.
4	<p>Always explicitly state your supported runtime versions in the application configuration file.</p> <pre><?xml version="1.0"?> <configuration> <startup> <supportedRuntime version="v2.0.5500.0"/> <supportedRuntime version="v1.1.5000.0"/> </startup> </configuration></pre>
5	Avoid explicit custom version redirection and binding to CLR assemblies.
6	Avoid explicit preprocessor definitions (#define). Use the project settings for defining conditional compilation constants.
7	Do not put any logic inside AssemblyInfo.cs .
8	Do not put any assembly attributes in any file besides AssemblyInfo.cs .
9	Populate all fields in AssemblyInfo.cs such as company name, description, and copyright notice.
10	All assembly references in the same solution should use relative path.
11	Disallow cyclic references between assemblies.
12	Avoid multi-module assemblies.
13	Avoid tampering with exception handling using the Exception window (Debug Exceptions).
14	<p>Strive to use uniform version numbers on all assemblies and clients in the same logical application (typically a solution).</p> <div style="display: flex; justify-content: space-around;">   </div>
15	Link all solution-wide information to a global shared SolutionInfo.cs file.
16	Name your application configuration file as App.config , and include it in the project.

This material is intended for FPIGC use only. It must not be reproduced in whole or in part, in any form, or by any means without a formal agreement or the written consent of the Process Management Unit (PMU). Any hard copy or unprotected soft copy of this document shall be regarded as an uncontrolled copy.

Filename: ws-qms-gl-027 c# coding standard

Fujitsu Philippines Inc. Group of Companies	C# Coding Standard		
Document No.: WS-QMS-GL-027	Effective Date: September 2, 2008	Version: 3.00	Page 13 of 19

This document is classified as ☐ Public ☒ Private ☐ Confidential ☐ Secret

17	Modify Visual Studio 2005 default project structure to comply with your project standard layout, and apply uniform structure for project folders and files.
18	<p>A Release build should contain debug symbols.</p> 
19	Always sign your assemblies, including the client applications.
20	<p>Use password-protected keys.</p> 

Section 4

FRAMEWORK-SPECIFIC GUIDELINES

	Description
1.0	Data Access
1.1	Always use type-safe data sets or data tables. Avoid raw ADO.NET.
1.2	<p>Always use transactions when accessing a database.</p> <ul style="list-style-type: none"> a) Always use Enterprise Services or System.Transactions transactions. b) Do not use ADO.NET transactions by enlisting the database explicitly.

This material is intended for FPIGC use only. It must not be reproduced in whole or in part, in any form, or by any means without a formal agreement or the written consent of the Process Management Unit (PMU). Any hard copy or unprotected soft copy of this document shall be regarded as an uncontrolled copy.

Filename: ws-qms-gl-027 c# coding standard

Fujitsu Philippines Inc. Group of Companies	C# Coding Standard		
Document No.: WS-QMS-GL-027	Effective Date: September 2, 2008	Version: 3.00	Page 14 of 19

This document is classified as ☐ Public ☒ Private ☐ Confidential ☐ Secret

1.3	Always use transaction isolation level set to Serializable . Management decision is required to use anything else.
1.4	Do not use the Data Source window to drop connections on windows forms, ASP.NET forms or web services. Doing so couples the presentation tier to the data tier.
1.5	Avoid SQL Server authentication. Use Windows authentication instead.
1.6	Run components accessing SQL Server under separate identity from that of the calling client.
1.7	Always wrap your stored procedures in a high level, type safe class. Only that class invokes the stored rocedures. Let Visual Studio 2005 type-safe data adaptors automate as much of that as possible.
1.8	Avoid putting any logic inside a stored procedure. If you have anything more complex than simple switching logic to vary your query based on the parameter values, you should consider putting that logic in the business logic of the consuming code.
2.0	ASP.NET and Web Services
2.1	Avoid putting code in ASPX files of ASP.NET. All code should be in the code beside partial class.
2.2	Code in code beside partial class of ASP.NET should call other components rather than contain direct business logic.
2.3	Always check a session variable for null before accessing it.
2.4	In transactional pages or web services, always store session in SQL server.
2.5	Avoid setting the Auto-Postback property of server controls in ASP.NET to True.
2.6	Turn on Smart Navigation for ASP.NET pages.
2.7	Strive to provide interfaces for web services. See Appendix A of Programming .NET Components 2nd Edition.
2.8	Always provide namespace and service description for web services.
2.9	Always provide a description for web methods.
2.10	When adding a web service reference, provide meaningful name for the location.
2.11	<p>In both ASP.NET pages and web services, wrap a session variables in a local property. Only that property is allowed to access the session variable, and the rest of the code uses the property, not the session variable.</p> <pre> public class Calculator : WebService { int Memory { get { int memory = 0; object state = Session["Memory"]; if(state != null) { memory = (int)state; } return memory; } set { Session["Memory"] = value; } } [WebMethod(EnableSession=true)] public void MemoryReset() { Memory = 0; } } </pre>

Fujitsu Philippines Inc. Group of Companies	C# Coding Standard		
Document No.: WS-QMS-GL-027	Effective Date: September 2, 2008	Version: 3.00	Page 15 of 19

This document is classified as ☐ Public ☒ Private ☐ Confidential ☐ Secret

2.12	<p>Always modify client-side web service wrapper class to support cookies, since you have no way of knowing whether the service uses Session state or not.</p> <pre>public class Calculator : SoapHttpClientProtocol { public Calculator() { CookieContainer = new System.Net.CookieContainer(); Url = ...; } }</pre>
3.0	Multithreading
3.1	Use Synchronization Domains.
3.2	Never call outside your synchronization domain.
3.3	Manage asynchronous call completion on a callback method. Do not wait, poll, or block for completion.
3.4	<p>Always name your threads. The name is traced in the debugger Threads window, making debug sessions more productive.</p> <pre>Thread currentThread = Thread.CurrentThread; string threadName = "Main UI Thread"; currentThread.Name = threadName;</pre>
3.5	Do not call <code>Suspend()</code> or <code>Resume()</code> on a thread.
3.6	<p>Do not call <code>Thread.Sleep()</code>, except in the following conditions:</p> <ul style="list-style-type: none"> a) <code>Thread.Sleep(0)</code> is acceptable optimization technique to force a context switch. b) <code>Thread.Sleep()</code> is acceptable in testing or simulation code.
3.7	Do not call <code>Thread.SpinWait()</code> .
3.8	Do not call <code>Thread.Abort()</code> to terminate threads. Use a synchronization object instead to signal the thread to terminate.
3.9	<p>Avoid explicitly setting thread priority to control execution. You can set thread priority based on task semantic, such as below normal</p> <p>(<code>ThreadPriority.BelowNormal</code>) for a screen saver.</p>
3.10	Do not read the value of the <code>ThreadState</code> property. Use <code>Thread.IsAlive()</code> to determine whether the thread is dead or alive.
3.11	Do not rely on setting the thread type to background thread for application shutdown. Use a watchdog or other monitoring entity to deterministically kill threads.
3.12	Do not use thread local storage unless thread affinity is guaranteed.
3.13	Do not call <code>Thread.MemoryBarrier()</code> .
3.14	<p>Never call <code>Thread.Join()</code> without checking that you are not joining your own thread.</p> <pre>void WaitForThreadToDie(Thread thread) { Debug.Assert(Thread.CurrentThread.GetHashCode() != thread.GetHashCode()); thread.Join(); }</pre>
3.15	Always use the <code>lock()</code> statement rather than explicit <code>Monitor</code> manipulation.

Fujitsu Philippines Inc. Group of Companies	C# Coding Standard		
Document No.: WS-QMS-GL-027	Effective Date: September 2, 2008	Version: 3.00	Page 16 of 19

This document is classified as ☐ Public ☒ Private ☐ Confidential ☐ Secret

3.16	<p>Always encapsulate the <code>lock()</code> statement inside the object it protects.</p> <pre>public class MyClass { public void DoSomething() { lock(this) {...} } }</pre>
3.17	You can use synchronized methods instead of writing the <code>lock()</code> statement yourself.
3.18	Avoid fragmented locking.
3.19	Avoid using a <code>Monitor</code> to wait or pulse objects. Use manual or auto-reset events instead.
3.20	Do not use volatile variables. Lock your object or fields instead to guarantee deterministic and thread-safe access. Do not use <code>Thread.VolatileRead()</code> , <code>Thread.VolatileWrite()</code> , or the <code>volatile</code> modifier.
3.21	Avoid increasing the maximum number of threads in the thread pool.
3.22	<p>Never stack <code>lock</code> statements because that does not provide atomic locking. Use <code>WaitHandle.WaitAll()</code> instead.</p> <pre>MyClass obj1 = new MyClass(); MyClass obj2 = new MyClass(); MyClass obj3 = new MyClass(); //Do not stack lock statements lock(obj1) lock(obj2) lock(obj3) { obj1.DoSomething(); obj2.DoSomething(); obj3.DoSomething(); }</pre>
4.0	Serialization
4.1	Prefer the binary formatter.
4.2	Mark serialization event handling methods as private.
4.3	Use the generic <code>IGenericFormatter</code> interface.
4.4	Mark non-sealed classes as serializable.
4.5	When implementing <code>IDeserializationCallback</code> on a non-sealed class, make sure to do so in a way that allowed subclasses to call the base class implementation of <code>OnDeserialization()</code> .
4.6	Always mark un-serializable member variables as non serializable.
4.7	<p>Always mark delegates on a serialized class as non-serializable fields:</p> <pre>[Serializable] public class MyClass { [field:NonSerialized] public event EventHandler MyEvent; }</pre>
5.0	Remoting
5.1	Prefer administrative configuration to programmatic configuration.
5.2	Always implement <code>IDisposable</code> on single call objects.

This material is intended for FPIGC use only. It must not be reproduced in whole or in part, in any form, or by any means without a formal agreement or the written consent of the Process Management Unit (PMU). Any hard copy or unprotected soft copy of this document shall be regarded as an uncontrolled copy.

Filename: ws-qms-gl-027 c# coding standard

Fujitsu Philippines Inc. Group of Companies	C# Coding Standard		
Document No.: WS-QMS-GL-027	Effective Date: September 2, 2008	Version: 3.00	Page 17 of 19

This document is classified as ☐ Public ☒ Private ☐ Confidential ☐ Secret

5.3	Always prefer a TCP channel and a binary format when using remoting, unless a firewall is present.
5.4	<p>Always provide a null lease for a singleton object.</p> <pre>public class MySingleton : MarshalByRefObject { public override object InitializeLifetimeService() { return null; } }</pre>
5.5	Always provide a sponsor for a client activated object. The sponsor should return the initial lease time.
5.6	Always unregister the sponsor on client application shutdown.
5.7	Always put remote objects in class libraries.
5.8	Avoid using SoapSuds.
5.9	Avoid hosting in IIS.
5.10	Avoid using uni-directional channels.
5.11	<p>Always load a remoting configuration file in Main() even if the file is empty, and the application does not use remoting.</p> <pre>static void Main() { RemotingConfiguration.Configure("MyApp.exe.config"); /* Rest of Main() */ }</pre>
5.12	Avoid using Activator.GetObject() and Activator.CreateInstance() for remote objects activation. Use new instead.
5.13	Always register port 0 on the client side, to allow callbacks.
5.14	Always elevate type filtering to full on both client and host to allow callbacks.
6.0	Security
6.1	<p>Always demand your own strong name on assemblies and components that are private to the application, but are public (so that only you can use them).</p> <pre>public class PublicKeys { public const string MyCompany = "1234567894800000940000000602000000240000"+ "52534131000400000100010007D1FA57C4AED9F0"+ "A32E84AA0FAEFD0DE9E8FD6AEC8F87FB03766C83"+ "4C99921EB23BE79AD9D5DCC1DD9AD23613210290"+ "0B723CF980957FC4E177108FC607774F29E8320E"+ "92EA05ECE4E821C0A5EFE8F1645C4C0C93C1AB99"+ "285D622CAA652C1DFAD63D745D6F2DE5F17E5EAF"+ "0FC4963D261C8A12436518206DC093344D5AD293"; } [StrongNameIdentityPermission(SecurityAction.LinkDemand, PublicKey = PublicKeys.MyCompany)] public class MyClass {...}</pre>
6.2	Apply encryption and security protection on application configuration files.

Fujitsu Philippines Inc. Group of Companies	C# Coding Standard		
Document No.: WS-QMS-GL-027	Effective Date: September 2, 2008	Version: 3.00	Page 18 of 19

This document is classified as ☐ Public ☒ Private ☐ Confidential ☐ Secret

6.3	<p>When importing an interop method, assert unmanaged code permission, and demand appropriate permission instead.</p> <pre>[DllImport("user32", EntryPoint="MessageBoxA")] private static extern int Show(IntPtr handle, string text, string caption, int msgType); [SecurityPermission(SecurityAction.Assert, UnmanagedCode = true)] [UIPermission(SecurityAction.Demand, Window = UIPermissionWindow.SafeTopLevelWindows)] public static void Show(string text, string caption) { Show(IntPtr.Zero, text, caption, 0); }</pre>
6.4	Do not suppress unmanaged code access via the <code>SuppressUnmanagedCodeSecurity</code> attribute.
6.5	Do not use the <code>/unsafe</code> switch of <code>TlbImp.exe</code> . Wrap the RCW in managed code so that you could assert and demand permissions declaratively on the wrapper.
6.6	On server machines, deploy a code access security policy that grants only Microsoft, ECMA, and self (identified by a strong name) full trust. Code originating from anywhere else is implicitly granted nothing.
6.7	On client machines, deploy a security policy which grants client application only the permissions to execute, to call back the server and to potentially display user interface. When not using ClickOnce, client application should be identified by a strong name in the code groups.
6.8	<p>To counter a luring attack, always refuse at the assembly level all permissions not required to perform the task at hand.</p> <pre>[assembly:UIPermission(SecurityAction.RequestRefuse, Window=UIPermissionWindow.AllWindows)]</pre>
6.9	<p>Always set the principal policy in every <code>Main()</code> method to Windows</p> <pre>public class MyClass { static void Main() { AppDomain currentDomain = Thread.GetDomain(); currentDomain.SetPrincipalPolicy(PrincipalPolicy.WindowsPrincipal); } //other methods }</pre>
6.10	Never assert a permission without demanding a different permission in its place.
7.0	System.Transactions
7.1	Always dispose of a <code>TransactionScope</code> object.
7.2	Inside a transaction scope, do not put any code after the call to <code>Complete()</code> .
7.3	When setting the ambient transaction, always save the old ambient transaction and restore it when you are done.
7.4	In Release builds, never set the transaction timeout to zero (infinite timeout).
7.5	When cloning a transaction, always use <code>DependentCloneOption.BlockCommitUntilComplete</code> .
7.6	Create a new dependent clone for each worker thread. Never pass the same dependent clone to multiple threads.
7.7	Do not pass a transaction clone to the <code>TransactionScope</code> 's constructor.
7.8	Always catch and discard exceptions thrown by a transaction scope that is set to <code>TransactionScopeOption.Suppress</code> .
8.0	Enterprise Services
8.1	Do not catch exceptions in a transactional method. Use the <code>AutoComplete</code> attribute.

This material is intended for FPIGC use only. It must not be reproduced in whole or in part, in any form, or by any means without a formal agreement or the written consent of the Process Management Unit (PMU). Any hard copy or unprotected soft copy of this document shall be regarded as an uncontrolled copy.

Filename: ws-qms-gl-027 c# coding standard

Fujitsu Philippines Inc. Group of Companies	C# Coding Standard		
Document No.: WS-QMS-GL-027	Effective Date: September 2, 2008	Version: 3.00	Page 19 of 19

This document is classified as ☐ Public ☒ Private ☐ Confidential ☐ Secret

8.2	Do not call <code>SetComplete()</code> , <code>SetAbort()</code> , and the like. Use the <code>AutoComplete</code> attribute. <pre>[Transaction] public class MyComponent : ServicedComponent { [AutoComplete] public void MyMethod(long objectIdentifier) {...} }</pre>
8.3	Always override <code>CanBePooled</code> and return <code>true</code> (unless you have a good reason not to return to pool) <pre>public class MyComponent : ServicedComponent { protected override bool CanBePooled() { return true; } }</pre>
8.4	Always call <code>Dispose()</code> explicitly on a pooled objects unless the component is configured to use JITA as well.
8.5	Never call <code>Dispose()</code> when the component uses JITA.
8.6	Always set authorization level to application and component.
8.7	Set authentication level to <code>privacy</code> on all applications. <pre>[assembly: ApplicationActivation(ActivationOption.Server)] [assembly: ApplicationAccessControl(true, //Authorization AccessChecksLevel=AccessChecksLevelOption.ApplicationComponent, Authentication=AuthenticationOption.Privacy, ImpersonationLevel=ImpersonationLevelOption.Identify)]</pre>
8.8	Set impersonation level on client assemblies to <code>Identity</code> .
8.9	Always set <code>ComponentAccessControl</code> attribute on serviced components to <code>true</code> (the default is <code>true</code>) <pre>[ComponentAccessControl] public class MyComponent : ServicedComponent {...}</pre>
8.10	Always add to the <code>Marshaler</code> role the Everyone user <pre>[assembly: SecurityRole("Marshaler",SetEveryoneAccess = true)]</pre>
8.11	Apply <code>SecureMethod</code> attribute to all classes requiring authentication. <pre>[SecureMethod] public class MyComponent : ServicedComponent {...}</pre>

Definition of Terms

Terms	Description
None.	



Fujitsu Philippines Inc. Group of Companies	C# Coding Standard		
Document No.: WS-QMS-GL-027	Effective Date: September 2, 2008	Version: 3.00	Page 20 of 19

This document is classified as ☐ Public ☒ Private ☐ Confidential ☐ Secret

Sources

Lowy, Juval. "C# Coding Standard Guidelines and Best Practices Version 2.1." IDesign. Sept. 2005. 27 June 2006 <<http://idesign.net/idesign/DesktopDefault.aspx>>.

Change History

Version No.	Issue Date	Modified By	Description of Changes	Approved By	Change Ref. No.
0.01	07-04-2006		Initial Draft	NA	NA
1.00	8-25-2006		Official Release	L. Navarro	NA
1.01	04-24-2008	P. San Diego	Changed document no. format to AAA-SSS-CCC-NNN Where: AAA: Company name (FPI: Fujitsu Philippines, Inc.; WS: WeServ Systems International, Inc.) SSS: Applicable standard (ex. QMS or ISMS) CCC: Document Code NNN: Sequence Number (from 000 to 999)		CHR-2008-002
2.00	04-28-2008		Official Release	Jun Santos Linda Navarro  "approval of new naming convention d	NA
2.01	08-11-2008	C. Mendoza	Included options on document classification whether it's Public, Private, Confidential or Secret		CHR-2008-012
3.00	09-02-2008		Official Release	Jun Santos  "P-07-100-20080902 -QMS Release602 apj	NA