| **Code Review Checklist**  **(Driver OA Addins ver )** | | | | | |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Topic** | **Criteria** | **Remark** | **Answer** | **Evidence** |
| 1 | Coding Standard Technique / Measure | Does the code follow any coding standard that are platform (e.g. Zenon or VBS) specifics? |  | comply, Using lance hunt coding standard, ECMA and EN50128, except for unconditional jump using ECMA coding standard chapter 13.10 | Add sample code that using lance hunt standard as a reference |
| 2 |  | Does the code follow any coding style guide? |  | comply, using lancehunt chapter 1.4.2 | 1. Namespace and public class only stated once in one file 2. Comment using // for single line and /// block 3. Curly braces always write on new line. 4. Add Screenshoot to prove the criteria that has been stated in …. Id comment |
| 3 |  | No Unconditional Jumps |  |  |  |
| 4 |  | Limited size and complexity of Functions, Subroutines and Methods |  |  |  |
| 5 |  | Limited use of Global Variables |  |  |  |
| 6 |  | Checked for memory leaks? |  |  |  |
| 7 | Code Style | Avoid naming conflicts with existing .NET Framework namespaces, or types. |  |  |  |
| 8 |  | Use white space (CR/LF, Tabs, etc) liberally to separate and organize code. |  |  |  |
| 9 |  | All comments should be written in the same language, be grammatically correct, and contain appropriate punctuation. |  |  |  |
| 10 |  | Always apply C# comment-blocks (///) to public, protected, and internal declarations. |  |  |  |
| 11 | Language Usage | Do not omit access modifiers. Explicitly declare all identifiers with the appropriate access modifier instead of allowing the default. |  |  |  |
| 12 |  | Only declare member variables as private. Use properties to provide access to them with public, protected, or internal access modifiers. |  |  |  |
| 13 |  | Declare readonly or static readonly variables instead of constants for complex types. |  |  |  |
| 14 |  | Always explicitly initialize arrays of reference types using a for loop. |  |  |  |
| 15 | Flow Control | Avoid assignment within conditional statements. |  |  |  |
| 16 |  | Avoid creating recursive methods. Use loops or nested loops instead |  |  |  |
| 17 |  | Prefer nested if/else over switch/case for short conditional sequences and complex conditions. |  |  |  |
| 18 |  | Avoid explicit Boolean tests in conditionals. |  |  |  |
| 19 | Exception | Do not use try/catch blocks for flow-control. |  |  |  |
| 20 |  | Only catch exceptions that you can handle. |  |  |  |
| 21 |  | Never declare an empty catch block. |  |  |  |
| 22 |  | Avoid nesting a try/catch within a catch block |  |  |  |
| 23 |  | Only use the finally block to release resources from a try statement. |  |  |  |
| 24 |  | Always use validation to avoid exceptions. |  |  |  |
| 25 | Event, Delegates & Threading | Always check Event & Delegate instances for null before invoking. |  |  |  |
| 26 |  | Use the default EventHandler and EventArgs for most simple events. |  |  |  |
| 27 |  | Always derive a custom EventArgs class to provide additional data. |  |  |  |
| 28 | Object Composition | Always declare types explicitly within a namespace. Do not use the default “{global}” namespace. |  |  |  |
| 29 |  | 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. |  |  |  |
| 30 |  | Consider using internal or private access modifiers for types and members unless you intend to support them as part of a public API. |  |  |  |
| 31 |  | Never use the protected access modifier within sealed classes unless overriding a protected member of an inherited type. |  |  |  |
| 32 |  | Always call Close() or Dispose() on classes that offer it. |  |  |  |
| 33 |  | Wrap instantiation of IDisposable objects with a “using” statement to ensure that Dispose() is automatically called. |  |  |  |
| 34 |  | Always implement the IDisposable interface & pattern on classes referencing external resources. |  |  |  |
| 35 |  | Avoid implementing a Finalizer. Never define a Finalize() method as a finalizer. Instead use the C# destructor syntax. |  |  |  |
| 36 | Configuration | Is appropriate change history documented? |  |  |  |
| 37 |  | Are the interfaces and the parameters there of properly documented? |  |  |  |

Note :