Description

Company: Company Name
Program: Program Name
Language: C#
Last Version Complete: XXX

Revision History

Code Review Rev No.	Purpose/Description	Program Version	Reviewer	Start Date	End Date	Peer Reviewer	Peer Review Date

TEMPLATE Approver	Date	Rev
John Schweitzer	4/30/2008	05-02

File's Functions Line Count Consolidated Report for

Company Name, Program Name

On the Application level, no more than 50%
exceeding 60 lines, no more than 5%
exceeding 120 lines, and none exceeding 240

Code Review Supervisor: Replace example languages and insert lines as needed. When the reports are compiled, link the cells in the table below to the appropriate spreadsheet files.

Language	Less than 60+N/A	60 to 120	120 to 240	More than 240	Total
C#	#REF!	#REF!	#REF!	#REF!	#REF!
Total %	#REF!	#REF!	#REF!	#REF!	#REF!

VVSG Section									VVSG v.1: 5.2.2	VVSG v.1: 5.2.3.a	VVSG v.1: 5.2.3.b	Verity Std sec.: 3.3.1	Hart Verity Coding Std sec.: 3.2	VVSG v.1: 5.2.3.b 5.2.7.a
General Description	**Review Date**	**Reviewer Name**	**Program Name**	**Program Version**	**TDP Date**	**File Path**	**File name**	**Class or Function name**	**Self-modifying code**	**Specific function**	**Module has unique name**	**Meaningful names**	**Hart Naming Conventions**	**Module has header**
VVSG or Standard Description									Self-modifying, dynamically loaded, or interpreted code is prohibited	Module performs a specific function	Uniquely and mnemonically named using names that differ by more than a single character	Do use meaningful names for various types, functions, variables, constructs and data structures. Their use should be plainly discernable from their name alone.	3.2.1 Use PascalCasing for classes enumerations, methods , functions, public properties, events, resources, and externally visible constants in C#. 3.2.2 Use PascalCasing preceded by 'I' for interfaces 3.2.3 Use PascalCasing preceded by 'T' for templates and generic type parameters 3.2.4 Use all capital letters separated by underscores for	Header describes purpose, other units needed, inputs, outputs, files read or written, globals, revision records (for modules greater than 10 lines) Header comments shall provide the following information: 1) The purpose of the unit and how it works; 2) Other units called and the calling sequence; 3) A description of input parameters and outputs; 4) File references by name and method of access (read, write, modify , append, etc.); 5) Global variables used; and 6) Date of creation and a revision record;
Language or Vendor Adjustments														Hart uses /// comments for classes, properties, and methods
								Total Accept	0	0	0	0	0	0
								Total Reject	0	0	0	0	0	0
								Total N/A	0	0	0	0	0	0
								Total Blank	5	5	5	5	5	5
								Total	5	5	5	5	5	5

VVSG v.1: 5.2.3.c	VVSG v.1: 5.2.3.e	VVSG v.1: 5.2.3.e	VVSG v.1: 5.2.3.f	VVSG v.1: 5.2.4.a	VVSG v.1: 5.2.4.I	VVSG v.1: 5.2.4.ii	VVSG v.1: 5.2.4.iii	MS Std sec.: 2.2	Verity Std sec.: 2.1	MS Std sec.: 2.5	MS Std sec.: 2.5	Verity Std sec.: 3.8.4	MS Std sec.: 2.6	MS Std sec.: 2.7	MS Std sec.: 2.8	MS Std sec.: 2.8.1
Required resources	**Single Entry Point**	**Single Exit Point**	**Control structures**	**Acceptable Constructs**	**Vendor Defined Constructs with Justification**	**Execution through Control Constructs**	**Program re-direction**	**Do not use tabs**	**Line length**	**Local variables have minimum scope**	**Local variable declaration and initialization**	**Initialize pointer variables**	**Parameters ordered in groups**	**One statement per line**	**Use enums**	**Flag enums**
All required resources, such as data accessed by the module, should either be contained within the module or explicitly identified	Module has a single entry point	Module has a single exit point	Sequence, Conditionals, Top & Bottom tested loops, Switch..Case, For loops	Acceptable constructs are Sequence, If-Then-Else, Do-While, Do-Until, Case, and the General loop (including the special case for loop); Note: This criteria will be "Accept" except in the case where the vendor defines code constructs.	If the programming language used does not provide these control constructs, the vendor shall provide them (that is, comparable control structure logic). The constructs shall be used consistently throughout the code. No other constructs shall be used to control program logic and execution;	While some programming languages do not create programs as linear processes, stepping from an initial condition, through changes, to a conclusion, the program components nonetheless contain procedures (such as "methods" in object-oriented languages). Even in these programming languages, the procedures must execute	Logic that evaluates received or stored data shall not re-direct program control	All code should be written using four spaces for indentation.	Maximum line length is 90 characters for C/C++	Do declare local variables in the minimum scope block that can contain them, typically just before use if the language allows; otherwise, at the top of that scope block	Do initialize variables when they are declared. Do declare and initialize/assign local variables on a single line where the language allows it. Do not declare multiple variables in a single line.	Initialize pointers (when) they are declared (and) set them to NULL after freeing them.	Do order parameters, grouping the in parameters first, the out parameters last.	Do not put more than one statement on a single line	Do use an enum to strongly type parameters, properties, and return values that represent sets of values. Do not use an enum for open sets. Do not use Enum.IsDefined for enum range checks in .NET.	Do apply the System.Flags Attribute to flag enums in .NET. Do not apply this attribute to simple enums. Do use powers of two for the flags enum values.
									Hart allows 150 chars per line for C# and				Very similar to MS Std sec.: 4.9.4			
					N/A											
					N/A											
					N/A											
					N/A											
					N/A											
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0
5	5	5	5	5	0	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5

MS Std sec.: 2.10	Verity Std sec.: 3.7.3	MS Std sec.: 2.11.2	MS Std sec.: 2.11.6	MS Std sec.: 3.1.1	MS Std sec.: 3.2.2	MS Std sec.: 3.5 3.10	MS Std sec.: 3.7	MS Std sec.: 3.7, 3.9.1	MS Std sec.: 3.8	MS Std sec.: 3.10	MS Std sec.: 3.11.1	MS Std sec.: 3.11.2	MS Std sec.: 3.11.5	MS Std sec.: 3.12.2	MS Std sec.: 3.12.3	MS Std sec.: 3.13.1
Braces and indentation	**Braces around single line conditionals**	**File has header comments**	**TODO comments**	**Precompiled Header**	**Include guards within header files**	**Define named constants as 'const' values**	**Use sizeof(var)**	**Use ARRAYSIZE for arrays**	**UNICODE code**	**Macros**	**Validating Parameters**	**Reference Parameters**	**Return Values**	**Structure Initialization**	**Structs vs. Classes with no methods**	**Data Members**
Do use Allman bracing style. The style puts the brace associated with a control statement on the next line, indented to the same level as the control statement. Statements within the braces are indented to the next level.	Use braces around single line conditionals	Do have a file header comment at the start of every human-created code file. The header comment templates are as follows: /********** ***\ Module Name: <File Name> Project: <Sample Name> <Description of the file> ********** ***/	Do not use TODO comments in any released samples	Do not use precompiled headers. Remove #include<stdafx.h> from all source files.	Do use include guards within a header file (internal include guards) to prevent unintended multiple inclusions of the header file	Do define named constants as 'const' values, instead of "#define" values. Do not use "#define" values for constant values.	Do use sizeof(var) instead of sizeof(TYPE) whenever possible. To be explicit about the size value being used whenever possible write sizeof(var) instead of sizeof(TYPE_OF_VAR). Do not code the known size or type of the variable. Do reference the variable name instead of the variable type in sizeof.	Do not use sizeof for arrays to get the element number. Use ARRAYSIZE. Do use ARRAYSIZE() as the preferred way to get the size of an array. Do derive the array size from the variable rather than specifying the size in your code.	Do write explicitly UNICODE code. Don't use TCHAR or ANSI code. This means: Use the wide char types including wchar_t, PWSTR, PCWSTR instead of the TCHAR versions. Don't use the TEXT macro, instead use the L prefix for creating Unicode string constants L"string value".	Do not use the following existing macros: SIZEOF(), IID_PPV_ARG() (use IID_PPV_ARGS() instead)	Do validate parameters to functions that will be used by the public.	Do not use ref parameters for output because it makes it hard to determine whether a variable is modified (an output) at the call site. Use pointers instead.	Do test the return of a function, not the out parameters, in the caller.	Do use "= {}" to zero structure memory.	Do use a structure to define a data aggregate that does not contain functions. Use a class if and only if there are member functions included.	Do not declare public data members. Use inline accessor functions for performance. Do initialize member variables in the same order that they were defined in the class declaration.
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5

MS Std sec.: 3.13.2	MS Std sec.: 3.13.3	MS Std sec.: 3.13.4	MS Std sec.: 3.13.5	MS Std sec.: 3.13.6	MS Std sec.: 3.13.7	MS Std sec.: 3.14.1	MS Std sec.: 3.14.3	MS Std sec.: 3.15	MS Std sec.: 3.16.1	MS Std sec.: 3.16.2	MS Std sec.: 3.17	Hart Verity Coding Std sec.: 2.2.2	Discrepancy		
Constructors	**Destructors**	**Overloaded Operators**	**Overloaded Function Parameters**	**Virtual Functions**	**Abstract Classes**	**COM Interfaces**	**COM Classes**	**Allocations**	**Errors**	**Exceptions**	**Resource Cleanup**	**Methods and Functions must be less than 240 lines**	**Disc. #**	**Description**	**Date Opened**
Do define copy constructors as taking a 'const' reference type. Do define all single parameter constructors, by default, with the 'explicit' keyword, so that they are not conversion constructors.	Do use a destructor to centralize the resource cleanup of a class that is freed via delete. If resources are freed before destruction, make sure the fields are reset (e.g. set pointers to NULL) so that a destructor will not try to free them again. Do declare the destructor as "virtual" for classes that contain at least one other virtual function. If the class does not contain any virtual functions, then do not declare the destructor as virtual.	Do not overload operator&&, operator|| or operator,. Do not change the semantics of the operators if you choose to overload them. For example, do not re-purpose the '+' operator for performing subtraction.	Do not arbitrarily vary parameter names in overloads. If a parameter in one overload represents the same input as a parameter in another overload, the parameters should have the same name. Parameters with the same name should appear in the same position in all overloads.	Do use virtual functions to implement polymorphism	Do provide a protected constructor. Do identify abstract methods by making them pure virtual. Do provide a public, virtual destructor if you allow deletion via a pointer to the abstract class or a protected, non-virtual destructor to disallow deletion via a pointer to the abstract class. Do explicitly provide protected copy constructor and assignment operators or private unimplement	Do use IFACEMTHODIMP and IFACEMTHODIMP_ for method declarations in COM interfaces. Do order interface methods in your class definition in the same order they are declared in their definition.	Do declare private destructors (or protected if you expect people to derive from you) for classes that implement COM objects that are allocated on the heap. Do initialize the m_cRef to 1 on construction for classes that implement COM object. Do return an HRESULT from every COM method (except AddRef and Release).	Do ensure that all allocated memory is freed using the same mechanisms. Objects allocated using 'new' should be freed with 'delete'. Allocations made using 'vector new' should be freed using 'vector delete'.	Do check return values for function calls and handle errors appropriately.	Do throw exceptions by value and catch exceptions by reference. When re-throwing exceptions do re-throw exceptions using "throw;" instead of "throw <caught exception>". Do not allow exceptions to be thrown out of destructors. Do not use "catch(…)".General exceptions should not be caught. Only catch the specific exceptions that the function knows how	Dynamically allocated memory / resources must be appropriately cleaned up before you exit from a function	Executable code in methods and functions must be less than 240 lines.			

0	0	0	0	0	0	0	
0	0	0	0	0	0	0	
0	0	0	0	0	0	0	
5	5	5	5	5	5	5	
5	5	5	5	5	5	5	

		Fixed Discrepancy			
TDP Date	**Reviewer Name**	**Date Closed**	**TDP Date**	**Reviewer Name**	**Notes**

Source Code Review Detail

Client Name:	Company Name
Report Date:	
Program:	Program Name
Version:	XXX
Programming Language:	C#

Total Modules Reviewed	0
Total Modules with Discrepancy	0

VSS Req.	Criterion	Definition		
Vol. 1 Section 4.2.2-Integrity			**Accept**	**Reject**
VVSG v.1: 5.2.2	**Self-modifying code**	Self-modifying, dynamically loaded, or interpreted code is prohibited	0	0
Vol. 1 Section 4.2.3- Modularity			**Accept**	**Reject**
VVSG v.1: 5.2.3.a	**Specific function**	Module performs a specific function	0	0
VVSG v.1: 5.2.3.b	**Module has unique name**	Uniquely and mnemonically named using names that differ by more than a single character	0	0

VVSG v.1: 5.2.3.b 5.2.7.a	**Module has header**	Header describes purpose, other units needed, inputs, outputs, files read or written, globals, revision records (for modules greater than 10 lines) Header comments shall provide the following information: 1) The purpose of the unit and how it works; 2) Other units called and the calling sequence; 3) A description of input parameters and outputs; 4) File references by name and method of access (read, write, modify , append, etc.); 5) Global variables used; and 6) Date of creation and a revision record;	0	0
VVSG v.1: 5.2.3.c	**Required resources**	All required resources, such as data accessed by the module, should either be contained within the module or explicitly identified	0	0
VVSG v.1: 5.2.3.e	**Single Entry Point**	Module has a single entry point	0	0
VVSG v.1: 5.2.3.e	**Single Exit Point**	Module has a single exit point	0	0
VVSG v.1: 5.2.3.f	**Control structures**	Sequence, Conditionals, Top & Bottom tested loops, Switch..Case, For loops	0	0
Vol. 1 Section 4.2.4-Control Constructs			**Accept**	**Reject**
VVSG v.1: 5.2.4.a	**Acceptable Constructs**	Acceptable constructs are Sequence, If-Then-Else, Do-While, Do-Until, Case, and the General loop (including the special case for loop); Note: This criteria will be "Accept" except in the case where the vendor defines code constructs.	0	0

VVSG v.1: 5.2.4.I	**Vendor Defined Constructs with Justification**	If the programming language used does not provide these control constructs, the vendor shall provide them (that is, comparable control structure logic). The constructs shall be used consistently throughout the code. No other constructs shall be used to control program logic and execution;	0	0
VVSG v.1: 5.2.4.ii	**Execution through Control Constructs**	While some programming languages do not create programs as linear processes, stepping from an initial condition, through changes, to a conclusion, the program components nonetheless contain procedures (such as “methods” in object-oriented languages). Even in these programming languages, the procedures must execute through these control constructs (or their equivalents, as defined and provided by the vendor);	0	0
VVSG v.1: 5.2.4.iii	**Program re-direction**	Logic that evaluates received or stored data shall not re-direct program control	0	0
Vol. 1 Section 4.2.5-Naming Conventions			**Accept**	**Reject**
MS Std sec.: 2.2	**Do not use tabs**	All code should be written using four spaces for indentation.	0	0
Verity Std sec.: 2.1	**Line length**	Maximum line length is 90 characters for C/C++	0	0
Vol. 1 Section 4.2.6-Coding Conventions			**Accept**	**Reject**
MS Std sec.: 2.5	**Local variables have minimum scope**	Do declare local variables in the minimum scope block that can contain them, typically just before use if the language allows; otherwise, at the top of that scope block	0	0

MS Std sec.: 2.5	**Local variable declaration and initialization**	Do initialize variables when they are declared. Do declare and initialize/assign local variables on a single line where the language allows it. Do not declare multiple variables in a single line.	0	0
MS Std sec.: 2.6	**Parameters ordered in groups**	Do order parameters, grouping the in parameters first, the out parameters last.	0	0
#REF!	**#REF!**	#REF!	#REF!	#REF!
MS Std sec.: 2.7	**One statement per line**	Do not put more than one statement on a single line	0	0
MS Std sec.: 2.8	**Use enums**	Do use an enum to strongly type parameters, properties, and return values that represent sets of values. Do not use an enum for open sets. Do not use Enum.IsDefined for enum range checks in .NET.	0	0
MS Std sec.: 2.8.1	**Flag enums**	Do apply the System.FlagsAttribute to flag enums in .NET. Do not apply this attribute to simple enums. Do use powers of two for the flags enum values.	0	0
MS Std sec.: 2.10	**Braces and indentation**	Do use Allman bracing style. The style puts the brace associated with a control statement on the next line, indented to the same level as the control statement. Statements within the braces are indented to the next level.	0	0

MS Std sec.: 2.11.2	**File has header comments**	Do have a file header comment at the start of every human-created code file. The header comment templates are as follows: /*****************\ Module Name: <File Name> Project: <Sample Name> <Description of the file> *****************/	0	0
MS Std sec.: 2.11.6	**TODO comments**	Do not use TODO comments in any released samples	0	0
MS Std sec.: 3.1.1	**Precompiled Header**	Do not use precompiled headers. Remove #include<stdafx.h> from all source files.	0	0
MS Std sec.: 3.2.2	**Include guards within header files**	Do use include guards within a header file (internal include guards) to prevent unintended multiple inclusions of the header file	0	0
MS Std sec.: 3.5 3.10	**Define named constants as 'const' values**	Do define named constants as 'const' values, instead of "#define" values. Do not use "#define" values for constant values.	0	0
MS Std sec.: 3.7	**Use sizeof(var)**	Do use sizeof(var) instead of sizeof(TYPE) whenever possible. To be explicit about the size value being used whenever possible write sizeof(var) instead of sizeof(TYPE_OF_VAR). Do not code the known size or type of the variable. Do reference the variable name instead of the variable type in sizeof.	0	0

MS Std sec.: 3.7, 3.9.1	**Use ARRAYSIZE for arrays**	Do not use sizeof for arrays to get the element number. Use ARRAYSIZE. Do use ARRAYSIZE() as the preferred way to get the size of an array. Do derive the array size from the variable rather than specifying the size in your code.	0	0
MS Std sec.: 3.8	**UNICODE code**	Do write explicitly UNICODE code. Don't use TCHAR or ANSI code. This means: Use the wide char types including wchar_t, PWSTR, PCWSTR instead of the TCHAR versions. Don't use the TEXT macro, instead use the L prefix for creating Unicode string constants L"string value".	0	0
MS Std sec.: 3.11.1	**Validating Parameters**	Do validate parameters to functions that will be used by the public.	0	0
MS Std sec.: 3.11.2	**Reference Parameters**	Do not use ref parameters for output because it makes it hard to determine whether a variable is modified (an output) at the call site. Use pointers instead.	0	0
#REF!	**#REF!**	#REF!	0	0
#REF!	**#REF!**	#REF!	#REF!	#REF!
MS Std sec.: 3.11.5	**Return Values**	Do test the return of a function, not the out parameters, in the caller.	0	0
MS Std sec.: 3.12.2	**Structure Initialization**	Do use "= {}" to zero structure memory.	0	0
MS Std sec.: 3.12.3	**Structs vs. Classes with no methods**	Do use a structure to define a data aggregate that does not contain functions. Use a class if and only if there are member functions included.	0	0
Vol. 1 Section 4.2.7 -Comments			**Accept**	**Reject**

MS Std sec.: 3.13.1	**Data Members**	Do not declare public data members. Use inline accessor functions for performance. Do initialize member variables in the same order that they were defined in the class declaration.	0	0
MS Std sec.: 3.13.2	**Constructors**	Do define copy constructors as taking a 'const' reference type. Do define all single parameter constructors, by default, with the 'explicit' keyword, so that they are not conversion constructors.	0	0
MS Std sec.: 3.13.3	**Destructors**	Do use a destructor to centralize the resource cleanup of a class that is freed via delete. If resources are freed before destruction, make sure the fields are reset (e.g. set pointers to NULL) so that a destructor will not try to free them again. Do declare the destructor as "virtual" for classes that contain at least one other virtual function. If the class does not contain any virtual functions, then do not declare the destructor as virtual.	0	0
#REF!	**#REF!**	#REF!	#REF!	#REF!

#REF!	**#REF!**	#REF!	**Less than 60+N/A**	**60 to 120**	**120 to 240**	**More than 240**	**Total**
			#REF!	#REF!	#REF!	#REF!	#REF!
			#REF!	#REF!	#REF!	#REF!	#REF!